Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

INTRODUCCIÓN A LOS SISTEMAS FÍSICO-CIBERNÉTICOS COMO ACTUALIZACIÓN EN LAS PRÁCTICAS DE LABORATORIO DEL CURSO DE SISTEMAS DE CONTROL

Oscar Rolando Ramírez Milián
Asesorado por la Inga. Ingrid Salomé Rodríguez de Loukota

Guatemala, junio de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

INTRODUCCIÓN A LOS SISTEMAS FÍSICO-CIBERNÉTICOS COMO ACTUALIZACIÓN EN LAS PRÁCTICAS DE LABORATORIO DEL CURSO DE SISTEMAS DE CONTROL

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

OSCAR ROLANDO RAMÍREZ MILIÁN
ASESORADO POR LA INGA. INGRID SALOMÉ RODRÍGUEZ DE LOUKOTA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, JUNIO DE 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Julio Rolando Barrios Archila
EXAMINADORA	Inga. Ingrid Salomé Rodríguez de Loukota
EXAMINADOR	Ing. Jose Aníbal Silva de los Angeles
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

INTRODUCCIÓN A LOS SISTEMAS FÍSICO-CIBERNÉTICOS COMO ACTUALIZACIÓN EN LAS PRÁCTICAS DE LABORATORIO DEL CURSO DE SISTEMAS DE CONTROL

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 2 de noviembre de 2014.

Oscar Rolando Ramírez Milián

Guatemala 4 de febrero de 2016

Ingeniero
Carlos Eduardo Guzmán Salazar
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Estimado Ingeniero Guzmán.

Me permito dar aprobación al trabajo de graduación titular: "**Introducción a los sistemas físico-cibernéticos como actualización en las prácticas de laboratorio del curso de Sistemas de Control**", del señor Oscar Rolando Ramírez Milián, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo de graduación y, yo, como su asesora, nos hacemos responsables por el contenido y conclusiones del mismo.

Sin otro particular, me es grato saludarle.

Atentamente,

Inga. Ingrid Rodríguez de Loukota
Colegiada 5,356
Asesora

Ingrid Rodríguez de Loukota
Ingeniera en Electrónica
colegiado 5356

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 17. 2016.
Guatemala, 12 de FEBRERO 2016.

Señor Director
Ing. Francisco Javier González López
Director Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: INTRODUCCIÓN A LOS SISTEMAS FÍSICO- CIBERNÉTICOS COMO ACTUALIZACIÓN EN LAS PRÁCTICAS DE LABORATORIO DEL CURSO DE SISTEMAS DE CONTROL, del estudiante OSCAR ROLANDO RAMÍREZ MILIÁN, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

Ing. Carlos Eduardo Guzmán Salazar
Coordinador Área Electrónica

sro

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 17. 2016.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto bueno del Coordinador de Área, al trabajo de Graduación del estudiante; OSCAR ROLANDO RAMÍREZ MILIÁN Titulado: INTRODUCCIÓN A LOS SISTEMAS FÍSICO – CIBERNÉTICOS COMO ACTUALIZACIÓN EN LAS PRÁCTICAS DE LABORATORIO DEL CURSO DE SISTEMAS DE CONTROL, procede a la autorización del mismo.

Ing. Francisco Javier González López

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 4 DE ABRIL 2016.

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.D.278-2016

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica al trabajo de graduación titulado: **INTRODUCCIÓN A LOS SISTEMAS FÍSICO-CIBERNÉTICOS COMO ACTUALIZACIÓN EN LAS PRÁCTICAS DE LABORATORIO DEL CURSO DE SISTEMAS DE CONTROL,** presentado por el estudiante universitario: **Oscar Rolando Ramírez Milián,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, junio de 2016

/cc

ACTO QUE DEDICO A:

Mis padres	Irma Yolanda Milián Peláez y Oscar Enrique Ramírez Díaz, por su incondicional apoyo económico, moral y sabios consejos durante los años de carrera. Sin ellos no hubiera sido posible.
Mis amigos	Karina González, Ricardo Oliva, Rodrigo Chang, Victor Salazar, Carlos Oxom, Hugo López, Silvio Urizar, Hector Cojulún, Ana Marroquín, Luis Herrera y muchos más que, entre proyectos, aulas y aventuras, hicieron de la Universidad la mejor época de mi vida.
Mis hermanos	Luis, Irma y Rubí Ramírez, por ser personas con las que pude contar.
Buenos catedráticos	Iván Morales, Vera Marroquín, José Carlos Bonilla,Ingrid Rodríguez y Byron Arrivillaga, por ser una fuente de inspiración para las futuras generaciones de profesionales y formar mejores personas, tan necesarias para un mundo mejor.
Las fuerzas caóticas del universo	Cuyo entendimiento aún está fuera del alcance del hombre, las cuales conspiraron para que los resultados se dieran de la mejor manera.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala

Por brindar la oportunidad de estudiar a los ciudadanos de este país y formar profesionales que de una forma u otra lo transforman.

Facultad de Ingeniería

Por ser un excelente patrón, brindarme la oportunidad de tener un trabajo y con ello permitir sostenerme en mis estudios. Sin esa oportunidad no me hubiera sido posible seguir adelante.

Los ciudadanos trabajadores de Guatemala

Quienes llevan en los hombros más de quinientos años de una historia fomentada en la desigualdad y oportunismo; el peso de un Estado fallido, corrupto e impotente ante los embates de un mundo destruido por la sociedad del consumo y, a pesar de ello, honradamente realizan su trabajo y su mejor esfuerzo día a día.

Ellos, al contribuir con sus impuestos, otorgan una oportunidad a muchos guatemaltecos de tener educación superior en la Universidad de San Carlos de Guatemala, ayudando así poco a poco a vencer las barreras intelectuales y estereotipos, cuyo efecto se ve reflejado en los problemas sociales de nuestro país y el mundo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
$\mathbb{C}$	Conjunto de los números complejo.
$\mathbb{Z}$	Conjunto de los números enteros.
$\mathbb{N}$	Conjunto de los números naturales.
$\mathbb{R}$	Conjunto de los números reales.
$\mathbf{A^B}$	Conjunto de todas las funciones $f: A \to B$.
$\exists$	Cuantificador lógico, utilizado para darle denotar que al menos un elemento de un conjunto posee una cualidad específica. Se lee existe.
$\nexists$	Cuantificador lógico, utilizado para darle denotar que ningún elemento de un conjunto posee una cualidad específica. Se lee no existe.
$\forall$	Cuantificador lógico, utilizado para darle una cualidad específica a la totalidad de los elementos de un conjunto. Se lee para todo.
$\boldsymbol{\Delta}$	Denota diferencia entre dos entidades matemáticas; definida de forma específica para el tipo de entidad.
$\frac{\partial y}{\partial x}$	Derivada parcial de *y* respecto a la variable *x*.
$\frac{dy}{dx}$	Derivada total de *y* respecto a la variable *x*.
$\mathbf{F}$	Faradio(s).
$\boldsymbol{f: A \to B}$	Función *f* cuyo dominio es el conjunto *A* y cuya imagen es el conjunto *B*.
sgn	Función signo cuyo dominio son los números reales y cuya imagen son los números -1,0, 1. La función

	devuelve el valor 1 si el valor que esta valuando es mayor que cero, -1 si es menor que cero y cero en caso de ser cero.
H	Henrio(s).
Hz	Hertz.
$\in$	Indica pertencia a un conjunto.
$\int \boldsymbol{f}\ \boldsymbol{dx}$	Integral de *f* respecto a *x*.
Ω	Ohmio(s).
∇	Operador nabla. Obtiene el gradiente de una función.
$\mathbb{R}^{\mathbb{R}}$	Se hace referencia al conjunto de todas las funciones continuas.

GLOSARIO

Base ortonormal Conjunto mínimo de vectores capaz de generar un espacio vectorial, cuyos elementos son linealmente independientes, ortogonales entre sí y cuya norma es 1. Por ejemplo, para un espacio vectorial $\mathbb{R}^3$ usando coordenadas rectilíneas, los vectores x, y, z forman una base para $\mathbb{R}^3$, dado que se puede escribir cualquier vector $v \in \mathbb{R}^3$ usando solamente los vectores x, y y z, pero sin uno de ellos sería imposible escribirlos. Otro ejemplo, un espacio vectorial $\mathbb{R}^2$ usando coordenadas polares, los vectores 0 y r forman una base para $\mathbb{R}^2$.

Bit Elemento del conjunto $S_2 = \{0,1\}$. Unidad básica de información en computación y comunicaciones digitales.

Byte Unidad de información digital formada de 8 bits.

CAS Computer Algebra System, es un software que facilita matemática simbólica.

Compliancia Tasación de la propiedad de un órgano hueco que le permite el alargamiento o distensión en resistencia al retorno hacia sus dimensiones originales. Es el recíproco de elastancia.

Conjunto finito Sea C un conjunto y $|C|$ su cardinalidad. Si existe un número $M \in \mathbb{N}$, tal que $|C| < M$, se dice que C es finito.

Contable Conjunto en el cual todos elementos pueden relacionarse mediante una función biyectiva con un algún subconjunto de los naturales.

Creackear Aplicar ingeniería inversa a un sistema electrónico o de información con el fin de dañar o lograr fines maliciosos.

Eigenfunción Función propia de un valor lineal A en un espacio funcional, se define como cualquier función distinta de cero tal que al aplicarle el operador devuelve exactamente la misma función; pero multiplicada por un factor de escala. Por ejemplo, si un operador es una derivada $\frac{d^2}{dt^2}$, una eigenfunción será e^{KT} .

Entorno Al hablar de un sistema es la parte del universo que se encuentra fuera de las fronteras de ese sistema. Al hablar de software, puede referirse al espacio visual o lógico donde se ubican los accesos a ciertas herramientas o variables.

Fenómeno Cualquier cosa, entidad o manifestación física que pueda ser observable.

Información Del latín *in*for*mare*, que significa dar forma o formar una idea de algo. Información es cualquier tipo de patrón que influencia la formación o transformación de otros patrones. En teoría de la información es tomada como una secuencia de símbolos del un alfabeto, digamos un alfabeto de entrada $\mathcal{X}$ y alfabeto de salida y. El procesamiento de la información consiste en una función de entrada-salida que mapea cualquier secuencia de entrada desde $\mathcal{X}$ a una secuencia de salida desde y, dicho mapeo puede ser probabilista o determinista. Puede ser con memoria o sin memoria. Información también tiene una definición formal en la física. En 2003, J.D. Bekenstein sugiere que una creciente tendencia en física sea definir al universo de la física como un ente hecho puramente de información. La Teoría del Todo sugiere un nuevo paradigma, en el cual virtualmente todo, desde partículas y campos, hasta entidades biológicas y conscientes, puede ser descritos por patrones de información.

Interfaz En informática, se utiliza para nombrar a la conexión física y funcional entre dos sistemas o dispositivos de cualquier tipo dando una comunicación entre distintos niveles.

IoT Escenario en que los objetos, animales y personas con identificadores únicos tienen la habilidad de transferir información sobre la red sin necesidad de interacción humano-humano o humano-máquina. IoT

	ha evolucionado de la convergencia de las redes inalámbricas *wireless*, sistemas microelectromecánicos y la internet.
Isotrópico	Unif*or*me en todas las direcciones.
Magnitud	Propiedad que poseen los fenómenos o las relaciones entre ellos, que permite que puedan ser medidos.
Medir	Comparar con una unidad de la misma naturaleza llamada patrón.
Microcontrolador	Computadora con capacidad más reducida, pero con una arquitectura diseñada para interactuar con dispositivos periféricos. Ideal para sistemas embebidos.
N-tupla	Secuencia de *n* elementos, donde *n* es un entero no negativo. Existe una única 0-tupla y es una secuencia vacia. Por ejemplo, elementos (x, y, z) del conjunto en $\mathbb{R}^3$son una 3-tupla.
Observador	Cualquier ente, ser, persona, grupo de personas u objeto que analiza, diseña, modela, construye o describe un sistema sin ser parte del mismo.
Operador	En matemática e ingeniería, se hace referencia al término operador a la aplicación entre dos conjuntos que tiene como imagen conjunta. Por ejemplo, el

producto escalar es una aplicación entre dos elementos que pertenecen al mismo espacio vectorial, el cual es un conjunto y va hacia el campo de los reales.

Orden exponencial Se dice que f es de orden exponencial c si existen constantes $c, M > 0$ y $T > 0$ tales que $|f(t)| < Me^{cT}$ para todo $t > T$.

Ortogonal En álgebra lineal, es la propiedad que tiene un elemento de un espacio vectorial en referencia a otro perteneciente al mismo espacio que al operar entre ambos elementos el producto escalar, definido de una forma específica para el espacio, de dar como resultado cero. Si el producto escalar entre varios elementos del espacio vectorial es cero, se conocen como ortogonales entre sí.

Proceso Acción de un fenómeno natural o de una operación artificial vista como una serie sucesiva de acciones más pequeñas.

Protocolo Conjunto de reglas y estándares que controlan la secuencia de mensajes que ocurren durante una comunicación entre dispositivos.

Pseudovector Magnitud física que presenta propiedades de covariancia o transformación bajo reflexiones

	anómalas, presentando violaciones aparentes de la paridad física.
Releé	Interruptor donde el contacto se ve controlado por una bobina.
Tensión	También es conocido como voltaje o diferencia de potencia. Término utilizado normalmente para hacer referencia a la diferencia del potencial eléctrico entre dos puntos.

RESUMEN

En el presente trabajo de graduación se desarrolló una guía didáctica para ser utilizada en el Laboratorio de Sistemas de Control, impartido en el Laboratorio de Electrónica de la Escuela de Mecánica Eléctrica, que cursan los estudiantes de Ingeniería Eléctrica, Mecánica Eléctrica y Electrónica, en la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala. Dicho trabajo puede tomarse como referencia para otras universidades con objetivos similares.

La guía tiene un enfoque hacia el estudio de sistemas controlados, cuya implementación es el resultado de una fusión entre componentes físicos y tecnologías de la información, y son llamados sistemas físicocibernéticos; los cuales tendrán un rol muy importante en la actual tendencia de la IoT. La guía presenta conceptos y métodos básicos para el estudio más profundo de estos sistemas y trata algunas generalidades que los estudiantes de las carreras mencionadas con anterioridad pueden utilizar al profundizar en este tópico.

OBJETIVOS

General

Tratar y ampliar temas necesarios para que sirvan como soporte para que estudiantes de las carreras de Ingeniería Eléctrica, Mecánica Eléctrica y Electrónica puedan desarrollar aplicaciones que involucren sistemas fisicos-cibernéticos, los cuales estarán prácticamente embebidos en todos los aspectos de la sociedad en un futuro cercano.

Específicos

1. Dar una introducción a los microcontroladores para el control de sistemas físicos, básicos en los sistemas embebidos. También, una introducción a la modelación y simulación que permiten hacer aplicaciones de forma metódica.

2. Escribir ejercicios prácticos viables para un estudiante, para que pueda comprender la ciencia detrás de la implementación práctica.

3. Dar fundamentos para que un estudiante pueda entrar en contacto con tecnología útil en este tipo de sistemas, abarcando algoritmos y dispositivos.

4. Adaptar el laboratorio al estado socioeconómico de la población estudiantil presente en la Facultad de Ingeniería.

5. Estimular la investigación en dicha área.

INTRODUCCIÓN

En el capítulo 1 se presenta una introducción muy general a los sistemas de control y sistemas físicocibernéticos. Se dan algunas definiciones útiles durante el resto del trabajo y se presentan las fases a seguir en su implementación.

En el capítulo 2 se presenta una introducción a la etapa de modelación y simulación necesaria para ganar un conocimiento profundo del comportamiento de un sistema físico, vital en el diseño y análisis de un sistema físicocibernético. Se presentan algunos ejercicios recomendados sobre el uso del lagrangiano como herramienta para el modelo de algunos sistemas físico y la ejecución de simulaciones.

En el capítulo 3 se presenta una introducción a las dinámicas discretas. Se dan a conocer modelos, además, se tratan detalles de la arquitectura, funcionamiento, componentes de los procesadores como unidades de control de información. Se tomó como herramienta didáctica, en el estudio de dinámicas discretas, el microcontrolador tm4c123gh6pm basado en la ISA ARM Cortex-M4. Como ejercicio propuesto está la implementación de dinámicas discretas en un microcontrolador.

En el capítulo 4 se presenta el enlace entre los mundos físico y cibernético, se desglosa una lista de detalles de actuadores y sensores, básicos en el desarrollo de sistemas controlados de lazo cerrado. Este es el corazón del trabajo. Se dan a conocer algunos protocolos de comunicación entre dinámicas

discretas, se hace un análisis del ruido en las mediciones de un sensor y señales de los actuadores.

Como ejercicio práctico se proponen tres prácticas que, a consideración, se deben tratar para consolidar la información proporcionada en este capítulo. La primera de ellas es el control por micropaso de un motor paso a paso, que representa un lazo abierto. La implementación de un filtro de Kalman, que es un algoritmo básico para la toma de mediciones en entornos ruidosos y representa uno de los mayores avances en el campo de la robótica, automatización y ciencias aeroespaciales del siglo XX. Por último, la simulación de un control PID, que representa el algoritmo de control más utilizado en la industria, y contrastar los resultados de la simulación con una implementación real, materializada en un servomotor con modulación analógica.

1. TEORÍA DE SISTEMAS, CONTROL Y SISTEMAS FÍSICO-CIBERNÉTICOS

1.1. Control de entornos físicos

En el diario vivir se encuentran muchas tareas que realizar, ya sea porque son absolutamente necesarias o porque hacen las vidas un poco más cómodas; mantener la temperatura exacta en una caldera, para evitar que esta explote y ponga en riesgo la vida de muchas personas, o simplemente mantener una temperatura agradable en una habitación durante una mañana calurosa son ejemplos de estas tareas. Ambos son sistemas físicos bajo un determinado control. Muchos de ellos deben cumplir con ciertos objetivos: deben ser estables y robustos ante perturbaciones; eficientes según un criterio preestablecido. No es lo mismo el control de temperatura para un químico que reacciona de forma diferente a distintas temperaturas, que un control para la comodidad en una habitación. Dado que se intenta manipular el entorno físico, el control de un entorno se ve sujeto a muchas limitaciones, los criterios deben ser reales.

Los controles sobre entornos físicos son más fáciles de llevar utilizando componentes eléctricos y electrónicos. Hacerlo de forma mecánica sería sumamente difícil e impreciso. Además, en los últimos años se ha visto un crecimiento exponencial de las telecomunicaciones, y todas las redes manifiestan la tendencia de converger en una sola; es el inicio de la IoT, para bien o para mal de la humanidad, por ello los sistemas físicos controlados actualmente deben ser capaces de comunicarse con otros sistemas no necesariamente físicos, como redes sociales o cualquier otro sistema de

información. Por ejemplo, un automóvil no tripulado debe llevar el control de muchas variables que manipulan su dinámica y, además, ser capaz de comunicarse utilizando una serie de protocolos e infraestructura para brindar información de su estado. Otro ejemplo, es una red de sensores en una fábrica; para mostrar el estado de cada uno de los procesos y maquinaria, además de tomar los datos y manipular los procesos para que funcione correctamente; debe llevar registro de los datos y procesos con el fin de facilitar a las personas que controlan toma de decisiones. Este tipo de relación, controles de variables físicas con sistemas de información, sería sumamente difícil o casi imposible, además de poco práctico, utilizando tecnología analógica. Por esa razón, el enfoque de este texto es la construcción de controles utilizando microcontroladores y computadoras.

Figura 1. **Automóvil autónomo desarrollado por Google**

Fuente: *Compañía Google*. http://www.newyorker.com/wp-content/uploads/2012/11/self-driving-car-465.jpg. Consulta: junio de 2015.

1.2. Señales, variables y parámetros

Al hablar de sistemas automáticamente se hace referencia a variables, señales y parámetros.

Tabla I. **Definición de variable**

Una variable es una abstracción, por lo general hecha por un observador, de elementos, características, propiedades, estados, o relaciones presentes en un fenómeno.

Fuente: elaboración propia.

Al conducir un automóvil, se pueden observar muchas variables que entran en juego. La velocidad, posiblemente sea la más evidente de ellas, porque representa un estado del vehículo, pero existen muchas más dentro de este sistema: la temperatura del motor; el flujo de gasolina o la presión de las válvulas. Otras variables se encuentran ligadas al usuario, como la posición del timón; la configuración de la palanca de velocidades o la inclinación del acelerador. La fricción de las llantas con el suelo es una variable que representa una relación entre dos objetos. Todas ellas son una abstracción de una relación o estado. Del mismo modo, la configuración de la palanca de velocidades representa una característica. En la caja de velocidades, una configuración en primera indica que la caja será capaz de darle más tracción a las llantas, aunque se reduzca la velocidad del vehículo. El color del vehículo es otra variable que representa una característica. Si una variable describe características o propiedades se le llama cualitativa. Si es posible asociarle una cantidad o magnitud se le llama cuantitativa. En algunos casos, es posible medir dichas variables con un instrumento, en otros sólo se les asocia una magnitud para completar un modelo o análisis. El color del vehículo anterior es una variable cualitativa. Los flujos de gasolina se pueden medir, por lo que se

les puede asociar una cantidad o magnitud, por tanto es una variable cuantitativa. Cuando una variable no depende de ninguna otra, se le llama independiente y cuando depende de otras variables se le llama dependiente. En los sistemas que se tratan en este trabajo, el tiempo siempre será una variable independiente. Por lo general, la mayoría de las variables pueden ser expresadas como una función del tiempo.

Tabla II. **Definición de señal**

Sea *s* una función *s*: T → A, donde T representa al conjunto de n-tuplas formadas por *n* variables independientes y A representa a un conjunto cualquiera. *s* es una señal si lleva información.

Fuente: elaboración propia.

Siguiendo con el ejemplo del automóvil, todos los cambios en las variables del vehículo se ven reflejados de una u otra forma, por ejemplo, al momento de presionar el freno, el vehículo disminuye su velocidad drásticamente debido a la fricción de las llantas con el suelo. En dicho momento, se envió cierta información al sistema para indicar que se deseaba reducir la velocidad del vehículo. Presionar a cierta profundidad el freno genera reacciones en el sistema hidráulico del sistema de frenado; dependiendo que tan profundo se encuentra el freno, cambia la presión en el sistema hidráulico que es parte del sistema de frenado del vehículo. Dicha presión, el automóvil la mide e interpreta generando una reacción en el movimiento del vehículo. El flujo de gasolina que pasa a través de las válvulas puede medirse y traducirse a pulsos eléctricos utilizando un sensor, dichos pulsos los interpreta una computadora con tal de hacer eficiente el consumo de combustible. La presión en el sistema hidráulico o los pulsos eléctricos son ejemplos de señales.

Existe cierta confusión con la definición de señales y variables. La diferencia radica en que las variables son abstracciones para entender, interpretar, construir o modelar un fenómeno y se les asocia con tal de entender y analizar su comportamiento. De cierto modo son inherentes al sistema, un sistema no dejará de comportarse de cierta forma u otra solo por la forma en que se interpreten sus variables.

El sol siempre tuvo masa a pesar de que hace unos siglos no pudiera estimarse, en cambio, una señal es una forma de visualizar o representar alguna variable, por ejemplo el momento de un vehículo es parte de su naturaleza, pero al medir la velocidad utilizando un instrumento se interpreta una señal y se asocia una magnitud que represente ese momento. Una señal siempre lleva información de la variable que representa. Otro ejemplo es la configuración de la caja de velocidades, variable que puede tomar los valores *Retroceso, Primera, Segunda, Tercera, Cuarta o Quinta*, pero la información que recibe el sistema es la fuerza transmitida por la palanca a través de una serie de engranajes para cambiar la configuración de la caja y transmitir diferentes torsiones al automóvil. Una señal siempre lleva información de una variable, en el ejemplo del flujo de gasolina los pulsos eléctricos que genera el sensor de flujo son las señales que la computadora interpreta y asocia a determinado patrón de pulsos eléctricos un flujo específico. En la mayoría de sistemas la forma más útil de representar una señal es como una función del tiempo. Por lo general, se le asocian señales a todas aquellas variables que cambian en el tiempo. A la variable que representa el volumen del tanque de gasolina, no es necesario asociarle una señal que el sistema interprete o mida constantemente ya que siempre tendrá el mismo valor. A todas aquellas variables que no cambian en el tiempo se les llamará parámetros.

Tabla III. **Definición de parámetro**

Un parámetro es cualquier variable constante para todas las variables independientes del fenómeno.

Fuente: elaboración propia.

1.2.1. Clasificación de señales

Es posible clasificar las señales por los siguientes criterios:

- Por la naturaleza de sus magnitudes: es posible tener señales eléctricas, neumáticas, hidráulicas, mecánicas, entre muchas otras.
- Por su relación con el tiempo: señales de tiempo continuo se caracterizan por presentar un valor en cualquier instante, relacionadas con tecnología analógica o de tiempo discreto. Se caracterizan por presentar un valor solo en un número contable de valores, relacionadas con tecnologías digitales.
- Por su relación con el entorno: es posible tener señales físicas, por ejemplo, un pulso eléctrico, una presión variable o lógicas como las interrupciones en un microcontrolador.
- Por la cantidad de valores que puede tomar: digitales, binarias o analógicas.

1.3. Sistemas

Se usará la palabra sistema para hacer referencia a un conjunto de elementos que se relacionan entre sí. Un sistema es la parte del universo que se estudia, mientras que el entorno es el resto del universo que se encuentra fuera de las fronteras del sistema. El cuerpo de un ser vivo es un sistema, todos

los órganos se comunican entre sí, además realizan una función específica con el fin de mantener con vida al animal o planta. Un avión es un sistema, un conjunto de válvulas, motores, circuitos, engranajes y muchos otros elementos se ven relacionados con el fin ya sea de levantar vuelo o de sostener un estado estable en el aire. También se citan algunos sistemas más simples, por ejemplo una palanca, se compone de la palanca y el punto de apoyo, ambos elementos se relacionan entre sí; también un péndulo o una caja de engranajes, todos ellos tienen elementos relacionándose entre sí. La definición 4 lo explica de manera más formal.

Tabla IV. **Definición de sistema**

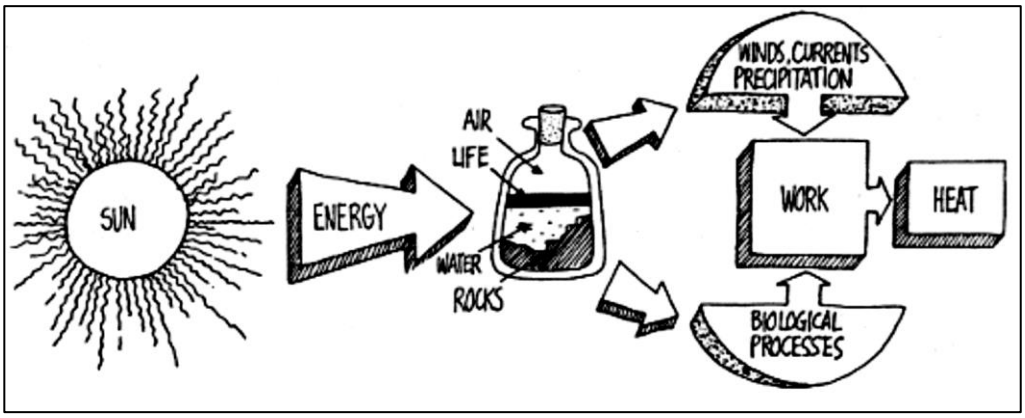
Sea *S* un conjunto de elementos escogidos por un observador, los cuales forman una entidad. Si los elementos de *S* se encuentran relacionados entre sí de una u otra manera, y *S* se encuentra delimitado espacial y temporalmente por fronteras (reales o impuestas por el observador), rodeado o influenciado por su entorno, descrito por una estructura o propósito y expresado por su funcionamiento, entonces *S* es un sistema.

Fuente: elaboración propia.

El clima es un sistema, ya que se encuentra delimitado por las fronteras del planeta Tierra, entre los elementos se pueden mencionar componentes como nubes, características o variables como lluvias, presiones o temperaturas. Siendo su entorno el mismo planeta Tierra, su estructura todas las capas de la tierra como la ionósfera, la atmósfera o la biosfera y su funcionamiento se puede apreciar todos los días. Un programa de computadora es un sistema, ya que es un conjunto de sentencias que se relacionan entre sí, cuyo entorno está conformado por los componentes de la computadora, delimitado espacialmente por la memoria que ocupa y utiliza, descrito por funciones, procesos en un lenguaje y un algoritmo que expresa su funcionamiento.

Qué es un sistema o qué no es un sistema en sí depende mucho del observador, por ejemplo, un globo de helio para *A* puede ser nada más un elemento sin más variables que el mismo globo. Pero para *B* ese mismo globo puede representar un sistema termodinámico donde el helio es analizado por variables propias del entorno, como presión o temperatura, y se encuentra delimitado por el globo como frontera.

Figura 2. **Planeta Tierra visto como un sistema**



Fuente: *Laboratorio de electrónica*. http://pespmc1.vub.ac.be/macroscope/MACROSCFIG5.GIF. Consulta: junio de 2015.

1.3.1. Entradas, salidas y procesos

Un sistema puede intercambiar materia, energía o información con su entorno. A las variables que representan esa materia, energía o información se les llama entradas o salidas dependiendo del caso. Por lo general se considera una señal como función del tiempo (como variable independiente) con imagen en estas variables (información, energía, materia).

Tabla V. **Definición de entrada**

Sea *x* una señal *x*: T → A, donde T es el conjunto de n-tuplas formadas por *n* variables independientes, y A representa a un conjunto de magnitudes o propiedades de la misma naturaleza. *x*(*t*) es recibido por el sistema y entregado por el entorno para *t* ϵ T si y solo sí *x*(*t*) es una entrada del sistema o una señal de entrada al sistema.

Fuente: elaboración propia.

No en todos los sistemas tiene sentido asociar una función del tiempo a las entradas o salidas, por ejemplo, en un sistema de encriptación, las entradas y salidas serían cadenas de caracteres. No tendría sentido buscar una función tal que para un tiempo t_A asocie una cadena *A*.

Tabla VI. **Definición de salida**

Sea *y* una señal *y*: T → A, donde T donde T es el conjunto de n-tuplas formadas por *n* variables independientes, y A representa a un conjunto de magnitudes o propiedades de la misma naturaleza. *Y* (*t)* es recibido por el entorno y entregado por el sistema para *t* ϵ T si y solo sí *y* (*t)* es una salida del sistema o una señal de salida desde el sistema.

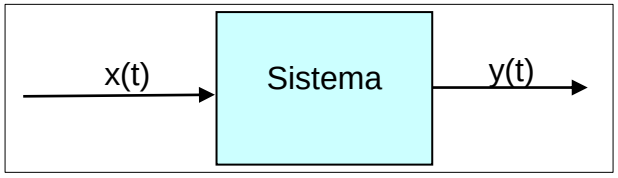
Fuente: elaboración propia.

De cierta forma, los conceptos de entradas y salidas se encuentran relacionados con el concepto de proceso. Un proceso es una acción que es vista como una sucesión progresiva de otras acciones más pequeñas. La mayoría de veces, un proceso puede ser visto como un sistema adentro de un sistema más grande, dicho proceso tiene entradas y salidas y su función principal es transformar las señales de entrada en señales de salida y para realizar análisis o modelos matemáticos de este tipo de procesos o sistemas. Frecuentemente los relacionamos con el concepto matemático de operador.

1.3.2. Diagramas de bloques

Es muy útil realizar abstracciones de los sistemas que se están modelando, diseñando o analizando. Para esta labor se hará uso de los diagramas de bloque. Un diagrama de bloques es un grafo compuesto por otros grafos llamados bloques que se conectan entre sí. Dichas conexiones se interpretan como señales. Un bloque representa un proceso o sistema, la convención de los bloques para representar un sistema es usando un cuadro con el nombre u operador asociado, inscrito en el cuadro. Las entradas se representan como flechas apuntando hacia dentro del bloque y las salidas como flechas dirigiéndose hacia fuera del bloque tal como se muestra en la figura 3.

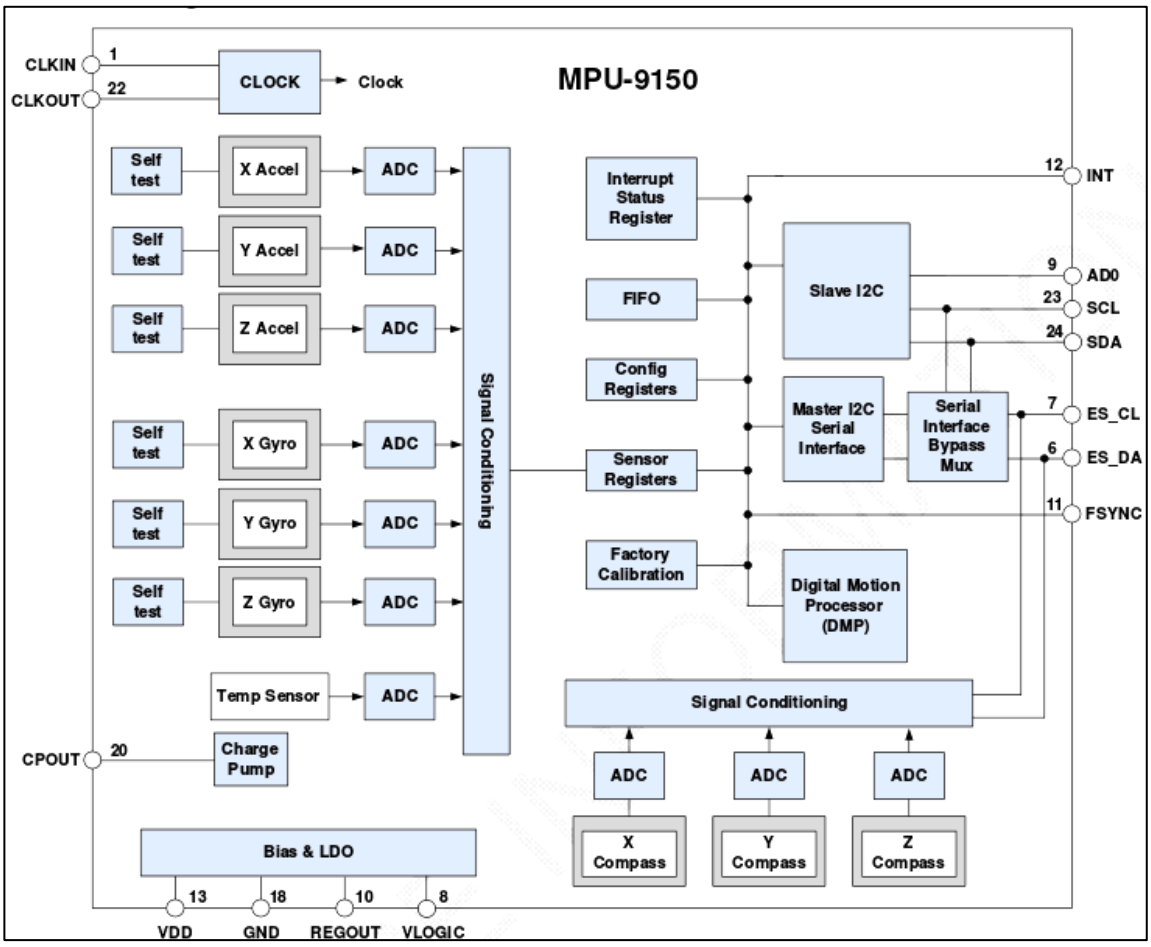
Figura 3. **Bloque de un sistema**



Fuente: elaboración propia, empleando Inkscape.

Se pueden utilizar estos diagramas para representar gráficamente el funcionamiento o la estructura de un sistema. En la figura 4 se muestra el diagrama de bloques de la estructura de un sensor MPU9150. Los diagramas de bloques ofrecen cierta versatilidad para modelar sistemas, ya que adentro de un bloque podemos colocar más bloques. Es una forma apropiada para representar sistemas que se componen de otros sistemas.

Figura 4. **Estructura interna del sensor MPU9150**



Fuente: *MPU-9150 Product Specification. Document Number PS-MPU-9150A-00. Revisión 4.3*. https://strawberry-linux.com/pub/MPU-9150_DataSheet_V4%203.pdf. Consulta: junio de 2015.

El ejemplo del sensor MPU9150 visto como un sistema, lleva a dar una definición alternativa de sistema.

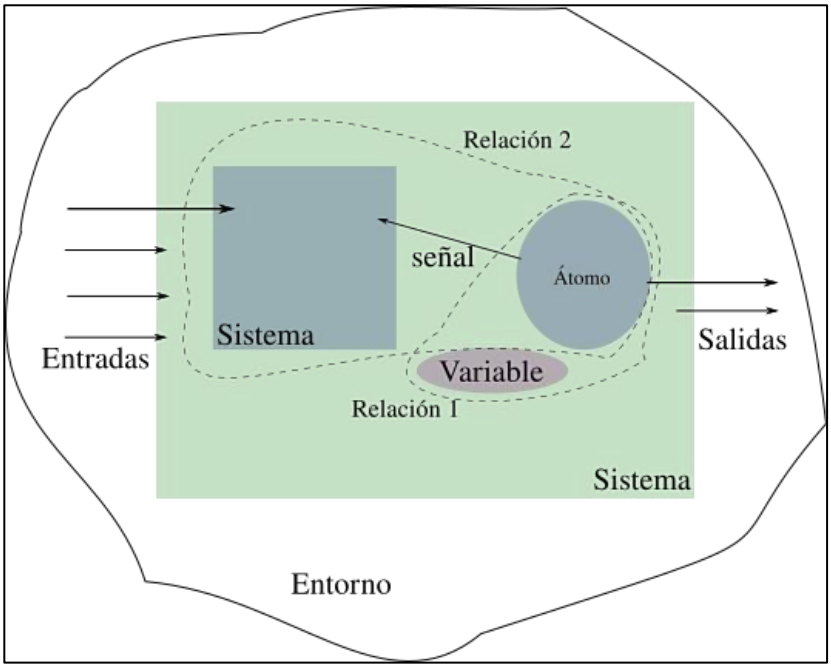
Tabla VII. **Definición de sistema**

Un sistema *S* es un conjunto formado por:
○ Sistemas ○ Señales ○ Variables (mínimo: 1) ○ Fenómenos (mínimo: 1) ○ Relación entre elementos internos, o elementos del entorno y elementos internos

Fuente: elaboración propia.

Un elemento a tal que $a \in S$ y no es un sistema se le llama átomo o elemento minimal de un sistema.

Figura 5. **Sistema compuesto por relaciones entre sus elementos, señales, variables, átomos y otros sistemas**



Fuente: elaboración propia, empleando Inkscape.

La figura 5 ilustra la definición 7. Como resultado al ejemplo del sensor y la definición 7 se tiene como resultado la definición 8.

Tabla VIII. **Definición de subsistema y supersistemas**

Un sistema *K* con elementos que también cumplen con la definición 7, es un supersistema. A un elemento *S* tal que $S \epsilon K$ y cumple con la definición y se le llamará subsistema.

Fuente: elaboración propia.

Para un subsistema *S*, el resto de subsistemas y el entorno del supersistema *K* forman el entorno de *S*. Así como en la figura 4, el entorno del sistema DMP es el resto de componentes internos del sensor y el entorno del sensor, que representa un supersistema para el sistema DMP. Sin embargo, al momento de desarrollar un proyecto más grande que involucre a varios de estos sensores, sería útil tratar al sensor como un subsistema del proyecto mayor. De igual forma que en el ejemplo del globo, que es un sistema, un átomo o elemento minimal, un supersistema o un subsistema depende del observador. Por ejemplo, un automóvil, para un observador que analiza su funcionamiento puede representar un supersistema, pero para otro que analiza el tráfico en una avenida, representa un elemento minimal.

1.4. Sistemas abiertos, sistemas cerrados y sistemas controlados

El modelo del sistema solar regido por las leyes de Kepler cumple con la definición de un sistema cerrado. Primero que nada, esta idealización del sistema solar cumple con la definición de sistema; es un conjunto elementos el sol y los planetas desde Mercurio a Neptuno; además, se encuentra limitado al espacio geométrico de las trayectorias de los planetas en torno al sol en un espacio euclidiano cuyo origen es el centro de masa de este último. En esta

idealización del sistema solar, el entorno y el resto del universo no se toma en cuenta y por tanto no hay ninguna entrada del sistema que provenga de su entorno.

Tabla IX. **Definición de sistema cerrado**

Sea *S* un sistema para un observador, *S* es un sistema cerrado si y solo sí no tiene señales de entrada.

Fuente: elaboración propia.

Tabla X. **Definición de sistema abierto**

Sea *S* un sistema para un observador, *S* es un sistema abierto si y solo sí posee señales de entrada y mantiene cierta interacción con su entorno. Puede o no tener señales de salida

Fuente: elaboración propia.

El vehículo de los ejemplos anteriores es un claro ejemplo de sistema abierto. Recibe entradas de un usuario ajeno a él, también recibe entradas del paisaje por donde transita, tal como una piedra, un bache o un túmulo.

Tabla XI. **Definición de control**

Sea *K* un supersistema formado por un subsistema *C* y un subsistema *S*, decimos que *K* es un sistema de control, si *C*, quien recibe el nombre de sistema controlador o planta de control, genera señales capaces de cambiar el comportamiento o estado de *S*, quien recibe el nombre de sistema controlado.

Fuente: HEYLIGHEN, Francis and VRANCKX, *An. Principia cybernetica web, web dictionary of cybernetics and systems.* p. 130.

Los sistemas controlados se encuentran en casi cualquier aplicación industrial; una faja moviéndose a una velocidad controlada para transportar envases en una fábrica; un servomotor el cual es un motor de corriente directa con un mecanismo para llevarlo a una posición deseada; la tensión del motor es manipulada para sostener la posición. Incluso se encuentran sistemas controlados en la misma naturaleza. El caminar implica un control que lleva a cabo el cuerpo para evitar perder el equilibrio, el vuelo de las aves cuando migran lleva un control para evitar que colisionen o se pierdan En el caso del ser humano al caminar, la variable que se encuentra bajo control es el equilibrio, en el caso de las aves, su posición en el aire. Detrás de todos estos ejemplos existen sistemas que generan señales que cambian las variables de los sistemas controlados, los cuales reciben el nombre de planta de control. En ambos casos, la planta de control es el sistema nervioso; en el caso del servomotor, un circuito PID es el encargado de generar las señales para manipular su estado.

Tabla XII. **Definición de *feedback***

Sea *K* un sistema de control, *C* el controlador y *S* el sistema controlado, subsistemas de *K*. Si existen señales de *S* que son procesadas por *C* y afectan de algún modo a las señales que genera *C* para cambiar el estado o comportamiento de *S*. *K* es un Lazo cerrado, un sistema con *Feedback* o un sistema retroalimentado. Si ninguna de las señales *S* afecta de algún modo las señales de *C. K* es un Lazo abierto o un sistema sin *Feedback*.

Fuente: HEYLIGHEN, Francis and VRANCKX, *An. Principia cybernetica web, web dictionary of cybernetics and systems.* p. 130.

No hay mucho que decir de los lazos abiertos, no es difícil de ver que estos sistemas no pueden satisfacer muchos criterios, o generar comportamientos estables o críticos. Por ejemplo, al quemar fuegos pirotécnicos, la persona que los prende genera una señal de energía que cambia su estado. En ningún momento las señales que estos generan vuelven

o regresan de alguna forma al sistema. Un divisor de tensión colocado a una fuente con el fin de tener un valor de tensión constante a su salida, no hay forma de evitar que este reduzca su tensión, al colocar una cara en su salida. Al contrario, si el regulador se encuentra retroalimentado, el sistema será más estable a perturbaciones. Los efectos de la retroalimentación o *feedback* se hacen presentes en el comportamiento de los sistemas controlados, y afectan propiedades como la estabilidad, ganancia global, perturbaciones o sensibilidad. Un ejemplo de lazo cerrado es el proceso de regulación de glucosa que lleva a cabo el cuerpo humano. Órganos como el páncreas o el hígado generan hormonas encargadas de regularla. En los lazos cerrados artificiales, se puede encontrar una gran variedad de ejemplos, casi cualquier aplicación industrial o casera llevan un lazo cerrado. El refrigerador es un ejemplo, mantiene la temperatura interior, la sensa y manipula su comportamiento.

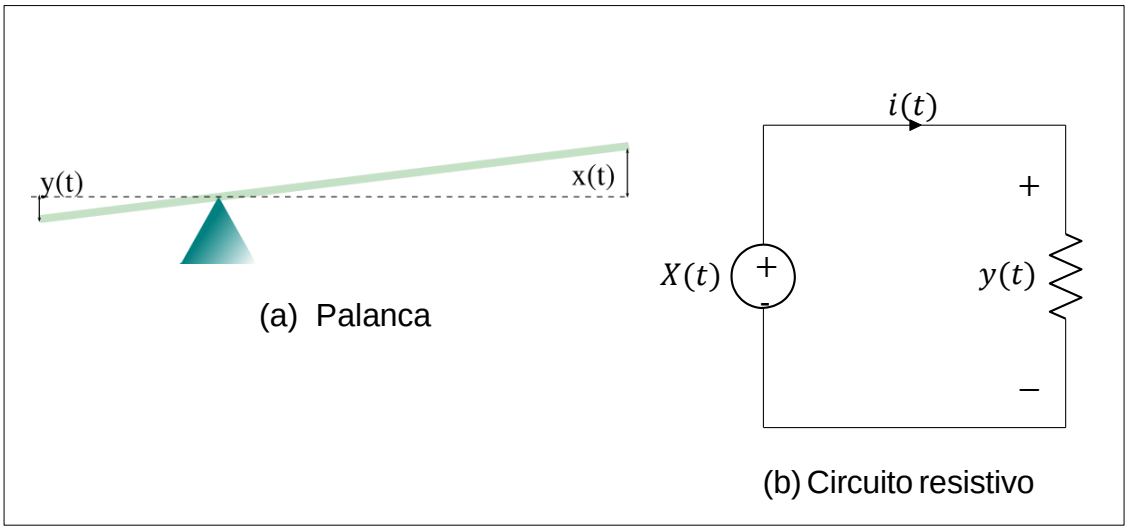
1.4.1. Tipos de sistemas de control

Cuando se habla de un sistema dinámico o simplemente de una dinámica, se hace referencia a todo aquel sistema cuyo comportamiento actual depende de acciones pasadas. El comportamiento de un sistema masa-resorte depende del estado inicial del resorte, el comportamiento de un circuito RLC depende de la carga o corriente almacenada en los componentes reactivos (capacitancias e inductancias), éstos son ejemplos de sistemas dinámicos. Si el sistema depende únicamente de la entrada actual se le llama sistema estático o simplemente estática.

En el caso del circuito RLC, si se retiran los componentes reactivos del circuito se obtiene un circuito puramente resistivo y el comportamiento ya no depende de un estado inicial, sino únicamente de la fuente conectada al circuito. Este circuito puramente resistivo es un sistema estático. Otro ejemplo

de un sistema estático es una palanca, la posición del otro extremo solo depende de la posición del extremo que se está manipulando, sin importar donde estaba anteriormente. Dinámica en sistemas controlados no necesariamente implica movimiento y estática no necesariamente implica reposo.

Figura 6. **Sistema estático**



Fuente: elaboración propia, empleando Inkscape.

De acuerdo al tipo de señales que un sistema maneje, es posible clasificarlos como eléctricos, en caso de tener solamente señales eléctricas, mecánicos en el caso de tener solamente señales mecánicas, como una bicicleta; neumáticos como una pistola de clavos; hidráulicos como el sistema cardiovascular, entre muchos otros. En el caso de tener señales de distinta naturaleza, serían sistemas híbridos. También se pueden clasificar aquellos sistemas donde el tiempo es una variable independiente, como Sistemas de tiempo continuo, o Sistemas de tiempo discreto. En los sistemas de tiempo continuo las señales pueden tomar valores en cualquier instante en un intervalo

continuo de tiempo; en los sistemas de tiempo discreto, las señales pueden tomar valores solo en un conjunto contable de valores; que por lo general son resultado de un muestreo de una señal de tiempo continuo. Por el número de salidas y entradas que tiene un sistema se pueden mencionar los siguientes tipos:

- SISO (*single input-single output)*: sistemas con una sola entrada y una sola salida.
- SIMO (*single input-multiple outputs*): sistemas con una sola entrada y más de una salida.
- MISO (*multiple inputs-single output*): sistemas con más de una entrada y una sola salida.
- MIMO (*multiple inputs-multiple outputs*): sistemas con más de una entrada y más de una salida.

Según la posibilidad de poder predecir el comportamiento de un sistema, se puede decir que hay dos clases de sistemas:

- Sistemas deterministas los cuales su comportamiento futuro es predecible dentro límites de tolerancia.
- Sistemas estocásticos los cuales resulta imposible predecir el comportamiento futuro. Estos sistemas tienen más de una variable aleatoria. Los sistemas deterministas, cuyo comportamiento es muy difícil de predecir, a pesar que es posible hacerlo, se les llama sistemas caóticos.

1.5. ¿Por qué utilizar computadoras para el control de sistemas físicos?

Las computadoras ofrecen ventajas y facilidades a la hora de implementar un sistema, ya que al momento de utilizarlas se está implementando un sistema digital, que muy superior en la mayoría de a uno analógico. Es posible mencionar algunas ventajas como la reproducibilidad de los resultados; un sistema digital diseñado correctamente permite tener una misma salida para el mismo conjunto de entradas; un circuito analógico depende de la temperatura, voltaje de alimentación, antigüedad de los componentes, incertezas de los valores de los componentes, entre otros.

La flexibilidad es otra ventaja, se puede adaptar un sistema a otro entorno diferente, muchas veces haciendo cambios solamente en código, a diferencia de un sistema analógico en el cual se necesitaría un diseño completamente nuevo del sistema. La escalabilidad; si un sistema empieza a involucrar más variables, es posible agregar otros sistemas y hacer que éstos se comuniquen entre ellos mediante un protocolo con el fin de tomar en cuenta las nuevas variables. El precio y economía hacen que los sistemas digitales se posicionen sobre los analógicos. Se puede escoger la unidad de procesamiento acorde al sistema que se utilizará.

En los últimos, años los precios en los sistemas de computación han disminuido notoriamente. El almacenamiento de la información y comunicación con el mundo exterior, ésta ventaja pone a la cabeza sistemas que utilizan computadoras. Es posible almacenar en ellas gran cantidad de datos para llevar registro del comportamiento del sistema y así poder acceder y manipular dicha información desde cualquier parte del mundo.

1.6. Interacción máquina-entorno y sistemas físico-cibernéticos

Un sistema de interacción máquina-entorno es un sistema en el que máquinas diseñadas para llevar control de entidades o magnitudes físicas se comunican entre sí para llevar a cabo uno o varios objetivos. Las aplicaciones actuales necesitan de un control más riguroso, por lo que en los últimos años se ha desarrollado dicho control con computadoras, ya que estas resuelven en gran manera la optimización en dichos sistemas. Cerca del año 2006, Hellen Gill desarrollará, un nuevo término, sistemas físico-cibernéticos, o por sus siglas en inglés CPS (*cyber-physical system*), que engloba dichos sistemas. De hecho, dicho término involucra más que un sistema de control llevado a cabo con computadoras, engloba desde redes, computadoras y modelos físicos hasta la interacción final con el humano que maneja o tiene alguna especie de relación con dicho sistema. Podría resumirse como la intersección de lo físico y lo cibernético.

1.6.1. Ejemplos de un CPS

El siguiente ejemplo ofrece una mejor idea de una aplicación del término CPS. En un futuro cercano, dichos ejemplos podrán ser implementados, alargando así la esperanza de vida de los que tengan acceso a dicha tecnología.

Ejemplo:

- La cirugía a corazón abierto frecuentemente requiere detener el corazón, realizar la cirugía, y luego reiniciar el corazón. Es extremadamente peligrosa y conlleva muchos efectos secundarios. Un número de equipos de investigación han estado trabajando en una alternativa donde el

cirujano puede operar en un corazón latiendo en lugar de detenerlo. Existen dos ideas clave que hacen esto posible. “Primero, los instrumentos pueden ser robóticamente controlados de manera que ellos se muevan con el movimiento del corazón”[1]. Un cirujano puede entonces usar una herramienta y aplicar presión constante en un punto del corazón mientras continúa latiendo.

Segundo, un sistema de video estereoscópico puede presentar al cirujano una ilusión de video de un corazón detenido. Para el cirujano el corazón parece estar detenido, pero en realidad, el corazón continúa latiendo. Para realizar dicho sistema quirúrgico se requiere un extensivo modelo del corazón, las herramientas, el hardware de computación y software. Especializado que requiere un cuidadoso diseño que asegura precisión en el tiempo y seguridad ante comportamiento errático para manejo de mal funcionamiento. Requiere de un detallado análisis de los modelos y los diseños para proveer una alta confianza.

El ejemplo anterior ilustra la aplicación de un CPS en el campo de la medicina. Es posible encontrar ejemplos en el campo de la automatización y robótica.

Los accidentes aéreos resultan en terribles tragedias. Existe un proyecto llamado SoftWalls, que tiene como objetivo evitar algunos de estos accidentes,

Ejemplo:

- En 2001, Edward Lee propuso una nueva tecnología para control de vuelo llamada SoftWalls. La estrategia de SoftWalls es almacenar una

[1] LEE, Edwar; SEESHIA, Sanjit. *Introduction to embedded systems*. p. 97.

base de datos 3D de llamadas *no-fly zones* o espacios restringidos, a bordo cada aeronave y hace cumplir el espacio restringido usando el sistema de control de la aeronave. Cada aeronave tendrá su propio sistema SoftWalls. También, la base de datos requerirá una firma digital para actualizar las *no-fly zones*, de tal forma que el sistema sea imposible de creackear. SoftWalls no es un sistema de control autónomo. No remueve las entradas del piloto cuando la aeronave se dirige a una *no-fly zone*, en vez de eso lugar el controlador añade una desviación o tendencia a las entradas del piloto y nunca remueve su autoridad.

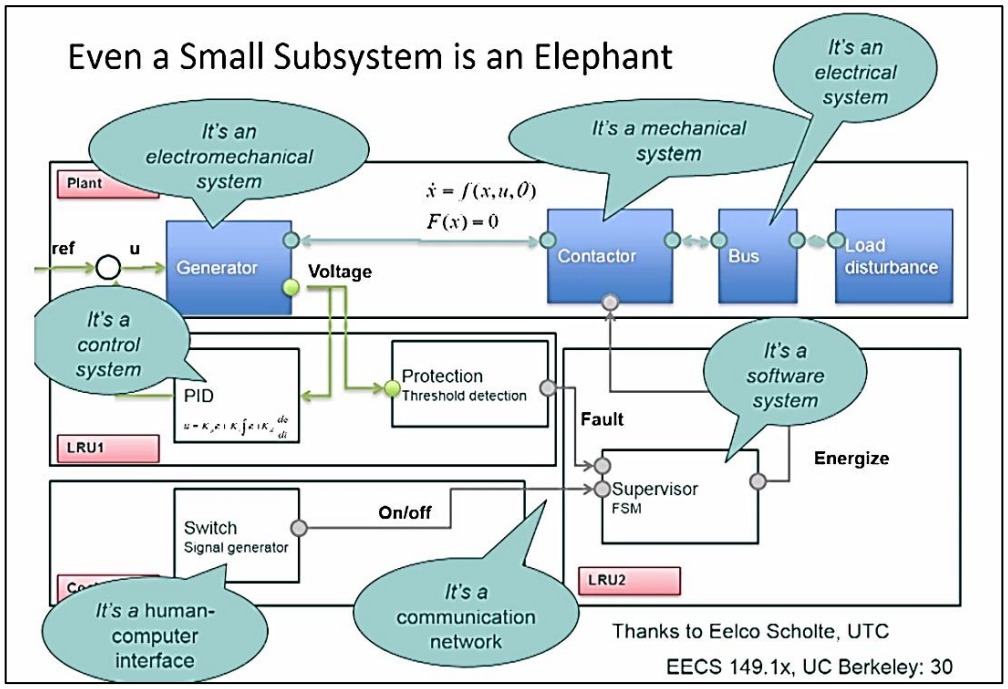
Un piloto que se aproxima a una *no-fly zone* y mantiene su curso de vuelo, será desviado hasta que sea seguro continuar en el curso de vuelo. Un piloto que escoja hacerlo de manera más rápida podrá hacerlo. Un piloto que intente llegar a una *no-fly zones* le será imposible. A través de esto, Soft Walls, maximizará la autoridad del piloto sujeto a la restricción de que ninguna *no-fly zones* es permitida (el sistema puede incluir condiciones para la comodidad.) Esto hará que el piloto tenga una mayor maniobrabilidad en el caso de una emergencia. Aunque SoftWalls no es una estrategia de control autónoma, se encuentra relacionada al control autónomo de la aeronave poeque es un problema de evasión de colisiones.

Ambos ejemplos representan escenarios futuristas de los CPS. Estos ejemplos serán una realidad en unos cuantos años, y el ingeniero practicante debe conocer los conceptos básicos que le permitan enfrentarse a problemas relacionados con estos sistemas.

1.7. Partes de un CPS

Divide et impera, dividir el sistema complejo que implica un CPS en subsistemas es la forma más conveniente de construir en CPS. Haciendo funcionar cada uno de ellos por separado y luego pasando a la totalidad del CPS.

Figura 7. **División de un CPS en subsistemas**



Fuente: *Cyber-Physical Systems, BerkeleyX - EECS149.1x. Lecture: Challenges of CPS Design. Plataforma EDX*. p. 88.

La figura 7 tiene un enunciado que dice: "*even a small subsystem is an elephant*", que al traducirlo al español dice: "aún un pequeño subsistema es un elefante", para hacer evidente el hecho que cualquier CPS, por más pequeño

que sea, puede descomponerse en varios subsistemas. Entre los subsistemas de un CPS destacan:

- Sistemas mecánicos: son sistemas con solamente señales lumínicas, mecánicas, hidráulicas, neumáticas o térmicas, generalmente son sistemas cuyas salidas y entradas se relacionan con el entorno. Una caja de engranajes o toda la estructura aerodinámica de un avión son ejemplos de estos sistemas.

- Sistemas electromecánicos: son sistemas con señales mecánicas, lumínicas, hidráulicas, neumáticas o térmicas y señales eléctricas. Generalmente las salidas y entradas en estos se relacionan con el entorno, pero también cuentan con señales internas de los CPS. Ejemplos de estos sistemas son toda clase de sensores que generan señales eléctricas, un teclado, un motor eléctrico una electro-válvula. Cualquier elemento capaz de transformar energía eléctrica en energía neumática, mecánica, térmica, lumínica o hidráulica en la activación o ejecución de un proceso con la finalidad de generar algún efecto sobre otro sistema (generalmente mecánico), llamado actuador.

- Sistemas eléctricos: son sistemas con solamente señales eléctricas, generalmente las salidas y entradas son señales internas del CPS. Representan una capa intermedia entre los sistemas electromecánicos y sistemas de software y control. En ellos encontramos señales de voltaje y corriente.

- Sistemas de control: en un CPS, se encarga de procesar señales eléctricas provenientes de sensores en señales eléctricas que van hacia el sistema electromecánico. Puede ser construidos usando software o

hardware. Para lograr mejores resultados en los CPS se usan tecnologías digitales.

- Sistemas de software: sistemas compuestos solamente por señales y variables lógicas. Lo describen algoritmos y generalmente se ejecutan dentro de una computadora. Son sistemas sumamente abstractos y muy amplios. Se encargan de manejo de protocolos, interfaces de usuario, interfaces entre subsistemas de propios. Almacenan datos y los procesan. Los sistemas de control o de comunicación pueden ser subsistemas de este sistema.

- Redes de comunicación: representan sistemas embebidos en los sistemas de software y sistemas eléctricos cuyo objetivo es comunicar el CPS con otros CPS o sistemas de información.

- Interfaces humano-máquina: representan sistemas embebidos en los sistemas mecánicos, electromecánicos, eléctricos y de software que le presentan una forma de comunicarse con el CPS a un usuario final.

1.8. Ciclo de construcción de un CPS

Generalmente, se modela el proceso de construcción de un CPS en tres fases: modelo, diseño y análisis. Se encuentran conectadas entre sí formando un ciclo.

1.8.1. Modelo, ¿qué hace nuestro sistema?

Los modelos reflejan propiedades del sistema. Como primer paso, al construir un CPS se debe definir el conjunto de variables que describen las características dinámicas del sistema. Las leyes físicas que gobiernan la

operación de los sistemas en la vida real, resultan la mayor parte del tiempo bastante complejas. Debido a esto, es necesario utilizar la simulación de los sistemas. Mediante el uso de métodos numéricos y una computadora es posible predecir hasta cierto punto el comportamiento del sistema. Modelar es ganar un entendimiento más profundo mediante la imitación.

1.8.2. Diseño, ¿cómo lo hace nuestro sistema?

El diseño es la construcción estructurada de artefactos que manipulen o interpreten el sistema físico. Esta parte requiere una gran cantidad de conocimientos técnicos por parte de las personas que diseñan un sistema físico-cibernético, que abarcan desde conocimiento sobre computadoras, teoría de circuitos, conocimientos de la física del sistema, redes y técnicas de control.

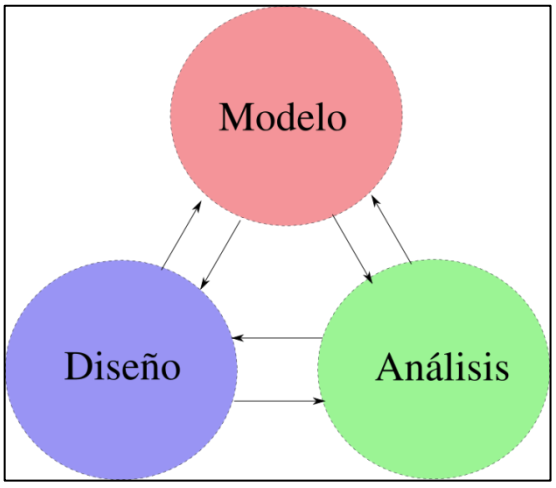
1.8.3. Análisis, ¿por qué funciona o falla nuestro sistema?

El análisis es el proceso de ganar un entendimiento más profundo mediante la disección del sistema en sistemas más pequeños. Cada sistema debe estar diseñado para cumplir con ciertos requerimientos. Existen diversas técnicas que nos permiten alcanzar dichos requerimientos. Esta parte se encuentra enlazada con la parte de modelo. Al construir un sistema se realiza un modelo para simularlo y estudiarlo.

Con base en los resultados arrojados, se realiza un análisis del sistema ya modelado y se verifica que pueda cumplir con los requerimientos. Luego se modifica el diseño; es principalmente por esta razón que se usarán sistemas digitales y computadoras en los sistemas de control, por la relativa flexibilidad que estos ofrecen para modificar un sistema ya construido.

Al construir un CPS, se realiza un modelo del sistema a construir y se hacen simulaciones para predecir su comportamiento. Luego, se procede a la fase de implementación en la que surgirán problemas que no se tomaron en cuenta durante la fase de modelo y simulación y se regresa a la fase del modelo; se modelan y simulan los cambios propuestos y ya una vez modelado y simulado el sistema con los cambios, se implementan y se procede a la etapa de análisis, donde se verifica que el sistema cumpla con ciertos requerimientos. Si el sistema no los cumple, se ajusta el diseño y se regresa de nuevo a la etapa de análisis. Si el sistema cumple con los requisitos, en determinado entorno y condiciones, es necesario pasar a la etapa de modelo y simulación pero ahora para modelar y simular un sistema más complejo que tome en cuenta las condiciones y el entorno, haciendo de la construcción de un CPS un ciclo, pasando de una fase a otra con el fin de lograr un sistema que haga lo esperado en las condiciones necesarias.

Figura 8. **Ciclo de la construcción de un CPS**



Fuente: elaboración propia, empleando Inkscape.

1.9. Prácticas de laboratorio propuestas módulo 1, instalando el software necesario

El uso de computadoras ha facilitado la resolución de problemas y el área de los sistemas de control no es la excepción. En este laboratorio se utilizará una computadora para realizar simulaciones de los sistemas físicos. Gracias a la velocidad de procesamiento, es posible obtener soluciones utilizando métodos numéricos de las ecuaciones que los modelan. Se utilizará de igual forma una computadora para diseñar el *firmware* que correrá en un microcontrolador para el control de un sistema físico. Además, se utilizará para analizar el sistema de control y obtener datos de él. Para todo lo mencionado anteriormente es necesario instalar en ella el software adecuado para realizar dichas tareas.

1.9.1. Práctica 1, instalar el software

El primer laboratorio consiste en instalar el software necesario para realizar los diagramas, *firmware* y modelos durante el curso.

- SCILAB: una herramienta de código abierto, diseñada para el análisis numérico por computadora y un lenguaje de programación de alto nivel orientado al análisis numérico, muy similar a MATLAB. En la instalación debe asegurarse que se encuentre instalado XCOS, que es una suite para sistemas de control.

- Un IDE de C para ARM: si desea puede instalar Code Composer IDE desarrollado por Texas Instruments, para la programación de sus diversos productos. En este laboratorio se usará la tarjeta de desarrollo TivaC de Texas Instruments. El IDE anterior es cerrado y necesita licencia. Si se desea es posible configurar Eclipse con GNUToolchain

obteniendo así un IDE muy similar a CCS pero de código abierto, sin licencia ni restricciones de memoria, véase el siguiente enlace. Configuración GNUToolchain y Eclipse.

- TivaWare: librería de código abierto, con licencia *Royalty-free* desarrollada por Texas Instruments para algunos de sus microcontroladores. Incluye una gran cantidad de funciones para uso de periféricos, manipulación de sensores y desarrollo de aplicaciones.

- Python versión 2.7: Es posible descargar algún IDE o editor para facilitar el uso de este lenguaje. Eclipse con el plugin PyDev es recomendable.

- PySerial: SciPy y Numpy para Python 2.7. Librerías útiles para el desarrollo de aplicaciones técnicas y científicas.

- GNUPlot: una poderosa herramienta para realizar gráficas.

2. MODELO Y SIMULACIÓN DE SISTEMAS FÍSICOS. DINÁMICAS CONTINUAS

Etimológicamente, el término modelo proviene del latín *modus*, que significa pequeña medida. Esto da una idea general de qué es un modelo. Al hablar de sistemas físico-cibernéticos o CPS, es importante controlar algunos fenómenos que suceden en la naturaleza y para ello primero es necesario saber cómo se comportan dichos fenómenos. Muchos fenómenos son modelados por diversas teorías de la física, es posible encontrar algunos que solamente pueden ser modelados por la mecánica clásica, otros por la teoría electromagnética, otros por las leyes de la termodinámica u otros por la mecánica cuántica. También es posible encontrar fenómenos o sistemas muy complejos que a veces requieren conceptos de varias teorías para modelar su comportamiento, abarcar cada una de las distintas teorías de la física.

Con el modelo, se obtiene una aproximación de la realidad. Es muy difícil tomar en cuenta todas las variables posibles en el modelo. Si bien los modelos no son copias de la realidad, ellos reducen en algún sentido la complejidad y permiten medir propiedades. Noam Chomsky, en una entrevista realizada por Alexander Cockburn, en otoño de 1994, se le preguntó qué era un modelo a lo que él respondió:

> Cuando tú estudias objetos en la naturaleza, tienes que abstraer del fenómeno lo irrelevante que obscurece la naturaleza. Esto es llamado idealización (lo cual es un tanto erróneo porque esto te lleva más cerca de la realidad). Si estudias los planetas, por ejemplo, ayuda pensar que ellos son puntos que tienen masa y se mueven en trayectorias elípticas alrededor de otros puntos. Claro, los planetas no son puntos –un punto no tiene dimensiones –pero si los tratas como tal, puedes predecir y entender el Sistema Solar más claramente. Eso es un modelo. Los científicos tienen que hacer esto todo el tiempo al estudiar un fenómeno complejo– que es por eso que ellos hacen experimentos en lugar de

> tomar fotografías de cualquier cosa fuera de sus horizontes. Ellos construyen modelos en los que capturan los aspectos cruciales del mundo mientras hacen a un lado las irrelevancias. Tú puedes construir modelos que no tienen relación a la realidad. Algunos modelos de libre mercado, por ejemplo, quitan algunos factores significativos, como la intervención del Estado, el hecho que el capital es móvil y la mano de obra relativamente inmóvil y así sucesivamente. De hecho, ellos te llevan más lejos de la forma en el que el mundo realmente funciona. El proceso de construir modelos no es uno mecánico. Como cuando tú comienzas a hacer estudios serios en ciencia, tú no aprendes reglas, tú ganas ciertas habilidades. Una terrible cantidad de ellas es percepción, intuición e imaginación[2].

Al construir un CPS, es imprudente aventurarse a diseñar sin antes realizar un modelo del fenómeno que se quiere manipular o construir. Para que un modelo sea verdaderamente útil, es necesario que sea posible de alguna forma representarlo o describirlo matemáticamente.

Muchos modelos se pueden representar con un conjunto de ecuaciones, pero solo un conjunto extremadamente reducido de ellos es predecible de forma analítica. En el resto de los casos, se recurre a la simulación, tal como lo hace ver Vannevar Bush.

El análisis matemático es muy importante, pero es necesario hacer una serie de simplificaciones y asunciones para que el modelo pueda ser aplicable. Las asunciones no siempre pueden ser justificadas y muchas no pueden ser justificadas más que a través de experimentos o simulaciones. Existen muchas definiciones alrededor del término simulación. Ceiller, en su libro *Continuos System Modelling* desarrolla los conceptos de modelo, experimento y simulación. Un sistema es una fuente potencial de datos. Un experimento es el proceso de extraer datos de un sistema. Un modelo *M* para un sistema *S* y un experimento *E* es cualquier cosa a la que *E* se puede aplicar con el fin de responder preguntas de *S.* Una simulación es un experimento realizado en el modelo.

[2] CHOMSKY, Noam; COCKBURN, Alexander. *The golden age is in us*, *Noam Chomsky intervicurd by Alexander Cockburn*. https://chomsky.info/19940622/. Consulta: octubre de 2015.

Una simulación es una herramienta para extraer datos de un modelo, de la misma forma que lo es un osciloscopio para la realidad. Una muy buena herramienta es software de simulación, que utiliza métodos numéricos. La definición en la publicación de Ceiller no implica que sea la única forma de hacerlo, pero si lo hace una forma capaz de adaptarse a la mayoría de modelos.

En este capítulo, se hará énfasis en modelos y simulaciones de problemas mecánicos y circuitos eléctricos. El movimiento mecánico y el estado de circuitos pasivos, usualmente se modela usando ecuaciones diferenciales o su equivalente en ecuaciones integrales. Dichos modelos funcionan bien para movimientos suaves o comportamientos continuos. Pero no son adecuados cuando se tienen comportamientos no continuos, como colisiones mecánicas o algunos circuitos activos. Dichas colisiones o circuitos pueden modelarse como eventos discretos e instantáneos. Es conveniente cambiar los modelos de ecuaciones diferenciales a bloques de comportamiento o modo. A los problemas que mezclan modelos discretos y continuos se les llama modelos híbridos.

2.1. Variables de estado, ecuaciones de movimiento y marcos de referencia

El estado de un sistema se refiere a las condiciones pasadas, presentes y futuras del sistema. Las variables de estado de un sistema se definen como un conjunto mínimo de variables, cuyo conocimiento en cualquier tiempo t_0, y el conocimiento de la información de la entrada de excitación que se aplica luego de t_0, son suficientes para determinar el estado del sistema en cualquier tiempo $t > 0$. Las ecuaciones de movimiento son una formulación matemática que predice la evolución temporal de un sistema físico en un espacio determinado.

Estas ecuaciones relacionan varias variables que caracterizan el estado físico del sistema (aceleración, velocidad, masa, inercia, entre otras) con otras magnitudes físicas que pueden provocar un cambio de estado en el sistema (Fuerza de Hooke en un resorte, por ejemplo).

Se puede ver a las ecuaciones de movimiento como descripciones matemáticas del sistema en términos de variables dinámicas. Normalmente se utilizan coordenadas de un sistema de referencia y de tiempo, aunque muchas veces se pueden encontrar en términos del momento. El movimiento de un cuerpo puede ser descrito únicamente en relación a algo más, como otros cuerpos, observadores o un conjunto de coordenadas en el tiempo y el espacio. Ese algo es llamado marco de referencia. Un marco de referencia inercial es aquel en que la variación del momento lineal del sistema es igual a las fuerzas reales sobre el sistema. En la historia, el primer ejemplo de ecuación de movimiento que se introdujo en la física fue la segunda ley de Newton, una definición alternativa para un marco de referencia inercial es aquél en el que se cumplen las leyes de Newton.

2.1.1. Mecánica newtoniana

La mecánica es la parte de la física que estudia el movimiento. La mecánica newtoniana también es conocida como mecánica vectorial y es una formulación de la mecánica clásica que estudia el movimiento de partículas y sólidos en un espacio tridimensional euclidiano. Encuentra sus bases en las transformaciones de Galileo y las leyes de Newton. En 1686, Isaac Newton presentó tres leyes de movimiento en el *Principia Mathematica Philosophae Naturalis*, las cuales, junto con las transformaciones de Galileo, dieron origen a toda esta teoría. Las leyes de Newton son:

- "Un cuerpo permanece en reposo o en movimiento constante, a no ser que sea obligado a cambiar su estado por la acción sobre él de alguna fuerza.
- Un cuerpo con una fuerza actuando sobre él, se mueve de tal forma que la razón de cambio de su momento siempre es igual al de la fuerza.
- Si dos cuerpos ejercen fuerzas uno sobre el otro, estas fuerzas son iguales en magnitud y contrarias en dirección"[3].

La mecánica newtoniana es un modelo físico-macroscópico, utilizado para describir el movimiento de los cuerpos en el espacio relacionando este movimiento con sus fuerzas. Históricamente, la mecánica newtoniana fue el primer modelo dinámico capaz de hacer predicciones importantes sobre el movimiento de los cuerpos, incluyendo las trayectorias de los planetas. Ésta es suficientemente válida para la mayoría de los casos prácticos cotidianos en una gran cantidad de sistemas. Esta teoría, por ejemplo, describe con gran exactitud sistemas como cohetes, movimiento de planetas, moléculas orgánicas, trompos, trenes y trayectorias de móviles en general:

$$\boldsymbol{v} = \frac{d\boldsymbol{r}}{dt} \qquad \text{[Ec. 2.1]}$$

De la misma forma se llama aceleración **a**, a la razón de cambio en el tiempo de la distancia **v**. Y por eso se tiene:

$$\boldsymbol{a} = \frac{d\boldsymbol{v}}{dt} \qquad \text{[Ec. 2.2]}$$

El momento o ímpetu p es una medida del estado de un cuerpo o partícula y se define como la velocidad v escalada por la masa *m* de la partícula o cuerpo:

$$\boldsymbol{p} = m\boldsymbol{v} \qquad \text{[Ec. 2.3]}$$

[3] NEWTON, Isaac. *Principia Mathematica Philosophae Naturalis.* p. 45.

De la segunda ley de Newton se tiene, una representación matemática para la fuerza:

$$\boldsymbol{F} = \frac{d\boldsymbol{p}}{dt} \qquad \text{[Ec. 2.4]}$$

Si se supone que la masa es constante, se obtiene:

$$\boldsymbol{F} = \frac{d\boldsymbol{p}}{dt} = m\frac{d\boldsymbol{v}}{dt} = m\boldsymbol{a} \qquad \text{[Ec. 2.5]}$$

Se dice que una fuerza **F** realiza trabajo, cuando altera el estado de movimiento en un cuerpo y es equivalente a la energía necesaria para desplazarlo de manera acelerada. Se llama trabajo elemental, δW, de la fuerza **F** durante el desplazamiento elemental ±**r** al producto escalar $\boldsymbol{F} \cdot \delta\boldsymbol{r}$; esto es:

$$\delta W = \boldsymbol{F} \cdot \delta\boldsymbol{r} \qquad \text{[Ec. 2.6]}$$

Cuando la fuerza se aplica sobre la partícula, en una trayectoria C:

$$W = \int c\ \boldsymbol{F} \cdot d\boldsymbol{r} \qquad \text{[Ec. 2.7]}$$

Si la fuerza es conservativa, el trabajo a lo largo de cualquier camino *C* sólo depende del punto inicial y el punto final, siendo totalmente independiente del resto de la trayectoria. Si $\boldsymbol{F}$ es conservativa y C es cualquier trayectoria que comienza en A y finaliza en B.

$$W = \int_A^B \boldsymbol{F} \cdot d\boldsymbol{r} \qquad \text{[Ec. 2.8]}$$

De esa cuenta se sabe que existe una función *V*: $R^3 \rightarrow R$ que depende de la posición $\boldsymbol{r}$, llamada potencial de $\boldsymbol{F}$; tal que:

$$F = -\nabla V(r) \qquad \text{[Ec. 2.9]}$$

Recibe ese nombre debido a que representa a la energía potencial. Energía que un objeto, debido a su posición en un campo de fuerza o un sistema tiene por la configuración de sus partes. Como $\boldsymbol{r}$ es una función del tiempo, y si se supone que la masa no cambia durante todo el trayecto de *C*.

$$\int_c \boldsymbol{F} \cdot d\boldsymbol{r} = \int_0^t \boldsymbol{F} \cdot \boldsymbol{v} dt = \int_0^t \frac{d\boldsymbol{p}}{dt} \cdot \boldsymbol{v} dt = \int_0^t \boldsymbol{v} \cdot d\boldsymbol{p} = \int_0^t \boldsymbol{v} \cdot d(m\boldsymbol{v}) =$$
$$= m \int_0^t \boldsymbol{v} \cdot d\boldsymbol{v}$$

[Ec. 2.10]

Si se sabe que la norma al cuadrado un vector, es el producto escalar consigo mismo:

$$d\boldsymbol{v}^2 = d(\boldsymbol{v} \cdot \boldsymbol{v}) = d\boldsymbol{v} \cdot \boldsymbol{v} + \boldsymbol{v} \cdot d\boldsymbol{v} = 2\boldsymbol{v} \cdot d\boldsymbol{v} \qquad \text{[Ec. 2.11]}$$

Se pueden relacionar los diferenciales de la siguiente forma:

$$\boldsymbol{v} \cdot d\boldsymbol{v} = \frac{1}{2} dv^2 \qquad \text{[Ec. 2.12]}$$

Si la fuerza es conservativa, el resultado de la integral solo depende del punto de inicio *A*, el cual lleva en ese punto una velocidad $\boldsymbol{v}A$, y del punto de fin *B*, el cual lleva en ese punto una velocidad $\boldsymbol{v}B$ de la trayectoria *C*:

$$\Delta Ek = \int_c \boldsymbol{F} \cdot d\boldsymbol{r} = m\int_c \boldsymbol{v} \cdot d\boldsymbol{v} = \frac{m}{2}\int_c dv^2 = \frac{m}{2}v_B^2 - \frac{m}{2}v_A^2 = T_B - T_A$$

[Ec. 2.13]

TA es la energía necesaria para llevar una partícula desde el reposo hasta la velocidad que lleva en el punto A, de la misma forma para TB y es conocida como energía cinética. Por otro lado, el resultado de esta integral:

$$\Delta Ev = \int_c F \cdot dr = \int_A^B -\nabla V \cdot dr = V(A) - V(B) = V_A - V_B$$

[Ec. 2.14]

Igualando ambas expresiones:

$$T_B - T_A = V_A - V_B \qquad \text{[Ec. 2.15]}$$

Por tanto, para cualquier par de puntos *A*, *B* tal que *A*, *B* ϵ $\mathbb{R}$; se tiene:

$$E = T_A + V_A = T_B + V_B \qquad \text{[Ec. 2.16]}$$

Donde *E* representa la energía total de la partícula, y es una constante para cualquier punto en el espacio. A veces es un tanto trabajoso expresar las derivadas; y las derivadas en el tiempo son bastante comunes. La nomenclatura que se encuentra en algunos libros de física para las derivadas de primer orden,

es un punto arriba de la variable derivada $\frac{dx}{dt} = \dot{x}$ y dos puntos para las derivadas de segundo orden $\frac{d^2x}{dt^2} = \ddot{x}$. No es muy común encontrar derivadas de orden superior pero se puede usar la misma notación.

Las leyes de Newton permiten tener otros resultados, que muchas veces son considerados como leyes. Vale la pena mencionar, cuerpos girando alrededor de un eje ya que son dinámicas comunes en algunos modelos, por ejemplo en un helicóptero o en un motor eléctrico. Una partícula girando alrededor de un eje presenta una velocidad angular. Se define como un vector perpendicular a $\boldsymbol{r}$ y $\boldsymbol{v}$, tal que

$$\boldsymbol{v} = \boldsymbol{\omega} \times \boldsymbol{r} \qquad \text{[Ec. 2.17]}$$

De la misma forma es posible enunciar una expresión para aceleración angular $\boldsymbol{a}$:

$$\boldsymbol{a} = \boldsymbol{a} \times \boldsymbol{r} \qquad \text{[Ec. 2.18]}$$

Se designa el símbolo $\boldsymbol{L}$ al momento angular de una partícula con respecto a un punto O, y es el análogo rotacional del momento lineal. Es un pseudovector que representa el producto de la inercia rotacional de un cuerpo y su velocidad de rotación en torno a eje determinado:

$$\boldsymbol{L} = \boldsymbol{r} \times \boldsymbol{p} \qquad \text{[Ec. 2.19]}$$

Cuando un cuerpo se encuentra girando alrededor de un eje, una torsión externa debe ser aplicada para cambiar su momento angular. La torsión es el equivalente rotacional de la fuerza. Encuentra la siguiente definición:

$$\boldsymbol{r} = \frac{d\boldsymbol{L}}{dt} = \frac{d(\boldsymbol{r}\times\boldsymbol{p})}{dt} = \frac{d\boldsymbol{r}}{dt}\times\boldsymbol{p} + \boldsymbol{r}\times\frac{d\boldsymbol{p}}{dt} \qquad \text{[Ec. 2.20]}$$

Pero el vector p es paralelo a $\frac{d\boldsymbol{r}}{dt}$ y por tanto su producto vectorial es 0, por tanto:

$$\boldsymbol{r} = \boldsymbol{r}\times\frac{d\boldsymbol{p}}{dt} = \boldsymbol{r}\times\boldsymbol{F} \qquad \text{[Ec. 2.21]}$$

Obteniendo así una expresión equivalente a la del momento angular, pero para fuerza. De la ecuación 2.19 y la igualdad para un triple producto escalar $a\,\boldsymbol{x}\,(b\times c) \;\; = b\,(a\cdot c) - c\,(a\cdot b)$**:**

$$\boldsymbol{L} = \boldsymbol{r}\times\boldsymbol{p} = \boldsymbol{r}\times(m\boldsymbol{v}) = m\boldsymbol{r}\times(\boldsymbol{\omega}\times\boldsymbol{r}) = m\boldsymbol{r}(\boldsymbol{r}) = m\boldsymbol{\omega}(\boldsymbol{r}\cdot\boldsymbol{r}) - m\boldsymbol{r}(\boldsymbol{\omega}\cdot\boldsymbol{r})$$

[Ec. 2.22]

Siendo $\boldsymbol{\omega}$ perpendicular a **r** y de la norma $r\cdot r = r^2$

$$L = mr^2\omega \qquad \text{[Ec. 2.23]}$$

La ecuación anterior introduce el concepto de masa angular o inercia rotacional. En una partícula la masa angular o inercia rotacional $I = mr^2$ es el término que acompaña a la velocidad angular $\boldsymbol{\omega}$. Si se desea encontrar el momento de un cuerpo *D* se puede considerar un sistema formado de partículas cada una con masa m_i y una distancia r_i de un eje:

$$L = \sum_{t\in D} m_\mathrm{i} r_t^2\omega$$

[Ec. 2.24]

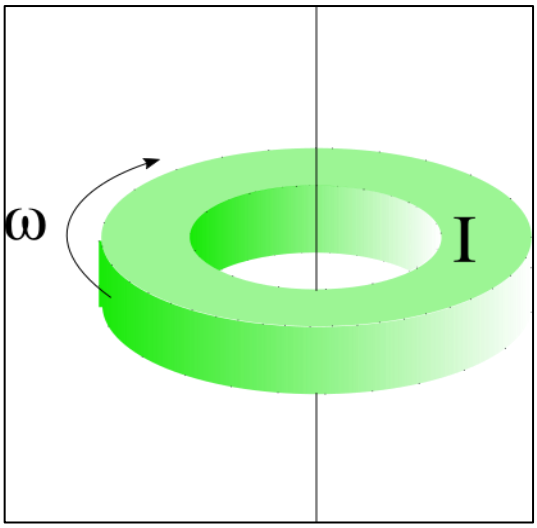
Si las partículas son lo suficientemente pequeñas:

$$L = \omega \int_D r^2 dm = I\omega \qquad \text{[Ec. 2.25]}$$

Donde I es el resultado de la integral. Con base en estos resultados se puede encontrar una expresión para la energía cinética correspondiente a la rotación de un cuerpo alrededor de un eje:

$$E = \frac{I}{2}\omega^2 \qquad \text{[Ec. 2.26]}$$

Figura 9. **Disco de inercia I girando a una velocidad ω**



Fuente: elaboración propia, empleando Inkscape.

2.1.2. Mecánica lagrangiana

La mecánica newtoniana conceptualmente es más simple que otras formulaciones de la mecánica clásica, por lo que aunque resulta útil en

problemas relativamente sencillos, su uso en problemas complicados puede ser más complicado que las otras formulaciones. Después del impacto que Newton generó en la mecánica con el *Principia Mathematica Philosophae Naturalis* a finales del siglo XVII, en el siglo XVIII la mecánica sufre una reinvención por Leonhard Euler y Joseph-Louis Lagrange, en términos del cálculo de variaciones. El cálculo de variaciones aborda problemas que pretenden maximizar o minimizar una cantidad, pero a diferencia de los problemas del cálculo elemental, no pretenden encontrar un número sino una función que lo haga. Véanse los siguientes ejemplos.

- Geodésica euclidiana. Considérese el camino que une dos puntos, (x_1, y_1) *y* (x_2, y_2), en un espacio euclidiano de dos dimensiones. Supongamos que la curva que une estos puntos está dada por $y = y(x)$. Entonces sea *J* (*y*) la longitud de la curva *y* del punto (x_1, y_1) al punto (x_2, y_2). Del cálculo se sabe que:

$$J(y) = \int_{(x_1,y_1)}^{(x_2,y_2)} ds$$

[Ec. 2.27]

$$= \int_{x_1}^{x_2} \sqrt{1 + (\dot{y}(x))^2}\, dx$$

[Ec. 2.28]

Donde $\dot{y}(x) = \frac{dy}{dx}$ el problema de la geodésica euclidiana consiste en encontrar la curva *y* que minimiza J (*y*). Se ha dicho que la distancia entre dos puntos es la línea recta, pero ¿Cuál es la base para decirlo? Intuitivamente se asume como cierto, pero, ¿cómo se puede estar seguro de ello?

- Braquistócrona. Suponga que a una partícula se le permite libremente deslizarse por una curva desde el reposo en un punto $P = (x_1,\ y_1)$ hasta el origen $O = (o, o)$. El problema consiste en encontrar la curva y $Y =\ y\ (x)$ que minimiza el tiempo de descenso. Sea J (y) el tiempo de descenso que depende de la forma de la curva. Entonces se debe minimizar la expresión

$$J(y) = \int_{(x_1,y_1)}^{(0,0)} \frac{1}{v}\, ds$$

[Ec. 2.29]

De la ecuación de la energía se sabe:

$$E = mgy(x_1) = \frac{1}{2}mv^2 + mgy(x)$$

[Ec. 2.30]

$$\Rightarrow v = \sqrt{2g(y_x}\,(x_1) - y(x))$$

[Ec. 2.31]

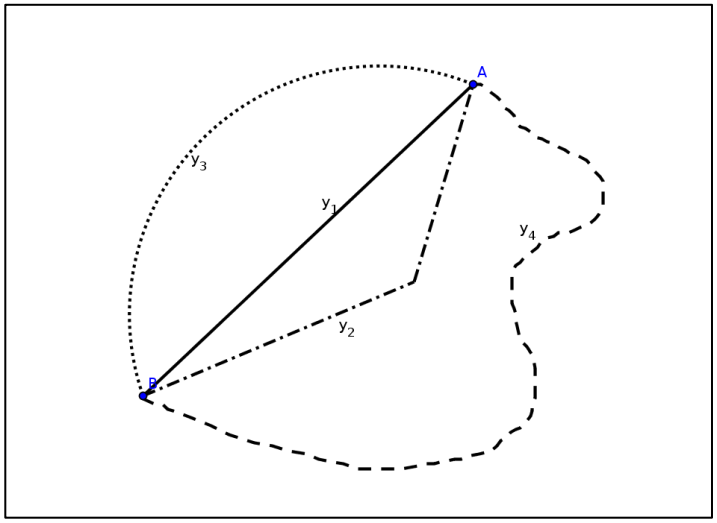
Por tanto, se tiene la siguiente expresión a minimizar:

$$J(y) = \int_{x^1}^{0} \frac{\sqrt{1 + \left(y_x(x)\right)^2}}{\sqrt{2g\left(y(x^1) - y(x)\right)}}\, dx$$

[Ec. 2.32]

Este problema lo planteó Johann Bernoullien en 1696.

Figura 10. **Diferentes curvas en un espacio euclidiano**



Fuente: elaboración propia, empleando Inkscape.

- Ecuación de Euler-Lagrange

Suponga que tiene una función *F* que es al menos dos veces derivable con respecto a todos sus argumentos. A través de cualquier curva $Y = y(x)$, la cual es al menos dos veces derivable en el intervalo [a, b], con $y(a)$ y $y(b)$ fijos. Un valor estacionario es un mínimo local, máximo local o punto silla de *J.* El problema consiste en encontrar la curva $Y = y(t)$ que genera un valor estacionario de *J,* también conocida solución estacionaria de *J,* si:

$$J(y) = \int_a^b F(t, y, \dot{y})dt$$

[Ec. 2.33]

Tabla XIII. **Teorema de ecuación de Euler-Lagrange**

La función $u = u(t)$ que genera un valor estacionario al funcional *J* satisface la ecuación de Euler- Lagrange en el intervalo $[a, b]$:

$$\frac{\partial F}{\partial u} - \frac{d}{dt}\left(\frac{\partial F}{\partial u}\right) = 0$$

[Ec. 2.34]

Fuente: MORIN, David. *Introduction to Classical Mechanics*. p. 88.

Esta ecuación genera un conjunto de ecuaciones diferenciales, cuyas soluciones devuelven la función u tal que *J* (u) es un máximo, mínimo o punto silla local de *J.* Por ejemplo, para la geodésica euclidiana, $F(\mathrm{x}, \mathrm{y}, \dot{\mathrm{y}}) = \sqrt{1 + (\dot{\mathrm{y}})^2}$. Si en lugar de *t* se toma a *x* como la variable independiente, simplemente para hacer ver que se encuentra en un plano cartesiano 6, entonces $F(\mathrm{x}, \mathrm{u}, \dot{\mathrm{u}}) =$ y *u* necesariamente deben cumplir:

$$\frac{\partial F}{\partial u} - \frac{d}{dx}\left(\frac{\partial F}{\partial u}\right) = 0$$

[Ec. 2.35]

$$\frac{\partial \sqrt{1 + (u)^2}}{\partial u} - \frac{d}{dx}\left(\frac{\partial \sqrt{1 + (u)^2}}{\partial u}\right) = 0$$

[Ec. 2.36]

$$0 - \frac{d}{dx}\left(\frac{u}{\sqrt{1 + (u)^2}}\right) = 0$$

[Ec. 2.37]

$$\frac{\ddot{\mathrm{u}}}{(1 + (u)^2)^{\frac{1}{2}}} - \frac{(\ddot{\mathrm{u}})^2 \ddot{\mathrm{u}}}{(1 + (u)^2)^{\frac{3}{2}}} = 0$$

[Ec. 2.38]

La solución a la ecuación diferencial anterior lleva a la curva con la menor longitud entre dos puntos:

$$\frac{(1+(u)^2)\ddot{u}}{(1+(u)^2)^{\frac{3}{2}}} - \frac{(\ddot{u})^2\ddot{u}}{(1+(u)^2)^{\frac{3}{2}}} = 0$$

[Ec. 2.39]

$$\frac{(\ddot{u})^2\ddot{u}}{(1+(u)^2)^{\frac{3}{2}}} = 0$$

[Ec. 2.40]

$$\ddot{u} = 0$$

[Ec. 2.41]

La solución da como resultado $U(x) \ = c_1 + \ c_2 x,$ la cual debe satisfacer $U(x_1) \ = y\ (x_1) \ = y_1$, y $U(x_2) \ = y\ (x_2) \ = y(x_2) \ = y_2$, por lo tanto:

$$u(x) = \frac{y_2 - y_1}{x_2 - x_1} x + y_1$$

[Ec. 2.42]

Tabla XIV. **Lema de forma alternativa**

La ecuación de Euler-Lagrange dada en el teorema 2.1 es equivalente a $$\frac{\partial F}{\partial t} - \frac{d}{dt}\left(F - u\frac{\partial F}{\partial \dot{u}}\right) = 0$$ [Ec. 2.43]

Fuente: MALHAM, Simon. J. *An introduction to Lagrangian and Hamiltonian mechanics*. p. 97.

De la ecuación de Euler-Lagrange se pueden obtener varias generalizaciones:

- Derivadas de orden superior. Suponga que se pide encontrar la curva *y* $y = \mathrm{y(t)}$ que devuelve un valor estacionario del funcional:

$$J(y) = \int_a^b F(t, y, \dot{y}, \ddot{y}) dt$$

[Ec. 2.44]

Sujeto a $y(a), y(b), \dot{y}(a) y\, \dot{y}(a)$ fijos. Acá la cantidad funcional a optimizar depende de la curvatura y del camino. La curva $u = \mathrm{u(t)}$ que devuelve un punto estacionario del funcional *J* necesariamente satisface la condición:

$$\frac{\partial F}{\partial u} - \frac{d}{dt}\left(\frac{\partial F}{\partial \dot{u}}\right) + \frac{d^2}{dt^2}\left(\frac{\partial F}{\partial \ddot{u}}\right) = 0 \qquad \text{[Ec. 2.45]}$$

- Múltiples variables dependientes de una variable independiente. Suponga que se pide encontrar una solución estacionaria de varias dimensiones $y = \mathrm{y(t)}\, \epsilon\, \mathbb{R}^{\mathrm{N}}$ que devuelve un valor estacionario en el funcional:

$$J(y) = \int_a^b F\,(t, y, \dot{y}) dt \qquad \text{[Ec. 2.46]}$$

Donde **y**(a) y **y**(b) están fijos. Nótese que *x* 2 [a,b], pero acá $y = \mathrm{y(t)}$ es una curva en un espacio de *N* dimensiones, pudiéndola representar:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ yN \end{pmatrix} \qquad \dot{y} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \vdots \\ \dot{y}N \end{pmatrix}$$

[Ec. 2.47]

La curva $u = \mathrm{u(t)}$ si es una solución estacionaria de *J,* necesariamente satisface la condición:

$$\frac{\partial F}{\partial u_{\mathrm{i}}} - \frac{d}{dt}\left(\frac{\partial F}{\partial \dot{u}_{\mathrm{i}}}\right) = 0 \qquad \forall i \in N, 1 \leq i \leq N$$

[Ec. 2.48]

- Múltiples variables independientes. Suponga que se pide encontrar el campo escalar $y = \mathrm{y(x)}$, tal que $x = \epsilon\, \Omega \subseteq \mathbb{R}^{\mathrm{N}}$, y devuelve un valor estacionario del funcional:

$$J(y) = \int_{\Omega} F(x, y, \nabla y) dV$$ [Ec. 2.49]

Donde $dV = \mathrm{dx^1 dx^2 \ldots dxn}$ y $\mathrm{y(x)}$ se encuentra fijo en la frontera $\partial\Omega$ del dominio -. Sea $y_{\mathrm{x}} = \nabla \mathrm{y} = \nabla_{\mathrm{x}} \mathrm{y}$ el gradiente de *y* con respecto al vector **x**, el vector de derivadas parciales con respecto a cada una de las componentes $x_{\mathrm{i}}\ (i = 1,2, \ldots N) de$ x:

$$\nabla_x y \begin{pmatrix} \frac{\partial y}{\partial x^1} \\ \frac{\partial y}{a x^2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{pmatrix}$$

[Ec. 2.50]

Por tanto, si $\frac{\partial u}{\partial x_{\mathrm{i}}} = u_{\mathrm{xi}}$, se tiene:

$$(\nabla_{ux} F) = \begin{pmatrix} \frac{\partial F}{\partial u_{x^2}} \\ \frac{\partial F}{\partial u_{x^2}} \\ \vdots \\ \frac{\partial F}{\partial u_{x^2}} \end{pmatrix}$$

[Ec. 2.51]

El campo $u = u(\mathrm{x})$ que es una solución estacionaria en *J*, necesariamente cumple:

$$\frac{\partial F}{\partial u} = \nabla \cdot (\nabla_{ux} F) = 0$$

[Ec. 2.52]

Donde $\nabla \cdot$ representa el operador de divergencia con respecto a $\boldsymbol{x}$.

El problema del gradiente medio. Ecuación de Laplace. Encontrar un campo $\Psi = \Psi(\mathrm{x}),\ x = (\mathrm{x}_1, x_2, x_3)^{\mathrm{T}}\, \epsilon\, \Omega \subseteq \mathbb{R}^3$, que genera un valor estacionario en el siguiente funcional:

$$J(\Psi) = \int_{\Omega} |\nabla \Psi|^2 dV.$$

[Ec. 2.53]

En este caso el integrando del funcional *J* es:

$$F(x, \Psi, \nabla \Psi) = |\nabla \Psi|^2 + (\Psi_{x_1})^2 + (\Psi_{x_2})^2 + (\Psi_{x_3})^2$$

[Ec. 2.54]

Nótese que F solamente depende de las derivadas parciales de Ψ, por tanto $\frac{aF}{\partial \Psi} = 0$. Usando la condición de Euler-Lagrange en F se tiene:

$$\frac{\partial F}{\partial \Psi} - \nabla \cdot (\nabla_{\Psi x} F) = 0 \qquad \text{[Ec. 2.55]}$$

$$0 = \begin{pmatrix} \partial/\partial_{x^1} \\ \partial/\partial_{x^2} \\ \partial/\partial_{x^3} \end{pmatrix} \cdot \begin{pmatrix} \partial F/\partial \Psi_{x^1} \\ \partial F/\partial \Psi_{x^2} \\ \partial F/\partial \Psi_{x^3} \end{pmatrix} = 0 \qquad \text{[Ec. 2.56]}$$

$$-\frac{\partial}{\partial_{x1}}(2\Psi_{x1}) - \frac{\partial}{\partial_{x2}}(2\Psi_{x2}) - \frac{\partial}{\partial_{x3}}(2\Psi_{x3}) = 0 \qquad \text{[Ec. 2.57]}$$

$$-2(\Psi_{x1x1} + \Psi_{x2x2} + \Psi_{x3x3}) = 0 \qquad \text{[Ec. 2.58]}$$

$$\Psi_{x1x1} + \Psi_{x2x2} + \Psi_{x3x3} = 0 \qquad \text{[Ec. 2.59]}$$

$$\frac{\partial^2\Psi}{\partial x_1^2} + \frac{\partial^2\Psi}{\partial x_2^2} + \frac{\partial^2\Psi}{\partial x_3^2} = 0 \qquad \text{[Ec. 2.60]}$$

$$\nabla^2\Psi = 0 \qquad \text{[Ec. 2.61]}$$

Esta es la ecuación de Laplace para Ψ en el dominio Ω; las soluciones son llamadas funciones armónicas. Este es un famoso resultado utilizado frecuentemente en el análisis de oscilaciones mecánicas y electromagnéticas.

Restricciones en los caminos. Un problema común en optimización es encontrar valores estacionarios de *J* con respecto a caminos que se encuentran restringidos en alguna forma. Suponga que se nos pide encontrar una solución estacionaria del funcional:

$$J(y) = \int_a^b F(t, y, \dot{y})dt$$

[Ec. 2.62]

Donde $y = (y_1,\ y_2, \ldots, y^{\mathrm{N}})^{\mathrm{T}} \epsilon\ \mathbb{R}^{\mathrm{N}}$ sujeto a *m* restricciones:

$$G_k(t, y) = 0$$

[Ec. 2.63]

Donde $k \epsilon [1,2,\dots,m]$. Un caso similar se encuentra en el cálculo de varias variables, y se ve resuelto por los multiplicadores de Lagrange. Inspirados en los multiplicadores de Lagrange, se plantea un procedimiento similar para el cálculo de variaciones, con el fin de resolver dicho problema. Observe que la ecuación 2.63 implica:

$$\int_a^b \lambda_k\,(t) G_k(t,y)dt = 0 \qquad \forall k \in \{1,2,\dots,N\}$$

[Ec. 2.64]

Se hace una analogía a los multiplicadores de Lagrange en el cálculo de varias variables. La idea consiste en transformar el problema con restricciones en un problema equivalente sin restricciones. Se plantea el funcional *J,* el cual conserva el mismo valor que $J\ (\boldsymbol{y})$ para todas las curvas restringidas por todas las condiciones G_K*:*

$$\tilde{J}(y,\lambda) = J(y) + \widetilde{\sum_{k=1}^{m}} \int_a^b \lambda_k(t) G_k(t,y)dt$$

[Ec. 2.65]

Este nuevo funcional *J,* transforma el problema de optimizar muchas variables $yi\ (t\ )$, con $i \epsilon\ [1,2\dots N]$, dependientes en el tiempo, sujetas a un conjunto de restricciones, a un problema de optimizar muchas variables $yi\ (t\ ), con\ k \epsilon \{1,2\dots N\}, y\ \lambda_K\ (t\ ), con\ i \epsilon \{1,2\dots m\}$, pero sin restricciones:

$$\tilde{J}\ (y,\boldsymbol{\lambda}) = \int_{\boldsymbol{a}}^{\boldsymbol{b}} \boldsymbol{F(t,y,\dot{y})dt} + \sum_{\boldsymbol{k-1}}^{\boldsymbol{m}} \int_{\boldsymbol{a}}^{\boldsymbol{b}} \boldsymbol{\lambda_k(t) G_k(t,y)dt}$$

[Ec. 2.66]

$$\tilde{J}\,(y,\dot{y}) = \int_a^b (F(t,y,\dot{y}) + \sum_{k-1}^{m} \lambda_k(t) G_k\,(t,y))\,dt$$

[Ec. 2.67]

El integrando *F* del funcional *J* sería el siguiente:

$$\tilde{F}\,(t,y,\boldsymbol{\lambda}) = \boldsymbol{F}(\boldsymbol{t},\boldsymbol{y},\dot{y}) + \sum_{\boldsymbol{k-1}}^{\boldsymbol{m}} \boldsymbol{\lambda_k}\,(\boldsymbol{t})\boldsymbol{G_k}(\boldsymbol{t},\boldsymbol{y})$$

[Ec. 2.68]

Por tanto, el vector $\{y_1, y_2, \ldots, y^{\mathrm{N}}, \Lambda_1, \Lambda_2, \ldots, \Lambda_m\}^{\mathrm{T}}$ que devuelven un valor estacionario de *J* deben satisfacer:

$$\frac{\partial \tilde{F}}{\partial y_t} - \frac{d}{dt}\left(\frac{\partial \tilde{F}}{\partial \dot{y}_t}\right) = 0 \qquad \forall i \in \{1,2,\ldots,N\}$$

[Ec. 2.69]

$$\frac{\partial \tilde{F}}{\partial \lambda_t} - \frac{d}{dt}\left(\frac{\partial \tilde{F}}{\partial \dot{\lambda}_t}\right) = 0 \qquad \forall k \in \{1,2,\ldots,m\}$$

[Ec. 2.70]

De la ecuación anterior se puede ver $\frac{\Lambda\dot{\mathrm{F}}}{\partial \Lambda} = G\mathrm{K}\,(t,y) = 0\; y\; \frac{\Lambda\dot{\mathrm{F}}}{\partial \Lambda}$, implicando $G\mathrm{K}\,\{t\,,\boldsymbol{y}\} = 0$, lo que indica que todo es consistente, de lo cual se simplifican las ecuaciones a:

$$\frac{\partial \tilde{F}}{\partial y_t} - \frac{d}{dt}\left(\frac{\partial \tilde{F}}{\partial \dot{y}_t}\right) = 0 \qquad \forall i \in \{1,2,\ldots,N\} \qquad \text{[Ec. 2.71]}$$

$$G_k(t,y) = 0 \qquad \forall k \in \{1,2,\ldots,m\} \qquad \text{[Ec. 2.72]}$$

Se asume que se puede resolver este sistema de ecuaciones, encontrando así el vector $\{\dot{y}, \Lambda\}^{T}$ de funciones que devuelven un valor estacionario de J, entonces ẏ devuelven un valor estacionario para *J*, recuérdese el hecho que $\hat{J}\{t, y, \Lambda\} = J(t, y) + 0$, para todas las funciones que satisfacen todas las restricciones. Es frecuente encontrar problemas con restricciones espaciales. Es viable resolver por el método de los multiplicadores de Lagrange.

Tabla XV. **Definición de restricciones holonómicas**

Para un sistema con posiciones dadas por $\boldsymbol{r}_i(t)$, $\forall i \in \{1,2,\ldots,N\}$, las restricciones que pueden ser expresadas en la forma $$\boldsymbol{g}(r_1, r_2, \ldots, r_N, t) = 0$$ [Ec. 2.73] Se dice que son holonómicas.

Fuente: MALHAM, Simon. J. *An introduction to Lagrangian and Hamiltonian mechanics*. p.15.

Tabla XVI. **Definición de grados de libertad**

Se denominan grados de libertad al conjunto de variables independientes entre sí, pero dependientes del tiempo, de las cuales depende el integrando *L* de un funcional *J*, de la forma: $$J(q) = \int_{t_1}^{t_2} L(r_1, \dot{r}_1, \ldots, r_1^k, r_2, \dot{r}_2, \ldots, r_2^k, \ldots\ldots, r_M^K, t)dt$$ [Ec. 2.74] En un problema sin restricciones, el número de grados de libertad *N*, es la suma de los tamaños de los *M* vectores $\mathbf{r}_i$ diferentes que depende el integrando. Si el problema presenta *m* restricciones holonómicas, es posible demostrar que el número de grados de libertad es *N - m*

Fuente: MALHAM, Simon. J. *An introduction to Lagrangian and Hamiltonian mechanics.*p. 15.

- Lagrangiano y mecánica clásica: a principios del siglo XIX las ideas de Euler y Lagrange fueron refinadas por Carl Gustav Jacob Jacobi y William Rowan Hamilton. Según esta nueva formulación a la mecánica, al contrario que lo propone Newton, las partículas no siguen trayectorias ya que se ven afectadas por una fuerza externa. En vez de esto, entre todas las posibles trayectorias entre dos puntos, ellas escogen aquella que minimiza una magnitud llamada acción. El objetivo es buscar una función *L*, tal que los caminos que toman las partículas en un período de tiempo comprendido entre $t_1 \; y \; t_2$ son curvas estacionarias para la integral:

$$S(q) = \int_{t_1}^{t_2} L(q, \dot{q}, t) dt \qquad \text{[Ec. 2.75]}$$

Donde $q = \{q_1, q_2, \ldots, qn\}^{\mathrm{T}}$ es un vector de variables dependientes en el tiempo, y $q = \{q_1, q_2, \ldots, qn\}^{\mathrm{T}}$ su razón de cambio en el tiempo. El resultado *S*, de la integral, recibe el nombre de acción, y el integrando su Lagrangiano. Nótese que estando $t_1 \; y \; t_2$ fijos, lo único que determina el resultado de *S* es la curva, $q = q(t)$, que toma la partícula. Es importante hacer ver la solución del problema no será única ya que existen muchos Lagrangianos diferentes que tienen las mismas curvas estacionarias:

Tabla XVII. **Teorema de no unicidad del lagrangiano**

Sea la acción $S\{\boldsymbol{q}\} = \int \frac{t^2}{t^1} L\{\mathrm{q}, \mathrm{q}, \mathrm{t}\} dt$. Y sea $\boldsymbol{u}: t \rightarrow \mathbb{R}^{\mathrm{N}}$ un vector de variables dependientes en el tiempo. Existe más de un Lagrangiano *L* para el valor *S* (**u**).

Fuente: MALHAM, Simon. J. *An introduction to Lagrangian and Hamiltonian mechanics.* p. 16.

Estando conscientes de la existencia de varios lagrangianos, se procede a la búsqueda de uno que describa el movimiento de las partículas en un

potencial de fuerza. Se necesita encontrar uno que cumpla con las siguientes condiciones o principios:

- La trayectoria de una partícula depende únicamente de su posición y velocidad inicial. Sin pérdida de la generalidad, y sin violar el teorema 2.2, el lagrangiano *L* puede depender de derivadas de un orden *n*. Se sabe empíricamente, debido al éxito de la teoría de Newton, que se puede predecir la trayectoria que seguirá una partícula únicamente sabiendo su posición y velocidad inicial. Dado que se busca una solución estacionaria, las ecuaciones de Euler-Lagrange deben ser de segundo orden y el Lagrangiano de primer orden para ello. Dicho de otra forma, esperar que las ecuaciones de Euler-Lagrange sean de segundo orden implica que el Lagrangiano solo puede depender de la posición y la velocidad, y no así de la aceleración o derivadas de orden superior de la posición con respecto al tiempo.

- La ecuación de movimiento de una partícula libre, es invariante a traslaciones y rotaciones. Si se considera la ecuación de movimiento de una partícula, en la ausencia de un campo de fuerza en un espacio unif*or*me e isotrópico, dicha ecuación no puede depender del vector posición de la partícula con respecto a ningún sistema de coordenadas, porque que el espacio es idéntico en todos los puntos. De hecho, no puede depender de la dirección de la velocidad ya que el espacio es idéntico en todas direcciones. Como resultado el Lagrangiano de una partícula libre es una función de la norma de su velocidad v^2.

- La ecuación de movimiento de una partícula libre es invariante a las transformaciones de Galileo. Considérese el movimiento de una partícula libre, en dos marcos de referencia inerciales relacionados por una

transformación de Galileo: el primer marco de referencia *A,* se encuentra en reposo con respecto a un operador y el segundo, *A*', se encuentra moviéndose con una velocidad uni*for*me muy pequeña **∂u** con respecto al primer marco de referencia. Los vectores de velocidad en ambos marcos se encuentran relacionados por la transformación de Galileo:

$$v' = v + \partial u \qquad \text{[Ec. 2.76]}$$

De forma similar, los lagrangianos en los dos marcos estarán relacionados por:

$$L'((v')^2) = L(|v + \partial u|^2) \qquad \text{[Ec. 2.77]}$$

$$= L((v + \delta u) \cdot (v + \delta u)) \qquad \text{[Ec. 2.78]}$$

$$= L(v^2 + 2v \cdot \delta u + \delta u^2) \qquad \text{[Ec. 2.79]}$$

$$\cong L(v^2) + 2\left(\frac{dL}{dv^2}\right) v \cdot \delta u \qquad \text{[Ec. 2.80]}$$

En la ecuación 2.80 se usó una aproximación de Taylor para la expansión de $L\{v^2\}$ considerando ±**u** un elemento infinitesimal. Si $\boldsymbol{x}$ es un vector compuesto de *n* variables dependientes en el tiempo x_{i}, cada una representando una coordenada cartesiana para la posición de la partícula, y $\mathrm{v} = \dot{\mathrm{x}} = \frac{dx}{dt} + \frac{x^2}{2!}$ un vector representando a la velocidad de la partícula compuesto por la derivada de las coordenadas $\dot{x}t$. Entonces la norma de la velocidad es una sumatoria de los cuadrados de las derivadas de cada coordenada, todas en relación al tiempo, $v^2 = \dot{\mathrm{x}} \cdot \dot{\mathrm{x}} = \sum_t^n = 0\, \dot{\mathrm{x}}t^2$. Si se considera que una partícula entre todas las trayectorias, escoge una curva

estacionaria para la acción, *L* debe cumplir las ecuaciones de Euler-Lagrange. Si se quiere que esta mecánica sea invariante a las transformaciones de Galileo, entonces *L*' también debe de cumplir las ecuaciones de Euler-Lagrange, dado que de esta forma no se altera el principio de las curvas estacionarias. Por tanto, la expresión 2.80 debe cumplir Euler-Lagrange, sin importar rotaciones ni traslaciones.

$$\frac{\partial L}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) = 0 \Rightarrow \frac{\partial L'}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L'}{\partial \dot{x}_i}\right) = 0 \quad \forall i \in \{1, 2, \dots, n\}$$

[Ec. 2.81]

Por tanto $\forall i \, \epsilon \, \{1,2,\dots,n\}$ se tiene:

$$\frac{\partial L'}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L'}{\partial \dot{x}_i}\right) = \frac{\partial\left(L(v^2) + \left(\frac{dL}{dv^2}\right)\boldsymbol{v}\cdot\delta\boldsymbol{u}\right)}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial\left(L(v^2) + \left(\frac{dL}{dv^2}\right)\boldsymbol{v}\cdot\delta\boldsymbol{u}\right)}{\partial x_i}\right)$$

[Ec. 2.82]

$$= \frac{\partial L}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) + \frac{\partial\left(2\left(\frac{dL}{dv^2}\right)\boldsymbol{v}\cdot\delta\boldsymbol{u}\right)}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial\left(2\left(\frac{dL}{dv^2}\right)\boldsymbol{v}\cdot\delta\boldsymbol{u}\right)}{\partial x_i}\right)$$

[Ec. 2.83]

$$= \frac{\partial\left(2\left(\frac{dL}{dv^2}\right)\boldsymbol{v}\cdot\delta\boldsymbol{u}\right)}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial\left(2\left(\frac{dL}{dv^2}\right)\boldsymbol{v}\cdot\delta\boldsymbol{u}\right)}{\partial x_i}\right)$$

[Ec. 2.84]

$$= \frac{\partial\left(2\left(\frac{dL}{dv^2}\right)\sum_{i=1} n\,\delta u_i \dot{x}_i\right)}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial\left(2\left(\frac{dL}{dv^2}\right)\sum_{i=1} n\,\delta u_i \dot{x}_i\right)}{\partial \dot{x}_i}\right)$$

[Ec. 2.85]

$$= \frac{\partial \left(2 \left(\frac{dL}{dv^2}\right) \sum_{i=1} n \, \delta u_i \dot{x}_i\right)}{\partial x_i}$$

[Ec. 2.86]

Y dado que para una transformación de Galileo **∂u** es constante:

$$\frac{\partial L'}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L'}{\partial \dot{x}_i}\right) = 2\delta u_i \frac{\partial \left(\frac{dL}{dv^2}\dot{x}_i\right)}{\partial x_i} - 2\delta u_i \frac{d}{dt}\left(\frac{\partial \left(\frac{dt}{dv^2}\dot{x}_i\right)}{\partial \dot{x}_i}\right)$$

[Ec. 2.87]

$$= -2\delta u_i \frac{d}{dt}\left(\frac{\partial \left(\frac{dL}{dv^2}\dot{x}_i\right)}{\partial \dot{x}_i}\right)$$

[Ec. 2.88]

$$= -2\delta u_i \frac{d}{dt}\left(\frac{\partial \left(\frac{dL}{dv^2}\dot{x}_i\right)}{\partial \dot{x}_i}\dot{x}_i + \left(\frac{dL}{dv^2}\right)\frac{\partial \dot{x}_i}{\partial \dot{x}_i}\right)$$

[Ec. 2.89]

Como el Lagrangiano para una partícula libre de fuerzas en un espacio unif*or*me es una función únicamente de la magnitud de la velocidad v^2 de la partícula, su derivada en función de la norma de la velocidad de la partícula, será una función de la misma:

$$dL \, \frac{dL}{dv^2} = f(v^2) = f(\sum_{t=1}^{n} (\dot{x}t^2)) \qquad \text{[Ec. 2.90]}$$

$$\frac{\partial L'}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L'}{\partial \dot{x}_i}\right) = 2\delta u_i \frac{d}{dt}\left(\frac{\frac{dL}{dv^2}\dot{x}_i}{\partial x_i}\right) - 2\delta u_i \frac{d}{dt}\left(\frac{\frac{dL}{dv^2}\dot{x}_i}{\partial x_i}\right)$$

[Ec. 2.91]

$$0 = -2\delta u_i \frac{d}{dt}\left(2\ \frac{df(v^2)}{dv^2}\dot{x}_t 2 + f(v^2)\right)$$

[Ec. 2.92]

Para que esta última ecuación sea consistente y *L*' sea invariante a las transformaciones galileanas, entonces $\forall i\ \epsilon\ \{1,2,\ldots,n\}$:

$$\frac{d}{dt}\left(2\frac{df(v^2)}{dv^2}\dot{x}_i{}^2 + f(v^2)\right) = 0 \Longrightarrow$$

[Ec. 2.93]

$$2\frac{df(v^2)}{dv^2}\dot{x}_i{}^2 + f(v^2) = constante$$

[Ec. 2.94]

$$2\frac{df(v^2)}{dv^2}\dot{x}_i{}^2 + f(v^2) = c$$

[Ec. 2.95]

Resolver la ecuación diferencial anterior lleva a la forma que tendrá el lagrangiano. El problema es que la ecuación diferencial depende de una función diferente, para cada $i\ \epsilon\ \{1,2,\ldots,n\}$, la ventaja es que se encuentran sujetas a la condición $\sum_{t\,=\,1}^{n} \dot{x}^2 = v^2$. Por tanto, es posible sumar todas las expresiones con el fin de llegar a un resultado:

$$2\frac{df(v^2)}{dv^2}\dot{x}_i{}^2 + nf(v^2) = C_1 \qquad \text{[Ec. 2.96]}$$

Esta ecuación diferencial tiene como solución:

$$\frac{dL}{dv^2} = f(v^2) = C_1 + C_2(v^2)^{-n/2} \qquad \text{[Ec. 2.97]}$$

$$\Longrightarrow L = C_1 v^2 + \frac{C_2}{1+\frac{n}{2}} + C_3 \qquad \text{[Ec. 2.98]}$$

Esta sería la forma más general del lagrangiano, pero el término $\{v^2\}1 - n/2$ depende de la dimensión del espacio, entonces nuestro lagrangiano sería distinto para un espacio de dos dimensiones que para un espacio de tres dimensiones, lo cual no es consistente, ya que una curva puede tener en un espacio de tres dimensiones, un equivalente en un espacio de dos dimensiones, pero sus lagrangianos serían diferentes y eso contradiría el hecho que se buscaron invariantes a las transformaciones de Galileo, por tanto $C_2 = 0$. Entonces la forma más general para un lagrangiano invariante a transformaciones galileanas, para una partícula en un espacio unif*or*me e isotrópico sin fuerza externa que la afecte, sería:

$$L = av^2 + \beta \qquad \text{[Ec. 2.99]}$$

Donde $a\ y\ \beta$ son constantes, que aún no se han definido. Para una partícula que no se encuentra afecta por una fuerza externa, basándose en el hecho que se busca una estacionaria para el lagrangiano, se tiene la ecuación de movimiento:

$$\frac{\partial L}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) = 0$$

[Ec. 2.100]

$$\frac{\partial (a\dot{x}\cdot\dot{x} + \beta)}{\partial x_i} + \frac{d}{dt}\left(\frac{\partial (a\dot{x}\cdot\dot{x} + \beta)}{\partial x_i}\right) = 0$$

[Ec. 2.101]

$$\frac{\partial \left(a\sum \left(\dot{x_i}^2\right) + \beta\right)}{\partial x_i} + \frac{d}{dt}\left(\frac{\partial \left(a\sum \left(\dot{x_i}^2\right) + \beta\right)}{\partial \dot{x}_i}\right) = 0$$

[Ec. 2.102]

$$0 + \frac{d}{dt}(2a\dot{x}_i) = 0$$

[Ec. 2.103]

$$\frac{d^2x_i}{dt^2} = 0$$

[Ec. 2.104]

De la ecuación anterior se puede concluir lo siguiente:

$$\frac{d^2x_i}{dt^2} = 0 \Longrightarrow \frac{dx_i}{dt} = constante$$

[Ec. 2.105]

Sin importar los parámetros $a\ y\ \beta$, pero esto no es nada más que la primera ley de Newton, que enuncia que una partícula en un marco de referencia inercial y sin fuerzas que la afectan, su velocidad permanece constante. En en el enfoque del cálculo de variaciones, la primera ley de Newton es una consecuencia de asumir que las leyes de esta mecánica son invariantes a las transformaciones de Galileo:

- El potencial de una fuerza conservativa es descrito por una función de las coordenadas. La presencia de una fuerza, rompe la invariancia galileana de un espacio porque la partícula experimenta en general, fuerzas de diferente magnitud y orientación en diferentes lugares del espacio. Los potenciales de una fuerza conservativa son escalares que dependen únicamente de su posición, no así de la velocidad. En este caso, el Lagrangiano de una partícula en un campo de potencial, que es un escalar, debe depender de las coordenadas de la partícula en las formas que puede describir un escalar. Por el momento se denotará como

$P\ (\boldsymbol{x}, t\ )$, por ejemplo, en un espacio de 3 dimensiones, el lagrangiano podría ser:

$$L = a\sum_{i=1}^{3} \dot{x}_i^2 + \beta + P(x,t)$$

[Ec. 2.106]

Esta es la forma más general que es consistente con las simetrías y condiciones que se impusieron al principio. Si se aplica la ecuación de Euler-Lagrange a este Lagrangiano, se obtiene:

$$\frac{\partial L}{\partial x_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) = 0$$

[Ec. 2.107]

$$\Longrightarrow \frac{\partial P}{\partial x_i} 2a \frac{d^2 x_i}{dt^2} = 0$$

[Ec. 2.108]

Se toma $\mathrm{a} = 1 + \frac{m}{2}$, para hacer consistente el lagrangiano con la segunda ley de Newton. Se identifica a *P* como -*V,* que es la energía potencial de la partícula en el campo. La constante *B* es irrelevante y por simplicidad se toma como 0. Por tanto la ecuación de movimiento de una partícula bajo la acción de una fuerza conservativa en un espacio determinado puede ser derivada de las ecuaciones de Euler-Lagrange, tomando el lagrangiano como:

$$L = \frac{1}{2}mv^2 - v(x,t)$$

[Ec. 2.109]

Tabla XVIII. **Definición de acción y lagrangiano**

Generalizando para *N* partículas interactuando entre sí a través de fuerzas conservativas y demostrando que la energía cinética para cada partícula se deriva de la ecuación de Euler-Lagrange. El Lagrangiano correspondiente al sistema sería:

$$L = T - V = \sum_{j=1}^{N} \frac{1}{2} m_j u_j^2 - v(x_j, t)$$

[Ec. 2.110]

Donde u_j es la velocidad de cada partícula, x_j su vector de coordenadas y m_j su masa. $V\ (x_j, t)$ representa la energía potencial de cada partícula. Siendo V el total de la energía potencial de interacción de las partículas:

$$V = \sum_{j=1}^{N} v(x_j, t)$$

[Ec. 2.111]

y T la energía cinética del sistema, tomando el valor

$$T = \sum_{j=1}^{N} \frac{1}{2} m_j u_j^2$$

[Ec. 2.112]

En otras palabras, las trayectorias de N partículas interactuando a través de fuerzas conservativas serán de tal forma tal que la acción

$$S = \int_{t_1}^{t_2} T - V dt$$

[Ec. 2.113]

encuentra un extremo (valor mínimo, máximo o punto silla local) en dichas trayectorias, donde $T\ y\ V$ son la energía cinética y potencial del sistema.

Fuente: elaboración propia, basado en el trabajo de David Morin. *Introduction to Classical Mechanics*. p. 66.

La ventaja de analizar o modelar un sistema mecánico, usando la formulación lagrangiana, es que se tiene resumida toda la dinámica de *N* partículas en una sola magnitud escalar L. De igual forma, es necesario resolver un conjunto de ecuaciones diferenciales para encontrar la posición de una

partícula. La segunda parte de la definición 15 es conocida como el principio de Hamilton de la mínima acción, en honor a William Rowan Hamilton, quien fue el primero en expresarlo de esta forma. En la tabla XIX enuncia la versión desarrollada por él. Es llamado de la mínima acción porque muchas veces la curva no solo representa un valor estacionario sino un mínimo del funcional.

Tabla XIX. **Teorema de principio de Hamilton de la mínima acción**

Sea T la energía cinética de un sistema y sea V la energía potencial de un sistema. Llamamos lagrangiano L a la diferencia $L = T - V$. Se define como acción S en un tramo de tiempo que va desde t_1 a t_2, donde $\boldsymbol{q} = (q_1, \ldots, q_2)^{\mathrm{T}}$, al funcional:

$$S = \int_{t_1}^{t_2} L(\mathrm{q}, \mathrm{q}, t)dt$$

[Ec. 2.114]

Entonces el camino correcto del movimiento de un sistema mecánico con restricciones holonómicas, y fuerzas externas conservativas, de un instante t_1 a un instante t_2 es la solución estacionaria de la acción. Por tanto, el camino correcto del movimiento $\boldsymbol{q} = \boldsymbol{q}(t), \text{con } \boldsymbol{q} = (q_1, \ldots, qn)^{\mathrm{T}}$, necesariamente satisface, para todo $j \in \{1,2, \ldots, n\}$, la ecuación de movimiento de Lagrange:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}j}\right) - \frac{\partial L}{\partial qj} = 0$$

[Ec. 2.115]

Fuente: MALHAM, Simon. J. *An introduction to lagrangian and hamiltonian mechanics.* p. 80.

- Coordenadas generalizadas. Hasta ahora, la formulación de un Lagrangiano se basó en considerar la mecánica invariante a transformaciones galileanas y explícitamente se expresaron las energías potenciales y cinéticas con coordenadas cartesianas. Sin embargo, en la formulación hecha por Hamilton, el sistema de coordenadas no entra en el enunciado del principio o en la definición del lagrangiano o la de la acción, es más, el enunciado se encuentra basado en un conjunto de coordenadas cualquiera. La curva estacionaria debe ser la misma sin importar el sistema que se este utilizando para describirla.

La principal diferencia entre un sistema Cartesiano y uno no Cartesiano es que la expresión para la energía cinética en general será diferente a $\frac{1}{2}mv^2$. Teniendo esto en consideración, es posible utilizar sistemas de coordenadas generalizadas sin ningún problema, para plantear las energías cinéticas y potenciales de un sistema con el fin de describir su movimiento. El mismo Lagrange en su libro *Méchanique Analytique* advierte:

"No encontrará figuras en esta obra. Los métodos que yo expongo no requieren ni construcciones, ni razonamientos geométricos o mecánicos, sino solamente operaciones algebraicas, sometidos a una marcha regular y unifo*r*me. Aquellos que gusten del Análisis, verán con deleite la Mecánica convertirse en una nueva rama y me agradecerán haber ampliado así este ámbito"[4].

- Fuerzas no conservativas. El principio de Hamilton encuentra sus bases en el concepto de fuerzas conservativas. Se cree que todas las fuerzas microscópicas en el universo son conservativas en naturaleza. Las fuerzas no conservativas nacen solo cuando se intenta modelar fenómenos microscópicos complejos empíricamente; modelar la fricción entre dos superficies es un ejemplo de ello. Se toma en cuenta la traslación de las superficies, sin considerar la dinámica de las partículas que a nivel microscópico interaccionan. Dicha interacción se ve reflejada en energía térmica que no se tomó en cuenta en el modelo. Es muy complicado considerar estos fenómenos en sus modelos, por lo que se debe encontrar una forma de insertarlos al método de Lagrange. Asumiendo que podemos identificar un conjunto de N coordenadas

[4] LANGRANGE, Louis Joseph. *Mécanique analytique, l'institut des Sciences, Lettres et Arts, du Bureau des Longitudes*. p. 215.

generalizadas $qj\,, j\,\epsilon\,\{1,2,\ldots,N\}$ que describen el estado del sistema únicamente. Se construye $x_{\mathrm{i}} = X(\,q_{\mathrm{i}}\,,t\,)$, donde X es una transformación de la coordenada general q_{i} y el tiempo t , a coordenadas cartesianas. En la ausencia de fuerzas conservativas las ecuaciones de movimiento tienen la forma:

$$\frac{\partial L}{\partial qj} - \frac{d}{dt}\left(\frac{\partial L}{\partial qj}\right) = 0 \qquad \forall j \in \{1,2,\ldots,N\}$$

[Ec. 2.116]

Ahora considérese el efecto de la interacción no conservativa entre partículas, descrita empíricamente por una fuerza $\boldsymbol{F} = (F_1, F_2, \ldots, F^{\mathrm{N}})^{\mathrm{T}}$ con componentes cartesianas. Es posible demostrar que la mecánica Lagrangiana y la mecánica de Newton serán equivalentes si se agrega un término Qj, representando una fuerza generalizada:

$$\frac{\partial L}{\partial qj} - \frac{d}{dt}\left(\frac{\partial L}{\partial qj}\right) = -Q_j \qquad \forall j \in \{1,2,\ldots,N\}$$

[Ec. 2.117]

Donde la fuerza generalizada Qj:

$$Q_j = \sum_{i=1}^{N} F_i \frac{\partial x_i}{\partial q_i}$$

[Ec. 2.118]

Fuerzas disipativas. Las fuerzas disipativas son un caso especial de las fuerzas no conservativas. Una fuerza disipativa, **R** contrarresta el movimiento de una partícula o cuerpo, por lo que su dirección es opuesta al vector dirección

de la velocidad. Ejemplos de este tipo de fuerzas son el amortiguamiento de Coulomb, o el amortiguamiento viscoso. Una fuerza de este tipo depende únicamente de su velocidad, de su dirección, de su magnitud o de ambas. Considérese un sistema dinámico tal que cada partícula del sistema se ve afectada por una fuerza disipativa R:

$$R_{\mathrm{i}} = -h_{\mathrm{i}}(v_{\mathrm{i}})\frac{v\mathrm{I}}{v_{\mathrm{i}}} \quad \forall i\epsilon\,\{1,2,\ldots,N\}$$

[Ec. 2.119]

Donde *vi* es la norma de la velocidad $\boldsymbol{v}_{\mathrm{i}}$. Si se desea encontrar una fuerza generalizada $\boldsymbol{Q} = (Q_1, Q_2, \ldots, N)^{\mathrm{T}}$, para poder insertarla en las ecuaciones de movimiento. Al transformar a coordenadas generalizadas $q = (q_1, q_2, \ldots, qn)$, se obtiene:

$$Q_j = -\sum_{i=1}^{N} h_j(v_j)\frac{\boldsymbol{v_i}}{v_i}\cdot\frac{\partial \boldsymbol{r}_i}{\partial qj}$$

[Ec. 2.120]

Si la velocidad es la razón de cambio en el tiempo de la posición:

$$v = \frac{dx}{dt} \Longrightarrow$$

[Ec. 2.121]

$$\frac{\partial \boldsymbol{v}}{\partial \dot{q}j} = \frac{\partial}{\partial \dot{q}j}\left(\frac{dx}{dt}\right)$$

[Ec. 2.122]

$$\frac{\partial \boldsymbol{v}}{\partial \dot{q}j} = \frac{\partial}{\partial \dot{q}j}\left(\sum_{i=1}^{n}\frac{\partial \boldsymbol{r}}{\partial \dot{q}j}\dot{q}j\right)$$

[Ec. 2.123]

$$\frac{\partial \boldsymbol{v}}{\partial \dot{q} j} = \frac{\partial \boldsymbol{r}}{\partial q j}$$

[Ec. 2.124]

Por otro lado:

$$2v \cdot \frac{\partial v}{\partial \dot{q}_j} = \frac{\partial (v \cdot v)}{\partial \dot{q}_j} = \frac{\partial v}{\partial \dot{q}_j} \cdot v + v \cdot \frac{\partial v}{\partial \dot{q}_j} = \frac{\partial v^2}{\partial \dot{q}_j} = 2v \frac{\partial v}{\partial \dot{q}_j}$$

[Ec. 12.125]

$$\Longrightarrow \boldsymbol{v} \cdot \frac{\partial \boldsymbol{v}}{\partial \dot{q}_j} = v \frac{\partial v}{\partial \dot{q}_j}$$

[Ec. 2.126]

Entonces $\forall j \, \epsilon \, \{1,2,\ldots,N\}$

$$Q_j = -\sum_{i=1}^{N} h_i \, (v_i) \frac{\boldsymbol{v_i}}{v_i} \cdot \frac{\partial \boldsymbol{v_i}}{\partial \dot{q}_j}$$

[Ec. 2.127]

$$Q_j = -\sum_{i=1}^{N} h_i \, (v_i) \frac{\partial \boldsymbol{v_i}}{\partial \dot{q}_j}$$

[Ec. 2.128]

$$Q_j = -\sum_{i=1}^{N} \boldsymbol{h_i} \, (v_i) \frac{\partial \boldsymbol{v_i}}{\partial \dot{q}_j}$$

[Ec. 2.129]

Lo grandioso de la mecánica Lagrangiana es que un escalar resume la dinámica de un sistema. Buscar una expresión escalar para las fuerzas disipativas sería lo más conveniente. Considérese la siguiente expresión, basada en una función de la norma de la velocidad v:

$$f(v) = \int_0^v h(v)dv \Longrightarrow h(v)\frac{\partial v}{\partial \dot{q}j} = \frac{af(v)}{\partial \dot{q}_j} = \frac{af(v)}{\partial \dot{q}_j}\frac{\partial v}{\partial \dot{q}j}$$

[Ec. 2.130]

Entonces, la fuerza generalizada Q_J estaría dada de la siguiente forma:

$$Q_j = -\sum_{i=1}^{N}\frac{\partial v_i}{\partial \dot{q}j}$$

[Ec. 2.131]

$$= -\frac{\partial}{\partial \dot{q}j}\left(\sum_{i=1}^{N} f(v_{\boldsymbol{i}})\right)$$

[Ec. 2.132]

$$= -\frac{\partial}{\partial \dot{q}j}\left(\sum_{i=1}^{N}\int_0^{v_{\boldsymbol{i}}} h(v_{\boldsymbol{i}})dv_{\boldsymbol{i}}\right)$$

[Ec. 2.133]

Interpretando la ecuación 2.129, se ve que Q_J permite insertar en la ecuación de movimiento de Lagrange, una expresión para la potencia, $P = \sum_{i=1}^{N}\int \mathrm{v_i}\, h(v_\mathrm{i})dv_\mathrm{i}$, generada por las fuerzas viscosas, ya que $h\ (v_\mathrm{i})$ representa la norma de R_i, la cual se encuentra en la dirección contraria de la velocidad **v***i* de la partícula, su producto punto es $R_\mathrm{i} \cdot dv_\mathrm{i} = (v_\mathrm{i})dv_\mathrm{i}$. Por tanto, para un sistema en presencia de fuerzas no conservativas disipativas, la ecuación de movimiento de Lagrange sería:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial q_J}\right) - \frac{\partial L}{\partial q_J} + \frac{\partial P}{\partial q_J} = 0$$

[Ec. 133]

Donde P representa la potencia disipada por las fuerzas R_i:

Tabla XX. **Teorema de ecuación general de movimiento de lagrange**

Sea $L = T - V$ el Lagrangiano de un sistema dinámico, el cual se encuentra bajo el efecto de fuerzas conservativas, fuerzas no conservativas disipativas, las cuales disipan una potencia P, y otras fuerzas no conservativas $\boldsymbol{F}_J = ((F_J)_1, \ldots, (F_J)_M)^{\mathrm{T}}$, cuyas transformaciones a fuerzas generalizadas están dadas por $\boldsymbol{Q}_J = ((Q_J)_1, \ldots, (Q_J)_M)^{\mathrm{T}}$. T y V son la energía cinética y potencial del sistema respectivamente. Las ecuaciones generales de movimiento de Lagrange en un espacio de N grados de libertad generan, en su solución las ecuaciones de movimiento del sistema dinámico.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}j}\right) - \frac{\partial L}{\partial \dot{q}j} + \frac{\partial p}{\partial \dot{q}j} - \sum_{i=1}^{M}(Q_i)_i = 0 \qquad \forall i \in \{1,2, \ldots \ldots N\}$$

[Ec. 2.135]

Fuente: MALHAM, Simon. J. *An introduction to lagrangian and hamiltonian mechanics*. p. 100.

2.1.3. Algunos modelos importantes

- Resortes ideales: la forma más común de representar matemáticamente la relación de la fuerza F ejercida por un resorte ideal con la elongación o alargamiento δ provocado por la fuerza externa aplicada al extremo del mismo:

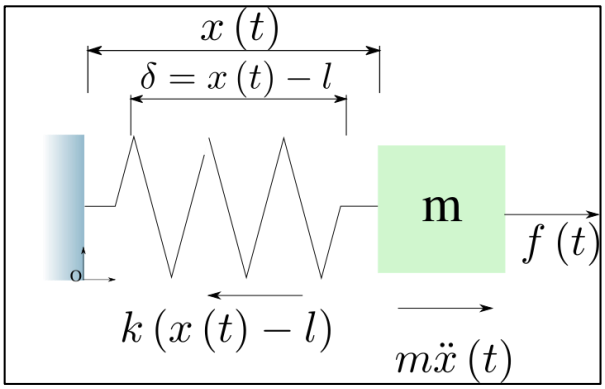
$$F = -k\delta \qquad \text{[Ec. 2.136]}$$

Donde k por lo general es conocida como constante elástica del resorte, el caso más común es tomarla como una constante, pero puede ser variable en el tiempo, y δ es la elongación, que es la variación que experimenta su longitud. Todos los resortes en la vida real, de alguna forma u otra no son ideales. Lo que implica que no cumplen con la ecuación 2.136. Sin embargo, si δ es muy pequeña, la fuerza F se puede expresar de forma lineal. La energía de deformación o energía potencial elástica U_{K} asociada al estiramiento del resorte viene dada por la siguiente ecuación:

$$U_K = \frac{1}{2} k\delta^2 \quad \text{[Ec. 2.137]}$$

Un resorte ideal es un elemento que almacena solamente energía potencial y su equivalente es el capacitor.

Figura 11. **Representación de un resorte ideal, con una masa a un extremo**



Fuente: elaboración propia, empleando Inkscape.

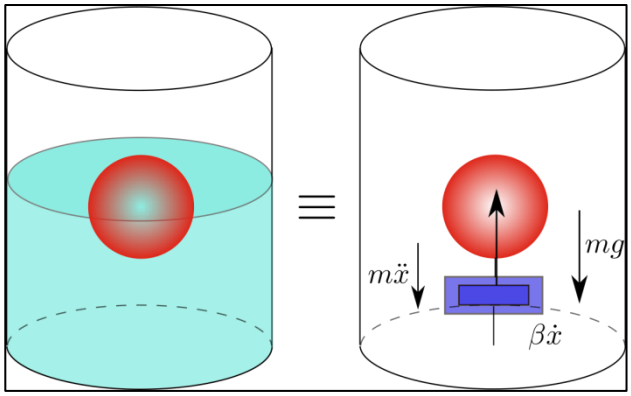
La figura 11 muestra un sistema dinámico de masa-resorte, bajo la acción de una fuerza externa $f\,(t)$. Se puede apreciar la representación más común de un resorte ideal. En la figura se pueden apreciar las fuerzas que afectan a la masa. La elongación $\delta = x(t) = l$ depende del tiempo, l es una constante propia del resorte que es la distancia original del resorte cuando no se encuentra comprimido ni estirado, se puede ver que cuando $x(t) = l$ se tiene que $\delta = 0$.

- Amortiguamiento: se define como la capacidad de un sistema o cuerpo para disipar energía cinética en otro tipo de energía. Existen diversas modelizaciones de amortiguamiento. El más simple de ellos, consta de

una partícula o masa concentrada, que va perdiendo velocidad bajo la acción de una fuerza de amortiguamiento proporcional a su velocidad:

$$\mathrm{R(t)} = \beta \frac{dx(t)}{dt} \qquad \text{[Ec. 2.138]}$$

Figura 12. **Abstracción de la fricción ejercida por un fluido**



Fuente: elaboración propia, empleando Inkscape.

Donde B es el amortiguamiento real del sistema medido en $N/(m/s)$. Por lo general, se asume que es una constante. Esta fuerza también es conocida como amortiguamiento viscoso o amortiguamiento lineal, debido a que es una relación lineal entre la fuerza aplicada y la velocidad. Este modelo es aproximadamente válido para describir la amortiguación por fricción entre superficies de sólidos, o el frenado de un sólido en el seno de un fluido en régimen laminar y es por eso que recibe el nombre de fricción viscosa. En la figura 12, se presenta un cuerpo de masa *m* cayendo en un fluido, se puede ver la abstracción que se hace de la fricción ejercida por el fluido. Si β es constante la potencia disipada está dada por una fuerza viscosa:

$$P = \frac{1}{2}\,\beta(\dot{x})^2 \qquad \text{[Ec. 2.139]}$$

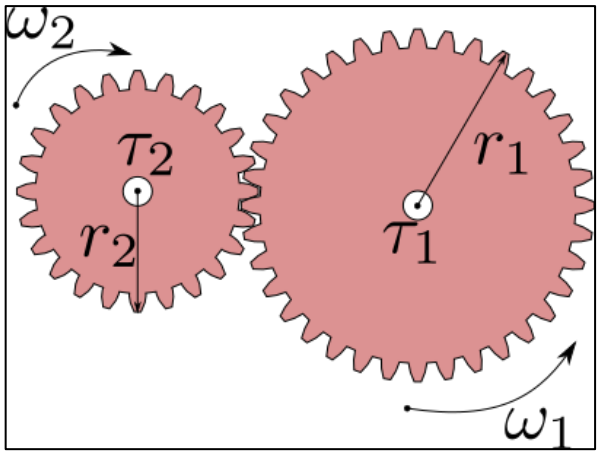
- Fricción de Coulomb: es una fuerza que tiene una amplitud constante con respecto al cambio de velocidad, pero el signo de la fuerza de fricción cambia al invertir la dirección de la velocidad. Es posible expresarla con la siguiente expresión matemática:

$$Rc\,(t) = Fc\frac{\dot{x}}{|\dot{x}|} \qquad \text{[Ec. 2.140]}$$

Donde $x(\dot{t})$ es la velocidad, con que se mueve la partícula en un entorno dado.

- Engranaje: es un elemento utilizado para transferir potencia de un eje girando a otro, con el fin de alterar algunas propiedades como velocidad o torsión.

Figura 13. **Dos engranajes acoplados**



Fuente: elaboración propia, empleando Inkscape.

En la práctica, los engranajes tienen inercia y fricción entre los dientes. A menudo no es posible despreciarlos. Usualmente son diseñados para reducir la fricción y operar silenciosamente. Una acción suave y sin vibraciones se asegura dando una forma geométrica adecuada a los dientes que son los encargados de transferir la potencia.

Al momento de construir un sistema mecánico es posible encontrar engranajes de diferentes formas, por ejemplo, engranajes de espuela, que son los básicos y son los que se muestran en la figura 13. También podemos encontrar engranajes helicoidales, cónicos y tornillos sin fin, que son formas más avanzadas para la transmisión de potencia y sus diseños son más complicados independientemente de la forma del engranaje Si se tienen dos engranajes acoplados, se consideran los siguientes hechos en los modelos que involucren engranajes.

- Los dientes de ambos engranajes, son del mismo tamaño:

$$\frac{2\pi r_1}{N_1} = \frac{2\pi \mathrm{r}_2}{N_2} \Longrightarrow \qquad \text{[Ec. 2.141]}$$

$$\frac{r_1}{N_1} = \frac{\mathrm{r}_2}{N_2} \qquad \text{[Ec. 2.142]}$$

$$\frac{N_2}{N_1} = \frac{\mathrm{r}_2}{r_1} \qquad \text{[Ec. 2.143]}$$

- La distancia que viaja cada engranaje sobre su superficie es la misma:

$$\theta^1 r^1 = \theta_2 r_2 \Longrightarrow \qquad \text{[Ec. 2.144]}$$

$$\frac{\theta_2}{\theta_2} = \frac{\mathrm{r}_2}{r_1} \qquad \text{[Ec. 2.145]}$$

- Si se considera $\omega_1 = \ddot{\Theta}_1$ y $a\ \omega_2 = \ddot{\Theta}_2$ como las velocidades angulares de cada engranaje, del hecho anterior se concluye:

$$\Delta\theta_1 r_1 = \Delta\theta_2 r_2 \qquad \text{[Ec. 2.146]}$$

$$\frac{\Delta\theta_1}{\Delta t} r_1 = \frac{\Delta\theta_2}{\Delta t} r_2 \qquad \text{[Ec. 2.147]}$$

$$\lim_{\Delta t - 0} \frac{\Delta\theta_1}{\Delta t} r^1 = \lim_{\Delta t - 0} \frac{\Delta\theta_2}{\Delta t} r^2 \qquad \text{[Ec. 2.148]}$$

$$\dot{\theta}_1 r_1 = \dot{\theta}_2 r_2 \qquad \text{[Ec. 2.149]}$$

$$\omega_1 r_1 = \omega_2 r_2 \qquad \text{[Ec. 2.150]}$$

$$\frac{\omega_1}{\omega_2} = \frac{r_2}{r_1} \qquad \text{[Ec. 2.151]}$$

Considérese que el diseño de los engranajes reduce las pérdidas por fricción. Entonces el trabajo por ambos engranajes es el mismo:

$$\tau_1\theta_1 = \tau_2\theta_2 \Longrightarrow \qquad \text{[Ec. 2.152]}$$

$$\frac{\theta_1}{\theta_2} = \frac{\tau_2}{\tau_1} \qquad \text{[Ec. 2.153]}$$

Agrupando todos los hechos anteriores en una sola expresión se tiene:

$$\frac{\tau_2}{\tau_1} = \frac{\theta_1}{\theta_2} = \frac{r_2}{r_1} = \frac{\omega_1}{\omega_2} = \frac{N_2}{N_1}$$

[Ec. 2.154]

En la figura 14 se muestra una abstracción de un tren de engranajes. De igual forma que en el desplazamiento lineal, para el desplazamiento angular vale la pena tratar torsiones equivalentes a las fuerzas lineales anteriormente mencionadas.

- Inercia rotacional. Es la propiedad de un elemento de almacenar energía cinética rotacional.

- Resorte torsional. Como el resorte lineal, la constante del resorte torsional K, dada en torsión por unidad de desplazamiento angular, puede representar la compliancia de una varilla o un eje sujeto a una torsión aplicada. Dicha torsión puede representarse por la ecuación:

$$T = K\theta(t) \qquad \text{[Ec. 2.155]}$$

- Fricciones para movimiento de rotación. Los tipos de fricción mencionados anteriormente tienen un equivalente para torsiones

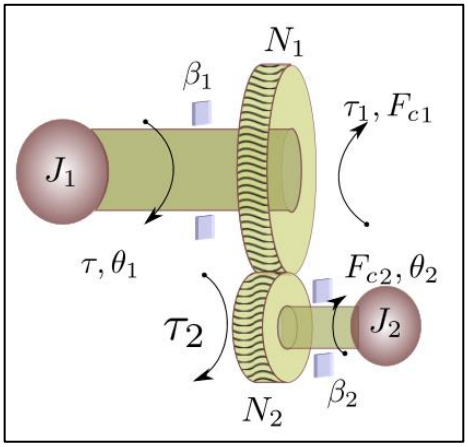
Torsión viscosa: $T(t) = \beta\ddot{\Theta}$

[Ec. 2.156]

Torsión de Coulomb: $T\,(t) = Fc\frac{\ddot{\Theta}}{|\ddot{\Theta}|}$

[Ec. 2.157]

Figura 14. **Tren de engranajes**



Fuente: elaboración propia, empleando Inkscape.

En la figura 14, se muestra un tren de engranajes, cuyo comportamiento es posible aproximar usando torsiones viscosas, de Coulomb o resortes torsionales. La figura 15 no ilustra el tren de engranajes en sí, sino un modelo del tren, sujeto a los parámetros $\beta_1 \, y \, \boldsymbol{\beta}_2$ para las fricciones viscosas, $Fc_{\,1} \, y \, Fc_{\,2}$ como fricciones de Coulomb, $J_1 \, y \, J_2$ inercias o cargas que llevan a cuesta los engranajes , y τ es una torsión externa aplicada a este sistema. Para la inercia J_1 se tiene esta ecuación:

$$J_1\ddot{\Theta}_1 = \tau - \beta_1\ddot{\Theta}_1 - Fc_1\frac{\ddot{\Theta}}{|\ddot{\Theta}_1|} - \tau_1$$

[Ec. 2.158]

Para la inercia J_2 se tiene esta ecuación:

$$J_2\ddot{\Theta}_2 = \tau_2 - \beta_2\ddot{\Theta}_2 - Fc_2\frac{\ddot{\Theta}_2}{|\ddot{\Theta}_2|}$$

[Ec. 2.159]

Las torsiones $\tau_1 \, y \, \tau_2$ se encuentran relacionadas de la siguiente forma:

$$\tau_1 = \frac{N_1}{|N_2|}\tau_2$$

[Ec. 2.160]

Lo cual implica

$$\tau_1 = \frac{N_1}{N_2}\tau^2 \qquad \Rightarrow$$

[Ec. 2.161]

$$= \frac{N_1}{N_2}\left(J_2\ddot{\theta}_2 + \beta_2\ddot{\theta}_2 + F_{c2}\frac{\dot{\theta}_2}{|\dot{\theta}_2|}\right) \qquad \Rightarrow$$

[Ec. 2.162]

$$= \left(\frac{N_1}{N_2}\right)^2 J_2\ddot{\theta}_2 + \left(\frac{N_1}{N_2}\right)^2 \beta_2\ddot{\theta}_2 + \left(\frac{N_1}{N_2}\right)^2 F_{c2}\frac{\dot{\theta}_2}{|\dot{\theta}_2|}$$

[Ec. 2.163]

Al final se sustituye la expresión anterior en la ecuación para el engranaje 1:

$$J_1\ddot{\theta}_1 = \tau - \beta_1\ddot{\theta}_1 - F_{c1}\frac{\dot{\theta}}{|\dot{\theta}_1|} - \left(\frac{N_1}{N_2}\right)^2 J_2\ddot{\theta}_1 - \left(\frac{N_1}{N_2}\right)^2 \beta_2\ddot{\theta}_1 - \left(\frac{N_1}{N_2}\right) F_{c2}\frac{\dot{\theta}_2}{|\dot{\theta}_2|}$$

[Ec. 2.164]

$$\left(J_1 + \left(\frac{N_1}{N_2}\right)^2 J_2\right)\ddot{\theta}_1 = \tau - \left(\beta_1 + \left(\frac{N_1}{N_2}\right)^2 \beta_2\right)\ddot{\theta}_1 - \left(F_{c1}\frac{N_1}{N_2} + F_{c2}\frac{\dot{\theta}_2}{|\dot{\theta}_2|}\right)$$

[Ec. 2.165]

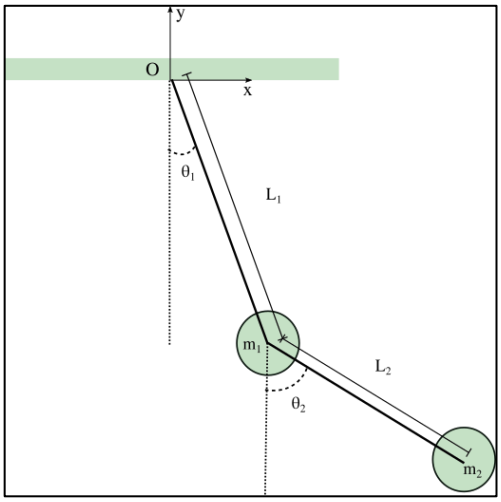
Teniendo así la ecuación de movimiento para θ_1. De la ecuación 2.164 se concluye que se puede reflejar la inercia, la fricción, compliancia, torsión, velocidad y desplazamiento desde un lado del tren.

2.1.4. Ecuaciones de movimiento

- Péndulo doble. Mecánica de Newton

Considérese una cuenta de masa *m*1, la cual cuelga del punto *O*, sostenida por una varilla de longitud l_1 y masa despreciable. Del centro de masa cuelga otra cuenta con una masa m_2. Esta última cuenta se sostiene por una varilla de longitud l_2 de masa despreciable.

Figura 15. **Péndulo doble**



Fuente: elaboración propia, empleando Inkscape.

En esta ocasión, se obtendrán las ecuaciones de movimiento del sistema, utilizando las leyes de Newton. Si se usa el sistema de referencia mostrado en la figura 16, se pueden dar las posiciones de las cuentas en términos de $\theta_1 \, y \, \theta_2$. Usando el sistema de coordenadas ortogonales (x, y) y tomando como origen el punto O. Se puede decir que la posición $\boldsymbol{r}_1$ de la cuenta que cuelga por la varilla de longitud l_1 está dada por:

$$r_1 = l_1(sin\theta_1, cos\theta_1) \qquad \text{[Ec. 2.166]}$$

De igual forma en $\boldsymbol{r}_2$, la posición de la segunda cuenta, esta dada por:

$$r_2 = r_1 + l_2(sin\theta_2, cos\theta_2) \qquad \text{[Ec. 2.167]}$$

Donde $\theta_1 \, y \, \theta_2$ dependen del tiempo. Considerando las varillas rígidas, $l_1 \, y \, l_2$ son constantes. De las expresiones para $\boldsymbol{r}_1 \, y \, \boldsymbol{r}_2$ se puede obtener las velocidades $\dot{r}_1 \, y \, \dot{r}_2$:

$$\dot{\boldsymbol{r}}_1 = l_1\dot{\theta}_1(\cos\theta_1, -\sin\theta_1) \qquad \text{[Ec. 2.168]}$$

$$\dot{\boldsymbol{r}}_2 = \dot{\boldsymbol{r}}_1 + l_2\dot{\theta}_2(\cos\theta_2, -\sin\theta_2) \qquad \text{[Ec. 2.169]}$$

respectivamente. De igual forma se hace para obtener las aceleraciones $\ddot{r}_1 \, y \, \ddot{r}_2$:

$$\ddot{\boldsymbol{r}}_1 = l_1\ddot{\theta}_1(\cos\theta_1, -\sin\theta_1) - l_1\left(\dot{\theta}_1\right)^2(\sin\theta_1, \cos\theta_1) = \ddot{\theta}_1\dot{\boldsymbol{r}}_1 - \left(\dot{\theta}_1\right)^2\boldsymbol{r}_1$$

[Ec. 2.170]

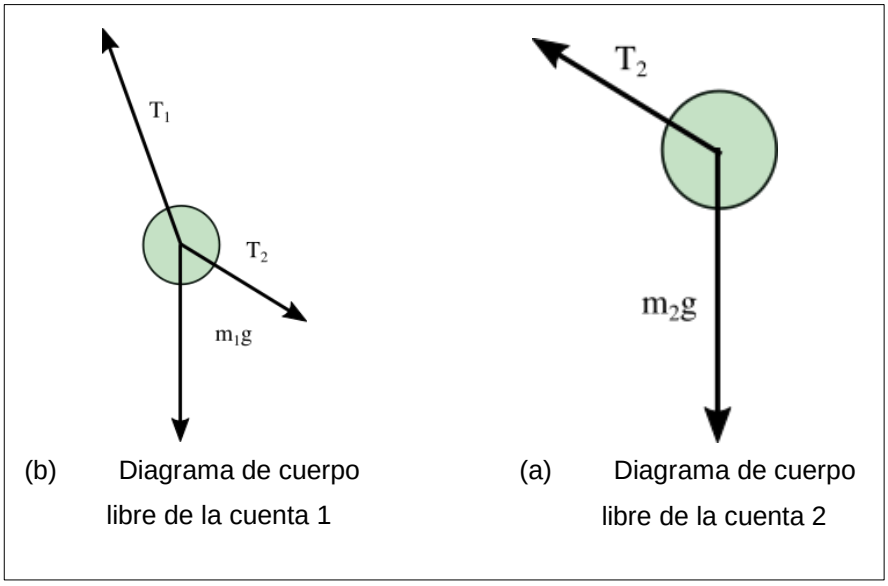
$$\ddot{\boldsymbol{r}}_2 = \ddot{\boldsymbol{r}}_1 + l_1\ddot{\theta}_1(\cos\theta_1, -\sin\theta_1) - l_1\left(\dot{\theta}_1\right)^2(\sin\theta_1, \cos\theta_1) = \ddot{\boldsymbol{r}}_1 + \ddot{\theta}_1\dot{\boldsymbol{r}}_1 - \left(\dot{\theta}_1\right)^2\boldsymbol{r}_1$$

[Ec. 2.171]

La figura 16 muestra los diagramas de cuerpo libre para cada una de las cuentas. Donde $\boldsymbol{T}_1 \, y \, \boldsymbol{T}_2$ son las representaciones vectoriales de las fuerzas de

tensión presentes en cada varilla. Y $\boldsymbol{g}$ es la representación vectorial de la fuerza de gravedad. Todas ellas en el sistema de referencia, anteriormente descrito.

Figura 16. **Diagramas de cuerpo libre**



Fuente: elaboración propia, empleando Inkscape.

Del diagrama de cuerpo libre 16a, se plantea

$$m_1\dot{\mathrm{r}}_1 = T_1 - T_2 + m_1 g$$

[Ec. 2.172]

Como $\boldsymbol{T}_1\, y\, \boldsymbol{T}_2$ son una representación vectorial, se pueden escribir como un vector unitario de dirección, escalado por una magnitud:

$$m_1\ddot{\boldsymbol{r}}_1 = -\boldsymbol{T}_1\frac{\boldsymbol{r}_1}{|\boldsymbol{r}_1|} + \boldsymbol{T}_1\frac{\boldsymbol{r}_2-\boldsymbol{r}_1}{|\boldsymbol{r}_2-\boldsymbol{r}_1|} + m_1\boldsymbol{g} \qquad \text{[Ec. 2.173]}$$

$$m_1\ddot{\boldsymbol{r}}_1 = -\boldsymbol{T}_1\frac{\boldsymbol{r}_1}{\boldsymbol{l}_1} + \boldsymbol{T}_2\frac{\boldsymbol{r}_2-\boldsymbol{r}_1}{\boldsymbol{l}_2} + m_1\boldsymbol{g}$$

[Ec. 2.174]

De igual forma para la segunda cuenta, de la figura 16b se puede plantear:

$$m_2\dot{\mathrm{r}}_2 = T_2 + m_2 g \qquad \text{[Ec. 2.175]}$$

Y de la misma forma que en la primera cuenta se escriben las fuerzas de tensión como magnitud y dirección:

$$m_2\ddot{\boldsymbol{r}}_2 = -\boldsymbol{T}_2\frac{\boldsymbol{r}_1-\boldsymbol{r}_2}{|\boldsymbol{r}_2-\boldsymbol{r}_1|} + m_1\boldsymbol{g} \qquad \text{[Ec. 2.176]}$$

$$m_2\ddot{\boldsymbol{r}}_2 = -\boldsymbol{T}_2\frac{\boldsymbol{r}_1-\boldsymbol{r}_2}{l_2} + m_1\boldsymbol{g} \qquad \text{[Ec. 2.177]}$$

Ahora se procede a plantear sistemas de ecuaciones que involucren a las fuerzas de tensión y los ángulos. Planteando una ecuación por cada componente que forma cada vector:

$$m_1 l_1\left(\ddot{\theta}_1\cos\theta_1 - \left(\dot{\theta}_1\right)^2\sin\theta_1\right) = -T_1\sin\theta_1 + T_2\sin\theta_2$$

[Ec. 2.178]

$$-m_1 l_1\left(\ddot{\theta}_1\cos\theta_1 - \left(\dot{\theta}_1\right)^2\sin\theta_1\right) = -T_1\sin\theta_1 + T_2\sin\theta_2 - m_1 g$$

[Ec. 2.179]

$$m_2\left(l_1\ddot{\theta}_1\cos\theta_1 - l_1\left(\dot{\theta}_1\right)^2\sin\theta_1 + l_2\ddot{\theta}_2\cos\theta_2 - l_2\left(\dot{\theta}_2\right)^2\sin\theta_2\right)T_2\sin\theta_2$$

[Ec. 2.180]

$$-m_2\left(l_1\ddot{\theta}_1\cos\theta_1 - l_1\left(\dot{\theta}_1\right)^2\sin\theta_1 + l_2\ddot{\theta}_2\cos\theta_2 - l_2\left(\dot{\theta}_2\right)^2\sin\theta_2\right) - T_2\sin\theta_2 - m_2 g$$

[Ec. 2.181]

Si se escala ambos lados de la ecuación 2.177 por un factor $sin\theta_1$ y la ecuación 2.178 por un factor $cos\theta_1$, se obtiene:

$$m_1 l_1 \left(\ddot{\theta}_1 \cos\theta_1 - \left(\dot{\theta}_1\right)^2 \sin\theta_1\right) = -T_1 \sin\theta_1 + T_2 \sin\theta_2 \sin\theta_1$$

[Ec. 2.182]

$$-m_1 l_1 \left(\ddot{\theta}_1 \cos\theta_1 - \left(\dot{\theta}_1\right)^2 \sin\theta_1\right) = -T_1 \sin\theta_1 + T_2 c \sin\theta_1 - m_1 g \cos\theta_1$$

[Ec. 2.183]

$$\cos^2\theta_1 + \sin^2\theta_1 = \cos^2\theta_2 + \sin^2\theta_2 = 1$$

[Ec. 2.184]

$$\cos(\theta_2 - \theta_1) = \cos\theta_2 \cos\theta_1 + \sin\theta_2 \sin\theta_1$$

[Ec. 2.185]

Al combinar las ecuaciones 2.181 y 2.182, se tiene una expresión para $= m_1 l_1 (\ddot{\Theta}_1)^2$:

$$-m_1 l_1 \left(\ddot{\theta}_1\right)^2 = -T_1 + T_2 \cos(\theta_2 - \theta_1) - m_1 g \cos\theta_1$$

[Ec. 2.186]

Por otro lado escalando ambos lados la ecuación 2.178 por un factor $cos\theta_1$ y la ecuación 2.179 por un factor $-sin\theta_1$:

$$m_1 l_1 \left(\ddot{\theta}_1 \cos^2\theta_1 - \left(\dot{0}_1\right)^2 \sin\theta_1 \cos\theta_1\right) = -T_1 \sin\theta_1 \cos\theta_1 + T_2 \sin\theta_2 \cos\theta_1$$

[Ec. 2.187]

$$m_1 l_1 \left(\ddot{\theta}_1 \cos^2\theta_1 - \left(\dot{0}_1\right)^2 \sin\theta_1 \cos\theta_1\right) = -T_1 \cos\theta_1 \sin\theta_1 + T_2 \cos\theta_2 \sin\theta_1 + m_1 g \sin\theta_1$$

[Ec. 2.188]

Teniendo en cuenta:

$$\sin(\theta_2 - \theta_1) = \sin\theta_2 \cos\theta_1 - \sin\theta_1 \cos\theta_2$$

[Ec. 2.189]

si se combinan las ecuaciones 2.186 y 2.187, se puede encontrar una expresión para $m_1 l_1 \ddot{\theta}_1$:

$$m_1 l_1 \ddot{\theta}_1 = T_2 \sin(\theta_2 - \theta_1) + m_1 g \sin\theta_1$$

[Ec. 2.190]

Ahora se escalan ambos lados de la ecuación 179 por un factor $sin\theta_2$ y la ecuación 180 por un factor $cos\theta_2$:

$$m_2 \left(l_1 \ddot{\theta}_1 \sin\theta_1 \cos\theta_2 - l_1 \left(\dot{\theta}_1\right)^2 \sin\theta_1 \sin\theta_2 + l_2 \ddot{\theta}_2 \sin\theta_2 \cos\theta_2 - l_2 \left(\dot{\theta}_2\right)^2 \sin^2\theta_2 \right) = -T_2 \sin^2\theta_2$$

[Ec. 2.191]

$$-m_2 \left(l_1 \ddot{\theta}_1 \sin\theta_1 \cos\theta_2 - l_1 \left(\dot{\theta}_1\right)^2 \sin\theta_1 \sin\theta_2 + l_2 \ddot{\theta}_2 \sin\theta_2 \cos\theta_2 - l_2 \left(\dot{\theta}_2\right)^2 \sin^2\theta_2 \right) = -T_2 \sin^2\theta_2 - m_2 g \cos\theta_2$$

[Ec. 2.192]

Al combinar las expresiones anteriores se obtiene una expresión para $m_2 l_2 (\ddot{\theta}^2)^2$:

$$m_2 l_2 \left(\dot{\theta}_2\right)^2 = m_2 l_1 \ddot{\theta}_1 \sin(\theta_2 - \theta_1) - m_2 l_1 \left(\dot{\theta}_1\right)^2 \cos(\theta_2 - \theta_1) + T_2 + m_2 g \cos\theta_2$$

[Ec. 2.193]

Para encontrar una expresión para $m_2 l_2 \ddot{\Theta}_2$, se escala la ecuación 179 por un factor $cos\theta_2$ y la ecuación 2.180 por un factor $-sin\theta_2$:

$$m_2\left(l_1\ddot{\Theta}_1\cos\theta_1\cos\theta_2 - l_1\left(\dot{\theta}_1\right)^2\sin\theta_1\cos\theta_2 + l_2\ddot{\Theta}_2\cos\theta_2 - l_2\left(\dot{\theta}_2\right)^2\sin\theta_2\cos\theta_2\right) = -T_2\cos\theta_2\cos\theta_2$$

[Ec. 2.194]

$$m_2\left(l_1\ddot{\Theta}_1\sin\theta_1\sin\theta_2 + l_1\left(\dot{\theta}_1\right)^2\cos\theta_1\sin\theta_2 + l_2\ddot{\Theta}_2\sin\theta_2 + l_2\left(\dot{\theta}_2\right)^2\cos\theta_2\sin\theta_2\right) = -T_2\cos\theta_2\sin\theta_2 + m_2 g\cos\theta_2$$

[Ec. 2.195]

Y combinando las ecuaciones anteriores se obtiene:

$$m_2 l_2\ddot{\theta}_2 = m_2 l_1\ddot{\theta}_1\cos(\theta_2-\theta_1) - m_2 l_1\left(\dot{\theta}_1\right)^2\sin(\theta_2-\theta_1) + m_2 g\sin\theta_2$$

[Ec. 2.196]

Multiplicando ambos lados de la ecuación 2.192 por sin $(\theta_2-\theta_1)$ y reduciendo términos:

$$m_2 l_2\left(\ddot{\theta}_2\right)^2\sin(\theta_2-\theta_1) = m_2 l_1\sin(\theta_2-\theta_1)\left(\ddot{\theta}_1\sin(\theta_2-\theta_1)^2\cos(\theta_2-\theta_1) + \frac{g}{l_1}\cos\theta_2\right) + T_2\sin(\theta_2-\theta_1)$$

[Ec. 2.197]

De la ecuación anterior y la ecuación 2.185:

$$m_2 l_2 \left(\ddot{\theta}_2\right)^2 \sin(\theta_2 - \theta_1) = m_2 l_1 \sin(\theta_2 - \theta_1) \left(\ddot{\theta}_1 \sin(\theta_2 - \theta_1)^2 \cos(\theta_2 - \theta_1) + \frac{g}{l_1} \cos\theta_2\right) + m_1 l_1 \ddot{\theta}_1 - m_1 g \sin\theta_1$$

[Ec. 2.198]

Despejando las derivadas de mayor orden de la ecuación 2.197 y la ecuación 195 se obtiene las ecuaciones de movimiento:

$$\ddot{\theta}_1 = \frac{m_2 \sin(\theta_2 - \theta_1) \left(l_2 \left(\dot{\theta}_2\right)^2 \cos(\theta_2 - \theta_1) - g \cos\theta_2\right)}{m_1 l_1 + m_2 l_1 \sin^2(\theta_2 - \theta_1)}$$

[Ec. 2.199]

$$\ddot{\theta}_2 = -\frac{l_1}{l_2} \ddot{\theta}_1 \cos(\theta_2 - \theta_1) - \frac{l_1}{l_2} \left(\ddot{\theta}_1\right)^2 \sin(\theta_2 - \theta_1) + \frac{g}{l_2} \sin\theta_2$$

[Ec. 2.200]

- Péndulo doble. Mecánica analítica. Para encontrar las ecuaciones de movimiento, usando la formulación de Lagrange, es necesario plantear una expresión para la energía cinética y potencial del sistema, y así poder escribir el Lagrangiano del mismo. Los grados de libertad a utilizar serán θ_1 y θ_2. Para escribir el lagrangiano se necesita la norma cuadrada de la velocidad, v $1/2$, de la primera cuenta (la que cuelga del origen *O*), y la norma cuadrada de la velocidad, $v2/2$, de la segunda cuenta. Si se escriben los vectores en el sistema de coordenadas (x, y), la posición de la primera cuenta está dada por:

$$m_i \ddot{\boldsymbol{r}}_1 = -T_1 - T_2 + m_1 g \qquad \text{[Ec. 2.201]}$$

La posición de la segunda cuenta esta dada por $\boldsymbol{r}_2$, escribir la posición de la segunda cuenta es más sencillo al considerar la diferencia vectorial $\boldsymbol{r}_2 - \boldsymbol{r}_1$:

$$m_i \ddot{\boldsymbol{r}}_1 = -\boldsymbol{T}_1 \frac{r_1}{|r_1|} + \boldsymbol{T}_1 \frac{r_2 - r_1}{|r_2 - r_1|} + m_1 g$$

[Ec. 2.202]

Buscando una expresión para la velocidad $\boldsymbol{v}_1$, se tiene:

$$v_1 = \dot{r_1}$$ [Ec. 2.203]

$$v_1 = \frac{dr_1}{dt}$$ [Ec. 2.204]

$$v_1 = l_1\left(\theta_1 \dot{\cos}(\theta_1), -\dot{\theta^1} \sin(\theta_1)\right)$$ [Ec. 2.205]

De la misma forma para la segunda expresión:

$$v_2 = \dot{r_2}$$ [Ec. 2.206]

$$v_2 = \dot{t}_{\,2} - \dot{r}_1 + \dot{r_1}$$ [Ec. 2.207]

$$v_2 = \frac{d(r_2 - r_1)}{dt} + \dot{r_1}$$ [Ec. 2.208]

Se procede a calcular $v^2_1 = |\boldsymbol{v}_1|^2$:

$$v_1^2 = v_1 \cdot v_1$$ [Ec. 2.209]

$$= l_1^2(\theta_1^2 cos^2\theta_1 + \theta_1^2 sin^2\theta_1)$$ [Ec. 2.210]

$$= l_1^2 \dot{\theta}_1^2$$ [Ec. 2.211]

Para la segunda cuenta se procede de la siguiente forma:

$$v_2^2 = v_2 \cdot v_2 \qquad \text{[Ec. 2.212]}$$

$$= \left(\frac{d(r_2 - r_1)}{dt}\dot{\boldsymbol{r}}_1\right) \cdot \left(\frac{d(r_2 - r_1)}{dt} + \dot{\boldsymbol{r}}_1\right)$$

[Ec. 2.213]

$$= \left|\frac{d(r_2 - r_1)}{dt}\right|^2 + |\dot{\boldsymbol{r}}_1|^2 - 2\left(\frac{d(r_2 - r_1)}{dt}\right) \cdot (\dot{\boldsymbol{r}}_1)$$

[Ec. 2.214]

$$= l_2^2\dot{\theta}_2^2 + l_1^2\dot{\theta}_1^2 - 2l_1l_2\dot{\theta}_1\dot{\theta}_2(\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2)$$

[Ec. 2.215]

$$= l_2^2\dot{\theta}_2^2 + l_1^2\dot{\theta}_1^2 - 2l_1l_2\dot{\theta}_1\dot{\theta}_2(\cos(\theta_2 - \theta_1))$$

[Ec. 2.216]

Dado que solo dos cuerpos se encuentran en el sistema, ambos se considerados partículas, la energía cinética T del sistema está dada por:

$$T = \tfrac{1}{2}m_1(v_1^2) + \tfrac{1}{2}m_2(v_2^2) \qquad \text{[Ec. 2.217]}$$

$$= \frac{1}{2}m_1\left(l_1^2\dot{\theta}_1^2\right) + \frac{1}{2}m_2\left(l_2^2\dot{\theta}_2^2 + l_1^2\dot{\theta}_1^2 - 2l_1l_2\dot{\theta}_1\dot{\theta}_2(\cos(\theta_2 - \theta_1))\right)$$

[Ec. 2.218]

$$= \frac{1}{2}(m_1 + m_2)\left(l_1^2\dot{\theta}_1^2\right) + \frac{1}{2}m_2\left(l_2^2\dot{\theta}_2^2\right) - l_1l_2m_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_2 - \theta_1)$$

[Ec. 2.219]

El cálculo de la energía potencial V es más simple, y está dado por:

$$V = -m_1 \mathrm{g} \cdot \boldsymbol{r}_1 - m_2 \mathrm{g} \cdot \boldsymbol{r}_2 \qquad \text{[Ec. 2.220]}$$

$$= -m_1 \mathrm{g} l_1 \cos\theta_1 - m_2 \mathrm{g}(l_1 \cos\theta_1 + l_2 \cos\theta_2) \qquad \text{[Ec. 2.221]}$$

$$= -(m_1 + m_2)\mathrm{g} l_1 \cos\theta_1 - m_2 \mathrm{g} \cos\theta_2 \qquad \text{[Ec. 2.222]}$$

Dadas las energías cinética y potencial, $T\ y\ V$ respectivamente, se escribe una expresión para el lagrangiano:

$$L = T - V \qquad \text{[Ec. 2.223]}$$

$$= \frac{1}{2}(m_1 + m_2)\left(l_1^2 \dot{\theta}_1^2\right) + \frac{1}{2} m_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_2 - \theta_1) + (m_1 + m_2)\mathrm{g} l_1 \cos\theta_1 + m_2 \mathrm{g} l_2 \cos\theta_2$$

[Ec. 2.224]

Ahora se enuncian las ecuaciones de movimiento de Lagrange. Para el grado de libertad θ_1, se tiene:

$$\frac{\partial L}{\partial \dot{\theta}_1} = \left(m_1 + m_2 l_1^2 \dot{\theta}_1 - l_1 l_2 m_2 \dot{\theta}_2 \cos(\theta_2 - \theta_1)\right) \qquad \text{[Ec. 2.225]}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) = (m_1 + m_2) l_1^2 \dot{\theta}_1 + l_1 l_2 m_2 \ddot{\theta}_2 \cos(\theta_2 - \theta_1) \qquad \text{[Ec. 2.226]}$$

$$\frac{\partial L}{\partial \theta_1} = l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \sin(\theta_2 - \theta_1) - (m_1 + m_2) g l_1 \sin\theta_1 \qquad \text{[Ec. 2.227]}$$

La ecuación de Euler-Lagrange debe cumplir para θ_1:

$$0 = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) - \left(\frac{\partial L}{\partial \theta_2}\right) \qquad \text{[Ec. 2.228]}$$

$$0 = (m_1 + m_2) l_1^2 \ddot{\theta}_1 + m_2 l_1 \ddot{\theta}_2 - m_2 l_1^2 \ddot{\theta}_2^2 \sin(\theta_2 - \theta_1) + (m_1 + m_2) g l_1 \sin\theta_1$$

[Ec. 2.229]

Continuando para el grado de libertad θ_2:

$$\frac{\partial L}{\partial \dot{\theta}_2} = m_2 l_2^2 \dot{\theta}_2 + m_2 l_1 \dot{\theta}_1 \cos(\theta_2 - \theta_1) \qquad \text{[Ec. 2.230]}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) = m_2 l_2^2 \ddot{\theta}_2 + m_2 l_1 l_2 \ddot{\theta}_1 \cos(\theta_2 - \theta_1) - m_2 l_1 l_2 \dot{\theta}_1 \left(\dot{\theta}_2 - \dot{\theta}_1\right) \sin(\theta_2 - \theta_1)$$

[Ec. 2.231]

$$\frac{\partial L}{\partial \theta_2} = m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \sin(\theta_2 - \theta_1) - m_2 g l_2 \sin\theta_2$$

[Ec. 2.232]

La ecuación de Euler-Lagrange debe cumplir para θ_2:

$$0 = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) - \frac{\partial L}{\partial \theta_2}$$

[Ec. 2.233]

$$0 = m_2 l_2^2 \ddot{\theta}_2 + m_2 l_1 l_2 \ddot{\theta}_1 \cos(\theta_2 - \theta_1) + m_2 l_1 l_2 \dot{\theta}_1^2 \sin(\theta_2 - \theta_1) + m_2 g l_2 \sin\theta_2$$

[Ec. 2.234]

Con base en las ecuaciones de movimiento de Lagrange escritas para θ_1y θ_2, se expresan las ecuaciones de movimiento del sistema:

$$(m_1 + m_2)l_1\ddot{\theta}_1 + m_2 l_2\ddot{\theta}_2 \cos(\theta_2 - \theta_1) = m_2 l_2\ddot{\theta}_2^2 \sin(\theta_2 - \theta_1) - g\sin\theta_1$$

[Ec. 2.235]

$$l_2\ddot{\theta}_2 + l_1\ddot{\theta}_1 \cos(\theta_2 - \theta_1) - l_1\ddot{\theta}_1^2 \sin(\theta_2 - \theta_1) - g\sin\theta_2$$

[Ec. 2.236]

Si se multiplican ambos lados de la ecuación 2.235 por un factor $= m_2\cos(\theta^2 - \theta^1)$ y se suma el resultado a la ecuación 2.234, se puede ver que son iguales a las encontradas usando mecánica newtoniana. A pesar que la formulación de la mecánica lagrangiana es más compleja y requiere de un nivel de abstracción mayor que la newtoniana, la deducción de ecuaciones de movimiento se vuelve metódica. Este es un buen ejemplo de como un problema puede resolverse usando mecánica de Newton requiere de cierto ingenio y conocimiento de algunas identidades trigonométricas.

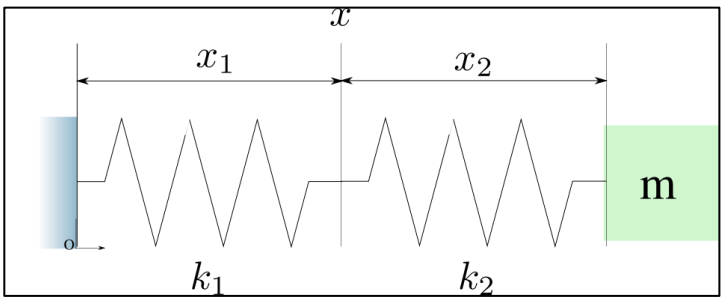
Por lado la resolución usando mecánica Lagrangiana es mucho más simple y metódica. La principal ventaja de modelar con mecánica Lagrangiana es que se pueden generalizar las ecuaciones, haciendo posible la simulación de modelos más generales, cosa que se encuentra muy limitada modelando con mecánica de Newton.

Otro ejemplo: si este sistema en lugar de haber tenido dos cuentas, hubiera tenido diez, por decir un número, la formulación con mecánica de Newton hubiera sido sumamente extenuante y enredada. Sin embargo, al usar la mecánica de Lagrange es posible generalizar la solución, con base en

matrices de rotación y traslación para determinar la posición de cada una de las cuentas. Dicho cálculo no sería realizado a mano sino por algún CAS como Maxima o algún software numérico como SCILAB. A continuación, se dan los siguientes ejemplos de modelos con mecánica Lagrangiana.

- Grados de libertad. Resorte doble, Considérese otro ejemplo. Se usará el principio de Hamilton para calcular la constante equivalente, keq de dos resortes sin masa en serie, tal como se muestra en la figura 17.

Figura 17. **Dos resortes en serie**



Fuente: elaboración propia, empleando Inkscape.

Cada resorte tiene constantes de elongación $k_1\, y\, k_2$, respectivamente. Al final de los dos resortes, se encuentra un objeto de masa m. Se usa el sistema de coordenadas cartesianas. Como el movimiento del objeto es lineal y las fuerzas que lo afectan se encuentran en la misma dirección, solo es necesaria la coordenada x. Las elongaciones de los resortes se encuentran determinadas por $\delta_1 = x_1 - l_1\, y\, \delta_2 = x_2 - l_2$. Por tanto la energía potencial *V* se encuentra en dichos términos. Teniendo así:

$$V = \frac{1}{2}k_1(x_1 - l_1)^2 + \frac{1}{2}k_2(x_2 - l_2)^2 \qquad \text{[Ec. 2.237]}$$

La energía cinética T , es:

$$T = \frac{1}{2} m\dot{x}^2 \qquad \text{[Ec. 2.238]}$$

Sea $X(x_2, x_2, t\,) = x$, la transformación de grados de libertad a coordenadas cartesianas y está dada por:

$$x = X(x_1, x_2, t) = x_1 + x_2 \qquad \text{[Ec. 2.239]}$$

Por tanto la velocidad $\dot{\mathrm{x}}$ está dada por:

$$\dot{\mathrm{x}} = \dot{\mathrm{x}}_1 + \dot{\mathrm{x}}_2 \qquad \text{[Ec. 2.240]}$$

Teniendo así una expresión para el lagrangiano del sistema en términos de $x_1\, y\, x_2$:

$$L(x_1, x_2) = T - V = \frac{1}{2} m(\dot{x}_1 + \dot{x}_2)^2 - \frac{1}{2} k_1 (x_1 - l_1)^2 - \frac{1}{2} k_2 (x_2 - l_2)^2$$

[Ec. 2.241]

Se escribe la ecuación de movimiento de Lagrange para x_1:

$$0 = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_1}\right) - \frac{\partial L}{\partial \dot{x}_1} \qquad \text{[Ec. 2.242]}$$

$$k_1(x_1 - l_1) + m(\ddot{x}_1 + \ddot{x}_2) \qquad \text{[Ec. 2.243]}$$

$$k_1(x_1 - l_1) + m\ddot{x} \qquad \text{[Ec. 2.244]}$$

Luego, la ecuación de movimiento de Lagrange para x_2:

$$0 = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_1}\right) - \frac{\partial L}{\partial \dot{x}_1} \quad \text{[Ec. 2.245]}$$

$$k_2(x_2 - l_2) + m(\ddot{x}_1 + \ddot{x}_2) \quad \text{[Ec. 2.246]}$$

$$k_2(x_2 - l_2) + m\ddot{x} \quad \text{[Ec. 2.247]}$$

Al combinar ambas ecuaciones se obtiene:

$$K_1(x_1 - l_1) = K_2(x^2 - l^2) \quad \text{[Ec. 2.248]}$$

El objetivo es escribir una ecuación para x de la forma:

$$m\ddot{x} = -keq(x - l_0) \quad \text{[Ec. 2.249]}$$

Considérese la identidad racional:

$$\frac{A}{B} = \frac{C}{D} \quad \text{[Ec. 2.250]}$$

$$\frac{A}{B} + 1 = \frac{C}{D} + 1 \quad \text{[Ec. 2.251]}$$

$$\frac{A}{B} + \frac{B}{B} = \frac{C}{D} + \frac{D}{D} \quad \text{[Ec. 2.252]}$$

$$\frac{A + B}{B} = \frac{C + D}{D} \quad \text{[Ec. 2.253]}$$

Por tanto de la ecuación 2.247:

$$\frac{k_1}{k_2} = \frac{x_2 - l_2}{x_1 - l_1} \quad \text{[Ec. 2.254]}$$

$$\frac{k_1 + k_2}{k_2} = \frac{x_2 - l_2 + x_1 - l_1}{x_1 - l_1} \quad \text{[Ec. 2.255]}$$

$$\frac{k_1 + k_2}{k_2} = \frac{x_2 + x_1 - l_2 - l_1}{x_1 - l_1} \quad \text{[Ec. 2.256]}$$

$$x_1 - l_1 = \frac{k_2}{k_1 + k_2}(x - l_0) \quad \text{[Ec. 2.257]}$$

$$k_1(x_1 - l_1) = \frac{k_2 k_1}{k_1 + k_2}(x - l_0) \quad \text{[Ec. 2.258]}$$

$$-m\ddot{x} = \frac{k_2 k_1}{k_1 + k_2}(x - l_0) \quad \text{[Ec. 2.259]}$$

De la cual se concluye que $Keq = \frac{k_1 k_2}{k_1 + k_2}$ y $\mathrm{l}_0 = \mathrm{l}_1 + \mathrm{l}_2$

- Una fuerza externa. Suponga ahora que la masa se ve afectada por una fuerza que varía en el tiempo $f\,(t\,)$, con una sola componente en x. Cuando existen fuerzas externas, la ecuación de movimiento de Lagrange, queda:

$$\frac{\partial L}{\partial x_1} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_1}\right) = -Q_1 \quad \text{[Ec. 2.260]}$$

$$\frac{\partial L}{\partial x_2} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_2}\right) = -Q_2 \quad \text{[Ec. 2.261]}$$

Donde $Q_1\ y\ Q_2$ están dadas por la transformación X:

$$Q_1 = f(t)\frac{\partial X}{\partial x_1} = f(t)\frac{\partial (x_2 + x_1)}{\partial x_1} = f(t)$$

[Ec. 2.262]

y

$$Q_2 = f(t)\frac{\partial X}{\partial x_2} = f(t)\frac{\partial (x_2 + x_1)}{\partial x_2} = f(t)$$

[Ec. 2.263]

por tanto las ecuaciones de movimiento, si el sistema se ve afectado por una fuerza externa $f\ (t\ )$, serían:

$$-k_1(x_1 - l_1) - m(\ddot{x}_1 + \ddot{x}_2) + f(t) = 0$$

[Ec. 2.264]

y

$$-k_2(x_2 - l_2) - m(\ddot{x}_1 + \ddot{x}_2) + f(t) = 0.$$

[Ec. 2.265]

Al realizar el procedimiento mostrado para encontrar la constante, se obtiene:

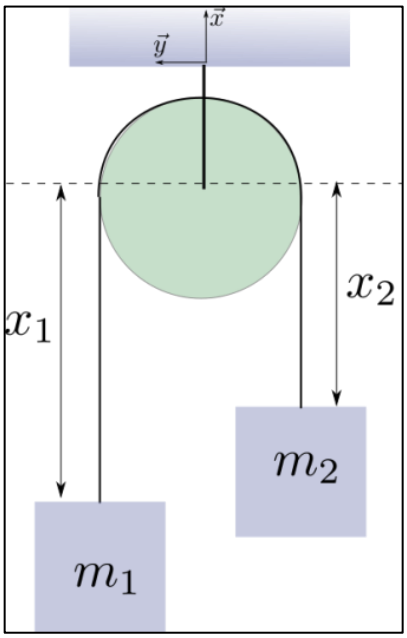
$$m\ddot{x} = f(t) - keq(x - l_0)$$

[Ec. 2.266]

Es posible sustituir dos resortes en serie, por uno solo con una constante keq.

- Fuerzas de restricción. Máquina de Atwood. Cuando se modelan fenómenos usando mecánica de Newton, usualmente es necesario introducir en los modelos fuerzas de restricción, aquellas fuerzas que limitan la libertad espacial de una partícula, tal como la tensión. El principio de d' Alembert enuncia que dichas fuerzas no realizan trabajo, y dado a ello al utilizar mecánica lagrangiana son necesarias para encontrar las ecuaciones de movimiento, pero se puede dar el caso que se deseen conocer por algún motivo.

Figura 18. **Máquina de Atwood**



Fuente: elaboración propia, empleando Inkscape.

En el ejemplo del péndulo doble, cuando se hizo el modelo con mecánica de Newton, se consideraron las fuerzas de tensión que restringen el movimiento de las dos cuentas, como parte del modelo. Al plantear el modelo con mecánica lagrangiana no formaron parte del procedimiento. Si se desea tratar con este

tipo de fuerzas, se plantea el modelo como un problema de optimización con restricciones. Suponga que se pide encontrar las ecuaciones de movimiento y fuerzas de tensión, en una máquina de Atwood, ver figura 18. Olvídese por un momento de las fuerzas de tensión y se plantean las restricciones del problema. Debe suponerse que las masas $m_1\ y\ m_2$ se encuentran unidas por una polea sin inercia ni fricción. Y ambas masas se encuentran atadas por una cuerda. Si se considera lo suficientemente rígida para evitar elongaciones significativas en ella se puede decir que la longitud de la cuerda es constante, la suma de ambas distancias $x_1\ y\ x_2$ será una constante:

$$x_1 + x_2 = l \qquad \text{[Ec. 2.267]}$$

Se procede a identificar la restricción como:

$$0 = g(x_1, x_2, \dot{x}_1, \dot{x}_2, t) = x_1 + x_2 - l \qquad \text{[Ec. 2.268]}$$

$$0 = g(x_1, x_2, \ddot{x}_1, \ddot{x}_2, t) = x_1 + x_2 - l \qquad \text{[Ec. 2.269]}$$

Se plantea el lagrangiano del sistema:

$$L(x_1, x_2, t) = \frac{1}{2}m_1\dot{x}_1^2 + \frac{1}{2}m_2\dot{x}_2^2 - m_1 g x_1 - m_2 g x_2$$

[Ec. 2.270]

Pero debido que se trata de un problema con restricciones, se hará uso de los multiplicadores de Lagrange para plantear un integrando alternativo, con el fin de eliminar las restricciones:

$$\tilde{L}(x_1, x_2, t) = \frac{1}{2} m_1 \dot{x}_1^2 + \frac{1}{2} m_2 \dot{x}_2^2 - m_1 \mathrm{g} x_1 - m_2 \mathrm{g} x_2 + \sum_{k=1}^{N} \lambda_k(t)\, \mathrm{g}_k(x_1, x_2, \ddot{x}_1, \ddot{x}_2, t)$$

[Ec. 2.271]

Ya que solo hay una restricción:

$$\tilde{L}(x_1, x_2, t) = \frac{1}{2} m_1 \dot{x}_1^2 + \frac{1}{2} m_2 \dot{x}_2^2 - m_1 \mathrm{g} x_1 - m_2 \mathrm{g} x_2 + \lambda(x_1, x_2, \ddot{x}_1, \ddot{x}_2, t)$$

[Ec. 2.272]

Se plantean las ecuaciones de movimiento para Ĺ:

$$\frac{\partial \tilde{L}}{\partial x_1} - \frac{d}{dt}\left(\frac{\partial \tilde{L}}{\partial \dot{x}_1}\right) = 0$$

[Ec. 2.273]

Y

$$\frac{\partial \tilde{L}}{\partial x_2} - \frac{d}{dt}\left(\frac{\partial \tilde{L}}{\partial \dot{x}_2}\right) = 0$$

[Ec. 2.274]

Para x_1 se tiene que:

$$m_1 \mathrm{g} - m_1 \dot{x}_1 + \lambda = 0 \qquad \text{[Ec. 2.275]}$$

Para x_2 se tiene que:

$$m_2 \mathrm{g} - m_2 \dot{x}_2 + \lambda = 0 \qquad \text{[Ec. 2.276]}$$

Al combinar ambas ecuaciones se obtiene:

$$m_1 \mathrm{g} - m_1 \dot{x}_1 - m_2 \mathrm{g} - m_2 \dot{x}_2 = 0$$

[Ec. 2.277]

La ecuación 266 implica:

$$\dot{x}_1 + \dot{x}_2 = 0 \qquad \text{[Ec. 2.278]}$$

$$\dot{x}_1 + \dot{x}_2 = 0 \qquad \text{[Ec. 2.279]}$$

Por lo que:

$$m_1 \mathrm{g} - m_1 \dot{x}_1 - m_2 \mathrm{g} - m_2 \dot{x}_1 = 0$$

[Ec. 2.280]

Resolviendo así la ecuación para $\ddot{\mathrm{x}}_2$:

$$\dot{x}_1 = \frac{m_2 - m_1}{m_1 + m_2} \, \mathrm{g} \qquad \text{[Ec. 2.281]}$$

de forma similar para x_2:

$$\dot{x}_1 = \frac{m_1 - m_2}{m_1 + m_2} \, \mathrm{g} \qquad \text{[Ec. 2.282]}$$

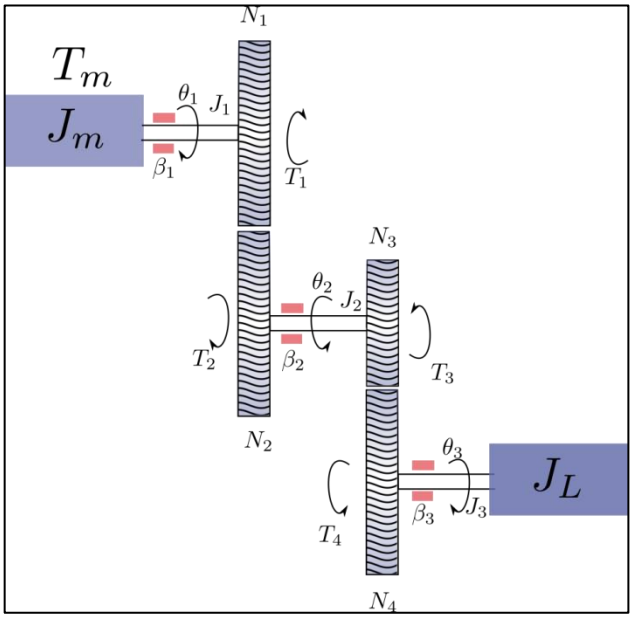
El multiplicador Λ tiene una interpretación física. Es el único valor ligado a restricciones en las ecuaciones de movimiento. Además, se aprecia que las dimensionales del multiplicador de Lagrange son de fuerza. Entonces dichos multiplicadores juegan el rol de fuerzas ligadas a las restricciones, por tanto, para encontrar la tensión se sustituye Λ por T en cualquiera de las dos ecuaciones y se despeja:

$$T = -m_1 g + m_1 \dot{x}_1 = -m_2 g + m_2 \dot{x}_2$$

[Ec. 2.283]

- Tren de N engranajes. En la figura 19 se presenta un tren de engranajes, plantear el juego de ecuaciones, utilizando mecánica de Newton, y luego resolverlas para encontrar una ecuación de movimiento resulta muy tedioso.

Figura 19. **Tren de engranajes**



Fuente: elaboración propia, empleando Inkscape.

Utilizando un modelo que usa mecánica lagrangiana, se simplifica el cálculo de las ecuaciones de movimiento para el tren de engranajes . Si se desean encontrar los valores de las torsiones de restricción T_1, T_2, T_3 y T_4 se usan los multiplicadores de Lagrange. Si se desea obviarlos, simplemente no se da el enfoque de problema de optimización con restricciones. En este ejemplo,

no se desea encontrar dichas torsiones de restricción, por tanto, se obvian los multiplicadores y cuidadosamente se plantea el lagrangiano, y demás magnitudes importantes, teniendo en cuenta las restricciones.

Debe quedar claro que se plantea el problema con los multiplicadores si se desea conocer el valor de las torsiones o fuerzas de restricción, de lo contrario se plantean las ecuaciones y se realizan sustituciones en base a las restricciones, antes de escribir la ecuación de Euler-Lagrange. Ya sea que interese o no conocer las expresiones que corresponden a cada una de las torsiones o fuerzas de restricción, es recomendable escribirlas para tenerlas presentes y poder plantear un modelo adecuado. Para el caso del tren de engranajes de la figura 19, solamente se tienen dos restricciones:

$$\frac{N_1}{N_2}\theta_1 = \theta_2$$

[Ec. 2.284]

$$\frac{N_3}{N_4}\theta_2 = \theta_3$$

[Ec. 2.285]

Con estas igualdades presentes, se procede a plantear el lagrangiano del tren de engranajes, para luego escribir la ecuación de movimiento de Lagrange. El sistema no presenta resortes torsionales u otro tipo de almacenamiento de energía potencial por tanto $V = 0$. Esto implica que el lagrangiano solo esté dado por la energía cinética:

$$L = T - V = T = \frac{1}{2}J_m\dot{\theta}_1^2 + \frac{1}{2}J_1\dot{\theta}_1^2 + \frac{1}{2}J_2\dot{\theta}_2^2 + \frac{1}{2}J_3\dot{\theta}_3^2 + \frac{1}{2}J_L\dot{\theta}_3^2$$

[Ec. 2.286]

Antes de escribir la ecuación de Lagrange, hay que observar que hay torsiones disipativas y torsiones externas. Se debe de tener en cuenta que las torsiones externas no disipan potencia del sistema, al contrario, se la inyectan. Entonces se pueden plantear como potencias disipadas, pero con signo contrario en el sistema. La potencia Pm entregada por la torsión τm, es equivalente a $Pm = \tau m\theta_1$. Una vez hechas estas aclaraciones, se plantea la potencia disipada del sistema.

$$P = \frac{1}{2}\beta_1\dot{\theta}_1^2 + \frac{1}{2}\beta_2\dot{\theta}_2^2 + \frac{1}{2}\beta_3\dot{\theta}_3^2 - \tau_m\dot{\theta}_1$$

[Ec. 2.287]

Si se quiere encontrar la ecuación de movimiento de la carga, se escribe el lagrangiano y la potencia en términos de θ_3. Si se desea encontrar la ecuación de movimiento del eje del motor se expresa todo en términos de θ_1. En este ejemplo se hará para el eje del motor.

$$L = \frac{1}{2}J_m\dot{\theta}_1^2 + \frac{1}{2}J_1\dot{\theta}_1^2 + \frac{1}{2}J_2\left(\frac{N_1}{N_2}\dot{\theta}_1\right) + \frac{1}{2}J_3\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\dot{\theta}_1\right)^2 + \frac{1}{2}J_L\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\dot{\theta}_1\right)^2$$

[Ec. 2.288]

$$P = \frac{1}{2}\beta_1\dot{\theta}_1^2 + \frac{1}{2}\beta_2\left(\frac{N_1}{N_2}\dot{\theta}_1\right)^2 + \frac{1}{2}\beta_3\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\dot{\theta}_1\right)^2 - \tau_m\dot{\theta}_1$$

[Ec. 2.289]

Considerando la potencia disipada, las ecuaciones de Euler-Lagrange quedan:

$$0 = \frac{d}{dt}\left(\frac{\partial L}{\partial\dot{\theta}_1}\right) - \frac{\partial L}{\partial\dot{\theta}_1} + \frac{\partial P}{\partial\dot{\theta}_1}$$

[Ec. 2.290]

$$0 = J_m\ddot{\theta}_1 + J_1\ddot{\theta}_1 + J_2\left(\frac{N_1}{N_2}\right)^2\ddot{\theta}_1 + J_3\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\right)^2\ddot{\theta}_1 + J_L\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\right)^2\ddot{\theta}_1 + \beta_1\dot{\theta}_1 + \beta_2\left(\frac{N_1}{N_2}\right)^2\dot{\theta}_1 + \beta_3\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\right)^2\dot{\theta}_1 - \tau_m$$

[Ec. 2.291]

$$0 = J_{eq}\ddot{\theta}_1 + \beta_{eq}\dot{\theta}_1 - \tau_m$$

[Ec. 2.292]

Donde:

$$J_{eq} = J_m + J_1 + J_2\left(\frac{N_1}{N_2}\right)^2 + J_3\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\right)^2 + J_L\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\right)^2$$

[Ec. 2.293]

$$\beta_{eq} = \beta_1 + \beta_2\left(\frac{N_1}{N_2}\dot{\theta}_1\right)^2 + \beta_3\left(\frac{N_1}{N_2}\frac{N_3}{N_4}\dot{\theta}_1\right)^2$$

[Ec. 2.294]

Como conclusión, en un tren de engranajes las constantes son transferibles de un punto del tren a otro, simplemente escalando por la relación de dientes adecuada.

2.1.5. Circuitos eléctricos

Los circuitos eléctricos encuentran sus bases en la teoría electromagnética, en la mecánica cuántica y en la teoría de semiconductores, y realiza un modelo real que involucra muchas variables. La ventaja que algunas de ellas pueden obviarse en ciertos casos. Por ejemplo, cuando se modela un transistor en un amplificador de audio, no se toma en cuenta su comportamiento cuántico, simplemente se considera una función que caracteriza su comportamiento para determinados voltajes, corrientes y tiempos. Este

transistor se puede simplificar más aún su modelo al transformarlo en una serie de elementos compuestos únicamente por dos terminales con un único voltaje entre ellas y una única corriente atravesando de una terminal a otra. A estos elementos se les conoce como elementos discretos, y a esta serie de abstracciones para un fenómeno que involucra circuitos eléctricos, se le conoce, por lo general como modelo de parámetros concentrados, conocido en inglés como *lumped matter discipline*.

Esta serie de abstracciones forman un método que simplifica el análisis de un sistema eléctrico real distribuido en el espacio, mediante la creación de elementos discretos que aproximan el comportamiento real. La ventaja de usar un modelo de este tipo es que matemáticamente, se puede reducir las ecuaciones en derivadas parciales espaciales y temporales en ecuaciones diferenciales ordinarias solo en el tiempo, con un número conjunto finito de parámetros. En estos modelos se asume que los componentes se encuentran conectados por conductores ideales.

Es posible hacer este tipo de simplificaciones siempre y cuando las señales operen en una escala temporal mucho mayor a los tiempos de propagación de las ondas electromagnéticas a través de los elementos discretos. A continuación se hace mención de algunos elementos discretos:

Resistencia. Es un elemento, que disipa energía, en general transforma energía eléctrica en térmica. Obedece la ley de Ohm, la cual enuncia que en un conductor, la corriente que lo atraviesa es proporcional a la diferencia de potencial de sus extremos. Dicho de otra forma, la densidad de corriente J y el campo eléctrico E se relacionan por una constante σ llamada conductividad del conductor:

$$J = \sigma \boldsymbol{E} \qquad \text{[Ec. 2.295]}$$

Y dicha constante depende del tipo de conductor. En modelos de parámetros condensados por lo general se asocia una constante R llamada resistencia, a estos elementos. Y por la ley de Ohm, se dice que:

$$v = Ri \qquad \text{[Ec. 2.296]}$$

Donde i es la corriente que lo atraviesa y v es el voltaje entre sus terminales. La potencia que disipa un resistor, en estos modelos está dada por la ecuación:

$$P = \frac{1}{2} Ri \qquad \text{[Ec. 2.297]}$$

Figura 20. **Representación de una resistencia**

Fuente: elaboración propia, empleando Inkscape.

Capacitores. Los capacitores son almacenes de energía potencial, dado que almacenan carga. Se componen de dos conductores separados por un dieléctrico. Se define la capacitancia, como la razón de cambio de la carga almacenada por el capacitor, en relación al voltaje.

$$C = \frac{dq}{dv} \qquad \text{[Ec. 2.298]}$$

Por lo general se asume, en los modelos de componentes condensados a C como una constante. Obteniendo $Cv = q,$ si quisiéramos asociar un equivalente mecánico sería un resorte. Además, al considerar la corriente $i,$ que atraviesa los elementos en un modelo de este tipo:

$$i = \frac{dq}{dt} \qquad \text{[Ec. 2.299]}$$

Se tienen algunas expresiones para relacionar, el voltaje entre las terminales, la carga y la corriente del capacitor:

$$v = \frac{1}{C}\int_{-\infty}^{t} idt \qquad \text{[Ec. 2.300]}$$

$$i = C\frac{dv}{dt} \qquad \text{[Ec. 2.301]}$$

La energía potencial almacenada por el capacitor, en términos de la carga está dada por

$$E = \frac{1}{2}vq = \frac{1}{2}\frac{q^2}{C} = \frac{1}{2}Cv^2 \qquad \text{[Ec. 2.302]}$$

Figura 21. **Representación de un capacitor**

Fuente: elaboración propia, empleando Inkscape.

- Inductores. Son elementos que se resisten al cambio de la corriente eléctrica que los atraviesa. Consisten en un conductor, (como un cable), embobinado de alguna forma en algún núcleo. Se define la inductancia L, como:

$$L = \frac{d\emptyset}{di} \qquad \text{[Ec. 2.303]}$$

Donde i es la corriente atravesando la bobina y $\emptyset$ el campo magnético atravesando el núcleo. Por lo general se toma a L como una constante, aunque no siempre es así porque los núcleos en la naturaleza presentan muchas no linealidades. Cualquier cambio en la corriente crea un campo magnético $\emptyset$, el cual induce un voltaje v en el inductor. De la ley de Faraday es posible escribir una expresión que modele dicho comportamiento y relacione el voltaje y la corriente.

$$v = \frac{d\emptyset}{dt} \qquad \text{[Ec. 3.304]}$$

Considerando que se tomó como lineal la relación del campo y la corriente (inductancia), se puede dar una aproximación para una relación del voltaje y la corriente.

$$v \cong L\frac{di}{dt} \qquad \text{[Ec. 2.305]}$$

Cuando la corriente eléctrica pasa a través de ellos, la energía es almacenada temporalmente como campo magnético. Y el campo magnético almacenado se reflejado como una corriente atravesando la bobina. Dicha corriente es vista como electrones en movimiento. Por tanto, se dice que almacena energía cinética:

$$E = \frac{1}{2}Li^2 \qquad \text{[Ec. 2.306]}$$

Figura 22. **Representación de un inductor**

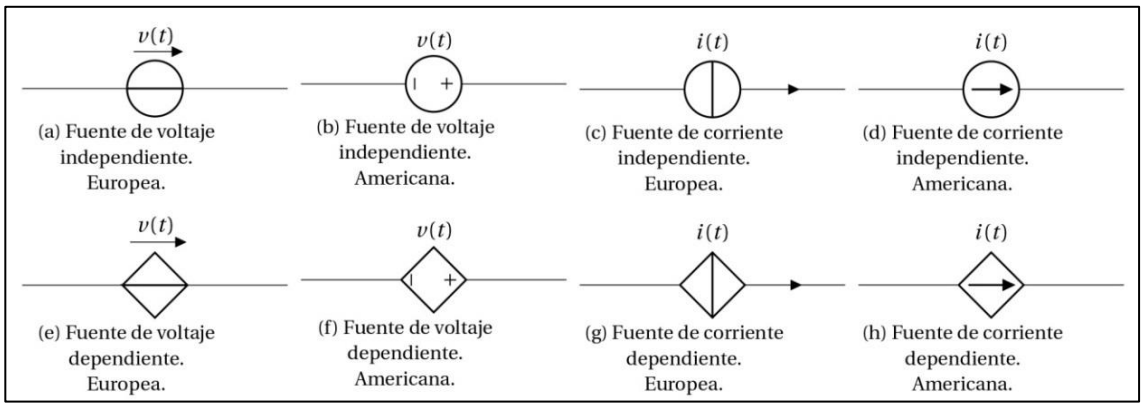
Fuente: elaboración propia, empleando Inkscape.

Fuentes de voltaje y corriente. Es un elemento discreto que sostiene un voltaje entre sus terminales, dado por una función en el tiempo u otras corrientes y voltajes. Una fuente de corriente es un elemento discreto que sostiene una corriente a través de él. Cuando alguna fuente depende de otros valores de corriente o voltaje dentro del circuito, se le llama una fuente dependiente y tiene una representación distinta de las fuentes que no depende de estos parámetros, llamadas fuentes independientes o externas. Las fuentes dependientes son utilizadas por lo general, para modelar elementos que tienen más de dos terminales. Muchos sensores, micrófonos o motores pueden tener representaciones discretas de fuentes de voltaje o corriente y formar parte de un modelo eléctrico. La potencia entregada por una fuente de corriente o voltaje está dada por:

$$P = vi \qquad \text{[Ec. 2.307]}$$

Donde *i* es la corriente atravesando la fuente y *v* es el voltaje entre sus terminales.

Figura 23. **Representación de fuentes**



Fuente: elaboración propia, empleando Inkscape.

Tabla XXI. **Definición de nodo**

Cualquier punto donde se conectan terminales de dos o más elementos, se conoce como nodo. Cualquier elemento o conductor ideal que conecte dos nodos se conoce como rama. Una malla es un camino cerrado a través de varias ramas de un circuito.

Fuente: elaboración propia.

Tabla XXII. **Teorema de leyes de Kirchhoff**

Ley de nodos. En cualquier nodo, la suma de las corrientes que entran en ese nodo es igual a la suma de las corrientes que salen. De forma equivalente, la suma de todas las corrientes que pasan por el nodo es igual a cero.

$$\sum_{k=1}^{n} I_k = I_1 + I_2 + I_3 \ldots + I_n = 0$$

[Ec. 2.308]

Ley de mallas. En una malla, la suma de todas las caídas de tensión es igual a la tensión total suministrada. De forma equivalente, la suma algebraica de las diferencias de potencial eléctrico en una malla es igual a cero.

$$\sum_{k=1}^{n} I_k = V_1 + V_2 + V_3 \ldots + V_n = 0$$

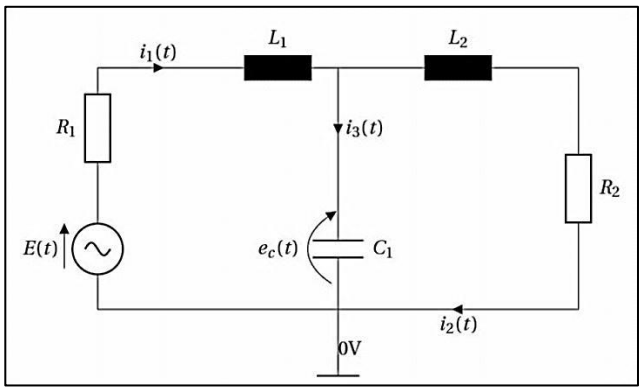
[Ec. 2.309]

Fuente: *Wikipedia.* https://es.wikipedia.org/wiki/*Leyes_de_Kirchhoff*. Consulta: octubre de 2014.

2.1.6. Ecuaciones de estado. Circuito RLC

El mejor ejemplo de las variables y ecuaciones de estado se encuentra en un circuito RLC. Se definen las variables de estado. Para este circuito la forma más práctica es utilizar las variables $i_1(t)$ para la corriente en $L_1, i_2(t)$ para la corriente en $L_2\ y\ ec\ (t)$ para la tensión del capacitor C_1.

Figura 24. **Circuito RLC de 2 mallas**



Fuente: elaboración propia, empleando Inkscape.

Un inductor es un almacén de energía cinética eléctrica y el capacitor es un almacén de energía potencial eléctrica. Al asignar $i_1(t), i_2(t)\ y\ ec\ (t)$ como las variables de estado, se tiene una descripción completa de los estados pasados, presentes y futuros. Utilizando las leyes de Kirchoff, se puede plantear el siguiente conjunto de ecuaciones:

$$R_1 i_1(t) + L_1 \frac{di_1(t)}{dt} + e_c(t) = E(t) \qquad \text{[Ec. 2.310]}$$

$$L_2 \frac{di_2(t)}{dt} + R_1 i_1(t) - e_c(t) = 0 \qquad \text{[Ec. 2.311]}$$

$$C \frac{de_c(t)}{dt} = i_1(t) - i_2(t) \qquad \text{[Ec. 2.312]}$$

Despejando la derivada de mayor orden, se obtienen las siguientes ecuaciones:

$$\frac{di_1(t)}{dt} = \frac{R_1}{L_1} i_1(t) - \frac{1}{L_1} e_c(t) + \frac{E(t)}{L_1} \qquad \text{[Ec. 2.313]}$$

$$\frac{di_2(t)}{dt} = \frac{R_2}{L_2} i_2(t) - \frac{1}{L_2} e_c(t) \qquad \text{[Ec. 2.314]}$$

$$\frac{de_c(t)}{dt} = \frac{1}{C_2} i_1(t) - \frac{1}{C_1} i_2(t) \qquad \text{[Ec. 2.315]}$$

Como se tiene un sistema lineal es posible expresar en forma matricial las ecuaciones de estado:

$$\begin{pmatrix} \frac{di_1(t)}{dt} \\ \frac{di_2(t)}{dt} \\ \frac{de_c(t)}{dt} \end{pmatrix} = \begin{pmatrix} -\frac{R_1}{L_1} & -\frac{1}{L_1} & 0 \\ 0 & \frac{R_2}{L_2 L_2} & 1 \\ \frac{1}{C_1} & -\frac{1}{C_1} & 0 \end{pmatrix} \begin{pmatrix} i_1(t) \\ i_2(t) \\ e_c(t) \end{pmatrix} + \begin{pmatrix} \frac{1}{L_1} \\ 0 \\ 0 \end{pmatrix} E(t)$$

[Ec. 2.316]

La matriz que acompaña al vector que contiene las variables de estado se conoce como matriz de estado. Dicho problema puede ser resuelto, usando mecánica lagrangiana. Siguiendo el mismo método de encontrar la energía cinética y potencial, las potencias disipadas y entregadas. Para resolverlo por el método de Lagrange, considérese a las corrientes i_1, i_2 e i_3 como los grados de libertad, por el momento. La energía potencial requiere que dichas corrientes se encuentren en términos de integrales, por lo que resulta conveniente cambiar los grados de libertad a q_1, q_2 y q_3, de donde

$$i_1 = \frac{dq_1}{dt} \dot{q}_1 \qquad \text{[Ec. 2.317]}$$

$$i_2 = \frac{dq_2}{dt} \dot{q}_2 \qquad \text{[Ec. 2.318]}$$

$$i_3 = \frac{dq_3}{dt}\dot{q}_3, \qquad \text{[Ec. 2.319]}$$

Donde q_{i} representa carga. La energía cinética del sistema T, se encuentra determinada por las corrientes que atraviesan las bobinas.

$$T = \frac{1}{2}L_1(\dot{q}_1)^2 + \frac{1}{2}L_2(\dot{q}_2)^2 \qquad \text{[Ec. 2.320]}$$

La energía potencial del sistema se encuentra determinada por la carga almacenada en el capacitor.

$$v = \frac{1}{2}\frac{q_3^2}{C} \qquad \text{[Ec. 2.321]}$$

La carga almacenada por el capacitor q_3 es equivalente a la carga inicial más la diferencia de las cargas q_1 y q_2. Otra forma de verlo es que el sistema restringidos por la ley de nodos $i_1 + i_2 + i_3 = 0$. Sea cual sea la forma de verlo lleva a la siguiente restricción.

$$\dot{q}_1 - \dot{q}_2 - \dot{q}_2 = 0 \qquad \text{[Ec. 2.322]}$$

$$q_1 - q_2 = q_3 + Q_0 \qquad \text{[Ec. 2.323]}$$

La potencia disipada del sistema está determinada por las resistencias, y la potencia entregada por la fuente

$$P = \frac{1}{2}R\dot{q}_1^2 + \frac{1}{2}R\dot{q}_2^2 - E\dot{q}_1 \qquad \text{[Ec. 2.324]}$$

El lagrangiano del sistema, en términos de q_1 y q_2, se encuentra dado por:

$$L = T - V = \frac{1}{2}L_1(\dot{q}_1)^2 + \frac{1}{2}L_2(\dot{q}_2)^2 - \frac{1}{2}\frac{q_3^2}{C}$$

[Ec. 2.325]

Entonces se procede a encontrar la ecuación de movimiento para q_1

$$0 = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_1}\right) - \frac{\partial L}{\partial q_1} + \frac{\partial P}{\partial \dot{q}_1}$$

[Ec. 2.326]

$$0 = L_1\ddot{q}_1 + R_1\dot{q}_1 + \frac{1}{C}(q_1 - q_2 - Q_0) - E$$

[Ec. 2.327]

y de la misma forma para q_2

$$0 = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_2}\right) - \frac{\partial L}{\partial q_2} + \frac{\partial P}{\partial \dot{q}_2}$$

[Ec. 2.328]

$$0 = L_2\ddot{q}_2 + R_2\dot{q}_2 + \frac{1}{C}(q_1 - q_2 - Q_0)$$

[Ec. 2.329]

El voltaje en el capacitor se hace notar como $ec = \frac{q_3}{C} = \frac{(q_1 - q_2 - Q_0)}{C}$ entonces, contando las ecuaciones de movimiento y la restricción se tiene el mismo sistema de ecuaciones.

$$0 = L_1\ddot{q}_1 + R_1\dot{q}_1 + e_c - E \quad \Longrightarrow \quad 0 = L_1 i_1 + R_1 i_1 + e_c - E$$

[Ec. 2.330]

$$0 = L_2\ddot{q}_2 + R_2\dot{q}_2 + e_c \quad \Longrightarrow \quad 0 = L_2 i_2 + R_2 i_2 + e_c$$

[Ec. 2.331]

$$0 = \dot{q}_1 - \dot{q}_2 + C\dot{e}_c \qquad \Longrightarrow \qquad 0 = i_1 - i_1 - C\dot{e}_c$$

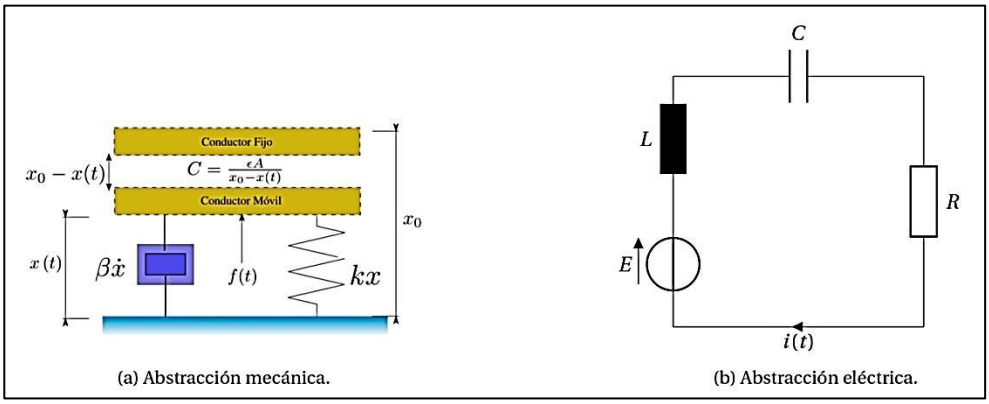
[Ec. 2.332]

Obteniendo así el mismo sistema de ecuaciones, con el método de mallas.

- Lagrangiano de un sistema electromecánico. Micrófono

La figura 25 muestra un micrófono capacitivo. Un micrófono capacitivo requiere alimentación de una batería o fuente externa. La señal de audio resultante es más fuerte que la señal resultante de un micrófono dinámico. Un capacitor formado por dos platos conductores, uno fijo y uno movible, hechos de materiales sumamente delgados funcionan como diafragma. Cuando recibe vibraciones del exterior, los platos se acercan y se alejan. Cuando los platos se acercan, la capacitancia incrementa y ocurre aparece una corriente de carga. Cuando los platos se alejan, la capacitancia decrece y ocurre una corriente de descarga. Las vibraciones recibidas pueden representarse por una fuerza $f\ (t\ )$ que se le aplica al plato movible.

Figura 25. **Abstracción eléctrica y mecánica de un micrófono**



Fuente: elaboración propia, empleando Inkscape.

Dado que es un capacitor de placas paralelas, la capacitancia se encuentra dada por:

$$C = \frac{A\in}{x_0 - x(t)} \qquad \text{[Ec. 2.333]}$$

Donde $\in$ es la constante dieléctrica del aire y A es el área de los platos. Para dicho sistema considérese dos grados de libertad q y x, siendo la carga y el desplazamiento del plato móvil respectivamente. La energía cinética del sistema está determinada por la bobina y la masa del plato en movimiento.

$$T = \frac{1}{2}L\dot{q}^2 + \frac{1}{2}m\dot{x}^2 \qquad \text{[Ec. 2.334]}$$

La energía potencial del sistema se encuentra dada por el resorte y el capacitor

$$v = \frac{1}{2C}q^2 + \frac{1}{2}kx^2 \qquad \text{[Ec. 2.335]}$$

Dando como resultado el lagrangiano:

$$L = T - V = \frac{1}{2}L\dot{q}^2 + \frac{1}{2}m\dot{x}^2 - \frac{1}{2C}q^2 + \frac{1}{2}kx^2 \qquad \text{[Ec. 2.336]}$$

$$= \frac{1}{2}L\dot{q}^2 + \frac{1}{2}m\dot{x}^2 - \frac{1}{2}\frac{(x_0 - x)}{\in A}q^2 - \frac{1}{2}kx^2 \qquad \text{[Ec. 2.337]}$$

La potencia disipada por el sistema se encuentra dada por la resistencia y el amortiguamiento; la potencia entregada está dada por la fuente. Por tanto, la potencia P del sistema está dada por:

$$P = \frac{1}{2}R\dot{q}^2 + \frac{1}{2}\beta\dot{x}^2 - E\dot{q} \qquad \text{[Ec. 2.338]}$$

A pesar de que el sistema es híbrido, la energía solo depende de las variables internas del sistema, por tanto, las ecuaciones de movimiento solo dependen de q y x. Como $f\ (t\ )$ es una fuerza lineal y solo afecta al sistema mecánico, para los dos grados de libertad x y q se tiene $\mathrm{Q}_{\chi} = f(t)\frac{\partial_{\chi}(x,q)}{\partial_{\chi}} = f(t) y\ Qq = f(t)\frac{\partial_{\chi}(x,q)}{\partial q} = 0$. Se tiene la ecuación de movimiento de Lagrange para x:

$$Q_x = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} + \frac{\partial P}{\partial \dot{x}}$$

[Ec. 2.339]

$$f(t) = m\dot{x} - \frac{q^2}{2_{\epsilon}A} + kx + \beta\dot{x}$$

[Ec. 2.340]

Para q se tiene:

$$Q_q = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} + \frac{\partial P}{\partial \dot{q}}$$

[Ec. 2.341]

$$0 = L\ddot{q} + R\dot{q} + \frac{1}{\epsilon A}(x_0 - x)q - E$$

[Ec. 2.342]

Obteniendo así las ecuaciones de movimiento y estado del sistema. Se puede obtener el mismo resultado separando el sistema mecánico y eléctrico, plantear los diagramas de cuerpo libre para el sistema mecánico y las mallas

para el sistema eléctrico, pero existe el riesgo de cometer el error de obviar la fuerza que existe por el campo eléctrico presente en el capacitor.

2.2. Bloques de sistemas continuos y sus propiedades

Es posible modelar sistemas continuos obstrayendo sus propiedadesusando gratos llamados bloques y en ello hacer mapas mentales del comportamiento del sistema.

2.2.1. Sistemas continuos

Intuitivamente, se sabe que una recta es continua de un punto a otro porque no hay perforaciones o baches en el trayecto entre ellos. Si se considera el tiempo como una recta, de tal forma que cada punto de la recta corresponde a un instante, se dice que el tiempo es continuo. Los sistemas que tratan señales de tiempo continuo, son sistemas continuos. Todos los ejemplos tratados en este capítulo son de sistemas continuos, ya que es posible idealizar al tiempo de cada sistema como una recta. Los sistemas continuos necesitan un modelo general para poder ser insertados en modelos de sistemas más complejos.

Tabla XXIII. **Definición de modelo de un sistema continuo**

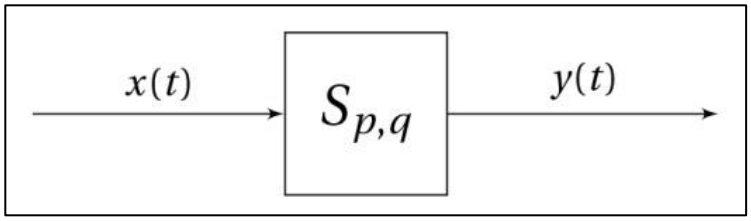
El modelo de un sistema continuo, es una función de la forma $S: X \rightarrow Y$. Donde X representa al conjunto de todas las funciones continuas de entrada $x: \mathbb{R} \rightarrow A$, para algún conjunto A. Y Y representa al conjunto de todas las funciones continuas de salida $y: \mathbb{R} \rightarrow B$. Para algún conjunto B.

Fuente: elaboración propia.

Un sistema continuo puede modelarse como una función, cuyos dominios son señales de entrada x y su imagen señales de salida y, ambas de tiempo continuo. El dominio de las señales lo forma el tiempo. De esta forma es posible simplificar la complejidad de un sistema continuo y reducirlo a transformaciones de entradas en salidas.

Una función modelando un sistema puede representarse como una caja o bloque, en la cual ingresan señales, este las transforma y entrega otras señales basadas en las entradas. En la figura 26 se muestra el bloque de una función S, que es un modelo de un sistema continuo. La función S puede depender también de parámetros propios del sistema p, q. En ese caso, se escribe el sistema como $Sp, q\ o\ S(p, q)$.

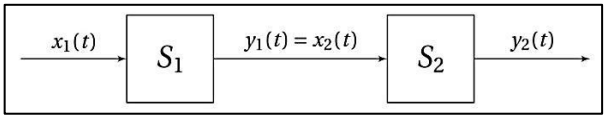
Figura 26. **Modelo de un sistema**



Fuente: elaboración propia, empleando Inkscape.

Un sistema continuo complejo puede modelarse como un conjunto de bloques de sistemas, o continuos, más simples conectados, formando un diagrama refleja del comportamiento del sistema. A estos modelos formados por varios bloques se les llama diagramas de bloques. Ofrecen versatilidad y simplicidad para modelar todo tipo de sistemas. Con ellos se describen las causas y efectos a través de todo el sistema. Cuando las salidas de un bloque, son las entradas de otro bloque, se dice que se encuentran en cascada o en serie.

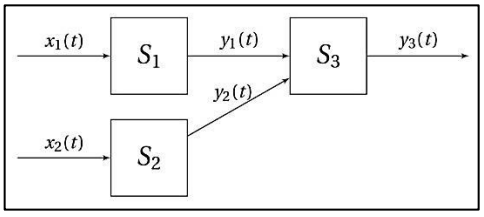
Figura 27. **Bloques en cascada o serie**



Fuente: elaboración propia, empleando Inkscape.

Es posible que las salidas de varios bloques, formen parte de las entradas de otro bloque, tal como se muestra en la figura 28.

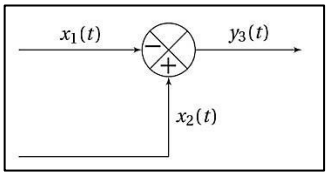
Figura 28. **Múltiples entradas**



Fuente: elaboración propia, empleando Inkscape.

La suma de dos señales forma uno de estos bloques. Recibe señales de entradas de varios bloques y devuelve una sola señal, y por lo común que resulta encontrarla se usa un bloque especial para ella.

Figura 29. **Bloque de suma**



Fuente: elaboración propia, empleando Inkscape.

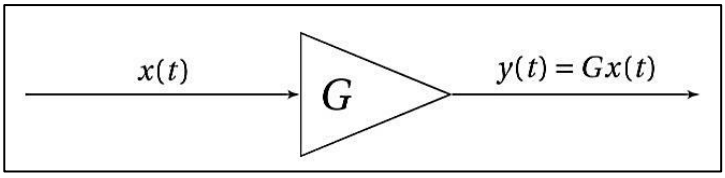
La figura 29 representa una función $S : (\mathbb{R} \to \mathbb{R})^2 \to (\mathbb{R} \to \mathbb{R})$ dada por

$$\forall t \epsilon \mathbb{R}, \forall\, x_1, x_2 \epsilon (\mathbb{R} \to \mathbb{R}), \quad \big(S(x_1, x_2)\big)(t) = -x_1(t) + x_2(t)$$

[Ec. 2.343]

La expresión anterior se lee para todo *t* que pertenezca a los reales, todos los x_1 y x_2 que pertenezcan al conjunto de señales continuas $\mathbb{R}^{\mathrm{R}}$, entonces *S*(*x*1,*x*2) es una señal que al valuar $x_1(t)$ y $x_2(t)$, tiene un valor de $(S\,(x_1, x_2))\,(t) = -x_1(t) + x_2(t)$. Otro modelo de un sistema sumamente importante, es el bloque de ganancia o escala que simplemente multiplica la señal de entrada por una constante.

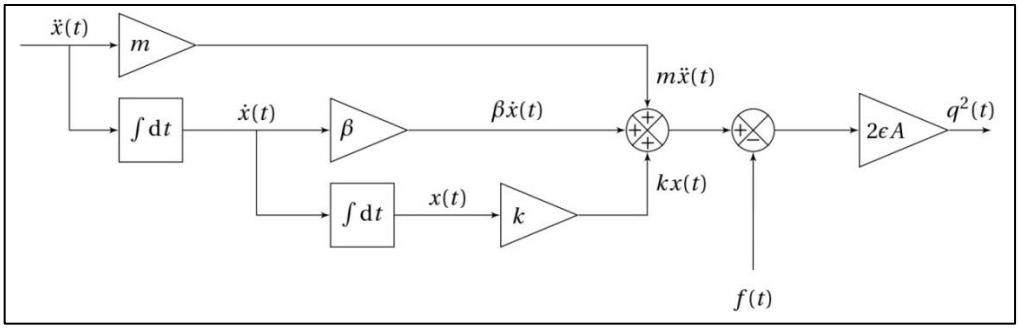
Figura 30. **Bloque de ganancia**



Fuente: elaboración propia, empleando Inkscape.

Representar sistemas físicos con diagramas de bloques hace más práctica su visualización:

Figura 31. **Diagrama de bloques del sistema mecánico de un micrófono**



Fuente: elaboración propia, empleando Inkscape.

2.2.2. Sistemas causales

Se dice que un sistema es causal cuando solo depende de su entrada actual y entradas anteriores.

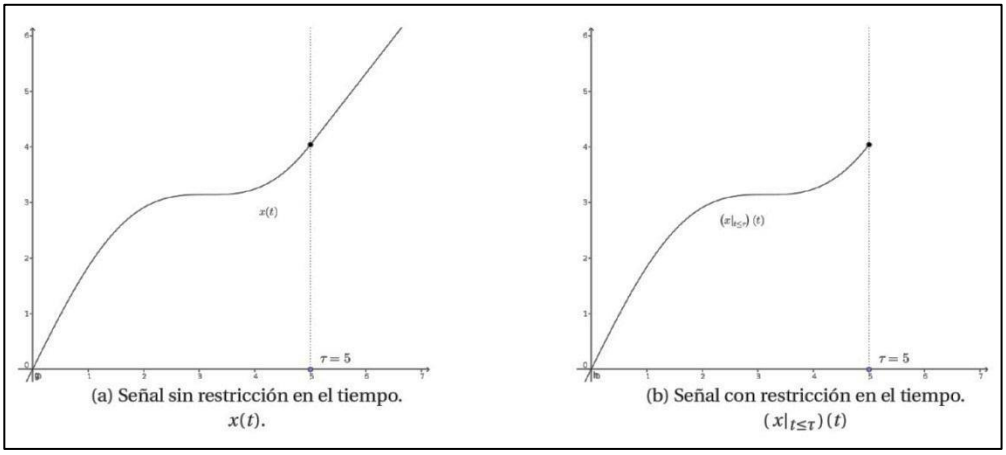
Tabla XXIV. **Definición de función restringida en el tiempo**

Considérese una señal continua $x: \mathbb{R} \rightarrow A$, para algún conjunto A. Sea $x\|t \leq T$, llamada función restringida en el tiempo, definida únicamente en $t \leq T$, y $x\|t \leq T(t) = x(t)$ para donde sí está definida.

Fuente: LEE, Edward y SEESHIA, Sanjit. *Introduction to embedded systems.* p. 120.

La definición anterior enuncia que si $x(t)$ son los valores de una señal, $(x|t \leq T)$ (t) representa el valor actual y valores anteriores de la señal.

Figura 32. **Restricción en el tiempo**



Fuente: elaboración propia, empleando Inkscape.

Ya definida una restricción en el tiempo, es posible dar una definición formal para un sistema causal.

Tabla XXV. **Definición de sistema causal**

Considérese un sistema continuo $S : X \to Y$, donde X es el conjunto A^{R} y Y es el conjunto B^{R}. Para un par de conjuntos A y B. S es un sistema causal si para todo $x_1, x_2 \in X \; y \tau \in \mathbb{R}$ $$x_1\|_{t\leq\tau}(t) = x_2\|_{t\leq\tau}(t) \Longrightarrow S(x_1)\|_{t\leq\tau} = S(x_2)\|_{t\leq\tau}$$ [Ec. 2.344]

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 120.

La definición anterior explica que un sistema es causal si para cualquier pareja de señales de entrada x_1 y x_2 idénticas hasta el instante $\boldsymbol{\tau}$, el sistema S entrega salidas idénticas hasta el instante $\boldsymbol{\tau}$. Se puede aplicar a una señal la misma restricción en el tiempo, pero sin contar el instante $\boldsymbol{\tau}$, esto con el fin de definir un sistema estrictamente causal.

Tabla XXVI. **Definición de sistema estrictamente causal**

Considérese un sistema continuo $S: X \to Y$, donde X es el conjunto A^{R} y Y es el conjunto B^{R}. Para un par de conjuntos A y B. S es un sistema estrictamente causal si para todo $x_1, x_2 \in X$ y $\tau \in \mathbb{R}$ $$x_1\|_{t\leq\tau}(t) = x_2\|_{t\leq\tau}(t) \Longrightarrow S(x_1)\|_{t\leq\tau} = S(x_2)\|_{t\leq\tau}$$ [Ec. 2.345]

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 88.

Es evidente que un sistema estrictamente causal también es un sistema causal. Una integral es un sistema estrictamente causal. Una suma no es estrictamente causal, pero es causal. Los sistemas estrictamente causales son útiles en la construcción de sistemas con *feedback*.

2.2.3. Sistemas sin memoria

Intuitivamente, un sistema tiene memoria si su salida depende de las entradas actuales sino de las pasadas o futuras, en caso de que este no sea causal.

Tabla XXVII. **Definición de sistema sin memoria**

Considérese un sistema continuo en el tiempo $S: X \to T$, donde $X = A^{\mathbb{R}}$ y $Y = B^{\mathbb{R}}$, para conjuntos A y B. Un sistema no tiene memoria si existe una función $f: A \to B$ tal que para $\boldsymbol{x} \in X$: $$\big(S(\boldsymbol{x})\big)(t) = f\ (\boldsymbol{x}(t))$$ [Ec. 2.346] Todo, dicho de otra forma, la salida del sistema $(S(\boldsymbol{x}))\ (t\ )$ en un tiempo t depende solo de la entrada $\boldsymbol{x}(t\ )$ en el tiempo t.

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 88.

Una suma es un sistema sin memoria, una integral no es un sistema sin memoria.

2.2.4. Sistemas lineales

La linealidad es una garantía sobre el comportamiento de un sistema. Asegura que la salida, para una determinada señal, conservará su forma, a pesar de que la entrada presente atenuaciones, escalamientos o cambios de referencia.

Por ejemplo, al hablar por teléfono, la persona que escucha del otro lado de la línea puede entender el mensaje independientemente si la persona que habla grita o susurra. Esto se debe a que tanto el micrófono como la bocina son lineales para determinados intervalos en los valores de la señal de audio. De esta forma, independientemente de la escala el audio conserva la forma de onda.

Tabla XXVIII. **Definición de linealidad**

Un sistema $S: X \rightarrow Y$, donde X y Y son conjuntos de señales continuas, es lineal si satisface la propiedad de superposición: $\forall x_1, x_2 \epsilon X \wedge \quad \forall a, b \epsilon\, \mathbb{R}, \qquad S(ax_1 + bx_2) = aS(x_1) + bS(x_2)$ [Ec. 2.347]

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 88.

.

2.2.5. Sistemas invariantes en el tiempo

Para definir invariante, primero es necesario definir un sistema que recibe el nombre de retraso.

Tabla XXIX. **Definición de retraso**

Sea $D\tau$: $X \rightarrow Y$ un sistema, donde X y Y son conjuntos de señales continuas en el tiempo, el cual se define como: $$\forall x \epsilon X \wedge \quad \forall t \epsilon \mathbb{R}, \quad \big(D_\tau(x)\big)(t) = x(t-\tau)$$ [Ec. 2.348] donde $\boldsymbol{\tau}$ es el parámetro de retraso en el sistema.

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 91.

Una vez definido un retraso, es posible definir un sistema invariante en el tiempo.

Tabla XXX. **Definición de sistema invariante en el tiempo**

Un sistema S: $X \rightarrow Y$ es invariante en el tiempo si: $$\forall x \epsilon X \wedge \quad \forall \tau \epsilon \mathbb{R}, \quad \big(D_\tau(x)\big) = D_\tau(S(x))$$ [Ec. 2.349]

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 92.

2.2.6. Estabilidad

Un sistema se dice de entrada acotada y salida acotada estable (entrada-salida y salida-acotada estable o solamente estable), si la señal de salida se encuentra acotada para todas las señales de entrada acotadas.

Tabla XXXI. **Definición de estabilidad**

La entrada $w(t)$ es acotada si existe un número real $A < \infty$ tal que $A, \forall t \epsilon \mathbb{R}$. La salida $v(t)$ es acotada si existe un número real $B < \infty$ tal que $B, \forall t \epsilon \mathbb{R}$. El sistema es estable si para cualquier entrada acotada por algún número real A, existe algún número real B que acota la salida.

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 94.

2.2.7. Sistemas LIT

Un sistema continuo LIT es aquel que cumple con ser lineal e invariante en el tiempo. Por ejemplo, una derivada es LIT. Considérese una derivada como un sistema cuya entrada es $x(t)$ y su salida es $y(t)$. Se tiene:

$$y(t) = \frac{dx(t)}{dt} \Rightarrow y(t-a) = \frac{d\,(x(t-a))}{dt}$$

Si se tienen dos señales continuas $x_1(t)\ y\ x_2(t)$, se obtiene:

$$\mathrm{y_1(t)} = \frac{dx_1\ \ (t)}{dt}, y_2(t) = \frac{\mathrm{dx_2(t)}}{dt} \Rightarrow\ y_1(t)\ + y_2(t) = \frac{\mathrm{d}}{dt}\ (x_1(t)\ + x_2(t))$$

El objetivo principal del modelado de dinámicas físicas es buscar un modelo LIT siempre que sea posible. Si existe alguna aproximación razonable que transforme un modelo no LIT en un modelo LIT, vale la pena hacerla. Una ventaja de los sistemas continuos es que posiblita el análisis de su comportamiento usando la transformada de Laplace.

- Transformada de Laplace

La transformada de Laplace es una herramienta matemática, cuyo principal objetivo es la resolución de ecuaciones diferenciales ordinarias. El dominio de la transformada $X(s)$ se conoce como dominio de la frecuencia o dominio de Laplace. Este nombre se debe a que dependiendo de los valores de s la salida de un sistema puede oscilar. En la tabla XXXIII se pueden ver algunas transformadas de funciones frecuentes en el análisis de sistemas.

Tabla XXXII. **Definición de transformada de Laplace**

Sea $x(t)$ una señal continua de la forma $x: \mathbb{R} \to \mathbb{R}$, solamente definida para $t > 0$, que satisface

$$\int_0^\infty |x(t)e^{-\sigma t}|dt < \infty,$$

[Ec. 2.350]

Para algún σ real, la transformada de Laplace $X(S)$ de $x(t)$, se define como:

$$\mathcal{L}\{x(t)\} = X(s) = \int_0^\infty x(t)\, e^{-st} dt$$

[Ec. 2.351]

Fuente: KUO, Benjamin. *Automatic control systems*. p. 77.

Tabla XXXIII. **Transformadas de algunas funciones**

Dominio del tiempo	Dominio de la frecuencia
t	$\frac{1}{s^2}$
$\delta(t)$	1
$u(t)$	$\frac{1}{s}$
$\sin(\omega t)$	$\frac{\omega}{s^2+\omega^2}$
$\cos(\omega t)$	$\frac{s}{s^2+\omega^2}$
e^{at}	$\frac{1}{s-a}$

Fuente: elaboración propia.

La transformada de Laplace tiene una serie de propiedades matemáticas muy interesantes. Algunas de ellas se pueden apreciar en la tabla XXXIV. La propiedad más útil que ofrece esta transformada en el análisis y modelo de sistemas LIT, es la capacidad de convertir un sistema de ecuaciones diferenciales en expresiones algebraicas, ofreciendo así un método algebraico para la resolución de ciertas ecuaciones diferenciales.

Tabla XXXIV. **Algunas propiedades de la transformada de Laplace**

Dominio del tiempo	Dominio de la frecuencia
$f(t)$	$F(s)$
$kf(t)$	$kF(s)$
$f_1(t) + f_2(t)$	$F_1(s) + F_2(s)$
$e^{-at}f(t)$	$F(s+a)$
$f(at)$	$\frac{1}{a}F\left(\frac{s}{a}\right)$
$f(t-T)$	$e^{-sT}F(s)$
$f_1(t)f_2(t)$	$F_1(s) * F_2(s)$
$f_1(t) * f_2(t)$	$F_1(s)F_2(s)$
$\frac{\mathrm{d}f(t)}{\mathrm{d}t}$	$sF(s) - f(0_-)$
$\frac{\mathrm{d}^n f(t)}{\mathrm{d}t^n}$	$s^n F(s) - \sum_{k=1}^{n} s^{n-k} f^{n-k}(0_-)$
$\int_{0_-}^{t} f(\tau)\mathrm{d}\tau$	$\frac{F(s)}{s}$
$f(\infty)$	$\lim_{s\to 0} sF(s)$
$f(0_+)$	$\lim_{s\to \infty} sF(s)$

Fuente: elaboración propia.

El tren de engranajes modelado, puede verse como un sistema con una torsión externa como señal de entrada y ya sea un ángulo o una velocidad angular como señal de salida. El comportamiento de dicho sistema se encuentra descrito por la ecuación 2.291. La solución de esa ecuación diferencial resulta en la salida de dicho sistema. No es tan fácil ver como la solución de una ecuación diferencial puede representar una transformación para una señal de entrada. Para este fin considérese una ecuación diferencial lineal de primer orden.

$$\frac{dy(t)}{dt} + P(t)y(t) = x(t)$$

[Ec. 2.352]

Es posible encontrar una solución analítica para dicho sistema. Multiplicar ambos lados de la igualdad por un factor $e\int p(t)dt$ y así poder agrupar ambos términos del lado izquierdo como una derivada de un producto $\frac{d}{dt}(e\int p(t)dt\ y\ (t))$, dicha operación simplifica la ecuación diferencial a:

$$\frac{d}{dt}\left(e^{\int P(t)dt}y(t)\right) = e^{\int P(t)dt}x(t)$$

[Ec. 2.353]

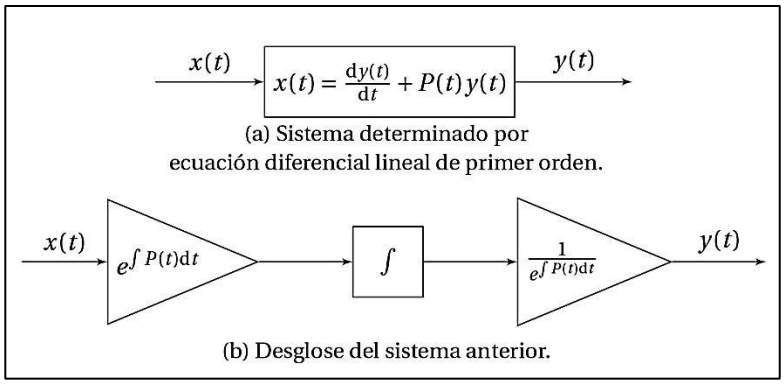
$$\Longrightarrow \quad y(t) = \frac{1}{e^{\int P(t)dt}y(t)}\int e^{\int P(t)dt}x(t)dt$$

[Ec. 2.354]

Ahora se encuentra más claro como la solución de una ecuación diferencial representa una transformación para una función de entrada $x(t\ )$. Es más, se puede ver que la transformación no solo depende de la función de entrada sino también de parámetros internos del sistema, tal como lo hace $P(t\ )$

para esta ecuación. La figura 33 ilustra el procedimiento anterior con diagramas de bloque.

Figura 33. **Bloques de un sistema continuo LIT**



Fuente: elaboración propia, empleando Inkscape.

Tabla XXXV. **Teorema de transformada de derivadas**

Si $f, \frac{df}{dt}, \ldots, \frac{\mathrm{dn^1f}}{dtn^{-1}}$ son funciones continuas en el intervalo $[0, \infty)$ y son de orden exponencial y si $\frac{dnf}{dtn}$ es continua por tramos en $[0, \infty)$, entonces:

$$\mathcal{L}\left\{\frac{d^n f(t)}{dt^n}\right\} = s^n F(s) - s^{n-1} f(0) - s^{n-2} \left.\frac{df(t)}{dt}\right|_{t=0} - \cdots - \left.\frac{d^n f(t)}{dt^n}\right|_{t=0}$$

[Ec. 2.355]

Fuente: elaboración propia.

Si se tiene una ecuación diferencial para una función continua $y(t)$ igualada a una función $x(t)$:

$$a_n = \frac{d^n y(t)}{dt^n} + a_{n-1} = \frac{d^{n-1} y(t)}{dt^{n-1}} + \cdots + a_0 y(t) = x(t)$$

[Ec. 2.356]

La transformada de Laplace de la ecuación anterior es una combinación lineal de transformadas por las propiedades de linealidad:

$$a_n\,\mathcal{L}\left\{\frac{d^n y(t)}{dt^n}\right\} + a_{n-1}\,\mathcal{L}\left\{\frac{d^{n-1} y(t)}{dt^{n-1}}\right\} + \cdots + a_0\mathcal{L}\{y(t)\} = \mathcal{L}\{x(t)\}$$

[Ec. 2.357]

Del teorema 2.6, la ecuación anterior se convierte en:

$$X(s) = a_n\left(s^n\mathrm{Y(s)} - s^{n-1}\mathrm{y(0)} - \cdots y^{n-1}(0)\right) + a_{n-1}\left(s^{n-1}\mathrm{Y(s)} + s^{n-2}y(0) - \cdots y^{n-1}(0)\right) + \cdots + a_0 Y(s)$$

[Ec. 2.58]

$$X(s) = a_n(s^n\mathrm{Y(s)} - a_n\left(s^{n-1}\mathrm{y(0)} - \cdots y^{n-1}(0)\right) + a_{n-1}\left(s^{n-1}\mathrm{Y(s)}\right)$$

$$-a_{n-1}\left(s^{n-2}y(0) - \cdots y^{n-2}(0)\right) + \cdots + a_0 Y(s)$$

[Ec. 2.359]

$$X(s) = \left(a_n s^n\mathrm{Y(s)} + a_{n-1}s^{n-1}\mathrm{Y(s)}\right) + \mathrm{G(s)}$$

[Ec. 2.360]

$$X(s) = (a_n s^n + a_{n-1}s^{n-1} + \cdots + a_o)\mathrm{Y(s)} + \mathrm{G(s)}$$

[Ec. 2.361]

$$X(s) = P(s)Y(s) - G(s) \Longrightarrow$$

[Ec. 2.362]

$$Y(s) = \frac{X(s)}{P(s)} + \frac{G(s)}{H(s)}$$

[Ec. 2.363]

Se observa como una ecuación diferencial se convierte en una expresión algebraica. Donde, *P*(*s*) es un polinomio de la forma $ansn + an - 1sn^{-1} + \cdots + a_0$

y $G(s)$ un polinomio de la forma $bn-1sn-1+bn-2sn-2+\cdots+b_0$. También se puede ver la multiplicación de funciones en el dominio de la frecuencia.

Otra propiedad interesante en la transformada de Laplace es que el equivalente de una multiplicación en el dominio de la frecuencia, es una convolución en el dominio del tiempo y viceversa. La convolución es una operación simétrica entre dos funciones cuya representación en el dominio de la frecuencia está dada por una multiplicación entre las dos transformadas.

Tabla XXXVI. **Definición de convolución en tiempo continuo**

Sean dos funciones continuas $h: \mathbb{R} \to \mathbb{R}$ y $f: \mathbb{R} \to \mathbb{R}$, definidas para $t \geq 0$. La convolución es una operación conmutativa entre dos funciones, f y h, tal que:

$$(f * h)(t) = \int_0^t f(\tau)h(t-\tau)\,d\tau = (h * f)(t) = \int_0^t h(\tau)\,f(t-\tau)d\tau$$

[Ec. 2.364]

Fuente: elaboración propia.

Con base en la definición de convolución y transformada de Laplace es posible demostrar en la tabla XXXVII.

Tabla XXXVII. **Teorema de convolución real**

Sean $x: \mathbb{R} \to \mathbb{R}$ y $h : \mathbb{R} \to \mathbb{R}$ dos funciones continuas, definidas para $t \geq 0$. Sean $X(s)$ y $H(s)$ sus respectivas transformadas de Laplace.

$$\mathcal{L}\{(f * h)(t)\} = F(s)H(s)$$

[Ec. 2.365]

Fuente: elaboración propia.

Al aplicar la transformada de Laplace a una ecuación diferencial se convierte en una ecuación algebraica que se puede resolver para $Y(s)$, es decir, $\mathrm{Y}(\mathrm{s}) = \frac{X(s)}{P(s)} + \frac{\mathrm{G}(\mathrm{s})}{P(s)}$. Ahora, como $\mathcal{L}\{y(t)\} = Y(s)$, si de alguna forma se pudiera regresar al dominio del tiempo se obtendría la solución $y(t)$ que se busca. En otras palabras, es necesario definir una transformada inversa $\mathcal{L} - {}^{1}\{Y(s)\}$, para hallar la función $y(t)$.

Tabla XXXVIII. **Definición de transformada inversa de Laplace**

Si $F(s)$ representa la transformada de Laplace de una función $f(t)$, es decir $\mathcal{L}\{f(t)\} = F(s)$, se dice que *f* (*t*) es la transformada inversa de Laplace de $F(s)$ y se escribe como $f(t) = \mathcal{L}^{-1}\{F(s)\}$.

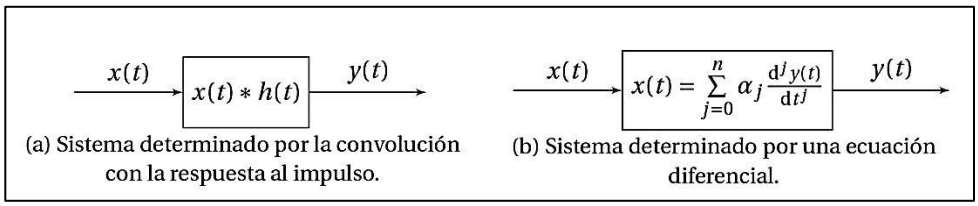
Fuente: elaboración propia.

Suponga que la salida de un sistema, cuando no recibe una señal de entrada (es decir $x(t) = 0, \forall t \geq 0$), se encuentra en reposo ($yn(t) = 0$, para $n = 0, 1, \ldots, n-1$), implica que el polinomio $G(s)$ de la ecuación 2.362 es igual a cero, porque todos sus coeficientes son cero. El tren de engranajes es ejemplo de ello, si no hay una torsión en la entrada de esta, la caja no se moverá, sino que permanecerá en el reposo. Entonces la ecuación 2.362 se reduce a:

$$Y(s) = H(s)X(s) \qquad \text{[Ec. 2.366]}$$

Donde $\mathrm{H}(\mathrm{s}) = \frac{1}{P(s)}$, $H(s)$ es conocida como una función de transferencia del sistema. De la ecuación 362, se infiere que un sistema formado por ecuaciones diferenciales ordinarias, como en el caso del tren de engranajes , puede ser visto como la convolución de la entrada con una función especial, $h(t)$.

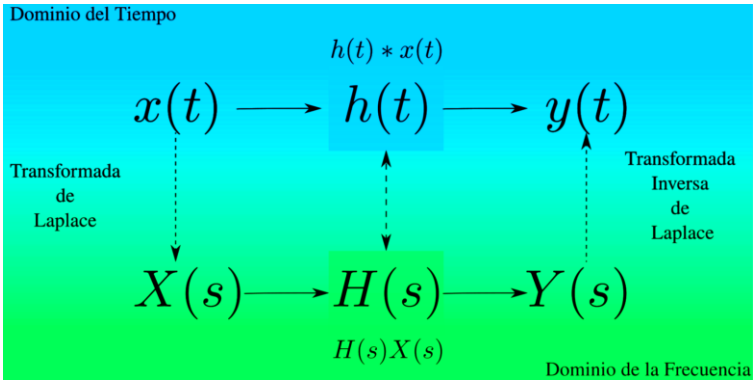
Figura 34. **Bloques equivalentes de un sistema continuo**



Fuente: elaboración propia, empleando Inkscape.

La representación en el dominio de la frecuencia de $h(t)$, está dada por $H(s)$ quién recibe el nombre de función de transferencia del sistema. Es posible determinar la salida de un sistema para una entrada $x(t)$, obteniendo la transformada de Laplace, $X(s)$, de dicha entrada; luego multiplicarla por la representación en frecuencia de $h(t)$ y obtener la inversa de dicho producto. La función *h*(*t*) recibe el nombre de respuesta al impulso, ya que un impulso $\delta(t)$ en el dominio de la frecuencia es una constante de valor 1 y por tanto la salida de un sistema a una señal de impulso será siempre la representación en el tiempo de la función de transferencia del sistema.

Figura 35. **Resolución de problemas**



Fuente: elaboración propia, empleando Inkscape.

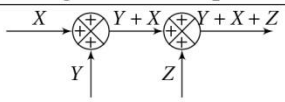
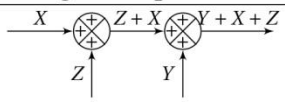
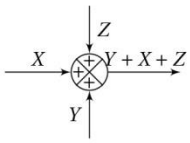
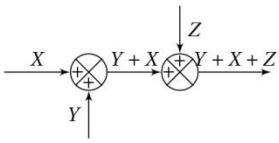
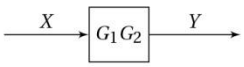
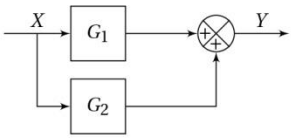
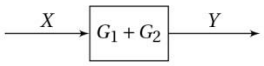
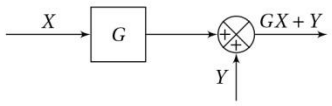
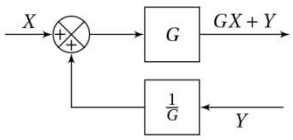
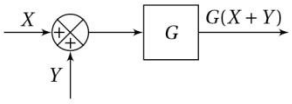
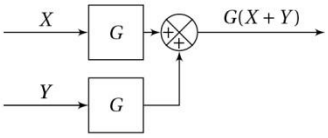
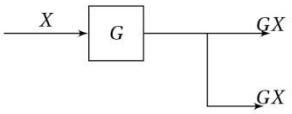
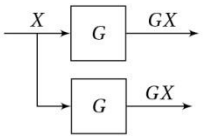
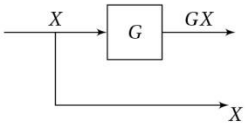
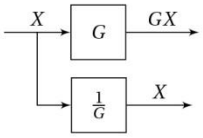
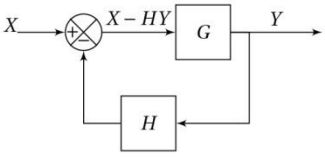
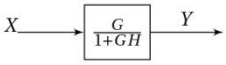
- Álgebra para bloques lineales

El álgebra de bloques es una herramienta útil para tratar diagramas de bloques formados por sistemas lineales. Suponga que se tiene una señal $x(t)$ y dos sistemas lineales e invariantes el tiempo, $S_1\ y\ S_2$, que pueden modelarse por ecuaciones diferenciales, y $H_1(s)\ y\ H_2(s)$ representan sus funciones de transferencia respectivas, si inicialmente ambos sistemas se encuentran en reposo, y $x(t)$ es la entrada de S_1, es fácil ver que si la salida de S_1 es la entrada de S_2, es decir ambos sistemas se encuentran en cascada; es posible sustituir ambos bloques por uno solo cuya función de transferencia está dada por $H_1(s)H_2(s)$. De igual forma si $x(t)$ es la entrada de ambos bloques y las señales de salida de S_1 y S_2 se suman, es decir se encuentran en paralelo relacionados por una suma; es posible cambiar ambos bloques por uno solo cuya función de transferencia este dada $H_1(s) + H_2(s)$.

Sin ninguno de los dos se encuentra en reposo es posible sustituirlos por un bloque más general. De esa cuenta puede verse que una configuración compleja de sistemas, sistemas en cascada, en paralelo y algunos otros que son consecuencia de estos, pueden reducirse a un solo bloque.

En la tabla XXXIX se pueden apreciar algunas sustituciones de bloques lineales útiles con una función de transferencia equivalente a la compuesta por varios bloques.

Tabla XXXIX. **Transformación de bloques**

Descripción	Diagrama de Bloques	Diagrama equivalente
Conmutación para la suma.		
Distribución para la suma.		
Conmutación para la multiplicación		
Distributiva para la multiplicación.		
Movimiento a la izquierda de una suma.		
Movimiento a la derecha de una suma.		
Movimiento a la izquierda de un producto.		
Movimiento a la derecha de un producto.		
Lazo Cerrado a Lazo Abierto.		

Fuente: elaboración propia.

El último bloque de la tabla XXXIX es conocido bloque de lazo cerrado, *feedback* o retroalimentación. Dicho bloque puede ser sustituido por uno solo debido a la propiedad de linealidad de las funciones de transferencia. Como puede verse en dicho lazo cerrado si se tiene una entrada X, la salida Y está dada por

$$Y = G(X - HY) \qquad \text{[Ec. 2.367]}$$

Por la propiedad, de linealidad se puede resolver dicha ecuación para Y, solo haciendo uso de sumas y multiplicaciones,

$$Y = GX - HY \qquad \text{[Ec. 2.368]}$$

$$Y + GHY = GX - HY \qquad \text{[Ec. 2.369]}$$

$$Y(1 + GH) = GX \qquad \text{[Ec. 2.370]}$$

$$Y(1 + GH)(1 + GH)^{-1} = GX(1 + GH)^{-1} \qquad \text{[Ec. 2.371]}$$

$$Y = \left(\frac{G}{1 + GH}\right)X \qquad \text{[Ec. 2.372]}$$

De esa forma se puede concluir que toda vez se tenga un lazo cerrado o *feedback* se puede cambiar por un solo bloque, llamado lazo abierto una función de transferencia equivalente a la del *feedback*.

2.3. Análisis numérico y simulación por computadora. SCILAB

El principal objetivo de un modelo es brindar información de un sistema, por lo que es necesario extraerla y planearla de forma comprensible. En esta tarea es de mucha ayuda una computadora y programas computación científica.

Existen dos clases de este tipo de software. La primera constituye los Sistemas de Computación Algebraica (*Computer Algebra Systems, CAS*) los cuales realizan computación simbólica, similares a la computación hecha a mano por matemáticos o científicos, Sage o Maxima son algunos ejemplos. La segunda clase, la constituyen los Paquetes de Análisis Numérico (*Numerical Analysis Software, NAS*), que se encuentran basados en un conjunto de algoritmos encargados de obtener una solución numérica a problemas que envuelven análisis matemático. Como: SCILAB, GNU Octave, Matlab.

Ambos tipos ofrecen ventajas y desventajas. El problema de usar un CAS para obtener información es que limita el rango de problemas que se pueden simular o analizar, dado que usa matemática simbólica y muchos modelos resultan en ecuaciones cuya solución no tiene una forma cerrada, dejando como única salida la utilización de métodos numéricos para hallar la solución de dicho problema. Un NAS ofrece mayor versatilidad en la resolución de problemas sin una solución cerrada. La desventaja es que se obtiene una solución con errores de redondeo. Pero en algo que imita la realidad es un precio accesible a pagar.

2.3.1. SCILAB

Es un lenguaje interpretado con objetos de tipo dinámico. SCILAB corre en las principales plataformas: Unix/Linux, MicrosoftWindows yMacOSX. Compilar

el código fuente también es posible. SCILAB puede ser usado como un *scripting language* para probar algoritmos o par realizar computación numérica. La sintaxis de SCILAB es muy simple, está basada en el uso de matrices que son los objetos fundamentales en el análisis numérico científico. SCILAB se encuentra diseñado para computación científica y provee fácil acceso a una gran cantidad de librerías dedicadas a áreas como el álgebra lineal, la integración numérica y la optimización. Existen numerosas *toolboxes* que incrementan la funcionalidad de SCILAB. Una muy importante es Xcos, utilizada como herramienta gráfica de simulación y modelado.

La herramienta de simulación escogida fue, SCILAB, por las siguientes razones:

- Lo compone un conjunto muy completo de paquetes para análisis numérico.

- Sintaxis similar a la utilizada en a Matlab y GNU Octave, cinvirtiéndola en una herramienta familiar para muchos usuarios.

- Utiliza el uso comercial y no comercial de forma libre. Totalmente gratuita y sin costo.

- Cuenta con la herramienta XCos, utilizada para la construcción de sistemas por diagramas de bloques. Para aquellos familiarizados con Simulink o con la Suite de Simulación y Control de LabVIEW, verán esta herramienta muy parecida.

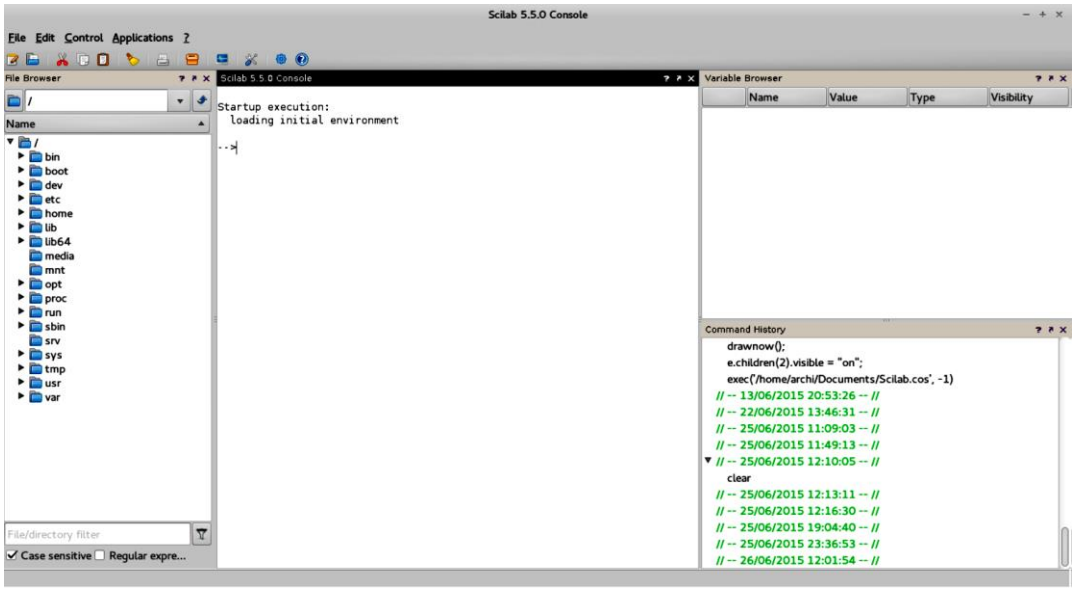
- Es *open-source*; por lo que es posible ver y modificar el código fuente de las librerías y paquetes. A pesar que lo componen miles de líneas y

modificar el código posiblemente esté fuera del alcance de un usuario común, es parte de la filosofía *Open-Source*, que permite el desarrollo de nuevas herramientas con base en ella. Actualmente la versión de la licencia de uso es CECILL 2.1.

Primeros pasos:

La versión a utilizar en este texto es la 5.5.0. Al abrir SCILAB lo primero que aparece es el entorno.

Figura 36. **Entorno de SCILAB**

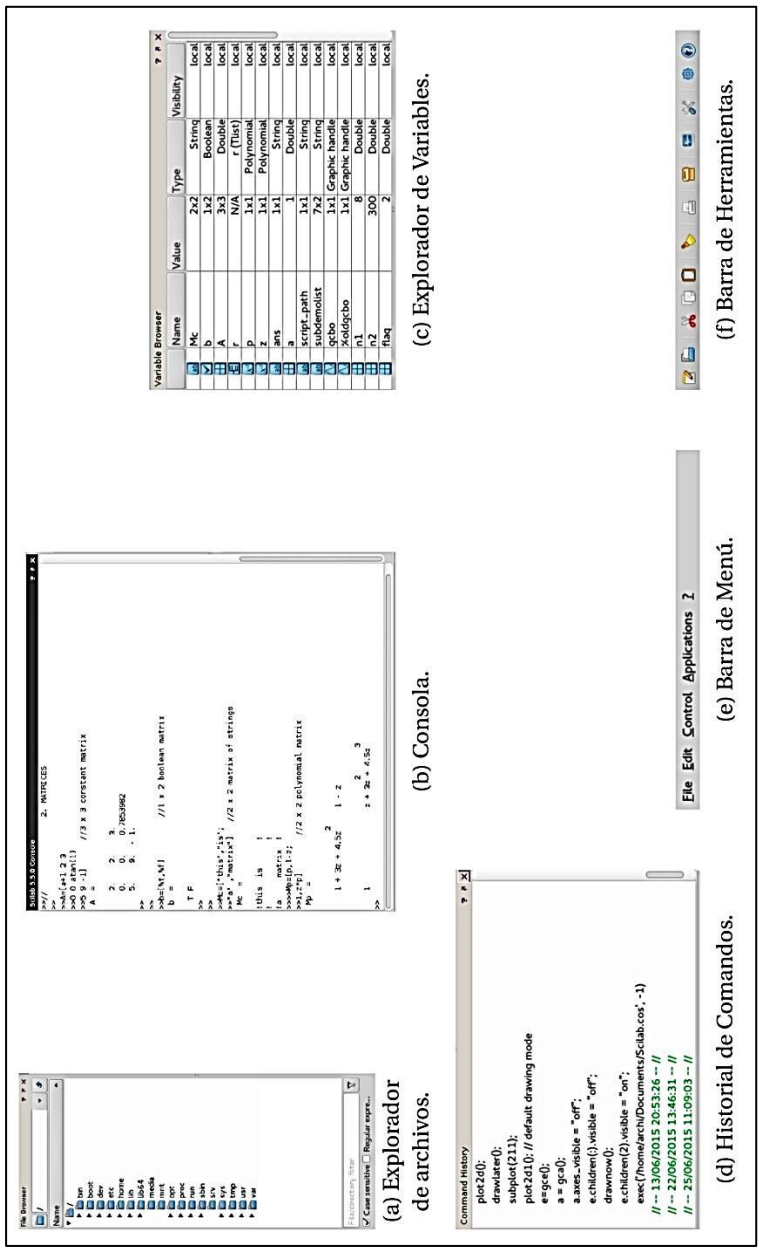


Fuente: elaboración propia, empleando SCILAB.

El entorno con la configuración por defecto se compone por la Consola, el Explorador de archivos, el Explorador de variables y el Historial de comandos.

Las barras de herramientas cambian al enfocarse en los diferentes componentes del entorno.

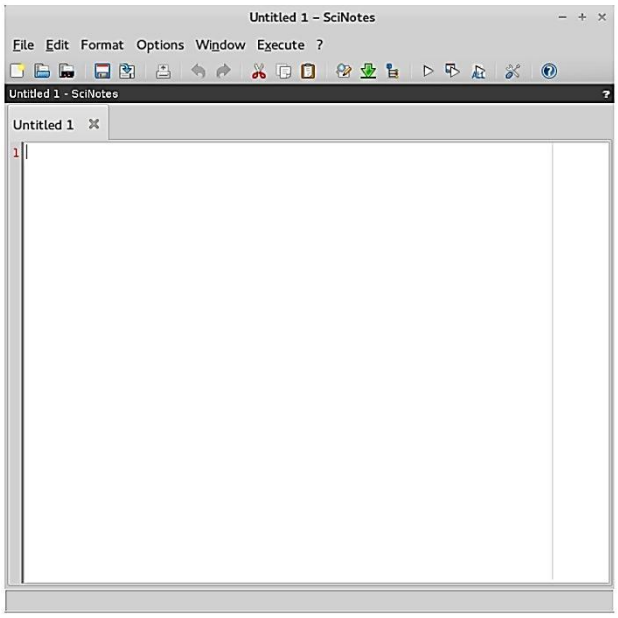
Figura 37. **Partes que componen el entorno de SCILAB**



Fuente: elaboración propia, empleando SCILAB.

Por lo general, SCILAB se usa mediante *scripting* y la consola solo es un apoyo para la ejecución de programas y algoritmos. Posee un editor de texto propio llamado Scinotes para el desarrollo de *scripts*, aunque pueden utilizarse otros editores como Gedit. Abrir el editor es muy sencillo, basta con presionar su ícono o escribir en la consola: →scinotes(). Al hacerlo aparecerá un entorno similar al de la figura 38.

Figura 38. **Scinotes**



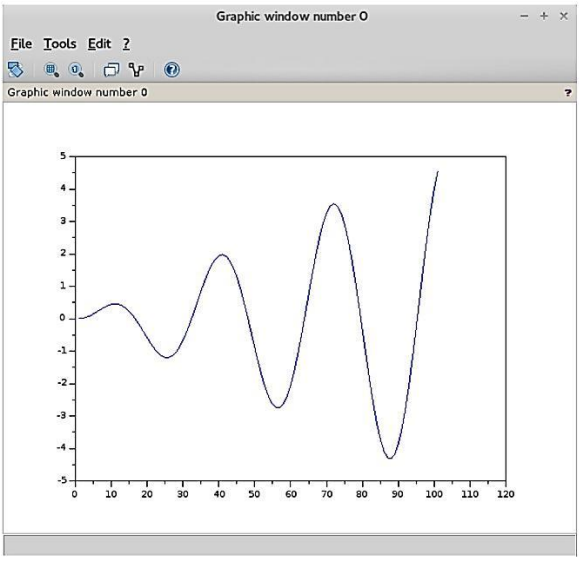
Fuente: elaboración propia, empleando SCILAB.

A continuación se presenta el ejemplo de una gráfica. Se procede a escribir el código 2.1 en Scinotes. Una vez finalizado debe guardarse el *script* en algún lugar del disco duro. Las extensiones '*.sce' o '*.sci' son las más comunes para almacenar este tipo de documentos. Luego se presiona el botón, llamado Execute e inmediatamente emergerá una ventana con una gráfica, como la mostrada en la figura 39.

Código 2.1:

```
1 // ////////////////////////////////////////
2 // Autor : Oscar Ram írez .
3 // Universidad de San Carlos de Guatemala
4 // ////////////////////////////////////////
5
6 // Declaración de parámetros
7 x =[0:.1:10] ';
8 A =0.5* x;
9 // Puntos a graficar
10 y=A.* sin (2*x);
11 // GRÁFICA
12 plot (y)
```

Figura 39. **Ventana de gráfica 1**



Fuente: elaboración propia, empleando SCILAB.

SCILAB tiene una sintaxis basada en matrices, muy similar a GNU Octave y Matlab, las cuales son los elementos básicos en el análisis numérico. Actualmente se ha convertido en un lenguaje que incluye soporte para la manipulación de diferentes tipos de objetos. En SCILAB, los objetos nunca se declaran ni se alojan en memoria explícitamente; tienen un tipo y tamaño dinámico y cambia de acuerdo a las operaciones y funciones aplicadas a ellos.

En el ejemplo x=[0:.1:10]' se declaró una matriz de 101£1, dicha matriz por ser solo de una dimensión se le llama vector. Dicho vector al ser declarado de esa forma, se incrementa de 1 en 1 desde el 0 hasta el 10. El apóstrofe indica transpuesta, por tanto transforma lo que antes era un vector fila de 1£100 en un vector columna de 100£1. Luego se declaró la multiplicación por un escalar de la matriz. El resultado arrojado por operador '*', depende del tipo de variables que se encuentre operando.

Por ejemplo para la sentencia A=0.5*x, la operación es un escalar por una matriz, y SCILAB realiza un escalamiento de todos los valores de la matriz por el valor del escalar. En la sentencia y=A.*sin(2*x) el operador '.*' realiza una operación de producto término a término, esto es posible porque ambos operandos son matrices de la misma dimensión, arrojando como resultado una matriz de la misma dimensión de los operandos pero sus entradas son resultado del producto de las entradas del primer operando multiplicadas por su análoga en el segundo operando. Cabe resaltar que la sentencia sin(2*x) genera una matriz del mismo tamaño de 2*x donde las entradas son el seno de cada entrada de la matriz 2*x, funciones sobre matrices devuelven matrices. En el código 2.2 se puede apreciar mejor el uso del producto término a término.

Tabla XL. **Código 2.2: Producto término a término**

```
!M=[1 , 2, 3; 4, 5, 6; 7, 8, 9]
M =
1.      2.      3.
4.      5.      6.
7.      8.      9.
!M.*M
ans =
1.      4.      9.
16.     25.     36.
49.     64.     81.
```

Fuente: elaboración propia, empleando SCILAB.

La sentencia plot(y) se encarga de imprimir en la ventana de gráfica a la matriz y=A.*sin(2*x), en este caso es interpretada como un vector, por ser una matriz 101£1, donde el eje de las abscisas corresponde a la posición y el eje de las ordenadas corresponde al valor de la matriz en dicha posición. El objeto principal de SCILAB son las matrices numéricas, aunque es posible encontrar otro tipo de objetos, como escalares (un escalar puede ser visto como una matriz de 1£1), cadenas, matrices booleanas o matrices polinomiales. Sus funciones y operaciones son matriciales, un ejemplo de ello es el *script* de la gráfica o la toma de la consola en el código 2.5.

Tabla XLI. **Código 2.3: Función de una matriz**

```
!sin(M* %pi)
ans =
10^( -14) *
0.0122465 - 0.0244929 0.0367394
- 0.0489859 0.0612323 - 0.0734788
0.0857253 - 0.0979717 0.1102182
```

Fuente: elaboración propia, empleando SCILAB.

El resultado de esta matriz, dado que todos son senos de un valor entero, debería de ser una matriz cero, pero cabe recordar que SCILAB no es un software de matemática simbólica sino de análisis numérico y es por eso que arroja una aproximación del resultado. De no tomar en cuenta este aspecto, los resultados podrían contener errores.

Por ejemplo, al comparar dos números que matemáticamente son iguales aunque se encuentren escritos de forma diferente, para SCILAB pueden ser diferentes, ya que acarrea errores de aproximación. La forma correcta de comparar dos números es comparar la distancia entre ellos contra una tolerancia determinada.

Tabla XLII. **Código 2.4: SCILAB es un software de análisis numérico**

```
!1. -0.25 -0.25 -0.25 -0.25==1. -0.4*0.25
ans =
F
!abs ((1. -0.25 -0.25 -0.25 -0.25) -(1. -4*0.25) )< %eps
ans = T
```

Fuente: elaboración propia, empleando SCILAB.

SCILAB también cuenta con constantes propias, tal como %pi y %eps. las de SCILAB llevan como primer caracter el símbolo ‘%’ antes del nombre de la constante. En la tabla XLIII se pueden ver algunas de ellas y una breve descripción de su uso o valor. Cabe resaltar que SCILAB puede manejar complejos.

Tabla XLIII. **Constantes ya definidas por defecto**

Símbolo	Valor	Descripción
%pi	$\pi = 3{,}1415927\ldots$	Pi
%e	$e = 2{,}7182818\ldots$	Constante de Napier
%i	$i = \sqrt{-1}$	Unidad imaginaria
%inf	∞	Infinito, excepción numérica, no el concepto matemático.
%nan		Not a number, excepción numérica.
%eps	$2{,}220 * 10^{-16}$	Precisión de máquina.
%s	s	Variable polinomial.
%z	z	Variable polinomial.
%t, %T	true	Valor booleano.
%f, %F	false	Valor booleano.

Fuente: elaboración propia.

Existen varios operadores y símbolos en SCILAB, la mayoría orientados a operaciones con matrices, algunos depende, de la forma de usarse pueden realizar distintas acciones. Se pueden apreciar algunas descripciones y símbolos en la tabla XLIV.

Tabla XLIV. **Operadores y símbolos en SCILAB**

operador o Símbolo	**Descripción**
;	Fin de expresión o separador de columnas en una matriz.
’	Transpuesta conjugada o delimitador para cadenas.
.’	Transpuesta no conjugada.
[], []’	Declaración de matriz o vector, el apostrofe indica que se declara como columnas en caso de ser un vector. En caso de ser matriz devuelve la transpuesta de la declarada dentro de [].
()	Paréntesis usados para varios propósitos.
+,-	Suma o resta.
, .	Multiplicación dependiendo del tipo de los operandos y multiplicación término a término.
/, ./	División derecha y división derecha término a término .
\, .\	División izquierda y división izquierda término a término.
^ o **, .^	Potencia y potencia término a término.
.*.	Producto de Kronecker.
./. , .\.	División izquierda y derecha de Kronecker.
|	OR Lógico.
&	AND Lógico.
~	NOT Lógico.
==, >=, <=, >, <	Igual, mayor o igual, menor o igual, mayor que, menor que.
<>, ~=	No igual.

Fuente: elaboración propia.

Es posible trabajar en SCILAB con dos tipos de comandos. El primero de ellos son los *scripts* compuestos por una serie de comandos cuyo fin es automatizar la computación. Los valores devueltos por los comandos, en caso que al final de una sentencia no aparezca el caracter ';', en los *scripts* por lo general se muestran en la consola, pero las gráficas se muestran en la ventana de gráficas. El segundo son las funciones o macros los cuales son pequeños programas que interaccionan con el entorno a través de entradas y salidas. Una serie de *built-in functions* se presentan en la tabla XLV.

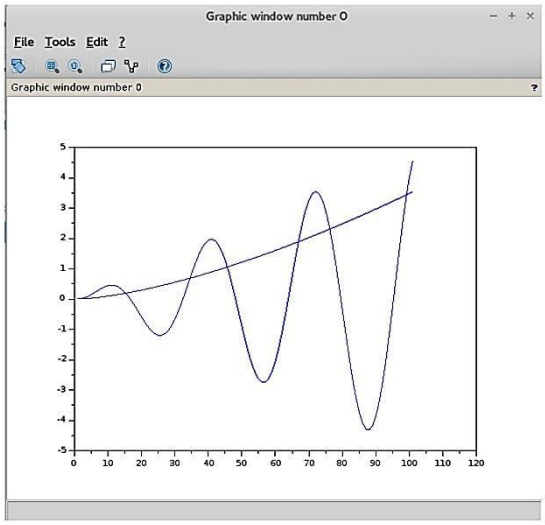
Tabla XLV. **Algunas *built-in functions* matemáticas**

Comando	Descripción
sin(), cos(), tan(), cotg()	Funciones trigonométricas.
asin(), acos(), atan()	Arco funciones.
sinh(), cosh(), tanh(), coth()	Funciones hiperbólicas.
asinh(), acosh(), atanh()	Funciones hiperbólicas inversas.
sqrt(), exp()	Raíz cuadrada y exponencial.
sum()	Suma de las entradas de una matriz.
min(), max()	Valor mínimo y máximo de las entradas de una matriz.
abs(), sign()	Norma de un número, y signo de un número real.
real(), imag()	Parte real e imaginaria de un número complejo.
phasemag()	Devuelve la fase y magnitud de un complejo para ambos resultados se iguala a una matriz de dos columnas por ejemplo, [phase, magn]=phasemag(z).

Fuente: elaboración propia.

En el código 2.1 si se cambia la sentencia y=A.*sin(2*x) a y=A.*sqrt(2*x), suponiendo que no se ha cerrado la ventana de Gráfica, y se ejecuta el programa de nuevo, aparecerán en la misma ventana de visualización ambas gráficas.

Figura 40. **Ventana de gráfica 2**



Fuente: elaboración propia, empleando SCILAB.

Se puede evitar este comportamiento en las gráficas colocando la sentencia clf antes de comenzar el *script*. Este comando hará que SCILAB limpie la ventana antes. Existen otros comandos que realizan acciones parecidas, *clear* remueve los objetos o variables del espacio de trabajo y libera la memoria; clc limpia la consola. Un comando que puede generar matrices similares a x en el ejemplo de la gráfica, es linspace. Dicho comando recibe 3 parámetros, el primero es la primera entrada del vector, el segundo parámetro es la última posición y el último es el número de entradas que tendrá el vector.

Tabla XLVI. **Código 2.5: *linspace***

```
!linspace (0 ,5 ,6)
ans =
0.        1.        2.        3.        4.        5.
!linspace (1 ,4 ,5)
ans =
1.        1.75      2.5       3.25      4.
!linspace ( -1. ,2. ,7)
ans =
- 1.      - 0.5     0.        0.5       1.        1.5       2.
```

Fuente: elaboración propia, empleando SCILAB.

El comando linspace es muy útil para generar vectores de la misma dimensión, evita el trabajo de calcular para cada vector la separación entre sus entradas. Otra ventaja al utilizar linspace es que genera un vector unif*or*memente espaciado al igual que al hacer la declaración con corchetes, pero linspace asegura que el fin y el inicio indicado sean parte del vector, cosa que no se encuentra asegurada para el fin al declarar con corchetes.

En el siguiente ejemplo se muestra la gráfica de una modulación por amplitud de una onda sinusoidal.

Tabla XLVII. **Código 2.6: gráfica de una modulación por amplitud**

```
1 // ////////////////////////////////////////
2 // Autor : Oscar Ram írez .
3 // Universidad de San Carlos de Guatemala
4 // ////////////////////////////////////////
5 clear , clc , clf;
6 // Frecuencia lineal portadora
7 f_port = 100;
8 // Frecuencia lineal modulada
9 f_mod = 10;
10 // Frecuencia angular portadora
11 w_port = 2* %pi* f_port ;
```

Continuación de la tabla XLVII.

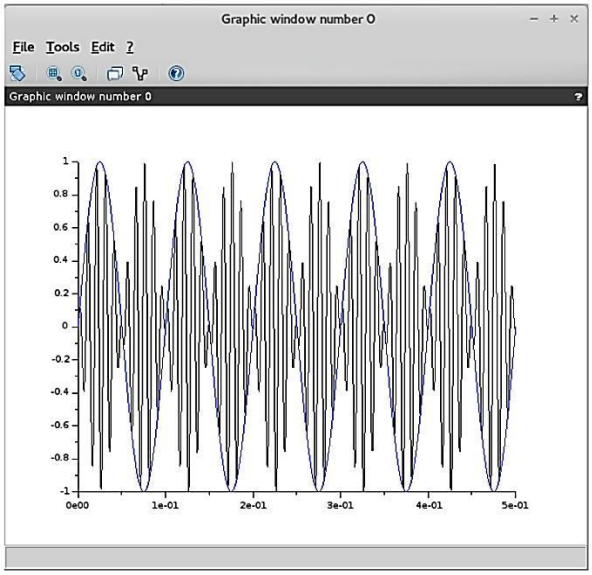
```
12 // Frecuencia angular modulada
13 w_mod = 2* %pi* f_mod ;
14 // Fase inicial
15 phi = %pi /4;
16 // Tiempo final
17 final = 5/ f_mod ;
18 // tiempo
19 t = linspace (0, final ,1000) ;
20 t = t'
21 // Moduladora
22 A = sin( w_mod *t);
23 // Portadora modulada
24 s = A.* sin( w_port *t + phi);
25 // Grá fica
26 plot2d (t ,[s,A] ,[1 ,2])
```

Fuente: elaboración propia, empleando SCILAB.

En este ejemplo se aseguró que tanto 0 como el valor de final se encuentren en la gráfica. El tiempo t es un vector unif*or*memente espaciado y posee 1 000 valores, comenzando en 0 y terminando en 5 periodos de la onda moduladora (la señal con la frecuencia baja). La matriz o vector A representa dicha señal. La matriz s, representa la portadora (la señal con la frecuencia alta), y la amplitud está dada por la moduladora (de ahí su nombre), por eso es que se efectúa el producto término a término. Véase que en esta ocasión se utilizó el comando plot2d, el cual ofrece más versatilidad a la hora de graficar.

En la figura 41, el eje de las abscisas ya no representa posiciones en el vector, sino el vector t. Se puede ver que sin necesidad de mandar a llamar dos veces a la gráfica sin limpiar, se pueden mostrar dos gráficas en la misma ventana. Da a cada un estilo diferente con el tercer parámetro [1,2]. El 1 representa un color negro, y el 2 uno azul.

Figura 41. **Gráfica de una modulación por amplitud**



Fuente: elaboración propia, empleando SCILAB.

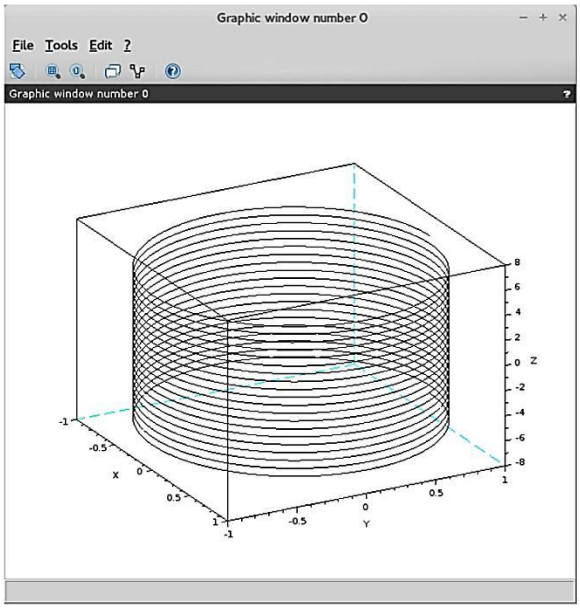
Esa es la razón por lo que se obtuvo la transpuesta de t, ya que si en lugar de colocar una matriz de $n \times 1$ *o* $1 \times n$, se coloca una matriz de $n \times m$ cada columna de dicha matriz representa un vector con valores para graficar en el eje de las ordenadas, $[s, A]$ es la matriz concatena, de los vectores columna s y A. Ambos de dimensión *n* =1, al igual que t, generando una matriz de $n \times 2$. Así como plot2d existen otros comandos útiles para gráficas, ejemplos de ellos son param3d1 y param2d1 útiles para gráficas de curvas paramétricas.

Tabla XLVIII. **Código 2.7: generación de una paramétrica**

```
!t= linspace ( -20* %pi , 20* %pi , 2000) ;
!param3d1 (sin(t), cos(t), t /10)
```

Fuente: elaboración propia, empleando SCILAB.

Figura 42. **Curvas paramétricas en tres dimensiones**



Fuente: elaboración propia, empleando SCILAB.

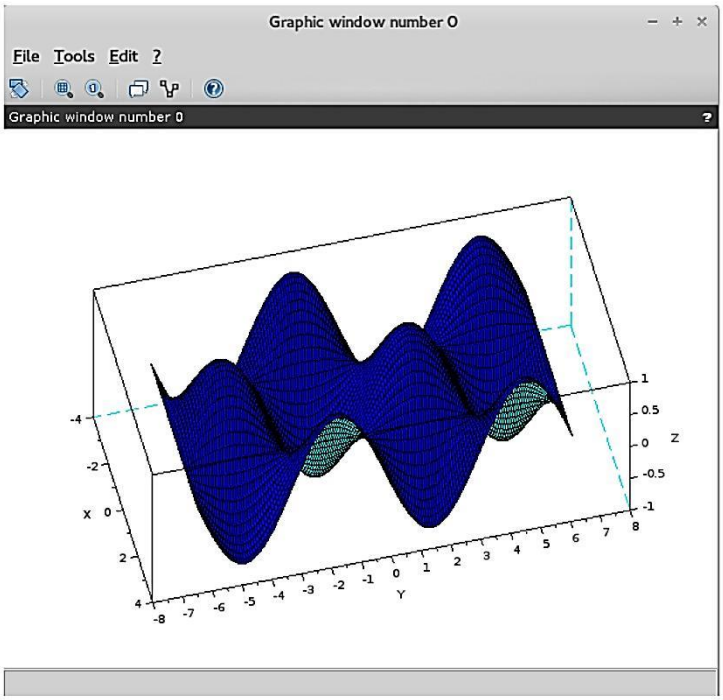
El comando plot3d es útil para generar gráficas de superficies en tres dimensiones, su uso es muy similar a plot2d. La diferencia es que recibe como primer parámetro un vector para el eje x, como segundo parámetro un vector para el eje *y* y un tercer parámetro una matriz para el eje z, esta matriz debe de ser de la dimensión de *x* por la dimensión de *y*, para poder asignar un punto en z a cada pareja *x y*. El siguiente ejemplo genera una gráfica para la superficie $z = cos(x)sin(y), = \pi, \leq x \leq \pi, -2\pi \leq 2\pi$

Tabla XLIX. **Código 2.8: generación de una paramétrica**

```
        !x= linspace (- %pi , %pi , 40);
        !y= linspace ( -2* %pi , 2* %pi , 160) ;
!plot3d (x,y, cos(x) '*sin(y))
```

Fuente: elaboración propia, empleando SCILAB.

Figura 43. **Gráfica de una superficie en tres dimensiones**



Fuente: elaboración propia, empleando SCILAB.

El comando cos(x)'*sin(x) genera la matriz $mi\,j = cos(xi\,)sin(y\,j\,)$ donde xi son entradas de un vector columna $x\,y$ $y\,j$ son entradas de un vector fila y. Existen muchas características y parámetros que incrementan la funcionalidad de los comandos mostrados hasta ahora. Es muy fácil acceder a ella basta con presionar la tecla F1, presionar el ícono o escribir en la consola. help();.

Un tipo de variable propio de SCILAB y útil en el análisis de sistemas continuos LIT son los polinomios. La mayoría de operaciones para matrices funcionan para los polinomios. Un polinomio puede ser definido en SCILAB usando el comando poly.

Tabla L. **Código 2.9: polinomios**

```
!p= poly ([ -5 ,5] , 's')
p =
2
- 25 + s
!q= poly ([ -5 ,5] , 's', 'c')
q =
- 5 + 5s
!p-q
ans =
2
- 20 - 5s + s
```

Fuente: elaboración propia, empleando SCILAB.

Es posible declarar un polinomio por sus raíces, tal como se hizo en el ejemplo para p, o por sus coeficientes, como q. Si se desea hacerlo por sus coeficientes es necesario agregar un parámetro más el caracter 'c', donde se le indica al comando que el vector representa coeficientes y no raíces.

SCILAB es versátil al manipular polinomios. Es posible realizar operaciones con polinomios por ejemplo sumas, restas y multiplicaciones. Es más, en SCILAB es posible declarar objetos racionales, compuestos por la división de dos polinomios; formar matrices con ellos y hasta obtener inversas. Cuando un objeto se compone de racionales, es posible obtener su polinomio numerador y su polinomio denominador, tal como se muestra en el código 2.10.

Tabla LI. **Código 2.10: objetos racionales**

```
!p*q
ans =
2 3
125 - 125 s - 5s + 5s
!r=p/q
```

Continuación de la tabla LI.

```
r =
2
- 25 + s
------
- 5 + 5s
!a=[1 ,r; 1 ,1]
a =
2
1 - 25 + s
- ------
1 - 5 + 5s
1 1
- -
1 1
!b=inv(a)
b =
2
- 5 + 5s 25 - s
---------- ----------
2 2
20 + 5s - s 20 + 5s - s
5 - 5s - 5 + 5s
---------- ----------
2 2
20 + 5s - s 20 + 5s - s
!b.num
ans =
2
- 5 + 5s 25 - s
5 - 5s - 5 + 5s
!b.den
ans =
2 2
20 + 5s - s 20 + 5s - s
2 2
20 + 5s - s 20 + 5s – s
```

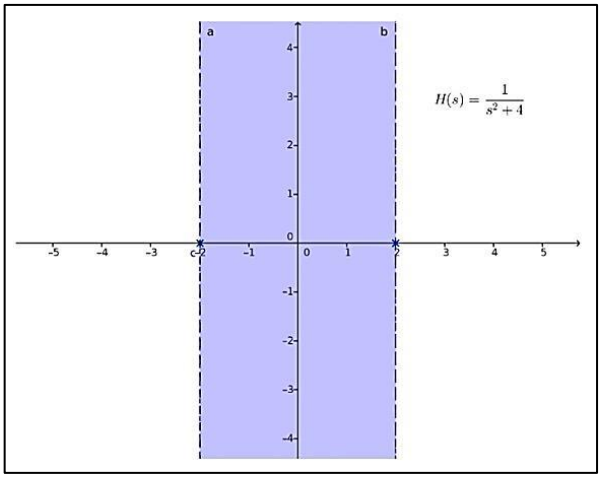
Fuente: elaboración propia, empleando SCILAB.

La versión bilateral de la transformada de Laplace brinda una mejor perspectiva del comportamiento de un sistema, y con ella es posible saber si un sistema es causal, anticausal o estable. En el análisis de funciones de transferencia, en el dominio de Laplace la variable *s* puede tomar valores

complejos y se asocia a este dominio un plano *x y*. Tomando el eje *x* como la parte real y el eje *y* como la parte imaginaria. Usualmente se coloca una £ en los puntos que corresponden a una raíz en el denominador de una función de transferencia y una *O* en los puntos que corresponden a una raíz en el numerador; estas raíces se conocen como polos y ceros de la función, respectivamente.

Se le llama regiones de convergencia a los semiplanos verticales o bandas verticales en el plano, que no contienen ningún polo. Y son útiles para verificar si el sistema es estable o no. Un sistema es estable si la región de convergencia contiene al eje *y*. Al considerar las funciones de entrada como eigenfunciones, los ceros y polos son útiles para el diseño de filtros. Para obtener ceros y polos de un polinomio, se utiliza el comando roots, el cual recibe como parámetro un polinomio. En el código 2.11, se obtuvieron los ceros y polos del objeto racional r. Se aclara que roots funciona solo para polinomios.

Figura 44. **Región de convergencia para un sistema estable**



Fuente: elaboración propia, empleando Inkscape.

Tabla LII. **Código 2.11: raíces de un polinomio**

```
!roots (r.num)
ans =
- 5.
5.
!roots (r.den)
ans =
1.
```

Fuente: elaboración propia, empleando SCILAB.

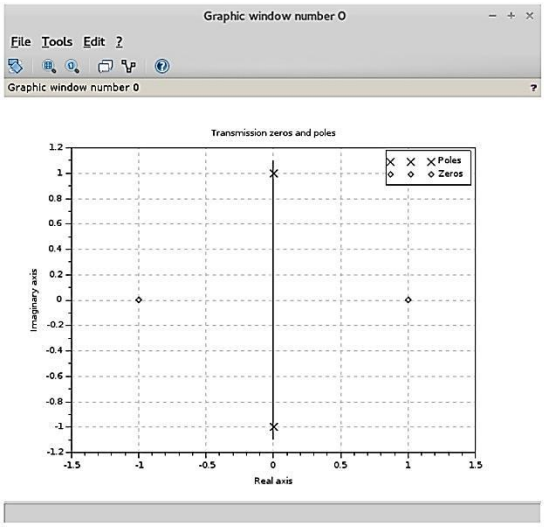
En estos ejemplos la variable que SCILAB reconoce como la constructora o semilla de polinomios es %s, al momento de declarar los polinomios, es la que se le ha indicado cuando se coloca en el comando poly el parámetro 's', aunque es posible declarar los polinomios con %z como constructora, colocando como parámetro 'z'. Es válido hacerlo con otras letras, pero las que SCILAB ya tiene reservadas son estas dos constantes como variables de polinomios. Por lo que es posible construir los polinomios sin usar poly, usando solo las constantes **%s** o %z.

Tabla LIII. **Código 2.12: variables de polinomios**

```
!k =12* %s +20* %s^2
k =
2
12s + 20s
!q+k
ans =
2
- 5 + 17s + 20s
```

Fuente: elaboración propia, empleando SCILAB.

Figura 45. **Comando plzr**



Fuente: elaboración propia, empleando SCILAB.

Para graficar los polos y ceros de una función de transferencia se utiliza la función plzr, la cual recibe como parámetro un objeto racional.

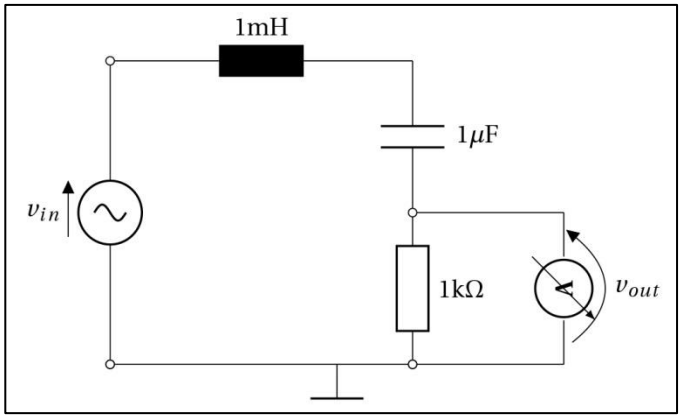
```
!p= poly ([ -1 ,1] , 's'); q= poly ([1 ,0 ,1] ,
's', 'c');
!r=p/q
r =
2
- 1 + s
-----
2
1 + s
!plzr ( syslin ('c', r))
```

Otras funciones muy útiles al trabajar con eigenfunciones y funciones de transferencia, y similares a plzr, son gainplot (genera gráficas de la magnitud

contra frecuencia), nyquist (genera gráficas de nyquist), bode(genera diagramas de bode), por mencionar algunas. En la documentación oficial se pueden hallar con más detalle. El comando syslin se tratará más adelante.

Utilizando SCILAB es posible generar simulaciones del comportamiento de un sistema determinado, utilizando su función de transferencia. Por ejemplo, suponga que se desea simular el circuito RLC de la figura 46. Dicho circuito se compone por una resistencia de un kilo-ohmio, $R = 1K\Omega$, una inductancia de un milihenrio, $L = 1mH$, y un capacitor de un micro-faradio, $C = 1\mu F$.

Figura 46. **Circuito RLC**



Fuente: elaboración propia, empleando Inkscape.

Dicho circuito se encuentra en reposo, capacitor e inductor descargados. En el instante $t = 0$ se conecta una fuente $vin = sin\{2 * \pi f\ t\}$, cuya frecuencia lineal es de quinientos hertz, $f = 500Hz$. Para construir la función de transferencia, es necesario encontrar las constantes impedancias en el dominio de la frecuencia de cada elemento discreto teniendo para la resistencia

R, para la inductancia $L = sL$ y para la capacitancia $\mathrm{C} = \frac{1}{sC}$. El circuito se compone de una malla por lo que la función de transferencia está dada por:

$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{RCs}{LCs^2 + RCs + 1}$$

[Ec. 2.373]

Dicha función de transferencia representa un sistema LIT, compuesto por ecuaciones diferenciales. Para visualizar el comportamiento del voltaje $vout$ se realiza una simulación en SCILAB, el código 2.13 es quien genera dicha simulación, muchas veces es más útil ver el comportamiento de una forma comprensible que encontrar la solución analítica del sistema.

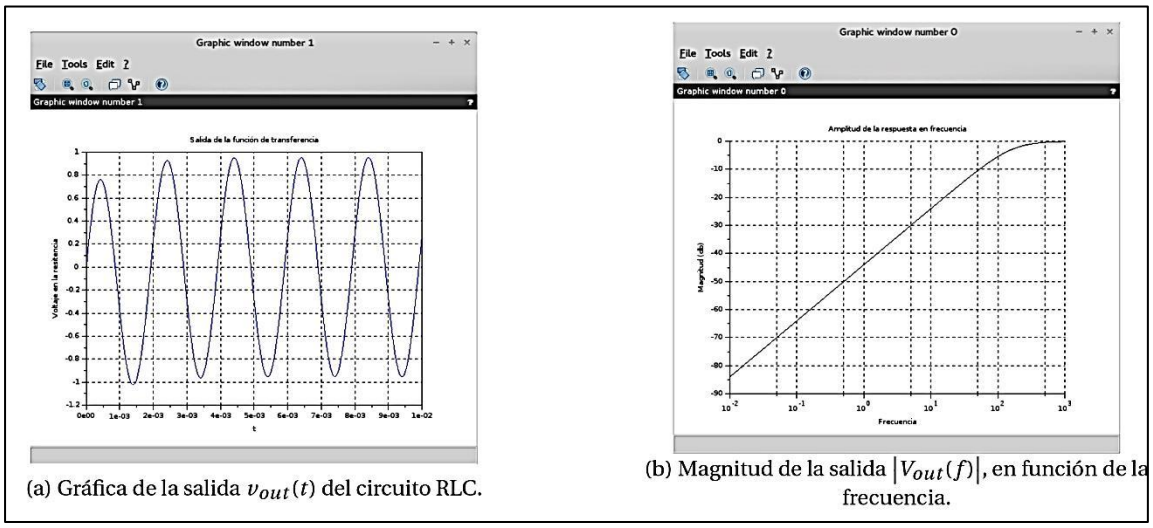
Tabla LIV. **Código 2.13: simulación de un sistema lineal**

```
// ////////////////////////////////////////
2 // Autor : Oscar Ram írez .
3 // Universidad de San Carlos de Guatemala
4 // ////////////////////////////////////////
5 clear ; clc; clf;
6 R =1000; L =1*10^ -3; C =1*10^ -6; f =500; s= poly (0, 's');
7 p=s*R*C; q=L*C*s^2+R*C*s+1; H=p/q;
8
9 sys = tf2ss (H); sys= syslin ('c', sys);
10 t= linspace (0 ,10*10^ -3 ,500) ;
11 u=sin (2* %pi*f*t);
12 y= csim (u, t, sys);
13
14 g0=scf (0) ; clf(g0);
15 gainplot (sys , 0.01 , 1000) ;
16 ax0 =gca ();
17 ax0 . grid =[1 ,1];
18 ax0 . x_label . text ='Frecuencia ';
19 ax0 . y_label . text ='Magnitud (db)';
20 ax0 . title . text ='Amplitud de la respuesta en frecuencia ';
21
22 g1=scf (1) ; clf(g1);
23 plot (t,y);
24 ax1 =gca ();
25 ax1 . grid =[1 ,1];
26 ax1 . x_label . text ='t';
27 ax1 . y_label . text ='Voltaje en la resitencia ';
28 ax1 . title . text ='Salida de la funci ón de transferencia ';
```

Fuente: elaboración propia, empleando SCILAB.

Como buena práctica, se deben declarar los valores de las constantes antes de continuar con el resto de los comandos. Ya que realizar un cambio al valor de una constante lo genera en el resto de los comando ahorrando sustituir en todos los valores. El *script* anterior genera la gráfica 48a, que es un reflejo del comportamiento del voltaje $vout\ (t\ )$ quien representa la salida del circuito, visto como un sistema con una función de transferencia $H(s)$, cuando $vin(t\ )$ es una entrada.

Figura 47. **Gráficas del análisis de una función de transferencia**



(a) Gráfica de la salida $v_{out}(t)$ del circuito RLC.

(b) Magnitud de la salida $|V_{out}(f)|$, en función de la frecuencia.

Fuente: elaboración propia, empleando SCILAB.

En la figura 47b se muestra el resultado obtenido con el comando gainplot, se muestra la reacción de la amplitud de un sistema lineal a una eigenfunción de la forma $Ae\int 2\pi\int\ t\ $. Para realizar la simulación es necesario escribir la función de transferencia de una forma que SCILAB pueda comprender, y es por eso que se usan los comandos tf2ss y syslin. La función syslin a forma de lista, a un sistema lineal para poder utilizarse en SCILAB. Los sistemas lineales que

puede interpretar syslin son sistemas de ecuaciones diferenciales e integrales tales que pueden ser descritos por la siguiente ecuación:

$$s\boldsymbol{x} = A\boldsymbol{x} + Bu \qquad \text{[Ec. 2.374]}$$

$$\boldsymbol{y} = C\boldsymbol{x} + Du \qquad \text{[Ec. 2.375]}$$

Donde *A*, *B*, *C* y *D* son matrices, y, x y u son vectores de funciones en el dominio de Laplace. Hasta el momento solo se han tratado con la transformada de Laplace sistemas SISO, pero es posible analizar sistemas MIMO compuestos por un conjunto de ecuaciones diferenciales. Los sistemas SISO pueden verse como un caso especial de los sistemas MIMO, donde el número de entradas y salidas corresponde a uno.

Cuando se trata con un sistema lineal MIMO compuesto por un conjunto de ecuaciones diferenciales, una forma conveniente de representar el problema es mediante matrices, y es por eso que el comando syslin usa la forma general descrita por matrices para estos sistemas. Considérese un sistema con entradas $u_1(t)\, y\, u_2(t)$ y salidas $y_1(t)$ y $y_2(t)$, determinadas la solución del siguiente conjunto de ecuaciones diferenciales.

$$\frac{d^2y_1(t)}{dt^4} + 4\frac{dy_1(t)}{dt} - 3y_2(t) = u_1(t) \qquad \text{[Ec. 2.376]}$$

$$\frac{dy_1(t)}{dt} + \frac{dy_2(t)}{dt} + 2y_2(t) = u_2(t) \qquad \text{[Ec. 2.377]}$$

Si se desea expresar dicho sistema en forma matricial, es necesario asignar variables intermedias, llamadas variables de estados. La principal

ventaja de tratar con sistemas con derivadas e integrales, es que se pueden construir las veces que sean necesarios nuevas variables que representan la derivada de otra, con el fin de reducir a un sistema como los trabajados por syslin y evitar derivadas de orden superior. Por ejemplo: para el sistema que se trata como ejemplo, las variables de estado quedarían de la siguiente forma:

$$x_1(t) = y_1(t)$$

[Ec. 2.378]

$$x_2(t) = \frac{dy_1(t)}{dt}$$

[Ec. 2.379]

$$x_3(t) = y_2(t)$$

[Ec. 2.380]

De esta forma para el ejemplo no se trata, con ninguna derivada de orden mayor a uno. Es posible reescribir el sistema en base a las variables declaradas anteriormente:

$$\frac{dx_2(t)}{dt} = 4x_2(t) + 3x_3(t) + u_1(t)$$

[Ec. 2.381]

$$\frac{dx_3(t)}{dt} = -x_2(t) - x_1(t) - 2x_3(t) + u_2(t)$$

[Ec. 2.382]

Es necesaria una expresión para $\frac{dx_1(t)}{dt}$, pero esta no es más que $x2(t)$ por la construcción de las variables de estado. Basándose en estas ecuaciones la forma matricial para el sistema tratado estaría dado por:

$$\begin{pmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \\ \frac{dx_3(t)}{dt} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -4 & 3 \\ -1 & -1 & -2 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u_1(t) \\ u_2(t) \end{pmatrix}$$

[Ec. 2.383]

$$\begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{pmatrix}$$

[Ec. 2.384]

Se considera a **x** como el vector de variables de estado en el dominio de Laplace y *A* a la matriz que lo acompaña llamada también matriz de estado, **u** el vector de entradas en el dominio de Laplace y *B* la matriz que lo acompaña; en la segunda ecuación se considera **y** el vector de salidas y *C* la matriz que acompaña a **x** y *D* en este caso es la matriz cero. Se puede ver de esta forma como las soluciones para un sistema de ecuaciones diferenciales pueden ser transformaciones para un conjunto de entradas. El encargado de transformar una función de transferencia a un sistema matricial es el comando tf2ss y así poder analizar de una forma más general el comportamiento del sistema y no solamente partiendo desde el reposo, como se había hecho con la función de transferencia.

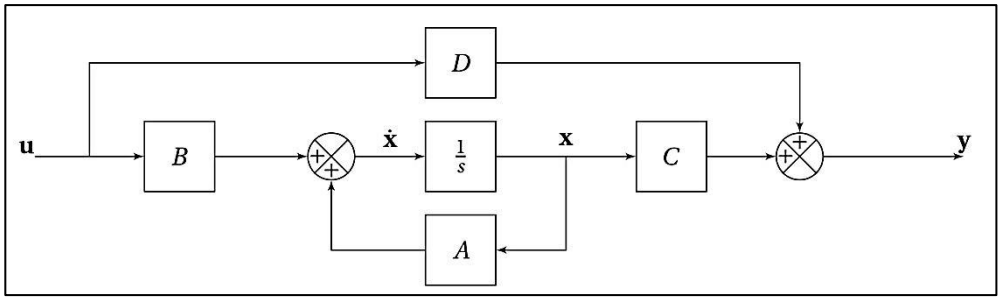
Una vez que el comando sys=tf2ss(H) ha transformado la función de transferencia a un espacio de estados (la forma matricial), y almacenado dicho espacio en la variable sys, se le da forma de lista para poder procesarlo y acá entra en juego el comando syslin. Luego se utiliza csim para obtener la salida del sistema y almacenarla en un vector. Es posible almacenar en un vector el comportamiento de las variables de estado si en lugar de almacenar en una sola variable, y=csim(u,t, sys), se almacena en una matriz [y,x]=csim(u,t,sys),

con las variables que almacenarán las salidas y las variables de estado respectivamente. El comando csim recibe como primer parámetro una matriz con las entradas dependientes en el tiempo dado como vector en el segundo parámetro, y como tercer parámetro el sistema a simular dado en una forma de lista.

En el código 2.13, se muestra la gráfica del comportamiento de la salida y. Además se muestra como manipular las ventanas de gráfica. El comando scf, indica que número de ventana se utilizará para graficar, en este código se utilizó la ventana 0 para graficar la ganancia en amplitud y la ventana 1 para graficar la salida. Es posible obtener una serie de propiedades con el comando gca del la ventana en uso. Al almacenarlas en un objeto es posible ir modificando la gráfica tal como se muestra en el código.

La figura 48 muestra como se estructuran los sistemas devueltos por tf2ss y que syslin convierte en listas.

Figura 48. **Diagrama de bloques de un sistema matricial**



Fuente: elaboración propia, empleando Inkscape.

Si las funciones de transferencia ahora son matrices, la propiedad de conmutabilidad en el álgebra para bloques lineales ya no es válida. Para ver el

funcionamiento del comando tf2ss(H) al ingresarlo en la consola, se pueden ver las matrices que componen el espacio de estados para $H(s)$.

Tabla LV. **Código 2.14: espacio de Estados de *H*(*s*)**

```
!tf2ss (H)
ans =
ans (1) (state - space system :)
!lss A B C D X0 dt !
ans (2) = A matrix =
- 1000000. - 30517.578
32768. - 4.997D -13
ans (3) = B matrix =
- 1000.
2.220D -13
ans (4) = C matrix =
- 1000. 2.220D -13
ans (5) = D matrix =
0.
ans (6) = X0 ( initial state ) =
0.
0.
ans (7) = Time domain =
[]
```

Fuente: elaboración propia, empleando SCILAB.

Puede verse que en lo devuelto por tf2ss(H) hay una matriz para los valores iniciales. Para indicarle a SCILAB que es un sistema continuo es necesario colocar como la cadena ’c’ parámetro en el comando syslin. De esa forma es posible simularlo con csim. Para dar condiciones iniciales, se debe modificar la lista para ingresar la matriz con el valor inicial de cada una de las variables de estado, sys.X0=[x0;x0], donde x0 es un valor específico para cada estado. Se debe tener cuidado como se configuran los valores iniciales para un sistema lineal que ha sido devuelto por tf2ss para evitar dar a un estado un valor que no corresponde. Si se modela desde el principio el sistema como un espacio de estados, en lugar de una función de transferencia, se es menos propenso a cometer ese error.

Hasta ahora se han utilizado comandos o funciones propias de SCILAB; sin estar claro qué son. Una función se reconoce por su sintaxis al llamarla. Por ejemplo, la sintaxis para la función o comando sin es la misma que para sinh. Existe una pequeña diferencia entre ambas funciones y es que sin es una primitiva o una función *hard-coded*, las cuales son escritas en los lenguajes en que se encuentra escrito SCILAB. Por otro lado, sinh es una función *SCILAB-coded*, y recibe este nombre porque esta escrita en SCILAB. Se puede verificar el tipo de ambas funciones con el comando typeof. Una función *SCILAB-coded* se puede definir usando las palabras reservadas function y endfunction. Puede ser ejecutada a través de un *script*, usando el comando exec. Es posible ver el código fuente de las funciones *SCILAB-coded* usando el comando fun2string o bien en Scinotes, posicionarse sobre el nombre de la función y presionar Ctrl+Clic. Las funciones devuelven objetos y reciben objetos como parámetros, incluso si estos objetos son otras funciones.

Tabla LVI. **Código 2.15: definición de una función**

```
!function y= funcion (x, g); y=g(x); endfunction
!funcion ( %pi , cos)
ans =
- 1.
```

Fuente: elaboración propia, empleando SCILAB.

Cuando se define una función se debe de dar un nombre a la función, una lista de argumentos y una lista de variables que se utilizan para devolver valores. La siguiente sintaxis se utiliza para definir una función:

```
1 function [< nombre1 >,< nombre2 >, ...]= < nombre de la funci ón >(< arg1 >, <arg2 >,
...)
2 <instrucciones >
3 endfunction .
```

Cuando una función se llama, las expresiones dadas como argumentos son primero evaluadas y luego sus valores son pasados a la función para que ser evaluados en la función. SCILAB usa un mecanismo de parámetros por valor para los tipos de argumento. Si un argumento de una instrucción de llamado para alguna función es un nombre de una variable y la variable no se altera o no realiza nada en la evaluación del cuerpo de la función, entonces la variable no se copia durante la llamada a la función. Para llamar la función se debe hacer de la siguiente manera: [<v1>,<v2>,...,<vp>]=<nombre de la función>(<expr1>,<expr2>,...)

Una variable que no es un argumento y no se encuentra definida localmente en la función puede tener un valor se se le asigna en el entorno donde se hace la llamada a la función. La variable en el entorno donde se hizo la llamada a la función no puede ser cambiada. Si se hace una asignación a esta variable usada, se crea una copia local de ella. La evaluación del cuerpo de la función se detiene cuando todas las <instrucciones> se ejecutaron ya o el flujo de instrucciones alcanza una instrucción return. Cuando la evaluación de la función finaliza, el flujo de instrucciones regresa al entorno de llamada a la función, precisamente a la llamada de la función. Los valores devueltos <nombre1>, <nombre2>,... conservan los valores que tenían cuando la evaluación de la función se detuvo.

Tabla LVII. **Código 2.16: ejecución de una función *SCILAB-coded***

```
!function [B,C]=f(A)
!B= string ( sign (A));
!B= strsubst (B,'1', '+');
!B= strsubst (B,' -+', '-');
!C=A (1:2 ,1) ;
!endfunction
![X,Y]=f([ -1 , -5 , -6;8 ,5 ,4])
```

Continuación de la tabla LVII.

```
Y =
- 1.
8.
X =
!- - - !
! !
!+ + + !
```

Fuente: elaboración propia, empleando SCILAB.

Todas las simulaciones se han hecho solamente en sistemas lineales. El problema es que la gran mayoría de modelos no son lineales, como en el ejemplo del micrófono; es más, muchas veces ni siquiera son totalmente continuos, tal como una colisión. Para simular un modelo en SCILAB, es necesario conocer la clase de modelo, para utilizar la herramienta correcta. En SCILAB es posible simular las siguientes clases de modelos:

- Ecuaciones diferenciales ordinarias. Son los modelos continuos más populares y son los que se han tratado hasta ahora, tienen la forma

$$\dot{y} = f(t, y) \qquad \text{[Ec. 2.385]}$$

Donde y y f tienen forma vectorial y t es un escalar representando al tiempo. Puede verse que la forma anterior para sistemas modelados con ecuaciones diferenciales ordinarias involucra solamente a la primera derivada, sin embargo, modelos involucrando ecuaciones diferenciales de orden superior pueden ser reescritos como el sistema de la ecuación 2.384 solamente agregando variables adicionales. La herramienta en SCILAB para la simulación de estos sistemas es ode. La cual asume que el sistema se ha escrito como un conjunto de ecuaciones con derivadas de primer orden. Por ejemplo, para la

ecuación de movimiento del péndulo doble, es posible escribirlo como un sistema de ecuaciones diferenciales de primer orden agregando variables adicionales.

$$\dot{\theta}_1 = \theta_3 \qquad \text{[Ec. 2.386]}$$

$$\dot{\theta}_2 = \theta_4 \qquad \text{[Ec. 2.387]}$$

$$\dot{\theta}_3 = \frac{m_2 \sin(\theta_1 - \theta_2)\,(l_2(\theta_4)^2 + l_1(\theta_3)^2 cos(\theta_1 - \theta_2) - \mathrm{g}\cos\theta_2) + m_1 \mathrm{g}\sin\theta_2}{m_1 l_1 + m_2 l_1 \sin^2(\theta_1 - \theta_2)}$$

[Ec. 2.388]

$$\dot{\theta}_4 = -\frac{l_1}{l_2}\dot{\theta}_3 \cos(\theta_1 - \theta_2) - \frac{l_1}{l_2}(\theta_3)^2 \sin(\theta_1 - \theta_2) + \frac{\mathrm{g}}{l_2}\sin\theta_2$$

[Ec. 2.389]

Dicha ecuación diferencial resulta en una familia de ecuaciones. Es necesario brindar información adicional para que la solución sea única. Dicha condición inicial es el valor de *y* a tiempo *t* específico. La existencia de una única solución pareciera ser un tema que solo incumbe a la matemática teórica, sin embargo, juega un papel muy importante para la simulación de sistemas, ya que las herramientas para simularlos tratan de alcanzar una determinada precisión y exactitud deseada y la estimación del error, se encuentra basado en el número de soluciones que posee una ecuación diferencial.

- Problemas con valores de frontera. Son ecuaciones diferenciales, pero con información dada en dos instantes o más de una vez de solamente una vez. Por ejemplo, un problema con valores de frontera para dos puntos toma la forma general:

$$\dot{y} = f(t, y), \quad t_0 \leq t \leq t_f \qquad \text{[Ec. 2.390]}$$

$$0 = B\left(y(t_0), y\left(t_f\right)\right) \qquad \text{[Ec. 2.391]}$$

Los problemas en las guías de onda son un ejemplo de este tipo. La ventaja de las ecuaciones diferenciales ordinarias es que las condiciones iniciales están dadas en un punto. Lo que implica que un método numérico dado un punto inicial, puede calcular el siguiente para el próximo paso. Para los problemas con valores de frontera esto no es posible por que la información se encuentra dada para dos puntos.

- Ecuaciones en diferencias. La segunda gran clase de modelos son las ecuaciones en diferencias. Estos problemas son de interés cuando hay cantidades en valores discretos o dichas cantidades cambian en tiempo discreto o bien es un fenómeno continuo, pero solo se puede observar a tiempos aislados. También, muchos métodos numéricos para ecuaciones diferenciales pueden resolver una ecuación en diferencias de forma aproximada. En las ecuaciones en diferencias, la variable independiente es un entero.

- Ecuaciones diferenciales algebraicas: muchos sistemas físicos por lo general son modelados con ecuaciones algebraicas y diferenciales. Estos sistemas toman la forma general:

$$F(t, y, \dot{y}) = 0 \qquad \text{[Ec. 2.392]}$$

son conocidas como DAE por sus siglas en inglés de *differential algebraic equations*. Un lagrangiano con restricciones holonómicas es un ejemplo de estos sistemas.

- Sistemas híbridos. muchos sistemas involucran una mezcla de sistemas discretos y continuos. Este tipo de sistemas son llamados por lo general sistemas híbridos. La naturaleza discreta de un sistema puede ocurrir de muchas maneras y esta información debe tomarse en cuenta en la simulación. Los tiempos en los que cambia una variable discreta son llamados eventos.

La herramienta principal para la simulación de ecuaciones diferenciales es ode. Al momento de considerar un integrador es necesario tomar en cuenta el tiempo inicial *t*0, el valor inicial *y*0 y la fórmula a evaluar, $f\ (y, t\ )$. Al hacer esto, el software escoge el método y las tolerancias, y ajusta el tamaño del *step* para satisfacer las tolerancias. Esto resulta en muchos más valores de la solución que los necesarios. Muchas veces los valores del tiempo utilizados por la herramienta no son los que se necesitan. Por lo que es posible ingresar un vector de tiempo para indicarle a la herramienta que tiempos utilizar. Quedando la llamada de forma simplificada de la siguiente forma:

```
! y=ode(y0 ,t0 ,t,f)
```

Acá f es una variable externa o cadena proveyendo la función que es el lado derecho de la ecuación 384. Si $t = [t\_0, t\_1, \ldots, t\_n]$ entonces se tendrá como salida $y = [y(t\_0), y(t\_1), \ldots, y(t\_n)]$. Considérese como ejemplo el siguiente sistema de ecuaciones:

$$\dot{y_1} = f(y_1, y_3) - y_1 \qquad \text{[Ec. 2.393]}$$

$$\dot{y_2} = \boldsymbol{b} y_1 - c y_2 \qquad \text{[Ec. 2.394]}$$

$$\dot{y_3} = y_2 - \boldsymbol{d} y_3, \qquad \text{[Ec. 2.395]}$$

Donde f es una función de las variables y_1, y_2. Las variables a, b, c y d son constantes, las cuales tienen los siguientes valores:

$$a = 0{,}187 \quad \text{[Ec. 2.396]}$$

$$b = 0{,}024 \quad \text{[Ec. 2.397]}$$

$$c = 0{,}062 \quad \text{[Ec. 2.398]}$$

$$d = 0{,}291 \quad \text{[Ec. 2.399]}$$

La función f , se encuentra definida como:

$$f(y_1, y_3) = \tau + y_1 máx\left\{s_{máx,} se^{-q\frac{y_3}{y_1}}\right\} \quad \text{[Ec. 2.400]}$$

donde $\tau, smáx, s$ y q son constantes con los siguientes valores:

$$\tau = 50\,000 \quad \text{[Ec. 2.401]}$$

$$s = 0{,}610 \quad \text{[Ec. 2.402]}$$

$$q = 3{,}443 \quad \text{[Ec. 2.403]}$$

$$s_{máx} = 0{,}1 \quad \text{[Ec. 2.404]}$$

Si se desea encontrar una solución puntal del sistema anteriormente descrito, es necesario conocer algún punto específico.

$$t_0 = 1\,970 \qquad \text{[Ec. 2.405]}$$

$$y_1(t_0) = 1{,}4x10^6 \qquad \text{[Ec. 2.406]}$$

$$y_2(t_0) = 1{,}3x10^5 \qquad \text{[Ec. 2.407]}$$

$$y_3(t_0) = 1{,}1x10^5 \qquad \text{[Ec. 2.408]}$$

Para simular este ejemplo en SCILAB es útil definir la función f ya sea en la consola o en Scinotes, para simplemente invocarla sin necesidad de escribir toda la expresión cada vez que se necesite. Se recomienda utilizar *scripts* ya que es más fácil modificarlo que volver a definir la función por completo en caso que haya un error.

Tabla LVIII. **Código 2.17: simulación utilizando ode. Función *f* del ejemplo**

```
1 function I=f(y1 ,y3 , param )
2 tau= param .tau;
3 s= param .s;
4 q= param .q;
5 smax = param . smax ;
6 s1=s* exp (-q*y3 ./ y1);
7 s1= max (smax , s1);
8 I=tau+s1 .* y1;
9 endfunction
```

Fuente: elaboración propia, empleando SCILAB.

Ahora para utilizar ode, es necesario definir la expresión que iguala a las derivadas.

Tabla LIX. **Código 2.18: simulación utilizando ode. Expresión para derivadas**

```
1 function Y= ders (t, y, param )
2 a= param .a;
3 b= param .b;
4 c= param .c;
5 d= param .d;
6 y1=y(1 ,:);
7 y2=y(2 ,:);
8 y3=y(3 ,:);
9
10 y1dot =f(y1 ,y3 , param )-a*y1;
11 y2dot =b*y1 -c*y2;
12 y3dot =y2 -d*y3;
13
14 Y=[ y1dot ; y2dot ; y3dot ]
15 endfunction
```

Fuente: elaboración propia, empleando SCILAB.

Lo más recomendable es crear una lista con los parámetros del sistema, por la misma razón que se creó una función para f , es más fácil identificar un error o corregirlo en caso que lo hubiese, y al alterar la definición de las constantes en el *script*. Para crear una lista en SCILAB, basta con declararla como un vector y luego nombrar cada uno de sus elementos, asignándoles su valor correspondiente, teniendo así la estructura <nombre de la Lista>.<nombre del elemento>=<valor del elemento>. Esto facilita ingresar los parámetros a una función solamente enviando el nombre de una variable y no cada uno de ellos.

Tabla LX. **Código 2.19: simulación utilizando ode. Lista de parámetros**

```
1 param =[]
2 param .tau =5 e4;
3 param .s =0.61;
4 param .q =3.443;
5 param . smax =0.1;
6 param .a =0.187;
7 param .b =0.024;
8 param .c =0.062;
9 param .d =0.291;
```

Fuente: elaboración propia, empleando SCILAB.

Para invocar un elemento de la lista basta con llamarla, colocar un punto y el nombre del elemento perteneciente a ella que se necesita en el momento de la invocación, <lista>.<elemento>. Dichas llamadas se hicieron en las funciones f y ders, así como el paso de todos los parámetros como un solo argumento.

Una vez que los parámetros se encuentran definidos, es necesario definir los vectores que formarán parte de la simulación, y la punto inicial de la solución fue dado en las ecuaciones desde 2.406 hasta 2.408.

Tabla LXI. **Código 2.20: simulación utilizando ode. Condiciones iniciales I**

```
1 Tbegin =1970;
2 y1_0 =1.4 e6;
3 y2_0 =0.13 e6;
4 y3_0 =0.11 e6;
5 y0 =[ y1_0 ; y2_0 ; y3_0 ];
6 t0= Tbegin ;
```

Fuente: elaboración propia, empleando SCILAB.

Además, se desea simular el ejemplo desde *t* teniendo un valor de 1 970 hasta 2 020, a intervalos de 1/12. Teniendo ya todos los argumentos necesarios para poder simular el ejemplo, utilizamos ode para ello. Se puede observar que en el vector del tiempo, se agregó una pequeña tolerancia para asegurarse que en la simulación el valor Tend se tome en cuenta.

Tabla LXII. **Código 2.21: simulación utilizando ode. Condiciones iniciales II**

```
1 Tend =2020;
2 Tstep =1/12;
3 t= Tbegin : Tstep :( Tend +100* %eps )
4 Sys =ode (y0 , t0 , t, ders );
```

Fuente: elaboración propia, empleando SCILAB.

Hasta este punto se ha cumplido con la definición de simulación. Se han extraído datos de un modelo. Sin embargo, es necesario graficar dichos datos, para ser comprensibles por un humano. Teniendo así un *script* que genera la gráfica 50.

Tabla LXIII. **Código 2.22: simulación utilizando ode. *Script***

```
1 clear ,clc; // Se limpian las variables y consola
2 // Se declara la funci ón f
3 function I=f(y1 ,y3 , param )
4 tau= param .tau;
5 s= param .s;
6 q= param .q;
7 smax = param . smax ;
8 s1=s*exp (-q*y3 ./ y1);
9 s1=max (smax , s1);
10 I=tau+s1 .* y1; //f(y1 ,y3 , param )
11 endfunction
12
13 // Expresi ón para el vector de derivadas
14 function Y= ders (t, y, param )
15 a= param .a;
16 b= param .b;
```

Continuación de la tabla LXIII.

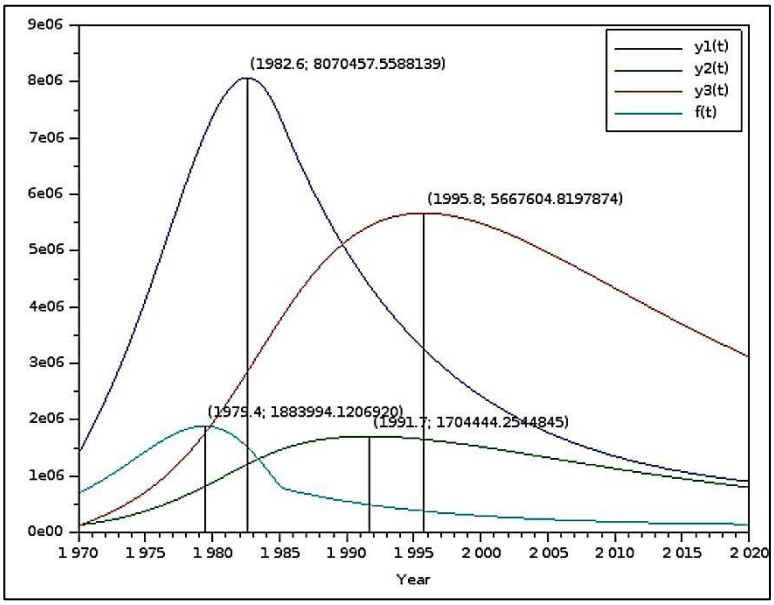
```
17 c= param .c;
18 d= param .d;
19 y1=y(1 ,:);
20 y2=y(2 ,:);
21 y3=y(3 ,:);
22
23 y1dot =f(y1 ,y3 , param )-a*y1;
24 y2dot =b*y1 -c*y2;
25 y3dot =y2 -d*y3;
26
27 Y=[ y1dot ; y2dot ; y3dot ] // Se debe devolver un vector de derivadas
28 endfunction
29
30 // Declaraci ón general de parámetros como lista
31 param =[]
32 param .tau =5 e4;
33 param .s =0.61;
34 param .q =3.443;
35 param . smax =0.1;
36 param .a =0.187;
37 param .b =0.024;
38 param .c =0.062;
39 param .d =0.291;
40
41 // Declaraci ón de variables .
42 Tbegin =1970;
43 y1_0 =1.4 e6;
44 y2_0 =0.13 e6;
45 y3_0 =0.11 e6;
46 y0 =[ y1_0 ; y2_0 ; y3_0 ];
47 t0= Tbegin ;
48 Tend =2020;
49 Tstep =1/12;
50 // Simulaci ón
51 t= Tbegin : Tstep :( Tend +100* %eps )
52 Sys =ode (y0 , t0 , t, ders );
53
 54 // Una vez ya simulado se procede a separar el resultado en los vectores LHY y
la funci ón f se guarda en I
55 L= Sys (1, :)
56 H= Sys (2, :)
57 Y= Sys (3, :)
58 I=f(L,Y, param );
59
60 // Se busca el valor má ximo y la posici ón donde se encuentra , así como en que
inst ánte t sucede
61 [Lmax , Lindmax ]= max(L); tL=t( Lindmax );
62 [Hmax , Hindmax ]= max(H); tH=t( Hindmax );
63 [Ymax , Yindmax ]= max(Y); tY=t( Yindmax );
64 [Imax , Iindmax ]= max(I); tI=t( Iindmax );
65
66 // Labels para los puntos má ximos
67 LText = msprintf ('( %4.1f; %0.7f)', tL , Lmax );
68 HText = msprintf ('( %4.1f; %0.7f)', tH , Hmax );
69 YText = msprintf ('( %4.1f; %0.7f)', tY , Ymax );
70 IText = msprintf ('( %4.1f; %0.7f)', tI , Imax );
71
72 // Gráfica del resultado
73 plot (t, [L;H;Y;I])
```

Continuación de la tabla LXIII.

```
74 // Se evita que el gráfico actual borre lo anterior por cada gráfica nueva que
se ponga en el.
75 set (gca () , " auto_clear ","off ");
76 // Se da el nombre de las gráficas como leyenda
77 legend ([ 'y1(t)';'y2(t)';'y3(t)';'f(t)']);
78 // Grá fica
79 xpolys ([tL ,tH ,tY ,tI;tL ,tH ,tY ,tI ] ,[0 ,0 ,0 ,0; Lmax ,Hmax ,Ymax , Imax ]);
80 // Se coloca el Label en algún punto del espacio .
81 xstring (tL , Lmax , LText );
82 xstring (tH , Hmax , HText );
83 xstring (tY , Ymax , YText );
84 xstring (tI , Imax , IText );
85 xlabel ('Year ');
86 // Al colocarse esta sentencia se permite que al volver a correr el script se
limpiará la pantalla de gráficos .
87 set (gca () , " auto_clear ", "off ")
```

Fuente: elaboración propia, empleando SCILAB.

Figura 49. **Gráficas de una simulación por ode**



Fuente: elaboración propia, empleando SCILAB.

El ejemplo anterior es conocido como LHY y es utilizado en el estudio del abuso de drogas. Es un modelo de tiempo continuo de la demanda de drogas por dos clases usuarios: los usuarios livianos denotados por $L(t)$ y los usuarios pesados denotados por $H(t)$. La variable $Y(t)$ representa el decaimiento de la memoria de los usuarios pesados en los últimos años, que funciona como disuasión para nuevos usuarios livianos.

Para el ejemplo anterior no se han modificado las opciones por defecto que del comando ode. Desafortunadamente, una simple llamada de ode no siempre funciona por lo que esta función ofrece una serie de opciones para mejorar el desempeño de la simulación. La resolución de una ecuación ordinaria es el resultado de usar un método de aproximación para encontrarla.

Para problemas con reinicios frecuentes, existen ventajas al utilizar métodos como Runge-Kutta, los cuales utilizan un paso fijo, sobre métodos que usan pasos diferentes como el método de Adams. Sin embargo, los métodos con pasos variables ofrecen menos computación. El orden de un método es un indicador del error relativo al tamaño del paso. Un método de cuarto orden se espera que tenga un error proporcional a h_4, asumiendo que *h* es el tamaño del paso. Por lo que ordenes altos son deseados. Para los métodos de pasos variables mientras más alto es el orden puede tomar pasos más pequeños que los esperados, y se debe tomar en cuenta para problemas con soluciones con muchas oscilaciones. El uso de un método de aproximación lleva implícita toda una teoría que escapa del alcance del texto. Se debe tener presente toda esa teoría al realizar una simulación relevante. Para el lector interesado en este tema se recomienda la publicación de G. Sallet ‘‘Ordinary Differential Equations with SCILAB,WATS Lectures, Provisional notes”.

Es posible modificar las opciones por defecto que ofrece el comando code por dos vías diferentes, la primera es utilizando la constante reservada %ODEPTIONS por un vector con las nuevas opciones, o llamando a la forma general del comando ode

Tabla LXIV. **Código 2.23: forma general de ode**

```
→[y,w,iw ]= ode ([ type ],y0 ,t0 ,t [, rtol [, atol ]],f [,jac] [,w,iw ])
```

Fuente: elaboración propia, empleando SCILAB.

Para la opción type existen hace referencia al método a utilizar. Por defecto se encuentra la opción lsoda escoge entre el método de Adams para ecuaciones firmes y el método BDF (*Backward Dif**ferentiation** For**mula*) para ecuaciones no firmes. No existe una definición universalmente aceptada de una solución firme, pero la idea principal es que se tiene algunos términos que pueden llevar a una variación rápida en la solución. Algunos métodos que se pueden listar:

- lsoda: El método por defecto.
- Adams: Método de paso variable, utilizado para problemas no firmes.
- Stiff: Método de paso variable, utilizado para problemas firmes. Método BDF.
- Rk: Método Runge-Kutta adaptivo de orden 4.
- Rkf: Otro método de Runge-Kutta hecho por Shampine y Watts. Se encuentra basado en el par 4 y
- 5 Runge-Kutta de Fehlberg. Es para soluciones firmes. No debe usarse cuando el usuario busca alta precisión.
- Fix: Similar a rkf, pero la interfaz solo utiliza los parámetros rtol y atol. Es el método más simple para probar.

- Root y discrete: Equivalentes para los comandos ode_root y ode_discrete.

Los parámetros rtol y atol, los cuales son contantes o vectores reales del mismo tamaño de y, cuyas entradas representan los errores relativos y absolutos como tolerancia que debe tener la solución. Los valores por defecto son rtol=1.de-7 y ato=1.d-7 para la mayoría de los métodos, sin embargo para los métodos rkf y fix presentan los valores rtol=1.d-3 y atol=1.d-4.

El parámetro jac es una función, lista o cadena que provee los métodos para el Jacobiano el cual puede ser utilizado en algunos métodos. Es recomendable variar estos valores en lugar del método. Los argumentos w y iw son vectores usados para almacenar información devuelta por el método. Cuando estos vectores son dados el método reinicia con los mismos parámetros que la detención previa. La variable del sistema %ODEOPTIONS permite cambiar algunos parámetros y la forma general de la variable esta dada por el vector:

%ODEOPTIONS =[itask ,tcrit ,h0 ,hmax ,hmin ,jactyp ,mxstep , maxordn , maxords ,ixpr ,ml ,mu]

El significado de cada una de ellas se puede encontrar en la documentación oficial de SCILAB. Resaltan h0 quien es el tamaño del primer paso que intenta el método. Puede utilizarse en el método de Runge-Kutta, donde la solución comienza cambiando lentamente, usar pasos pequeños sería un desperdicio de procesamiento. Hmax es el tamaño más grande del paso permitido a utilizar por el método, algunas veces los simuladores comienzan a utilizar pasos muy grandes y se ven incapaces de recuperarse de cambios rápidos o peor aún es posible que no tomen en cuenta un evento completo, hmin es el tamaño más pequeño que se acepta, al configurar su valor como

mayor que cero puede generar soluciones menos confiables, pero es útil en problemas con condiciones locales o pérdidas de suavidad que interfieren con una medida de error, mxstep es el máximo número de pasos que el método puede tomar, su valor por defecto es 500. Tener un número conjunto finito evita que el algoritmo pierda un tiempo muy largo debido al uso de pasos muy pequeños.

Por ejemplo simular la ecuación:

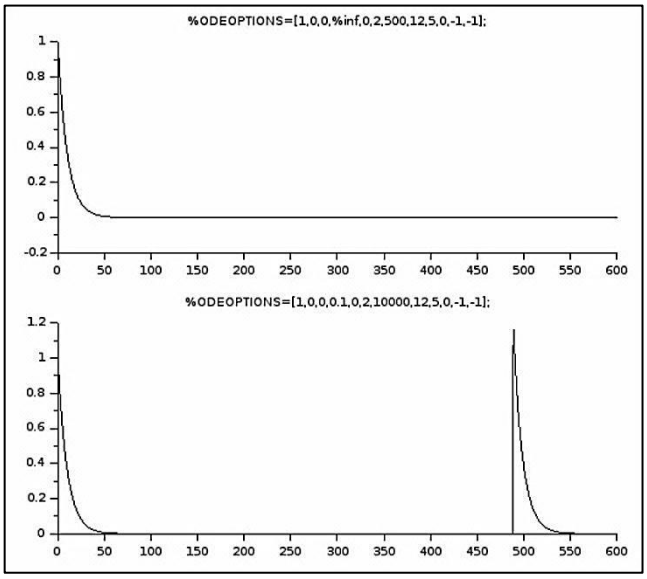
$$\dot{y} = -0{,}1y + \mathrm{g(t)}, \qquad \mathrm{y}(0) = 1, \qquad 0 \leq \mathrm{t} \leq 600 \qquad \text{[Ec. 2.409]}$$

donde $g\ (t\ )$ es cero en todos los valores a excepción 488,3 y 488,9 donde $g\ (488{,}3) = g\ (488{,}9) = 2$. Es posible dar una expresión para $g\ (t\ )$ de la siguiente manera:

$$g(t) = 0{,}5\big(1 + sign(\mathrm{t} - 488{,}3)\big)\big(1 - sign(\mathrm{t} - 488{,}9)\big) \qquad \text{[Ec. 2.410]}$$

En la figura 50 muestra las simulaciones realizadas con diferentes valores para %ODEOPTIONS, una de ella con las opciones por defecto y en la otra con hmax a 0,1 y mxstep a 10 000.

Figura 50. **Simulaciones con diferentes parámetros**



Fuente: elaboración propia, empleando SCILAB.

El siguiente *script* fue el utilizado para generar la figura 50

Tabla LXV. **Código 2.24: simulaciones con diferentes parámetros**

```
1 function z=g(t)
2 z =0.5*(1+ sign (t -488.3) )*(1 - sign (t -488.9) )
3 endfunction
4
5 function ydot = f(t,y)
6 ydot = -0.1* y+g(t)
7 endfunction
8
9 tt =0:0.1:600;
10 %ODEOPTIONS =[1 ,0 ,0 , %inf ,0 ,2 ,500 ,12 ,5 ,0 , -1 , -1];
11 y=ode (1,0,tt ,f);
12 subplot (211) ;
13 plot2d (tt ,y);
14 xtitle (' %ODEOPTIONS =[1 ,0 ,0 , %inf ,0 ,2 ,500 ,12 ,5 ,0 , -1 , -1]; ')
15
16 %ODEOPTIONS =[1 ,0 ,0 ,0.1 ,0 ,2 ,10000 ,12 ,5 ,0 , -1 , -1];
17 y=ode (1,0,tt ,f);
18 subplot (212) ;
19 plot2d (tt ,y);
20 xtitle (' %ODEOPTIONS =[1 ,0 ,0 ,0.1 ,0 ,2 ,10000 ,12 ,5 ,0 , -1 , -1]; ')
```

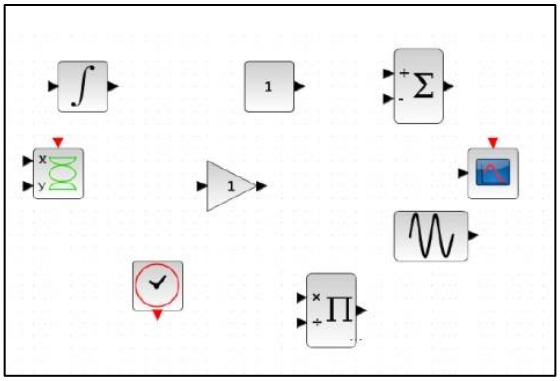
Fuente: elaboración propia, empleando SCILAB.

En la superior gráfica de la figura 51, el método que escogió el comando ode utiliza pasos adaptivos, y al ver que la pendiente de la función iba volviéndose una constante utilizó pasos más grandes cada vez con el fin de hacer eficiente la computación de la solución, obviando por completo el evento. Para la segunda simulación se forzó al programa a evitar pasos muy grandes y esa es la razón del comportamiento de las simulaciones. Como lección se tiene que, dado que la simulación es una herramienta para obtener datos de un modelo, es necesario conocerla bien para utilizarla correctamente, de la misma forma que se usaría la parte plana de un martillo para clavar y parte bifurcada para sacar el clavo de una superficie.

- XCOS

Muchas veces es un poco engorroso escribir código para realizar simulaciones, a pesar que ofrece alta flexibilidad, un mejor manejo de la herramienta y además de correr las simulaciones de manera más rápidamente, ya que la computadora no debe lidiar con interfaces. A pesar de todas estas ventajas que ofrece el código, es poco intuitivo, requiere de mayor análisis que una herramienta gráfica y además es necesario tener mayor cantidad de conocimientos sobre el lenguaje.

Figura 51. **Algunos bloques de Xcos**

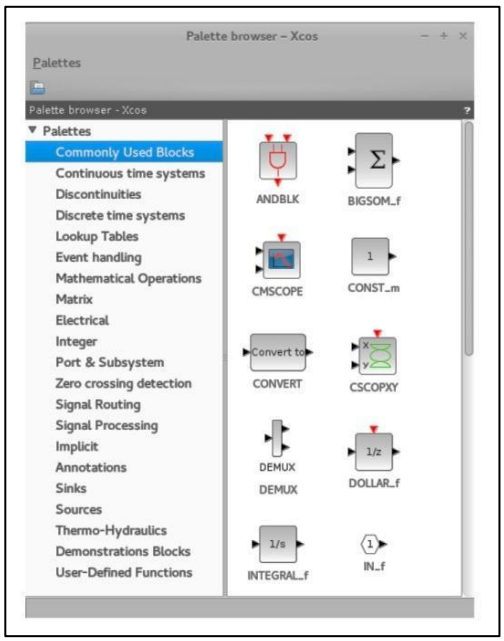


Fuente: elaboración propia, empleando SCILAB.

SCILAB ofrece un muy buen editor gráfico, basado en diagramación de bloques para realizar simulaciones. Esta *toolbox* es útil particularmente en teoría del control, procesamiento de señales y modelado en diversos dominios, especialmente discreto y continuo.

En Xcos el objeto principal son los bloques; un bloque en Xcos es un elemento que posee ya sea puertos de entradas o salidas, puertos de activación, puertos de tiempo continuo o discreto, entre otros. La figura 51 muestra la paleta, que contiene los bloques que pueden utilizarse en Xcos. La paleta se divide en varias regiones o pequeñas paletas, por lo que fácil encontrar un bloque determinado. Algunas paletas importantes que componen la paleta de Xcos son: bloques usados frecuentemente, Sistemas de tiempo continuo, Discontinuidades, Sistemas discretos, *Lookup Tables*, Eventos, Operaciones matemáticas, Matrices, Enrutamiento de señales, Procesamiento de señales, Expresiones definidas por usuario, Fuentes y Sumideros. Cada una de ellas contiene bloques específicos en cada una de las ramas mencionadas. El usuario puede verificar los bloques que hay en cada una de las paletas solamente haciendo *click* sobre su nombre.

Figura 52. **Paleta de bloques en Xcos**



Fuente: elaboración propia, empleando SCILAB.

En Xcos los bloques transmiten información entre ellos a través de enlaces a cada uno de sus puertos. Tipos de enlaces para un bloque en Xcos:

- Enlaces regulares: transmiten o reciben señales a través de estos puertos de un bloque a otro. Se encuentran identificados con un triángulo negro.

- Enlaces de activación: transmiten o reciben información del tiempo (discreto o continuo). Estos puertos se encuentran identificados por un triángulo rojo.

- Enlaces implícitos: frecuentes en Modelica, el cual es un lenguaje orientado a componentes, útil para el modelo de comportamientos

físicos, eléctricos, hidráulicos, mecánicos entre otros. Un enlace implícito no impone transferencia de información en una dirección conocida. Se identifican por un cuadro negro.

Como todo simulador, Xcos es una herramienta se debe conocer algunos bloques importantes en este entorno gráfico. La tabla XLIII muestra algunos de ellos así como alguna breve descripción. El usuario de Xcos solo podrá conectar enlaces del mismo tipo. El ejemplo por excelencia para simulaciones sencillas de sistemas continuos son circuitos RLC y en este caso se simulará el sistema lineal de la figura 46. Esta simulación se realizó anteriormente utilizando código.

Tabla LXVI. **Bloques en Xcos**

Nombre	Bloque	Paleta	Descripción
Integrador/ `INTEGRAL_m`		Sistemas de tiempo continuo	La salida `y(t)` es la integral de la entrada `u(t)` en el instante `t`. Numéricamente siempre es más robusto usar el bloque de integración en lugar que el de derivación.
Suma / `SUMMATION`		Operaciones Matemáticas	Realiza operaciones algebraicas de suma o resta, según la configuración.
Producto / `PRODUCT`		Operaciones Matemáticas	Realiza operaciones algebraicas de multiplicación o división, según la configuración.
Ganancia/ `GAINBLK`		Operaciones Matemáticas	La salida `y(t)` es la entrada `u(t)` escalada por constante definida por el usuario. Presenta opciones para desborde.
Tiempo Continuo / `CLOCK_c`		Fuentes	Este bloque genera una secuencia regular de eventos temporales con un periodo específico e inicio a un tiempo dado. Se utiliza para el bloque de visualización.
Constante / `CONST_m`		Fuentes	La salida `y(t)` es una constante definida por el usuario. No posee entradas.
Generador de sinusoidales / `GENSIN`		Fuentes	La salida `y(t)` es una sinusoidal con parámetros como magnitud, frecuencia y fase definidos por el usuario.
MUX / `MUX`		Enrutamiento de señales	Combina hasta 8 señales en una sola señal. Se utiliza por lo general cuando un bloque puede recibir una o varias señales para procesarlas. Por ejemplo para graficar en una misma pantalla de visualización.
Osciloscopio o *Scope* / `CSCOPE`		Sumideros	Este bloque es utilizado para mostrar una señal de entrada (o un vector de señales, por ejemplo la salida del bloque MUX) con respecto al tiempo, recibido de una fuente de tiempo. Se pueden modificar algunos parámetros para una mejor visualización.
Expresión de Usuario / `EXPRESSION`		Expresiones definidas por usuario	La salida `y(t)` de este bloque es una expresión matemática dependiente de máximo 8 entradas `u1, u2,...u8`. El nombre `u`, seguido de un número es obligatorio. Exactamente `u1` corresponde al primer puerto, `u2` al segundo puerto y así sucesivamente.

Fuente: elaboración propia.

Antes de simular es necesario un modelo. El modelo para dicho circuito está dado por las siguientes ecuaciones:

$$\frac{di(t)}{dt} = \frac{1}{L}\left(v_{in} - Ri(t) - e_c(t)\right)$$

[Ec. 2.411]

$$e_c(t) = \frac{1}{C}\int_{t_0}^{t} i(\tau)d\tau$$

[Ec. 2.412]

$$v_{out} = Ri(t)$$

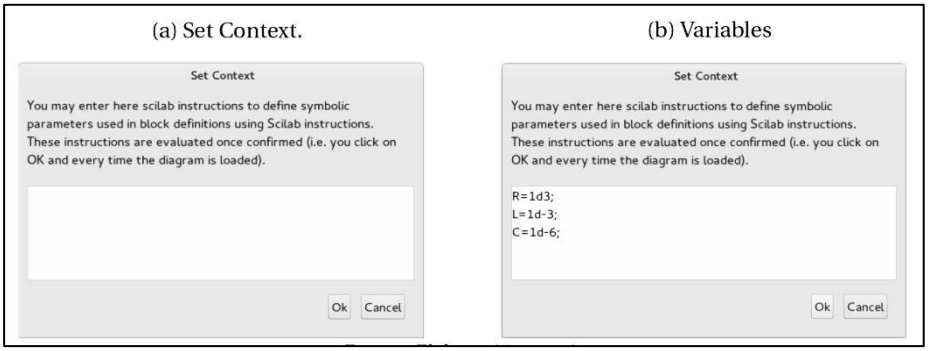
[Ec. 2.413]

Dichas ecuaciones se han expresado de esa forma por razones que se harán evidentes al momento de desarrollar la simulación en Xcos. Para iniciar el entorno Xcos basta con escribir en la consola:

→xcos

o bien presionando en el ícono de consola. Al hacerlo aparecerá el entorno gráfico junto con la paleta. Una vez iniciado el entorno Xcos, el primer paso a seguir es definir las variables a utilizar en el esquema, configurar el contexto. Para ello se hace hace clic en el menú Simulation y luego en Set Context. Al hacerlo emergerá una ventana como la de la figura 53a.

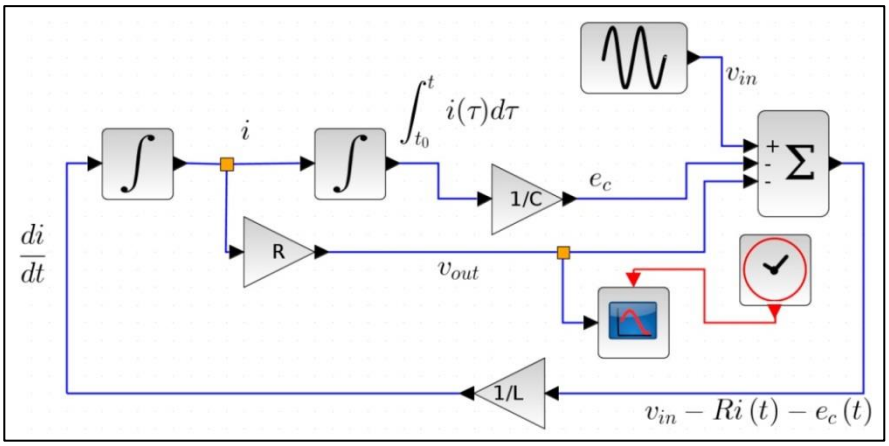
Figura 53. **Variables de entorno**



Fuente: elaboración propia, empleando SCILAB.

En la ventana se coloca el nombre de todas las constantes útiles en la simulación así como sus valores, tal como se muestra en la figura 53b, con el fin de evitar escribir los mismos valores frecuentemente y llevar un mejor control del flujo de la información, se le pone nombre a las cantidades. Una vez configuradas las variables se reproduce el esquema de la figura 54.

Figura 54. **Diagrama de bloques del circuito de la figura 46**

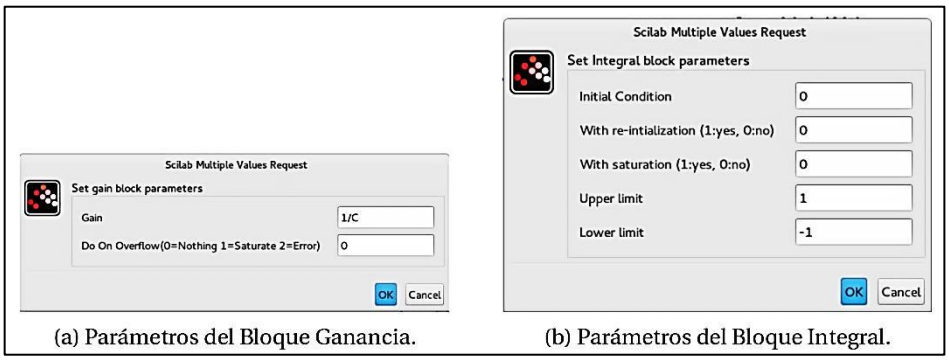


Fuente: elaboración propia, empleando SCILAB.

Para hacer los enlaces basta con hacer clic en alguno de los dos puertos que se comunican por el enlace y arrastrar el puntero hacia el otro puerto. Las etiquetas donde está la integral o la derivada en el diagrama, no tienen relevancia en el comportamiento de la simulación solo fueron puestas utilizando el bloque TEXT_F que se puede encontrar en la paleta Anotaciones.

Dicho bloque soporta comentarios en LATEX, como puede verse en la figura. Es posible configurar parámetros de cada bloque, por ejemplo, para el bloque Ganancia es posible configurar su constante dando doble *click* sobre él. De igual forma para modificar la fase, magnitud o frecuencia del generador de señales o bien para modificar el valor inicial en las integrales (ahora se ve porqué es más robusto usar un bloque de integral en lugar de un bloque de derivación). De cada bloque emergerán ventanas diferentes propias de cada uno de ellos, con parámetros que modifican el comportamiento de dicho bloque.

Figura 55. **Ventanas para configuración de bloques**



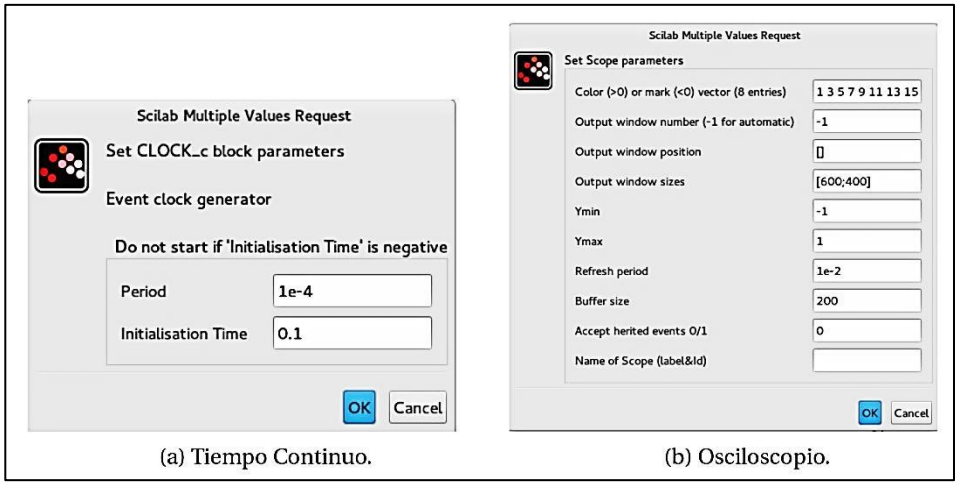
(a) Parámetros del Bloque Ganancia. (b) Parámetros del Bloque Integral.

Fuente: elaboración propia, empleando SCILAB.

El bloque de tiempo continuo genera un evento cada cierto tiempo, el cual es configurado por el usuario. En la ventana de configuración del tiempo continuo (el reloj con marco rojo), dicho tiempo aparece como *Period* o periodo.

En este ejemplo el periodo se configuro a $1 \times 10 - 4$. Este evento generado sirve como referencia para el bloque Osciloscopio, que se basa en esta señal para generar las gráficas. La figura 56 muestra las configuraciones empleadas en este ejemplo para los bloques de Tiempo continuo y Osciloscopio.

Figura 56. **Configuración de la presentación de gráficas**



(a) Tiempo Continuo. (b) Osciloscopio.

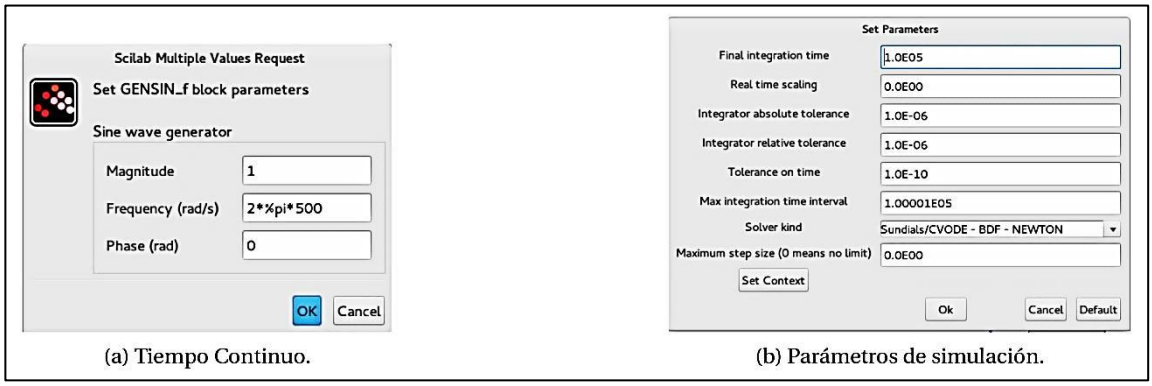
Fuente: elaboración propia, empleando SCILAB.

El tiempo de refresco que se muestra en esta figura, $1 \times 10 - 2$, representa el tiempo en que actualiza las gráficas el bloque osciloscopio, los valores Ymin y Ymax sirven para configurar el valor mínimo y máximo mostrados en el área de visualización de las gráficas.

Si el circuito se encuentra en reposo y la señal de entrada es una sinusoidal, el bloque GENSIN se encuentra configurado con magnitud de 1 y frecuencia de 500Hz. La interfaz gráfica solo es un medio más amigable para generar simulaciones, y al igual que al simular con código, es necesario configurar parámetros de simulación donde los usuarios escogen tolerancias y

métodos. Para poder alterar los parámetros de simulación basta con dar clic en Simulation y luego en Setup. Al hacerlo emergerá una ventana similar a de la figura 57 b.

Figura 57. **Configuración de simulación y entrada al sistema**

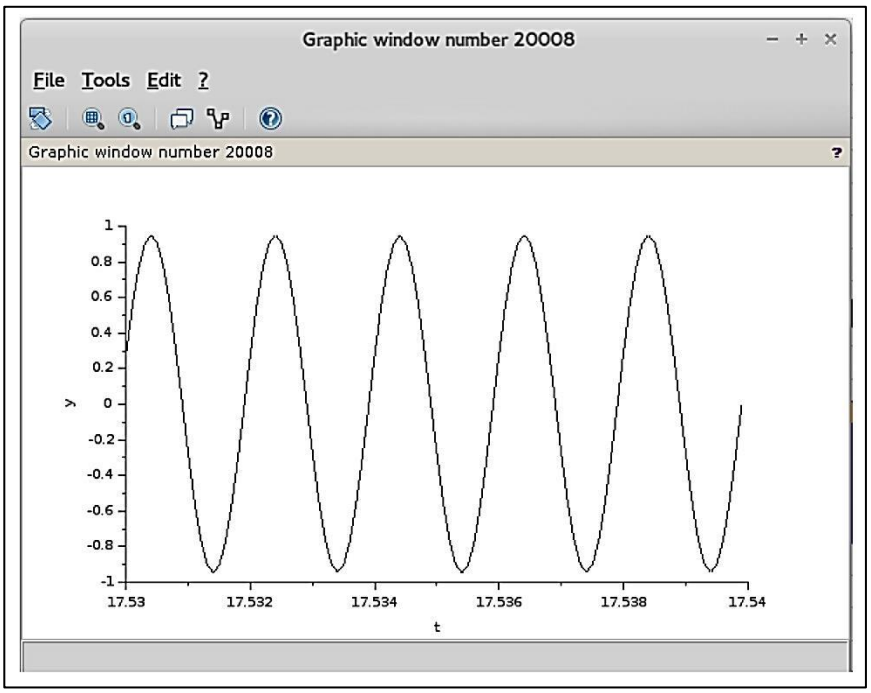


(a) Tiempo Continuo. (b) Parámetros de simulación.

Fuente: elaboración propia, empleando SCILAB.

La figura 57 b muestra la configuración que se tiene para este ejemplo. Esta ventana tendrá diferentes opciones dependiendo el método a utilizar. Se muestra en la figura 59 el resultado de la simulación, la cual es más dinámica que la gráfica generada con código, ya que la herramienta Scope se comporta como un osciloscopio, tal como su nombre lo indica, con los parámetros configurados en la figura 56 b.

Figura 58. **Salida en osciloscopio**



Fuente: elaboración propia, empleando SCILAB.

2.4. Prácticas de laboratorio propuestas. Módulo 2. Realizando modelos y simulaciones

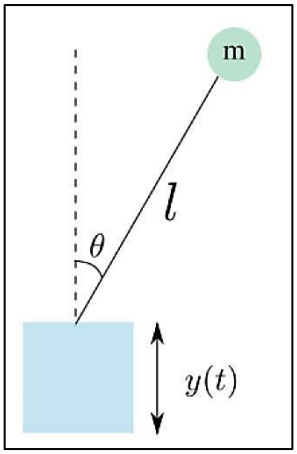
Se presenta una serie de problemas propuestos para resolver como parte del laboratorio del curso de Sistemas de Control, utilizados para evaluar la capacidad de modelar y simular sistemas físicos continuos. Se recomienda el simulador SCILAB, por contar con herramientas útiles como ode y csim. No es obligatorio utilizar solamente código en las simulaciones; es posible simular con alguna interfaz gráfica como Xcos. El estudiante es libre de escoger la herramienta y el software que considere conveniente.

2.4.1. Modelos

Cualquier aproximación de la mecánica da las ecuaciones de movimiento correspondientes a los siguientes sistemas:

- Péndulo invertido: consiste de una masa m al final de una varilla sin masa de longitud l . El otro lado de la varilla se hace oscilar verticalmente con una posición dada por $y(t) = Acos(\omega t)$, donde $A \ll l$. Ver la siguiente figura. Utilizar como grado de libertad θ.

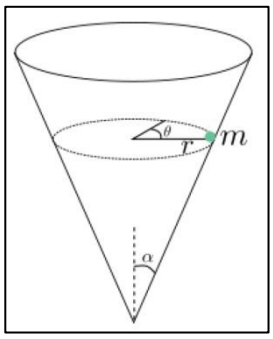
Figura 59. **Péndulo invertido**



Fuente: elaboración propia, empleando Inkscape.

- Movimiento en un cono: una partícula de masa m se desliza adentro de la superficie sin fricción de un cono. El cono se encuentra con la punta en el suelo y su eje es vertical. El medio ángulo del cono en la punta, está dado por a Sea $r\ (t)$ la distancia de la partícula al eje, $y\ \theta\ (t)$ el ángulo con respecto al eje del cono. Ver la siguiente figura.

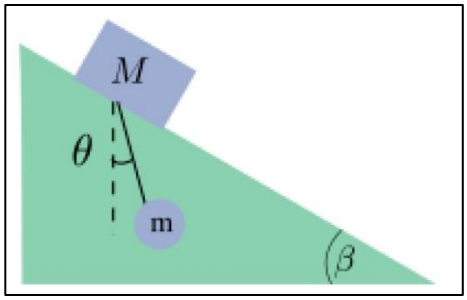
Figura 60. **Movimiento en un cono I**



Fuente: elaboración propia, empleando Inkscape.

- Péndulo en un plano: una masa M se encuentra libre de deslizarse en un plano inclinado sin fricción de ángulo β. Un péndulo de longitud l y masa m cuelga de *M*.

Figura 61. **Movimiento en un cono II**



Fuente: elaboración propia, empleando Inkscape.

2.4.2. Simulaciones

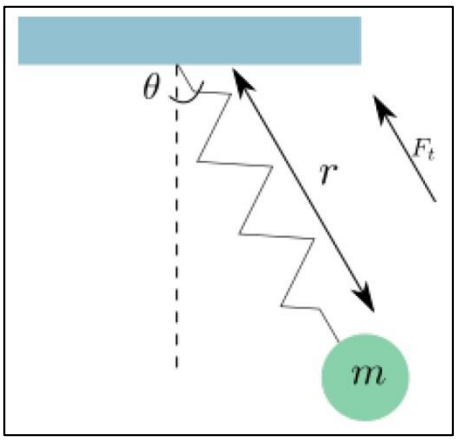
- Péndulo invertido

Realizar una simulación sobre el modelo del péndulo invertido, figura 59 (primer problema de la sección de modelos). Para la simulación hay que tomar en cuenta los siguientes parámetros: la longitud de la varilla es de un metro, la amplitud *A* de la oscilación es de 10 cm. Para la frecuencia de la oscilación en $y(t)$ se deben utilizar varios valores: $\omega = 10rad/s, y\ \omega\ \ 100rad/s\ y\ \omega = 500rad/s$; por lo que se deben de presentar las gráficas de cada una de ellas. Los valores iniciales están dados por $\theta(0) = 0{,}1m\ y\ \theta(0) = 0$.

- Péndulo y resorte

En la figura 62 se muestra un resorte con una constante de Hooke de 170N/m, el cual se encuentra colgando de un punto en una superficie, el resorte se encuentra a un ángulo θde la vertical. La elongación del resorte esta dada por $\delta = r = L$, donde r es la distancia del punto que cuelga el resorte a la posición de una masa de 1,815 Kg al final del resorte; La constante definida en la elongación posee un valor de $L = 15cm$. El resorte presenta una fuerza de pre-tensión $Ft = 5{,}84N$ la cual es constante en la componente radial. Dar las gráficas del comportamiento de $\theta\ y\ r$. Así como el espacio geométrico ocupado por la masa *m*, durante un minuto. Tómese la gravedad como 9,81m/s². Asuma que $\theta = \pi/6$, el resorte no presenta elongación y se encuentra en reposo en $t = 0$.

Figura 62. **Resorte como péndulo**



Fuente: elaboración propia, empleando Inkscape.

- Micrófono capacitivo

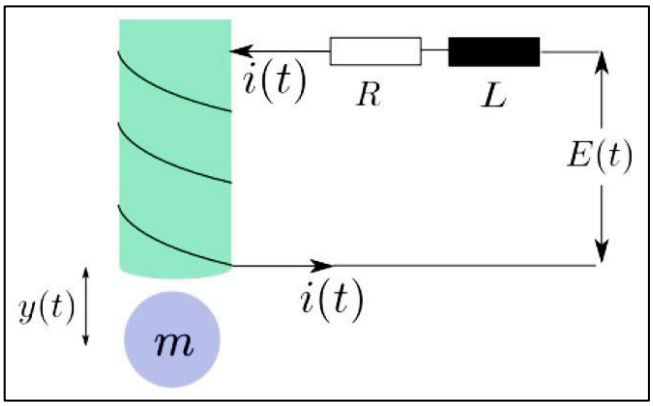
Se tiene un micrófono capacitivo cuyo modelo fue desarrollado anteriormente, dicho modelo considera un conductor rígido en lugar de una membrana que puede variar su forma, tal como un tambor, por lo que realizar una simulación sobre el modelo representará solamente una aproximación a la realidad, sin embargo dará una idea de lo que sucede realmente al usar como transductor de voz un micrófono capacitivo, y explicará la presencia de armónicos. Los parámetros del micrófono son los siguientes: la densidad superficial del plato movible está dada por $\sigma m = 0{,}0445\, Kg/m^2$, la distancia máxima entre los conductores es de $x0 = 2{,}09\, mm$ y los platos son circulares con un radio de 4,45 mm, la constante de Hooke es *k* Æ 3 162 N/m y la constante de viscosidad está dada por $\beta = 1{,}827£10 \times {}^{5}Kg/(m \cdot s)$. El espacio que se encuentra entre los platos está relleno de aire, por tanto la constante de permeabilidad es muy similar a la del vacío por lo que tómese como $\epsilon = 8{,}8541878£10 - {}^{12}$. Si dicho micrófono se conecta en serie con una resistencia

de 1K-. En el modelo se consideran inductancias que aparecen por diversas situaciones las cuales forman 1mH. El circuito es alimentado por una batería de 5V. Ver figura 25. ¿Como sería la corriente en el circuito si el micrófono se ve excitado por una onda sinusoidal a 440 Hz con una presión máxima de 31 Pa. ¿Tiene consistencia el resultado obtenido con la realidad? Considérese que el micrófono inicialmente se encuentra en reposo.

- Suspensión magnética

En la figura 64 se muestra la suspensión magnética de una esfera. La esfera se encuentra hecha de acero y está suspendida en el aire mediante fuerza electromagnética generada por un electroimán. La resistencia de la bobina está dada por $R = 50\Omega$ y la inductancia por $L = L_0/y(t)$, donde $L_0 = 1m(H \cdot m)$. El voltaje aplicado $E(t)$ es una constante $E_0 = 50V$. La fuerza que se opone a la gravedad y es generada por el electroimán está dada por $\mathrm{F} = K\frac{i^2(t)}{y^2(t)}$, donde $K = 0{,}1(Nm)/A$. La masa de la esfera de acero es de cien gramos, $m = 0{,}1Kg$.

Figura 63. **Suspensión magnética**



Fuente: elaboración propia, empleando Inkscape.

Expresar el modelo del sistema de la forma:

$$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}.\boldsymbol{e})$$

[Ec. 2.414]

Donde x es un vector cuyas componentes son las variables de estado $y(t)$, $\dot{y}(t)$ y i (t). Donde f es una función de dicho vector y un vector de entradas **e**; para este ejemplo se tiene la fuerza de gravedad y la tensión de la fuente como las componentes de **e**. En otras palabras,

$$\boldsymbol{x} = \begin{pmatrix} \mathrm{y(t)} \\ \dot{\mathrm{y}}\mathrm{(t)} \\ \mathrm{i(t)} \end{pmatrix} \qquad y \qquad \boldsymbol{e} = \begin{pmatrix} E_0 \\ m\mathbf{g} \end{pmatrix}$$

[Ec. 2.415]

Para poder expresar el sistema como lineal es necesario sustituir f $(\boldsymbol{x}, \boldsymbol{e})$, por una aproximación ˜$f$ $(\boldsymbol{x}, \boldsymbol{e})$. Para este fin es posible expresar f , de la forma:

$$\tilde{f}_i(\boldsymbol{x}, \boldsymbol{e}) \cong f_{\boldsymbol{i}}(\boldsymbol{x}_0, \boldsymbol{e}_0) + \sum_{j=1}^{n} \left.\frac{\partial f_{\boldsymbol{i}}(\boldsymbol{x}, \boldsymbol{e})}{\partial x_j}\right|_{\boldsymbol{x}_0, \boldsymbol{e}_0} \left(x_j - x_{0j}\right) + \sum_{j=1}^{k} \left.\frac{\partial f_{\boldsymbol{i}}(\boldsymbol{x}, \boldsymbol{e})}{\partial e_j}\right|_{\boldsymbol{x}_0, \boldsymbol{e}_0} \left(e_j - e_{0j}\right)$$

[Ec. 2.416]

Luego se realiza un cambio de variable:

$$\hat{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_0 \qquad \text{[Ec. 2.417]}$$

$$\hat{\boldsymbol{e}} = \boldsymbol{e} - \boldsymbol{e}_0 \qquad \text{[Ec. 2.418]}$$

Además, dado que $\dot{\mathrm{x}}_0\ f\ (\boldsymbol{x}_0, \boldsymbol{e}_0)$, se dice que:

$$\frac{d\hat{\boldsymbol{x}}}{dt} = \frac{d\boldsymbol{x}}{dt} - f(\boldsymbol{x}_0, \boldsymbol{e}_0)$$

[Ec. 2.419]

Si se aproxima $\dot{\mathrm{x}}(t)$ con $f\ (\boldsymbol{x}, \boldsymbol{e})$, se tiene:

$$\frac{d\hat{\boldsymbol{x}}}{dt} = A\hat{\boldsymbol{x}} + B\hat{\boldsymbol{e}}$$

[Ec. 2.420]

Donde A es la matriz de entradas:

$$a_{ij} = \left.\frac{\partial f_{\boldsymbol{i}}(\boldsymbol{x}, \boldsymbol{e})}{\partial x_j}\right|_{\boldsymbol{x}_0, \boldsymbol{e}_0}$$

[Ec. 2.421]

Para todo $i\epsilon 2\ \{1, 2, \ldots, n\}$ y todo $j\ \epsilon\ \{1, 2, \ldots, n\}$. B es la matriz de entradas:

$$b_{ij} = \left.\frac{\partial f_{\boldsymbol{i}}(\boldsymbol{x}, \boldsymbol{e})}{\partial e_j}\right|_{\boldsymbol{x}_0, \boldsymbol{e}_0}$$

[Ec. 2.422]

Para todo $i\ \epsilon\ \{1, 2, \ldots, n\}$ y todo $j\ \epsilon\ \{1, 2, \ldots, k\}$. Realizar una aproximación lineal para el sistema de suspensión magnética, para el punto $\boldsymbol{x}_0\ y\ \boldsymbol{e}_0$. En el caso que f sea lineal para **x** o para **e**, no tiene sentido realizar la aproximación para alguno de ellos, como en de este problema en el cual solo se debe considerar la aproximación para $\boldsymbol{x}$ y no para $\boldsymbol{e}$.

Para que una linealización sobre los puntos $\boldsymbol{x}_0\, y\, \boldsymbol{e}_0$ sea válida es necesario que los cambios alrededor de estos puntos sean pequeños, y por esa razón se conocen como puntos de equilibrio. Para el sistema de suspensión magnética considere el punto de equilibrio $\boldsymbol{x}_0$, es el punto en que la esfera se encuentra en reposo; el punto en que la gravedad cancela la fuerza del electro-imán. Considérese otra entrada para agregarla al vector **e**, la cual es 0 en todos los puntos excepto en el instante tp, donde tiene un valor de 0,05*mg* en dirección de la gravedad, dicha fuerza representa a alguien que golpea la esfera en el instante tp con el fin de generar una perturbación. Es posible modelar dicha fuerza como una delta de Kronnecker escalada y desfasada:

$$P(t) = 0{,}05mg\delta(\mathrm{t} - t_p)\mathrm{N}$$

[Ec. 2.423]

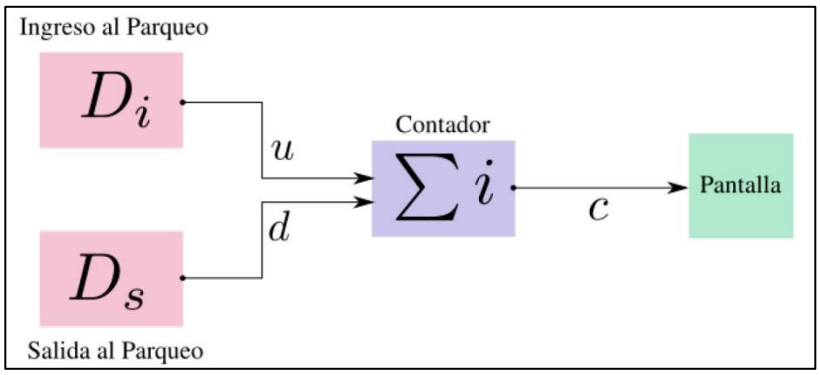
Simular el modelo que se encuentra dado por f y el modelo que se encuentra dado por la aproximación lineal f . Comparar los resultados. ¿Qué se puede concluir con la simulación? ¿Es viable realizar una aproximación para convertir el modelo en lineal? Como sugerencia para la simulación tómese $t_p = 0$.

3. SISTEMAS DISCRETOS

Los modelos de sistemas empotrados por lo general incluyen sistemas discretos y sistemas continuos. Algunos sistemas continuos pueden ser modelados por ecuaciones diferenciales ordinarias, tal como se mostró en el capítulo 2. Sin embargo, esta forma de modelar sistemas discretos no es la indicada, ya que los componentes en un sistema continuo cambian suavemente y en uno discreto lo hacen abruptamente.

Un sistema discreto opera en una secuencia contable de pasos y se dice que tiene una dinámica discreta, en otras palabras, sus señales son discretas o de tiempo discreto. Estos sistemas son comunes en sistemas construidos por el hombre, como circuitos digitales o sistemas de atención al cliente, como un *call center* o la ventanilla de un banco. Un buen ejemplo de estas dinámicas es la automatización de un parqueo en un centro comercial, el cual dispone con un sistema que cuenta el número de automóviles que entran y salen con el fin de llevar un control de cuántos automóviles hay parqueados dentro y muestra a las personas que deseen ingresar a él, cuantos lugares hay disponibles. Un diagrama del funcionamiento general de este sistema se muestra en la figura.

Figura 64. **Caricatura del sistema de conteo de un parqueo**



Fuente: elaboración propia, empleando Inkscape.

El contador de vehículos es un sistema compuesto por pequeños sistemas. Los sistemas $Di\ y\ Ds$ producen señales $u\ y\ d$, las cuales forman entradas de un sistema Contador que son interpretadas, a su vez genera una señal c, que a su vez es la entrada de otro sistema llamado Pantalla y muestra de forma comprensible la señal c. El sistema Contador lleva un conteo activo con base en las señales de entrada *u* y *d*, incrementando en una unidad la variable i cada vez que exista un evento discreto en u o disminuyéndola una unidad cada vez que exista un evento discreto en d. El conteo comienza en un valor inicial i_0 el cual es un parámetro del sistema Contador. Un evento discreto ocurre en un instante en lugar de ocurrir durante un intervalo en el tiempo. En el sistema contador, la señal u es una función de la forma:

$$u: \rightarrow \{ausente, presente\} \qquad \text{[Ec. 3.1]}$$

Lo que significa que para cualquier tiempo $t\ \epsilon\ \mathbb{R}$, la entrada $u(t)$ se encuentra en ausente o presente. El estado, símbolo o valor ausente, como desee tomarse, indica que no hay un vehículo entrando en dicho instante y el

estado, símbolo o valor presente indica que en ese preciso instante ha entrado un vehículo al parqueo. Teniendo un significado similar para la señal *d*, solamente que indicando que el vehículo se encuentra saliendo en lugar de entrando. Cada vez que $u(t)\ o\ d(t)$ sea presente, el sistema Contador realizará un cambio en la variable *i* , y dicho cambio se verá reflejado en un valor entero en la señal *c*, pudiendo así dar la siguiente expresión matemática para *c*,

$$c: \mathbb{R} \rightarrow \{ausente\} \cup \mathbb{Z} \qquad \text{[Ec. 3.2]}$$

Puede verse que para el sistema Contador no hay necesidad de hacer algo cuando ambas entradas se encuentran ausentes. El contador necesariamente necesita operar cuando alguna de las entradas está en presente, la mayor parte del tiempo las variables pasarán en ausente ya que la salida o entrada de un vehículo se considera instantánea, por tanto se verán cambios en la salida del sistema abruptos, por decirlo de algún modo y solamente cuando suceda un evento ligado a la entrada o salida de un vehículo. Dicho comportamiento hace a este sistema una dinámica discreta.

Hasta ahora no se ha dado una definición formal del tiempo discreto y solo se ha hecho referencia que las señales discretas se mantienen ausentes la mayor parte tiempo. La tabla LXVII da consistencia al significado de una señal discreta.

Tabla LXVII. **Definición de señal discreta y tiempo discreto**

Sea *e* una señal de la forma: $e: \mathbb{R} \rightarrow \{ausente\} \cup X$ [Ec. 3.3] Sea $T \subseteq \mathbb{R}$ el conjunto de instantes donde *e* no es ausente. Dicho de otra forma: $T = \{t \in \mathbb{R}: e(t) \neq ausente\}$ [Ec. 3.4] Entonces *e* es discreta si y solo si existe una función inyectiva $f: T \rightarrow N$, tal que para todo $t_1, t_2 \epsilon\, T$, se tiene que $t_1 \leq t_2$ implica que $f\,(t_1) \leq f\,(t_2)$. En otras palabras T preserva el orden. El conjunto T es llamado tiempo discreto.

Fuente: elaboración propia.

En la tabla LXVII, la existencia de la función uno a uno (inyectiva) asegura que se pueden contar los eventos en un orden temporal. La dinámica de un sistema discreto puede ser descrita como una secuencia de pasos que es llamada reacción. Cada una de ellas se asume instantánea. Las reacciones de un sistema discreto son disparadas por el entorno en que opera el sistema discreto.

Considérese un sistema discreto con señales de entrada y salida, se le llama puerto a una variable que toma el valor de una señal discreta en un momento dado, representando dentro del sistema a la señal de entrada o salida en ese instante. Por lo general los puertos reciben el mismo nombre que la señal que representan.

Tabla LXVIII. **Definición de valuación**

Sea P un conjunto de puertos dado, tal que $P = \{P_1, \dots, Pn\}$; para cada puerto $p \in P$ existe un conjunto Vp , llamado el tipo de p, al momento de tener una reacción se dice que cada puerto es una variable con un valor en el conjunto $Vp \cup \{ausente\}$. Una valuación, es una asignación para cada $pn \in P$de un valor en Vp , o una aseveración que pn se encuentra ausente.

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 115.

Los circuitos digitales por naturaleza son sistemas discretos, operan en una secuencia contable de pasos determinados por señales en forma de flancos generados por el entorno. Estos flancos representan eventos y los circuitos de lógica digital responden a ellos. Los modelos de estos sistemas son más cualitativos que cuantitativos, ya que en lugar de plantear una ecuación diferencial y asociar variables al estado del sistema, por ejemplo, la velocidad o el momento, este último en dinámicas discretas se describe utilizando algún lenguaje.

3.1. Máquinas de estados

Intuitivamente, el estado de un sistema es su condición en un instante dado. En general, el estado afecta como el sistema reacciona a las entradas, es posible decir que el estado es un resumen del pasado.

Para el ejemplo del sistema Contador, el estado $s(t)$ en un tiempo dado *t* es un entero, tal que el espacio de estados $Estados \subset \mathbb{Z}$. Una implementación práctica de dicho sistema tiene un número positivo conjunto finito *M* de estados.

Por lo que el espacio de estados puede ser considerado como

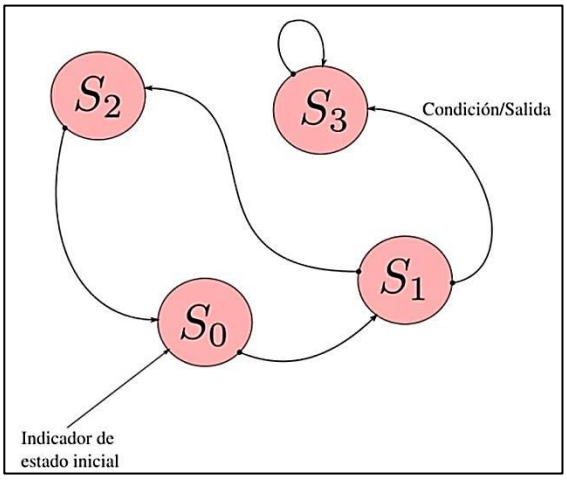
$$Estados = \{0, 1, 2, \dots, M\} \qquad \text{[Ec. 3.5]}$$

Una máquina de estados es un modelo de un sistema con dinámicas discretas que a cada reacción asocia valuaciones de los puertos de entrada a valuaciones de los puertos de salida donde la asociación puede depender del estado actual. Una máquina de estados finitos, FSM (por sus siglas en inglés, *Finite-StatesMachine*) es una máquina de estados donde el conjunto de Estados formado por los posibles estados de la dinámica es conjunto finito. Si el conjunto de estados es razonablemente pequeño es posible representar la dinámica con un grafo como el que se muestra en la figura 65, donde cada estado se representa por una burbuja, por ejemplo, para la figura 65 el conjunto de estados esta dado por:

$$Estados = \{S^0, S^1, S^2, S^3\} \qquad \text{[Ec. 3.6]}$$

Al comienzo de cada secuencia de reacciones, existe un estado inicial, tal como se muestra el estado S_0 en la figura. Para indicar que un estado es el comienzo; en este tipo de grafos se usa una flecha señalando al estado.

Figura 65. **Máquina de estados finitos**

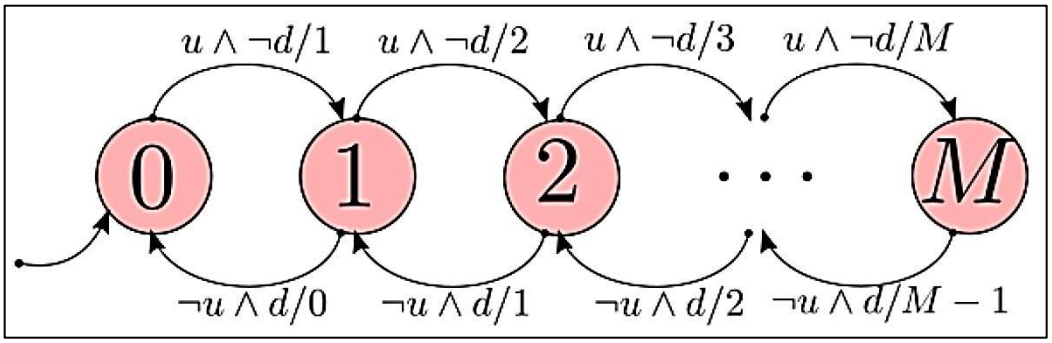


Fuente: elaboración propia, empleando Inkscape.

Las transiciones entre estados gobiernan la dinámica discreta de la máquina de estados, así como la asociación de valuaciones de las entradas con valuaciones de las salidas. Una transición se representa con una flecha curva, yendo de un estado a otro; también puede comenzar y terminar en el mismo estado. En la figura 65 se muestra una etiqueta en la transición de $S_1\ a\ S_3$, la condición determina cuando la transición tiene lugar como una reacción y la salida es el conjunto de valores generados por cada reacción.

Una condición es una sentencia que es verdadera cuando la transición tiene lugar, cambiando de estado. Cuando una transición tiene lugar se dice que se encuentra habilitada, si la transición no está habilitada dicha sentencia evalúa falso. Una acción es una asignación de valores (incluido ausente) a los puertos de salida. Cualquier puerto de salida no mencionado en una transición es tomado como ausente. Implícitamente todas las salidas son tomadas como ausente. En la figura se muestra una máquina de estados finitos para el contador del parqueo. Los puertos que se están tomando como entradas son *u* y *d* cuyo tipo está dado por el conjunto {presente, ausente}, y el puerto que se toma como salida es *c*, cuyo tipo está dado por el conjunto $\{0, 1, \ldots, M\}$.

Figura 66. **FSM para implementación del contador en el parqueo**



Fuente: elaboración propia, empleando Inkscape.

Para la figura el conjunto Estados $= \{0, 1, 2, \ldots, M\}$, indica los estados de la dinámica discreta. La transición del estado 0 al 1 tiene lugar cuando se cumple la condición $\wedge\neg d$. Las señales que pueden tomar solamente dos valores, presente y ausente, se conocen como puras, ya que llevan implícita toda la información de un fenómeno en dicho conjunto de valores. Dado que este tipo de señales toman solamente dos valores, para modelos y algunas implementaciones puede tomarse al valor presente como verdadero y ausente como falso, siendo viable modelar las reacciones expresadas con señales puras con el álgebra de Boole.

En la sección 3.3 se dará más detalle de este tipo de sentencias. Por tanto, la sentencia anterior será cierta únicamente cuando *u* se encuentre en presente y *d* en ausente, y por tanto se llevará a cabo la transición de 0 a 1 y colocando 1 en el puerto *c*. La salida del puerto *c* no se encuentra explícitamente nombrada porque solo hay un puerto de salida.

El entorno determina cuando una máquina de estados debe reaccionar. Existen varios mecanismos de accionar una máquina de estados, entre ellos destaca el accionamiento por eventos y el accionamiento por tiempo. Cuando el entorno determina que una máquina de estados debe reaccionar, las entradas tendrán una valuación. El estado de la máquina asignará una valuación a los puertos de salida y posiblemente cambie a un nuevo estado. Si ninguna condición para la transición es cierta en el estado actual la máquina permanecerá en el mismo estado. Es posible para todas las entradas encontrarse ausentes en una reacción. Aún en este caso, puede ser posible para una condición evaluar verdadero, y en ese caso la transición se llevará a cabo para dicha condición.

Una máquina de estados provee un robusto modelo de un sistema discreto, si se trata de construir uno permite la manipulación automatizada del comportamiento de un sistema, con base en las señales generadas por el entorno en que se encuentra. Presenta un modelo para la automatización de un sistema debido a que, dependiendo de las entradas generadas por el sistema, genera transiciones y salidas adecuadas al estado de un sistema, colocando en uno nuevo al sistema. Debido a esta razón las máquinas de estados también son conocidas como autómatas.

- Funciones de actualización

Los grafos para el modelo de máquinas de estados finitos son ideales cuando el número de estados es pequeño. Es necesaria una notación más general cuando el número de estados es muy grande, incluso si no son finitos. Las funciones de actualización vienen a satisfacer esta necesidad, dichas funciones son una notación matemática con el mismo significado que el grafo. En otras palabras, es una referencia a una definición formal de una máquina de estados.

Tabla LXIX. **Definición de máquinas de estado**

Una máquina de estados es una tupla de compuesta de 5 elementos:

$$(Estados, Entradas, Salidas, Actualización, EstadoInicial) \quad \text{[Ec. 3.7]}$$

Donde

- Estados es un conjunto de estados
- Entradas es un conjunto de valuaciones
- Salidas es un conjunto de valuaciones
- Actualización una función de la forma $Estados \times Entradas \rightarrow Estados \times Salidas$, la cuál asigna a cada estado y valuación en las entradas únicamente un siguiente estado y una valuación en las salidas
- Estado Inicial es el estado donde comienza la máquina de estados

Fuente: LEE, Edward; SEESHIA, Sanjit. *Introduction to embedded systems.* p. 119.

Una máquina de estados finitos tiene un comportamiento acorde a un conjunto de reacciones. Para cada reacción la máquina de estados tiene un estado actual, y cada reacción puede generar una transición a un estado siguiente, el cual será el estado actual en la siguiente reacción. Es posible contar estos estados comenzando con el 0 para el estado inicial. Específicamente sea $s : N \rightarrow Estados$ una función que da el estado de la máquina de estados a la reacción $n \epsilon N$. Inicialmente $s(0) = Estado\ Inicial$. $Sea\ x : N \rightarrow Entradas$ y $y : N \rightarrow$ Salidas las valuaciones en cada reacción de las entradas y las salidas respectivamente. Por tanto $x(0) \epsilon\ Entradas\ y\ y(0) \epsilon$ Salidas representan las primeras valuaciones.

$$\big(s(c+1), y(n)\big) = Actualización\ \big(s(n), x(n)\big)$$

[Ec. 3.8]

Esto da el siguiente estado y la salida en términos del estado actual y la entrada. La función de actualización representa todas las transiciones, condiciones y salidas en la máquina de estados. También puede utilizarse el término función de transición.

Las valuaciones de las entradas y las salidas también tienen una forma matemática formal. Supóngase que una máquina de estados finitos tiene los puertos de entrada $P = \{p_1, \ldots, PN\}$, donde cada $p \epsilon P$ tiene un tipo correspondiente Vp . Entonces Entradas es el conjunto de funciones de la forma:

$$i = P \rightarrow V_{p_1} \cup V_{p_2} \ldots \cup V_{p_M} \cup \{ausente\}$$

[Ec. 3.9]

Donde para cada $p \epsilon P, i\,(p) \epsilon \cup Vp\,[\{ausente\}$ da el valor del puerto p. Entonces así una función $i \epsilon Entradas$ es una valuación para los puertos de entrada. Una analogía similar sigue para las valuaciones de los puertos de salida.

$$o = Q \rightarrow V_{q_1} \cup V_{q_2} \ldots \cup V_{q_M} \cup \{ausente\}$$

[Ec. 3.10]

Donde $Q = \{q_1, \ldots, qM\}\, y\ cada\ q \epsilon Q$ tiene un tipo correspondiente Vq.

Por ejemplo, para la máquina de estados del contador puede ser representada matemáticamente como sigue:

$$Estados = \{0, 1, \ldots, M\}$$

[Ec. 3.11]

$$Entradas = (\{u, d\} \rightarrow \{presente, ausente\})$$

[Ec. 3.12]

$$Salidas = (\{c\} \rightarrow \{0, 1, \ldots, M, ausente\})$$

[Ec. 3.13]

$$EstadoInicial = 0$$

[Ec. 3.14]

Y la función de actualización dada por:

$$Actualización\,(s, i) = \begin{cases} (s+1, s+1) & Si\ s < M \wedge i(u) = presente \wedge i(d) = ausente \\ (s-1, s-1) & Si\ s > 0 \wedge i(u) = ausente \wedge i(d) = presente \\ (s, ausente) & De\ otro\ modo \end{cases}$$

[Ec. 3.15]

Para todo $i \in Entradas\ y\ s \in Estados$. Nótese que una valuación para la salida $o \in Salidas$ es una función de la forma $o : \{c\} \rightarrow \{0, 1, \ldots, M, ausente\}$. En la ecuación 3.10 la primera alternativa da la valuación de la salida como $o(q) = s + 1, donde\ q \in \{c\}$. Cuando hay varios puertos de salida es necesario ser más explícito en que puerto se está asignando un valor determinado. Tanto i como *o* son valuaciones, en cada estado.

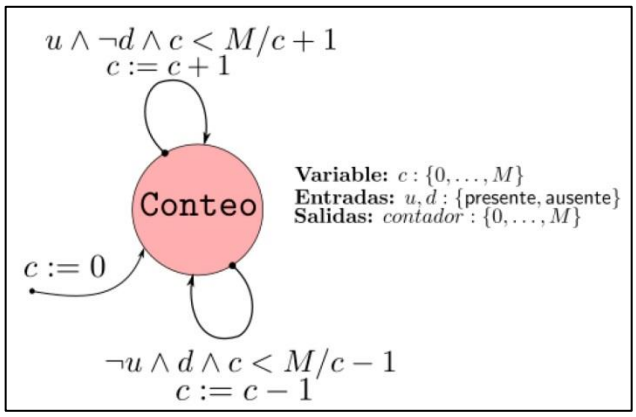
Si el conjunto de estados es conjunto finito, se tienen dos opciones para la definición de una máquina de estados ya sea como la tupla de la definición 31 o como un grafo que contiene un conjunto finito de nodos y conjunto finito de flancos etiquetados de alguna manera. Ambos pudiendo así modelar la dinámica de un sistema, usando la que convenga para una determinada situación.

3.1.1. Máquinas de estados extendidas

La notación de máquinas de estados resulta muy complicada cuando el número de estados se torna demasiado grande. El ejemplo del contador de autos en un parqueo refleja este punto. Si *M* es demasiado grande, el grafo sería extenso a pesar que se puede resumir dicha información de otra manera.

Para resumir dicha información se utiliza una máquina de estados extendida, la cual modela con variables que pueden modificarse e interpretarse a lo largo de las transiciones. Para el ejemplo del parqueo la máquina de estados que define su dinámica puede ser representada de forma más compacta. Tal como se muestra en la figura 67.

Figura 67. **Máquina de estados extendida para el contador**



Fuente: elaboración propia, empleando Inkscape.

Si se toma una variable *c*, declarada explícitamente para evitar que se confunda con el puerto de salida. La transición indicando el estado inicial indica que esta variable tiene un valor de cero al inicio. La transición superior hacia el mismo estado tiene lugar cuando *u* se encuentra en presente, *d* en ausente y la variable *c* es menor que *M*. Cuando esta transición toma lugar la máquina produce una salida *contador* con el valor $c + 1$, y el valor de *c* se incrementa en 1. La transición inferior hacia el mismo estado tiene lugar cuando *u* se encuentra en ausente, *d* en presente y la variable *c* es mayor que 0. Al tener lugar dicha transición la máquina de estados produce una salida con un valor de $c - 1$ y disminuye en uno el valor de *c*. Nótese que *M* es un parámetro y no una variable. Se toma como una constante durante la ejecución de las reacciones en la máquina de estados.

Una máquina de estados extendida se diferencia de una máquina de estados finitos en que muestra explícitamente las declaraciones de variables para hacer más fácil determinar cuando un identificador en la condición o la acción o la entrada o la salida hace referencia a la variable. Las variables que

han sido declaradas pueden tener un valor inicial, el cual se muestra en la transición hacia el estado inicial. Las transiciones tendrán la forma:

Condición/ Salida
Conjunto de acciones

La notación es similar a una máquina de estados finitos, con una condición y una salida en cada transición, pero con la diferencia que existe un conjunto de acciones a ejecutar luego de evaluar una condición y antes de habilitar la transición. El estado de una máquina extendida de estados incluye no solo la información acerca del estado discreto en el que se encuentra la máquina sino también sobre los valores de cualquier variable. El número de posibles estados en este tipo de modelos puede ser muy grande, incluso hasta infinito. Las máquinas de estados extendidas pueden ser o no máquinas de estados finitos y en particular es común que tengan un número de estados muy grande.

3.1.2. Máquinas de Mealy y máquinas de Moore

Las máquinas de estados descritas hasta este momento son conocidas como máquinas de Mealy, nombradas de esa manera en honor de George H. Mealy. Son caracterizadas por producir salidas cuando una transición toma lugar. Una alternativa es conocida como máquina de Moore, produce salidas cuando la máquina se encuentra en un estado, ya que la transición fue ejecutada. Entonces, la salida se encuentra definida por el estado actual y no por la transición. Las máquinas de Moore son nombradas así en honor al ingeniero Edward F.Moore.

La diferencia entre ambas máquinas es casi imperceptible pero es muy importante. Ambas son sistemas con dinámicas discretas, y por tanto la operación de ambas consisten en una secuencia de reacciones discretas.

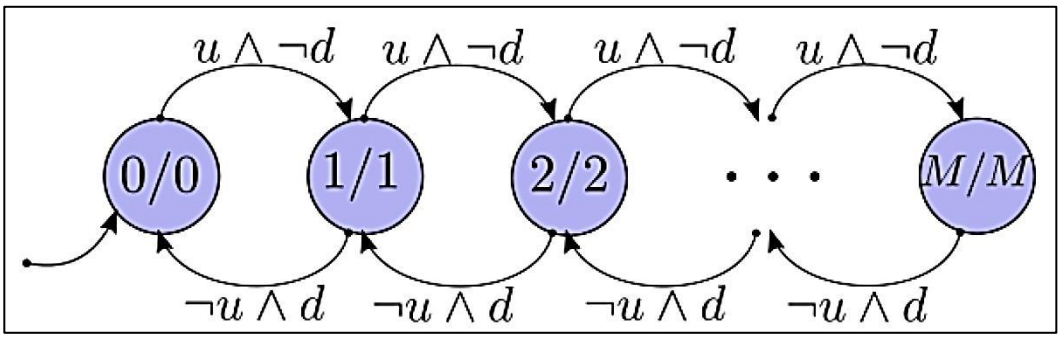
Para una máquina de Moore, a cada reacción, la salida se encuentra definida por el estado actual. La salida a un tiempo de reacción no depende de la entrada en ese tiempo, haciendo a este tipo de máquinas estrictamente causales. Una máquina de Moore cuando reacciona siempre reporta la salida asociada con el estado actual. Una máquina de Mealy no produce alguna salida hasta que haya sido explícitamente declarada en la transición. Cualquier máquina de Moore puede ser convertida en un equivalente de Mealy. Una máquina de Mealy puede ser convertida en una máquina de Moore casi equivalente, con la diferencia que en la salida es producida en la siguiente reacción y no la actual. Las máquinas de Mealy tienden a ser más compactas y producen una salida casi instantánea que responde a la entrada.

Una versión de Moore para el contador del parqueo se muestra en la figura 69, la cual usa una notación similar:

$$Estado/Salida$$

La notación siempre utiliza la diagonal para indicar la salida, solamente que en lugar de indicarla en la transición se hará en el estado.

Figura 68. **Máquina de Moore para ejemplo del contador**



Fuente: elaboración propia, empleando Inkscape.

Nótese que esta máquina de Moore no es equivalente a la de Mealy planteada anteriormente, por ejemplo, que en la primera reacción se tiene $que\ u = \ presente\ y\ d = \ ausente$, la salida en ese instante será 0 en la figura 68 y 1 en la 67. La salida de una máquina de Moore representa el número de automóviles en el parqueo en el instante de llegada de un nuevo automóvil, no el número de automóviles después de la llegada del nuevo automóvil. La salida de una máquina de Moore depende únicamente del estado actual, y la salida de una máquina de Mealy depende tanto del estado como las entradas en un instante.

3.2. Sistemas de numeración binaria y hexadecimal

Un número es una idea, que expresa cantidad en relación a su unidad. Los números naturales son los más simples y se usan para contar, por más simple que parezca esta acción es muy poderosa y engloba a cualquier sistema discreto. Para comunicarse y realizar abstracciones es necesario representar los números de determinada manera, y para esto se utiliza un sistema de

numeración, que no es más que un conjunto de reglas y símbolos para dar una representación única a cada número.

Conforme avanzó la historia, la humanidad prefirió los sistemas de numeración posicionales sobre los sistemas no posicionales, tal como el sistema de numeración utilizado por los antiguos romanos. Los sistemas posicionales utilizan un conjunto de b símbolos permitidos, la cantidad de símbolos *b* se conoce como base del sistema y es la cantidad de cosas que es posible asociar a un solo elemento del conjunto para representar dicha cantidad; el valor de cada símbolo o dígito depende de su posición relativa que se determina por la base. Ejemplo de ello es el sistema decimal, en donde cada posición equivale a una potencia de diez; el conjunto de símbolos permitidos en el sistema decimal $S_{10} = \{0,1,2,3,4,5,6,7,8,9\}$ sirve para relacionar una determinada cantidad de potencias de diez. Donde la potencia determina la posición en que se encuentra. Por ejemplo, el número 165 representa 1 ciento de unidades más seis decenas de unidades más cinco unidades. De igual forma es posible escribir dicho número utilizando solamente tres símbolos $S_3 = \{0,1,2\}$. Donde la posición indica una potencia de tres.

$$165_{10}=2*3^4+0*3^3+0*3^2+1*3^1+0*3^0=20\ 010_3^2$$

[Ec. 3.16]

Si se reduce el conjunto de símbolos permitidos a dos elementos y se construye un sistema de numeración basado en este conjunto de dos elementos recibirá como nombre sistema binario, el cual tiene como base un conjunto $S_2 = \{0,1\}$, y a cada dígito o posición de un número escrito en este sistema se le conoce como bit. Por ejemplo, el número 9 710 es posible desglosarlo de la siguiente forma utilizando potencias de base dos.

$$1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 1100001_2 = 97_{10}$$

Es posible solamente con dos elementos escribir cualquier número natural, tal como es posible escribirlos con tres o diez elementos, y dado a ello al construir dispositivos, se optó por asociar estos símbolos a los valores presente y ausente de una señal pura.

De esta forma es posible representar cualquier número con señales simples que provienen del entorno o del mismo sistema. De esta manera es que las computadoras procesan información, convirtiendo números a señales de voltaje o corriente y a este proceso se le llama codificación.

Para codificar números se asocia un “1” a determinado valor de una señal pura y “0” a otro valor. El problema de utilizar el sistema binario, por el hecho de ser el más simple de todos, es que para escribir los números se necesitan más posiciones que en los otros sistemas. Por ejemplo, el número 309 410 utiliza solamente cuatro posiciones en el sistema decimal pero su representación en el sistema binario, 110 000 010 110_2, requiere doce posiciones. Para lidiar con este problema se optó por un sistema de dieciseis símbolos conocido como hexadecimal, solamente para representación en diagramas, modelos y es exclusivamente para uso de los humanos, las computadoras siguen utilizando dos señales; dicho sistema es compatible, por decirlo de alguna manera, con el sistema binario, ya que dieciséis es la cuarta potencia de dos, lo que significa, que cualquier combinación de cuatro bits pueden ser asociada a un símbolo del conjunto $S_{16} = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$ y cualquier elemento de S_{16} se puede representar con un conjunto de cuatro bits, y cualquier elemento de S_{16}puede representar a un conjunto de 4 bits.

En el ejemplo anterior el número 110000010110_2 es posible seccionarlo en cuartetos y representar cada cuarteto con un elemento de S_{16}, $1100\ 0001\ 01102 = C16_{16}$. Siendo el símbolo $C\ \epsilon\ S_{16}$ el utilizado para representar al cuarteto 1100_2, el símbolo $1\ \epsilon\ S_{16}$ para el cuarteto 0001_2 y el símbolo $6\ \epsilon\ S_{16}$ para el cuarteto 0110_2. Es frecuente al escribir o leer código de bajo nivel escribir los números en hexadecimal con la forma $0xC16$ en lugar del utilizar el subíndice 16.

3.3. Lógica binaria

Los filósofos griegos en búsqueda de la verdad fueron los primeros en crear un sistema lógico binario, llamado lógica proposicional. Una proposición, para fines prácticos, es cualquier sentencia que puede ser solamente o verdadera o falsa, por ejemplo: Cinco es un número primo.

Proposiciones complejas pueden ser formadas con otras proposiciones más simples enlazadas por conectivos lógicos. Por ejemplo: Cinco es un número primo y diez es un número par, es una sentencia compuesta por dos sentencias atómicas, cada una pudiendo tomar un valor de Verdadero o Falso, pero la sentencia completa tomará un valor de verdadero o falso dependiendo de los valores de las sentencias enlazadas por el conectivo “y”. En este ejemplo la sentencia será verdadera.

Eventualmente se saltó de una lógica proposicional a una lógica simbólica. En el siglo XVII Gottfried Leibniz, funda los principios de la lógica simbólica en su trabajo *Calculus ratiocinator*, el cual fue un anticipo a la lógica matemática o una lógica algebraica. A pesar de que este trabajo fue el primero de este tipo, era desconocido para una gran comunidad y muchos de los avances hechos

por Leibniz fueron alcanzados de nuevo por matemáticos como George Boole y Augustus De Morgan, más de un siglo después.

Sea $p \in \{Verdadero, Falso\}$ una proposición. En una formulación más formal de la lógica los conectivos lógicos son considerados operadores de las proposiciones y son conocidos como operadores lógicos. Se mencionan a continuación tres operadores lógicos básicos.

Negación. Es un operador lógico de la forma.

$$\neg: \{Verdadero, Falso\} \rightarrow \{Verdadero, Falso\}$$

[Ec. 3.17]

Si se aplica el operador: a una proposición p tal que $p = Verdadero$, se tendrá como resultado una proposición $q = \neg(p) = \neg p = Falso$. Si se le aplica a una proposición $p = Falso$, se tendrá como resultado una proposición $q = \neg(p) = \neg p = Verdadero$. A esta operación se le conoce también como complemento, debido a que una proposición p es un elemento del conjunto $\{Verdadero, Falso\}$, al momento de aplicarle a la proposición el operador: se obtiene el complemento de la proposición vista como un conjunto, $\{p\}^c$.

Conjunción. Es un operador lógico de la forma.

$$\wedge: \{Verdadero, Falso\} \times \{Verdadero\ Falso\} \rightarrow \{Verdadero\ Falso\}$$

[Ec. 3.18]

Si se tiene un par de proposiciones (p, q), entonces la proposición $\wedge(p, q) = p \wedge q$ será verdadera solamente si ambas proposiciones son verdaderas. Si alguna de ellas fuera falsa la sentencia completa sería falsa.

Disyunción. Es un operador lógico de la forma.

$$\wedge: \{Verdadero, Falso\} \times \{Verdadero\ Falso\} \rightarrow \{Verdadero\ Falso\}$$

[Ec. 3.19]

Si se tiene un par de proposiciones (p, q), para que la proposición $\vee(p, q) = p \vee q$ sea verdadera basta que alguna de las dos sea verdadera. Para que la proposición $p \vee q$ sea falsa es necesario que ambas sean falsas.

Es posible enlazar varias veces sentencias y operadores para construir expresiones cada vez más complejas. A continuación, se muestran algunas tablas, conocidas como tablas de verdad con el resultado de aplicar los tres operadores anteriores a una proposición o un par de proposiciones dependiendo del caso:

Tabla LXX. **Tablas de verdad para operadores básicos**

p	$\neg p$
Verdadero	Falso
Falso	Verdadero

(a) Negación.

p	q	$p \wedge q$
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

(b) Conjunción.

p	q	$p \vee q$
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

(c) Disyunción.

Fuente: elaboración propia.

Existe un operador muy especial que vale la pena tratar, el operador Disyunción exclusiva. Dicho operador puede escribirse como una expresión utilizando solamente los tres operadores básicos mencionados con anterioridad. La Disyunción exclusiva es un operador de la forma.

$$\bigoplus: \{Verdadero, Falso\} \times \{Verdadero\ Falso\} \rightarrow \{Verdadero\ Falso\}$$

[Ec. 3.20]

Pudiendo utilizarse el símbolo $\bigoplus o \veebar$, para representar dicha operación entre proposiciones. Al aplicar el operador $\bigoplus o \veebar$, independientemente de cual se utilice, a un par de proposiciones (p, q) se tiene como resultado una proposición tal que $\bigoplus (p, q) = p \bigoplus q = (p \vee q) \wedge \big(\neg(p \wedge q)\big)$, teniendo como tabla de verdad la tabla LXXI.

Tabla LXXI. **Disyunción**

p	q	$p \vee q$
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Fuente: elaboración propia.

Es común representar los valores de Verdadero y Falso con 1 y 0 respectivamente, dando origen al álgebra de Boole que es una estructura algebraica que esquematiza un conjunto de operaciones lógicas. Existe una conexión entre la lógica binaria y la aritmética, ya que cualquier número se puede representar con 1 y 0. Dicha conexión es la que permite que las computadoras puedan sumar, multiplicar e incluso dividir. Por simples que

parezcan estas operaciones para los humanos no fue hasta 1930 que Claude Shannon se dio cuenta que se podían aplicar el conjunto de reglas del álgebra booleana a circuitos eléctricos, luego con la llegada del transistor se pudo disminuir el tamaño de estos circuitos notablemente.

La lógica binaria se manifiesta físicamente en la electrónica digital. El flujo de pulsos eléctricos puede ser representado por medio de bits, y puede ser controlado por una combinación de dispositivos electrónicos. El diseño de electrónica digital escapa del alcance de este texto, pero es necesario estar consciente de que es posible realizar operaciones aritméticas y lógicas utilizando dispositivos electrónicos.

3.4. Procesadores embebidos

Un procesador es cualquier dispositivo electrónico capaz de ejecutar operaciones lógicas y aritméticas. Cuando un procesador forma parte de un sistema mecánico o eléctrico, se dice que es un procesador embebido. Además, debido a que dichos dispositivos en sus orígenes eran de un considerable tamaño, abarcando varias habitaciones, a medida que fueron disminuyendo su tamaño recibieron el nombre de microprocesadores, el cuál puede parecer un tanto irónico ya que la capacidad de estos comparados con sus ancestros es abismalmente mayor. A continuación, se presentan una serie de procesadores que es posible encontrar en sistemas físicocibernéticos. Se mencionan solamente aquellos basados en tecnologías digitales:

- Microcontroladores: un microcontrolador es una pequeña computadora con distintos periféricos y unidades de memoria integrados en un mismo empaquetado. Se encuentra constituido principalmente por una CPU simple, unidades de memoria, dispositivos de entrada y salida,

temporizadores, entre otros periféricos. Más de la mitad de las computadoras vendidas actualmente son microcontroladores aunque esto es difícil de decir porque no existe mayor diferencia entre un microcontrolador y una computadora de propósito general. Los Microcontroladores más simples operan en palabras de 8 bits y están destinados para aplicaciones que requieren pequeñas cantidades de memoria y funciones lógicas simples. Los Microcontroladores, por su naturaleza de estar destinados para sistemas empotrados, de preferencia deben gastar pequeñas cantidades de energía, la mayoría de ellos viene con un modo de hibernación que reduce la potencia consumida a nanowatts. Muchos sistemas embebidos como una red de sensores o dispositivos de vigilancia han demostrado que pueden operar durante años utilizando pequeñas baterías.

- DSP: muchas aplicaciones embebidas requieren un procesamiento de señales ya que algunas de ellas pueden requerir uso de vídeo, audio o filtros para análisis de sensores entre otros requerimientos. Los DSP son procesadores con arquitecturas diseñadas para el procesamiento de señales. Los primeros DSP integrados aparecieron cerca de los años ochenta, comenzando con el *Western Electric DSP1* construido por Bell Labs, el S28211 de AMI, el TMS32010 de *Texas Instruments* entre otros.

  Luego aparecieron aplicaciones con estos dispositivos como módems, sintetizadores de voz y controladores para audio, gráficas y discos. Los DSP incluyen una unidad de hardware multiacumulativo, variantes de la arquitectura Harvard (para soportar múltiples datos simultáneamente y programas de búsqueda) y modos de direccionamiento que soportan autoincremento, *buffers* circulares y direccionamiento bit*reversed* y algunos soportan hasta cálculo de FFT. Los DSP son difíciles de

programar comparados con las arquitecturas RISC, principalmente por las funciones altamente especializadas y arquitecturas de memoria asimétricas. Aún hoy en día, los programas en C hacen un uso extensivo de librerías que se encuentran *hard-coded* en *assembly* para tomar ventaja de las características más esotéricas de estas arquitecturas.

- PLC**:** recibe su nombre de sus siglas en inglés, *programmable logic controller*. Es un microcontrolador especializado para automatización industrial. Tuvieron su origen como reemplazos para circuitos de control que utilizaban releés eléctricos para manipular maquinaria. Están diseñados para operar en entornos hostiles, como aquellos con altas temperaturas presentes, entornos húmedos o con mucho polvo. La forma más común de programar un PLC es usando una lógica en escalera, una notación por lo general usada para la construcción de circuitos lógicos usando interruptores y releés. Una notación común es representar un contacto por dos barras verticales, y una bobina por un círculo. Actualmente los PLC no son más que Microcontroladores puestos en paquetes resistentes con interfaces I/O diseñadas para aplicaciones industriales. La lógica de escalera es una notación gráfica para la realización de programas.

- FPGA: recibe su nombre de sus siglas en inglés *field programmable gate array*. Es un dispositivo lógico de dos dimensiones de celdas lógicas y *switches* programables. Estos dispositivos pueden configurar un circuito de electrónica digital, dentro de ellos. Su modo de operar es muy distinto al de los Microcontroladores, DSP o PLC. En lugar de constituirse por una arquitectura de una unidad central de procesamiento y memoria para almacenamiento y ejecución de instrucciones, se compone de tablas de búsqueda y un arreglo de transistores lo que permite sintetizar cualquier

circuito de electrónica digital dentro de ella, incluidos los microprocesadores. A pesar de su relativamente reciente salida al mercado, han empezado a ganar terreno en los sistemas embebidos, debido a su tremenda flexibilidad de prácticamente volverse cualquier circuito electrónico, permitiendo un gran paralelismo en las aplicaciones, aún su precio con relación a los Microcontroladores es mucho mayor.

Todos los dispositivos mencionados con anterioridad, a excepción de las FPGA, en principio son computadores y se constituyen básicamente de una unidad central de procesamiento CPU (siglas en inglés de *central processing Unit*), memoria y periféricos. Algunos de ellos con arquitecturas más sofisticadas que las de otros, pero todos cumplen en principio con tener estos tres elementos. Todos los dispositivos con estos elementos son capaces de almacenar grandes cantidades de datos, y ejecutar programas a gran velocidad. Los programas son conjuntos de instrucciones almacenados en la memoria que son ejecutados de forma compleja pero bien definida. Una computadora siempre ejecuta los programas de la misma forma, siguiendo las instrucciones al pie de la letra. Las primeras computadoras eran gigantescas, conforme fue avanzando el tiempo, disminuyeron su tamaño y aumentaron su capacidad. Con el tiempo se comenzó a llamarlas microcomputadoras por el tamaño que poseen en relación a sus antecesoras.

3.5. Microcontroladores

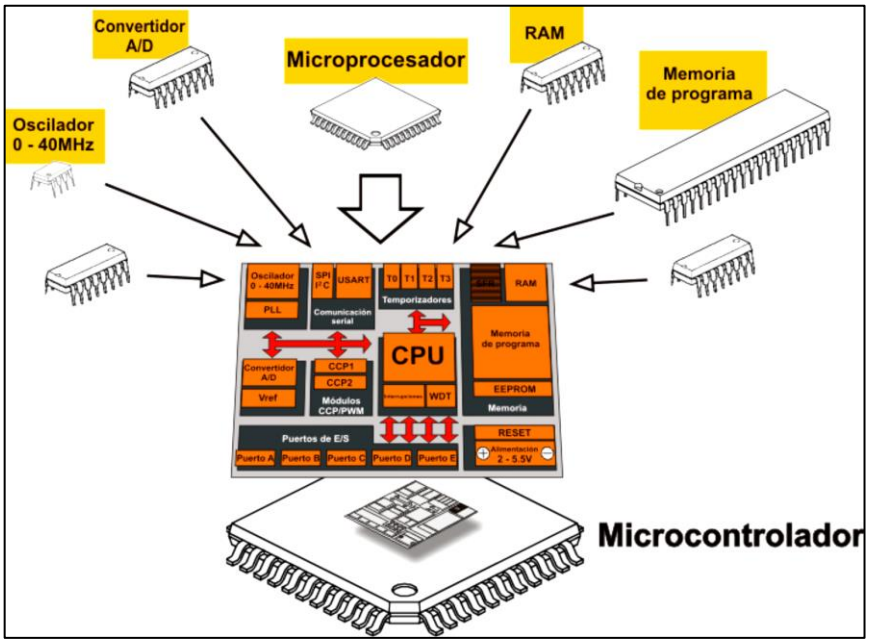
El primer microcontrolador apareció en el mercado en 1974. Se llamaba TMS 1000 y fue desarrollado por Texas Instruments. El dispositivo combinaba una memoria solo de lectura, una memoria de lectura y escritura, un procesador y un reloj; todo en un mismo *chip* y se encontraba enfocado para sistemas embebidos. Actualmente el mercado de los Microcontroladores es muy variado

y es posible encontrar Microcontroladores para *hobbistas* con grandes comunidades en línea para algunos de ellos, tal como es el caso de la plataforma de desarrollo Arduino.

3.5.1. Estructura de un microcontrolador

Actualmente en computadoras de propósito general, la variedad de arquitecturas se encuentra limitada con la arquitectura Intel x86 dominando el mercado.

Figura 69. **Estructura de un microcontrolador**



Fuente: *Sistema de ordenador*. http://www.mikroe.com/img/publication/spa/picbooks/programming-in-c/chapter/01/fig0-1.gif. Consulta: agosto de 2015.

La situación es muy distinta para el mercado de los Microcontroladores, a pesar de que existen algunas compañías muy conocidas como Texas

Instruments o Microchip, no existe el mismo dominio del mercado. La tremenda cantidad de arquitecturas puede ser abrumadora para el diseñador de sistemas embebidos. Conocer la tecnología detrás de un microcontrolador es muy importante para escoger el adecuado para una determinada aplicación. Un microcontrolador se compone de forma de una unidad central de procesamiento (CPU), memorias y diversos periféricos que permiten una interacción final con el entorno, todos ellos encapsulados en un integrado. Esto hace posible construir sistemas digitales que se relacionen con el entorno de forma compacta, sin necesidad de construir una placa para conectar los componentes a la CPU.

Esto limita la flexibilidad de construcción para un sistema que permita agregar los periféricos necesarios, sin embargo resulta mucho más práctico utilizar un SOC en las aplicaciones que si bien no es posible colocar los periféricos con una comunicación directa con la CPU es posible encontrar en la vastedad del mercado de los microcontroladores alguno que se adapte a las necesidades de la aplicación y es por ello que se han popularizado en los últimos días.

3.5.2. Arquitecturas de microprocesadores

Al evaluar procesadores es importante entender la diferencia entre una ISA (por sus siglas de *instruction set architecture*), la implementación de un procesador o un chip. Una ISA es una definición para el conjunto de instrucciones que el procesador puede ejecutar y algunas restricciones estructurales como el tamaño de palabra. La definición de una palabra depende de la arquitectura del microprocesador, por ejemplo, una palabra para una arquitectura de 64 bits se definirá como el un conjunto de ocho bytes, sin embargo para una arquitectura de 8 bits se definirá como un byte. Existen

especialmente dos filosofías al hablar de ISA que gobiernan el mercado de microprocesadores.

La primera de ellas es llamada CISC por sus siglas en inglés de *complex instruction set computer* y su conjunto de instrucciones es extenso ya que el procesador se diseña para realizar complejas y específicas tareas usando menos instrucciones que su contraparte, aunque cada instrucción consuma más energía. La otra filosofía es RISC por sus siglas en inglés de *reduced instruction set computer* y su conjunto de instrucciones es pequeño (relativamente hablando), diseñado para realizar operaciones más básicas y simples, empleando más instrucciones para tareas complejas. Escoger un microcontrolador basado en RISC o CISC, depende de la aplicación, una arquitectura RISC es preferida sobre una CISC cuando se trata de sistemas embebidos donde los sistemas son alimentadosmuchas veces por baterías y algunos de ellos no los componen algoritmos complejos, por lo que el uso de la energía prevalece sobre el tiempo de computación.

3.5.3. Arquitecturas de memoria

Existen tres tipos de problemas relacionados con la memoria. El primero es que resulta frecuente la necesidad de mezclar diferentes tecnologías de memoria en el mismo sistema embebido. Muchas de ellas son volátiles, lo que significa que el contenido de la memoria se pierde al perder la energía que alimenta los circuitos. Muchos sistemas embebidos necesitan algo de memoria no-volátil y algo de memoria volátil, y adentro de estas categorías hay diversas opciones y cada opción trae diferentes consecuencias al comportamiento del sistema.

La segunda, es la jerarquía en la memoria, dado que las memorias con gran capacidad y bajo consumo son lentas, por lo que es necesario alcanzar un desempeño a un costo razonable. Memorias rápidas deben ser mezcladas con memorias lentas. El tercero, el espacio de direcciones de una arquitectura del procesador se divide para dar acceso a varios tipos de memoria, y así proveer soporte a modelos de programación comunes y diseñar direccionamientos para periféricos diferentes a las memorias, tales como módulos de GPIO, módulos de PWM, temporizadores entre muchos otros.

3.5.3.1. RAM

La memoria de acceso aleatorio (siglas en inglés de *random access memory*) se utiliza como memoria de trabajo de computadoras para el sistema operativo, los programas y la mayor parte del software. Se denominan de acceso aleatorio porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder (acceso secuencial) a la información de la manera más rápida posible. Existen dos tipos de memoria RAM: la SRAM (siglas en inglés de *static* RAM) y la DRAM (siglas en inglés de *dynamic* RAM).

Una memoria SRAM es más rápida que una memoria DRAM, además de ser más cara. Por lo general las SRAM son utilizadas para memoria caché y las memorias DRAM para la memoria principal del procesador. La memoria DRAM sostiene la información por un corto tiempo, por lo que cada localidad de memoria debe ser refrescada constantemente, el tiempo de refresco será especificado por el fabricante. El simple hecho de leer la memoria refrescará las posiciones que son leídas, sin embargo las aplicaciones pueden no acceder a todas las posiciones de memoria en el tiempo especificado, por lo que las DRAM deben ser acompañadas de un controlador que asegure que todas las

posiciones de memorias son refrescadas frecuentemente para sostener la información. El controlador de la memoria detendrá los accesos si la memoria se encuentra en uso con un refresco cuando el acceso es iniciado. Esto introduce variaciones en los tiempos del programa.

3.5.3.2. Non-Volatile Memory

Los sistemas embebidos necesitan almacenar información aún cuando no hay energía que alimente los circuitos. Para ello las memorias novolátiles ofrecen una solución a este problema. La primera aproximación de una memoria no volátil fue las de núcleo-magnético, donde un anillo ferromagnético era magnetizado para almacenar información.

Actualmente, la memoria básica no volátil es la ROM (siglas en inglés de *read only memory*), cuyo contenido se encuentra fijo desde su fabricación. Este tipo de memoria es muy útil para dispositivos producidos en masa que solamente necesitan un programa e información constante almacenada. Estos programas son conocidos como *firmware*, dando a entender que no es lo mismo que el software ni tan manipulable como él.

En el mercado actual hay muchas variantes de una memoria ROM, una de ellas es la EEPROM (siglas en inglés de *electrically-erasable programmable*) la cuál viene de formas diferentes. La escritura en estas memorias por lo general toma más tiempo que la lectura, además el número de escrituras se encuentra limitado en la vida útil del dispositivo. Una memoria *flash* es un tipo de memoria EEPROM muy útil, comúnmente usada para almacenar *firmware* o datos de los usuarios que deben perdurar a pesar que no exista energía.

La memoria *flash* fue inventada en la década de los 80 por el Dr. Fujio Masuoka, a pesar de ser una forma muy conveniente de almacenar datos aún presenta algunos desafíos para los diseñadores de sistemas embebidos. Usualmente las memorias *flash* poseen una velocidad de lectura aceptable, sin embargo no es tan rápida como una SRAM o DRAM, por lo que la información alojada en una memoria *flash* debe de moverse a una *RAM* para ser usada por el programa.

Vale la pena mencionar los discos duros que son memorias novolátiles. Son capaces de almacenar gran cantidad de datos pero el tiempo de acceso se vuelve muy largo. En particular, los sistemas mecánicos que componen el funcionamiento de estas memorias requieren que un lector se posicione en una posición deseada y esperan a que ello suceda para poder leer una dirección de memoria deseada. El tiempo en que lo hacen es variable y difícil de predecir. Además, los hacen vulnerables a vibraciones comparados con memorias de estado sólido (las mencionadas anteriormente) y por tanto no representan una buena opción a utilizarla en sistemas móviles o cambios bruscos, tal como muchos CPS.

3.5.4. Jerarquía de la memoria

Muchas aplicaciones requieren significativas cantidades de memoria, más de la disponible en el integrado que empaqueta el microprocesador. Muchos microprocesadores utilizan jerarquía en la memoria, la cual combina diferentes tecnologías de memoria para incrementar la capacidad en conjunto mientras se optimiza costo, latencia y consumo de energía. Es común encontrar una pequeña cantidad de memoria *on-chip* con tecnología SRAM que será utilizada en conjunto con una mayor cantidad de memoria *off-chip* con tecnología DRAM. Estas pueden ser mezcladas con otros tipos de memoria como discos duros,

que tienen una gran capacidad de almacenamiento, pero carecen de acceso aleatorio y por tanto pueden ser lentas en la lectura y escritura.

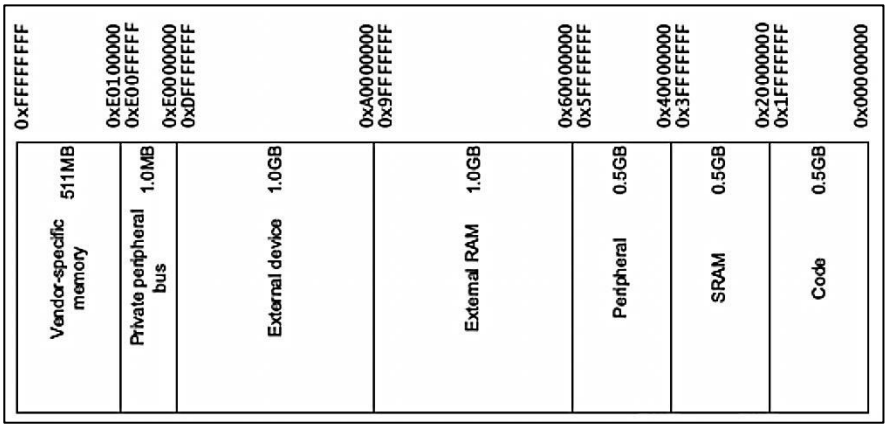
Para un programador de aplicaciones de propósito general, podría no ser relevante como se encuentra fragmentada la memoria a través de tecnologías. Y por lo general en aplicaciones de propósito general es utilizado el esquema llamado memoria virtual, que hace que diferentes tecnologías sean transparentes al programador, mostrándole un espacio de direcciones continuo. El sistema operativo o el hardware puede proveer un mecanismo de traducción de direcciones, que convierten direcciones lógicas en direcciones físicas apuntando a posiciones de las memorias con diferentes tecnologías. Esta traducción por lo general se realiza con ayuda de un TLB (siglas en inglés de *translation lookaside buffer*) que acelera la conversión de direcciones. Para diseñadores de sistemas embebidos que necesitan de un muy buen control del tiempo deben evitar esta técnica ya que puede introducir latencias imprevistas debido a que puede resultar muy difícil predecir cuando tardará la memoria en acceder, por lo que debe tener conocimiento de la forma en que se encuentra distribuida la memoria a través de las diferentes tecnologías.

3.5.4.1. Mapas de memoria

El mapa de memoria de un procesador define como las direcciones de memoria se encuentran distribuidas en el hardware. El espacio total de la memoria se encuentra restringido por el ancho de direcciones propio del procesador. Por ejemplo, un procesador 32 bits puede acceder a 2^{32} casillas de memoria, tal es el caso de alguno basado en Cortex-M4 con capacidad de almacenamiento hasta 4 GB, porque cada casilla en esta arquitectura es de 1 byte. Por lo general el ancho de direcciones depende del número de bits de la arquitectura a excepción de los microprocesadores basados en arquitecturas de

8 bits donde el ancho de direcciones es frecuentemente más alto para tener una aplicación práctica, por lo general 16 bits.

Figura 70. **Mapa de memoria Cortex-M4**



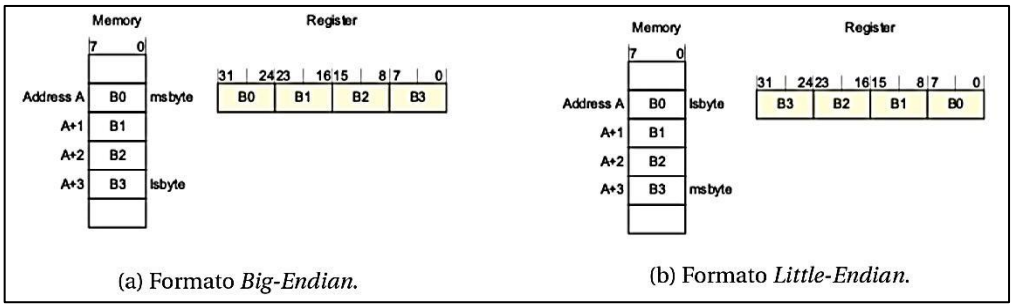
Fuente: *Cortex-M4 Devices. Generic User Guide*. http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf. Consulta: agosto de 2015.

En la figura 70, se muestra la distribución de la memoria en un microprocesador Cortex-M4 en la que el mapa se encuentra fijo. El ancho de las casillas es de 1 byte (8 bits) pudiendo así dar acceso a 4GB de memoria. Un procesador basado en una arquitectura Cortex-M4 ve la memoria como una colección de bytes enumerados en orden ascendente desde cero. Por ejemplo, los bytes de 0 a 3 almacenan una palabra, por el hecho de ser una arquitectura de 32 bits las palabras abarcan 4 bits, los bits del 4 al 7 almacenan la segunda y así sucesivamente.

El orden de almacenamiento de los bits de una palabra puede hacerse de dos formas: *big-endian* y *little- endian*. En la figura 71, se muestran los dos formatos. El formato *big-endian* almacena los bits más significativos en las

direcciones más bajas; el formato *little-endian* almacena los bits más significativos en las direcciones más altas. Procesadores como los basados en Intelx86 por defecto utilizan *Little-endian* y procesadores basados en PowerPC de IBM utilizan *Big-endian*. Algunos procesadores son capaces de soportar ambos formatos.

Figura 71. **Formatos de direccionamiento**



Fuente: *Cortex-M4 Devices. Generic User Guide*. http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf. Consulta: agosto de 2015.

3.5.4.2. Registros

Los registros de un procesador son casillas de almacenamiento de alta velocidad que se encuentran dentro del procesador y varían de arquitectura en arquitectura. Los registros pueden ser implementados directamente utilizando *flip-flops* en los circuitos internos del procesador, o pueden implementarse en un simple banco, usualmente utilizando una SRAM.

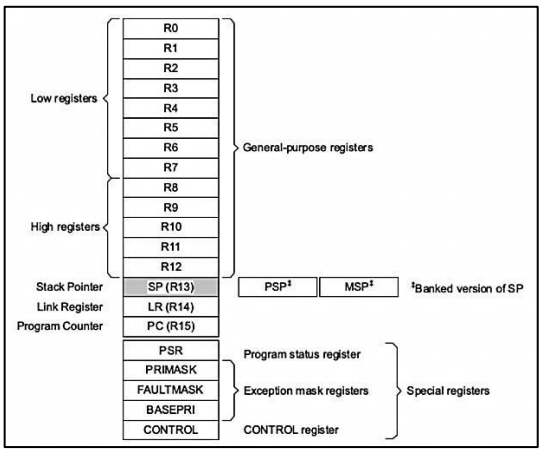
El número de registros que son parte de los circuitos internos del procesador por lo general es pequeño, por el costo de los bits que toma identificar un registro en una instrucción. Una ISA generalmente provee

instrucciones que pueden acceder o uno, dos o tres registros. Para manejar eficientemente el almacenamiento de los programas en la memoria, estas instrucciones no pueden requerir demasiados bits para codificarlos y por tanto no es posible dedicar muchos bits en la identificación de registros.

Por ejemplo, si un procesador posee 16 registros, debe dedicar 4 bits de una palabra para identificar un registro, aparte de asignar algunos para identificar la instrucción en sí, que se codifica todo como una instrucción. Una instrucción de adición, por ejemplo, realiza la suma de dos registros y la almacena en un tercer registro y todo debe ser codificado en una instrucción.

En la figura 72 se puede observar los registros internos de un procesador basado en Cortex-M4. Los registros desde R0 a R12 son registros de propósito general y contienen ya sea información o direcciones. El registro R13, es también conocido como *Stack Pointer* traducido al español como puntero de pila y siempre señala a la cima de la pila. La pila es una región de la memoria ocupada dinámicamente por el programa en un patrón LIFO (siglás en inglés *Last In First Out*). El registro PC contiene la dirección actual de la instrucción que se ejecuta en el programa. Existen registros con funciones especiales por lo que son llamados de esa manera; registros como IPSR, BASEPRI o PRIMASK tienen propiedades especiales en el manejo de interrupciones.

Figura 72. **Registros Core un Cortex-M4**



Fuente: *Cortex-M4 Devices. Generic User Guide*.
http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf.
Consulta: agosto de 2015.

3.5.5. Periféricos

- Buses: un bus en una computadora es un sistema digital que transfiere datos entre los componentes de una computadora o entre varias computadoras. Está formado por cables o pistas en un circuito impreso, dispositivos pasivos o de circuitos integrados. La función del bus es la de permitir la conexión lógica entre distintos subsistemas de un sistema digital, enviando datos entre diferentes dispositivos. Todos los buses en una computadora tienen funciones especiales, como ejemplo, las interrupciones y las DMA que permiten que un dispositivo periférico acceda a una CPU o a la memoria usando el mínimo de recursos. Es posible mencionar los siguientes tipos de bus:

- Bus en paralelo: es un bus en el cual todos los bits de una palabra (la cantidad dependerá de la arquitectura) son enviados y recibidos al mismo tiempo. Han sido usados frecuentemente en las computadoras, desde el bus principal del procesador hasta tarjetas de expansión de vídeo e impresoras. El ancho de datos enviado es grande y los datos son transmitidos a una frecuencia moderada y la tasa de datos enviada es igual al ancho de datos por la frecuencia de funcionamiento.

- Bus en serie: en estos buses los datos son enviados bit a bit y son reconstruidos por registros y rutinas.

Vale la pena mencionar los siguientes buses presentes en una computadora.

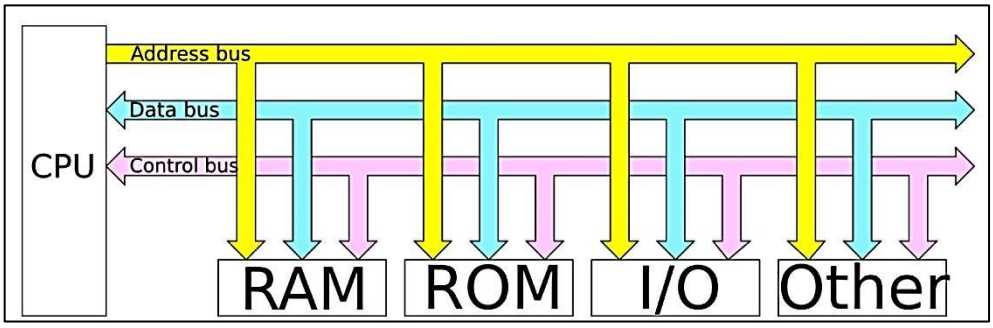
- Bus de control: permite enviar señales de control de la unidad central de procesamiento a periféricos, memorias y otros dispositivos. Controla acceso y flujo de direcciones en la transmisión de datos.

- Bus de direcciones: es un canal independiente del bus de datos y en él se establece la dirección de memoria del dato a utilizar por el procesador. Este bus es un conjunto de líneas eléctricas necesarias para establecer una dirección de memoria. La capacidad máxima de la memoria que puede utilizar un procesador depende de este bus.

- Bus de datos: su función es la de transportar los datos entre la CPU y los periféricos.

En la figura 73 se muestran los buses de dirección, datos y control de una computadora y su conexión con la CPU así como los distintos periféricos.

Figura 73. **Buses de dirección, datos y control en una computadora**



Fuente: *Buses: de control, de direcciones y de datos.* https://es.wikipedia.org/wiki/Bus_(informática)#/media/File:Computer_buses.svg. Consulta: agosto de 2015.

Algunos diseños permiten multiplexar el bus de datos con el de memorias, utilizando como árbitro un bus de control para discernir el rol que esta jugando el bus para determinado momento.

3.5.5.1. Salidas y entradas digitales

Los Microcontroladores se encuentran diseñados para operar en un entorno físico. Comunicarse con el exterior podría atribuirse como la razón de la existencia de los Microcontroladores. Una parte importante en un microcontrolador son las entradas y salidas de propósito general. Una entrada/salida de propósito general es un pin del microcontrolador que puede ser utilizada por algún otro sistema discreto o físico para obtener o brindar información representada como señales eléctricas. Los pines de propósito

general, en varios Microcontroladores pueden ser utilizados de distintas formas. Representan un camino para la información hacia otros periféricos que realizan funciones específicas.

Las entradas/salidas digitales son un periférico que permite a los pines relacionados con él, ser configurados para tener una baja impedancia y generar señales eléctricas, específicamente dos señales eléctricas relacionadas con valores lógicos 1 o 0, en otras palabras representar una salida digital; o bien para tener una alta impedancia y recibir datos, pudiendo interpretar solamente dos niveles de voltaje (o corriente en algunos casos) y asociando la lectura a un valor lógico 1 o 0, en otras palabras representar una entrada.

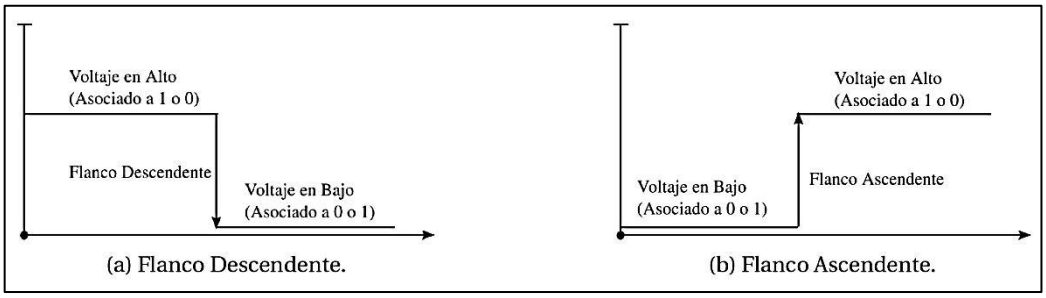
3.5.5.2. Reloj del sistema, temporizadores y contadores

Los contadores y temporizadores son útiles en la medición de eventos y tiempo. No existe ninguna diferencia entre un contador y un temporizador más que la aplicación que se le da al circuito de electrónica digital encargado de contar eventos relacionados con alguna señal eléctrica, de forma independiente a la CPU. En otras palabras, mientras la CPU se encuentra realizando alguna operación, al mismo tiempo estos circuitos pueden encontrarse contando los eventos de una señal sin hacer uso de ella. Resultan prácticos ya que la CPU puede leer el valor del conteo cuando sea necesario brindando cierto grado de paralelismo.

El evento más común para generar el conteo en estos periféricos son los flancos, los cuáles no son más que un cambio en el voltaje reflejado en un cambio lógico de la señal. Por lo general, las entradas digitales son capaces de interpretar solamente dos señales, el cambio de un estado de voltaje (de los

dos que es capaz interpretar) a otro es llamado flanco. En la figura 74 se pueden apreciar las variaciones de un voltaje a otro, por lo general en los diagramas dicho cambio es representado por una flecha indicando la dirección del cambio.

Figura 74. **Flancos descendente y ascendente**



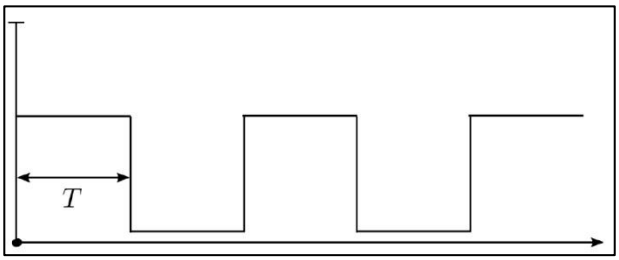
Fuente: elaboración propia, empleando Inkscape.

Cuando el cambio del voltaje es de uno más alto a uno más bajo, el cambio es llamado flanco negativo o descendente, al contrario, si el cambio de voltaje se produce de uno más bajo a uno más alto el cambio es llamado flanco positivo. Los flancos son la forma más común de representar eventos ya que un evento es útil considerarlo si representa un cambio de algún fenómeno, ya que puede generar una transición en una dinámica discreta (modelada por una máquina de estados), de lo contrario la dinámica permanecerá en el estado que se encuentra. La diferencia de un contador con un temporizador, es la fuente que utilizan los dispositivos para el conteo de flancos.

Un contador es capaz de realizar el conteo de alguna fuente de flancos cualquiera que fuere, por ejemplo, la señal recibida por una entrada digital o bien por otro periférico, tal como un comparador o un sensor. Los contadores se usan para realizar el conteo de eventos aleatorios, eventos cuyo tiempo de

ocurrencia es imposible determinarlo sin embargo es necesario registrar cuando suceden; este tipo de conteos son llamados asíncronos. Un temporizador utiliza como fuente de flancos una señal cuadrada periódica, lo que asegura que los flancos suceden de forma equidistante en el tiempo, separados entre sí por un período *T*. Los eventos son generados periódicamente por lo que permite discretizar el tiempo, generando unidades básicas de tiempo marcadas por el período *T* de la señal cuadrada.

Figura 75. **Señal de reloj**



Fuente: elaboración propia, empleando Inkscape.

Por tanto, si un temporizador ha contado *n* flancos desde que se ha iniciado el conteo (positivos o negativos, en algunos Microcontroladores es posible configurar la dirección que el temporizador toma como evento) el tiempo que ha transcurrido es *nT*. Dando así una forma aproximada de medir el tiempo transcurrido desde iniciado el conteo de flancos. Este tipo de conteo es llamado síncrono, y la señal cuadrada es llamada señal de reloj. Los contadores y temporizadores pueden verse como un almacenamiento de *k* bits donde se lleva control de la cantidad de flancos ocurridos por una fuente de flancos. El máximo de flancos capaz de contar un temporizador o contador, *N*max esta dado, ya que comienza en cero, por:

$$Nmax = 2^{K} - 1 \qquad \text{[Ec. 3.21]}$$

Cabe resaltar que las computadoras trabajan con electrónica digital secuencial, la cual permite ejecutar en un orden específico distintas operaciones, reflejadas como instrucciones, ejecutando una tras otra cada vez que sucede un flanco. Para que sea posible ejecutar de forma secuencial las instrucciones, es necesaria una señal de reloj que marque el ritmo de ejecución de las instrucciones, y esta la obtiene de una fuente externa. La señal que marca el ritmo para casi todos los casos (si no es que en todos) es periódica por lo que es llamada reloj del sistema.

3.6. Programas y lenguajes de programación

Un programa es un conjunto de valores que se almacenan en algún lugar de la memoria de una computadora y que esta última interpreta. Un programa en su más pura expresión se ve como una muy larga sucesión de bits, ya que es lo único que puede entender una computadora. Escribir programas de esa manera es una tarea demasiado complicada, extenuante y propensa a ser creada con errores por esa razón los primeros operadores de computadoras decidieron reemplazar las cadenas de bits por palabras y letras provenientes del inglés dando así origen al lenguaje ensamblador o *assembly*. El lenguaje sigue la misma estructura del lenguaje de máquina, pero las palabras y letras son más fáciles de recordar que los números. Más tarde aparecieron diferentes lenguajes denominados lenguajes de alto nivel debido a que tienen una estructura sintáctica similar al lenguaje utilizado por los seres humanos.

3.6.1. Proceso de compilación

La herramienta para convertir un lenguaje de alto nivel a un lenguaje de máquina se llama compilador. Un compilador representa el puente entre el lenguaje de alto nivel y el hardware. Verifica la sintaxis del lenguaje de alto

nivel, genera de forma eficiente código objeto, realiza la organización en tiempo de ejecución y da formato a la salida de acuerdo a las convenciones del enlazador y el ensamblador. Un compilador consiste en las siguientes fases:

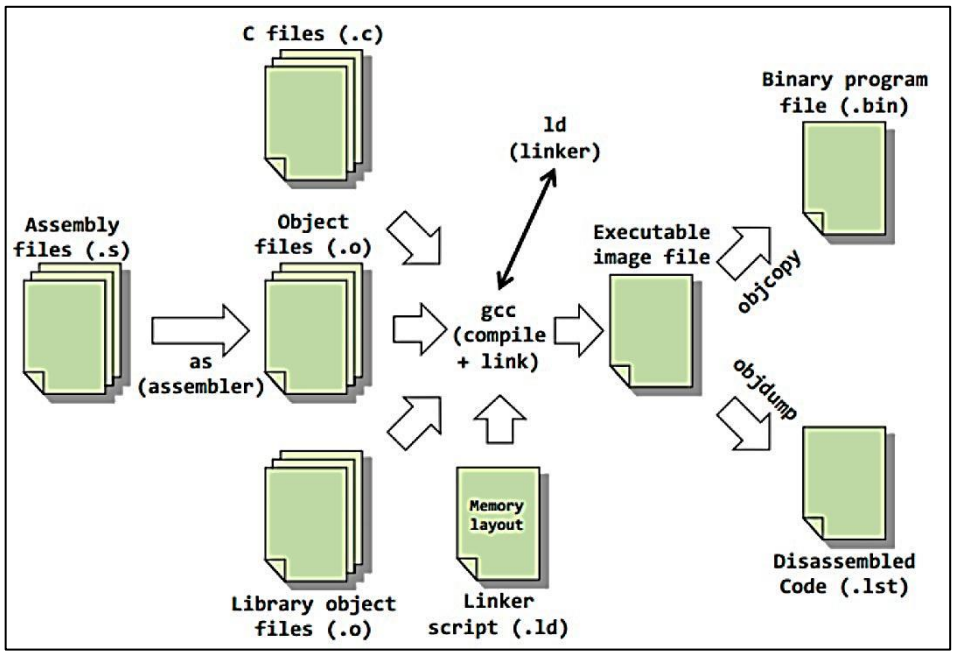
- *The-front-end:* revisa sintaxis y semántica, genera una representación intermedia del código para ser procesado por la fase *middle-end*. Realiza revisiones de tipos colectando información, genera errores y advertencias, si hay alguna que sea relevante.

- *The-middle-end*: realiza optimización, incluyendo remociones de código inservible o inalcanzable, descubrimiento y propagación de valores constantes, reubicación de procesos a lugares con menor ejecución o especialización de la computación basada en el contexto. Genera otra representación intermedia para ser procesada por la fase *back-end*.

- *The-back-end:* genera el código en *assembly*, realiza la ubicación de los procesos en los registros. Optimiza el código para el uso del *hardware*.

Varios códigos escritos en lenguajes de alto nivel, como C, hacen uso de secciones que se encuentran en *assembly* o bloques de código en lenguaje de máquina ya listos para ejecutarse, pero que deben ser ubicados en alguna parte de la memoria por lo que el código pasa por etapas de ensamblaje y de enlazamiento conocidas como *assembler* y *linker*.

Figura 76. **Compilación por GCC Toolchain**



Fuente: *Toolchain*. http://www.bogotobogo.com/cplusplus/images/embedded_toolchain/CompilerAssemblerToolchain_gcc.png. Consulta: diciembre de 2014.

En la figura 76, se muestra el trabajo que hace un compilador (GCC, GNU Cross Compiler), un ensamblador y un enlazador, en este caso GNU Toolchain, como puede verse en la imagen un ensamblador genera código de máquina con base en un código escrito en *assembly*. El compilador traduce el lenguaje C a código de máquina y luego con la ayuda de un enlazador realiza la ubicación en memoria y otros procesos necesarios para tener un programa funcionando en el procesador.

3.6.2. Lenguaje C

El lenguaje C proporciona una gran flexibilidad de programación y una muy baja corrección de inconsistencias, de forma que el lenguaje deja bajo la responsabilidad del programador acciones que otros lenguajes realizan por si mismos. Todo programa de C consta básicamente de un conjunto de funciones, y una función llamada main, que es la primera en ejecutarse al comenzar el programa, llama al resto de funciones que compongan el programa.

Se diseñó para ser compilado por algún compilador directo y así dar acceso de la memoria a bajo nivel y proveer construcciones léxicas que representen el código de máquina de la forma más cercana posible. Este lenguaje se encuentra disponible una gran cantidad de computadoras desde Microcontroladores hasta supercomputadoras.

3.6.2.1. Identificadores

En el lenguaje C, un identificador es cualquier palabra no reservada que obedece tres reglas:

- Todos los identificadores de variables comienzan con una letra o un guion bajo (_).
- Los identificadores de variables pueden contener letras, guiones bajos y otros dígitos.
- Los identificadores de variables no pueden coincidir con las palabras reservadas del lenguaje.

En particular, C es sensible al contexto haciendo los identificadores Variable, VARIABLE y VaRiAbLe todos diferentes entre ellos. La longitud

máxima de un identificador depende del compilador que se esté usando, pero generalmente, suelen ser de 32 caracteres, ignorándose todos aquellos que compongan el identificador y sobrepasen la longitud máxima.

3.6.2.2. Tipos de datos y variables

Todos los programas trabajan manipulando algún tipo de información. Una variable en C se define como un identificador que será tratado como un tipo predefinido de dato. El compilador reservará determinado espacio en memoria para poder almacenar información en el, y dependiendo del tipo con que ha sido declarada la variable manipulará esa información de una u otra manera. El identificador le servirá al compilador y principalmente a los humanos para evitar tratar directamente con las direcciones en la memoria, trabajo que puede volverse sumamente tedioso.

En C, toda variable, antes de poder ser usada, debe ser declarada, especificando con ello el tipo de dato que almacenará. Toda variable en C se declara de la forma:

<tipo de dato > <nombre de la variable >;
En caso de tener quererse declarar varias variables se hace de la forma:
<tipo de dato > <nombre de la variable 1>, ... , <nombre de la variable n >;
Teniendo así algunos ejemplos de declaración de variables en C:

Teniendo así algunos ejemplos de declaración de variables en C:

```
char a;
int b, c;
double d;
float e;
```

Un tipo de dato de tipo char siempre tendrá ocho bits, sin embargo un tipo de dato int, el número de bits que tendrá depende de la arquitectura del procesador y el compilador. En C un tipo de dato int tiene un tamaño mínimo de 16 bits pudiendo almacenar enteros en el rango comprendido entre -32 767 y 32 767. Los tipos de dato *float* y *double* son conocidos como flotantes ya que representan números con punto decimal, pero su precisión es variable. La diferencia entre ambos tipos radica en el número de bits que se reserva en memoria, siendo para double el doble que para float; lo que implica que se tiene más precisión al trabajar con un tipo de dato double que float. El tipo de dato float y double interpretan los bits que representan los números de la siguiente forma:

$$(-1)s \times c \times bq \qquad \text{[Ec. 3.22]}$$

Donde s es un bit asignado al signo, c es un número de determinada cantidad de bits, q es un número de determinada cantidad de bits y *b* puede ser dos o diez, dependiendo de la interpretación, muchos compiladores los interpretan de acuerdo al estándar IEEE 754. Un tipo de dato void indica sin tipo y tiene un uso especial.

3.6.2.3. Modificadores de tipo

Los modificadores se aplican sobre los tipos de datos citados con anterioridad permitiendo cambiar el tamaño o la forma de manipular la información de la variable que se ha declarado haciendo uso de los modificadores.

Tabla LXXII. **Modificadores de tipo**

Modificador	Tipo	
signed	***char***	***char***
unsigned	***char***	***char***
long	***int***	***double***
short	***int***	

Fuente: elaboración propia.

El modificador *signed* toma uno de los bits asignados en memoria para el signo del número. El modificador *unsigned* trata a los números como enteros positivos por lo que el bit de signo no se toma en cuenta y es posible almacenar valores cuyo valor es el doble de lo que es posible si se usa *signed*. Además es posible aplicar dos modificadores seguidos a un mismo tipo de dato. Por ejemplo, la declaración:

- *unsigned long int a*

Permite asignar un espacio en memoria de 32 bits sin contar el signo para la variable a. Si se hubiera utilizado el modificador *signed*, un bit se toma para definir el signo, siendo posible almacenar en la variable a enteros $desde\ -2^{31}\ hasta\ 2^{31}\ -1$.

Además de los modificadores aplicados al tipo, existen los modificadores de acceso, que limitan el uso que puede darse a las variables declaradas. Los modificadores de acceso anteceden a la declaración del tipo de dato de una variable. Es posible mencionar dos modificadores de acceso:

- *const*: la declaración de una variable como *const* permite asegurar que el valor de la variable no será modificado durante el flujo del programa, el

cuál conservará el mismo valor que se le asignó en el momento de su declaración. Por ejemplo, si se hace la siguiente declaración de variable:

const char x = 5

Cualquier intento posterior de modificar el valor de x en el flujo del programa el compilador producirá un error en la compilación.

- Volatile: esta declaración indica al compilador que dicha variable puede modificarse por un proceso externo al programa, y por ello no debe optimizarla. Forza cada vez que se usa la variable se realice comprobación de su valor.

Los modificadores *const* y *volatile* pueden usarse de forma conjunta, ya que no son mutuamente excluyentes. Por ejemplo, si se declara una variable que actualizará el reloj del sistema, (proceso externo al programa), y no se desea modificarla en el interior del programa. En ese caso se declarará: volatile const unsigned long int hora.

3.6.2.4. Declaración de variables

En C, las variables pueden ser declaradas en cuatro lugares del programa conocidos como alcances o ámbitos:

- Fuera de todas las funciones del programa, son llamadas variables globales, accesibles desde cualquier parte del programa.
- Dentro de una función, son llamadas variables locales, accesibles tan solo por la función en las que se declaran.

- Como parámetros a la función, accesibles de la misma forma que si se declararan dentro de la función.
- Dentro de un bloque de código del programa, accesible tan solo dentro del bloque donde se declara. Esta forma de declaración puede interpretarse como una variable local del bloque.

Es posible modificar el alcance de los datos, y esto se hace con los especificadores de almacenamiento. Estos especificadores de almacenamiento cuando se usan deben preceder a la declaración del tipo de la variable. Se pueden mencionar los siguientes especificadores de almacenamiento:

- Auto: se usa para declarar que una variable local existe solamente mientras se esté en una subrutina o bloque de programa donde se declara, pero, dado por defecto a toda variable declarada y no suele usarse.

- Extern: se usa en el desarrollo de programas compuestos por varios módulos. El modificador extern se usa sobre las variables globales del módulo, de forma que si una variable global con este especificador, el compilador no aparta espacio en memoria para ella, sino que, tan solo tiene en cuenta que dicha variable ya ha sido declarada en otro módulo del programa y es del tipo de dato que se indica.

- Static: este especificador actúa según el ámbito donde se declaró la variable:

    - Para variables locales, este especificador indica que la variable debe almacenarse de forma permanente en memoria, como si fuera una variable global, pero su alcance será el que

correspondería a una variable local declarada en la subrutina o bloque. El principal efecto que provoca la declaración como static de una variable local es el hecho de que la variable conserva su valor entre llamadas a la función.

- Para variables globales, el especificador *static* indica que dicha variable global es local al módulo del programa donde se declara, por tanto, no será conocida por ningún otro módulo del programa.

- Register: se aplica solo a variables locales de tipo *char* e *int*. Dicho especificador indica al compilador que mantenga esa variable en un registro de la CPU y no cree por ello una variable en la memoria, en caso de ser posible. El compilador ignorará dicha declaración caso de no poder hacerlo. El uso de variables con especificador de almacenamiento register permite colocar en algún registro de la CPU variables muy frecuentemente usadas, tales como contadores de bucles.

3.6.2.5. Operadores aritméticos

Es posible modificar el valor numérico de una variable aplicando a ella operadores aritméticos. A continuación, se describen los operadores aritméticos en orden de prioridad:

- Incremento y decremento. El operador de incremento es representado en C por el símbolo ++ y el operador de decremento por el símbolo --. Se pueden utilizar solo en variables de tipo *int* y *char*. Estos operadores incrementan o disminuyen en una unidad el valor de la variable a la que se aplican, respectivamente. Estos operadores pueden suceder o preceder a la variable. Cuando alguno de los dos operadores precede a

la variable operando, C primero realiza el incremento o disminución, dependiendo del que se esté utilizando y después usa el valor de la variable operando, realizando la operación en orden contrario en caso de suceder a la variable.

- Más y menos unario**.** Son símbolos utilizados para definir el signo del valor de la variable a la que se aplica. Son representados por los símbolos + y -; siempre preceden a la variable a la que se aplican. El más unario solo existe para tener simetría con el menos unario, sin embargo, no altera el valor de la variable. El menos unario cambia el signo del valor de la variable a la que se aplica.

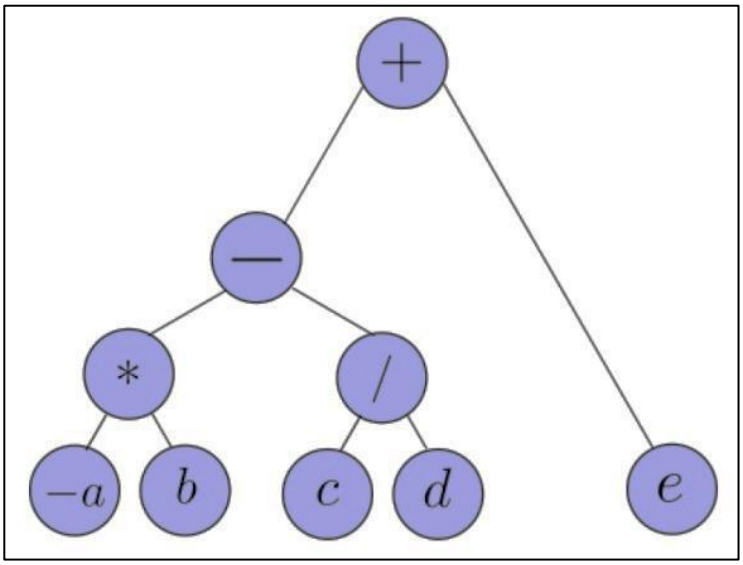
- Multiplicación, división y módulo. Estos operadores realizan las operaciones que sus nombres indican. Siendo el símbolo * usado para representar la multiplicación de dos números, el símbolo/para la división entre dos números y el símbolo % para el obtener el residuo de la división de dos números. Se debe aclarar que al trabajar con distintos tipos de variables, en los operadores multiplicación y división ya que módulo sólo es útil en enteros, se realizan conversiones implícitas de los operandos, cambiado los tipos de los operandos de acuerdo a reglas que se enuncian más adelante.

- Suma y resta. Estos operadores realizan las operaciones que sus nombres indica. Siendo el símbolo + utilizado para representar la suma de dos números y el símbolo - para la resta entre dos números. Al igual que la multiplicación y división, la suma y la resta al aplicarla a operandos de distintos tipos realizaran cambios implícitos en el tipo de los operandos.

Se han enunciado a los operadores en orden descendente de prioridad, o ascendente en jerarquía, esto es si se encuentra una serie de operaciones sin algún orden de ejecución establecido por los símbolos (), se ejecutarán las operaciones con menor jerarquía, mayor prioridad, antes que las de menor prioridad, mayor jerarquía. Por ejemplo, en la operación: -a*b-c/d+e, se ejecutará primero la operación -a ya que es un operador menos unitario y tiene una mayor prioridad que el resto de operaciones. Luego se ejecutarán las multiplicaciones y divisiones, dejando por último las sumas y las restas siendo equivalente la siguiente expresión:

(((-a)*b)-(c/d))+e, dado que tanto la resta y la suma tienen la misma prioridad pero siendo la resta la que aparece primero. En la figura 77 se muestra el árbol de jerarquías para la operación mostrada como ejemplo, donde se puede ver la jerarquía de las operaciones.

Figura 77. **Árbol de jerarquías**



Fuente: elaboración propia, empleando Inkscape.

3.6.2.6. Operadores relacionales y lógicos

El lenguaje C interpreta de forma muy particular los estados de verdad, cualquier valor diferente de cero puede ser interpretado como una expresión lógica binaria cuyo estado de verdad es Verdadero. De igual forma, cero puede ser interpretado como una expresión lógica binaria cuyo estado de verdad es Falso. Los operadores lógicos y relacionales (comparaciones y conectivos lógicos) en C devuelven como resultado un valor distinto de cero si la expresión escrita con los operadores es verdadera y cero si es falsa. A continuación, se presentan los operadores lógicos y relacionales en orden ascendente en jerarquía (descendente en prioridad).

- Negación. Esta es una operación lógica cambia el estado de verdad de una sentencia. Es decir, si se aplica a un valor distinto de cero devolverá cero y viceversa. El símbolo en C para esta sentencia es !. Siendo de los operadores relacionales y lógicos el que posee mayor prioridad.

- Comparación de orden. Estos operadores comparan dos números por su posición en la recta numérica siendo en C el símbolo > para la operación mayor que, el símbolo >= para la operación mayor o igual que, el símbolo < para la operación menor que y el símbolo <= para la operación menor o igual que.

- Igualdad y no igualdad. Comparan dos cantidades y verifica igualdad entre los operandos. Para la igualdad en C se tiene el símbolo ==, para verificar no igualdad en C se tiene el símbolo !=.

- Operación AND. Evalúa dos expresiones. Si alguna es cero, el resultado será cero. El símbolo para esta operación es &&.

- Operación OR. Evalúa dos expresiones. El resultado será cero en caso que ambas expresiones sean cero. El símbolo para esta operación es ||.

Los operadores lógicos y relacionales poseen una jerarquía mayor que los operadores aritméticos, dicho de otra forma, son menos importantes que los operadores aritméticos.

3.6.2.7. Operación asignación

Este operador en C tiene como símbolo =, y tiene una marcada diferencia con el símbolo ==; ya que C no diferencia la asignación de la comparación. El programador debe remarcar la operación que desea realizar, utilizando correctamente estos símbolos.

El operador asignación almacena en el resultado de la expresión del lado izquierdo del operador, el resultado de la expresión del lado derecho del operador. Este operador es el de menor prioridad de todos, por tanto, el último en ejecutarse.

3.6.2.8. Operadores sobre BITS

En C es posible manipular los bits que componen el valor de una determinada variable, ya que cualquier número es posible como una combinación de unos y ceros, con los operadores sobre bits los cuales se mencionan a continuación.

- Complemento a dos. Cualquier uno de la representación binaria del valor de una variable se cambia por cero. Cualquier cero de la representación binaria del valor de una variable se cambia por uno. Recibe su nombre

debido a que cambiar el valor de todos los bits resulta en el complemento a dos de un número. Tiene la misma prioridad que los operadores ++ y --.

- Desplazamiento a la izquierda y derecha. En el lenguaje C se para el desplazamiento a la derecha se utiliza el símbolo >>, y para el desplazamiento a la izquierda el símbolo <<. Por ejemplo, la expresión x<<3 corre todos los bits tres posiciones hacia la izquierda rellenando con ceros los 3 bits menos significativos. Por otro lado, la expresión x>>3 corre todos los bits tres posiciones hacia la derecha rellenando con ceros los 3 bits más significativos. La prioridad de estos operadores se encuentra entre la prioridad de los operadores aritméticos y los operadores relacionales y lógicos.

- Multiplicación, suma y suma exclusiva lógicas binarias. Para la multiplicación binaria se utiliza el símbolo &, el resultado de operar dos variables con este símbolo es que efectúa una multiplicación lógica binaria bit a bit, en los dos operandos por ejemplo la expresión 12&6 tendrá como resultado 4 ya que la representación binaria del 12 está dada por 11002 y la representación del seis está dada por 01102.

  Si se multiplica lógicamente cada bit se puede ver que solamente en el tercer bit de derecha a izquierda, se tienen como resultado uno y en el resto se tiene cero por tanto el resultado será 01002 el cuál es un 4 en representación decimal. Para la suma se utiliza el símbolo | y para la suma exclusiva el símbolo ^, realizando las respectivas operaciones bit *a*bit. Estos operadores tienen una mayor prioridad que los operadores lógicos && y ||, pero menor que el resto de los operadores lógicos y relacionales.

3.6.2.9. Sentencia de control *if*

Esta sentencia permite la ejecución de un bloque de código en caso de ser cierta la condición que se indica en su sintaxis. La sintaxis de la sentencia if se muestra a continuación:

```
if( condici ón)
{
sentencias ;
...
}
else
{
sentencias ;
...
}
```

Donde else es una sentencia opcional que permite ejecutar un bloque de sentencias en caso de resultar falsa la condición. Es posible anidar sentencias, colocar dentro del bloque a ejecutar más sentencias if, construyendo así programas con flujos de ejecución más complejos.

3.6.2.10. Sentencia de control *switch*

La sentencia switch permite que ciertas condicionales tengan una forma más práctica de escribir que las que se escribirían con sentencias if y else. La sentencia switch tiene la siguiente sintaxis:

```
switch ( variable )
{
case constante_1 :
```

```
sentencia ;
break ;
case constante_2 :
sentencia ;
break ;
...
default :
sentencia ;
}
```

La variable que recibe como parámetro la sentencia switch, debe ser de tipo *char* o *int*, la sentencia comparará el valor de la variable en cada constante y en caso de coincidir con alguna, se ejecutarán las sentencias que se encuentran entre los dos puntos y la sentencia *break*. La sentencia default se usa para indicarle al compilador las sentencias que debe ejecutar en caso de no coincidir con ninguna de las constantes declaradas.

3.6.2.11. Ciclo *for*

Los ciclos permiten repetir un bloque de instrucciones muchas veces, ahorrando al programador el trabajo de hacerlo y haciendo un uso efectivo de la memoria. El ciclo for permite ejecutar un bloque muchas veces y se compone de tres expresiones principales que definen el comportamiento del ciclo. La sintaxis del ciclo se muestra a continuación:

```
for ( Inicializaci ón; Condici ón; Incremento o Decremento )
{
sentencias ;
}
```

La sentencia de inicialización, se ejecuta antes de interactivar las sentencias enmarcadas por las llaves. El ciclo continúa repitiéndose siempre y cuando la condición dada entre los dos puntos sea cierta. La sentencia Incremento o Decremento se ejecuta al final de cada iteración y antes de la comprobación de la condición. La estructura más común de este ciclo es que las tres sentencias juntas tengan como fin llevar el conteo de las veces que se ejecuta el ciclo; la primera sentencia dando el valor inicial de una variable que sirve como conteo, la segunda sentencia controla cuantas veces se repite el ciclo y la última ejecuta la acción del incremento o decremento de la variable.

3.6.2.12. Ciclos *while y do-while*

El ciclo while ejecuta las sentencias enmarcadas por las llaves siempre y cuando la condición cierta. La sintaxis del ciclo es la siguiente:

```
while ( condici ón)
{
sentencias ;
}
```

El ciclo *while* siempre evalúa primero la condición y de ser cierta ejecuta las sentencias, esto implica que el ciclo puede nunca ejecutar la sentencias, en caso de ser falsa la condición desde un inicio. Existen casos en los que es necesario primero ejecutar las sentencias y luego evaluar la condición. Para estos casos es posible utilizar el ciclo *do-while*, que primero ejecuta las sentencias y después de realizar una iteración realiza la comprobación. La sintaxis del ciclo es la siguiente:

```
do
{
```

```
sentencias ;
}
while ( condici ón);
```

De esta forma se puede asegurar que el ciclo siempre se ejecutará al menos una vez.

3.6.2.13. Sentencias de control *break* y *continue*

Las sentencias de control *break* y *continue* permiten controlar la ejecución de los bucles. La sentencia *break* causa la salida del bucle en el cual se encuentra y ejecuta la sentencia que esté inmediatamente después del bucle.

La sentencia *continue* causa que el programa vaya directamente a comprobar la condición del bucle en los *bucles while y do-while*, o bien, que ejecute el incremento y después compruebe la condición en el caso del bucle *for*.

3.6.2.14. Arreglos

Los arreglos en C permiten almacenar información relacionada, utilizando un solo identificador usando un índice para diferenciar las variables almacenadas bajo el mismo nombre. Un arreglo puede ser visto como un conjunto ordenado de datos o una lista ordenada agrupando variables del mismo tipo. Un arreglo permite al programador organizar conjuntos de información eficientemente y de forma intuitiva. Los arreglos en C son declarados de la siguiente forma:

```
<tipo > <nombre >[<nú mero de elementos >];
```

En caso que se desee dar un valor inicial al arreglo se tiene la siguiente sintaxis:

<tipo > <nombre >[<nú mero de elementos >]={ elemento_1 , ... , elemento_n };

Por ejemplo si se desea declarar un conjunto de datos de 5 elementos de tipo entero se haría de la siguiente forma:

char variable [5]={ ’a’,’b’,’c’,’d’,’e’};

Para obtener los valores del arreglo basta con llamar al arreglo por su nombre con la posición del valor que se desea obtener. Cabe resaltar que la enumeración de las posiciones comienza en cero, por tanto en un arreglo de 5 posiciones la primera posición corresponde al índice 0 y la última posición corresponde al índice 4. Por ejemplo, si se escribe la sentencia variable[0] se tiene como resultado ’a’ y si se escribe variable[4] se obtiene ’e’. El lenguaje C no comprueba el tamaño de los arreglos. Por ejemplo escribir una rutina

```
int a [10];
int i;
for (i =0; i <100; i ++) {
a[i]=i;
}
```

Esta no generará errores en la etapa de compilación, sin embargo, el programa tendrá un funcionamiento incorrecto. Es posible declarar arreglos de varias dimensiones los cuales pueden verse arreglos de arreglos. Los que a su vez pueden ser arreglos de otros arreglos y así sucesivamente. La declaración de arreglos de más de una dimensión se realiza de forma parecida a la de una dimensión, la sintaxis de la declaración de un arreglo de varias dimensiones es:

<tipo > <nombre >[< tam_1 >][< tam_2 >]...[tam_N];

3.6.2.15. Punteros

Los punteros son herramientass poderosas que ofrece el lenguaje C a los programadores. Permiten una gran flexibilidad del uso de la memoria en una computadora, sin embargo, suelen ser fuentes frecuentes de errores en los programas, produciendo fallos muy difíciles de localizar y depurar.

Un puntero es una variable que contiene una dirección de memoria. Regularmente esa dirección es una posición de memoria de otra variable, por lo cual se dice que apunta a otra variable.

La sintaxis de declaración de una variable como puntero es:

<tipo > *< nombre >;

El tipo base de la declaración sirve para conocer el tipo de datos al que pertenece la variable a la cual apunta la variable de tipo puntero, esto es fundamental para poder leer el valor que almacena la sección de la memoria apuntada por la variable de tipo puntero y poder realizar operaciones aritméticas sobre ellos.

A continuación, se presentan algunos ejemplos de las variables puntero:

```
int *i;
char *c;
float *f;
```

Existen dos operadores útiles. El operador *Dirección*, representado en C por el símbolo & y el operador *Apuntador* representado por el símbolo *. Cada uno es el inverso del otro. El operador *Dirección* obtiene la dirección de memoria donde se aloja una variable. El operador *Apuntador* obtiene el valor que se aloja en una dirección de memoria. Por ejemplo

```
int *a, b, c;
b =15;
a=&b;
c=*a;
```

La variable b almacena el valor 15 en algún sector de la memoria, con la operación &b se obtiene la dirección de la memoria donde aloja su valor, y dicha dirección se almacena en la variable a. Luego la operación *a va a la dirección de memoria indicada por el valor de a y obtiene el valor que almacena esa dirección. Por tanto, el valor almacenado en la variable c será 15, ya que el valor de a es la dirección de b y se esta obteniendo el valor que almacena la dirección de b. Por tanto, la operación &(*b) es equivalente a la operación *(&b) y ambas devolviendo el valor de b. Es posible definir punteros a punteros. Donde la variable puntero a puntero tendrá la dirección de memoria de un puntero, el cual a su vez apunta a la dirección de memoria de otra variable.

3.6.2.16. Funciones

Una función es un conjunto de sentencias que en conjunto realizan una acción. Cada programa en C tiene al menos una función, la cual es *main*, pudiendo así todos los programas escritos en C declaras funciones adicionales. Las funciones declaradas por los programadores generalmente requieren de un tipo de dato, y será el tipo del valor devuelto por la función *return*. La sentencia *return* permite, primero que nada, salir de la función desde cualquier punto de

ella, y segundo devolver un valor del tipo de la función, en caso de ser necesario. La declaración de una función tiene la siguiente sintaxis:

```
<tipo > <nombre de la funci ón >(< lista de par á metros >){
sentencias ;
return <resultado >; // En caso de ser necesario .
}
```

El tipo de cada parámetro debe indicarse en la lista de parámetros que recibe la función.

Como ejemplo considérese que se quiere declarar una función que devuelva como resultado la suma de dos números enteros, la declaración de la función sería:

```
int suma ( int a, int b){
int resultado ;
resultado =a+b;
return resultado ;
}
```

Otro ejemplo puede ser la función signo, que devuelve 1 en caso que el valor a evaluar sea mayor que cero, -1 en caso de ser menor que cero y 0 en caso de ser cero. Dicha función es posible declararla en C de la siguiente manera, utilizando condicionales if anidadas.

```
int signo ( int a){
if(a >0) {
return 1;
}
else
{
```

```
if(a <0) {
return -1;
}
else {
return 0;
}
}
}
```

Adicionalmente puede darse el ejemplo del factorial de una función,

```
int factorial ( int a){
int contador ;
int fact =1;
for ( contador =1; contador <=a; contador ++) {
fact = fact * contador ;
}
return fact ;
}
```

En este último ejemplo puede verse como se utiliza un ciclo *for* para ejecutar, el conteo de enteros hasta llegar al valor dado como parámetro. Puede observarse en este mismo ejemplo como el operador de asignación tiene una prioridad más baja que el operador de multiplicación, en la sentencia fact=fact*contador; se ejecuta primero la multiplicación con el valor que posee la variable fact y luego se asigna el resultado de nuevo a ella sustituyendo que ya tenía almacenado. Para utilizar las funciones basta con llamarlas por su nombre, o identificador, y a la par la lista de parámetros consistente con su declaración.

Por ejemplo:

```
int num1 = -10, num2 =6;
suma (num1 , num2 ); // Devolver á como resultado -4.
signo ( num1 ); // Devolver á como resultado -1.
factorial ( num2 ); // Devolver á como resultado 720.
```

En los ejemplos anteriores se puede ver que las funciones devuelven un valor determinado, sin embargo, es posible construir funciones que no lo hacen, este tipo de funciones ejecutan una serie de sentencias sin devolver algún valor. El tipo de estas funciones es *void* y no llevan la sentencia *return* en ningún lugar del bloque de sentencias enmarcado por las llaves, ya no tiene sentido colocarla porque estas funciones no devuelven valor alguno.

3.6.2.17. Directivas del compilador

En un programa escrito en C, es posible incluir instrucciones para el compilador dentro del código del programa. Estas instrucciones no son traducidas a código de máquina ya que no tendrían un equivalente y sirven para modificar el comportamiento del compilador al interpretar el programa. Estas instrucciones dadas al compilador son llamadas directivas del preprocesador, aunque realmente no son parte del lenguaje C, expanden la capacidad del entorno de programación de C. Es posible mencionar alguna de estas directivas.

- #define. La directiva #define se usa para definir un identificador y una cadena que el compilador sustituirá por el identificador cada vez que se encuentre en el archivo que almacena el código fuente. Por ejemplo, la sentencia: “#define TRUE 1”, sustituirá por 1 cada vez que encuentre en el código fuente la palabra TRUE.

  Una característica interesante de esta directiva es que se pueden definir macros con argumentos. Debe quedar claro que estas instrucciones no

son transformadas por el compilador en código de máquina, solamente sustituye cadenas en el código fuente. En el caso que un macro posea parámetros, realizará la sustitución tomando en cuenta los parámetros dados. Por ejemplo:

```
#define MIN(a,b) if(a<b) ? a : b
```

Por ejemplo si el compilador encuentra en el programa una cadena MIN(x,y), sustituirá dicha cadena por:

```
if(x<y) ? x : y
```

y luego se generará el código de máquina correspondiente a las sentencias de C.

- #undef. La directiva #undef permite retirar cualquier definición hecha con la directiva define. Para indicar la definición no deseada, basta con llamar a la directiva y el nombre con que fue construida, tal como se muestra a continuación: #undef <nombre>. Por ejemplo, la sentencia

  ```
  #undef TRUE
  ```

  indicaría al compilador que la cadena TRUE ya no se encuentra definida para las siguientes sentencias.

- Condicionales de compilación. Las directivas #if, #ifdef, #ifndef, #else, #elif y #endif, permiten decirle al compilador que partes del programa debe compilar bajo distintas condiciones. El código 3.1 muestra un ejemplo de como utilizar estas directivas para compilar determinados sectores de un programa, dependiendo de condiciones establecidas.

Tabla LXXIII. **Código 3.1: ejemplo de condicionales de compilación**

```
# define VAL 20
# define EJ 0
# defin LengC
#if VAL > 100
Có digo a compilar .
...
# endif
#if EJ ==0
Có digo a compilar .
...
# else
Có digo a compilar .
...
# endif
# ifdef LengC
Có digo a compilar .
...
# endif
```

Fuente: elaboración propia, empleando C/tilaware.

El uso de estas directivas puede ser muy útil ya que permite definir plantillas o programas que se adapten a diferentes necesidades solamente cambiando constantes definidas por #define.

- #include. Esta directiva obliga al compilador a incluir otro archivo con código fuente en el que tiene la directiva #include y compilarlo. El nombre del archivo puede ir entre los signos < y > o bien entre comillas "". Las diferencias entre ellos dependen del compilador. En muchos compiladores, cuando se usan los símbolos < y >, estos buscan el archivo en una lista llamada *include path list*. Cuando se utilizan comillas buscará primero en el archivo local y luego iran a la *include path list*.

Tabla LXXIV. **Código 3.2: ejemplo de inclusión de archivos en otros**

```
# include <stdio .h>
# include <stdint .h>
# include " inc / driverlib .h"
```

Fuente: elaboración propia, empleando C/tilaware.

3.6.2.18. Conversión entre tipos

Es posible en C, que una expresión contenga diferentes tipos de datos, siendo el compilador el encargado de realizar las operaciones de forma correcta. Cuando un compilador de C encuentra que en una misma expresión aparecen dos o más tipos de datos, convierte todos los operandos al tipo del operando más grande que aparece en la expresión siguiendo las siguientes reglas:

- Todas las variables de tipo char y *short int* se convierten a *int*. De la misma forma todas las variables de tipo float a double.
- Para todo par de operandos, las siguientes verificaciones suceden en orden:
    - Si uno de los operandos es un *long double*, el otro se convierte en *long double*.
    - Si uno de los operandos es *double*, el otro se convierte a *double*.
    - Si uno de los operandos es *long*, el otro se convierte a *long*.
    - Si uno de los operandos es *unsigned*, el otro se convierte a *unsigned*.

Después de que el compilador aplique estas reglas de conversión, cada par de operandos será del mismo tipo, y el resultado será del tipo de los operandos.

Es posible forzar una conversión de tipos de datos. Esta conversión forzada se conoce como *cast*. Por ejemplo:

```
int a=3, b =2;
float c;
c=a/b;
```

Almacenará en la variable c el valor de 1 debido a que la operación se efectuó entre dos enteros. Sin embargo,

```
int a=3, b =2;
float c;
c=( float )a/b;
```

Almacenará en la variable c el valor 1,5, ya que (*float*)a cambiará el tipo de la variable a (esto lo hará solamente en la expresión, no en el resto del programa) a *float* forzando que las reglas de conversión implícita realicen una operación de flotantes generando como valor 1,5.

3.7. Diseño de una dinámica discreta

La implementación de una dinámica discreta en un dispositivo de electrónica digital es la mejor forma de aplicar los conceptos mencionados con anterioridad. Para ello es preciso escoger un dispositivo que se adapte a las circunstancias.

Implementar una dinámica discreta, por sencilla que sea, en un dispositivo electrónico requiere de cierto grado de conocimiento técnico del dispositivo que difiere de dispositivo a dispositivo; pudiendo cambiar completamente el paradigma del uso de un dispositivo a otro, por lo que desarrollar la implementación de una dinámica discreta de forma general resulta imposible para un solo texto, es el uso de un dispositivo en específico requeriría más de un texto debido a la gran cantidad de tecnicismos que son necesarios para abarcarlo completamente.

En este texto se trata de forma muy superficial algunos tópicos acerca de la implementación de un sistema discreto en un microcontrolador. Se ha escogido para el presente trabajo, el uso de microcontrolador es, debido a que se han popularizado en los últimos años, estando al alcance de las masas. Ya no es necesario poseer un conjunto de conocimientos esotéricos solo al alcance de personas extremadamente involucradas en el tema, existen comunidades en línea y la mayoría de los microcontroladores poseen muy buena documentación.

Además, en este trabajo se utilizará la plataforma de desarrollo TivaC, la cual cuenta con un microcontrolador con suficientes periféricos que lo hacen adaptable a varios sistemas que tengan alguna interacción con el entorno. Se basa en una arquitectura ARM Cortex- M4 cuyo uso permite abordar algunos aspectos relacionados con la implementación técnica, ofreciendo la oportunidad de contrastar la solución a nivel de ingeniería y el modelo matemático de una dinámica discreta.

3.7.1. Tiva c. Microcontrolador TM4C123GH6PM y librería Tivaware

La tarjeta de desarrollo TivaC es una plataforma de bajo costo para microcontrolador es basados en el ARM®Cortex-M4. La tarjeta posee una interfaz USB 2.0 con un microcontrolador TM4C123GH6PM, módulo de hibernación y el módulo de control movimiento por modulación de ancho de pulso (MC PWM). La Tiva C posee botones y un LED RGB para aplicaciones prácticas. Posee *pin headers* machos y hembra que la hacen versátil para muchas aplicaciones. Una Tiva C Launchpad posee las siguientes características:

- Microcontrolador Tiva TM4C123GH6PM.
- *Motion control* PWM.
- Conectores USB micro-A y micro-B para los dispositivos USB, almacenamiento, un LEDOTG (on-the-go) RGB para el usuario.
- Dos *switch* para el usuario.
- I/O conectados a *pin headers* hembra y macho de 0,1 pulgadas (0,254 mm).
- ICDI on-board.
- *Reset switch.*
- Soporte por TivaWare, como librerías para USB y periféricos

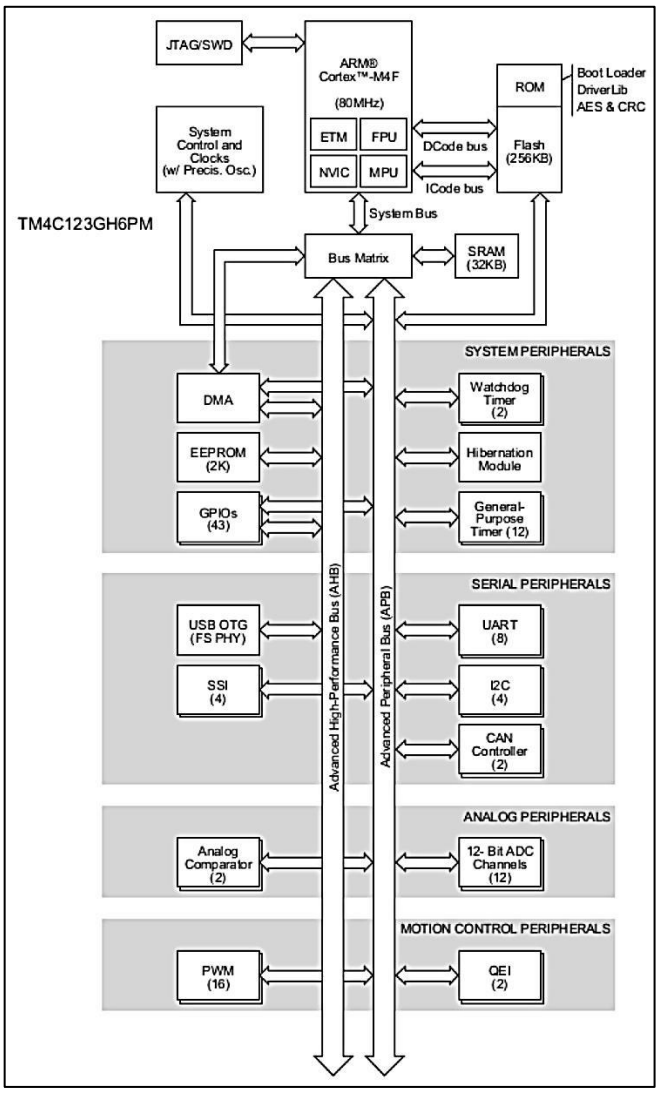
Un estudio completo de este microcontrolador es sumamente extenso, y su uso tedioso sin una librería que permita el uso de sus periféricos. Texas Instruments ha desarrollado la TivaWare, la cual presenta una licencia de uso *Royalty Free* y es posible utilizarla sin pagar por el uso.

- Nota: esta sección es una guía de como utilizar algunos periféricos del microcontrolador TM4C123GH6PM utilizando la librería TivaWare; el propósito de esta sección es demostrativo. Para unamejor referencia de un dispositivo se debe acudir a la hoja de datos. Para un buen uso y comprensión de la librería también se debe acudir a la documentación oficial.

3.7.2. Arquitectura del TM4C123X y configuración inicial

Para utilizar un microcontrolador se debe estar familiarizado con la arquitectura del mismo. En la siguiente figura se muestra un diagrama de bloques de los periféricos y buses que componen el microcontrolador tm4c123gh6pm.

Figura 78. **Diagrama interno del microcontrolador TM4C123GH6PM**



Fuente: *Tiva TM TM4C123GH6PM Microcontroller Datasheet*.
http://www.ti.com/lit/ds/spms376e/spms376e.pdf.
Consulta: agosto de 2015.

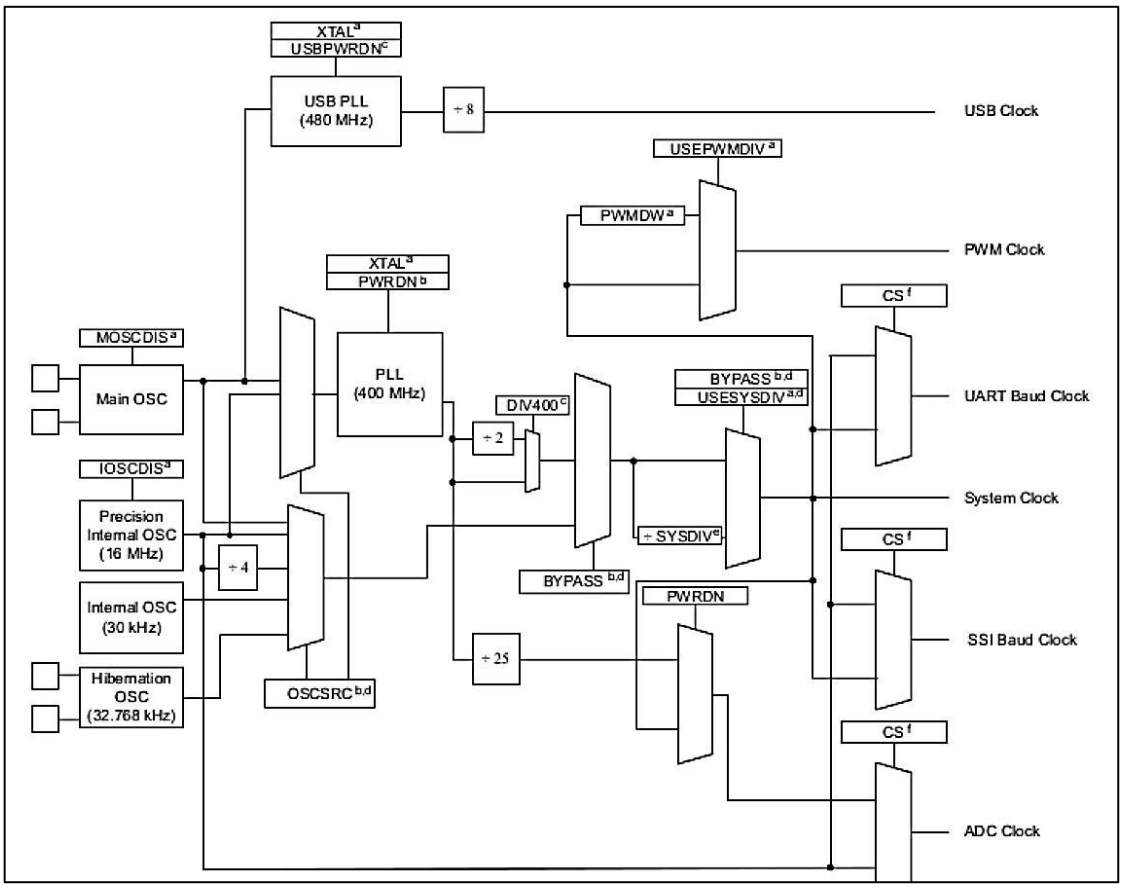
Como puede verse, hay varios buses para transmisión de información, cada uno dedicados para diferentes fines, pudiéndose así apreciar una arquitectura Harvard, la cuál separa buses para programa y datos del usuario.

El microprocesador de este microcontrolador es un ARM Cortex-M4 con arquitectura de 32 bits. Conocer el microcontrolador es muy importante para poder utilizarlo de forma correcta y poder entender algunas implicaciones que los programas que corren en él llevan. Llevaría varios textos abarcar tanto la arquitectura CortexM4 como la implementación de ella en el microcontrolador con todos los periféricos. A continuación se centra en un uso sumamente básico con el único fin de mostrar la implementación de dinámicas discretas en él.

Trabajar directamente con los registros del microprocesador es una tarea ardua y extensa, además de requerir un mayor conocimiento de la arquitectura. Por suerte se cuenta con un compilador que puede facilitar esta tarea y así evitar programar en código de máquina.

Configurar correctamente el reloj en un microcontrolador posiblemente sea la parte más importante en su uso, por lo que se comenzará con ello. En la figura 79 se muestra el diagrama de configuración del módulo de reloj. La hoja de datos del dispositivo indica que los registros (mapeados en la memoria) RCC y RCC2 son los encargados de manipular la configuración del reloj.

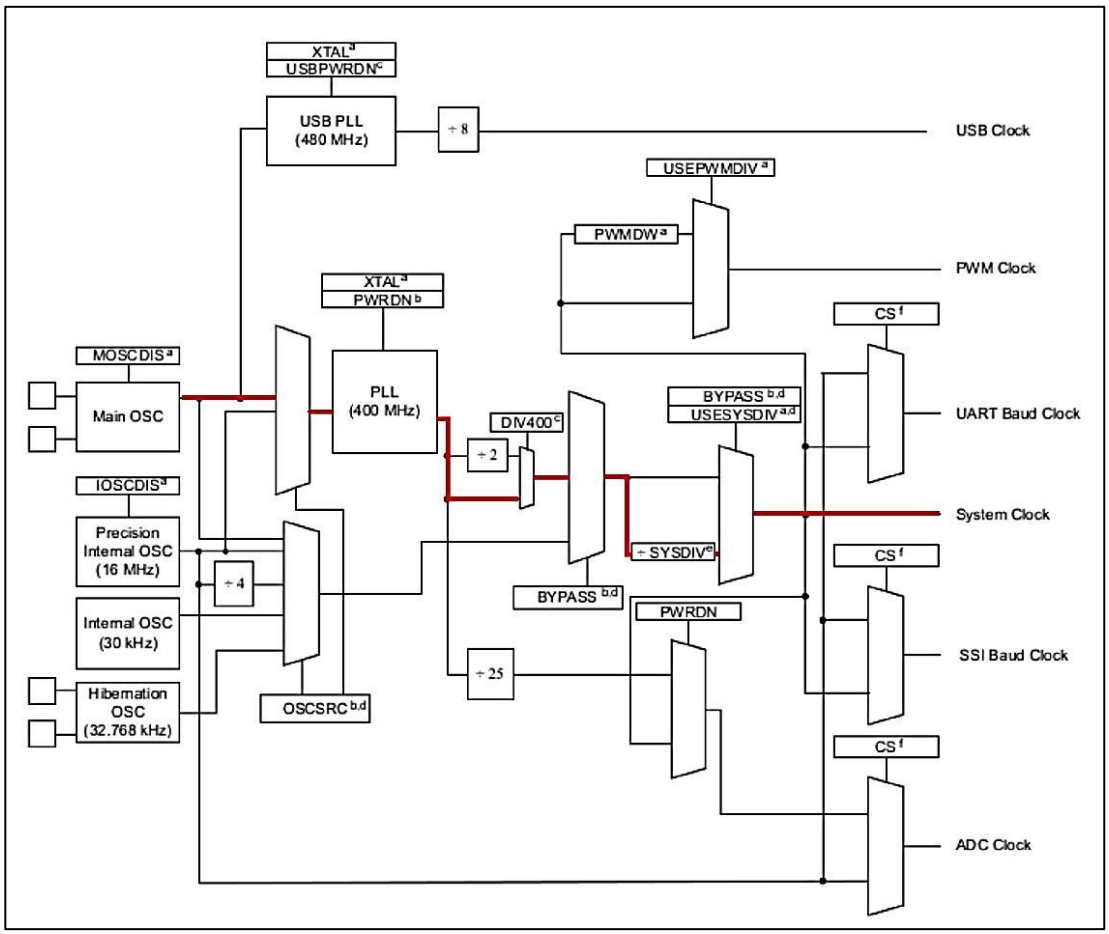
Figura 79. **Reloj TM4C123GH**



Fuente: *Tiva TM TM4C123GH6PM Microcontroller Datasheet*.
http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: agosto de 2015.

La tarjeta de desarrollo TivaC posee un cristal de 16 Mhz conectado como fuente principal de reloj. La implementación del Cortex-M4 en este procesador, soporta un reloj de hasta 80 Mhz. El microcontrolador posee un módulo de reloj con un PLL, multiplicadores y divisores de frecuencia que permiten entregar esta señal de reloj, si es configurado correctamente.

Figura 80. **Camino configurado por registros en módulo de reloj**



Fuente: Modificación hecha imagen tomada de *Tiva TM TM4C123GH6PM Microcontroller Datasheet*. http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: agosto de 2015.

Obtener la configuración (línea roja) que se muestra en la figura 80, se logra modificando los registros RCC y RCC2 con los valores que se presentan en la hoja de datos. El valor que almacenen estos registros determina el camino que tomará la señal proveniente del reloj. Para asegurar que el módulo de reloj entregue una señal de reloj con la frecuencia deseada, se debe configurar correctamente el módulo. En la hoja de datos se dan los siguientes pasos:

- Bordear el PLL y el divisor del reloj del sistema colocando un lógico en el bit BYPASS y poniendo en bajo el bit USESYS en el registro RCC, por tanto configurando el microcontrolador para que corra una fuente de reloj cruda11, permitiendo para la nueva configuración ser validada antes de cambiar el reloj del sistema al PLL.

- Seleccionar el valor del cristal en el campo XTAL y la fuente de oscilación en el campo OSCSRC y limpiar el bit PWRDN en los registros RCC/RCC2. Configurar el campo XTAL automáticamente traslada una configuración de PLL válida para el cristal apropiado y limpiar el bit PWRDN habilita y alimenta el PLL.

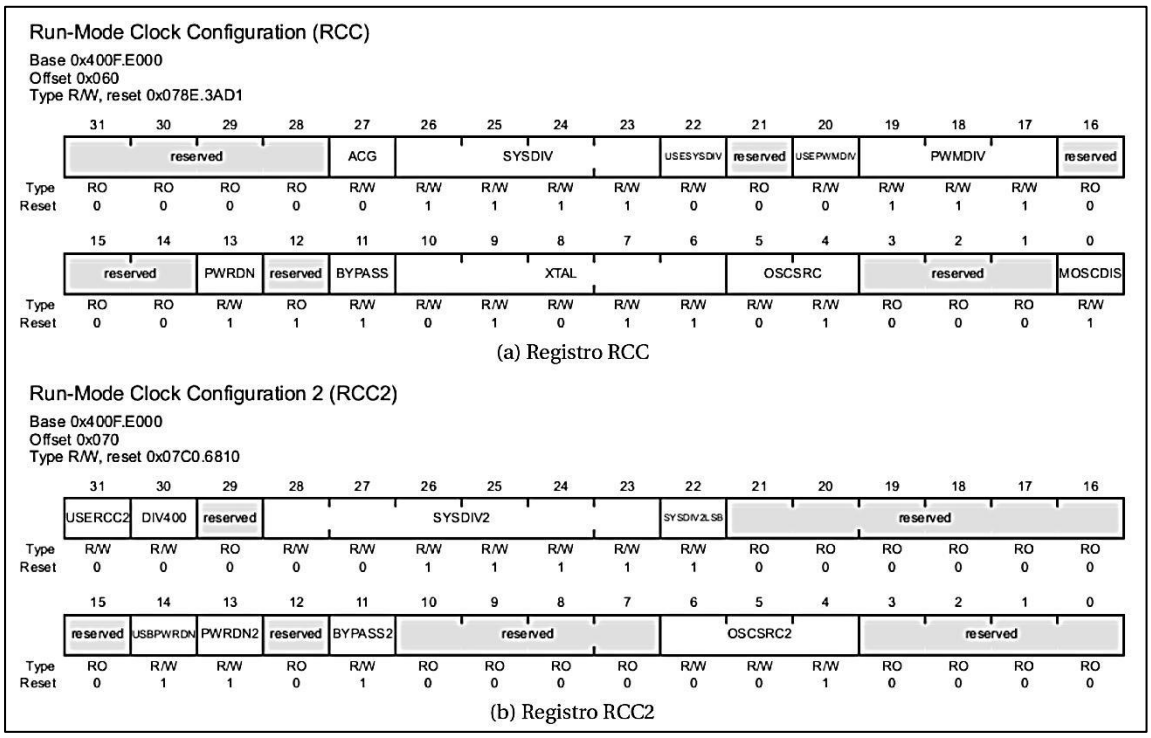
- Seleccionar el divisor del sistema deseado en el registro RCC/RCC2 y se configura el bit USESYS en el registro RCC. El campo SYSDIV determina la frecuencia del sistema para el microcontrolador.

- Esperar a que el PLL se enganche, el microcontrolador levantará una bandera poniendo un 1 lógico en el bit PLLLRIS, en el registro RIS (Raw Interrupt Status). Para esto se hace un *polling* a dicho bit.

- Habilitar el PLL poniendo un 0 lógico el bit de BYPASS en RCC/RCC2.

En las siguientes imágenes se pueden observar como se estructuran los registros RCC y RCC2.

Figura 81. **Registros RCC y RCC2**



Fuente: *Tiva TM TM4C123GH6PM Microcontroller Datasheet*.
http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: agosto de 2015.

Si el registro RCC2 es utilizado, se debe colocar un 1 lógico en el bit USERCC2 y utilizar el campo o bit apropiado. El registro RCC2 tiene campos que ofrecen mayor flexibilidad que el registro RCC. Cuando el registro RCC2 es utilizado, los valores de sus campos y bits son utilizados en lugar de los campos del registro RCC. De forma particular el registro RCC2 provee una mayor cantidad de configuraciones que el registro RCC, pero no se debe de olvidar la advertencia que da la hoja de datos acerca de su uso. Indica que se debe escribir en el registro RCC antes de hacerlo en el registro RCC2. Si es necesario escribir en el registro RCC luego de hacerlo en el registro RCC2, se debe acceder a otro registro después de acceder a RCC2 y antes de RCC.

Como puede verse en las figuras 83a y 83b los registros pueden ser accedidos usando direcciones de memoria, se ve que el registro tiene una dirección base y un *offset*. La forma de determinar la dirección del registro es sumando la base con el *offset*. En el lenguaje C se utiliza un puntero para acceder a una dirección de memoria. Por ejemplo

```
(*(( volatile unsigned long *)0 x400FE060 ));
```

permite acceder al registro RCC. Para entender la instrucción anterior es necesario seccionarla en partes. Si se escribe

```
(0 x400FE060 );
```

simplemente es un número que representa la dirección dememoria de RCC pero no el valor que se encuentra en esa dirección. Es necesario decirle al compilador que se trata de una dirección de memoria. Entonces se utiliza un puntero a memoria,

```
(*0 x400FE060 );
```

Esto intentará leer la dirección, pero no sabrá si leer 8, 16 o 32 bits por lo que el compilador marcará un error. Para resolverlo se utiliza una conversión forzada, colocando un nuevo tipo enfrente del objeto a convertir y forzando al compilador leer 32 bits, así que se agrega (unsigned long *) frente a la dirección de memoria. Pero dado que el compilador realiza una serie de simplificaciones antes de convertir de C a código de máquina, se utiliza el modificador volatile para evitar que exista algún conflicto ya que se está trabajando directamente con el hardware. Por lo que la sentencia se modifica a

```
(*(( volatile unsigned long *)0 x400FE060 ));
```

Escribir esto por cada vez que se utiliza el registro resulta mucho trabajo para el programador por lo que colocar al principio del programa la siguiente directriz:

```
# define RCC (*(( volatile unsigned long *)0 x400FE060 ))
```

Hará la sustitución de RCC por (*((volatile unsigned long *)0x400FE060)) en el programa antes de pasarlo a código de máquina. De igual forma, resulta trabajoso escribir las direcciones de todos los registros del procesador, por suerte existe una librería que se compone de todas las directivas de compilador para hacer la sustitución de cada dirección por una cadena identificando al registro de forma más amigable al programador. La librería es parte del conjunto de librerías para este microcontrolador, TivaWare. Para indicarle al compilador que todas las directrices están en esa librería se utiliza la directiva #include, con la dirección de la librería:

```
# include " inc / tm4c123gh6pm .h"
```

En la librería los identificadores de los registros varían ligeramente del nombre original asignado en la hoja de datos y dependen del periférico al que pertenecen y la función que realizan. Por ejemplo el nombre en la librería, para el registro RCC es SYSCTL_RCC_R, la parte del nombre SYSCTL hace referencia a control del sistema (en inglés *System Control*) y la R indica que es un registro, ya que la librería puede albergar definiciones para constantes, entre otros.

A continuación, se muestra una rutina escrita en C que configura el módulo de reloj:

Tabla LXXV. **Código 3.3: rutina de configuración de reloj**

```
1 void Clock_Init ( void ){
2 SYSCTL_RCC_R |= 0 x00000800 ; // Pone un 1 ló gico el bit BYPASS .
3 SYSCTL_RCC_R &= ~(0 x00400000 ); // Pone un 0 ló gico los bits USESYS .
4 SYSCTL_RCC_R &= ~(0 x000007C0 ); // Limpia el campo XTAL . Coloca ceros .
5 SYSCTL_RCC_R |= 0 x00000540 ; // Coloca el valor del cristal a 16 Mhz .
6
7 // Configura el PLL para usar 400 Mhz y aumentar la resoluci ón.
8 SYSCTL_RCC2_R |= 0 xC0000000 ;
9
10 SYSCTL_RCC2_R |= 0 x00000800 ; // Pone un 1 ló gico el bit BYPASS en RCC2 .
11
12 // Pone en bajo el bit PLLPWRDN para que el PLL empiece a funcionar .
13 SYSCTL_RCC2_R &=~(0 x00002000 );
14
15 SYSCTL_RCC2_R &=~(0 x1FC00000 );// Limpia el campo SYSDIV2 y SYSDIV2LSB
16 SYSCTL_RCC2_R |= 0 x01000000 ;// Indica el divisor en estos campos .
17
18 // Espera que el bit RIS de este registro se ponga en alto para indicar que
19 // el PLL corre correctamente .
20 while (( SYSCTL_RIS_R &0 x00000040 ) ==0)
21 {};
22
23 // Pone a funcionar el PLL poniendo en bajo el bit BYPASS .
24 SYSCTL_RCC2_R &=~(0 x00000800 );
25 }
```

Fuente: elaboración propia, empleando C/tilaware.

Solamente se están siguiendo los pasos que se indican en la hoja de datos para obtener la configuración que se muestra en la figura 82.

Se utilizan operadores binarios en lugar de solo escribir el valor que corresponde para evitar sobrescribir valores que ya se tienen con anterioridad.

Si se desea colocar un 1 lógico en algún bit o conjunto de bits sin alterar los demás, basta con hacer una disyunción lógica entre cero y los bits del registro que no se desean alterar, y hacer la disyunción entre uno y los bits del registro que se desean con un 1 lógico. Tal como la instrucción:

SYSCTL_RCC2_R |= 0 x00000800;

Esta operación asegura que el doceavo bit tenga un 1 lógico y que los demás bits queden con el valor que poseen.

Si se desea colocar un 0 lógico en algún bit o conjunto de bits sin alterar los demás, basta con hacer una conjunción lógica entre 1 y los bits del registro que no se desean alterar, y hacer la conjunción entre ceros y los bits del registro que se desean con un 1 lógico. Tal como la instrucción:

SYSCTL_RCC2_R &=~(0 x00002000);

Esta operación asegura que el catorceavo bit tenga un 0 lógico y que los demás bits queden con el valor que poseen. En esta instrucción se hace primero la negación de 0x00002000 dando como resultado 0xFFFFDFFF y es con este valor que se hace la conjunción bit a bit. A esta forma de colocar valores en registros y variables se le conoce como *friendly coding* ya que no altera los valores que se tenían anteriormente de los bits que no se desean modificar, solamente se alteran los que se desean modificar.

El conjunto de librerías TivaWare trae una librería especial dedicada a periféricos de control del sistema que evita escribir una rutina como Clock_Init. La rutina SysCtlClockSet, realiza un procedimiento parecido a la rutina Clock_Init mostrada con anterioridad, con la diferencia que permite ingresar parámetros para manipular la frecuencia de reloj que se le entrega al procesador, entre otras configuraciones.

SysCtlClockSet (SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);

La sentencia anterior muestra una configuración equivalente a la otorgada al módulo de reloj por la rutina Clock_Init, utilizando la librería TivaWare. En esta sentencia se utilizaron palabras reservadas por la librería que indican configuraciones a los campos de los registros RCC y RCC2. Al igual que las directivas con identificadores para los registros mapeados en la memoria, existen definiciones para valores fijos. Si se indaga en la librería driverlib/sysctl.h es posible encontrar sentencias como:

```
# define SYSCTL_SYSDIV_2_5 0 xC1000000
```

Se puede apreciar que esta directiva de compilador realizará una sustitución, antes de pasar a código de máquina, de la palabra SYSCTL_SYS_2_5 por el valor 0xC1000000, cada vez que la encuentre en el programa.

Si se es operador puede verse que el valor a sustituir por SYSCTL_SYSDIV_2_5 es el resultado de la operación 0x01000000|0xC0000000 que son las configuraciones que se han dado en la rutina Clock_Init, para que no se aplique el divisor a la señal proveniente del PLL y se entregue una señal de reloj de 80 Mhz al procesador. De la misma forma puede verse que los identificadores SYSCTL_XTAL_16MHZ, SYSCTL_OSC_MAIN y SYSCTL_USE_PLL en las directivas para valores 0x00000540, 0x00000000 y 0x00000000 respectivamente. Cada uno brindando configuraciones específicas a los registros de control del módulo de reloj. Para poder hacer uso de estas constantes, funciones y procedimientos ya desarrollados en la librería se debe agregar utilizando la directiva:

```
# include " driverlib / sysctl .h"
```

Al igual que para la librería tm4c123gh6pm.h, se le debió indicar en algún momento que la librería forma parte de los archivos a tomar en cuenta a la hora de compilar, agregando la ruta de ubicación de la librería en algún parámetro del compilador que se encuentra utilizando.

3.7.3. Uso de las entradas de propósito general. GPIO

Una vez configurado el reloj, es posible configurar otros periféricos. Las entradas y salidas de propósito general permiten comunicar el mundo exterior con la lógica del programa corriendo en el interior del microprocesador.

El tm4c123gh6pm tiene 6 módulos de GPIO, sumando un total de 43 pines con posibilidad de formar una entrada o salida de propósito general, todos ellos distribuidos en 6 puertos etiquetados con una letra de la A a la F; un puerto puede tener hasta 8 pines. Los pines toleran hasta 5*V* de tensión en caso de ser configurados como entrada.

Como puede verse en la figura 80 los módulos de GPIO pueden ser accedidos por dos buses diferentes, el bus APB (siglas en inglés de *Advanced Peripheral Bus*) y el bus AHB (siglas en inglés de *Advanced High-Per*for*mance Bus*). El uso del primero permite compatibilidad con dispositivos anteriores. El uso del segundo permite mejor desempeño en el acceso que el uso primero.

El uso de estos buses es mutuamente excluyente. No pueden usarse al mismo tiempo. Los módulos de GPIO cuando son accedidos mediante el AHB pueden conmutar en un ciclo de reloj, a diferencia de cuando son accedidos por el APB tardarán dos ciclos de reloj en conmutar. En aplicaciones sensibles al consumo de energía, el APB presenta una mejor alternativa a utilizar.

En la figura 84 se muestran las bases de los registros relacionados a los módulos de GPIO dependiendo del bus con que se está accediendo. Por ejemplo, para acceder algún registro relacionado al GPIO puerto A, utilizando el APB, tendrá como base 0x40004000. Sin embargo, si accede a un registro de este mismo puerto por medio del AHB tendrá como base 0x40058000; el *offset* será el mismo cambiando únicamente la base dependiendo del bus de acceso. La librería TivaWare por defecto utiliza el APB.

El módulo de GPIO permite flexibilidad para multiplexar en los pines otros periféricos con funciones más específicas. Posee compuertas *Schmitt-triggered* para homologación de los niveles de voltaje y es posible configurar la corriente máxima que pasa a través de los dispositivos con valores de 2, 4 y 8 mA.

Figura 82. **Base de los registros relacionados a los módulos de GPIO dependiendo del bus de acceso y el puerto**

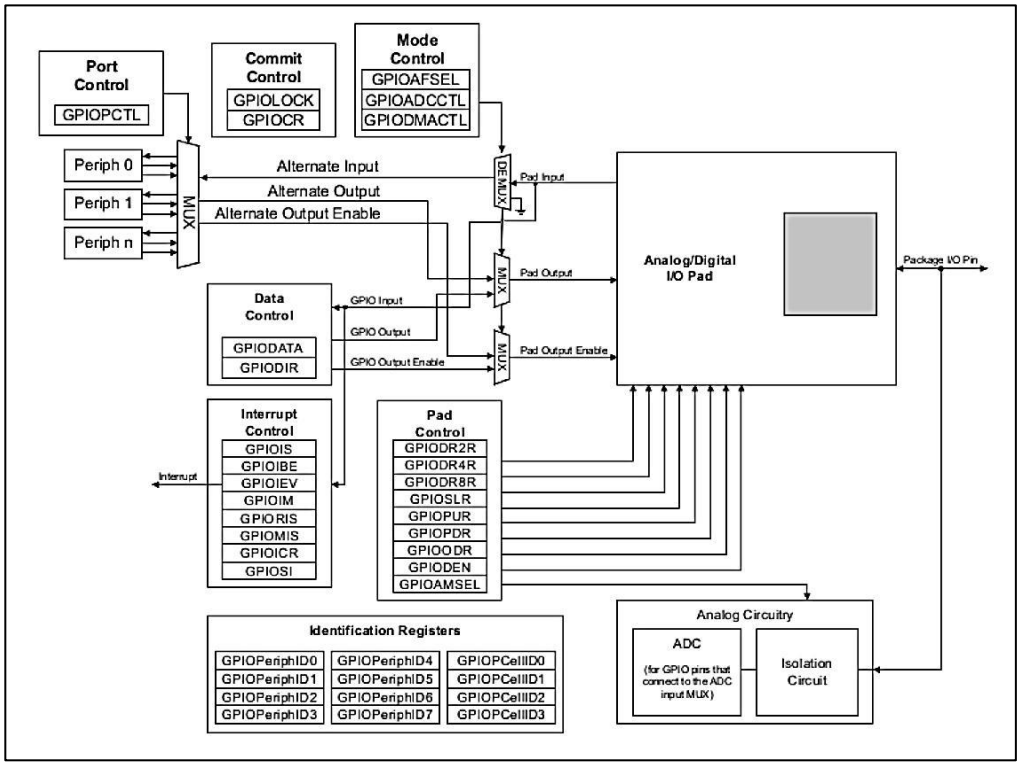
```
GPIO Port A (APB): 0x4000.4000
GPIO Port A (AHB): 0x4005.8000
GPIO Port B (APB): 0x4000.5000
GPIO Port B (AHB): 0x4005.9000
GPIO Port C (APB): 0x4000.6000
GPIO Port C (AHB): 0x4005.A000
GPIO Port D (APB): 0x4000.7000
GPIO Port D (AHB): 0x4005.B000
GPIO Port E (APB): 0x4002.4000
GPIO Port E (AHB): 0x4005.C000
GPIO Port F (APB): 0x4002.5000
GPIO Port F (AHB): 0x4005.D000
```

Fuente: *TM4C123g LaunchPadWorkshopWorkbook*. https://training.ti.com/tm4c123g-launchpad-workshop-series-2-15-introduction-code-composer-studio. Consulta: agosto de 2015.

La configuración inicial de alguno de los seis periféricos es muy similar a la configuración del reloj manipulando registros para cambiar la forma en que se

comporta el módulo. En la figura 82, se muestra el diagrama de bloques de un módulo de GPIO.

Figura 83. **Modelo de bloques general de un módulo de GPIO**



Fuente: *Tiva TM TM4C123GH6PM Microcontroller Datasheet*. http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: agosto de 2015.

En la hoja de datos puede encontrase el proceso a seguir para dar una configuración inicial al periférico, para ello son necesarios más registros que para configurar el reloj del sistema. Para un periférico en particular se deben seguir los siguientes pasos, descritos con los nombres de los registros como aparecen en la hoja de datos:

- Habilitar el reloj del puerto configurando los bits apropiados en el registro RCGCGPIO. También se pueden configurar los registros SCGCGPIO y DCGCPIO para habilitar el reloj en los modos *sleep* y *deep-sleep* respectivamente, que el microcontrolador soporta. El microcontrolador ofrece registros más generales para la habilitación del reloj. El registro que puede ser utilizado en lugar de RCGCGPIO es el RCGC2.

- Configurar la dirección de los puertos en el registro GPIODIR. Un 1 lógico en este registro indica salida y un cero lógico significa entrada.

- Configurar el registro GPIOAFSEL para programar cada uno de los bit como GPIO o función alternativa. Si un pin es escogido para tener una función alternativa, el campo PCMx debe ser configurado en el registro GPIOPCTL para el periférico requerido. Hay dos registros, GPIOADCCTL y GPIODMACTL, los cuales pueden ser utilizados para programar un pin de GPIO para levantar un *trigger* de ADC o de [1]DMA.

- Configurar la corriente que soportará en cada unos de los pines a través de los registros GPIODR2R, GPIODR4R y GPIODR8R.

- Programar cada uno de los pines para tener la funcionalidad de *pull-up*, *pull-down* o de colector abierto, a través de los registros GPIOPUR, GPIOPDR y GPIOODR. El *slew-rate* también puede ser configurado, si es necesario, a través del registro GPIOSLR.

- Para habilitar los pines como entradas/salidas digitales de propósito general, se debe colocar un 1 lógico en el bit DEN del registro GPIODEN. Para habilitar la función analógica en los pines (en caso se encuentre

disponible), se debe poner un 1 lógico en el bit GPIOAMSEL en el registro GPIOAMSEL.

- Dar el valor a los registros GPIOIS, GPIOIBE, GPIOBE, GPIOEV y GPIOIM para configurar el tipo, evento y máscara de las interrupciones en cada puerto.

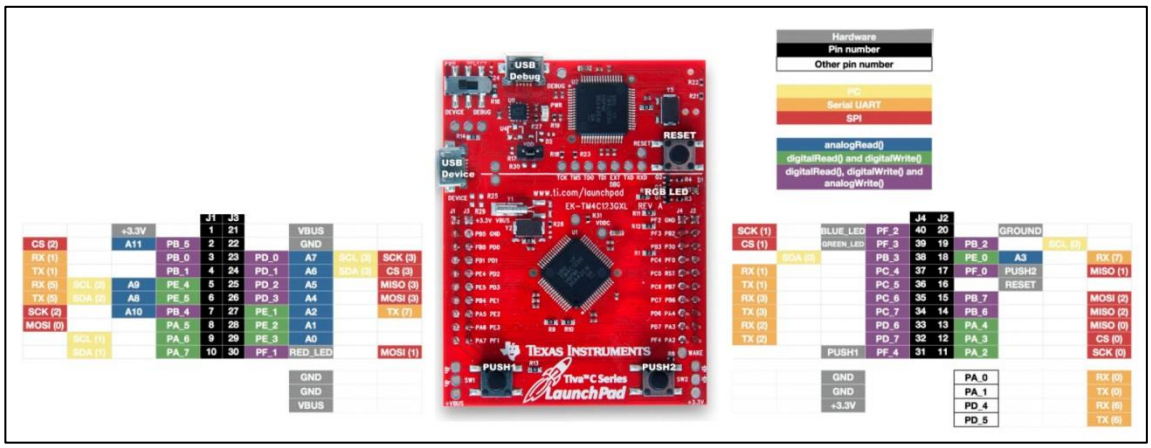
- Opcionalmente, el software puede bloquear las configuraciones de los pines con interrupciones no enmascarables y *JTAG/SWD* en el bloque GPIO, poniendo en alto los bits LOCK en el registro GPIOLOCK.

Del procedimiento descrito por la hoja de datos se resalta el hecho que se le entrega una señal de reloj al periférico para poder funcionar; de hecho en este microcontrolador cualquier periférico necesita de una señal de reloj para funcionar.

Debe de tomarse en cuenta que el procedimiento descrito por la hoja de datos dependerá del módulo de GPIO que se está configurando, ya que las entradas y salidas de varios periféricos es posible conectarlas a pines externos del microcontrolador. O como se desee ver, con que pines se puede acceder desde el entorno a determinados periféricos.

En la figura 84 se pueden ver los periféricos que se pueden encontrar en los pines. A dicha figura se le conoce como mapa de pines.

Figura 84. **Mapa de periféricos a pines del microcontrolador**



Fuente: *TI ARM Cortex-M LaunchPad Programming by Example*.
http://energia.nu/wordpress/wp-content/uploads/2014/01/tm4c123pinmap.png.
Consulta: agosto de 2015.

Por ejemplo, si se desea configurar el módulo F de GPIO se puede utilizar la siguiente rutina escrita en C, donde los pines 1, 2 y 3 se encontrarán como salidas digitales.

Tabla LXXVI. **Código 3.4: rutina de configuración de GPIO F**

```
1 void PortF_Init ( void ){
2 volatile unsigned long delay ;
3
4 SYSCTL_RCGC2_R |=0 x00000020 ; // Da reloj al perif é rico
5 delay = SYSCTL_RCGC2_R ; // Espera un ciclo .
6
7
8 // Coloca como salida el Pin1 ,Pin2 , Pin3 .
9 GPIO_PORTF_DIR_R |=0 x0000000E ;
10
11
            12 GPIO_PORTF_LOCK_R =0 x4C4F434B ; // Permite cambios en el puerto F.
13 GPIO_PORTF_CR_R |=0 x1F ; // Permite cambios en el puerto F.
```

Continuación de la tabla LXXVI.

```
14
15 // Deshabilita funciones alternativas en los pines .
16 GPIO_PORTF_AFSEL_R &=~(0 x0000000E );
17
18 // Deshabilita las funciones de perifericos en los pines
19 GPIO_PORTF_PCTL_R &=~(0 x0000FFF0 );
20
21 // Deshabilita funciones de ADC en los pines .
22 GPIO_PORTF_ADCCTL_R &=~(0 x0000000E );
23
24 // Deshabilitamos funciones de DMA en los pines .
25 GPIO_PORTF_DMACTL_R &=~(0 x0000000E );
26
27 // Permite como corriente de la salida a lo más 2 mA.
28 GPIO_PORTF_DR2R_R |=0 x0000000E ;
29
30 GPIO_PORTF_PUR_R |=0 x0000000E ; // Habilita las Pull Up ’s.
            31 GPIO_PORTF_DEN_R |=0 x0000000E ; // Configura como salidas digitales .
32
33 // Deshabilita modo anal ó gico en los pines .
34 GPIO_PORTF_AMSEL_R &=~(0 x00000001 );
35 }
```

Fuente: elaboración propia, empleando C/tilaware.

Para escribir la rutina anterior no hubo necesidad de buscar la base y el *offset* de los registros que corresponden a la configuración del GPIO F, ya que se utiliza la librería tm4c123gh6pm.h; sin embargo, para conocer la posición exacta de los bits de configuración en determinado registro o el comportamiento que define al módulo de los registros, sí es necesario consultar la hoja de datos del dispositivo.

Algunos periféricos vienen por defecto protegidos contra programación accidental de algunos pines. Los pines del 0 al 3 del Puerto C, el pin 7 del puerto D y el pin 0 del puerto F. Escribir a los bits protegidos de los registros GPIOAFSEL, GPIOPUR, GPIOPDR y GPIODEN no se lleva a cabo hasta que

los registros GPIOLOCK y GPIOCR hayan sido desbloqueados colocando los valores correctos de desbloqueo en los bits de estos registros.

Al igual que existen funciones, rutinas y constantes entre la TivaWare ofrecida por Texas Instruments para el control del módulo del reloj, existen para realizar configuraciones y manipulación de los registros de los periféricos GPIO. Para ello es necesario agregar las librerías hw_memmap, hw_types.h y gpio.h. Para ello es necesario colocar las directrices de compilador:

```
1 # include " inc / hw_memmap .h"
2 # include " inc / hw_types .h"
3 # include " driverlib / gpio .h"
```

Además que debe de entregarse una señal de reloj al periférico es necesario incluir la librería driverlib/ sysctl.h, aunque siempre será necesaria si se configura el reloj con esta librería.

El equivalente de realizar la configuración con la librería se resume a las siguientes líneas:

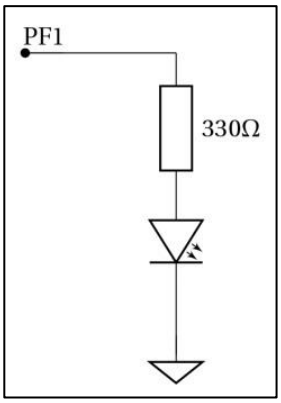
```
1 // Habilita el reloj en el puerto F.
2 SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOF );
3
4 // Configura Pin1 , Pin2 , Pin3 del puerto F como salidas .
5 GPIOPinTypeGPIOOutput ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 |
    GPIO_PIN_3 );
```

De forma similar a configurar un pin como salida digital con GPIOPinTypeGPIOOutput es posible configurarlo como entrada con GPIOPinTypeGPIOInput. Estas funciones reciben como parámetros la base

asociada a los registros del GPIO en un puerto y un valor que indica que pines reciben la configuración.

Una vez configurado el módulo de GPIO para el puerto F, considérese el circuito mostrado en la figura 87, donde un Led se encuentra conectado a un pin del puerto F, específicamente el pin 1.

Figura 85. **Led conectado al pin 1 del puerto F**



Fuente: elaboración propia, empleando Inkscape.

Si se desea que este led encienda basta con colocar un 1 lógico en el bit 1 del registro GPIODATA correspondiente al puerto F, ya que 1 lógico equivale a colocar un voltaje de 3.3V en el pin configurado como salida. Se utiliza la forma *friendly coding* con el fin de solamente modificar el registro que se desea. La sentencia en C que coloca un uno lógico en el bit 1 del puerto F es:

```
GPIO_PORTF_DATA_R |=0 x00000002;
```

Se debe aclarar que la enumeración de los bits comienza en 0, siendo el mismo índice para el número de pin, por tanto si se desea tener un 1 lógico en

el bit se debe colocar 0x02, si se desea *ten* el bit 2 se debe colocar 0x04, para el bit 3 el valor 0x08 y así sucesivamente corresponde una potencia de dos al número del bit que se desea con un uno lógico. Si se desean varios 1 lógicos, es posible hacerlos con una disyunción bit a bit con dichos valores. Por ejemplo, si se desea el 1 uno lógico en los bits 1, 2 y 3 es posible hacerlo con la siguiente sentencia en C:

```
GPIO_PORTF_DATA_R |=(0 x00000002 |0 x00000004 |0 x00000008 );
```

Es posible construir las máscaras para colocar ceros en lugar de unos, solamente realizando una conjunción y haciendo la operación bit a bit con 1 en los bits que desean no alterarse y 0 en el bit que desea alterarse. Por ejemplo, si se desea colocar un cero lógico en los bit 2 y 8 se tendría la siguiente sentencia en C:

```
GPIO_PORTF_DATA_R &=(0 xFFFFFFFD &0 xFFFFFFFB );
```

Es posible tener el caso donde haya intención de colocar valores que no sean solamente unos o ceros, para ello sería necesario leer el registro, limpiar los bits a modificar y luego sumar el valor que se desea. Por ejemplo, si se desea escribir un 1 lógico en los bits 3 y 1, pero un cero en el bit 2 se tendrían las siguientes instrucciones de C:

```
GPIO_PORTF_DATA_R &=(0 xFFFFFFF1 );
GPIO_PORTF_DATA_R |=(0 x0000000A );
```

En los casos anteriores, se realiza al menos un proceso de lectura del registro y luego uno de asignación, realizados con los operadores &= o |=. Puede que en C sea una línea pero al verlo en código de máquina el

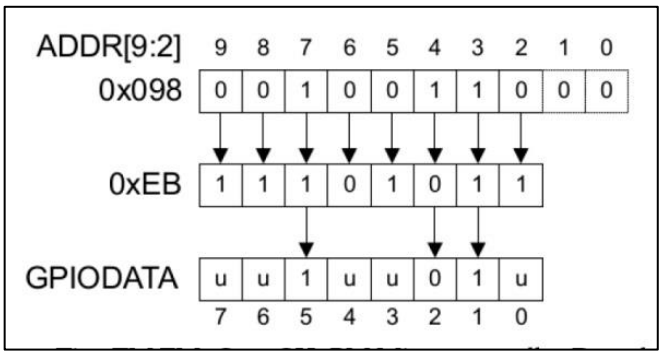
procesador necesita primero obtener el valor del registro y luego asignar el resultado, consumiendo varios ciclos de reloj.

El módulo de GPIO permite la modificación de bits individuales, usando los bits del 2 al 9 del bus de direcciones como una máscara. De esta forma se pueden modificar los bits deseados sin afectar otros pines y sin necesidad de recurrir a un proceso de lectura y escritura. Para lograr implementar esta característica, el registro GPIODATA abarca 256 posiciones en el mapa de memoria.

Durante una escritura, si el bit de dirección asociado con un bit de dato se encuentra con un 1 lógico se producirá una escritura en ese bit. Dicho de otra forma cada combinación posible de unos y ceros lógicos para 8 bits tiene una dirección de memoria asociada. La base del registro GPIODATA más el *offset* dado por la máscara de los bits que se desean modificar, corrida 2 bits hacia la izquierda, da como resultado esa dirección de memoria. Por ejemplo, si se desea modificar los bits 6 y 4, el *offset* para la base estaría dado por 0x140 el cuál es el resultado de correr dos bits hacia la izquierda la máscara que indica que bits a modificar, 0x50.

La hoja de datos para este proceso, como ejemplo, presenta la escritura del valor 0xEB a una dirección de memoria dada por la suma de la base del puerto y un *offset* de 0x98, cuyo resultado altera únicamente los bits 2, 3 y 5. La figura 88 muestra el proceso de escritura y el resultado.

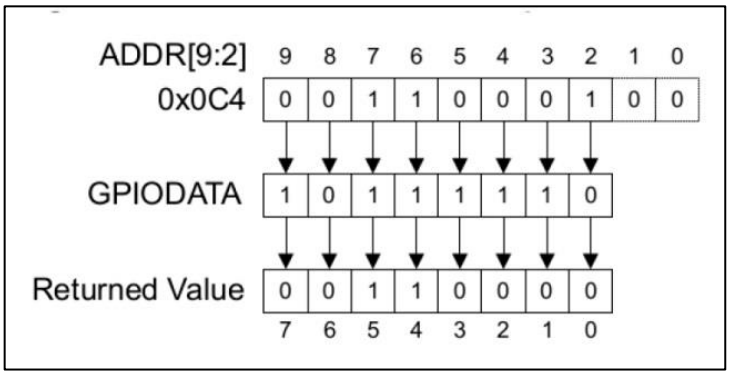
Figura 86. **Enmascaramiento por el bus de datos I**



Fuente: Fuente: *Tiva TM TM4C123GH6PM Microcontroller Datasheet*. http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: agosto de 2015.

En la figura 88 la u que se muestra significa sin cambio. Si se realiza una lectura, los bits que no desean ser leídos devolverán cero, la figura 89 muestra el resultado de leer utilizando esta técnica. El ejemplo tomado de la hoja de datos para la lectura muestra el resultado de leer una dirección con un *offset* de 0xC4 cuando el puerto tiene un valor de 0xBE.

Figura 87. **Enmascaramiento por el bus de datos II**



Fuente: Fuente: *Tiva TM TM4C123GH6PM Microcontroller Datasheet*. http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: agosto de 2015.

Utilizando la librería es posible escribir y leer valores en los puertos cuando están configurados como digitales, las funciones son: GPIOPinWrite y GPIOPinRead. La función GPIOPinWrite recibe tres parámetros: la dirección de la base, la máscara y el valor a escribir. Por ejemplo, escribir 0x05 en el puerto F solo alterando los bits 3, 2 y 1 se logra de la siguiente forma:

```
GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 , 0 x05 );
```

Dejando sin modificar el resto de los bits. La función GPIOPinRead recibe únicamente dos parámetros: la dirección de la base y la máscara. Esto se debe que con esta función se desea obtener el valor del puerto. Por ejemplo, para obtener el valor actual de los bits 0 y 4 se logra con:

```
GPIOPinRead ( GPIO_PORTF_BASE , GPIO_PIN_4 | GPIO_PIN_0 );
```

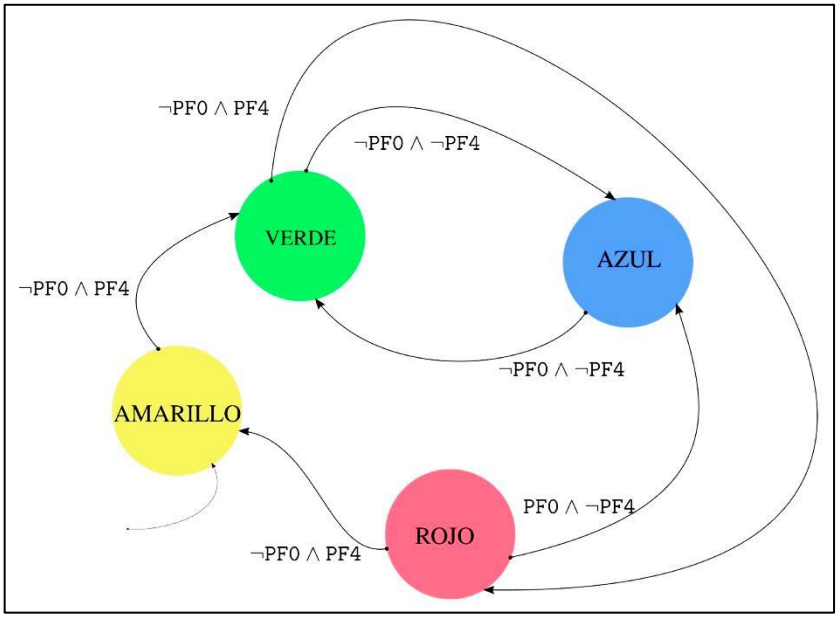
Dicha función devolverá un valor de 32 bits con los valores del puerto para los bits de la máscara, en este caso los bits 1 y 4, y en el resto de bits se obtendrá como resultado cero.

3.7.4. Implementación de una máquina de estados

En esta sección se muestra como ejemplo la implementación de una máquina de estados en el microcontrolador TM4C123GH6PM utilizando como lenguaje C y apoyado con la librería TivaWare.

Considérese la máquina de estados que se muestra en la figura 90. Esta máquina de estados es una de Moore, ya que las salidas dependen únicamente del estado en que se encuentra.

Figura 88. **Máquina de estados**



Fuente: elaboración propia, empleando Inkscape.

La máquina de estados de la figura 88 se puede representar en forma de tabla, conocida como tabla de estados, que contiene la misma información que el grafo y las funciones de actualización, pero su visualización puede ser más útil para el programador a la hora de implementarla en un microcontrolador.

Considerando que, al no existir transición para una entrada determinada, para esa entrada se considera una transición implícita hacia el mismo estado; por lo que la tabla para la máquina de estados de la figura 88 quedaría de la siguiente forma:

Tabla LXXVII. **Tabla de estados para la figura 88**

Índice	Estado	Salida	Siguiente Estado			
			¬PF4∧ ¬PF0≡0x00	¬PF4∧PF0≡0x01	PF4∧ ¬PF0≡0x02	¬PF4∧ PF0≡0x03
0	Amarillo	Amarillo	Amarillo	Amarillo	Verde	Amarillo
1	Rojo	Rojo	Rojo	Azul	Amarillo	Rojo
2	Verde	Verde	Azul	Verde	Rojo	Verde
3	Azul	Azul	Verde	Azul	Azul	Azul

Fuente: elaboración propia.

Como puede verse en la tabla, se ha asignado un índice a cada estado. Esto con el fin de poder implementar esta dinámica en un microcontrolador. También se puede observar que la tabla condensa el comportamiento de las transiciones entre estados dependiendo de la entrada que se tenga. Por ejemplo para el estado amarillo, si la entrada se compone de los bits PF4 y PF0, y en determinado momento es equivalente a 0x02 el estado siguiente será verde, si el estado es azul y la entrada es equivalente a 0x03 el siguiente estado será azul.

Para mostrar por completo la misma información del grafo y las funciones de actualización, se toma como convención que el estado con el índice 0 en la tabla sea el estado inicial de la máquina de estados.

Las salidas para cada estado tendrán asociado un valor discreto ya que es lo único que puede entender un microprocesador. Por ejemplo, para el estado azul el microcontrolador estará entregando una salida digital que sea capaz de generar el color azul. Siendo lo anterior análogo para el resto de los estados. La tarjeta de desarrollo TivaC cuenta con botones del tipo *push-button* conectados a los pines PF4 y PF0. Los botones se encuentran configurados con lógica

negada, es decir al tener presionado el botón se entrega al sistema un voltaje bajo (0V), y por tanto se interpreta la lectura del pin con un cero lógico.

La tarjeta también cuenta con un led RGB conectado a los pines PF1, PF2 y PF3. Estando el pin rojo del led conectado al pin PF1, el pin azul del led conectado al pin PF2 y el pin verde conectado al pin PF3. Por tanto, para implementar la máquina de estados se debe colocar como entrada los pines PF0 y PF4, y como salida los pines PF1, PF2 y PF3. Para generar el color rojo basta con cargar a la dirección GPIO_PORTF_BASE+0x38 el valor 0x2, para el color azul el valor 0x4, para el color verde el valor 0x8 y para el color amarillo el valor 0xA.

Para utilizar el microcontrolador sin necesidad de dedicar largas horas de trabajo configurando una serie de registros con el fin de hacerlo funcionar, hay que incluir las librerías necesarias para el control de periféricos y del sistema. Además es útil incluir una serie de definiciones para tipos de datos usando el estándar C99.

```
# include <stdio .h>
# include " inc / hw_memmap .h"
# include " inc / hw_types .h "
# include " driverlib / sysctl .h"
# include " driverlib / gpio .h "
# include " inc / tm4c123gh6pm .h"
```

Luego de agregar las librerías necesarias, utilizando la directiva #define se crean definiciones con el fin de representar el estado mediante un número, y para hacerlo de forma práctica, ese número será el índice mostrado en la tabla L.

```
# define Amarillo        0
# define Rojo            1
# define Verde           2
# define Azul            3
```

En C es posible crear almacenamientos compuestos, combinando elementos de diferentes tipos en una entidad, esto se conoce como estructura. Una estructura es un conjunto de variables que se referencian bajo el mismo nombre. Una estructura presenta una muy buena aproximación a un estado en una máquina de Moore.

La sintaxis de la declaración de una estructura en lenguaje C es:

```
struct nombre_estructura {
tipo nombre_variable ;
tipo nombre_variable ;
...
tipo nombre_variable ;
};
```

Para crear una estructura que aproxime un estado de la máquina de la figura 88 se propone el siguiente código:

```
struct Estado {
uint32_t Salida ;
uint32_t Tiempo ;
uint32_t Siguiente [4];
};
```

En la propuesta de estado, el elemento salida es una variable útil para albergar la salida un estado, cuyo valor se entrega en los pines de salida ya elegidos con anterioridad. El elemento siguiente es un vector utilizado para

representar los estados a los que puede dirigirse un estado al pasar una transición gobernada por una entrada, puede verse como la función de actualización para un estado de la máquina de estados. Cada elemento de este vector representa un estado para una entrada, es decir si un estado se encuentra en la posición cero indica que es el estado siguiente si la entrada es cero.

Se ha introducido una variante en las máquinas de estados con el elemento Tiempo, el cual permitirá al microcontrolador esperar un determinado tiempo antes de saltar de un estado a otro. Esto puede tener fines prácticos como un método antirebote o para marcar tiempos variables entre la transición de un estado a otro. Por ejemplo, si esta máquina de estados representara la dinámica discreta del control de una dispensadora de productos de diferentes tamaños, las salidas pueden indicar señales para sistemas mecánicos que despachan los productos. Puede que se deba sostener determinada salida por tiempos diferentes para diferentes productos.

Una vez creada la estructura Estado, se procede a crear un tipo de dato basado en ella, con el fin de declarar una variable que tenga la misma forma que la estructura. Para esto se utiliza *typedef* de la siguiente manera:

```
typedef const struct Estado EstadoTipo;
```

El modificador *const* permite que la variable creada bajo este tipo se aloje en la ROM. Sin él, el compilador pondrá la estructura en la RAM, permitiendo que sea cambiada dinámicamente. Una vez creado el tipo de dato es posible declarar una variable bajo el tipo EstadoTipo, la cual albergará todos los estados, formando así una aproximación para la máquina de estados.

Para hacerlo basta con utilizar el tipo EstadoTipo, ya creado anteriormente, de la misma forma que los tipos de datos nativos como *int* o *char*. Dado que una máquina de estados se compone de varios estados se debe declarar un vector para almacenar todas las estructuras que representan la aproximación de un estado. Quedando la declaración del vector que alberga la máquina de la siguiente forma:

```
EstadoTipo FSM [4]={
{0 x0A ,26666666 ,{ Amarillo , Amarillo ,Verde , Amarillo }},
{0 x02 ,26666666 ,{ Rojo ,Azul , Amarillo , Rojo }},
{0 x08 ,26666666 ,{ Azul ,Verde ,Rojo , Verde }},
{0 x04 ,26666666 ,{ Verde ,Azul ,Azul , Azul }}
};
```

La variable FSM, no es más que la información condensada de la tabla L. El orden de las estructuras que forman parte del vector de estructuras FSM debe ser el mismo mostrado en la tabla de estados. Por ejemplo, para el estado amarillo, posición cero de la variable FSM, los estados siguientes posibles están dados por el vector {amarillo,amarillo,verde,amarillo}, su salida por 0x0A y el valor del tiempo, en determinada escala, por 26666666. En esta máquina de estados, dado que es para fines demostrativos se considera que las transiciones entre estados se dan a tiempos iguales.

A continuación, se declaran variables que almacenarán los valores de las entradas y la variable que lleva control del estado actual de la máquina de estados:

```
uint32_t estado = Amarillo ;
uint32_t Entrada =0;
```

La variable estado, como su nombre lo indica, albergará al estado actual de la máquina de estados que se encuentra en implementación. Esta variable indicará la posición en la variable FSM para extraer la información del estado al cuál representa.

En la máquina de estados se presentan solamente cuatro estados, por lo que dos bits son suficientes para contarlos, siendo PF0 y PF4 capaces de generar combinaciones diferentes suficientes para abarcarlos a todos. La variable entrada es la encargada de indicar o enumerar la combinación dada por PF0 y PF4.

Como la máquina de estados es una de Moore el estado siguiente depende del estado actual y la entrada que se tiene; utilizando las dos variables anteriormente mencionadas se obtiene el estado siguiente:

FSM [estado]. Salida [Entrada];

El punto indica que se está llamando a un elemento de una estructura. Hasta ahora solo se han tratado aspectos relacionados con la máquina de estados, pero no se debe olvidar que además de asuntos relacionados con la dinámica se deben tratar asuntos relacionados con la tecnología a utilizar. Adentro del procedimiento principal *main*, se muestran algunos procedimientos tratados en las secciones anteriores de como configuración de periféricos y reloj del sistema.

```
void main ( void ){
SysCtlClockSet ( SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
| SYSCTL_OSC_MAIN );
SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOF );
```

```
GPIOPinTypeGPIOOutput ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3 );
GPIO_PORTF_LOCK_R =0 x4C4F434B ;
GPIO_PORTF_CR_R |=0 x1F ;
GPIOPinTypeGPIOInput ( GPIO_PORTF_BASE , GPIO_PIN_4 | GPIO_PIN_0 );
GPIOPadConfigSet ( GPIO_PORTF_BASE , GPIO_PIN_4 | GPIO_PIN_0 ,
GPIO_STRENGTH_2MA ,
GPIO_PIN_TYPE_STD_WPU );
```

Son instrucciones tratadas en secciones anteriores. Estas instrucciones permiten tener un reloj del sistema corriendo a 80Mhz, los pines PF0 y PF4 como entradas, con resistencias *pull-up*, y los pines PF1, PF2 y PF3 como salidas. Luego de configurar el dispositivo como se desea, se pasa a la dinámica de la máquina de estados.

```
// Salida depende del Estado Actual .
GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 , FSM
[ estado ]. Salida );
// Se espera un tiempo determinado por el estado .
SysCtlDelay ( FSM [ estado ]. Tiempo );
// Entrada en PF4 y PF0 . Manipulaci ón para tener solamente 4 posibles resultados .
Entrada =( GPIOPinRead ( GPIO_PORTF_BASE , GPIO_PIN_4 ) >>3);
Entrada |= GPIOPinRead ( GPIO_PORTF_BASE , GPIO_PIN_0 );
// Cambio a siguiente estado dependiendo de la entrada .
estado = FSM [ estado ]. Siguiente [ Entrada ];
```

Dado un estado, la salida de la máquina de estados de Moore depende únicamente de él. Luego se sostiene esta salida por un tiempo determinado, estos dependen de la dinámica, acá se considerarán de 1 segundo, para poder apreciar las transiciones. Después de la espera, se lee una entrada, en este caso una combinación de los pines PF4 y PF0, generando un número asociado a la combinación y representando el próximo estado.

Teniendo este número se salta al próximo estado, pasando a ser el estado actual. En este estado se regresa al punto donde se coloca la salida del estado actual, utilizando un ciclo *while*. A este proceso de preguntar constantemente las entradas se conoce como *polling*. Para el momento donde se espera el tiempo, la función encargada de hacer al microcontrolador perder ciclos de reloj es SysCtlDelay, y mide el parámetro que recibe es el número de tríos de ciclos de reloj que esperará el microcontrolador. Es decir, si el microcontrolador esta funcionando a 80Mhz y es necesario esperar 3 segundos se debe colocar en esta función 80000000.

SysCtlDelay (80000000); // Espera 3 segundos.

Para la parte donde se obtiene un valor para la variable Entrada, se realizan dos lecturas de pines ya que resulta lo más eficiente debido a la propiedad del microcontrolador de poder leer cualquier combinacón de pines configurados como entradas, donde los configurados como salidas devolverán siempre cero.

La primera lectura se realiza sobre el pin PF4 y se obtendrá en la lectura como valores solamente 0x00000010 o 0x00000000, se realiza un corrimiento de 3 bits hacia la derecha para cambiar los valores a 0x00000002 o 0x00000000, luego se realiza la lectura sobre el pin PF0 obteniendo de ella los valores 0x00000001 o 0x00000000 dependiendo y luego se realiza una disyunción bit a bit, con el fin de poder generar para la entrada los valores 0x00000000, 0x0000001, 0x00000002 o 0x00000003.

Si se hubiera realizado la lectura simplemente enmascarando los pines PF4 y PF0 se hubieran obtenido los valores: 0x00000010, 0x00000001, 0x00000011 o 0x00000010. Sin embargo, para movilizarse entre estados, el

elemento Salida de la estructura Estado hubiera tenido que tener como mínimo 17 posiciones para poder responder a la entrada 0x00000011, que es la mayor de las posibles, y rellenar las posiciones que no se pueden obtener con la entrada, debido al enmascaramiento con transiciones hacia el mismo estado.

Esta forma de implementar la máquina de estados solo utiliza movimientos en memoria. Tiene desventajas como la que se trató anteriormente, que en muchos caso habrá que hacer manipulación con los bits de las entradas para lograr satisfacer algunas disposiciones en el hardware de la tarjeta, los botones ya vienen soldados a la placa si se hubiera querido, por ejemplo utilizar los pines PF0 y PF1 come entradas para realizar solamente una lectura, se hubiera tenido que poner hardware externo a la tarjeta. Sin embargo, esta implementación ofrece hasta cierto mayor atomicidad en las instrucciones que la que ofrece una implementación con condicionales, aunque en algunos casos es muy difícil lograr una implementación práctica sin ellos. En el código 3.5 se puede apreciar el código completo de la implementación en el microcontrolador.

Tabla LXXVIII. **Código 3.5: implementación de una FSM de Moore en microcontrolador TM4C123GH6PM utilizando C como lenguaje de programación**

```
1 # include <stdint .h>
2 # include <stdbool .h>
3 # include " inc / tm4c123gh6pm .h"
4 # include " inc / hw_memmap .h"
5 # include " inc / hw_types .h"
6 # include " driverlib / sysctl .h"
7 # include " driverlib / interrupt .h"
8 # include " driverlib / gpio .h"
9 # include " driverlib / timer .h"
10
11
12 # define Amarillo 0
```

Continuación de la tabla LXXVIII.

```
13 # define Rojo 1
14 # define Verde 2
15
16 # define tAmarillo 160000000
17 # define tRojo 480000000
18 # define tVerde 320000000
19
20 # define LEDAmarillo 0 x0A
21 # define LEDRojo 0 x02
22 # define LEDVerde 0 x08
23
24 struct Estado {
25 uint32_t Salida ;
26 uint32_t Tiempo ;
27 uint32_t Siguiente ;
28 };
29
30 // Habilitar la creaci ón de estructuras de la forma de Estado
31 typedef const struct Estado EstadoTipo ;
32
33 // Creaci ón de una estructura para albergar una FSM
34 EstadoTipo FSM [3]={
35 { LEDRojo ,tRojo , Rojo },
36 { LEDVerde , tVerde , Verde },
37 { LEDAmarillo , tAmarillo , Amarillo }
38 };
39
40 uint32_t estado = Rojo ;
41
42
43 int main ( void )
44 {
45
46      SysCtlClockSet      (      SYSCTL_SYSDIV_2_5      |      SYSCTL_USE_PLL      |
SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN )
;
47
48 SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOF );
49  GPIOPinTypeGPIOOutput  (  GPIO_PORTF_BASE  ,  GPIO_PIN_1  |  GPIO_PIN_2  |
GPIO_PIN_3 );
50
51 SysCtlPeripheralEnable ( SYSCTL_PERIPH_TIMER0 );
52 TimerConfigure ( TIMER0_BASE , TIMER_CFG_PERIODIC );
53
              54 TimerLoadSet ( TIMER0_BASE , TIMER_A , FSM [ estado ]. Tiempo -1);
55
56 IntEnable ( INT_TIMER0A );
```

Continuación de la tabla LXXVIII.

```
57 TimerIntEnable ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
58 IntMasterEnable ();
59
60 TimerEnable ( TIMER0_BASE , TIMER_A );
61
62 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 ,
FSM [ estado ]. Salida
);
63 estado = FSM [ estado ]. Siguiente ;
64
65 }
66
67 void Transicion ( void )
68 {
69 TimerIntClear ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
70 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 ,
FSM [ estado ]. Salida
);
71 TimerLoadSet ( TIMER0_BASE , TIMER_A , FSM [ estado ]. Tiempo -1);
72 estado = FSM [ estado ]. Siguiente ;
73 }
```

Fuente: elaboración propia, empleando C/tilaware.

3.7.5. Uso de interrupciones. Temporizador utilizando Tivaware

Una interrupción es el cambio automático del flujo del software en respuesta a un evento. Estos eventos pueden ser generados por hardware, donde circuitos externos al microprocesador realizan un cambio de voltaje en un puerto de solicitud de interrupciones; pueden ser generados por software donde el programa que se encuentra ejecutándose dispara una interrupción ejecutando alguna instrucción especial o escribiendo a algún registro mapeado en la memoria. Las interrupciones pueden ser utilizadas en eventos poco frecuentes pero críticos como una falla en el poder, un mal uso del bus, errores de memoria y errores de máquina. Cuando una interrupción es disparada cuando el hardware interno detecta una falla es llamada excepción, como una

violación de acceso. El mecanismo que permite disparar una interrupción es llamado *trigger*.

En los primeros dos tipos de interrupción, el programa regresa al lugar donde fue interrumpido una vez terminada la ISR. No sucede lo mismo para el caso de la excepción, en lugar de ello el puntero del programa se sitúa en un lugar fijo, en el cual termina el programa fallido.

Ante el *trigger* de una interrupción, el hardware debe decidir como responder. Si las interrupciones se encuentran deshabilitadas el procesador no responderá a los *triggers*. El mecanismo para habilitar o deshabilitar depende de cada microprocesador, algunos ofrecen la flexibilidad de permitir algunas y no otras o hasta asignar prioridades de ejecución a unas ISR sobre otras.

El uso de interrupciones periódicas será útil para el uso de relojes en tiempo real, adquisición de datos y sistemas de control. Un controlador de interrupciones es la lógica que sigue el procesador para manipular y manejar las interrupciones; la mayoría soporta cierto número de ellas y niveles de prioridad. El vector de interrupciones, el vector de interrucciones, en inglés *interrupt vector table*, posee las direcciones de memoria a utilizar por la ISR, para ejecutar. Cuando el hardware solicita una interrupción en un pin de solicitud, cambiando el voltaje en un pin de solicitud, puede realizarse por nivel o por flanco. Para las interrupciones disparadas por nivel, lo más común es que el hardware solicitando interrupción sostenga el mismo voltaje hasta que reciba una señal por parte del procesador indicando que la interrupción será atendida por él. Para las interrupciones por flanco, el hardware realiza un cambio en el nivel de voltaje por un corto tiempo, el cual permite una reacción inmediata en el controlador de interrupciones.

3.7.6. Atomicidad

Una ISR puede ser invocada entre dos instrucciones del programa principal (o entre dos instrucciones de una ISR de una interrupción con menor prioridad). Uno de los mayores retos que enfrenta un diseñador de sistemas embebidos es razonar acerca de los posibles flujos que puede tomar el programa debido a interrupciones y por tanto predecir el comportamiento de la dinámica discreta. Por ejemplo, el valor de una variable puede cambiar durante la ejecución de la ISR, y al regresar de la rutina por el cambio en dicho valor cambiar el flujo del programa.

Un diseñador de sistemas embebidos debe lidiar con este problema e identificar que secciones del programa no se ven afectadas por la ejecución de una rutina. Si un bloque de código o sección del programa no altera su flujo de ejecución, al término de una ISR puede ser considerada como atómica.

El término atómico proviene del griego "*atomum",* que significa indivisible. Desafortunadamente resulta difícil identificar que secciones pueden ser consideradas atómicas, incluso si se programa en *assembly*, a pesar que es más fácil identificarlas, no es posible considerar cada instrucción de *assembly* atómica ya que varias ISAs incluyen instrucciones que realizan diversos procesos.

En un programa de C resulta más difícil reconocer las operaciones que son atómicas. Por ejemplo, la instrucción:

```
Count =2000;
```

En un microcontrolador de 8 bits, a esta instrucción le toma varias instrucciones de *assembly* para ejecutarse, y puede que una interrupción ocurra en medio de las instrucciones de *assembly* que se encargan de llevar a cabo la instrucción de C. Si la ISR también escribe un valor diferente en la variable Count, como resultado la variable se encontrará compuesta de 8 bits configurados en la ISR y el resto de los bits configurados por las instrucciones correspondientes al programa principal. Teniendo así un *bug* sumamente difícil de identificar. Peor aún, las ocurrencias de este *bug* son raras y pueden no manifestarse en las pruebas.

Dependiendo de lo minucioso que se sea para asumir que instrucciones o procesos son atómicos, se tendrá un mejor modelo de la dinámica pero con el riesgo que puede llegar a ser complicado.

3.7.7. Modelando interrupciones

El comportamiento de las interrupciones puede ser un tanto difícil de entender y puede generar fallas catastróficas por comportamientos inadecuados. La lógica de los controladores de interrupciones se encuentra de forma muy imprecisa en la documentación de los microprocesadores dejando muchos posibles comportamientos sin especificar. Una forma de hacer la lógica de las interrupciones más precisa, es con una máquina de estados finitos. Considérese el siguiente código en C, el cual es una implementación una dinámica discreta, donde se ven involucradas interrupciones.

Tabla LXXIX. **Código 3.6: interrupción en código**

```
1 volatile uint timerCount = 0;
2 void ISR ( void ) {
3 ... có digo que deshabilita interrupciones
4 // ESTADO D
5 if( timerCount != 0) {
6 // ESTADO E
7 timerCount --;
8 }
9 ... có digo que habilita interrupciones
10 }
11 int main ( void ) {
12 // Configuraci ón de la
13 SysTickIntRegister (& ISR);
14 timerCount = 2000;
15 // ESTADO A
16 while ( timerCount != 0) {
17 // ESTADO B
18 ... có digo ejecutandose durante 2 segundos
19 }
20 // ESTADO C
21 }
22
```

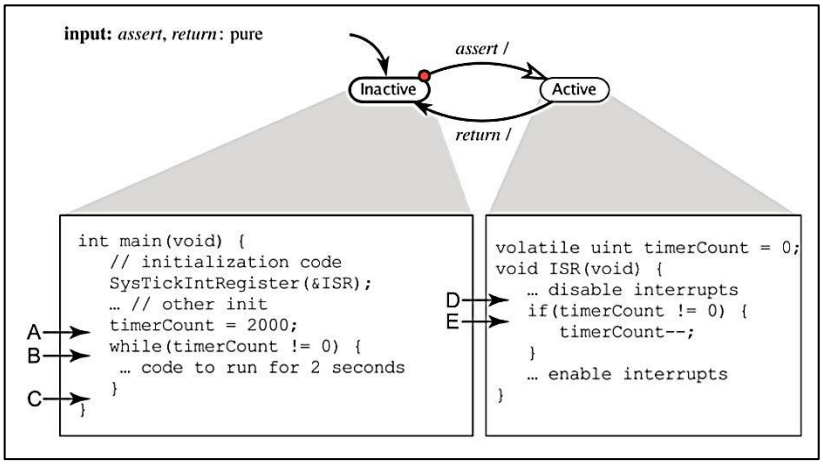
Fuente: elaboración propia, empleando C/tilaware.

En el código 3.6 pueden verse comentarios con etiquetas de la forma //ESTADO * entre algunas sentencias de código escrito en C, por tanto se asume que los bloques que se encuentran entre ellas son atómicos. Supóngase que la interrupción puede suceder en cualquier momento o parte del programa principal, rutina main, por lo que pasará a ejecutarse la rutina ISR, dejando inactiva la rutina main por un momento hasta culminar la rutina ISR.

En la figura 89 se puede ver el flujo del programa, donde se consideran *assert* y *return* como entradas, ambas señales puras, utilizadas como un modelo del controlador de interrupciones indicando los cambios efectuados en el flujo de la ejecución del programa. Al momento de tener assert como

verdadero se tiene un salto a la ejecución de la ISR. Al finalizar la ISR, la entrada *return* evaluará verdadero indicando el regreso al programa principal.

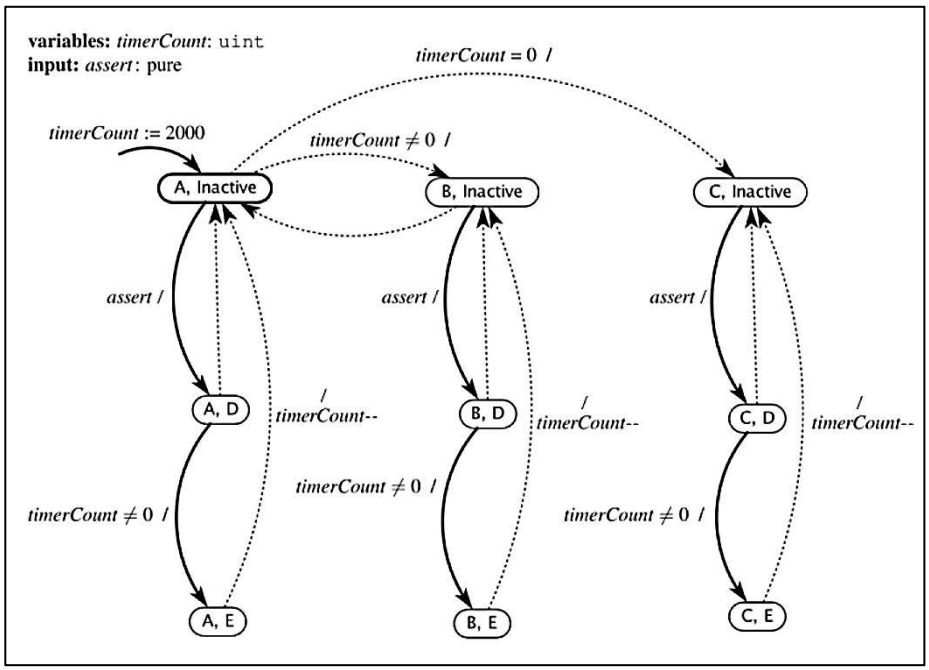
Figura 89. **Modelo del flujo del programa**



Fuente: LEE, E. A. y SESHIA, S. A. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. http://leeseshia.org/releases/LeeSeshia_DigitalV1_08.pdf. Consulta: agosto de 2015.

Se ha asumido que la interrupción puede suceder entre cualquier par de instrucciones atómicas, por lo que es posible modelar los estados como saltos desde una instrucción atómica a otra. La figura 90 muestra el modelo de la dinámica discreta del programa 3.6.

Figura 90. **Composición de máquinas de estados**



Fuente: LEE, E. A. y SESHIA, S. A. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. http://leeseshia.org/releases/LeeSeshia_DigitalV1_08.pdf. Consulta: agosto de 2015.

Los estados posibles para la rutina *main* se encuentran definidos en el conjunto $M = \{A, B, C\}$ y los estados posibles para la rutina ISR, se encuentran definidos por el conjunto $I = \{D, E\}$. Pueden suceder solamente dos casos:

- Que la interrupción suceda en algún punto de la ejecución del programa principal, se ejecute la rutina ISR y luego retorne al programa principal. En dicho caso se tendrán los estados $M \times I$:

$$M \times I = \{(A, D,), (A, E,), (B, D,), (B, E,), (C, D,), (E, C,)\}$$

[Ec. 3.23]

Cada pareja-estado del producto cartesiano $M \times I$ representa el origen y el destino de la interrupción, y el hecho de enumerarlos de esta forma se debe que pueden generarse diferentes comportamientos dependiendo del punto donde ha sucedido la interrupción y por tanto un estado diferente en el comportamiento del programa.

- Que la interrupción nunca suceda, ejecutándose el programa principal de forma lineal. Por lo que en este caso se tendrán el conjunto *M* de estados.

Los dos casos anteriores definen el comportamiento de la dinámica discreta, ambos deben considerarse, por tanto el conjunto mínimo de estados a considerar para el comportamiento del programa esta dado por:

$$S = (M \times F) \cup M \qquad \text{[Ec. 3.24]}$$

El objetivo es determinar todas las posibilidades en el comportamiento del sistema. Puede verse en la figura la clase de entradas que generarían una transición, por ejemplo, del estado A al estado (A, D) o del estado A al estado B. No tendría sentido considerar una transición por ejemplo del estado A a un estado (B, D) o (C, D), ya que la naturaleza de las parejas-estado es modelar el comportamiento del programa dado el punto de ocurrencia de la interrupción.

Este modelo resulta útil para comprender el comportamiento del programa, por ejemplo, si se deseará determinar si el programa es capaz de alcanzar la sección con la etiqueta //ESTADO C, del modelo puede verse que existe la posibilidad que esta parte nunca se alcance, por ejemplo, fuera el caso que la interrupción suceda con una frecuencia más alta que la de procesamiento puede atrapar el programa en un bucle.

Puede verse como el modelo es un tanto complicado para un programa sencillo. Puede imaginarse el lector lo complicado que puede resultar analizar sistemas con múltiples interrupciones y aún con niveles de prioridades e instrucciones difíciles de determinar atómicas. Esto hace este tipo de sistemas caóticos.

3.7.7.1. Uso de interrupciones de temporizador en microcontrolador TM4C123GH6PM

Se hará referencia al término excepción e interrupción indistintamente en esta sección, debido a que el procesador las maneja de la misma forma. Las interrupciones y ciertas rutinas especiales en este microcontrolador son manejadas por el NVIC, siglas en inglés de *Nested Vector Interrupt Controler*, el cual junto con el procesador da prioridades y maneja las interrupciones. El vector de interrupciones soporta:

- 78 interrupciones.
- Un nivel de prioridad programable de cero a siete para cada interrupción. Un nivel más alto corresponde a una prioridad más baja, por tanto el cero es la prioridad más alta.
- Baja latencia en el manejo de interrupciones y excepciones.
- Manejo dinámico de prioridades en las interrupciones.
- Agrupamiento de los valores de las prioridades.
- *Tail-chaining*, el cual permite transiciones eficientes entre interrupciones.
- Una interrupción externa no enmascarable (NMI).

Al hablar de interrupciones y excepciones en este microcontrolador es necesario mencionar los siguientes términos:

- Adelanto: cuando el procesador está ejecutando algún control de excepción, otra excepción se le puede adelantar si esto ocurre una con una prioridad más alta que la excepción que fue adelantada. Cuando una interrupción se le adelanta a otra se conocen como interrupciones enlazadas.

- Retorno: sucede cuando un control de excepción se encuentra completo, y no hay ninguna otra pendiente con suficiente prioridad para ser servida. Además la excepción que se encuentra completa no está llevando a cabo ninguna excepción *late-arriving*.

- *Tail-chaining:* este mecanismo acelera el servicio de excepciones. Al completar un servicio de excepción, si hay una excepción pendiente ya no se realiza el almacenamiento en pila y se pasa directo al servicio de la nueva excepción.

- *Late-arriving:* este mecanismo acelera el adelanto en una excepción. Si una excepción ocurre durante el momento que el procesador se encuentra guardando su estado en la pila, de una excepción previa; el procesador cambia y va a buscar al vector el servicio de la prioridad más alta. El almacenamiento en la pila no se ve afectado por la llegada de la nueva interrupción. El procesador puede aceptar el mecanismo de *Late-arriving* hasta que la primera instrucción de la excepción original se encuentra ejecutándose. Al retorno del control de excepción que ejecutó el *late-arriving*, se aplican las reglas normales de *tail-chaining*.

Una entrada de excepción ocurre cuando hay una excepción pendiente con suficiente prioridad y el procesador se encuentra en modo de ejecución. O bien la nueva excepción es de una prioridad más alta que la excepción que se

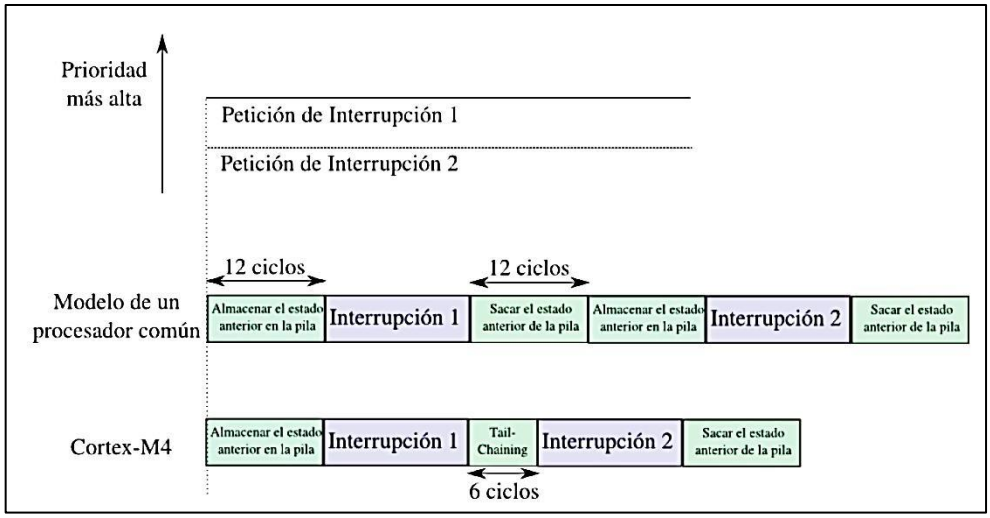
encuentra en proceso de ejecución. En este caso, la nueva excepción se le adelanta a la excepción original. Suficiente prioridad significa que la excepción tiene una mayor prioridad que los límites configurados por los registros para enmascaramiento. Una excepción con menor prioridad que estos límites se encuentra en estado pendiente pero jamás es atendida por el procesador.

A menos que el procesador se encuentre ejecutando los mecanismos de *late-arriving* o *tail-chaining*, cuando toma una excepción, el procesador empuja la información a la pila. Esta operación es referida como *stacking* y a la estructura de ocho palabras de datos se le llama como *stack frame*. Si se utilizan rutinas de punto flotante, el Cortex-M4F automáticamente almacena el estado del módulo de punto flotante.

En la mayoría de procesadores, el proceso para la ejecución de excepciones es bastante simple; solamente almacenan en la pila el estado actual luego ejecutan la rutina de interrupción y al estar completa, saca el estado de la pila y regresa a la ejecución de la rutina previa a la excepción.

Cuando la ejecución de una excepción termina, el *NVIC* busca una segunda excepción pendiente y ejecuta su rutina correspondiente. En la mayoría de los procesadores se ejecuta siempre el mismo proceso: almacenar en la pila, ejecutar rutina de excepción y volver a sacar el estado para regresar, para luego ejecutar la segunda excepción. Esto sería un desperdicio ya que se almacena y se saca la misma información por cada excepción que se ejecuta sucesivamente. En esta situación los Cortex-M4 manejan las excepciones con el mecanismo *tail-chaining*. Puede verse gráficamente este proceso en la figura 91.

Figura 91. **Comparación entre procesador común y Cortex-M4. *Tail-chaining***



Fuente: elaboración propia, empleando Inkscape.

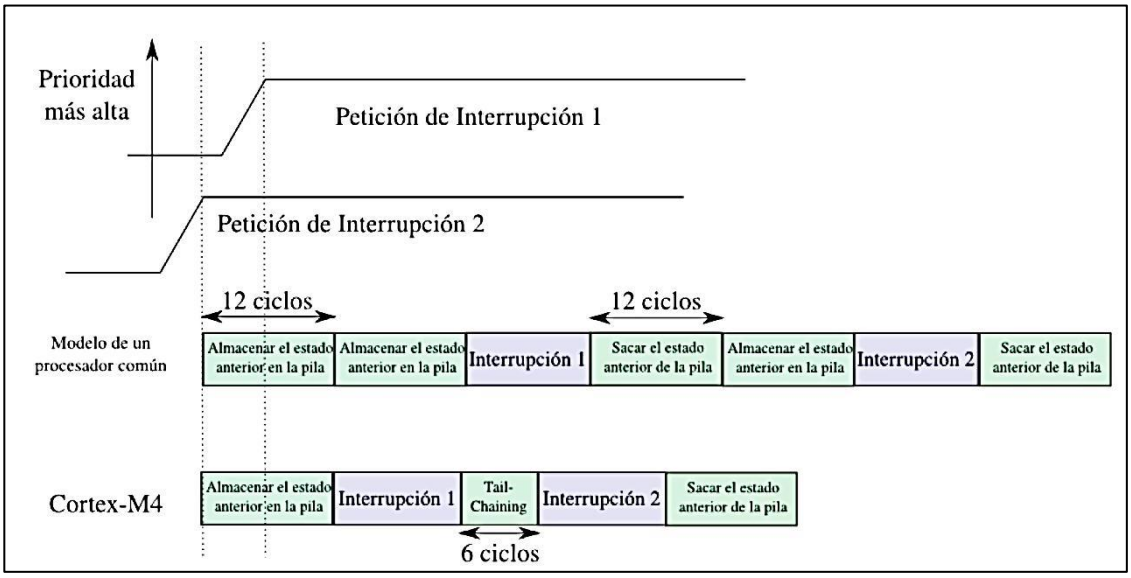
En seguida después del almacenamiento, el puntero de pila señala posición inferior del *stack frame*. El *stack frame* incluye la dirección de regreso, la cual es la siguiente instrucción en el programa interrumpido. El valor del registro PC (Program Counter) es restaurado al regreso de la excepción y el programa interrumpido continua su ejecución.

Al mismo tiempo que se ejecuta la operación de *stacking* el procesador busca y lee el inicio de la rutina propia de la excepción. Cuando el proceso de *stacking* se encuentra completo el procesador inicia la rutina con la dirección que tomó el vector. Paralelamente, el procesador escribe un valor EXEC_RETURN en el registro LR, indicando el puntero de pila que le corresponde al *stack frame* y el modo de operación en que se encontraba el procesador antes de la entrada a la excepción. Si no hay ninguna excepción con una prioridad más alta durante la entrada de la excepción, el procesador

inicia la ejecución de la rutina correspondiente y cambia el bit de estado a activo.

Si hay una excepción con una prioridad más alta durante la entrada de la excepción, proceso de *late arriving* que se muestra en la figura 92, el procesador inicia esta nueva rutina y no cambia el estado pendiente de la excepción anterior.

Figura 92. **Comparación entre procesador común y Cortex-M4. *Late-arriving***



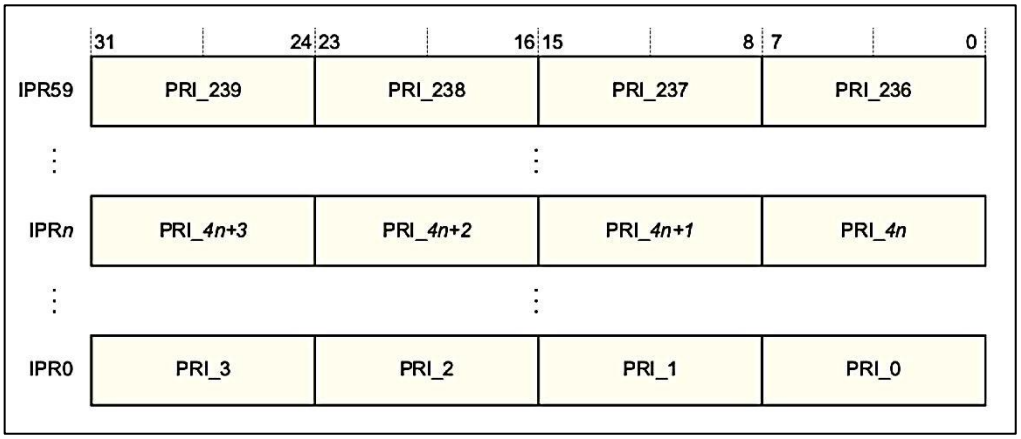
Fuente: elaboración propia, empleando Inkscape.

Al entrar una excepción el procesador automáticamente almacena su estado y lo recupera al salir de ella, sin alguna instrucción de cabecera y esto le permite un rápido manejo de las excepciones. Los registros NVIC_ISER0 al NVIC_ISER7 habilitan las interrupciones y muestran cuales se encuentran habilitadas. Siendo dichos registros de lectura y escritura, en la hoja de datos se

encuentra dicha información con las siglas RW. Poner un bit en alto en este registro habilitará una interrupción; sin embargo colocarlo en bajo no tendrá ningún efecto.

Para este efecto se utilizan los registros NVIC_ICER0 al NVIC_ICER7 los cuales deshabilitan las interrupciones, pero muestran las habilitadas. Los registros NVIC_ISPR0 al NVIC_ISPR7, fuerzan a las interrupciones a estar en estado pendiente y muestra cuáles se encuentran en este estado. De igual forma poner un bit en alto en este registro pondrá una interrupción en estado pendiente, sin embargo colocarlo en bajo no tendrá ningún efecto. Para este efecto se utilizan los registros NVIC_ICPR0 al NCVIC_ICPR7. Los registros NVIC_IPR0 al NVIC_IPR59 proveen un campo de ocho bits por cada interrupción, tal como se muestra en la figura 93.

Figura 93. **Registros de prioridades**



Fuente: *Hoja de datos del Cortex-M4//*. Consulta: agosto de 2015.

Cada campo de prioridad puede tener un valor de 0 a 255. Mientras más bajo el valor del registro más grande la prioridad de la interrupción correspondiente.

Cada excepción tiene un vector de 32 bits asociado, que apunta a la dirección de memoria donde se encuentra la ISR que maneja la excepción. Estos vectores se encuentran almacenados en la ROM, en la tabla de vectores. Para una interrupción asíncrona diferente que *reset*, el procesador puede ejecutar otra instrucción entre el momento que la excepción es disparada y el procesador entra al control de la excepción. La tabla de vectores contiene el valor inicial del puntero de pila y la dirección de inicio, también llamadas vectores de excepción, como todos los vectores que manejan una excepción. La tabla de vectores se construye usando como referencia la tabla LI.

Tabla LXXX. **Estructura de la tabla de vectores**

Tipo de Excepción	Número de Vector	Prioridad	Dirección de offset del vector	Disparo
-	0	-	0x0000.0000	Al reset el principio de la pila se carga desde la primera entrada de la tabla de vectores.
Reset	1	-3 (La más alta prioridad)	0x0000.0004	Asíncrono.
Interrupción no enmascarable (NMI)	2	-2	0x0000.0008	Asíncrono.
Hard Fault	3	-1	0x0000.000C	-
Manejo de Memoria	4	programable	0x0000.0010	Síncrono.
Bus Fault	5	programable	0x0000.0014	Síncrono cuando es preciso y asíncrono cuando es impreciso.
Usage Fault	6	programable	0x0000.0018	Síncrono.
-	7-10	-	-	Reservado.
SVCall	11	programable	0x0000.002C	Síncrono.
Monitor para Debug	12	programable	0x0000.0030	Síncrono.
-	13	-	-	Reservado.
PendSV	14	programable	0x0000.0038	Asíncrono.
SysTick	15	programable	0x0000.003C	Asíncrono.
Interrupciones de periféricos	16 en adelante	programable	0x0000.0040	Asíncrono.

Fuente: elaboración propia.

Las tablas LXXX y LXXXI muestran la descripción, la posición del vector en la *NVIC Table* y la dirección de memoria de los vectores que corresponden a las interrupciones generadas por los periféricos.

Tabla LXXXI. **Interrupciones de los periféricos**

Número de Vector	Número de Interrupción	Dirección de Vector	Descripción
0-15	-	0x0000.0000-0x0000.003C	Excepciones del Procesador
16	0	0x00000040	GPIO Puerto A
17	1	0x00000044	GPIO Puerto B
18	2	0x00000048	GPIO Puerto C
19	3	0x0000004C	GPIO Puerto D
20	4	0x00000050	GPIO Puerto E
21	5	0x00000054	UART0
22	6	0x00000058	UART1
23	7	0x0000005C	SSI0
24	8	0x00000060	I^2C0
25	9	0x00000064	Falta PWM0
26	10	0x00000068	Generador 0 PWM0
27	11	0x0000006C	Generador 1 PWM0
28	12	0x00000070	Generador 2 PWM0
29	13	0x00000074	QEI0
30	14	0x00000078	Secuenciador 0 ADC0
31	15	0x0000007C	Secuenciador 1 ADC0
32	16	0x00000080	Secuenciador 2 ADC0
33	17	0x00000084	Secuenciador 3 ADC0
34	18	0x00000088	Watchdog Timers 0 y 1
35	19	0x0000008C	Temporizador 16/32-Bits 0A
36	20	0x00000090	Temporizador 16/32-Bits 0B
37	21	0x00000094	Temporizador 16/32-Bits 1A
38	22	0x00000098	Temporizador 16/32-Bits 1B
39	23	0x0000009C	Temporizador 16/32-Bits 2A
40	24	0x000000A0	Temporizador 16/32-Bits 2B
41	25	0x000000A4	Comparador Analógico 0
42	26	0x000000A8	Comparador Analógico 1
43	27	0x000000AC	Reservado
44	28	0x000000B0	Sistema de Control
45	29	0x000000B4	Control de Flash y EEPROM
46	30	0x000000B8	GPIO Puerto F
47-48	31-32	0x000000BC-0x000000C0	Reservado
49	33	0x000000C4	UART2
50	34	0x000000C8	SSI1
51	35	0x000000CC	Temporizador 16/32-Bits 3A
52	36	0x000000D0	Temporizador 16/32-Bits 3B
53	37	0x000000D4	I^2C1
54	38	0x000000D8	QEI1
55	39	0x000000DC	CAN0
56	40	0x000000E0	CAN1
57-58	41-42	0x000000E4-0x000000E8	Reservado
59	43	0x000000EC	Módulo de Hibernación
60	44	0x000000F0	USB
61	45	0x000000F4	Generador PWM 3
62	46	0x000000F8	μDMA Software
63	47	0x000000FC	μDMA Error
64	48	0x00000100	Secuenciador 0 ADC1
65	49	0x00000104	Secuenciador 1 ADC1
66	50	0x00000108	Secuenciador 2 ADC1
67	51	0x0000010C	Secuenciador 3 ADC1
68-72	52-56	0x00000110-0x00000120	Reservado
73	57	0x00000124	SSI2
74	58	0x00000128	SSI3

Fuente: elaboración propia.

Tabla LXXXII. **Interrupciones de los periféricos, continuación**

Número de Vector	Número de Interrupción	Dirección de Vector	Descripción
75	59	0x0000012C	UART3
76	60	0x00000130	UART4
77	61	0x00000134	UART5
78	62	0x00000138	UART6
79	63	0x0000013C	UART7
80-83	64-67	0x00000140-0x0000014C	Reservado
84	68	0x00000150	I^2C2
85	69	0x00000154	I^2C3
86	70	0x00000158	Temporizador 16/32-Bits 4A
87	71	0x0000015C	Temporizador 16/32-Bits 4B
88-107	72-91	0x00000160-0x000001AC	Reservado
108	92	0x000001B0	Temporizador 16/32-Bits 5A
109	93	0x000001B4	Temporizador 16/32-Bits 5B
110	94	0x000001B8	Temporizador 32/64 Bits 0A
111	95	0x000001BC	Temporizador 32/64 Bits 0B
112	96	0x000001C0	Temporizador 32/64 Bits 1A
113	97	0x000001C4	Temporizador 32/64 Bits 1B
114	98	0x000001C8	Temporizador 32/64 Bits 2A
115	99	0x000001CC	Temporizador 32/64 Bits 2B
116	100	0x000001D0	Temporizador 32/64 Bits 3A
117	101	0x000001D4	Temporizador 32/64 Bits 3B
118	102	0x000001D8	Temporizador 32/64 Bits 4A
119	103	0x000001DC	Temporizador 32/64 Bits 4B
120	104	0x000001E0	Temporizador 32/64 Bits 5A
121	105	0x000001E4	Temporizador 32/64 Bits 5B
122	106	0x000001E8	Excepción del Sistema
123-149	107-133	0x000001EC-0x00000254	Reservado
150	134	0x00000258	Generador 0 PWM1
151	135	0x0000025C	Generador 1 PWM1
152	136	0x00000260	Generador 2 PWM1
153	137	0x00000264	Generador 3 PWM1
154	138	0x00000268	Falta PWM1

Fuente: elaboración propia.

Si el software no configura la prioridad de las interrupciones, todas las prioridades programables tienen un valor de cero. Si hay muchas interrupciones pendientes, la interrupción con la prioridad más alta se ejecuta primero. Si las interrupciones tienen la misma prioridad se ejecuta aquella con el número más pequeño. Si hay una rutina de interrupción que se está ejecutando y ocurre una

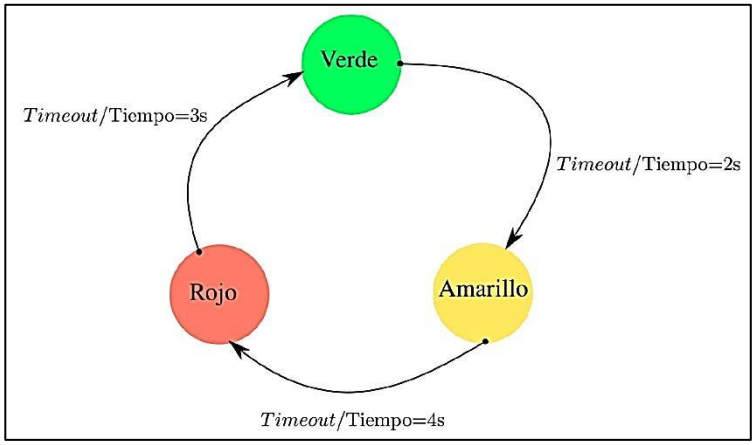
interrupción en ese momento, no se detiene la rutina si la interrupción que ocurrió durante, posee de la misma prioridad, solo se configura como pendiente.

Por suerte, todo este proceso de configuración puede ser resumido utilizando un compilador de C, la librería TivaWare, un archivo *STARTUP* y un archivo enlazador; siendo los que realizan la ardua tarea de configurar la *NVIC Table*. El archivo de STARTUP es un código escrito en C que da las configuraciones a la NVIC table. Algunos IDE como CCS al crear el proyecto generan su propio código STARTUP adaptado al dispositivo que se desea programar.

En este archivo se pueden encontrar qué rutinas se ejecutarán dependiendo del periférico que levante la interrupción. La posición del vector de rutinas que se puede encontrar en este archivo esta asociado a la posición de las tablas LXXX y LXXXI, por lo que si se declarará una ISR para un determinado periférico debe simplemente cambiarse el nombre que aparece asociado a la interrupción, sin alterar el orden de la tabla ya que podría causar algún conflicto. Se debe declarar la rutina en algún lugar del archivo que sea anterior a la declaración del vector de interrupciones, con el fin que el compilador conozca la rutina. Se debe utilizar el especificador extern si se declara fuera de este archivo, por ejemplo, en el archivo del programa principal.

Una vez que se ha tratado como el microcontrolador maneja las interrupciones, se desea implementar con fines demostrativos una dinámica discreta, representada como una máquina de estados por la figura 94.

Figura 94. **Máquina de estados de un semáforo simple**



Fuente: elaboración propia, empleando Inkscape.

La dinámica mencionada representa un semáforo, que debe cambiar de colores en los tiempos que se muestran en la figura. El evento *Timeout* que se muestra en la figura habilita la transición de un estado a otro. Este evento será marcado por un temporizador de propósito general, que es un periférico que ofrece el microcontrolador tm4c123gh60m y se configurará utilizando la librería TivaWare. La idea detrás de esto es utilizar el temporizador como una fuente de interrupciones para el microprocesador.

Se agregan al programa principal las librerías a utilizar:

```
# include <stdint .h>
# include <stdbool .h>
# include " inc / tm4c123gh6pm .h"
# include " inc / hw_memmap .h"
# include " inc / hw_types .h"
# include " driverlib / sysctl .h"
# include " driverlib / interrupt .h"
```

```
# include " driverlib / gpio .h"
# include " driverlib / timer .h"
```

Como se puede ver, las librerías a utilizar serán las mismas, con la diferencia que para el manejo de interrupciones y temporizadores es necesario agregar a las encargadas de su control y configuración. Estas librerías son timer.h y interrupt.h, para temporizadores e interrupciones respectivamente.

Para comodidad del programador se procede a nombrar algunos valores importantes en la implementación de la máquina de estados, como índice de los estados, tiempos y salidas.

```
# define Amarillo          0
# define Rojo              1
# define Verde             2
# define tAmarillo         160000000
# define tRojo             480000000
# define tVerde            320000000
# define LEDAmarillo       0 x0A
# define LEDRojo           0 x02
# define LEDVerde          0 x08
```

Se crean las estructuras y variables que almacenarán la información de la máquina de estados:

```
struct Estado {
uint32_t Salida ;
uint32_t Tiempo ;
uint32_t Siguiente ;
};
// Habilitar la creaci ón de estructuras de la forma de Estado
typedef const struct Estado EstadoTipo ;
```

```
// Creaci ón de una estructura para albergar una FSM
EstadoTipo FSM [3]={
{ LEDRojo ,tRojo , Rojo },
{ LEDVerde , tVerde , Verde },
{ LEDAmarillo , tAmarillo , Amarillo }
};
// Variable conteniendo el estado actual
uint32_t estado = Rojo ;
```

En la rutina principal *main* se escribirá la configuración inicial de los periféricos. Para el reloj principal del sistema se utilizará la configuración utilizada con anterioridad, una frecuencia de 80Mhz, un cristal de 16 Mhz y uso del PLL.

```
SysCtlClockSet ( SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN );
```

Se habilita el reloj al módulo de GPIO del puerto F y se configuran los pines PF1, PF2 y PF3 como salidas digitales.

```
SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOF );
GPIOPinTypeGPIOOutput ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 );
```

El microcontrolador tm4c123gh6pm posee seis módulos temporizadores de 16/32 bits capaces de funcionar como *one shot* o *periodic*. Cada temporizador puede ser visto como un único temporizador de 32 bits o bien como dos temporizadores independientes de 16 bits, a los que se hace referencia como *TimerA* y *TimerB*. La arquitectura de un módulo de los temporizadores escapa del alcance del presente trabajo y se recomiendo acudir a la hoja de datos.

Para los propósitos de la implementación, basta con mencionar que el temporizador al funcionar como *periodic,* puede realizar un conteo desde un valor determinado hasta alcanzar un valor deseado y al terminar regresará a su valor inicial y repite nuevamente el conteo. Al funcionar como *one shot*, al alcanzar el valor deseado regresa al valor inicial pero no continua con el conteo. En ambos casos es posible asociar una interrupción al evento de terminar el conteo, con el fin de comunicarle al procesador que se ha terminado el conteo. El módulo temporizador realiza el conteo a la frecuencia de la señal de reloj entregada al periférico.

Utilizando la librería es posible configurar el temporizador de la siguiente manera:

```
SysCtlPeripheralEnable ( SYSCTL_PERIPH_TIMER0 );
TimerConfigure ( TIMER0_BASE , TIMER_CFG_PERIODIC );
```

Un temporizador como cualquier otro periférico en este microcontrolador, necesita una señal de reloj, la palabra SYSCTL_PERIPH_TIMER0 habilita el reloj hacia el temporizador 0 de 16/32 bits. La sentencia Timer Configure recibe como parámetro la base de las direcciones de memoria de los registros referentes a un determinado módulo temporizador. Por ejemplo, para el temporizador cero el valor de la base esta dado por TIMER0_BASE, luego se coloca las configuraciones, la librería ofrece definiciones para estas configuraciones, tales como las palabras TIMER_CFG_PERIODIC o TIMER_CFG_ONE_SHOT, que configura el periférico como periódico o de un tiro.

En caso de desear configurar los subtemporizadores es posible hacerlo con palabras como TIMER_CFGA_PERIODIC, TIMER_CFGA_ONE_SHOT, TIMER_CFGB_PERIODICo TIMER_CFGB_ONE_SHOT. Donde la raíz

TIMER_CF*, denota el temporizador de 16 bits que se esta configurando. En la hoja de datos del dispositivo y en la hoja de datos de la librería puede verse una serie de configuraciones que pueden tomar los temporizadores y van desde la dirección del conteo hasta comportamientos del contador diferentes a los de un temporizador.

De forma predeterminada el conteo se hace de forma descendente, por lo tanto se le carga al temporizador un determinado valor y por cada flanco de la señal de reloj disminuirá el valor actual del conteo en una unidad hasta llegar a cero.

Para ingresar el valor inicial del conteo se utiliza la rutina TimerLoadSet, la cuál recibe como parámetro la base de las direcciones de los registros referentes a un módulo temporizador determinado, un valor que indica si se utiliza el *TimerA* o el *TimerB*. Los valores para indicarlo están definidos por la librería con las palabras Timer_A y Timer_B. En caso de usarse el *timer* completo, en lugar de dos pequeños *timers*, solamente se puede utilizar el valor Timer_A. Y como último parámetro recibe el valor en el cual comenzará el conteo descendente. La rutina configurando el tiempo a esperar por estado se vería así:

```
TimerLoadSet ( TIMER0_BASE , TIMER_A , FSM [ estado ]. Tiempo -1);
```

Donde el valor del tiempo está dado por el estado, dictado por la sentencia FSM[estado].Tiempo, se debe notar que el conteo llega a cero por lo que el cero se cuenta y por eso se resta uno, para ser consistente con el valor del tiempo.

Hasta ahora se encuentra configurado el temporizador cero, es necesario escribir código que permita generar interrupciones cuando el temporizador haya alcanzado cero en el conteo empezando desde el valor cargado por TimerLoadSet, con el fin que el procesador tenga una manera de conocer cuando el tiempo haya transcurrido.

Para ello se habilita una interrupción específica en la NVIC utilizando la función IntEnable, la cual recibe como parámetro el valor de la interrupción a habilitar. Para habilitar la interrupción por TimerA, el único a utilizar cuando se utiliza el temporizador como un contador de 32 bits, se utiliza el valor INT_TIMER0A. Luego se habilita al temporizador, para ser capaz generar interrupciones, además de indicarle que eventos son los que generaran una interrupción. Por ejemplo, un *timer* puede generar una interrupción al terminar un conteo o al realizar una operación de DMA. Por último, se habilitan las interrupciones en el procesador con la rutina IntMasterEnable, que no recibe parámetros.

En resumen, las interrupciones deben ser generadas por un periférico, controladas por un controlador de interrupciones y atendidas por el procesador. Por lo que se debe configurar cada uno de ellos para que realicen sus funciones en la ejecución de interrupciones.

```
TimerIntEnable ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
IntEnable ( INT_TIMER0A );
IntMasterEnable ();
```

Ya se han configurado interrupciones y el comportamiento del temporizador. Se procede a marcar el inicio del conteo. Utilizando la función TimerEnable, se habilita el conteo en el *timer*.

```
TimerEnable ( TIMER0_BASE , TIMER_A );
```

Esta función recibe como parámetros la base del temporizador y el temporizador de 16 bits con el que se está realizando el conteo, en caso de utilizar el *timer* como un contador de 32 bits solo está permitido usar el valor TIMER_A. ¿En qué parte del programa debe ir la implementación de la dinámica de la máquina de estados? Con esto configurado, el dispositivo para generar los eventos necesarios para poder tener transiciones entre estados se procede a configurar la rutina a ejecutar cada vez que el temporizador 0 termine un conteo.

Esta rutina debe ejecutarse tras cada evento de *Timeout* y debe realizar acciones para cada transición. Una de ellas es configurar el tiempo que debe tardarse para generar una nueva transición, además de configurar la adecuada para el siguiente estado. Para indicarle al procesador que esta rutina se ejecute al tener un evento de *Timeout* generado por el *timer* 0; es necesario dirigirse al archivo *STARTUP* donde se encuentra la *NVIC Table* que contiene todas las rutinas asociadas a una excepción. Se debe buscar la posición en la tabla que corresponde a la interrupción generada. Por ejemplo, usando el temporizador 0 como un contador de 32 bits, el único que es responsable de generar las interrupciones es el *TimerA* por lo que se debe buscar en la tabla la posición para el *TimerA* del temporizador 0.

El sector de la tabla para el tm4c123gh6pm tendría una estructura similar a la siguiente:

```
IntDefaultHandler ,              // ADC Sequence 3
IntDefaultHandler ,              // Watchdog timer
IntDefaultHandler ,              // Timer 0 subtimer          A
```

```
IntDefaultHandler ,          // Timer 0 subtimer      B
IntDefaultHandler ,          // Timer 1 subtimer      A
IntDefaultHandler ,          // Timer 1 subtimer      B
```

Debe cambiarse el nombre de la rutina IntDefaultHandler, o cualquier otro que se encuentre en su lugar por el nombre a utilizar por el *TimerA* del temporizador 0. En este programa se utilizó el nombre Transicion para ejecutar la ISR de las interrupciones generadas por el *TimerA* del temporizador 0. Quedando este sector de la tabla de una forma similar a la siguiente:

```
IntDefaultHandler ,          // ADC Sequence 3
IntDefaultHandler ,          // Watchdog timer
Transicion ,                 // Timer 0 subtimer A
IntDefaultHandler ,          // Timer 0 subtimer      B
IntDefaultHandler ,          // Timer 1 subtimer      A
IntDefaultHandler ,          // Timer 1 subtimer      B
```

Se debe tener cuidado de no cambiar el nombre de ninguna otra rutina, y de no alterar el tamaño de la tabla o posiciones de las rutinas, borrando o agregando elementos, ya que la posición en esta tabla indica la ISR en el NVIC.

Una vez alterada la tabla para indicarle la ISR, es necesario indicarle al compilador que esa rutina existe por lo que se declara en algún lugar del código anterior a la declaración de la tabla; no será acá donde se configurará lo que hace esta ISR al entrar por razones de comodidad se hará en el archivo del programa principal. El archivo de STARTUP se encuentra afuera del programa principal, por lo que para indicarle al compilador cuando revise este archivo que la rutina de la interrupción se encuentra en otro archivo se utiliza el

especificador de almacenamiento extern quedando la declaración de la rutina de la siguiente forma:

```
extern void Transicion ( void );
```

Ahora, en el archivo del programa principal se configura lo que hará la rutina cada vez que el *TimerA* del *timer* 0 genere una interrupción al procesador. En una máquina de Mealy la verificación de la sentencia que habilita una transición se realiza de primero por lo que una interrupción se adapta perfectamente a este modelo ya que para entrar a la ISR primero la interrupción debió haber ocurrido. Por tanto, primero se generan las salidas y se hacen las asignaciones a los registros del temporizador para realizar un conteo diferente y luego se realiza el cambio de estado, dependiendo de la transición habilitada.

```
void Trancision ( void )
{
TimerIntClear ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 , FSM
[ estado ]. Salida );
TimerLoadSet ( TIMER0_BASE , TIMER_A , FSM [ estado ]. Tiempo -1);
estado = FSM [ estado ]. Siguiente ;
}
```

El código de la rutina Trancision, además de la implementación del procedimiento descrito con anterioridad de la dinámica de una máquina de Mealy, lleva la sentencia TimerIntClear que limpia la bandera de la interrupción generada. A pesar que el *timer* se encuentra configurado como periódico no implica que levante la interrupción cada vez que suceda un evento. Al suceder un evento el *timer* modifica un bit (TnTORIS) colocando siempre un 1 lógico en él, en el registro GPTMRIS, para indicar que el evento ya sucedió. Y lo sostiene

hasta que se coloque por *software* un 0 lógico en el bit y se hace escribiendo un valor específico en el registro GPTMICR. La instrucción TimerIntClear, de la librería, realiza este trabajo. Para hacerlo recibe dos parámetros, la base del *timer* y el valor que indica el evento asociado a la interrupción. En este caso un *timeout* debido al *TimerA*, tiene asociado un valor específico y se puede acceder gracias la librería como TIMER_TIMA_TIMEOUT.

El código 3.7 muestra la totalidad del programa escrito en C, necesario la implementación de la máquina de Mealy descrita en la figura 94.

Tabla LXXXIII. **Código 3.7: uso de interrupciones para transiciones entre estados**

```
1 /*
2 * main .c
3 */
4 # include <stdint .h>
5 # include <stdbool .h>
6 # include " inc / tm4c123gh6pm .h"
7 # include " inc / hw_memmap .h"
8 # include " inc / hw_types .h"
9 # include " driverlib / sysctl .h"
10 # include " driverlib / interrupt .h"
11 # include " driverlib / gpio .h"
12 # include " driverlib / timer .h"
13
14
15 # define Amarillo 0
16 # define Rojo 1
17 # define Verde 2
18
19 # define tAmarillo 160000000
20 # define tRojo 320000000
21 # define tVerde 240000000
22
23 # define LEDAmarillo 0 x0A
24 # define LEDRojo 0 x02
25 # define LEDVerde 0 x08
26
27 struct Estado {
28 uint32_t Salida ;
29 uint32_t Tiempo ;
30 uint32_t Siguiente ;
```

Continuación de la tabla LXXXIII.

```
31 };
32
33 // Habilitar la creaci ón de estructuras de la forma de Estado
34 typedef const struct Estado EstadoTipo ;
35
36 // Creaci ón de una estructura para albergar una FSM
37 EstadoTipo FSM [3]={
38 {0 x0A , tAmarillo , Rojo },
39 {0 x02 ,tRojo , Verde },
40 {0 x08 , tVerde , Amarillo }
41 };
42
43 uint32_t estado = Rojo ;
44
45
46 int main ( void )
47 {
48
49 SysCtlClockSet ( SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN )
;
50
51 SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOF );
52 GPIOPinTypeGPIOOutput ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 );
53
54 SysCtlPeripheralEnable ( SYSCTL_PERIPH_TIMER0 );
55 TimerConfigure ( TIMER0_BASE , TIMER_CFG_PERIODIC );
56
57 TimerLoadSet ( TIMER0_BASE , TIMER_A , FSM [ estado ]. Tiempo -1);
58
59 IntEnable ( INT_TIMER0A );
60 TimerIntEnable ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
61 IntMasterEnable ();
62
63 TimerEnable ( TIMER0_BASE , TIMER_A );
64
65 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 , FSM [ estado ]. Salida
);
66 while (1)
67 {
68 }
69 }
70
71 void Timer0IntHandler ( void )
72 {
73 TimerIntClear ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
74 estado = FSM [ estado ]. Siguiente ;
75 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 , FSM [ estado ]. Salida
);
76 TimerLoadSet ( TIMER0_BASE , TIMER_A , FSM [ estado ]. Tiempo -1);
77 }
```

Fuente: elaboración propia, empleando C/tilaware.

Al momento de implementar este código podrá verse. Al iniciar el programa en la tarjeta TivaC, el color que mostrará el led será verde a pesar

que el estado que se declaró en la variable estado es rojo. Pero no debe olvidarse que se trata una implementación de una máquina de Mealy, y en este tipo de máquinas las salidas no dependen del estado sino de la entrada y el estado anterior mostrándolas en la transición, por ello se debe tener cuidado ya que se pueden obtener resultados similares con modelos diferentes, pero la implementación para ser consistente con el modelo, ya sea de Mealy o Moore, debe ser diferente entre sí. Debe notarse que para una máquina de Moore la salida debe ser única, para una máquina de Mealy se pueden tener varias salidas. Para una máquina de Moore la salida depende únicamente del estado actual, por lo que primero se asigna el estado y luego se ejecuta la salida. En una máquina de Mealy la salida depende del estado y la entrada, pero se ejecutan antes de pasar a un estado.

3.8. Prácticas de laboratorio. Módulo 3. Diseño e implementación de una dinámica discreta

Se presenta a continuación una serie de problemas propuestos para resolver como parte del laboratorio del curso de Sistemas de Control, los cuales tienen como intención evaluar la capacidad de diseñar modelos de dinámicas discretas que cumplan con algunas especificaciones, así como reforzar los conceptos teóricos dados al inicio del capítulo. Para ello, aunque es posible construirlos en cualquier dispositivo, se recomienda la tarjeta de desarrollo TivaC.

3.8.1. Elementos básicos de un microcontrolador

Utilizando el microcontrolador de su elección, realizar los siguientes mecanismos, debe considerarse que el microcontrolador escogido posea al

menos un *timer* y ser capaz de levantar interrupciones externas en los pines digitales:

- Se cuenta con un led RGB y dos botones, nombrados como 1 y 2, respectivamente. En un led RGB debe mostrarse el color verde al tener presionado el botón 1 y el botón 2 no, si se tiene presionado el botón 2 y el 1 no, debe mostrarse el color rojo en el led RGB, si se presionan ambos botones debe mostrarse el azul. En caso de no tener ningún botón presionado, el led debe de estar apagado.

- Haciendo uso de interrupciones en los pines digitales. Un led RGB se encuentra en color azul, al presionar un botón pasa a tener el color rojo y al volverlo a presionar el led pasará a tener el color azul nuevamente. Siendo posible alternar con un botón el color del led entre azul y rojo.

- Utilizando un *timer* del microcontrolador. Hacer un variador de frecuencia. Inicialmente el microcontrolador deberá generar una señal de 200 Hz en uno de los pines del microcontrolador. A medida que se presiona un botón, la frecuencia aumenta 10 Hz por cada vez que se presione el botón. Si se presiona un segundo botón, diferente del que aumenta la frecuencia, entonces la frecuencia de la señal deberá disminuir 10 Hz. Para la evaluación de esta parte el auxiliar del laboratorio puede revisar el funcionamiento con un osciloscopio.

3.8.2. Máquinas de Moore

La definición dada con anterioridad de máquinas de estados, al principio de este capítulo, era referente a máquinas de estados de Mealy. A continuación, se da una definición matemática de las máquinas de estado de Moore.

Tabla LXXXIV. **Definición de máquinas de Moore de estados finitos**

Una máquina de estados de Moore es una tupla de compuesta de 6 elementos:

$$\{Estados, Entradas, Salidas, Actualización, EstadoInicial, AsignaciónSalida\} \quad \text{[Ec. 3.25]}$$

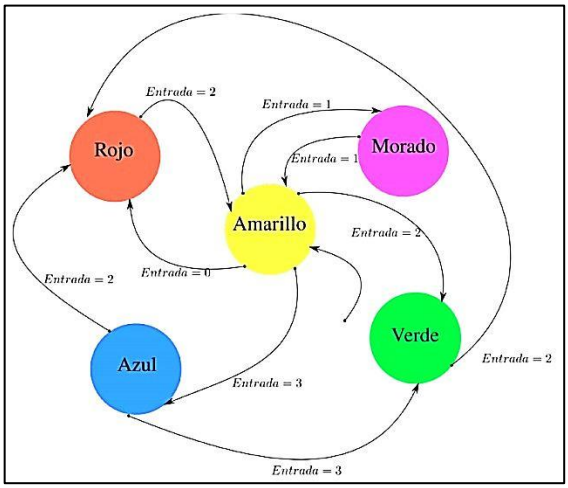
Donde

- Estados es un conjunto de estados.
- Entradas es un conjunto de valuaciones.
- Salidas es un conjunto de valuaciones.
- Actualización una función de la forma $Estados \times Entradas \rightarrow Estados$, la cual asigna a cada estado y valuación en las entradas unicamente un siguiente estado y una valuación en las salidas.
- Estado Inicial es el estado donde comienza la máquina de estados.
- AsignacionSalida es una función de la forma $Estados \rightarrow Salidas$.

Fuente: LEE, Edward. A.; SESHIA, Sanjit. A. *Introduction to embedded systems*. p. 198.

A continuación, se presenta una máquina de estados de Moore, en una versión de grafo, donde las salidas de cada estado están dadas por el nombre de los estados.

Figura 95. **Máquina de Moore**



Fuente: elaboración propia, empleando Inkscape.

- Resolver lo siguiente:

    - Dar la versión del modelo como una tabla de estados.
    - Dar la versión del modelo como tupla y función de actualización. Para ello se tiene que definir la función de actualización y de asignación de salidas.
    - Implementar la dinámica en unmicrocontrolador. Para ello debe contar por lo menos con un led RGB, 3 pines como salidas digitales, 2 pines como entradas digitales y algún dispositivo mecánico o eléctrico que genere las entradas (un botón por ejemplo).

3.8.3. Máquinas de Mealy

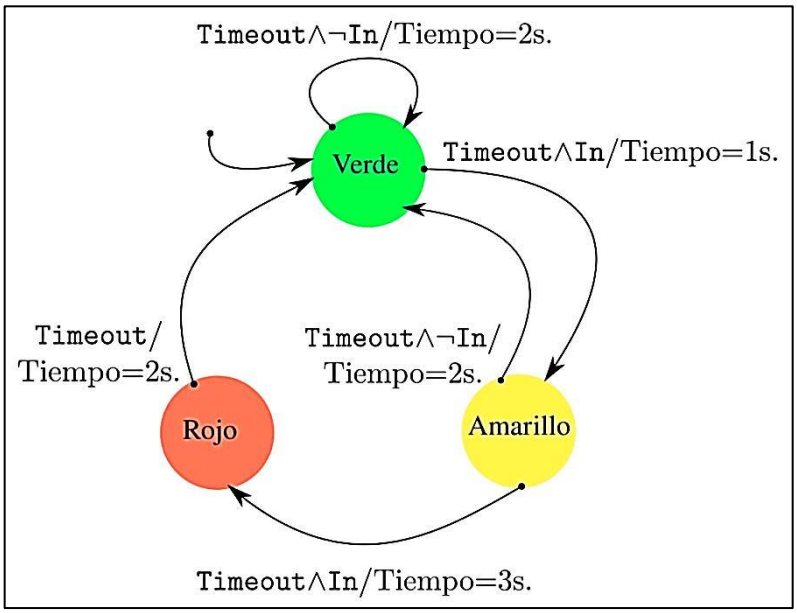
La máquina de estados de la figura 96 representa un semáforo con la capacidad de tomar decisiones en la presencia de peatones que deseen cruzar las calles, a diferencia a la mostrada en la figura 94 que solo toma en cuenta un evento llamado Timeout para realizar el salto a otro estado.

En esta máquina también existe el evento Timeout, indicado por algún temporizador con el fin de marcar cuando realizar una transición, por lo que si sucede el evento Timeout evaluará True. Sin embargo, también se tiene una entrada digital gobernada por un sensor de presencia para peatones queriendo cruzar la calle.

Inicialmente, el semáforo se encuentra en verde si no hay un peatón queriendo cruzar la calle, la entrada al sistema discreto In evaluará False; si hay un peatón In evaluará True. Por tanto, si no hay un peatón esperando a cruzar la calle al terminar los dos segundos que le corresponden a verde, regresará a

esperar dos segundos en este estado para evitar desperdicio de tiempo en la calle, de lo contrario si al terminar los dos segundos hay un peatón a la espera de cruzar la calle el semáforo pasará a amarillo para alertar a los vehículos, y si el peatón sigue a la espera pasará a rojo donde esperará tres segundos y luego regresará a verde. Si el peatón se ha ido del lugar ya que solo pasó por el lugar del sensor sin querer pasar la calle, entonces el semáforo pasará de nuevo a verde.

Figura 96. **Máquina de Mealy**



Fuente: elaboración propia, empleando Inkscape.

- Resolver lo siguiente:

    - Dar la versión del modelo de la figura 96 como funciones de actualización consistente con la definición 31.

- ¿Cómo modificaría la estructura Estado y FSM en el código para ser consistente a la definición 31? Es decir que el siguiente estado y la salida, ambos, dependen de la entrada y el estado.

- Implementar la máquina de estados de la figura 96 en un microcontrolador. Tener en cuenta que para ello necesitará un microcontrolador que posea al menos un *timer* y un módulo de salidas y entradas digitales de propósito general. Es necesario contar también con algún botón o cualquier otro dispositivo capaz de generar también la señal producida por el sensor en la entrada In.

3.8.4. Diseño de una máquina de estados

Una parte fundamental en ingeniería es diseñar sistemas para determinados problemas o aplicaciones. En esta parte utilizando un microcontrolador con entradas y salidas digitales de propósito general, implementar una dinámica discreta para el siguiente escenario:

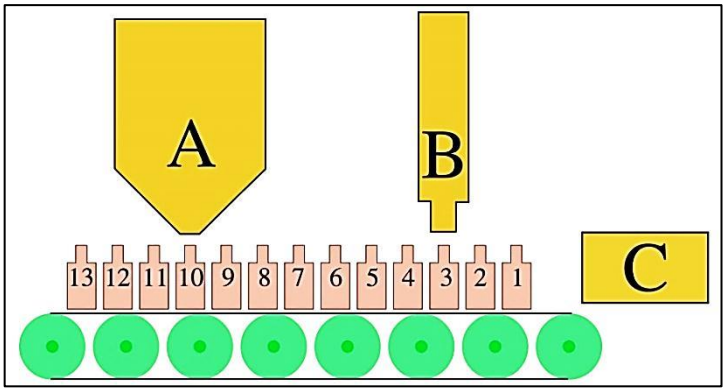
Es necesario automatizar una banda transportadora de botellas, a través de la banda se encuentran dos sensores que detectan si una botella se encuentra en la posición exacta A o B, una dispensadora de soda llena las botellas en la posición A y una selladora coloca las roscas en la boquilla en la posición B. Cuando un sensor detecta la presencia de una botella, ya sea en A o B, detiene el movimiento de la banda y activa, en caso de estar en A, la dispensadora por 3 segundos y luego activa de nuevo la banda, siempre y cuando no haya botella alguna en la posición B. En caso de estar en B, de igual forma detiene el movimiento de la banda y activa la selladora por 5 segundos, pasado este tiempo pone en movimiento de nuevo a la banda hasta llegar a la

posición **C** donde un contador lleva el control de cuantas botellas van pasando por ese lugar. Cada 3 botellas un etiquetador en C pone una estampa diferente a las dos botellas anteriores.

En resumen, el sistema se comporta de la siguiente manera:

- Si hay una botella en la posición A, se detiene la banda y se administra líquido durante 3 segundos.
- Si hay una botella en la posición B, se detiene la banda por 5 segundas y se coloca una tapa a la botella.
- En caso de no haber botella en los sensores A y B, la banda se debe estar moviendo para transportar las botellas.
- Cada tres botellas se etiquetan de forma diferente una botella.

Figura 97. **Caricatura del entorno**



Fuente: elaboración propia, empleando Inkscape.

Escoger un microcontrolador que se adapte a la dinámica discreta descrita (el TM4C123GH6PM fácilmente satisface las condiciones necesarias). Luego, diseñar una máquina de estados, ya sea de Mealy o Moore, para esta situación e implementarla en un microcontrolador. Para los sensores, utilizar botones o

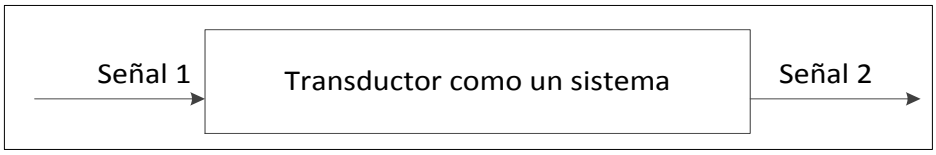
cualquier otro dispositivo para simular el comportamiento de cada sensor. Para la dispensadora y selladora utilizar led de diferentes colores para indicar que se está activando cada dispositivo. En el caso de la posición C utilizar un botón para simular la máquina que realiza el conteo de botellas y sostener una señal lumínica para las cuáles cuyo número es múltiplo de 3 y otra para el resto de las botellas. Para la banda transportadora utilizar un led para indicar cuando está funcionando.

4. ENLAZADO EN EL MUNDO FÍSICO Y EL MUNDO CIBERNÉTICO

4.1. Transductores

Al construir un sistema físicocibernético son necesarios dispositivos capaces de transmitir de un sistema a otras señales de distinta naturaleza, ya que en estos sistemas es necesario interactuar con el entorno donde se encuentra el sistema. En esta tarea se ven implicados los transductores, que son dispositivos que convierten un tipo de energía a otro. Es común encontrar transductores que convierten energía eléctrica, mecánica, química, electromagnética, acústica o térmica.

Figura 98. **Bloque de un sensor**



Fuente: elaboración propia, empleando Inkscape.

Dicho de otro modo, los transductores son los elementos de un sistema físicocibernético que lo relacionan con el entorno. Es posible mencionar dos tipos de transductores: sensores y actuadores. Un sensor siempre interpreta señales del entorno y las traduce a señales comprensibles por el sistema físicocibernético. Un actuador siempre interpreta señales del sistema físicocibernético, las convierte y luego entrega el resultado al entorno.

Actualmente, existe una fuerte tendencia hacia una tecnología capaz de enlazar profundamente el entorno físico en que nos encontramos con la información nuestra y del mundo a través de sensores y actuadores inteligentes. Por ejemplo, un sensor colocado en un equipo que mide la temperatura de este mismo, puede ser accedido a distancia y de esta forma monitorear el comportamiento del entorno y la forma en que éste lo afecta.

Es posible colocar la información en un servidor y acceder a ella desde cualquier parte del mundo, de esta forma si una compañía multinacional posee una gran cantidad de equipos distribuidos en diferentes partes del mundo y todos ellos susceptibles a diferentes variables del entorno, como humedad o temperatura, en lugar de crear centros de monitoreo para cada ubicación, puede centralizar el monitoreo en un solo punto geográfico ahorrando recursos y personal. Para ello es necesario que los sensores proporcionen la información que extraen del entorno en forma digital, para poder tener una forma de conectarse a la red.

Imagínese un escenario donde las cosas con las que las personas se relacionan diariamente tengan conexión a internet. Un escenario donde pueda trasladarse información de un lugar a otro en un instante a través de cosas tan cotidianas e insignificantes como un cepillo de dientes, un par de anteojos, zapatos, ropa interior, utensilios de cocina entre muchos otros. Esto sería un preludio a la IoT o *internet of things*. Pero para llevarse a cabo los transductores utilizados en estos sistemas deben poder transformar la información que extraen o entregan al entorno, como señales eléctricas, mecánicas, térmicas, químicas o lumínicas, a señales que sean comprendidas por una computadora.

4.1.1. Sensores

Es un dispositivo que mide una magnitud física. En sistemas electrónicos es común encontrar sensores que producen un voltaje proporcional a la magnitud física que se encuentran midiendo, aunque es posible encontrar algunos que generen señales de corriente. Siempre un sensor es considerado un sistema que tiene sus entradas provenientes del entorno y sus salidas se entregan a una unidad de procesamiento de señales.

Es frecuente encontrar en los sistemas embebidos que las señales de voltaje generadas por un sensor conectadas a un periférico de *ADC* en un microcontrolador. Otro caso frecuente es encontrar sensores capaces de comunicarse con una unidad de procesamiento digital mediante protocolos de comunicación digitales discretos. También es posible encontrar sensores que se encuentren en el mismo empaquetado de la unidad de procesamiento o encontrar tarjetas de desarrollo con unidades de procesamiento y sensores en la misma tarjeta.

4.1.2. Actuadores

Es un dispositivo que altera una magnitud física. En los sistemas electrónicos es común encontrar un sistema intermedio entre el actuador y la unidad de procesamiento, el cual se encarga de entregar al actuador la potencia necesaria para que realice su tarea, cualquiera que fuere. Las señales provenientes de la unidad de procesamiento son útiles únicamente para definir el comportamiento del actuador. Este sistema tiene su razón de ser, debido a que los sistemas de procesamiento digitales discretos, por el tamaño que poseen y frecuencia a la que operan, no pueden trabajar con señales de alta

potencia y conectar un actuador que utilice una potencia superior a la tolerada por la unidad de procesamiento la dañaría.

En muchos casos los sistemas de acople son transparentes para el diseñador, pudiendo conectar la unidad de procesamiento inmediatamente al actuador, en otras palabras la parte que realiza el trabajo, pudiendo conectar la unidad de procesamiento inmediatamente al actuador, en otras palabras la parte que realiza el trabajo y la parte que interpreta señales del procesador vienen embebidas en un empaquetado y no es necesario tomar en cuenta la forma en que trabajan por separado, sino la forma como lo hacen en conjunto.

4.2. Algunos sensores comunes

A continuación, se listan algunas magnitudes y sensores comunes de encontrar en sistemas físicocibernéticos.

4.2.1. Midiendo inclinación y aceleración, acelerómetro

Un acelerómetro es un instrumento que mide la aceleración propia, la cual es la aceleración que experimenta el dispositivo en relación a una caída libre. En cualquier punto del espacio el principio de equivalencia[5] garantiza la existencia de un marco inercial y un acelerómetro mide la aceleración relativa al marco de referencia. Por lo general, las mediciones hechas por un acelerómetro son hechas en términos de fuerza-G.

Un acelerómetro en reposo relativo a la superficie de la tierra indicará 1*g* hacia arriba,[6] porque en cualquier punto sobre la superficie la Tierra se

[5] También conocido como principio de Einstein y son varios conceptos relacionados con la gravedad y la masa inercial.

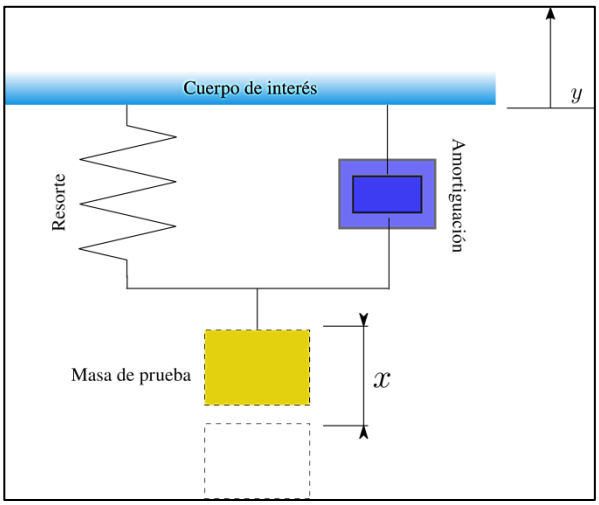
[6] $1g = 9{,}80665\ m/s^2$

encuentra acelerando hacia arriba en relación al marco de referencia inercial local (el marco de referencia del objeto en caída libre cerca de la superficie).

Para obtener la aceleración debida al movimiento con respecto a la Tierra, esta diferencia de gravedad se debe sustraer y hacer correcciones por los efectos causados por la rotación de la Tierra en relación al marco de referencia inercial. La razón de la aparición de una gravedad extra, es de nuevo el principio de Einstein, el cual enuncia que los efectos de la gravedad en un objeto son indistinguibles de la aceleración.

El principio del funcionamiento de un acelerómetro muy básico puede ser explicado por una simple masa *m* fija a un resorte con constante de elasticidad *k* que a su vez se encuentra fijo a un marco fijo. La masa usada en este modelo usualmente recibe el nombre de masa de prueba o masa sísmica. Para evitar que el sistema oscile se coloca una amortiguación con constante *b* y se modela en paralelo al resorte. Cuando el sistema completo se ve sujeto a una aceleración lineal, una fuerza igual a la masa *m* por la aceleración actual sobre la masa sísmica, causando que esta se desvíe de su posición. Esta desviación es medida por un medio apropiado y convierte esta medida en una señal eléctrica.

Figura 99. **Modelo de un acelerómetro**



Fuente: elaboración propia, empleando Inkscape.

Este es un simple problema de física que puede ser resuelto utilizando las leyes de Newton o en su defecto la mecánica de Lagrange. Desde el punto de vista de un observador estacionario se tiene la suma de las fuerzas, todas ellas en la misma dirección:

$$F_{Aplicada-}\ F_{Amortiguación-}\ F_{Resorte=m\ddot{x}} \qquad \text{[Ec. 4.1]}$$

Simplificando los términos se obtiene la clásica ecuación del resorte amortiguado:

$$m\ddot{x} + b\dot{x}\ + kx = F_{Aplicada} \qquad \text{[Ec. 4.2]}$$

Donde *x* es el movimiento relativo de la masa de prueba respecto al marco de referencia del dispositivo. Por lo que se puede ver el acelerómetro más básico es un sistema dinámico (con memoria), y las mediciones se encuentran amarradas a la sensibilidad $S = \frac{x}{a} = \frac{m}{k}$. Se puede ver por la forma de la

ecuación 4.2 que en las mediciones estarán presentes diversas oscilaciones, y este es un problema en el uso de un acelerómetro ya que cualquier medición que se haga será sensible a las vibraciones. Además, puede verse que si el dispositivo se encuentra moviéndose de tal forma que existan fuerzas inerciales debido a este movimiento, será muy difícil determinar la inclinación de un dispositivo utilizando solamente las lecturas de un acelerómetro.

Por ejemplo, si se intenta estabilizar un *quadcopter* o calibrar un sistema PID, se tendrán complicaciones utilizando la información pura proveniente del acelerómetro y será necesario acompañarse de otros dispositivos y aplicar una serie de filtros para quitar o minimizar el efecto de las vibraciones en las mediciones.

Actualmente, la mayoría de acelerómetros son implementados en silicio, donde bajo el efecto de la gravedad o alguna aceleración se deforman tiras de silicio, la circuitería mide la deformación y provee una lectura digital. Frecuentemente, se utilizan 3 acelerómetros y son empaquetados juntos, dando un acelerómetro de tres ejes. Suelen ser usados para medir la orientación relativa a la gravedad, más la aceleración en cualquier espacio tridimensional.

Es posible encontrar en el mercado, útiles en los CPS, acelerómetros digitales los cuales serán capaces de brindar la información utilizando algún protocolo como I2C, SPI o UART, y acelerómetros analógicos cuya salida será un voltaje en un rango predefinido que será necesario convertir utilizando un módulo de ADC.

El modelo expuesto como un sistema masa resorte amortiguado representa solamente un eje. Si se colocan perpendicularmente tres dispositivos de un solo eje se formaría un acelerómetro de 3 ejes. Para realizar

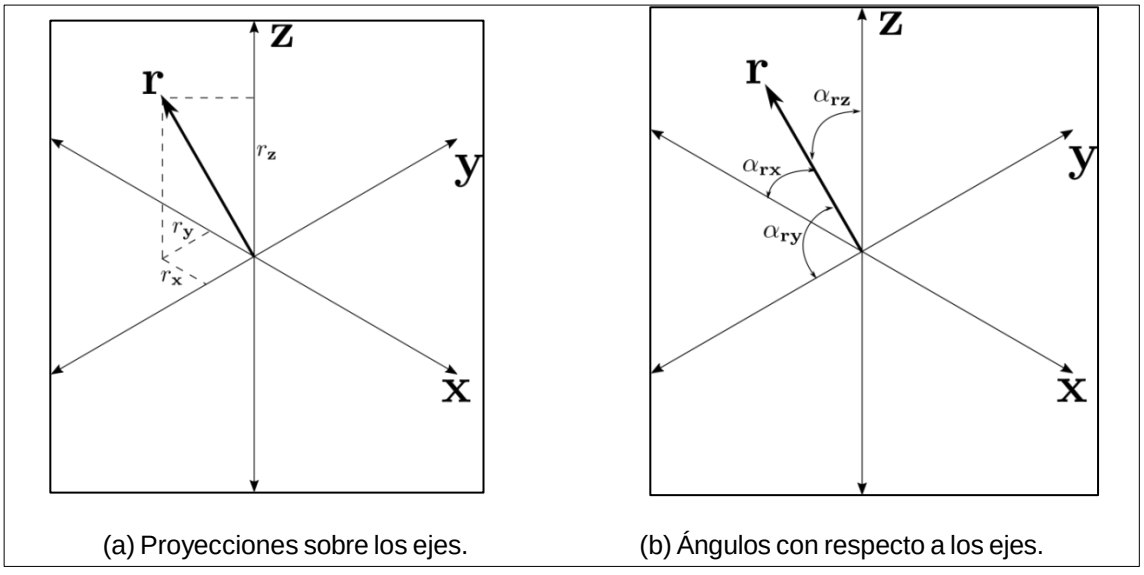
cálculos utilizando un acelerómetro de tres ejes es más práctico colocar un sistema de coordenadas fijo sobre el dispositivo, y considerar que las fuerzas se aplican alrededor de este sistema.

Considérese espacio euclidiano fijo en el dispositivo, compuesto por las componentes x, y y z; con el origen en su centro de masa. Sea r el vector de fuerza que mide el acelerómetro, el cual podría ser fuerza de gravedad, una fuerza inercial o una combinación de ambas; sean r_x, r_y y r_z, las proyecciones del vector r. Por lo que la norma $r = |r|$, estaría dada por:

$$r = r_x^2 + r_y^2 + r_z^2 \qquad \text{[Ec. 4.3]}$$

La figura 100 muestra el espacio fijo en el dispositivo con la fuerza **r** la cual el sensor se encuentra midiendo.

Figura 100. **Espacio euclidiano fijo en un acerómetro**



(a) Proyecciones sobre los ejes. (b) Ángulos con respecto a los ejes.

Fuente: elaboración propia, empleando Inkscape.

Supóngase por un momento que una unidad de procesamiento puede obtener los valores de las proyecciones utilizando, ya sea un protocolo digital o un módulo de ADC para leer una señal analógica. Si el dispositivo no se encuentra sujeto a ninguna otra fuerza más que la gravedad (es decir, se encuentra en reposo o se mueve a velocidad constante, para evitar la presencia de fuerzas inerciales), se puede asumir que el valor obtenido por las lecturas de las proyecciones es el valor de la gravedad. Si se desea calcular la inclinación del dispositivo en relación a la gravedad, se puede lograr calculando el ángulo del vector y el eje z. De forma similar se pueden calcular los ángulos que forma el vector r con respecto al eje y y el eje x si se desea saber la inclinación por eje:

$$\alpha_{xr} = arcos(r_x/r) \quad \text{[Ec. 4.4]}$$

$$\alpha_{yr} = arcos(r_y/r) \quad \text{[Ec. 4.5]}$$

$$\alpha_{zr} = arcos(r_z/r) \quad \text{[Ec. 4.6]}$$

Los valores normalizados de las proyecciones son conocidos como cosenos directores y básicamente representan un vector de magnitud uno.

$$cos_x = \cos(\alpha_{xr}) = r_x/r \quad \text{[Ec. 4.7]}$$

$$cos_y = \cos(\alpha_{yr}) = r_y/r \quad \text{[Ec. 4.8]}$$

$$cos_z = \cos(\alpha_{zr}) = r_z/r \quad \text{[Ec. 4.9]}$$

4.2.2. Midiendo posición y velocidad

Teóricamente, dada una medida *x* de la aceleración sobre el tiempo, es posible determinar la velocidad y ubicación de un objeto. Considérese un objeto moviéndose en una sola dirección. Sea la posición $p: \mathbb{R}_+ \to \mathbb{R}$, con una posición inicial p(0). Sea la velocidad del objeto $v: \mathbb{R}_+ \to \mathbb{R}$ con una velocidad inicial v(0). Entonces:

$$p(t) = p(0) + \int_0^t v(\tau)d\tau \qquad \text{[Ec. 4.10]}$$

$$y$$

$$v(t) = v(0) + \int_0^t x(\tau)d\tau \qquad \text{[Ec. 4.11]}$$

Si un sensor mide x´(*t*) la cual es la aceleración real *x*(*t*) con un sesgo constante *b*.

$$x'(t) = x(t) + b \qquad \text{[Ec. 4.12]}$$

Como resultado, el error en el cálculo de *p*(*t*) dará un error que crece proporcionalmente a t^2. Dicho error es llamado *drift*, y esta es la razón de porque es prácticamente inútil utilizar únicamente un acelerómetro para realizar mediciones de la posición. Medir posición es complicado, para ello resulta más útil utilizar un sistema de posición satelital GPS (siglas en inglés de *global positioning system*). Sin embargo, las señales provenientes de este sistema son relativamente débiles y fácilmente se ven obstruidas por edificios y otros obstáculos. Es posible usar un acelerómetro para palear este defecto para realizar mediciones y aproximar de mejor manera la posición, si esta se ve reiniciada durante cortos lapsos por el GPS a un valor confiable.

Existen otros mecanismos para realizar mediciones de posición en interiores o dentro de un área geográficamente relativamente pequeña, por ejemplo wifi *fingerprinting*, la cual se basa en medir la intensidad de una señal de wifi y utilizando una huella digital, donde la ubicación de un punto de acceso wifi es conocida, se puede triangular la posición del dispositivo realizando esta medición.

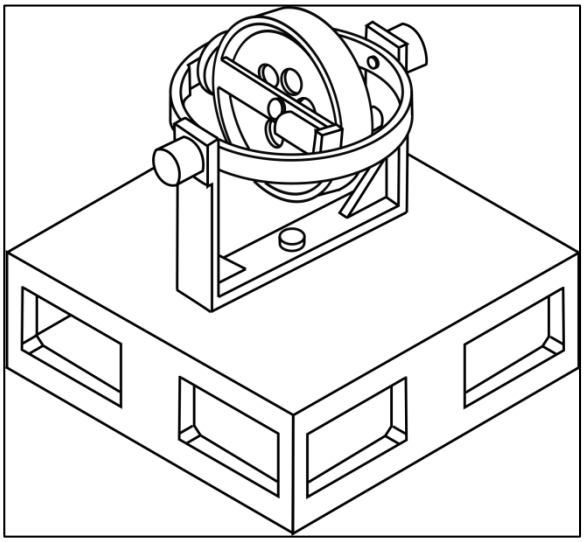
4.2.3. Midiendo la rotación, giroscopio

Un giroscopio es un instrumento que mide el cambio en la orientación. A diferencia de un acelerómetro, casi no se afecta por el campo gravitacional. Los giroscopios tradicionales son construidos mecánicamente, por monturas giratorias, tal como el que se muestra en la figura 101, el cual se basa en la conservación del momento angular. Existen muchos métodos diferentes para construir dispositivos capaces de calcular el cambio de orientación, por ejemplo, los giroscopios FOG (siglas en inglés de *fiber optic gyroscope*) los cuales usan la interferencia de la luz para detectar rotación mecánica.

Los giroscopios modernos, se basan en tecnología MEMS (siglas en inglés de *microelectromechanical systems*), que se centra en dispositivos electromecánicos muy pequeños, los cuales son construidos con elementos entre 1 y 100 micrómetros de tamaño generando dispositivos de alrededor 20 micrómetros de tamaño a un milímetro. Los giroscopios hechos con tecnología MEMS utilizan versiones de los giroscopios tradicionales a menor escala de forma tal que puedan ser implementados a la escala MEMS, y son utilizados en toda clase de sistemas complejos como sistemas de prevención de volteo de vehículos y sistemas de bolsas de aire; estabilización de imágenes en cámaras, sistemas de navegación; videojuegos, entre muchas otras.

Muchos giroscopios en un circuito integrado pueden entregar señales digitales o analógicas, y en muchos casos un solo empaquetado incluye un sensor para múltiples ejes.

Figura 101. **Giroscopio mecánico**

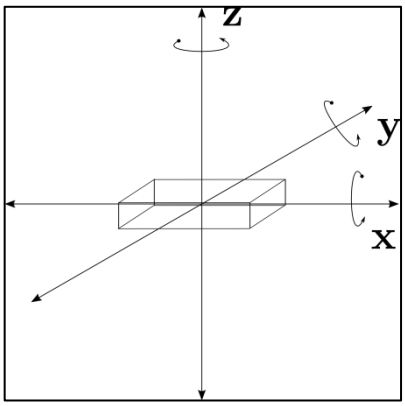


Fuente: elaboración propia, empleando Inkscape.

El modelo de un giroscopio es un tema complicado y requiere de una base matemática más sólida de la que se ha cubierto en este texto. Se indica que tipo de información puede obtenerse de un giroscopio.

En un giroscopio de 3 ejes, cada canal mide la velocidad angular alrededor de un eje en el sistema de coordenadas local fijo al dispositivo. La idea básica puede verse en la figura 102.

Figura 102. **Velocidades con respecto a cada uno de los ejes**



Fuente: elaboración propia, empleando Inkscape.

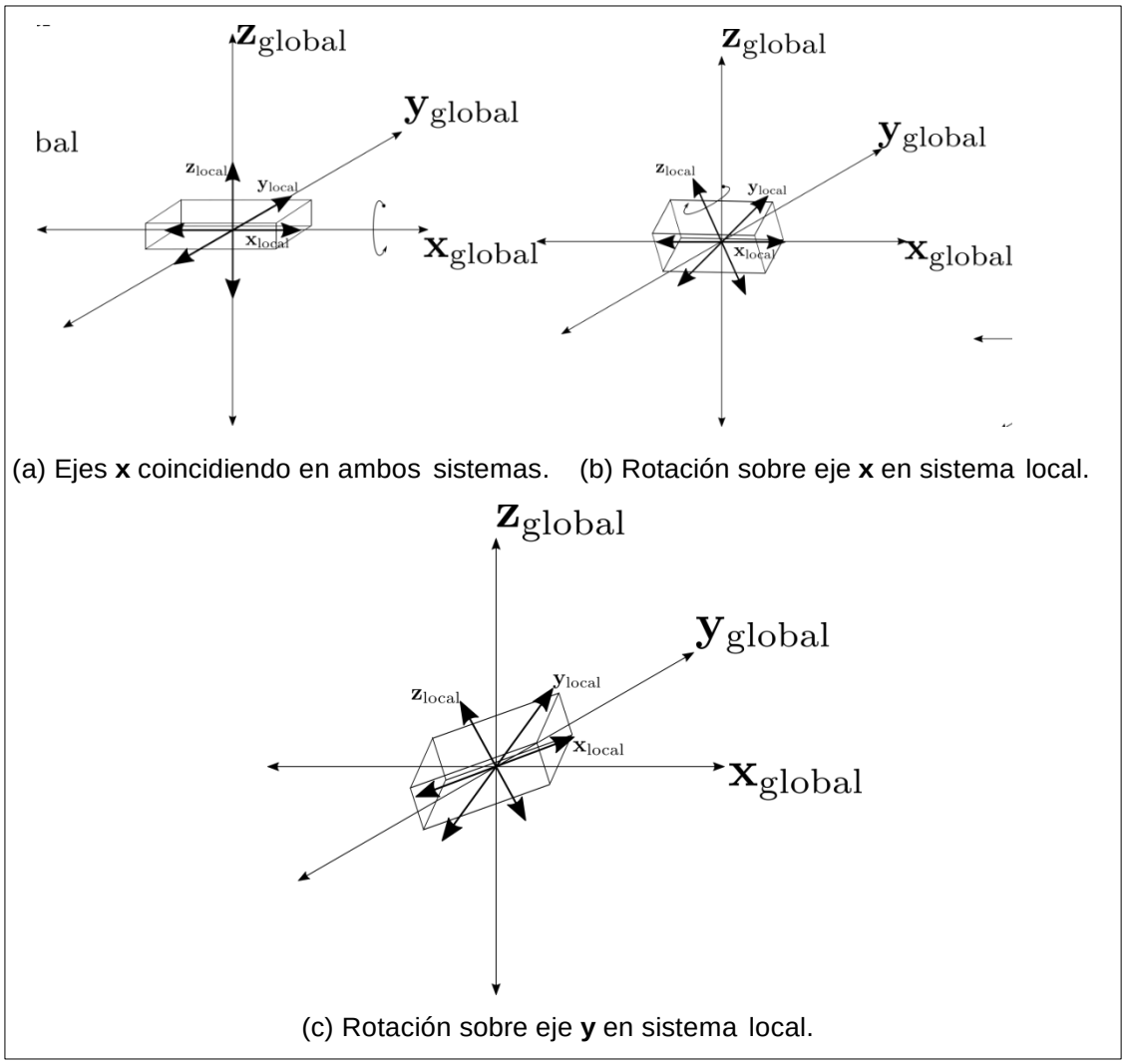
Si se desea obtener la orientación en un sistema de coordenadas globales, en condiciones ideales es posible hacerlo utilizando una expresión equivalente a las ecuaciones 4.10 y 4.11. Por ejemplo, para el ángulo $\theta(t)$ que existe entre un eje del sistema global y local, que cambia a una razón $\omega(t)$ sobre solamente un eje, si inicialmente el ángulo entre ellos es $\theta(0)$ es posible encontrarlo de la siguiente manera:

$$\theta(t) = \theta(0) + \int_0^t \omega(\tau) d\tau \qquad \text{[Ec. 4.13]}$$

Debe quedar claro que el cambio de la rotación se encuentra midiéndose en el sistema local y al tener varios ejes la ecuación anterior dejaría de ser válida. Supongase que al inicio el sistema global y el local coinciden y el dispositivo gira a una razón $\omega(t)$ sobre el eje x, medida por el giroscopio. En este caso la ecuación 4.13 sigue siendo válida porque tanto en el sistema local como en el global siguen coincidiendo. Sin embargo, si se mueve efectuando la rotación sobre otro eje, por ejemplo, el eje z en el sistema local, los ejes x del sistema local y global dejarán de coincidir y la medición del ángulo ya no será

respecto al mismo eje con el que se ha estado midiendo. El sistema de coordenadas local siempre es el mismo para el dispositivo, pero será diferente para un sistema de coordenadas global.

Figura 103. **Rotaciones en los ejes**



(a) Ejes **x** coincidiendo en ambos sistemas. (b) Rotación sobre eje **x** en sistema local.

(c) Rotación sobre eje **y** en sistema local.

Fuente: elaboración propia, empleando Inkscape.

El lector puede comprobar que realizar rotaciones no es un proceso conmutativo, ya que realizar una rotación sobre un eje y luego sobre otro no es equivalente a hacerlo a la inversa.

Para calcular la orientación con respecto a un sistema de coordenadas global, es necesario de hacer uso de algoritmos para refrescar constantemente el sistema de coordenadas local, respecto al global. Esto puede provocar problemas al momento de usarlo en alguna aplicación, el sistema de referencia casi siempre es escogido arbitrariamente para un observador, y puede no coincidir el eje real con el escogido. Por ejemplo, un *quadcopter* cuyo sistema de coordenadas global se ha escogido fijo al suelo; en un terreno rocoso el sistema de coordenadas global real será muy diferente con el que se ha hecho la aplicación ya que no hay forma de saber la posición inicial real del dispositivo.

Además, se debe tener en cuenta que en la práctica nada es ideal y existe *dr*i*ft* (puede ser muy grande, hasta 2 grad/s en algunos sensores) en las mediciones que devuelve el sensor, el cual se irá acumulando en el cálculo de la rotación. Para minimizar el efecto de estos problemas que por la misma naturaleza de un giroscopio ocurren, se debe de acompañar de filtros y otros dispositivos como un acelerómetro.

Al igual que los acelerómetros, los giroscopios pueden comunicarse con otros sistemas haciendo uso de señales digitales o analógicas, para ser procesadas por una unidad de procesamiento y poder así generar resultados interesantes.

4.2.4. *Encoder*

Es un dispositivo que mide posición angular o lineal de un mecanismo, como un motor en el caso de ser angular o una banda en el caso de ser lineal.

El nombre *encoder* es un anglicismo para codificador que es un dispositivo, circuito, programa, transductor, algoritmo o persona que convierte información de un formato a otro. En el caso de un transductor, que es el caso de interés, traduce posición a una señal digital o analógica.

Los codificadores pueden ser utilizados para control de velocidad, posición, como indicador de rotación, generadores de pulsos, servomecanismos, robots, o equipos de medición.

Los codificadores pueden ser etiquetados por el tipo de movimiento que leen para generar información, y básicamente se ven clasificados como codificadores lineales y rotatorios. Por la referencia que llevan de la posición, pueden ser relativos o absolutos; por el tipo de señales que entregan, digitales y analógicos; por el método que utilizan para medir la posición, se puede encontrar una gran gama de dispositivos ópticos, mecánicos, resistivos, magnéticos, entre muchos otros.

- Codificadores rotatorios

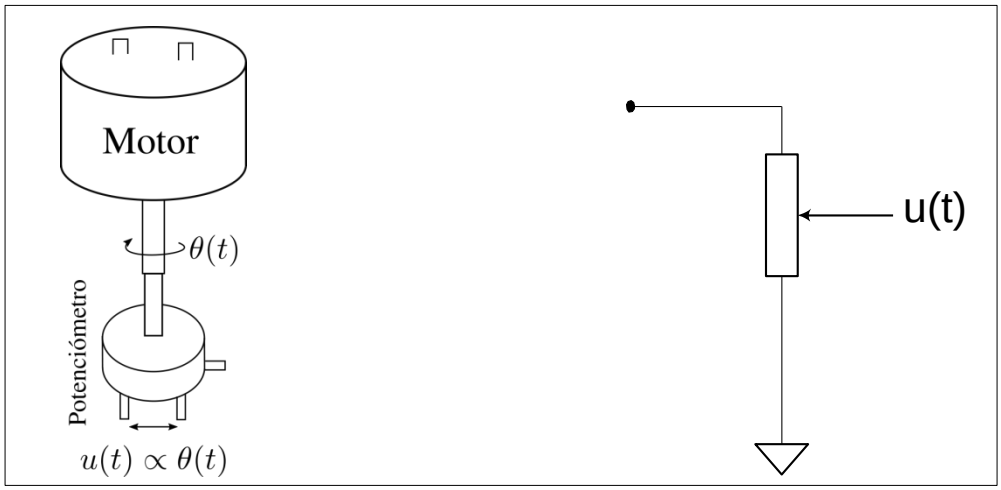
  Es común encontrar codificadores rotatorios (en inglés son conocidos como *shaft encoders* o *rotary encoders*) en sistemas donde se ve involucrado algún tipo de movimiento rotatorio. Se utilizan para encontrar la posición $\mu(t)$ de un objeto girando alrededor de solamente un eje. Los codificadores rotatorios pueden funcionar de muchas formas, sin embargo, encontrar resistivos, magnéticos u ópticos es común. Utilizar un canal de un giroscopio o acelerómetro es posible, sin embargo estos dispositivos tienen niveles de ruido mucho mayores que los mencionados anteriormente y en aplicaciones donde la precisión es muy importante lo mejor es evitarlos.

En la mayoría de los casos la referencia global nunca cambia o simplemente no tiene sentido considerar el sistema de coordenadas en el que se encuentra el dispositivo para efectuar la medición, por lo que resulta un desperdicio el uso de un giroscopio o acelerómetro para movimiento rotatorio sobre un solo eje.

Los codificadores rotatorios pueden ser absolutos o relativos. Un codificador absoluto mantiene la información de la posición aun cuando la energía se remueve del sistema. Una vez encendido el dispositivo la posición se encuentra disponible inmediatamente. La relación entre el valor del codificador y la posición física del movimiento rotatorio se encuentra fija desde el ensamblaje del sistema.

El sistema no necesita regresar a un punto para mantener la exactitud de la posición. El ejemplo más simple de un codificador rotatorio absoluto se puede encontrar en los servomecanismos simples, que utilizan un potenciómetro para llevar control del ángulo, siendo esta simple configuración un codificador rotatorio absoluto analógico resistivo. El voltaje $u(t)$, será directamente proporcional al ángulo $\theta(t)$ motor.

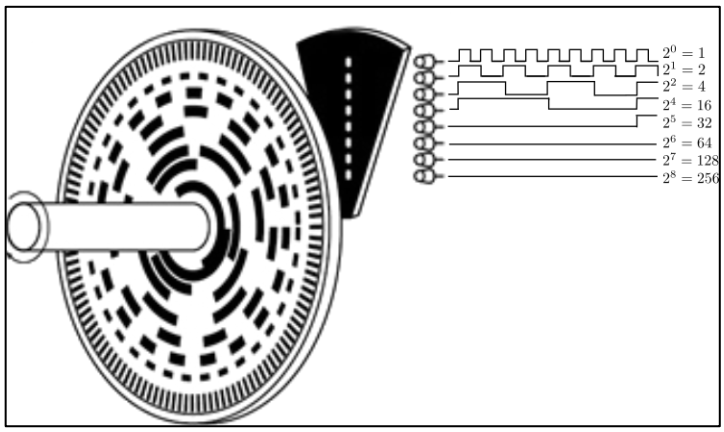
Figura 104. **Potenciómetro como codificador rotatorio absoluto de un motor**



Fuente: elaboración propia, empleando Inkscape.

Algunos codificadores absolutos rotatorios utilizan señales discretas para llevar control de la posición. Estos se componen de varios *n* discos o un solo disco segmentado en *n* anillos, los cuáles representarán un bit en la señal discreta generada, cada anillo divide la circunferencia en partes potencias de dos, el disco o anillo más externo al centro lo divide en 2^n partes binarias, el consecutivo hacia el interior en 2^{n-1} el siguiente en 2^{n-2} hasta que el disco o anillo más interno queda dividido en solamente dos partes. De esta forma, sensores ópticos, magnéticos o mecánicos pueden entregar una señal discreta con 2^n diferentes valores para cada una de las divisiones en el disco más alejado al centro, tal como el que se muestra en la figura 108.

Figura 105. **Codificador rotatorio absoluto discreto**



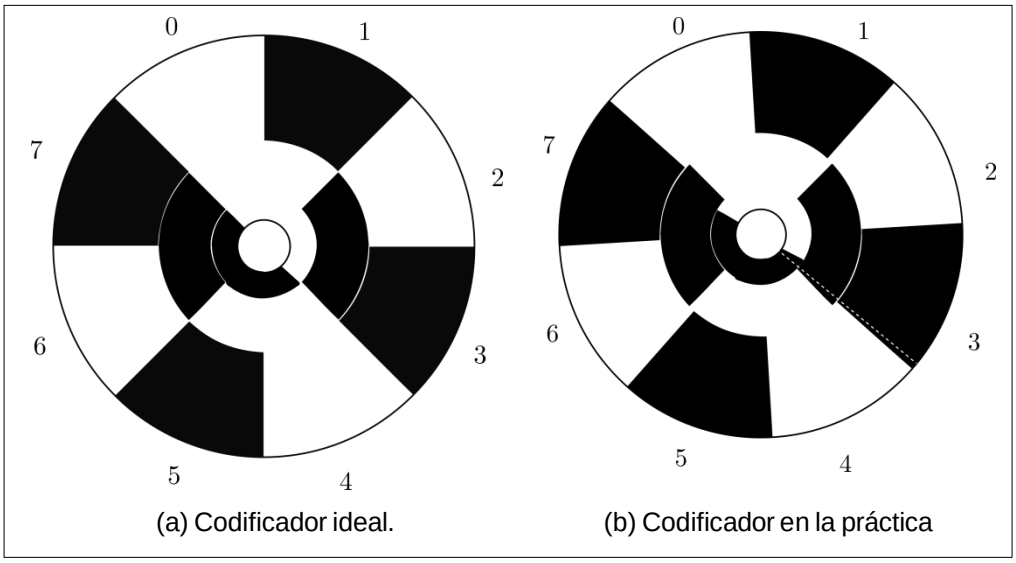
Fuente: *Codificador rotatorio absoluto discreto*. http://www.tamagawa-seiki.com/english/encoder/. Consulta: octubre de 2015.

En la figura 108 se puede ver como con 9 bits es posible llevar un control aproximado de la posición, con una incerteza de 180/512 = 0,3515625 grados. El codificador descrito anteriormente, segmentado en discos con divisiones que son múltiplos de potencias de dos, es conocido como codificador rotatorio absoluto estándar binario, y se muestra uno de 3 bits en la figura 109.

La codificación binaria estándar presenta un problema en la práctica, en la realidad ningún codificador tiene sus discos alineados a la perfección, y será propenso a generar serios problemas. Por ejemplo, si un sistema es capaz de leer muy rápido las señales generadas por el codificador, y los segmentos o los lectores no se encuentran perfectamente alineados y por tanto al cambiar de un estado a otro, las lecturas no ocurrirán al mismo tiempo. Primero se leerá un disco y luego otro al momento de generarse una transición de estado, arrojando valores incorrectos. Por ejemplo, en un codificador mal alineado al pasar de la posición 3 a la 4 se podría dar un desfase entre los discos de tal forma que el primero en leerse sea el disco más interno arrojando un valor de 7, en instantes

saliendo el del disco más externo y arrojando un valor de 6 y luego saliendo del disco medio tomando la señal discreta el valor de 4.

Figura 106. **Codificador rotatorio absoluto estándar binario de 3 bits**

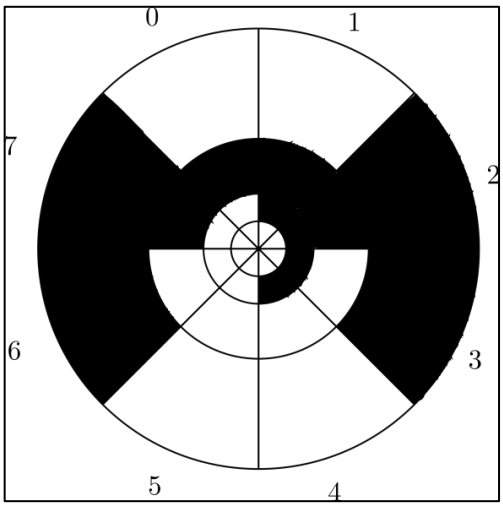


(a) Codificador ideal. (b) Codificador en la práctica

Fuente: elaboración propia, empleando Inkscape.

Entonces los valores 7 y 6 en la señal discreta serían erróneos y sucederían en cada transición de 3 a 4 y de la misma forma sin tener la capacidad de saber si la posición fue bruscamente cambiada o en realidad son esos los valores. Este tipo de problema en algunos sistemas podía generar serias dificultades, incluso dañar sistemas mecánicos, por ejemplo, un brazo mecánico que no puede girar más de 180º al tener las lecturas equivocadas puede realizar una maniobra que lo dañe. Para evitar este problema se utiliza la codificación Gray (en inglés conocida como *Gray encoding*, en honor a Frank Gray), que consiste en generar el cambio de solamente un bit entre dos estados de posición continuos, tal como se muestra en la figura 4.2.

Figura 107. **Codificador Gray rotatorio absoluto**



Fuente: elaboración propia, empleando Inkscape.

Con la codificación Gray el problema que sucedía con la codificación estándar no puede suceder, a pesar de que los discos se encuentren desfasados, debido a que el cambio sucede en un solo bit entonces la transición puede tomar solamente los valores de los dos estados continuos. El problema con ello es que requiere un poco de procesamiento extra para llevar el control de la posición para asignar los valores que corresponden. Dicho problema se puede solucionar en un microcontrolador, utilizando C como compilador, con un vector cuyo valor obtenido por el codificador apunta a la posición real del disco.

Por ejemplo, considérese el caso de utilizar la librería TivaWare para manipular un microcontrolador tm4c123gh6pm cuyo *firmware* necesita conocer la posición actual de un eje girando utilizando el codificador Gray rotatorio absoluto de 3 bits de la figura 110, y la señal generada por el codificador es leída por el puerto GPIO_PORTA del microcontrolador y almacenada en la

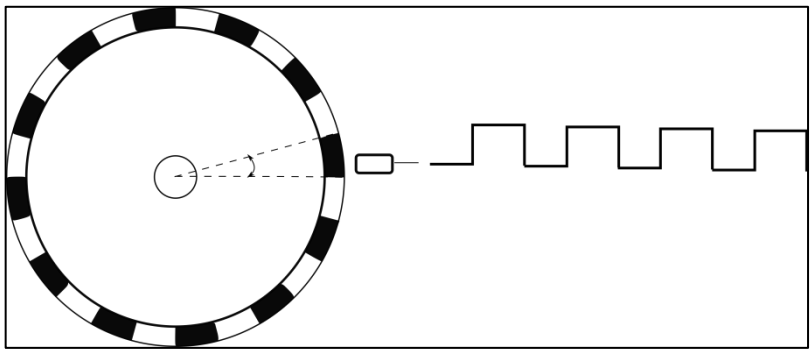
variable codificador. Una variable vector de tipo char, llamada código, realiza la conversión de codificación Gray a estándar; las posiciones del vector código almacenan la posición del motor, y el valor del codificador apunta a la posición del vector. Al recibir cero de la señal discreta significa que el motor se encuentra en la posición 5, al recibir 7 del codificador será porque el motor se encuentra en la posición del motor y así para cada una de las posiciones. Por tanto, si código convierte los valores Gray a estándar código[0]=5, codigo[1]=6, codigo[2]=0 y de esa manera hasta tener todas las posiciones.

```
1 | | char codificador =0;
2 | | char codigo [8]={5 ,6 ,0 ,7 ,4 ,3 ,1 ,2}
3 | | char pos =0;
4 | | ... // Configuraci ón
5 | | pos = codigo [ GPIOPinRead ( GPIO_PORTA_BASE , GPIO_PIN_0 | GPIO_PIN_1 |
      GPIO_PIN_2 )];
```

Se puede ver que incrementar el número de bits en el codificador incrementa la dificultad de realizar la conversión de código Gray a estándar.

Por otro lado, se encuentran los codificadores relativos que a diferencia de los absolutos, no son capaces de sostener una posición fija como referencia. Una versión de ellos es conocida como codificadores incrementales (en inglés se conocen como *incremental shaft encoders*), ya que un disco dividido unifo*r*memente genera una señal discreta binaria, un tren de pulsos con una frecuencia proporcional a la que se encuentra girando el disco, la medición de la posición se estima contando los pulsos de una señal discreta binaria, cada flanco indica que se ha avanzado el ángulo que corresponde a una división. La figura 108 muestra el proceso descrito anteriormente.

Figura 108. **Codificador incremental de 3 fases**

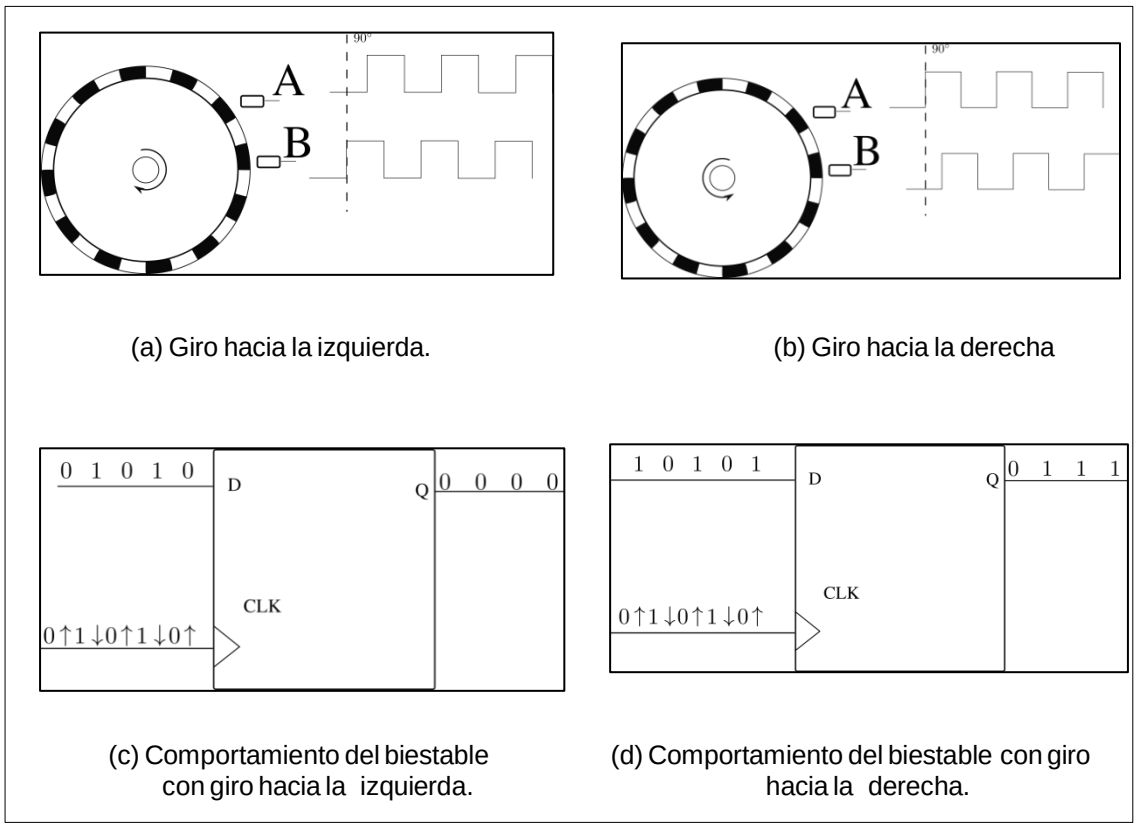


Fuente: elaboración propia, empleando Inkscape.

Los más comunes son los codificadores ópticos, los cuales se componen de un disco con divisiones, una hendidura fija, diodos emisores de luz y fototransistores, entre otros componentes. Un problema con los codificadores incrementales es que solamente contando pulsos es imposible conocer la dirección a la que gira el disco, ya que independientemente de la dirección de giro se generarán los pulsos y si el giro es arbitrario no hay forma de determinar la posición del eje. Para evitar este problema se colocan dos sensores *A* y *B* leyendo las divisiones del disco de tal forma que las señales generadas por ambos se encuentren desfasadas 90º una de la otra. De esa forma puede ser detectada la dirección, tal como se muestra la figura 109.

La forma de detectar la dirección del giro se puede lograr utilizando un circuito biestable (más conocido por su nombre en inglés *flip-flop*) tipo D, el cual es un sistema discreto con 2 entradas discretas binarias, D y CLK y una salida discreta binaria Q. Al tener un flanco positivo en la entrada CLK la salida Q tomará el valor de la entrada D, en caso de no existir flanco en la señal CLK, Q conservará el valor que tiene.

Figura 109. **Codificación incremental en cuadratura**

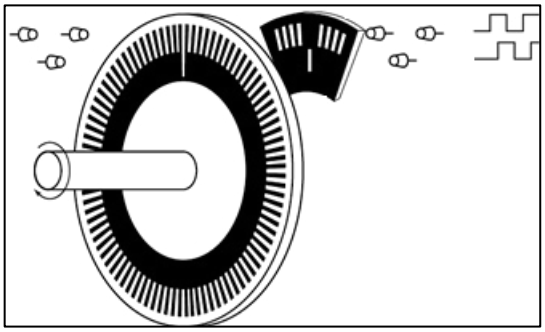


Fuente: elaboración propia, empleando Inkscape.

Un método simple para detectar la dirección de giro, es conectar ya sea A o B a D y a CLK. Como ejemplo considérese que *A* se ha conectado a B a D y a CLK, y cuando *A* o *B* leen una posición oscura tienen se encuentran en alto (1), y cuando leen una posición blanca se encuentran en bajo (0). Al estar los sensores *A* y *B* desfasados uno de otro, considérese el giro hacia la izquierda y que el valor de Q es 0, el primero (en realidad no importa) en leer una posición oscura será el sensor *B* y tendrá un flanco positivo cuando aún *A* se encuentra leyendo la posición blanca contigua por tanto Q toma el valor que se encuentra en D y en este caso será cero (o bajo). Al *A* leer la señal oscura no ocurre flanco

en *B* y por tanto Q conserva el valor anterior, y de esta forma continua hasta que se cambie de giro, ver figura 109.

Figura 110. **Codificador incremental de 3 fases**



Fuente: *Incremental encoder simplified structure*. http://www.tamagawa-seiki.com/english/encoder/. Consulta: octubre de 2015.

¿Qué sucede si el giro cambia de dirección hacia la derecha?, pues la salida Q se encuentra en cero, pero como el giro ha cambiado y los sensores se encuentran desfasados, cuando el sensor *A* pasa de una casilla blanca a una oscura leerá un valor alto, mientras *B* aún se encuentra en blanco, cuando *A* aún se encuentra en la casilla oscura *B* pasa de una casilla blanca a una casilla oscura generando un flanco positivo y Q toma el valor de *A*, el cual por estar en una casilla oscura es uno (alto), tal como se muestra en la figura 109. De esta forma Q se utiliza en los codificadores incrementales en cuadratura, como un bit de dirección que permite a un sistema diferenciar si se desea agregar o quitar una unidad del valor que ha acumulado durante el conteo de los pulsos generados por *A* (o en su defecto el sensor *B*).

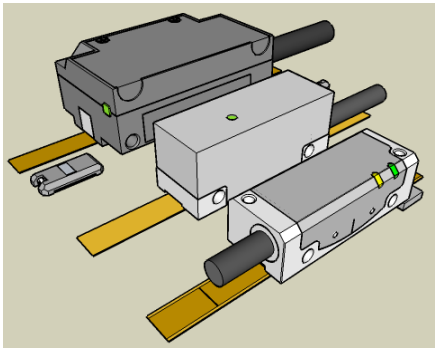
Algunos codificadores poseen una tercera fase, tal como se muestra en la figura 113 la cual permite conocer si el giro ha pasado por una referencia fija, si el codificador deja de funcionar se puede realizar una rutina de *home* para

ubicar la referencia y utilizarla como origen. La ventaja de utilizar codificadores incrementales sobre absolutos es que se puede llevar control de la posición utilizando solamente 3 bits, sin importar la resolución que el sensor utilice. La desventaja son las rutinas de *home* que muchas veces son necesarias incluir en los sistemas para poder llevar un control correcto de la posición.

- Codificadores lineales

  Un codificador lineal es un sensor con una escala que codifica la posición de un objeto desplazándose a lo largo de un eje, y no alrededor como lo hacen los codificadores rotatorios. De la misma forma es posible encontrar absolutos y relativos, analógicos y digitales. Los modos de construcción son diversos y pueden ser magnéticos, ópticos o mecánicos. Un codificador lineal común de encontrar es un eje con códigos marcados a lo largo que un sensor puesto en el objeto del que se desea conocer la posición identifica conforme el objeto recorre el eje. Es posible que se utilice un codificador incremental, similar al rotatorio, utilizando siempre 3 sensores para llevar control de la posición.

Figura 111. **Codificadores lineales**



(a) Codificadores ópticos

(b) Sensor ultrasónico.

Fuente: *MB1010 LV-MaxSonar.*
https://en.wikipedia.org/wiki/Linear_encoder#/media/File:Optical_Encoders.png
(bhttp://www.maxbotix.com/Ultrasonic_Sensors/MB1010.htm. Consulta: octubre de 2015.

Sensores ultrasónicos son frecuentes en situaciones para medir la distancia recorrida con respecto a una posición. El principio básico de estos dispositivos es que utilizan una frecuencia inaudible disparada por un emisor y un receptor, con la ayuda de una serie de filtros que evitan que otras frecuencias sean leídas, mide el tiempo en que tarda la onda reflejada en regresar. Al utilizar estos sensores se debe considerar que en aplicaciones en tiempo real la medida puede tener latencia y al diseñar el sistema que trabaja con esta variable se debe tener en cuenta para evitar comportamientos no deseados.

Los sensores magnéticos también forman una gran parte de la variedad de codificadores lineales, la distancia puede ser medida utilizando bobinas sensoras, efecto Hall o sensores con propiedades magnetoresistivas (propiedad de un material de cambiar su resistencia eléctrica cuando un campo magnético

externo se aplica al material). Existen formas de codificar distancias lineales, algunas de ellas usan la capacitancia o inductancia entre el lector y la escala.

4.2.5. Otros sensores útiles y magnitudes importantes

En la práctica, medir variables del entorno físico de diferentes tipos es necesario para diversos propósitos. Existen muchas otras variables aparte del espacio y tiempo que es necesario medir. Una variable común en muchos sistemas es la temperatura en motores de automóviles, protecciones ante grandes corrientes, procesos químicos industriales o de laboratorio, sistemas de climatización para residencias o piscinas, entre muchas otras aplicaciones. La temperatura es una medida del promedio de energía cinética de las partículas en una unidad de masa, expresada en grados en una escala estándar. Es común utilizar cualquiera de estos tres tipos de sensores para medir temperatura:

- Termopar: (en inglés *thermocouple*) se crea cuando dos metales diferentes se juntan y el punto de contacto produce un pequeño voltaje de circuito abierto como una función de la temperatura. Este voltaje es conocido como voltaje termoeléctrico o voltaje Peltier-Seebeck, el cual para pequeños cambios se comporta de manera lineal. Los termopares son económicos y pueden operar en un amplio rango de temperaturas. El Instituto Nacional Americano de Estándares (ANSI siglas en inglés de American National Standards Institute) asigna una letra mayúscula a cada tipo de termopar para indicar su composición. Por ejemplo, la letra K hace referencia al termopar formado por cromel (aleación de níquel y cromo) y alumel (níquel y aluminio).

- RTD: es un dispositivo de platino, en la mayoría de los casos, hecho de bobinas o películas de metal. Al calentarse, la resistencia del metal aumenta; al enfriarse, la resistencia disminuye. Al pasar un voltaje a través de este dispositivo genera un voltaje que al medirlo, es útil para determinar la resistencia y por tanto la temperatura. La relación entre la resistencia y la temperatura es aproximadamente lineal. Generalmente a 0° los RTD tienen una resistencia de 100 Ω y son capaces de medir temperaturas de 850°.

- Termistores: un termistor es un semiconductor hecho de óxidos de metal que están comprimidos en una sola pieza, disco, oblea u otra forma. Son sometidos a altas temperatura y luego son cubiertos con *epoxi* o vidrio. Al igual que los RTD, es posible pasar una corriente a través de ellos para leer un voltaje y determinar la temperatura. Los termistores tienen una resistencia más alta, entre 2000 Ω y 10 kΩ, su sensibilidad es de aproximadamente 200 Ω/°C, permitiendo alcanzar una alta sensibilidad en un rango de temperaturas menor a 300 °C.

Al diseñar un sistema de medición adecuado para un sensor de temperatura, se debe considerar que las señales de salida desde los sensores están generalmente en el orden de los milivoltios por lo que se debe amplificar la señal producida por ellos para prevenir ruido. Termopares que se montan o son soldados directamente en materiales conductivos como acero o agua introducen ruido en las mediciones, por lo que deben aislarse correctamente. Los RTD y termistores son dispositivos sensibles, se debe administrarles una corriente de excitación y luego leer el voltaje a través de las terminales. Si el calor adicional no se puede disipar, el calentamiento causado por la corriente de excitación puede incrementar la temperatura del elemento arriba de la temperatura ambiente y este calentamiento cambia la resistencia del RTD o

termistor, provocando errores en las medidas, se pueden minimizar los efectos del autocalentamiento del dispositivo al administrar menor corriente de excitación. Además, el filtrado de las señales es importante para limpiarlas de ruidos de alta frecuencia, por ejemplo los 60 Hz que son comunes en laboratorios y plantas. En pocas palabras la señal debe acondicionarse de manera adecuada para poder ser leída correctamente por una unidad de procesamiento.

Figura 112. **Sensores de temperatura**

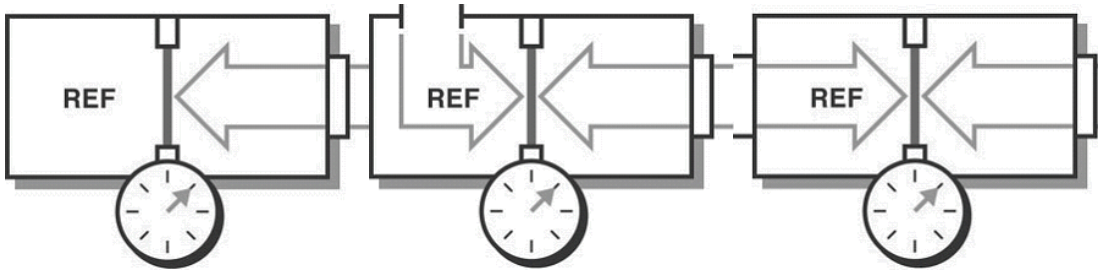
Fuente: elaboración propia, empleando Inkscape.

Una forma común de medir en CPS es la presión, que es la fuerza ejercida por unidad de área. Un cuerpo sumergido en un fluido experimentará presión, debido a la fuerza media que ejercen las moléculas y átomos del fluido golpeando constantemente la superficie del cuerpo sumergido. A mayor profundidad se encuentre el cuerpo dentro del fluido mayor será la presión experimentada. Por ejemplo, el peso del aire genera presión sobre los cuerpos que se encuentran sobre la superficie de la tierra, a mayor altitud menor la presión, de la misma forma en que el peso del agua sobre la superficie de los submarinos ejerce presión, a mayor profundidad que se encuentre el submarino mayor será la presión que ejerce el agua sobre su superficie.

El sistema internacional de medidas establece como medida de presión el Pascal (Pa) que tiene un equivalente a Newton sobre metro cuadrado (N/m^2). Otra medida común son los PSI (siglas en inglés de *pounds per square inch*), también suele medirse en atmósferas (equivalente a 101325 Pa), bars, pulgadas de mercurio (in Hg) o milímetros de mercurio (mm Hg).

Una medida de presión puede ser descrita por el tipo de medición efectuada. Existen tres tipos de mediciones de presión: absoluta, referenciada y diferencial. La presión absoluta es medida en relación al vacío. Dimensionales como PAA (pascal absoluto) o PSIA (*pounds per square inch Absoute*) son comunes al realizar este tipo de medición. La medición referenciada es medida en relación a la presión atmosférica ambiente, de forma similar a la presión absoluta las dimensionales PAG (siglas en inglés de *Pascal gauge*) o PSIG (siglas en inglés de *PSI gauge*) son usadas para hacer referencia a esta medida. La presión diferencial es similar a la presión referenciada, pero en lugar de medirla en relación a la presión ambiente, lo hace en relación a una referencia específica. De forma similar las dimensionales para este tipo de presión sería PAD (Pascal diferencial) y PSID (siglas en inglés de *pounds per inch d*if*ferential*).

Figura 113. **Tipos de mediciones de presión**



(a) Presión Absoluta. (b) Presión Referenciada. (c) Presión Diferencial.

Fuente: *Cómo medir la presión con sensores de presión*. http://www.ni.com/white-paper/3639/en/. Consulta: octubre de 2015.

Dada la gran cantidad de condiciones, rangos, materiales y situaciones para las que la presión puede ser medida, existen diferentes tipos de sensores para medirla. Regularmente la presión es convertida automáticamente a alguna otra forma intermedia, generalmente desplazamiento. El sensor convierte este desplazamiento en una salida eléctrica de corriente o voltaje. Los sensores más comunes de este tipo son los sensores de capacitancia variable, pizoeléctricos y las galgas extensiométricas.

Los sensores más comunes de presión son las galgas extensiométricas, y son capaces de medir cualquiera de los tres tipos de mediciones de presión mencionados con anterioridad. Una galga extensiométrica es un conductor dispuesto de forma que sea sensible a la deformación, lo que provocará cambios en sus dimensiones y por tanto en su resistencia. Cuando un conductor eléctrico es deformado dentro de su límite de elasticidad, tal que no se produzca rotura o deformación permanente, este se volverá más estrecho y alargado. Este hecho incrementa su resistencia eléctrica. Cuando el conductor es comprimido se acorta y ensancha, reduce su resistencia eléctrica. De esta

manera, al medir resistencia eléctrica, puede deducirse la magnitud del esfuerzo aplicado sobre el objeto y por tanto la presión. Son utilizados frecuentemente por sus bajos tiempos de respuesta, alrededor de 1 ms, a los cambios de presión, así como su gran rango de temperaturas de operación.

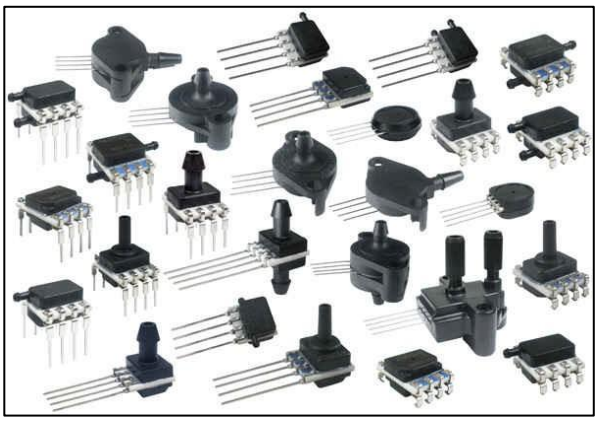
Los sensores de presión de capacitancia variable miden el cambio en la capacitancia entre un diafragma de metal y un plato fijo de metal. La capacitancia entre ambos metales cambia si la distancia entre los metales también cambia. Estos sensores de presión por lo general son muy estables y lineales, pero sensibles a altas temperaturas y más complicados de configurar que otros sensores.

Los sensores de presión pizoeléctricos aprovechan las propiedades naturales de los cristales como el cuarzo. Estos cristales generan una carga eléctrica cuando se encuentran bajo estrés mecánico. Estos sensores no requieren una fuente externa de excitación y son muy robustos. Sin embargo, sí se requiere de circuitería externa para amplificación y son muy susceptibles a las vibraciones.

El artículo *Pressure sensors: stability, sensitivity, accuracy, and other key specifications*[7], escrito por Gina Roos, presenta información útil para el uso correcto de estos sensores.

[7] *Sensores de presión: estabilidad, sensibilidad, exactitud y otras especificaciones clave.* http://www.digikey.com/ en/articles/ techzone/ 2011/dec/pressure-sensors-stability-sensitivit ya ccur acy-and-otherkey- specifications. Consulta: octubre de 2015.

Figura 114. **Sensores de presión**

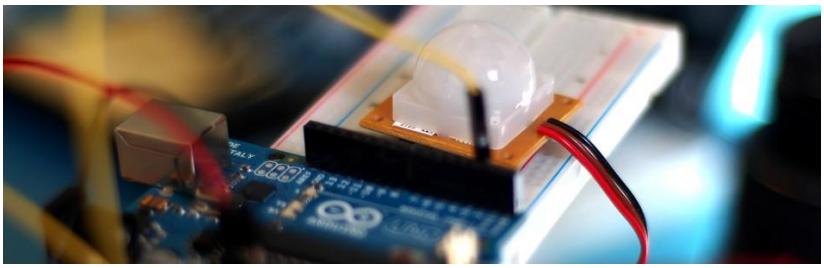


Fuente: *Sensores de presión: estabilidad, sensibilidad, exactitud y otras especi*f*icaciones clave.* http://www.digikey.com/en/articles/techzone/2011/dec/pressure-sensors-stability-sensitivity-accuracy-and-other-key-specifications. Consulta: octubre de 2015.

Existen muchas más magnitudes físicas posibles de medir utilizando sensores, por ejemplo, el sensor DHT11 de la compañía D-Robotics es popular entre los aficionados para medir humedad. Sensores de luz son frecuentemente utilizados en aplicaciones como detección de colores, detección de proximidad, luminiscencia, entre muchos otros. En el mercado se puede encontrar variedad de ellos, desde fotoresistencias, fotodiodos o fotoceldas hasta arreglos complejos de fototransistores.

Algunos otros sensores no leen magnitudes físicas propiamente sino que interpretan cualidades del entorno tal como los sensores de presencia, sensores de nivel de agua o sensores de movimiento.

Figura 115. **Sensor de presencia conectado a un microcontrolador Arduino**



Fuente: *Detección de movimiento con PIR + Arduino*. http://bildr.org/2011/06/pir_arduino/. Consulta: octubre de 2015.

4.3. Algunos actuadores comunes

Es importante reconocer el tipo de actuadores que son útiles para determinada aplicación, además de conocer la técnica para utilizarlos correctamente. En algunos casos, muchos detalles propios de la dinámica física son importantes y deben tomarse en cuenta para un diseño capaz de cumplir con las especificaciones requeridas; el diseñador debe utilizar diversas técnicas para un control correcto de los actuadores.

En muchos otros casos, el diseñador está construyendo un supersistema que utiliza los sistemas de control de los actuadores de forma transparente por lo que los detalles relacionados con la dinámica física y el control resultan transparentes para el diseñador. En estos casos una unidad de procesamiento se encarga de emitir las señales correspondientes hacia el actuador.

4.3.1. Actuadores lumínicos

Posiblemente, los sistemas lumínicos sean los más fáciles de manipular debido a su estrecha relación con sistemas eléctricos, y la forma de hacerlo depende de la tecnología que utiliza el actuador y la potencia necesaria para realizar las tareas requeridas por la aplicación. En algunos casos son utilizados como indicadores y en otros casos se busca manipular la energía entregada al entorno, por ejemplo, una incubadora o una casa automatizada.

Los diodos emisores de luz (led por sus siglas en inglés *light emitting diode*) son los más utilizados como indicadores de algún proceso o actividad que se está llevando a cabo en el sistema, debido a que operan con potencias relativamente bajas y a su alta eficiencia en comparación a otros transductores de energía eléctrica a electromagnética (Por ejemplo, los tubos Nixie). Su sistema de acople resulta muy sencillo para la mayoría de procesadores embebidos cuando se utilizan como indicadores. El sistema más simple se compone de una resistencia que limita la corriente que pasa a través de él. Detallar el funcionamiento de un led escapa del alcance del presente trabajo ya que para ello son necesarios conceptos de mecánica cuántica que no han sido tratados.

Existen otros actuadores lumínicos como lámparas incandescentes, tubos Nixie o lámparas halógenas. Cada uno de ellos con sus formas diferentes de operar y sistemas de acople distintos dependientes de las potencias con las que operan.

Figura 116. **Fotografía de diversos led**

Fuente: *RBG-LED. Jpg*. https://upload.wikimedia.org/wikipedia/commons/c/cb/RBG-LED.jpg. Consulta: septiembre de 2015.

4.3.2. Actuadores hidráulicos

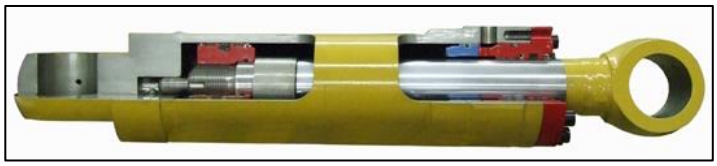
Un actuador hidráulico consiste de una estructura que utiliza líquidos para realizar una acción mecánica. El movimiento mecánico obtenido con estos actuadores puede ya sea lineal, rotatorio u oscilatorio. Ya que los líquidos resultan casi imposibles de comprimir, un actuador hidráulico puede ejercer una fuerza considerable, sin embargo, a cambio de tener respuestas rápidas a los cambios y suelen ser lentos.

El actuador hidráulico más simple que existe es el cilindro hidráulico el cual consiste de una recámara cilíndrica donde un pistón puede deslizarse. Los cilindros hidráulicos obtienen su potencia de fluidos hidráulicos presurizados, aplicados al pistón con el fin que pueda movilizarse en la recámara de un lado hacia otro.

Es posible al tratar con actuadores hidráulicos encontrar los términos simple y doble efecto. En el cilindro simple efecto la presión del fluido se aplica

en un solo lado del pistón y puede moverse solo en una dirección; un resorte es frecuentemente utilizado para dar regresar el pistón a su posición original. El cilindro doble efecto funciona aplicando presión de los líquidos en ambos lados del pistón; cualquier diferencia en la presión entre ambos lados generará movimiento del pistón de un lado hacia otro.

Figura 117. **Cilindro hidráulico**



Fuente: *Cilindro hidráulicos*. https://upload.wikimedia.org/wikipedia/en/4/47/Cutawaywe ldedcylinder544x123.jpg. Consulta: septiembre de 2015.

4.3.3. Actuadores neumáticos

Los actuadores neumáticos utilizan aire comprimido para realizar alguna acción mecánica. El movimiento puede ser lineal o rotatorio, dependiendo del tipo de actuador. Se busca utilizar actuadores neumáticos en aplicaciones donde se necesitan respuestas rápidas a cambios, ya que la fuente de potencia no necesita ser almacenada en reserva para la operación. Los actuadores neumáticos permiten generar grandes fuerzas desde cambios relativamente pequeños de presión. Estas fuerzas son frecuentemente utilizadas con válvulas para mover diafragmas para afectar el flujo del aire comprimido que pasa a través de la válvula.

Algunos actuadores neumáticos comunes en aplicaciones son:

- Actuadores rotatorios
- Cilindros simples y doble efecto
- Agarraderas o garras sujetadoras
- Músculos artificiales neumáticos
- Motores neumáticos
- Generadores de vacío
- Actuadores especiales con movimientos lineales y rotatorios

El aire se encuentra prácticamente en cualquier lugar en cantidades ilimitadas, puede ser transportado por tuberías incluso a través de largas distancias, por lo que un sistema neumático puede ser ubicado en cualquier parte. El aire comprimido puede ser almacenado en reservas y removido cuando se desee. El aire comprimido es relativamente insensible a fluctuaciones de temperatura, lo que asegura el funcionamiento incluso en condiciones extremas. Es un medio seguro ya que el aire comprimido no puede causar explosiones o incendios. No genera mayor contaminación, y además sus componentes son de fácil construcción, lo que las hace relativamente baratas, sin mencionar que pueden operar a altas velocidades mecánicas. Las herramientas neumáticas son seguras ante las sobrecargas.

A pesar de sus numerosas ventajas presentan algunos inconvenientes. El aire comprimido requiere una buena preparación, suciedad y condensaciones no pueden estar presentes en los mecanismos. Es irregular y no siempre es posible obtener velocidades unif*or*mes en los pistones. El aire comprimido resulta energéticamente económico solamente para cierta cantidad de fuerza requerida, bajo condiciones normales de trabajo a una presión entre 600kPa. a 700kPa. se obtienen como resultado fuerzas entre 40 000 y 50 000 Newtons. El

aire expulsado genera ambientes ruidosos, aunque este problema ha mermado considerablemente en los últimos años debido a la construcción de materiales absorbentes de sonido y silenciadores.

El control de estos actuadores en sistemas físicocibernéticos es llevado a cabo mediante válvulas accionadas eléctricamente conocidas como electroválvulas, las cuales manipulan el flujo de aire a través de ella dependiendo de los impulsos eléctricos que reciban por otro sistema.

4.3.4. Motores eléctricos

Un actuador eléctrico convierte energía eléctrica en energía mecánica. Se basan principalmente en la acción de campos magnéticos generados por la energía eléctrica. El actuador eléctrico más común es el motor eléctrico, el cuál puede funcionar con corriente alterna o continua.

Bajo el principio de Faraday, un motor es capaz de generar movimiento utilizando campo magnético y corriente eléctrica. El diseño de un motor eléctrico esta basado en el posicionamiento de conductores en un campo magnético y hacer circular a través de ellos corriente. Un embobinado tiene muchos conductores, o vueltas de cable, y la contribución individual de cada vuelta incrementa la intensidad de las fuerzas actuando.

La fuerza desarrollada de un embobinado depende de la corriente que pasa a través de él y la fuerza del campo magnético. Mientras mayor corriente atraviesa el embobinado, mayor torsión es obtenida. Básicamente un motor eléctrico se compone de una parte giratoria llamada rotor y una parte fija llamada estator. El campo magnético proveniente del rotor y el campo magnético del estator interaccionando entre ellos generan movimiento, siendo

este el principio básico del funcionamiento de un motor ya sea corriente alterna o continua la que atraviesa el embobinado. Se recomienda la serie de videos referentes a motores eléctricos del sitio http://www.learnengineering.org/ para una mejor comprensión del funcionamiento de estas máquinas.

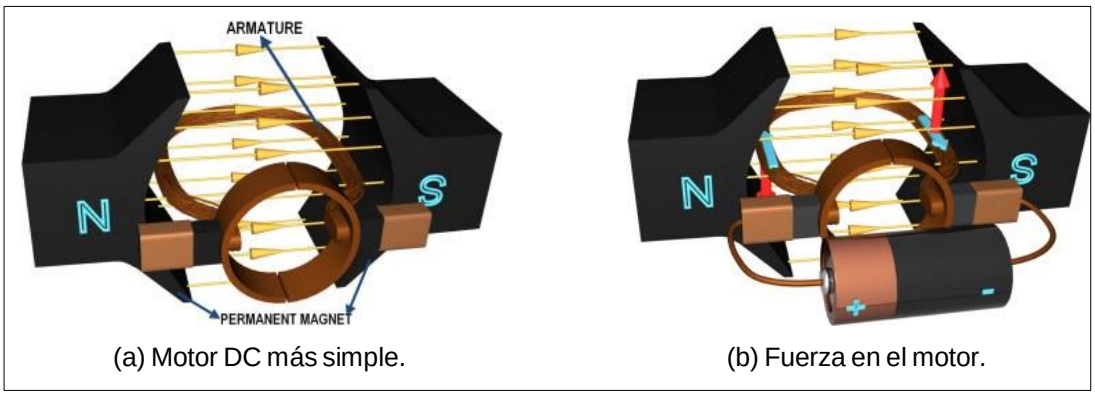
- Motor de corriente directa

Es una de las primeras máquinas eléctricas concebidas para transformar energía eléctrica en mecánica. La estructura del motor de corriente directa es relativamente simple, véase la figura 118, el estator es un imán permanente y la armadura, la cual es la parte giratoria, es un embobinado simple.

La armadura se conecta a una fuente de poder de corriente continua, a través de un anillo computador compuesto de dos semicircunferencias aisladas. Cuando la corriente fluye a través del embobinado una fuerza electromagnética es inducida de acuerdo a la ley de Lenz, entonces el embobinado girará, producto de la fuerza generada (línea roja en la figura 118) de acuerdo a ley de Lenz.

Conforme la armadura gira, el anillo conmutador cambia la polaridad del embobinado intercambiando los polos de la fuente de corriente directa, asegurando que las fuerzas en los lados del embobinado hagan siempre girar la armadura hacia el mismo lado, por lo que se conservará la dirección del vector de torsión.

Figura 118. **Motor DC principio básico I**

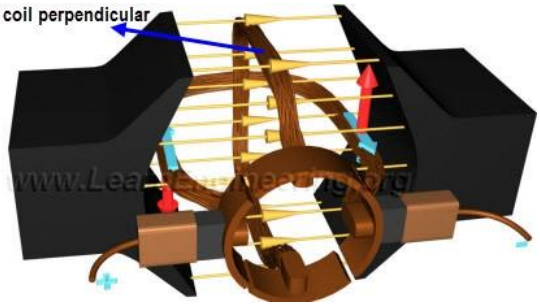


(a) Motor DC más simple. (b) Fuerza en el motor.

Fuente: *Motor de CC*. http://www.learnengineering.org/2014/09/DC-motor-Working.html. Consulta: octubre de 2015.

- Diferentes ángulos harán cambiar la magnitud del vector de torsión sobre la armadura, por ejemplo, si se encuentra totalmente horizontal la armadura experimentará la torsión máxima, por el contrario si la armadura se encuentra totalmente vertical la torsión en esa posición será cero. Para alivianar estos efectos se colocan varios embobinados desfasados, de tal forma que, si uno de ellos experimenta torsión cero, otro experimentará una torsión diferente haciendo el giro más suave.

Figura 119. **Motor DC principio básico II**



(a) Dos embobinados.

(b) Mejora de torsión con incremento de embobinados y disminuciónde reluctancia.

Fuente: *Motor de CC*. http://www.learnengineering.org/2014/09/DC-motor-Working.html. Consulta: octubre de 2015.

En la práctica, el embobinado de la armadura, en lugar de hacerse sobre bucles de aire, como en las figuras mostradas, se hace adentro de ranuras formadas por capas de metal altamente permeable. Y de esta forma mejora la interacción del flujo magnético. Por lo general trozos de grafito con un resorte (llamados de forma coloquial carbones), ayudan a mantener constante el contacto entre la fuente de poder y el anillo conmutador.

En la práctica los imanes permanentes son usados solamente para motores DC pequeños, la mayoría hacen uso de un embobinado en el estator para crear un electroimán y así generar el campo magnético en lugar de hacerlo con imanes permanentes. Los embobinados del estator y del rotor pueden

colocarse en serie o paralelo; se tienen diferentes resultados para cada uno. Los motores cuyos embobinados están en serie presentan una buena torsión inicial, pero la velocidad decae drásticamente al colocar una carga. Un motor en paralelo tiene una torsión inicial muy baja, pero puede girar a velocidad constante, a diferentes cargas en el motor.

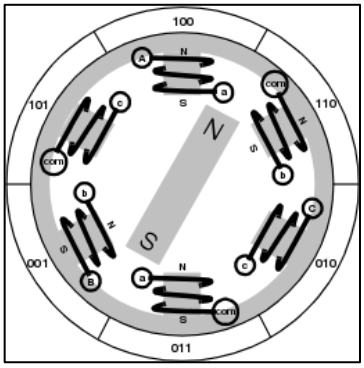
A diferencia de muchas otras máquinas eléctricas de corriente directa, los motores tienen la característica de producir una fuerza contra electromotriz. Un bucle girando en un campo magnético produce una fuerza electromotriz de acuerdo al principio de inducción de Faraday. La armadura giratoria introducirá una fuerza electromotriz que se opondrá al voltaje de entrada. La fuerza contra electromotriz disminuye la corriente de la armadura en un gran porcentaje y es proporcional a la velocidad del motor. Por lo que al principio es muy pequeña y puede darse una sobre corriente que queme el embobinado en motores CD muy grandes. Para evitar esto es recomendable un mecanismo que controle el voltaje de entrada al motor, que es conocido como rampa.

- Motores sin escobillas (*brushless*)

Su objetivo es hacer más eficientes y silencios los motores, y evitar pérdidas ocasionadas por la fricción entre el anillo conmutador y la fuente de poder. Las pérdidas generadas por fricción provocan un rápido deterioro del motor y lo hacen gastar más energía. Un motor sin escobillas, será más eficiente y silencioso.

El rotor en un motor *brushless* es un imán permanente. El estator tiene un arreglo de hilo conductor, tal como se muestra en la figura 120.

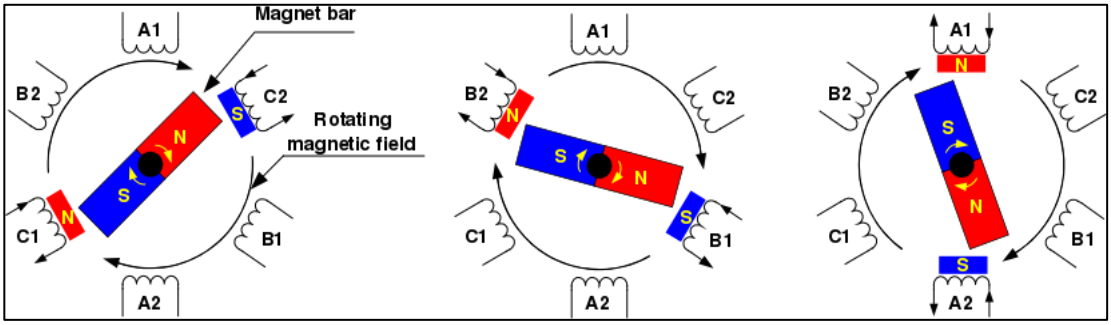
Figura 120. **Rotor es un imán permanente**



Fuente: *Brushless DCMotor ControlMade Easy*.Ward Brown, Microchip Technology Inc. Consulta: octubre de 2015.

Pasar corriente a través de una bobina generará un campo magnético. El secreto detrás de los motores *brushless* es ir pasando corriente a través de las bobinas de forma tal que puedan atraer y repeler al imán permanente que conforma el rotor para que este gire, tal como se muestra en la figura 121. En la figura, las bobinas C1 y C2 se activan de forma que C2 pueda atraer al polo norte del imán permanente y C1 al polo sur. Luego las bobinas B1 y B2, atrayendo al polo norte y sur respectivamente. Y por último las bobinas A1 y A2, al hacerlo el ciclo se invierte C1 atrae al polo norte y C2 al sur, y así para cada una de las bobinas, haciendo al rotor dar una vuelta completa.

Figura 121. **Motor *brushless* girando**

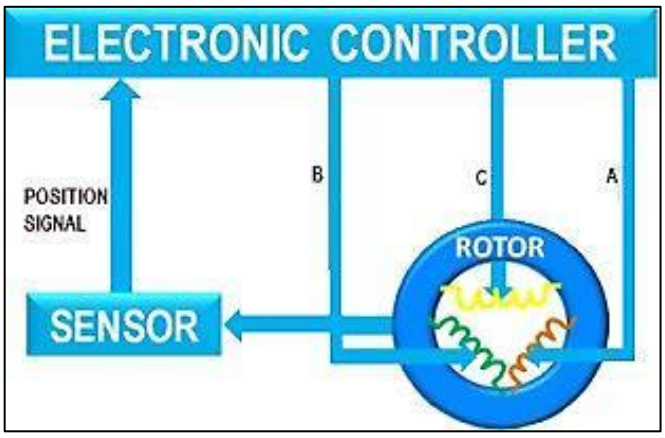


Fuente: ZHAO, Jian; YU, Yangwei. *Brushless DCMotor Fundamentals, Application Note-AN047*. https://www.monolithicpower.com/Portals/0/Documents/Products/ Documents/ appnotes/Brushless%20DC%20Motor%20Fundamentals.pdf. Consulta: octubre de 2015.

Es posible mejorar el comportamiento del motor haciendo que las bobinas continuas repelan los polos del imán permanente. Por ejemplo, cuando C1 atrae al polo sur y C2 al polo norte, se puede hacer que A2 repela al polo sur y A1 al polo norte haciendo impregnando una mayor torsión en el motor. Para esta configuración las bobinas necesariamente deben ser energizadas independientemente, aunque si se coloca en A1, B1 y C1 el mismo embobinado, pero en un sentido contrario en A2, B2 y C2 se logra el mismo efecto, solamente energizando una bobina.

Para lograr la secuencia descrita es necesario un control que vaya energizando las bobinas de acuerdo al paso del rotor. Por lo general, un sensor de efecto Hall determina la posición del rotor y en base a esta información un controlador electrónico decide que bobina energizar.

Figura 122. **Motor *brushless***



Fuente: *El uso de una ECU*.
http://www.learnengineering.org/2014/10/Brushless-DC-motor.html Consulta: octubre de 2015.

- Motores paso a paso (*stepper*)

Los motores paso a paso son dispositivos electromecánicos que convierten pulsos eléctricos en movimientos mecánicos discretos. El eje de un motor paso a paso gira en incrementos discretos cuando un conjunto de pulsos es aplicado al motor en la secuencia adecuada. Su forma de funcionar, visto muy superficialmente, es similar a la del motor *brushless* con la diferencia que energizar las bobinas se lleva a cabo de forma externa al motor.

La velocidad y dirección de la rotación del motor depende directamente de la forma en que sean entregados los pulsos al motor. La secuencia de los pulsos determina la dirección que lleva un motor paso a paso. La frecuencia con que son entregados los pulsos determina la velocidad de la rotación y el número de pulsos determina el ángulo recorrido total por el rotor.

Un motor paso a paso provee un posicionamiento preciso, donde el error no es acumulativo conforme se encuentra funcionando. Tienen una buena respuesta al iniciarse, detenerse o cambiar de giro. Ya que los motores paso a paso responden a una entrada digital formada por pulsos provee un control de lazo abierto, haciendo al motor fácil y barato de controlar solamente llevando la pista de los impulsos entregados. En estos motores es posible alcanzar un rango amplio de velocidades ya que la velocidad depende de la frecuencia de los pulsos. El problema de estos motores es que puede haber resonancia al no controlarlos correctamente y son difíciles de controlar a altas velocidades.

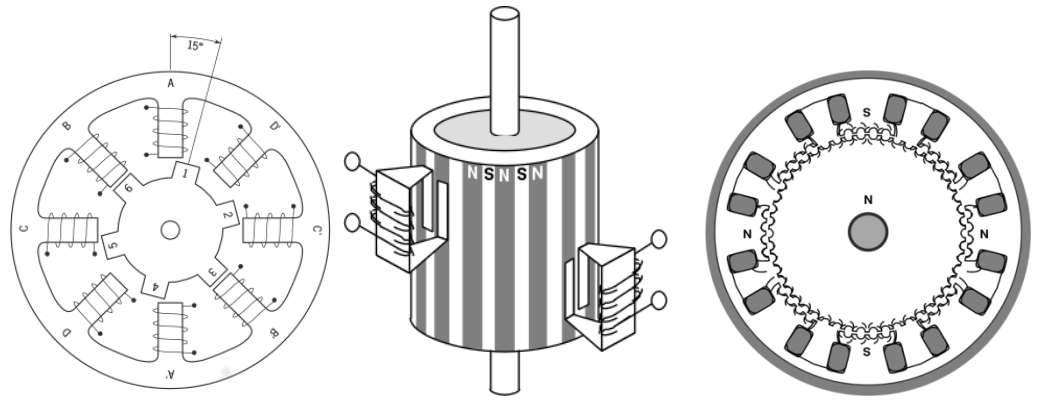
A continuación, se mencionan tres tipos de motores paso a paso:

- Motores de reluctancia variable: estructuralmente este motor es el más simple de comprender y es muy similar al motor sin escobillas. Consiste de un rotor de acero ligero formado por muchos dientes, y un estator formado por ranuras, tal como se muestra en la figura 126. El giro ocurre cuando los dientes en el rotor se ven atraídos a los polos energizados en el estator.

- Motores de imanes permanentes: frecuentemente llamados motores de lata, en su estructura tiene imanes permanentes y no posee dientes como el motor de reluctancia variable. En su lugar el rotor se encuentra magnetizado con polos alternantes entre norte y sur colocados en líneas paralelas al eje del motor. Los polos magnéticos del rotor proveen una mayor intensidad de flujo magnético y por esto los motores de imanes permanentes poseen una mayor torsión que los de reluctancia variable.

- Motores híbridos: el motor paso a paso híbrido es más caro que un motor de imanes permanentes, pero provee un mejor desempeño, con respecto a la resolución, velocidad y torsión. El rotor posee varios dientes, como un motor de reluctancia variable y contiene un imán concéntrico al eje principal magnetizado axialmente. Los dientes en el rotor ofrecen un mejor camino para el flujo magnético en determinadas posiciones. De esta forma se incrementa la retención y la torsión dinámica.

Figura 123. **Tipos de motores paso a paso**



(a) Motor de reluctancia variable. (b) Motor de imanes permanentes. (c) Motor Híbrido.

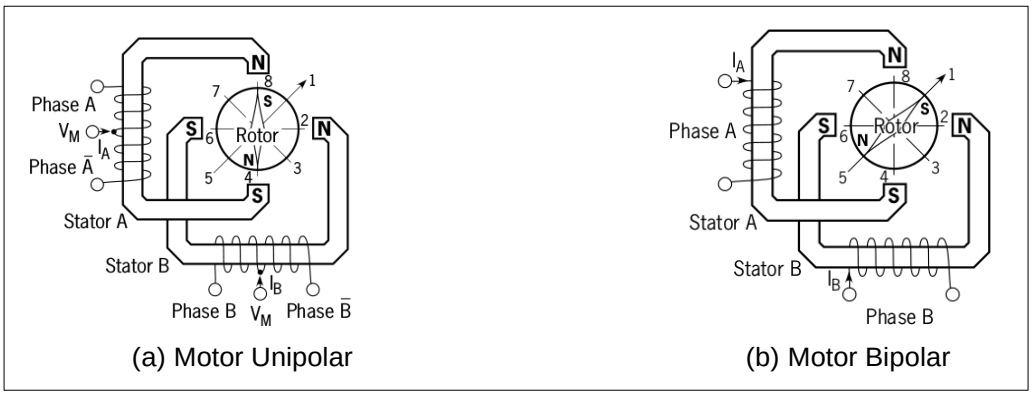
Fuente: *StepperMotor Basics, Industrial Circuits Application Note*. http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf. Consulta: octubre de 2015.

Un motor paso a paso es una buena elección cuando la aplicación requiere control del ángulo, velocidad, posición o sincronía. Debido a estas ventajas los motores *stepper* son comunes en aplicaciones como impresoras, equipo de oficina, discos duros, equipo médico, dispositivos fax, entre otros.

La torsión generada por un motor paso a paso depende de la frecuencia de los pasos, la corriente entregada a las bobinas y el diseño o tipo del controlador. En un motor paso a paso la torsión se desarrolla cuando el flujo magnético del rotor y estator se desplazan de uno a otro. El estator se encuentra construido de un metal muy permeable, la presencia de este material encausa el flujo magnético en mayor parte hacia caminos definidos por la estructura del estator, de la misma forma que un conductor eléctrico lo hace con la corriente eléctrica. De esta forma se concentra el flujo en los polos del estator. La torsión producida en el motor es proporcional a la intensidad de flujo magnético generado cuando se hace pasar corriente a través de las bobinas.

Usualmente los motores paso a paso poseen dos fases, pero existen algunos con 3 o 5 fases. Los de dos fases se pueden encontrar en dos modalidades: bipolares y unipolares. Los motores bipolares poseen dos fases con un embobinado por fase, y los unipolares de igual forma poseen dos fases, pero con un punto de unión central para ambas (*tap* central). Algunas veces los unipolares se conocen como motores de 4 fases, a pesar de poseer solo dos.

Figura 124. **Motores *stepper* de dos fases**



(a) Motor Unipolar (b) Motor Bipolar

Fuente: *StepperMotor Basics*, *Industrial Circuits Application Note*. http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf. Consulta: octubre de 2015.

Un polo se puede definir como una región de un cuerpo magnetizado donde el flujo magnético es concentrado. La figura 124 muestra motores paso a paso teniendo 2 polos, uno por fase en el estator, y uno por fase en el rotor. En la práctica, múltiples polos pueden agregarse tanto al estator como al rotor con el fin de incrementar el número de pasos que le toma al motor para dar una vuelta completa, con el fin de proveer una mejor resolución.

Para manipular el giro de un motor *stepper*, en referencia a la figura 124, es posible utilizar uno de los siguientes modos:

- Control de onda: también llamado de una fase, en este modo de control solamente un embobinado es energizado en un instante dado. El estator se energiza de acuerdo a la siguiente secuencia: $A \rightarrow B \rightarrow \bar{A} \rightarrow \bar{B}$ y el rotor se moverá a pausas de la posición $8 \rightarrow 2 \rightarrow 4 \rightarrow 6$. Motores unipolares y bipolares con la misma configuración en el embobinado las secuencias en la posición serán siempre las mismas. Sin embargo, este modo ofrece la desventaja que en un motor bipolar se utiliza solo el 50 % del embobinado y en uno unipolar solo el 25 %, restringiendo así al motor de alcanzar su máxima torsión posible.

- Control de paso completo: también llamado de dos fases, en este modo de control dos fases son energizadas al mismo tiempo en un instante dado. Para hacerlo se lleva a cabo la secuencia $A \rightarrow B \rightarrow \bar{A} \rightarrow \bar{B} \rightarrow \overline{AB}$, generando que el rotor se mueva en la siguiente secuencia: $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$. El modo de control de paso completo genera el mismo movimiento angular que el modo de control de onda, pero simplemente desfasado la mitad de un paso completo. Los motores unipolares generan menos torsión que los bipolares (si ambos poseen los mismos parámetros) ya

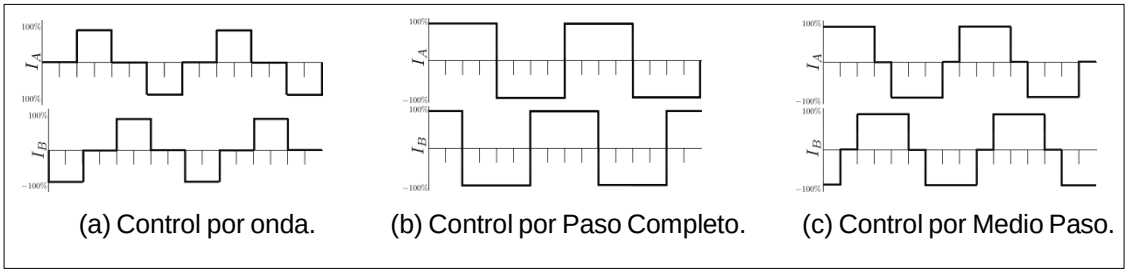
que solamente utilizan el 50 % de su embobinado en un instante dado a diferencia de un motor bipolar que utiliza el 100 %.

- Control de medio paso: también llamado de una y dos fases, en este modo se combinan los dos modos de control anteriores. Cada segundo paso de un ciclo solo una fase se activa y para los siguientes pasos una fase en cada estator. El estator se activa en la siguiente secuencia: $A \rightarrow B \rightarrow \bar{A} \rightarrow \bar{B} \rightarrow \overline{AB} \rightarrow A$. Provocando que el rotor se mueva en la siguiente secuencia de posiciones: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$. Mejorando la resolución del paso en el motor paso a paso.

- Control por micropaso: este control es una técnica de mover el flujo del estator de un *stepper* de forma más suave que en los modos de control de medio paso y paso completo. El resultado es un movimiento con menores vibraciones, menos ruidoso, pasos más pequeños y mejor posicionamiento. Un motor es síncrono lo que significa que la posición de detención estable del rotor se encuentra en sincronía con el flujo magnético en el estator. El rotor se hace rotar rotando el flujo magnético en el estator, haciendo que el rotor se mueva a la siguiente posición de detención estable. Cuando un motor paso a paso es controlado por paso completo o por medio paso el flujo del estator se encuentra girando 90° y 45° con respecto a cada paso en el motor.

Para comprender la forma que funciona el control por micropaso véase la figura 125, donde se muestra el comportamiento en el tiempo de las corrientes I_A y I_B en los embobinados *A* y *B* del estator, al aplicar las secuencias descritas con anterioridad para un motor bipolar.

Figura 125. **Corrientes I_A y I_a del embobinado en el estator**

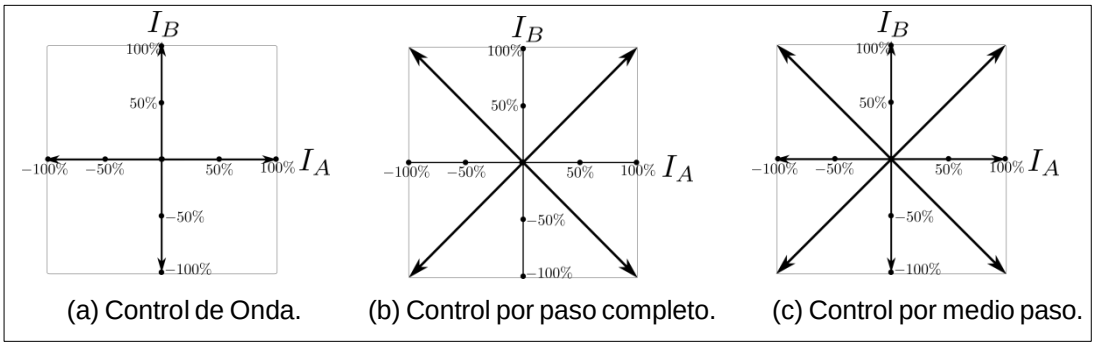


(a) Control por onda. (b) Control por Paso Completo. (c) Control por Medio Paso.

Fuente: elaboración propia, empleando Inkscape.

Las figuras anteriores pueden verse como ondas periódicas tomando valores, de acuerdo a un comprendido ángulo entre 0 y 2π. Como puede verse, las corrientes de *A* y *B* experimentan un desfase entre ellas. El comportamiento de ambas corrientes puede verse en un diagrama de fases, donde el eje de las abscisas representa la corriente I_A y el eje de las ordenadas, la corriente I_B. Y el ángulo entre la abscisa y el fasor representa el comportamiento de ambas corrientes en referencia al ángulo de la onda.

Figura 126. **Diagrama de fase**



(a) Control de Onda. (b) Control por paso completo. (c) Control por medio paso.

Fuente: elaboración propia, empleando Inkscape.

El ángulo en estos diagramas de fase representa el paso, por ejemplo para el control de paso completo, una secuencia de 4 pasos debe efectuarse para completar un ciclo, por lo que la circunferencia en los diagramas estará partida solamente en 4 ángulos diferentes. En el control por onda también deben efectuarse 4 pasos para completar un ciclo. En el caso del control por medio paso son necesarios 8 pasos para completar un ciclo, por lo que se tendrá dividida la circunferencia en 8 pasos mejorando así la resolución. Debe tenerse en cuenta que es el ángulo del fasor en los diagramas de fase el que determina la posición del motor, no así la magnitud, la cual esta relacionada al consumo de potencia y torsión generada del motor.

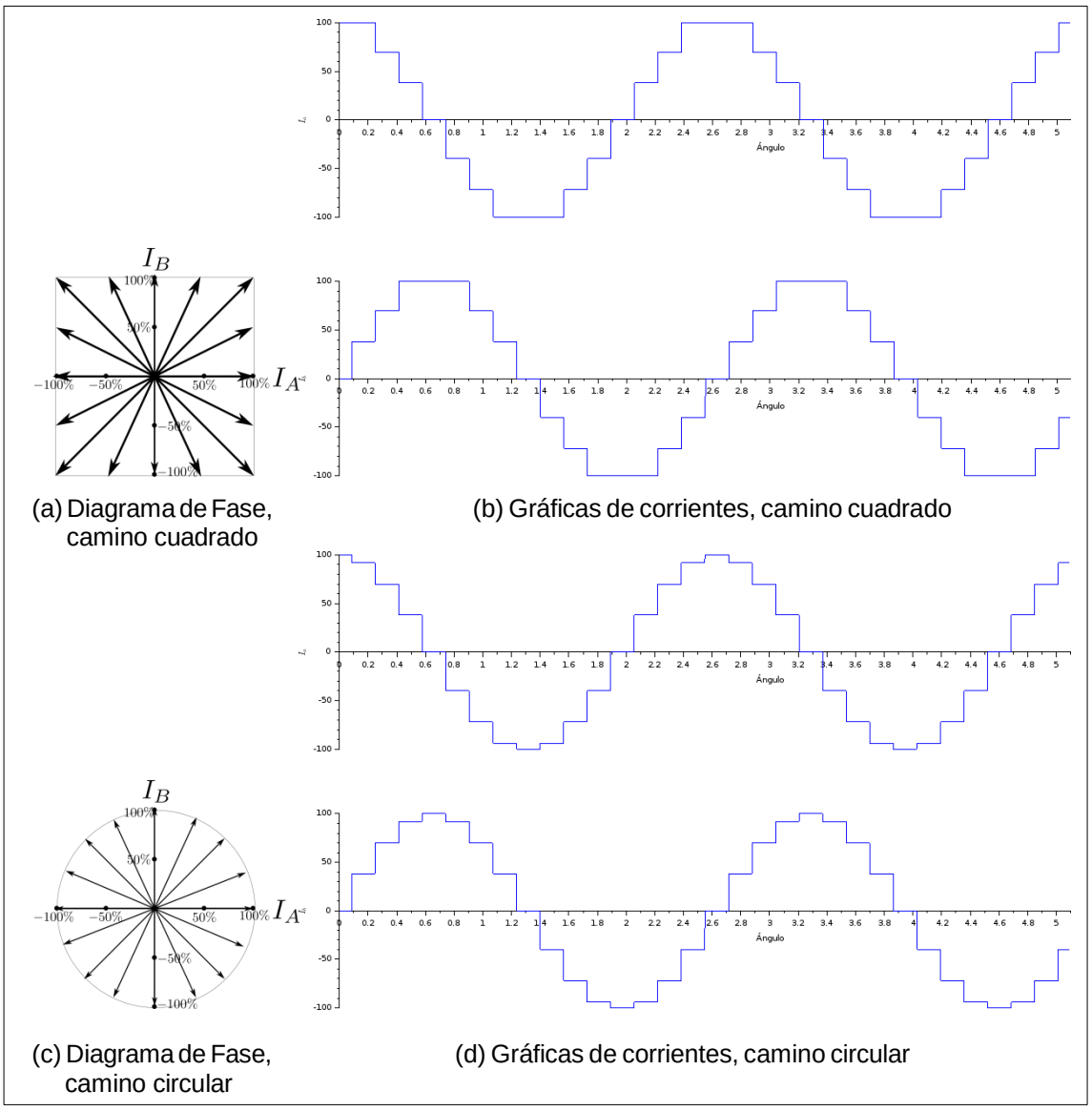
La idea básica detrás del control por micropaso es llevar el diagrama de fases a la implementación práctica. Si el controlador se encuentra diseñado con la capacidad de variar la magnitud de la corriente, el control por micropaso puede ser efectuado. Para ello se divide de manera regular un ciclo completo y se asignan las magnitudes de acuerdo a la torsión requerida por la aplicación, el número de niveles de cuantización deben ser los necesarios para cubrir todas los posibles valores que pueden tomar las corriente I_A y I_B. Este proceso puede resultar en tres tipos de control por micropaso, acorde a la magnitud asignada para cada sector dividido por los fasores representando al micropaso.

- Camino cuadrado: este método da la mayor torsión pico si se encuentra limitado por una fuente de voltaje. Su diagrama de fase consiste limitar al máximo de las corrientes que es posible entregar para determinado ángulo. La figura 127, muestra el diagrama de fase para un control micropaso para camino cuadrado de 16 micropasos por ciclo completo. Como resultado se obtienen las ondas de corriente para I_A y I_B que se muestran en la figura 127.

- Camino circular: este método es referido como método de seno y coseno, debido que las corrientes I_A, I_B son las paramétricas de una circunferencia. Es el método más común en *microstepping*, y a diferencia del método de camino cuadrado, es llamado circular ya que todos los fasores poseen la misma magnitud. La figura 127 muestra el diagrama de fase para un camino circular, el cual genera las corrientes mostradas.

- Camino arbitrario: este método se muestra simplemente con propósitos didácticos o bien para considerar otras posibilidades. Consiste en asignar las magnitudes de los fasores de forma arbitraria, a pesar que el diagrama de fase de este método un motor ideal puede producir las mismas rotaciones angulares que los dos métodos anteriores.

Figura 127. **Control por micropaso**



Fuente: elaboración propia, empleando Inkscape.

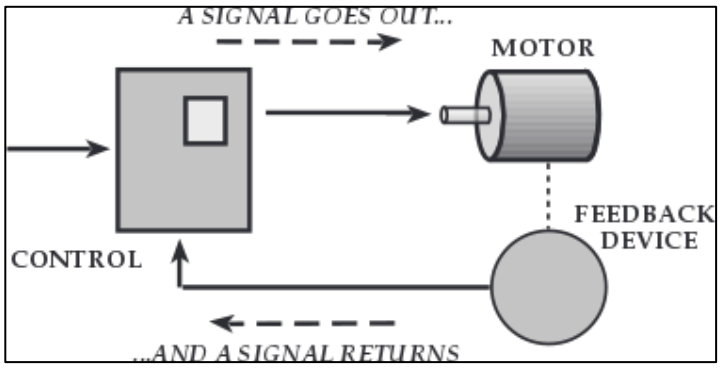
- Motores servo

Los motores servo no son en sí una clase específica de motor, el término hace referencia a un motor eléctrico en un sistema de control de lazo cerrado. Tienen sus aplicaciones en robots, maquinarías CNC o procesos de manufactura.

Un servomotor o motor servo es el ejemplo clásico de un lazo cerrado, el sistema controlado de este lazo es un motor eléctrico. Una planta de control trata que el motor pueda seguir una señal externa llamada seña de comando, generalmente proveniente de una unidad de procesamiento y mediante una señal de *feedback* o retroalimentación lleva el registro de la posición, velocidad o aceleración actual para así poder generar una señal, que es entregada al motor, y es adecuada para poder seguir la señal externa.

Estos motores también son llamados en su conjunto servomecanismos, aunque este término es más general y hace referencia al senso de error y retroalimentación negativa para corregir el desempeño de algún sistema. El tipo de motor eléctrico que forma parte de un servomecanismo no es relevante ya que cualquiera puede ser pareado por algún *encoder* para llevar registro de la posición o velocidad, la magnitud obtenida por el motor es comparada con la señal de comando y el controlador emite una señal que mediante un acople de impedancias se manipula la dinámica del motor. En los casos más simples solamente la posición es controlada y por lo general se utiliza como codificador un potenciómetro y *bang-bang* como método de control, es un anglicismo para hacer referencia a un control que cambia abruptamente entre dos estados, y el motor gira siempre a su máxima velocidad. Este tipo de mecanismo no es frecuente en la industria ya que no genera los resultados que esta requiere en determinadas aplicaciones.

Figura 128. **Lazo cerrado en un motor servo**



Fuente: *Servo Control Facts*. *Baldor motor and Drives*. http://www.baldor.com/Shared/manuals/1205-394.pdf. Consulta: octubre de 2015.

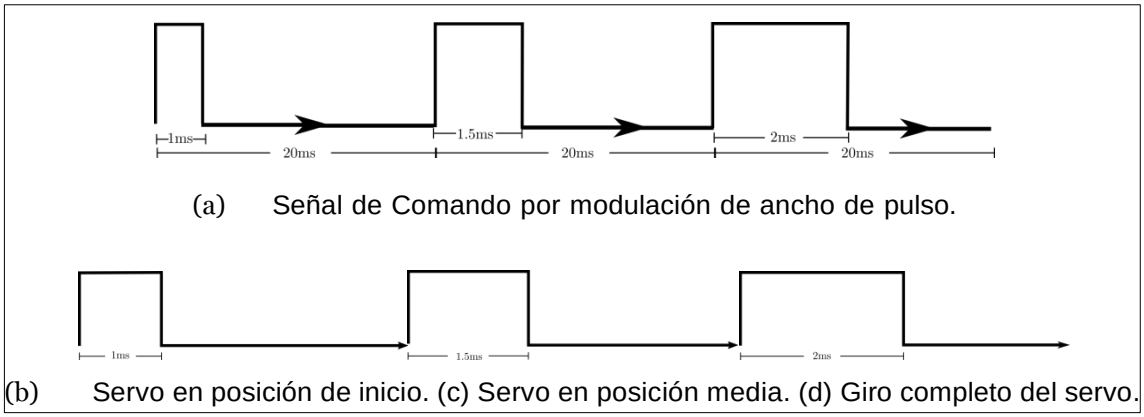
Los servomecanismos complejos utilizan las mediciones de posición y velocidad llevan a cabo un control de velocidad, en lugar de correr el motor a la máxima velocidad y con ello llevan a la posición al motor eléctrico. Junto con un algoritmo de control PID se le permite al motor llegar a la posición especificada en la señal de control de forma más precisa y rápida.

El objetivo de utilizar estos motores es separar en capas el sistema completo de tal forma que la unidad de procesamiento vea la posición o velocidad del motor como un lazo abierto, siendo responsable únicamente de generar la señal de comando adecuada al servo mecanismo. De esta forma todos los procesos llevados a cabo por el lazo cerrado que controlas las magnitudes de salida del motor son transparentes para cualquier sistema que dese utilizarlo.

Una forma común que usan los motores servo para comunicarse con una unidad de procesamiento es mediante una señal modulada en ancho de pulso. La señal de comando es una onda cuadrada con una frecuencia determinada,

algunos casos la frecuencia se encuentra entre 50 y 60 hertz. La posición o velocidad del motor pueden ser representados por una función afín del tiempo en alto de la señal.

Figura 129. **Señal de comando para servo común**



(a) Señal de Comando por modulación de ancho de pulso.

(b) Servo en posición de inicio. (c) Servo en posición media. (d) Giro completo del servo.

Fuente: elaboración propia, empleando Inkscape.

Por ejemplo, los servos que utilizan la modulación por ancho de pulso como señal de comando, en caso de utilizar 50 Hz, de los 20 ms que dura el periodo de la señal, si el tiempo en alto de la onda cuadrada es 1 ms el motor intentará llegar a la posición de arranque o inicio, si el tiempo en alto es de 2ms el motor intentará llegar a su posición máxima de giro. Si el tiempo en alto dura de 1 a 2 ms la posición del motor y el tiempo en alto conservan una relación afín. Es decir, si el ancho de pulso es 1,5 ms el motor estará a la mitad de su giro completo, si se encuentra en 1,75 ms estará a $\frac{3}{4}$ del giro completo del motor. Al momento de utilizar un servomotor se debe tener en cuenta que el motor tarda más de 20 ms en llegar a las posiciones indicadas por el ancho del pulso, por lo que la modulación requiere varios períodos con el mismo ancho para darle tiempo al motor de alcanzar la posición deseada.

En la práctica es posible encontrar servomecanismos que utilicen protocolos distintos al descrito anteriormente, variando desde la frecuencia de operación hasta la forma de entregar la información.

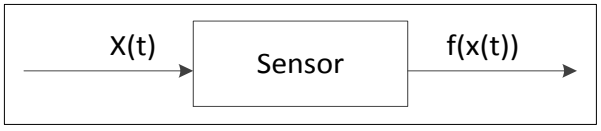
4.4. Modelos de sensores y actuadores

Sensores y actuadores conectan el mundo físico con el cibernético. Números en el mundo cibernético se encuentran relacionados con cantidades en el mundo físico. Tener un modelo correcto del sensor o actuador es sumamente importante para interpretar correctamente los datos que provienen del entorno y manipular de forma deseada las señales que se entregan a él.

4.4.1. Modelos lineales y afines

Algunos sensores pueden modelarse como sistemas sin memorias. Supóngase que se tiene una magnitud física expresada como una señal en el tiempo $x(t)$, la cual es codificada por un sensor. Y el sensor que entrega señales representativas de $x(t)$ a una unidad de procesamiento, es un sistema sin memoria. Una función característica de un sensor $f : \mathbb{R} \to \mathbb{R}$, obtiene las señales del entorno y las codifica para dar una salida de la forma $f(x(t))$, esta función f en un sensor es conocida también como función de distorsión del sensor por razones que serán evidentes en breve.

Figura 130. **Bloque de un sensor**



Fuente: elaboración propia, empleando Inkscape.

No todos los transductores pueden ser modelados como sistemas sin memoria, por lo que aquellos que pueden aproximarse a uno resultan útiles en el desarrollo de sensores. Aunque tener un sistema sin memoria facilita ya de por sí la interpretación de los resultados, siempre en ingeniería se buscan sistemas lineales en análisis de señales ya que ofrecen una garantía en el comportamiento de las salidas del sistema, frente a atenuaciones o amplificaciones que sucedan en las señales de entrada. Se dice que un sensor es lineal si existe una constante de proporcionalidad *a* 2 R, tal que su función característica esta dada de la siguiente forma:

$$f\big(x(t)\big) = ax\,(t), \qquad \forall x(t) \in \mathbb{R}, \qquad \text{[Ec. 4.14]}$$

Si existe una constante de sesgo $b \in \mathbb{R}$, tal que la función de distorsión que caracteriza al sensor es de la forma.

$$f\big(x(t)\big) = ax\,(t) + b, \qquad \forall x(t) \in \mathbb{R}, \qquad \text{[Ec. 4.15]}$$

Se dice que el modelo del sensor es afín. No todos los sensores afines son lineales, pero si todos los sensores lineales son afines, considerando la constante de sesgo como cero.

La constante de proporcionalidad representa la sensibilidad del sensor. Si es relativamente grande podrá hacer notorios los cambios pequeños en la señal entregada por el sensor. Algunos actuadores pueden ser representados por funciones afines y lineales, aunque muchos actuadores son sistemas con memoria y su manipulación es parte de la teoría de control, tal como un motor eléctrico de corriente continua que se hace pasar por un servomecanismo.

4.4.2. Rango de un sensor y cuantización

El rango de un sensor es el conjunto de valores que este puede interpretar, por lo general es un intervalo cerrado conexo continuo, es decir posee un valor mínimo $L \in \mathbb{R}$, un valor máximo $H \in \mathbb{R}$ y pertenece al conjunto $[L,H] \subset \mathbb{R}$. Un sensor modelado con una función afín, fuera del rango ya no puede seguir obedeciendo este modelo. Muchos sensores saturan al estar percibiendo valores fuera del intervalo [*L*, *H*]. Lo cual significa que sostienen un valor máximo o mínimo para valores fuera del rango de lectura, tal como se expresa a continuación:

$$f\big(x(t)\big) = \begin{cases} ax(t) + b & Si\ L \leq x\ (t) \leq H \\ aH + b & Si\ \ x\ (\ t\ )\ > H \\ aL + b & Si\ \ x\ (\ t\ )\ > L \end{cases} \qquad \text{[Ec. 4.16]}$$

La función característica del modelo de un sensor mostrada en la ecuación 4.16 no es afín, de hecho, todos los sensores tienen no linealidades similares a la ecuación 4.16, y más importante aún no todos se comportan como una función afín, muchos de ellos obedecen otros modelos.

Los sensores digitales no son capaces de distinguir dos valores muy cercanos de una magnitud física. Esto se debe a que un sensor digital para representar una magnitud física utiliza *n* bits. Debido a ello existen solamente 2^n mediciones diferentes que puede llevar a cabo un sensor digital. Dado que una cantidad en el mundo físico posee valores pertenecientes a un intervalo continuo solo una cantidad finita de ellos puede ser interpretada y el resto será aproximada por el sensor, es decir si *x* es una señal en el mundo físico, para cualquier instante *t*, el sensor debe de seleccionar para *x*(*t*) uno de los 2^n valores diferentes que puede ofrecer el sensor, lo cual recibe el nombre de cuantización.

La precisión Δ, de un sensor digital, es el valor absoluto de diferencia más pequeña posible entre dos valores de una cantidad física cuyas respectivas lecturas efectuadas por el sensor son distinguibles. Para un sensor digital ideal dos cantidades físicas que difieren por una precisión Δ, los valores digitales obtenidos de la cuantización diferirían por un solo bit.

Un parámetro importante al trabajar con sensores es el rango dinámico, que es un número $D \in \mathbb{R}_+$ cuyo valor está dado por la razón de la diferencia del valor máximo y el valor mínimo del rango, entre la precisión del sensor.

$$D = \frac{H-L}{\Delta}, \qquad \text{[Ec. 4.17]}$$

Donde el límite superior del sensor esta dado por *H*, y el inferior por *L* del rango. Por lo general el rango dinámico es usualmente medido en decibeles, quedando expresado de la siguiente manera:

$$D_{dB} = 20\ log_{10}\left(\frac{H-L}{\Delta}\right) \qquad \text{[Ec. 4.18]}$$

En general, un sensor digital ideal de *n* bits se encuentra modelado por una función $f: \mathbb{R} \to \{0,1,\dots,2^n\}$ la cual es llamada función de cuantización del sensor. Sea *g* (*x*(*t*)) una función afín definida para el rango del sensor tal que:

$$\mathrm{g}\big(x(t)\big) = \frac{2^n-1}{H-L}(x(t)-L), \qquad L \le x(t) \le H \qquad \text{[Ec. 4.19]}$$

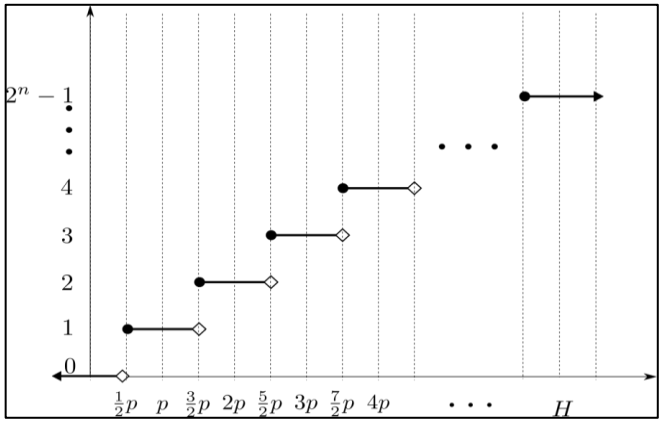
La función de distorsión del sensor digital, quedaría definida de la siguiente forma:

$$f\big(x(t)\big) = \begin{cases} \lg\big(x(t)\big) + \Delta/2 & Si\ L \leq x(t) \leq H \\ 2^n - 1 & Si\ \ x\ (t)\ > H \\ 0 & Si\ \ x\ (t) > H \end{cases} \qquad \text{[Ec. 4.20]}$$

El ⌊·⌋ es conocido como piso y aproxima de un número real al entero menor más próximo. La contraparte de este operador es el operador techo ⌈·⌉, el cual aproxima al entero mayor más próximo.

En la ecuación 4.20, se tiene que $\Delta = \frac{H-L}{2^n}$ se puede apreciar gráficamente el comportamiento de una función de distorsión como 4.20 en la figura 134, tomando a *p* con el mismo valor de la precisión.

Figura 131. **Función de distorsión de un sensor digital de n bits, con $\mathbf{L = 0}$ Y $\mathbf{p = \Delta}$**



Fuente: elaboración propia, empleando Inkscape.

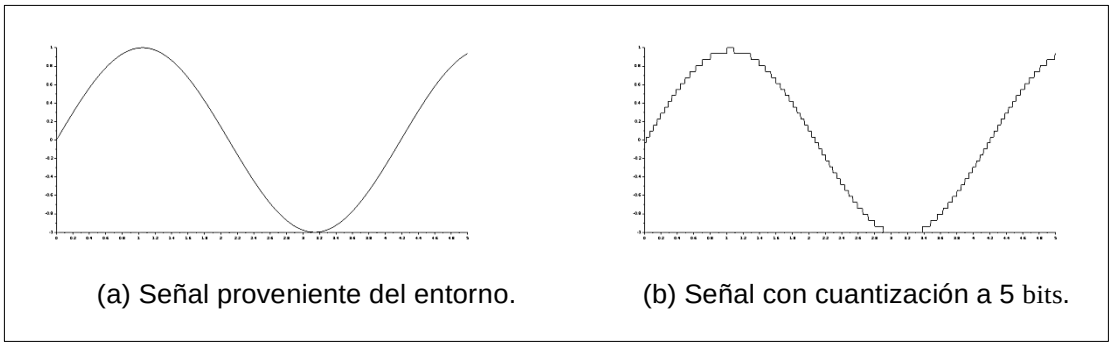
Para un sensor con una función de distorsión dictada por cuantización, como la que muestra en la figura 134, el rango dinámico está dado por:

$$D_{dB} = 20\ log_{\ 10}\left(\frac{H-L}{\Delta}\right) \qquad \text{[Ec. 4.21]}$$

Los sensores digitales devuelven valores discretos de la señal, comprendidos entre 0 y $2^n - 1$. En muchas situaciones es necesario tener una aproximación de los valores que el sensor se encuentra leyendo del entorno. Por ejemplo, para un sensor digital donde se aplicó la cuantización sobre la función afín 4.19, la aproximación se obtiene aplicando g^{-1} sobre la cuantización.

$$(g^{-1} o\ f)\ (t) = \frac{H-L}{2^n-1} f(t) + L \qquad \text{[Ec. 4.22]}$$

Figura 132. **Función de distorsión de un sensor de 5 bits**



(a) Señal proveniente del entorno. (b) Señal con cuantización a 5 bits.

Fuente: elaboración propia, empleando Inkscape.

La figura 132 b muestra como la señal se ve procesada por un sensor digital de 5 bits. No es necesario que el sensor haga la cuantización de una función afín, como *g* en el ejemplo, es posible encontrar situaciones donde el sensor toma datos del entorno en escalas logarítmicas o bien la aproximación que se planteó anteriormente se dé de forma diferente, tal como sucede al usar un *compander* en la obtención de una señal de voz; el cuál utiliza una escala logarítmica para interpretar la señal obtenida, con el fin de hacer más notorias las diferencias entre los valores pequeños y que las diferencias entre los valores grandes sean imperceptibles con el fin de utilizar eficiente el número de

bits que se poseen para transmitir una señal, ya que las diferencias entre los valores pequeños en una señal de voz son los que llevan mayor información.

4.4.3. Ruido

Intuitivamente, el ruido es la parte de la señal que no se desea tomar en cuenta. Es común modelar la señal *x´* obtenida por un sensor como la superposición de la señal proveniente del entorno *x*, la cual lee el sensor, y el ruido *n* al que se encuentra expuesta.

$$x´(t) = x(t) + n(t) \qquad \text{[Ec. 4.23]}$$

Esta ecuación permite modelar imperfecciones incluso aquellas no-lineales como el ruido debido a la cuantización llevada a cabo por algún sensor. Estimar que tanto ruido se percibe en una medición es muy importante. El valor RMS (siglas en inglés de *RootMean Square*) de la señal de ruido se encuentra asociado a su potencia, y se define de la siguiente manera:

$$N = \underset{T \neg \infty}{lím} \sqrt{\frac{1}{2T} \int_{-T}^{T} (n(\tau))^2 d\tau} \qquad \text{[Ec. 4.24]}$$

La relación señal a ruido SNR (siglas en inglés de *signal to noise ratio*) se encuentra definida en términos del valor RMS del ruido *N*.

$$SNR_{db} = 20\ log_{10}\left(\frac{X}{N}\right)\ dB \qquad \text{[Ec. 4.25]}$$

Donde X es el valor RMS de la señal de entrada x definido de igual forma que N para el ruido.

La ergodicidad es una propiedad muy importante de algunos sistemas y se hace constantemente referencia a este concepto al tratar con análisis de ruido; se tiene especial interés en sistemas ergódicos ya que el promedio temporal de ciertas magnitudes puede obtenerse como promedio sobre el espacio de estados lo cual simplifica las predicciones del comportamiento de estos sistemas.

Se dice que una señal *y*(*t*) es ergódica si valor esperado en relación al tiempo es igual al valor esperado en relación a los valores que esta puede tomar.

$$E[y(t)] = E_t[y(t)] = E_{y(t)}[y(t)] \qquad \text{[Ec. 4.26]}$$

Al tratar con análisis de ruido en señales, es útil considerar los procesos aleatorios estocásticos. Son aquellos compuestos de variables aleatorias cada una de ellas con una distribución de probabilidad asociada. En el caso de una señal estocástica *x*(*t*), puede ser vista en su conjunto como una variable aleatoria *X* o *x*, teniendo en cuenta que se trata de los valores de la imagen de la señal y no en la relación con el tiempo, que puede tomar cualquier valor atado a una distribución de probabilidad asociada.

Considerando una señal continua *y*(*t*) como aleatoria, ergódica y estocástica, es posible decir que su valor RMS de la siguiente manera.

$$N = \sqrt{E[y^2]} \qquad \text{[Ec. 4.27]}$$

Con base en la ecuación 4.27 es posible calcular la SNR de una señal expuesta al ruido provocado por la función de distorsión de un sensor digital. Considérese un sensor digital con rango definido por el intervalo simétrico

[-*A*, *A*], y para ser consistente $A > 0$, con una precisión $\Delta = \frac{A}{2^{n-1}}$ unif*or*me en todo el intervalo y la función de distorsión en la ecuación 4.20.

Es posible obtener un estimado de la SNR *X* de una señal obtenida por este sensor si se considera solamente señales que pueden tomar valores dentro del rango y que el sensor es capaz de obtener mediciones con las mismas características de todas las señales $x \in [-A, A]^{\mathbb{R}}$ independientemente de la señal que se trate; teniendo lo anterior en cuenta, la obtención de señales por este sensor puede ser considerada como un proceso ergódico y estocástico con una distribución de probabilidad unif*or*me $p(x(t)) = \frac{1}{2A}$ sobre los valores que puede tomar la señal para todo tiempo *t* .

$$x = \sqrt{\int_{-A}^{A} x^{2p(x)dx}} \qquad \text{[Ec. 4.28]}$$

$$= \sqrt{\int_{-A}^{A} \frac{x^2}{2A} dx} \qquad \text{[Ec. 4.29]}$$

$$= \sqrt{\frac{1}{2A}\frac{x^3}{3}\Big|_{-A}^{A}} \qquad \text{[Ec. 4.30]}$$

$$= \sqrt{\frac{A^2}{3}} \qquad \text{[Ec. 4.31]}$$

Todos los valores dentro del rango pueden tener solamente un error perteneciente al intervalo $e(t) \in \left[-\frac{\Delta}{2}, \frac{\Delta}{2}\right]$. Y si todos los valores en el rango tienen la misma probabilidad de ocurrencia el valor RMS del error debido a la cuantización del error esta dado por:

$$N = \sqrt{\int_{-\Delta}^{\Delta} e^2 p(e) de} \quad \text{[Ec. 4.32]}$$

$$= \sqrt{\int_{-\Delta/2}^{\Delta/2} \frac{e^2}{\Delta} de} \quad \text{[Ec. 4.33]}$$

$$= \sqrt{\frac{1}{\Delta}\frac{e^3}{3}\Big|_{-\Delta/2}^{\Delta/2}} \quad \text{[Ec. 4.34]}$$

$$= \sqrt{\frac{\Delta^2}{12}} \quad \text{[Ec. 4.35]}$$

Dado que el rango del sensor digital considerado es simétrico y la precisión unif*or*me en todo el intervalo se tiene que:

$$2A = 2^n \Delta \Longrightarrow A = \frac{2^n \Delta}{2} \quad \text{[Ec. 4.36]}$$

De esa cuenta es posible reducir la SNR, para el caso del sensor a consideración, en términos de la precisiónΔ; con el fin de reducir a una expresión que brinde una mejor perspectiva del comportamiento de la relación señal a ruido al tener un sensor con determinado número de bits.

$$S_{dB} = 20log_{10}\left(\frac{X}{N}\right) dB \quad \text{[Ec. 4.37]}$$

$$= 20log_{10}\left(\frac{\sqrt{\frac{A^2}{3}}}{\sqrt{\frac{\Delta^2}{12}}}\right) dB \quad \text{[Ec. 4.38]}$$

$$= 20log_{10}\left(\frac{\sqrt{\frac{2^{2n}(A^2)}{3}}}{\sqrt{\frac{\left(\frac{\Delta^2}{2}\right)}{3}}}\right)dB \quad \text{. 4.39]}$$

$$= 10log_{10}(4^n)dB \quad \text{[Ec. 4.40]}$$

$$= 10log_{10}(4)dB \approx 6{,}0206_n dB \quad \text{[Ec. 4.41]}$$

En la ecuación 4.41 puede verse que la relación señal a ruido incrementa en aproximadamente 6 dB por cada bit que el sensor tenga de resolución, es decir que el ruido de cuantización disminuye drásticamente en sensores con una alta resolución, medida en número de bits.

Esta consideración no siempre se puede aplicar en la práctica ya que muchos bits en un sensor eleva considerablemente su costo, además que la señal física se ve rodeada de muchos factores externos los cuales se manifiestan como ruido en la señal obtenida por el sensor, el ruido térmico, el ruido rosa o el ruido blanco son algunos modelos que pretenden explicar como estos factores afectan la señal física que el sensor se encuentra interpretando, y utilizar sensores con muchos bits sería inútil porque los niveles de ruido, debido a fenómenos del ambiente también serían percibidos por el sensor y por ende en las mediciones y afectarían de igual forma la señal obtenida por el sensor.

En la figura 133 se muestra una serie de gráficas con los efectos de cuantizar con diferentes cantidades de bits, una señal con ruido. Se muestra la señal sin presencia de ruido; la señal expuesta a ruido, y en el resto de la figura, los efectos de cuantizar una señal a diferente número de bits. Se puede observar un cambio notorio de la señal cuantizando a 3 bits a una señal cuantizando a 4 bits se puede observar como gradualmente las diferencias

entre cuantizar a un número menor o número mayor dejan de ser significativas. Por ejemplo, cuantizar a 5 bits o 6 bits resulta en señales muy similares, y una vez que la precisión es menor que los niveles de ruido que se perciben en la señal prácticamente es inútil aumentar el número de bits al cuantizar; tal como se aprecia en la figura 133, en la que se ha realizado la cuantización a 7 bits y 13 bits respectivamente.

En otras palabras, aumentar el número de bits, no solamente eleva los costos de construcción drásticamente, sino que pierde totalmente el sentido debido a fenómenos externos que se cuelan en las mediciones ya que una resolución muy pequeña hace indistinguibles estos fenómenos de la señal que se desea obtener, y en muchos casos no es posible eliminarlos en la medición de una señal.

Este fenómeno se puede explicar cualitativamente y cuantitativamente. Para ello considérese el mismo sensor con rango simétrico [-*A*, *A*], que se trató con anterioridad. Además, considérese el ruido que afecta a la señal del entorno como un fenómeno aleatorio, ergódico y estocástico.

La relación señal a ruido *SNR* sujeta a ruido externo y a ruido de cuantización puede modelarse de la siguiente manera:

$$SNR_{dB} = 20log_{10}\left(\frac{X}{N}\right) \qquad \text{[Ec. 4.42]}$$

$$= 10log_{10}\left(\frac{x^2}{N_n^2+N_q^2}\right) \qquad \text{[Ec. 4.43]}$$

Donde N_n es el valor RMS de la señal de ruido externo al sensor n(*t*), y N_q es el valor RMS del ruido de cuantización, visto como el error *q*(*t*). La razón de

escribir la relación señal a ruido de la forma que aparece en la ecuación 4.43 se hará evidente a continuación, y se podrá ver que son equivalentes. Recordando la ecuación 4.23, se tiene para una señal expuesta a ruido de cuantización y a ruido externo del sensor.

$$n(t) = q(t) + n(t) \qquad \text{[Ec. 4.44]}$$

La relación señal a ruido, en su definición estadística se tiene:

$$N = \sqrt{E[n^2(t)]} = \sqrt{E[n(t) + q(t))^2]} \qquad \text{[Ec. 4.45]}$$

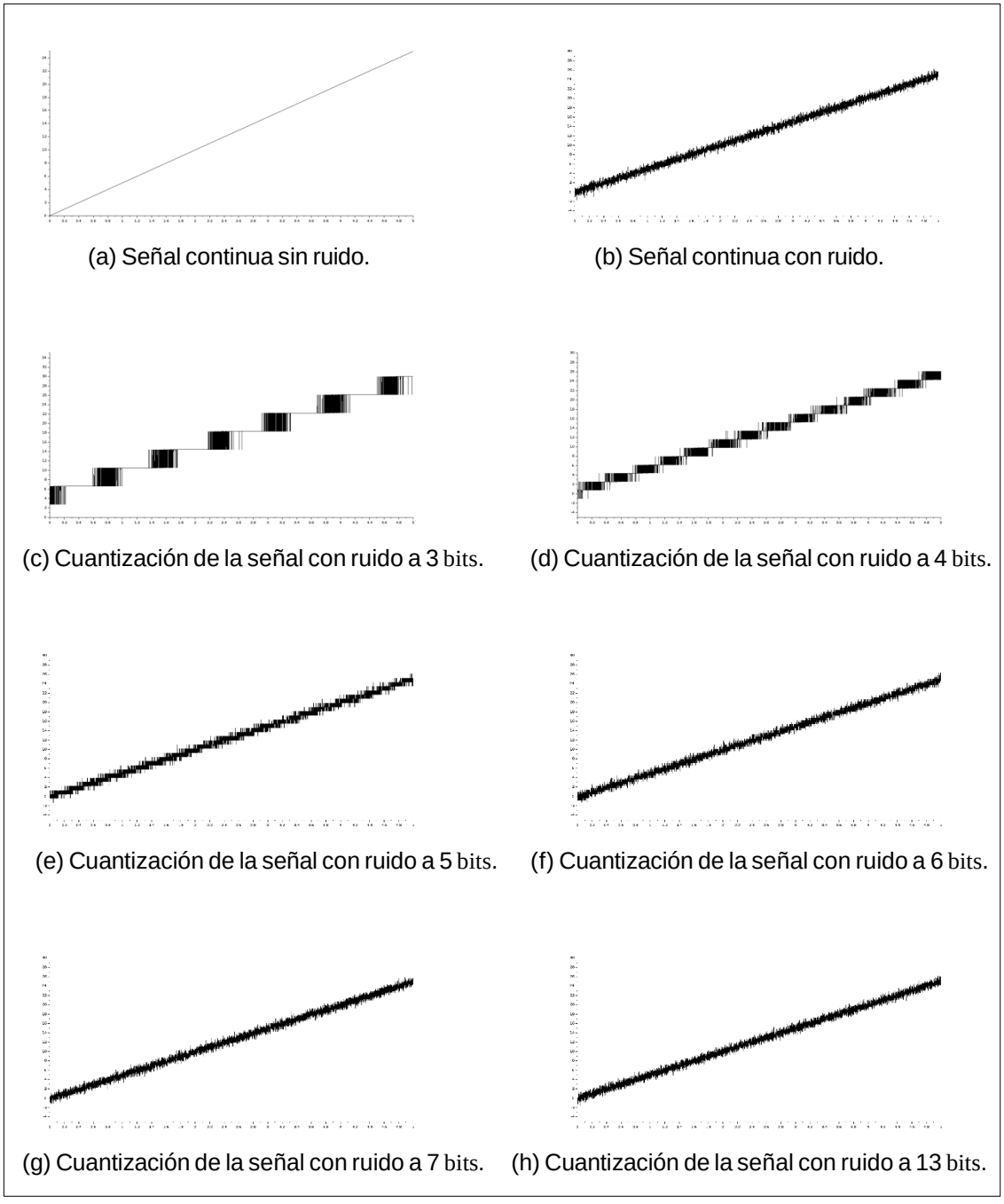
Hay que resaltar el hecho de que el ruido que afecta la señal del entorno es totalmente independiente de la forma en que el sensor la cuantiza y viceversa; por otro lado, la ecuación 4.45 muestra la esperanza del cuadrado de una suma de variables aleatorias, cuyo resultado puede ser simplificado si se considera variables aleatorias *X* i ∈X indexadas por un índice $i \in \mathcal{J}$

$$E\left[\left(\sum_{X_i \in \mathfrak{X}} X_i\right)^2\right] = E\left[\sum_{X_i \in \mathfrak{X}} X_i^2 + \sum_{\substack{i,j \epsilon \mathcal{J} \\ i \neq j}} X_i X_j\right] \qquad \text{[Ec. 4.46]}$$

$$= E\left[\sum_{X_i \in \mathfrak{X}} X_i^2\right] + E\left[\sum_{\substack{i,j \epsilon \mathcal{J} \\ i \neq j}} X_i X_j\right] \qquad \text{[Ec. 4.47]}$$

$$= \sum_{X_i \in \mathfrak{X}} E[X_i^2] + \sum_{\substack{i,j \in \mathcal{J} \\ i \neq j}} E[XiXj] \qquad \text{[Ec. 4.48]}$$

Figura 133. **Efectos de la cuantización de una señal con ruido**



(a) Señal continua sin ruido.

(b) Señal continua con ruido.

(c) Cuantización de la señal con ruido a 3 bits.

(d) Cuantización de la señal con ruido a 4 bits.

(e) Cuantización de la señal con ruido a 5 bits.

(f) Cuantización de la señal con ruido a 6 bits.

(g) Cuantización de la señal con ruido a 7 bits.

(h) Cuantización de la señal con ruido a 13 bits.

Fuente: elaboración propia, empleando SCILAB.

Para la ecuación 4.44, se tiene qu $\mathcal{X} = \{n(t), q(t)\} y\, \mathcal{I} = \{0,1\}$, además como se mencionó anteriormente tanto *n*(*t*) y *q*(*t*) son consideradas variables aleatorias independientes y en ambos casos se los valores negativos tienen la misma probabilidad de ocurrencia que los positivos, es decir ambas variables aleatorias tienen una media cero, por lo que la simplificación 4.48 para la ecuación 4.45, sería:

$$N^2 = E\left[\left(n(t)\right) + q(t))^2\right] \quad \text{[Ec. 4.49]}$$

$$= E\left[\left(n(t)\right) + q(t))^2\right] + E[2q(t)n(t)] \quad \text{[Ec. 4.50]}$$

$$= Var[n(t)] + Var[q(t)] + 2Covar[n(t), q(t)] \quad \text{[Ec. 4.51]}$$

$$= N_n^2 + N_q^2 \quad \text{[Ec. 4.52]}$$

Donde N_n, es el valor RMS de $n(t)\ y\ N_q$el valor RMS de *q*(*t*), y el resultado de la ecuación 4.52 se debe a que la covarianza entre dos variables aleatorias independientes es cero[8], Covar $[n(t)n(t)] = 0$

La ecuación 4.25 puede escribirse de nuevo aplicando los teoremas de logaritmos para la raíz y usar en la expresión los cuadrados de los valores RMS de cada una de las señales que son parte del análisis.

Por tratarse de un ejemplo común, se considera a la curva normal como la distribución asociada a la señal n(*t*) vista como un proceso aleatorio, estocástico y ergódico.

Considérese n como variable aleatoria con media cero y por fines demostrativos se asume que el 95 % a la vez el error no sobrepasa un décimo

del valor de *A*, implicando que la distribución posee una desviación estándar σ =0,05*A* aproximadamente.

$$p(n) = \mathcal{N}(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(n-\mu)^2}{2\sigma^2}} \quad \text{[Ec. 4.53]}$$

Considerando a la media del ruido $\mu = 0$, usando la sustitución $r = -$ y sabiendo que:

$$\frac{d}{dr}\left(e^{-r^2}\right) = -2re^{-r^2} \quad \text{[Ec. 4.54]}$$

$$\int_{-\infty}^{\infty} e^{-r^2} dr = \sqrt{\pi}, \quad \text{[Ec. 4.55]}$$

Se puede calcular el valor RMS *N* de la señal.

$$N_N = \sqrt{E[n^2]} \quad \text{[Ec. 4.56]}$$

$$= \sqrt{\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} n^2 e^{-\left(\frac{n}{\sigma\sqrt{2}}\right)^2} dn} \quad \text{[Ec. 4.57]}$$

$$= \sqrt{\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} \left(r\sigma\sqrt{2}\right)^2 e^{-r^2} dn} \quad \text{[Ec. 4.58]}$$

$$= \sqrt{\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} \left(r\sigma\sqrt{2}\right)^2 e^{-r^2} \left(\sigma\sqrt{2}\right) dr} \quad \text{[Ec. 4.59]}$$

$$= \sqrt{\frac{2\sigma^2}{\sqrt{\pi}} \int_{-\infty}^{\infty} r^2 e^{-r^2} dr} \quad \text{[Ec. 4.60]}$$

$$= \sqrt{\frac{2\sigma^2}{\sqrt{\pi}}\left(\left(r\frac{-e^{-r^2}}{2}\Big|_{-\infty}^{\infty} + \frac{1}{2}\int_{-\infty}^{\infty} e^{-r^2}dr\right)\right)} \quad \text{[Ec. 4.62]}$$

$$= \sqrt{\frac{2\sigma^2}{\sqrt{\pi}}\left(\frac{\sqrt{\pi}}{2}\right)} \quad \text{[Ec. 4.63]}$$

$$= \sigma \quad \text{[Ec. 4.64]}$$

$$\cong 0{,}05\ A \quad \text{[Ec. 4.65]}$$

De lo anterior se puede escribir una expresión para la relación señal a ruido SNR, donde la señal obtenida por el sensor con precisión Δ y rango [-*A*, *A*] se ve expuesta a diferentes fuentes de ruido.

$$SNR_{dB} = 10log_{10}\left(\frac{\frac{A^2}{3}}{\sigma^2+\frac{\Delta^2}{12}}\right) \quad \text{[Ec. 4.66]}$$

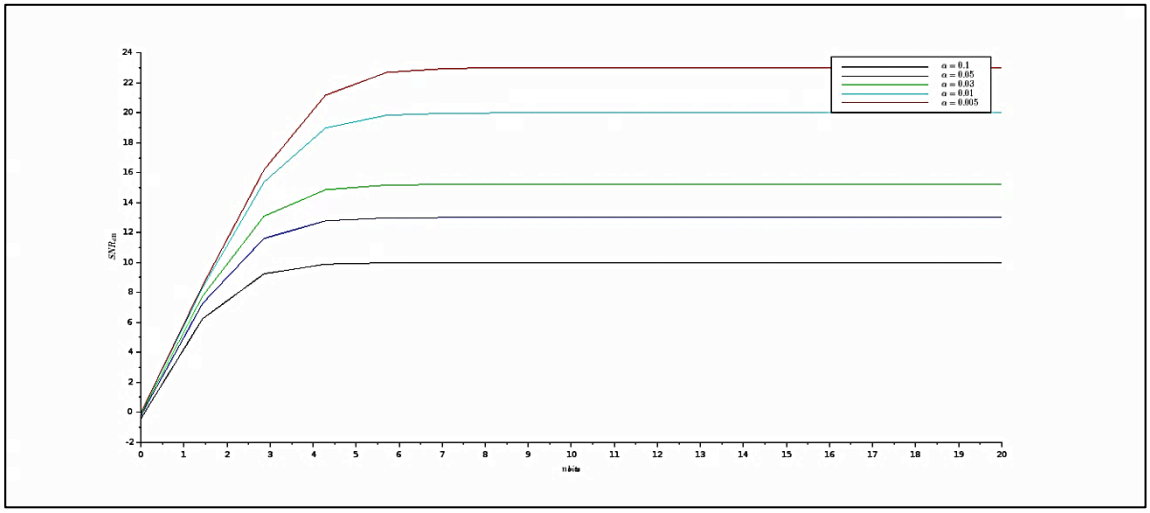
$$= 10log_{10}\left(\frac{\left(2^n(\Delta/2)\right)^2/3}{\left((0{,}05)2^n\Delta\right)^2+\Delta^2/12}\right) \quad \text{[Ec. 4.67]}$$

$$= 10log_{10}\left(\frac{2^{2n}}{(0{,}03)2^{2n}+1}\right) \quad \text{[Ec. 4.68]}$$

$$= 10log_{10}\left(\frac{2^{2n}}{\alpha 2^{2n}+1}\right) \quad \text{[Ec. 4.69]}$$

La ecuación 4.69 brinda una expresión más general para la relación señal a ruido, donde el parámetro α depende de la distribución a utilizar para el modelo del ruido, por lo que diferentes distribuciones variarán este parámetro.

Figura 134. **Diferentes SNR_{dB} contra bits de cuantización**



Fuente: elaboración propia, empleando SCILAB.

La figura 134 muestra el comportamiento de la relación señal a ruido, de diferentes distribuciones de probabilidad al cuantizar la señal a *n* bits. Puede verse como para el ejemplo expuesto (α =0,03, gráfica verde) utilizar más de 6 bits es prácticamente inútil.

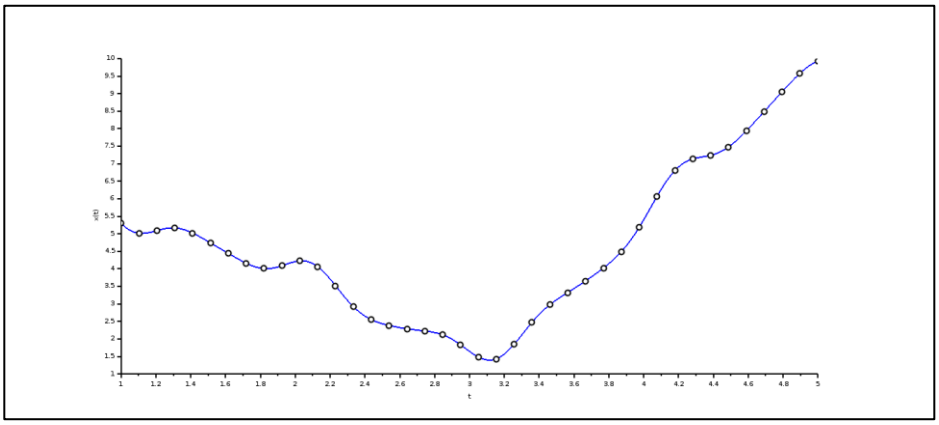
El análisis de ruido es un tema complejo. Solo se han tratado algunos ejemplos que explican resultados comunes en la práctica, su análisis detallado escapa del presente trabajo, sin embargo se debe resaltar que en las mediciones que se hagan utilizando cualquier sensor se deben tener presentes los efectos que este les causa.

4.4.4. Muestreo

Un sensor digital por su naturaleza discreta, solo es capaz de asociar valores para instantes en particular de una señal del entorno físico, en

naturaleza continua. Cuya idea intenta atrapar la figura 138, donde cada punto representa un valor capturado por el sensor en un instante dado. A este proceso de capturar un valor de la señal en instantes particulares se le conoce como muestreo.

Figura 135. **Muestreo por un sensor digital de una señal física x(t)**



Fuente: elaboración propia, empleando SCILAB.

Un muestreo uni*for*me es aquel donde hay una constante T_s , llamada período de muestreo y las capturas hechas por el sensor fueron tomadas se encuentran separadas entre sí por intervalos de tiempo con valor T_s . La señal resultante de realizar un muestreo puede ser modelada como la función *s*: $\mathbb{Z} \to \mathbb{R}$ definida de la siguiente manera:

$$\forall n \epsilon \mathbb{Z}, \quad s(n) = f\left(x(nT_s)\right) \qquad \text{[Ec. 4.70]}$$

La señal continua *x*(*t*) solo es observada en los tiempos $t = nT_s$, donde *n* es un entero, *f* es la función de distorsión del sensor generalmente asociada la cuantización. La frecuencia de muestreo $f_s = 1/T_s$, tiene unidades de muestras por segundo y generalmente Hertz es su dimensional. En algunos textos es

frecuente encontrar las señales resultantes de un muestreo, o simplemente señales discretas, utilizando corchetes en lugar de paréntesis para diferenciarlas de sus homólogas continuas; por ejemplos *s*[*n*]. Se debe tener presente en el muestreo de señales que existen infinitas *x* diferentes que tendrían exactamente la misma representación discreta *s*. Este fenómeno se conoce como *aliasing*. Considérese como ejemplo una señal continua de sonido de forma sinusoidal cuya frecuencia es de 1 kHz.

$$x(t) = \cos(2000\pi t) \qquad \text{[Ec. 4.71]}$$

Por este momento considérese un sensor ideal que no realiza proceso de cuantización, cuya función de distorsión f es el caso especial de ser la identidad, si se toman un muestreo de 10 000 muestras por segundo, se obtiene una frecuencia de muestreo $f_s = 10\ kHz$, la cual generaría una función de muestro dada por las muestras.

$$s(n) = f\big(x(nT_s)\big) = \cos\left(\frac{1}{5}\pi n\right) \qquad \text{[Ec. 4.72]}$$

Ahora considérese una señal de $11\ kHz$

$$x'(t) = \cos 22\ 000\pi t \qquad \text{[Ec. 4.73]}$$

Esta es muestreada a la misma frecuencia de muestreo que la señal anteriormente muestreada.

$$s'(n) = f\big(x'(nt_s)\big) \qquad \text{[Ec. 4.74]}$$

$$= \cos\left(\frac{11}{5}\pi n\right) \qquad \text{[Ec. 4.75]}$$

$$= \cos\left(\frac{11}{5}\pi n + 2\pi n\right) \quad \text{[Ec. 4.76]}$$

$$= \cos\left(\frac{11}{5}\pi n\right) \quad \text{[Ec. 4.77]}$$

Ambas señales sinusoidales de 11 y 1 kHz son aliases para la frecuencia de muestreo a 10kHz.

El fenómeno del *aliasing* es complejo, aunque es posible reducir sus efectos si se toma en cuenta el criterio de Nyquist-Shannon, en honor a Harry Nyquist y Claude Shannon. Existen muchas versiones de este criterio, también enunciado independientemente por E. T. Whittaker, Vladimir Kotelnikov entre otros. Por lo que es posible encontrarlo también con los nombres Nyquist–Shannon–Kotelnikov, Whittaker–Shannon–Kotelnikov, Whittaker–Nyquist–Kotelnikov–Shannon, y el teorema cardinal de interpolación; el cual relaciona las señales con la frecuencia a la cual realizar el muestreo para poder reconstruirlas nuevamente. De forma similar que la transformada de Laplace se utiliza para el análisis de algunos sistemas, es posible utilizar la transformada de Fourier para el análisis de señales.

Cuando $x(t)$ es una función con una transformada de Fourier $X(f)$.

$$x(f) = \int_{-\infty}^{\infty} x(t) e^{i2\pi f t}\, dt \quad \text{[Ec. 4.78]}$$

La fórmula de Poisson indica que las muestras $x\ (nT)$ de $x(t\ )$ son suficientes para crear una suma periódica de $X(\ f\ )$. Como resultado de muestrear una señal con espectro X(f) a una frecuencia de muestreo f_s se tendría como espectro $X_s(\ f\ )$.

$$x_s(f) = \sum_{k=-\infty}^{\infty} X(f - kf_s) \qquad \text{[Ec. 4.79]}$$

$$= \sum_{k=-\infty}^{\infty} T_s \cdot x(nt_s) e^{-i2\pi n T_s f} \qquad \text{[Ec. 4.80]}$$

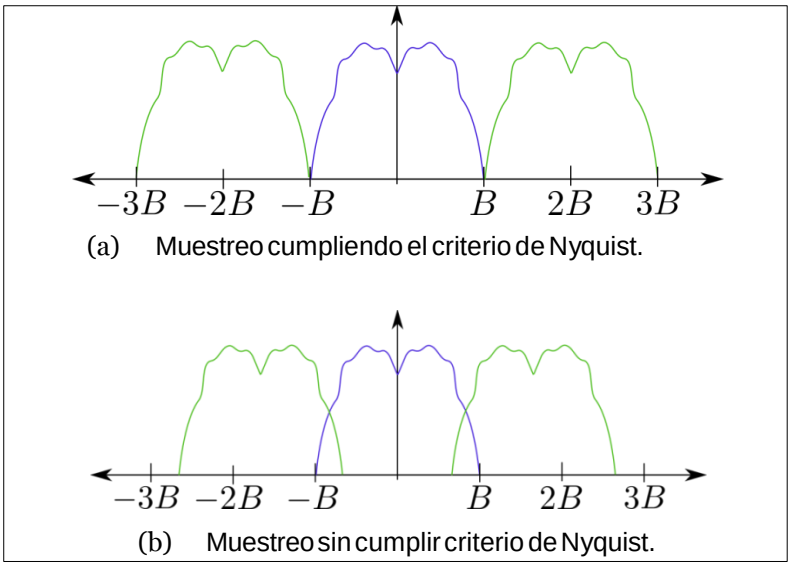
Esta es una función periódica y se tiene su equivalente como una serie de Fourier, con coeficientes $T_s x(nT_s)$ para enteros n.

Si $X(f)$ se encuentra limitada en ancho de banda por W

$$X(f) = 0 \qquad Si|f| > W \qquad \text{[Ec. 4.81]}$$

De la ecuación 4.80 se tiene que, si f_s ÈW se evitan los traslapes, por decirlo de alguna forma, entre los espectros que componen la señal $f_s > W$, tal como se muestra en la figura 139. De lo contrario el espectro sería de igual forma periódico, pero los traslapes ocurrirían, como se muestra en la figura 139, siendo imposible recuperar la señal original mediante procesos de filtrado.

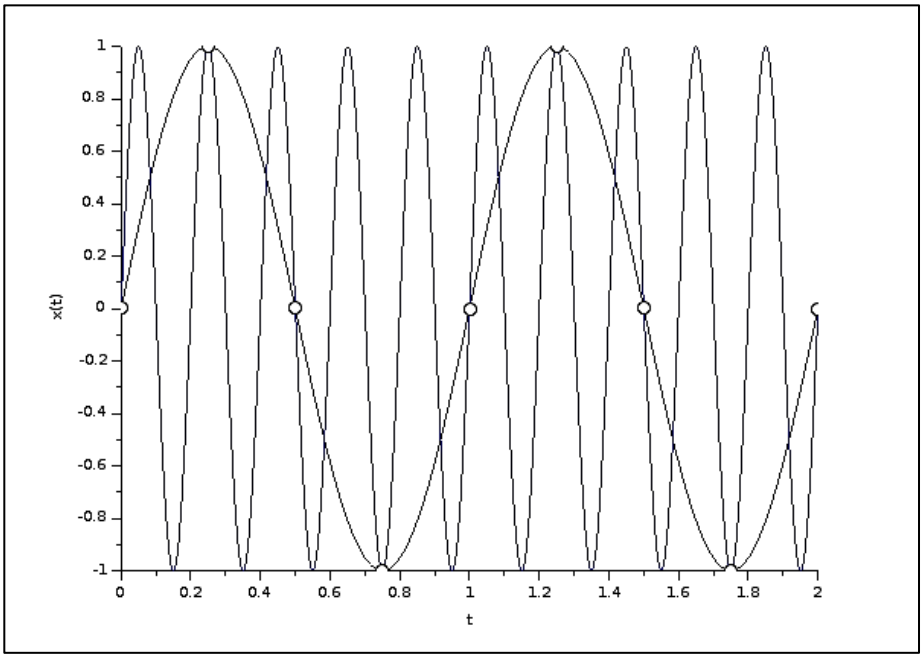
Figura 136. **Espectros de Fourier de señales muestradas**



Fuente: elaboración propia, empleando Inkscape.

En el artículo *Communication in presence of noise,* Shannon habla acerca de muestrear una señal a una frecuencia mayor a dos veces que la frecuencia más alta de una señal: "…Este hecho es de conocimiento común en el arte de la comunicación. La justificación intuitiva es que si una señal *f* (*t*) no contiene frecuencias mayores que *W*, ella no puede cambiar substancialmente a un nuevo valor en menos tiempo que la mitad de un ciclo de su frecuencia más alta, que es 1/2*W* …"[8].

Figura 137. ***Aliasing* de dos señales. Una a 1 Hz muestreadas a 4 Hz**



Fuente: elaboración propia, empleando SCILAB.

[8] SHANNON, Claude. *Communication in presence of noise*. https://web.stanford.edu/class/ee104/shannonpaper.pdf. Consulta: octubre de 2015.

4.4.5. Modelo matemático de un motor de corriente directa

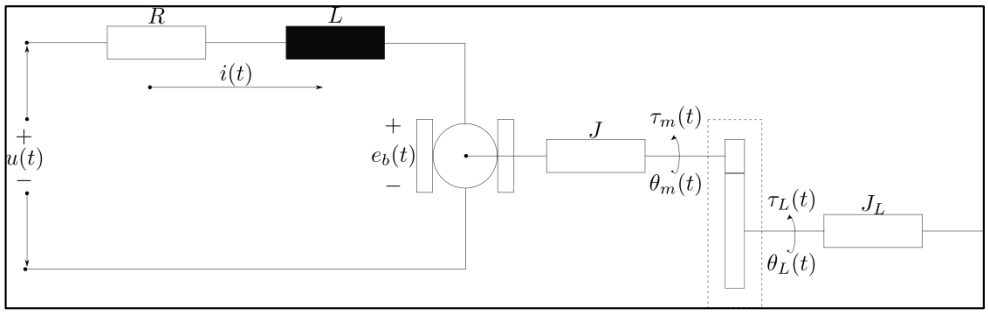
La figura 141 muestra el modelo eléctrico y mecánico de un motor de corriente directa que puede representarse con las siguientes ecuaciones:

$$u(t) = L\frac{di(t)}{dt} + Ri(t) + e_b(t) \qquad \text{[Ec. 4.82]}$$

$$\tau_m(t) = J\frac{d^2\theta m(t)}{dt^2} + \tau_l(t) + \tau_l(t) \qquad \text{[Ec. 4.83]}$$

donde *u*(*t*) es la tensión eléctrica aplicada al motor, *i*(*t*) la corriente eléctrica, $\theta_m(t)$ la posición angular del eje del motor, $e_b(t)$ la fuerza electromotriz, $\tau_m(t)$ el par del motor, $\tau_l(t)$ el par visto desde el eje del motor y $\tau(f)$ el par de la fricción. Los parámetros *R*, *L* y *J* representan la resistencia interna del motor, la inductancia del motor y el momento de inercia del rotor, respectivamente.

Figura 138. **Modelo de motor de corriente directa**



Fuente: elaboración propia, empleando Inkscape.

En este modelo el motor satisface las siguientes ecuaciones de acople electromecánico:

$$t_m(t) = k_m i(t) \quad \text{[Ec. 4.84]}$$

$$e_b(t) = k_b \dot{\theta}_m(t) \quad \text{[Ec. 4.85]}$$

Donde k_m es la constante de par y k_b la constante de la fuerza electromotriz.

El modelo clásico de la fricción consta de la superposición de tres componentes, la torsión seca o fricción de Coloumb τ_{fC}, el par de fricción viscosa $\tau_f V$ y el par de fricción estática τ_{fs}, por lo que es conocido como modelo CVS.

$$\tau_f(t) = \tau_{fC}(t) + \tau_{fv}(t) + \tau_{fS} \quad \text{[Ec. 4.86]}$$

Respectivamente cada una de las fricciones del modelo CVS pueden modelarse de la siguiente manera:

$$\tau_{fV} = B\dot{\theta}(t) \quad \text{[Ec. 4.87]}$$

$$\tau_{fC} = \tau_C sgn\left(\dot{\theta}_m(t)\right) \quad \text{[Ec. 4.88]}$$

$$\tau_{fS} = \begin{cases} \tau_e(t), & |\tau_e(t)| < \tau_{S,} \quad \dot{\theta}_m(t) = \ddot{\theta} = 0 \\ \tau_S sgn\big(\tau_m(t)\big), & |\tau_e(t)| < \tau_{S,} \ \dot{\theta}_m(t) = 0, \ddot{\theta} \neq 0 \end{cases} \quad \text{[Ec. 4.89]}$$

Y donde $\tau_e(t)$ representa el par externo, $\tau_e(t) = \tau_m(t) - \tau_l(t) - J\ddot{\theta}_m(t)$

Los parámetros (B, τ_C, τ_S) representan las constantes de fricción viscosa, de Coulomb y estática respectivamente y la función siguiente: $\mathbb{R} \to \mathbb{R}$, es la función de signo definida como:

$$sgn(x) = \frac{x}{|x|} \qquad \text{[Ec. 4.90]}$$

Al adecuar el par de la fricción a las ecuaciones del motor.

$$u(t) = L\frac{di(t)}{dt} + Ri(t) + k_b\frac{d\theta_m(t)}{dt} \qquad \text{[Ec. 4.91]}$$

$$k_m i(t) = J\frac{d^2m(t)}{dt^2} + B\frac{d\theta_m(t)}{dt} + \tau_c(t) \qquad \text{[Ec. 4.92]}$$

Un motor de corriente continua es posible verlo como un sistema cuya entrada es el voltaje entregado entre sus terminales y la salida el ángulo que este ha girado a un tiempo *t,* si se consideran las condiciones iniciales del sistema de ecuaciones diferenciales que modelan motores iguales a cero y utilizando la transformada de Laplace se obtiene.

$$U(s) = (Ls + R)I(s) + k_b s\Theta_m(s) \qquad \text{[Ec. 4.93]}$$

$$k_m I(s) = s(Js + B)\Theta_m(s) + T_c(s) \qquad \text{[Ec. 4.94]}$$

Al despejar $I(s)$ en ambas ecuaciones es posible realizar una substitución:

$$k_m U(s) = s\big((Ls + R)(Js + B) + k_b k_m\big)\Theta_m(s) + (Ls + R)T_c(s) \qquad \text{[Ec. 4.95]}$$

Considérese la situación donde los efectos debidos al juego de engranajes y cargas externas sea cero, donde el motor únicamente carga con su propia inercia. La función de transferencia $G_u(s)$ para dicha situación estaría dada por:

$$G_u(s) = \frac{\Theta_u(s)}{U(s)} = \frac{k_m}{s\big((Ls+R)(Js+B)+k_b k_m\big)} \qquad \text{[Ec. 4.96]}$$

Donde $\Theta_u(s)$ es la representación en el dominio de Laplace del ángulo Θ_u en el eje del motor sin carga alguna conectada.

Por otro lado, considérese la situación donde el voltaje de entrada a las terminales del circuito es cero. Sea $\Theta_{\tau c}$ la representación en el dominio de Laplace del ángulo del motor $\Theta_{\tau C}$, generado por una torsión τ_C en el sistema, sea G_{Tc} la función de transferencia que representa el sistema si la fuente conectada entre las terminales del motor forma un corto.

$$G_{T_C} = \frac{\Theta_{\tau c}}{T_c} = -\frac{Ls+R}{s\big((Ls+R)(Js+B)+k_m k_b\big)} = -\frac{Ls+R}{k_m} G_u(s) \qquad \text{[Ec. 4.97]}$$

Al sumar los efectos del ángulo $\theta_{\tau c}$ del giro de la carga y el ángulo μ_u, debido al funcionamiento del motor sin carga, se obtiene como resultado una solución para θ_m.

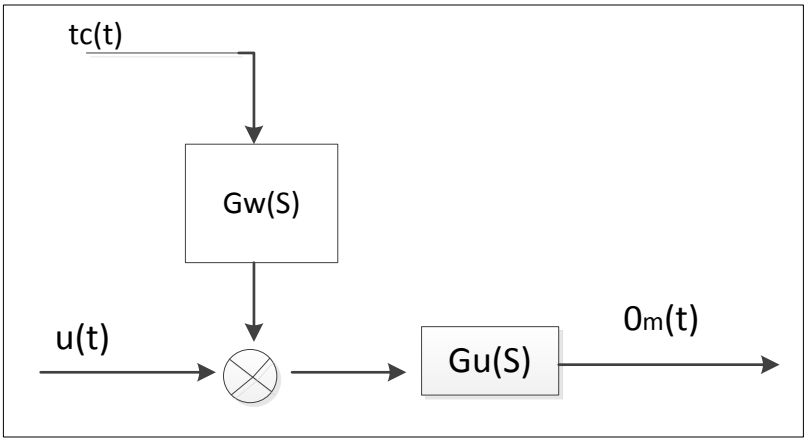
$$\theta_m(t) = \theta_u(t) + \theta_{\tau_c}(t) \qquad \text{[Ec. 4.98]}$$

Dicho razonamiento tiene una explicación más formal usando el principio de superposición. Considerando las condiciones iniciales nulas es posible escribir la salida del motor en la forma.

$$\Theta_m(t) = G_u\big(U(s) + W(s)\big) \qquad \text{[Ec. 4.99]}$$

Donde $W(s) = -G_\omega(s)T_c\,(s)$ y $G_\omega(s) = \frac{L_s+R}{k_m}$. La siguiente figura muestra la representación en diagrama de bloques del motor visto como un sistema.

Figura 139. **Diagrama de bloques de ecuación 4,99**



Fuente: elaboración propia, empleando Inkscape.

4.5. Periféricos útiles

Dado que un sistema físicocibernético integra computación y dinámicas físicas, los mecanismos que soportan la interacción con el mundo exterior son indispensables en cualquier diseño.

4.5.1. Interfaz de ADC

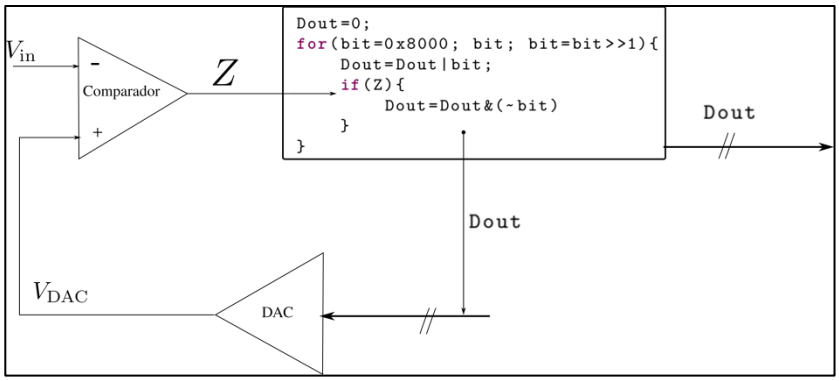
Un ADC (siglas en inglés de *analog to digital converter*) convierte una señal analógica proveniente del entorno en una señal digital discreta y se entrega a una unidad de procesamiento; en otras palabras, cuantiza una señal analógica. Puede ser visto como un transductor de voltaje a información.

Este dispositivo se encuentra comúnmente embebido en la mayoría de Microcontroladores y representa una interfaz entre otros sensores (analógicos) y el procesador.

Por la naturaleza de las señales con las que trabaja, un ADC solo puede generar 2^n valores diferentes para un rango continuo cerrado de voltajes, y en la gran mayoría bajo una función afín(con saturación). Además, la señal generada será discreta en el tiempo, con un periodo de muestreo T_s. Siendo la cantidad de bits *n* y el período de muestreo T_s, los parámetros más importantes y generales en todos los dispositivos ADC. En muchos ADC, es posible cambiar sus valores de saturación, es decir si el módulo ADC es capaz de convertir una señal de voltaje $v(t) \in [L, H]$ a valores discretos, los valores *L* y *H* pueden ser ajustados configurando el módulo ADC.

En el microcontrolador TM4C123GH6 PM posee dos módulos de ADC de 12 bits capaces de muestrear hasta 2 Msps (siglas de *mega samples per second*, indicando 1 millón de muestras tomadas en un segundo), los cuales utilizan la técnica de aproximaciones sucesivas, que se puede apreciar en la figura 140.

Figura 140. **Aproximaciones sucesivas**



Fuente: elaboración propia, inspirada en MOC de VALVANO, Jonathan y YERRABALLI, Ramesh. *Embedded Systems, Shape the World.* https://learningsciences.utexas.edu/innovators/jon-valvano-and-ramesh-yerraballi. Consulta: octubre 2015.

Para entender este método hay que considerar el algoritmo descrito en el recuadro. Como se ha tratado, se puede escribir cualquier número natural como una suma de potencias de dos, o potencias cualquier otro número entero, pero el caso de interés es el número dos, debido a que mediante un número conjunto finito de comparaciones entre dos valores se puede expresar cualquier número.

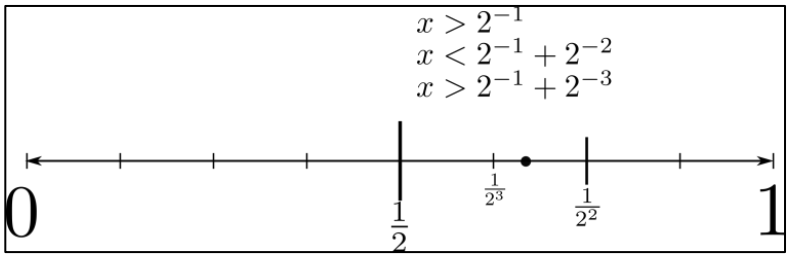
En el algoritmo descrito en la figura 143, una entrada V_{in} a un comparador con una variable que se actualiza en cada iteración del ciclo **for**, el cual pone en alto uno a uno cada bit de la variable llamada bit dejando los otros 11 restantes en bajo. Es evidente que cualquier valor en V_{in} mayor a 2^{12}-1, provocará al terminar todas las iteraciones del ciclo Dout tendrá un valor 0xFFF, ya que solo se irá acumulando y jamás el bit *Z* hará que se entre a la condicional. Considérese el caso puntual que V_{in} = 4062, el cual puede ser expresado en un sistema de numeración hexadecimal como 0xFDE.

Para la primera iteración 0xFDE > 0x800, por lo que el bit *Z* es falso y no entra a la condicional y se pone en alto elMSB de la variable Dout tomando así el valor 0x800, para la segunda iteración 0xFDE>0xC00, tomando Dout el valor 0xC00, y así Dout irá acumulando los MSB en alto hasta encontrar que 0xFDE<0xFE0 provocando que *Z* tome un valor en alto y con ello se pone en bajo específicamente el bit que hace esto cierto y la parte del algoritmo Dout=Dout&(~bit) provoca que específicamente este bit se coloque en bajo provocando que Dout no cambie su valor. Luego para la siguiente iteración, se irán poniendo en alto todos los bits consecutivos hasta encontrar alguna situación que haga cierta la condición V_{in} <Dout y de esa forma evitar que este último cambie su valor. Para este caso será cuando Dout=0xFDF regresando el valor Dout=0xFDE.

Una analogía de este método puede verse que al ir partiendo en mitades una recta, se puede aproximar la posición de un punto, sumando solo los

segmentos necesarios para llegar a él, tal como se muestra en la figura 141. Las divisiones se pueden formar cuantas veces se desee, sin embargo, en la práctica esto es imposible ya que para poder aproximar cualquier punto de la recta mediante divisiones serían necesarias infinitas de ellas.

Figura 141. **Bisección de una recta para aproximar la posición de un punto**

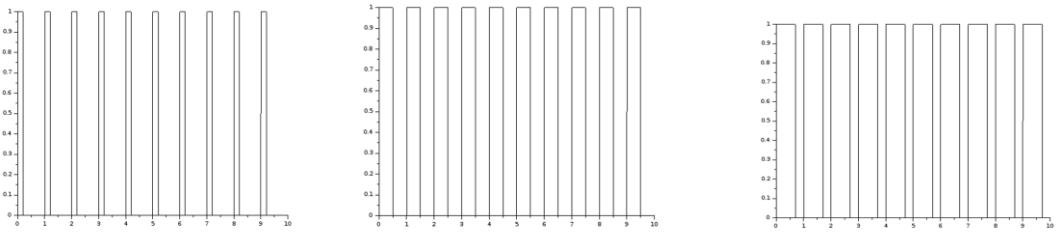


Fuente: elaboración propia, empleando Inkscape.

4.5.2. Módulo de PWM

Un módulo de PWM (siglas en inglés de *pulse-width modulation*) es un periférico, que genera una señal cuadrada (señal con solamente dos estados de voltaje, alto y bajo) a determinada frecuencia, con la propiedad de cambiar el porcentaje del período que la señal de voltaje pasa en alto, este porcentaje se conoce como ciclo de trabajo y en la literatura en inglés como *duty cycle*. Un ciclo de trabajo está al 100 % el voltaje siempre estará en alto. Si el ciclo de trabajo está al 0 % el voltaje siempre estará en bajo. En la figura 142 se observan varias señales de PWM, todas de ellas a una frecuencia de 1 Hz, pero con diferentes ciclos de trabajo.

Figura 142. **Señales de PWM a diferentes ciclos de trabajo**



(a) Ciclo de Trabajo 20%. (b) Ciclo de Trabajo 50%. (c) Ciclo de Trabajo 70%.

Fuente: elaboración propia, empleando SCILAB.

La forma más común de modificar el ancho de pulso, o ciclo de trabajo, es variando algún parámetro interno propio del periférico de PWM correspondiente. La cantidad de anchos de pulsos posibles de obtener depende de la resolución del periférico utilizado, la resolución se encuentra medida en número de bits. Siendo posible a una mayor resolución generar una mayor cantidad de ciclos de trabajo. Muchos microcontroladores proveen módulos de PWM, que por lo general se encuentran mapeados en la memoria, lo que permite modificar el parámetro del ciclo de trabajo modificando direcciones de memoria.

La técnica de variación de ancho de pulso, PWM, es poco utilizada en transmisión de información, sin embargo, resulta una forma efectiva para la transmisión de potencia variante en el tiempo a determinados sistemas. Cualquier sistema cuya respuesta a cambios de corriente o voltaje es lenta comparado a la frecuencia de la señal de PWM es candidato a ser controlado vía PWM.

Considérese que una onda periódica cuadrada $v(t)$, con período T , valor mínimo $V_{mín}$, valor máximo $V_{máx}$ y un ciclo de trabajo p, expresado como un número real entre 0 y 1; por tanto el valor promedio v esta dado:

$$\bar{v} = \frac{1}{T}\int_0^{pT} f(t)dt \qquad \text{[Ec. 4.100]}$$

Dado que $f(t)$ es un pulso cuadrado, se tiene que $f(t)$ 0 $V_{máx}$ para $0 < t < pT$ y $f(t) = V_{mín}$ para $pT < t < T$. Entonces la integral puede ser evaluada de la siguiente manera:

$$\bar{v} = \frac{1}{T}\left(\int_0^{pT} V_{máx}dt + \int_{pT}^{t} V_{mín}dt\right) \qquad \text{[Ec. 4.101]}$$

$$= \frac{1}{T}(pTV_{máx} + T(1-p)V_{mín}) \qquad \text{[Ec. 4.102]}$$

$$= pV_{máx} + (1-p)V_{mín} \qquad \text{[Ec. 4.103]}$$

Es común encontrar en los Microcontroladores que $y_{mín} = 0$ entonces se tiene que $y = py_{máx}$. Un sistema que no tiene capacidad de reacción a altas frecuencias es candidato a que una señal de PWM sirva para regular potencia, y es posible verlo expresando la función $f(t)$ como una serie de Fourier. Por facilidad considérese a $V_{mín} = 0$, entonces la serie de Fourier de la función $f(t)$ tiene la siguiente forma:

$$f(t) = pV_{máx} + \sum_{n=1}^{\infty} A_n \sin\left(\frac{2\pi nt}{T} + \emptyset n\right) \qquad \text{[Ec. 4.104]}$$

Donde $A_n = \frac{\sqrt{2}}{\pi n}V_{máx}$ y $Á_n = \frac{\pi}{2}2\ \pi np\ _2$. Si se tiene un sistema que no es capaz de reaccionar a altas frecuencias, y el valor de T es pequeño comparado con la

capacidad de reacción del sistema a controlar, de la ecuación 4.104 se puede ver que el término con mayor relevancia será $pV_{m´ax}$.

4.5.3. Protocolo de comunicación

Los Microcontroladores necesitan de periféricos especiales que puedan comunicarse con dispositivos mediante protocolos digitales discretos, los cuáles son capaces de interpretar solamente un número conjunto finito de valores en tiempos discretos de una señal óptica o eléctrica. Muchos sensores se comunican con las unidades de procesamiento utilizando protocolos.

Un protocolo es un conjunto de reglas, tanto físicas como lógicas que regulan la forma en que se entrega y envía la información de un punto a otro. Existen muchos protocolos de comunicación y básicamente aquellos con los que interaccionan directamente las unidades de procesamiento y los sensores digitales se pueden clasificar como paralelos y seriales.

Los protocolos en paralelo son aquellos que necesitan un canal de comunicación por cada bit que se está transmitiendo de la palabra definida en el protocolo. Un ejemplo de ello es los puertos LPT1 de las impresoras antiguas o los puertos IDE que se conectaban a los discos duros de las computadoras. La ventaja de estos protocolos es que pueden tomar instantáneamente los datos que se encuentran en el medio de transmisión.

En los últimos años es más frecuente encontrarse con protocolos seriales, los cuales necesitan solo un canal de comunicación para transmitir todos los bits de una palabra definida en el protocolo. Para ello es necesario definir un conjunto de reglas que permitan que la transmisión de la información entre dos dispositivos se lleve a cabo.

Los protocolos seriales pueden clasificarse en dos tipos: síncronos y asíncronos. Un protocolo síncrono es aquel que necesita un canal independiente de la comunicación que provea a ambos dispositivos una señal de reloj para que los datos transmitidos vayan en sincronía con la señal de reloj, y de esta forma exista una referencia del tiempo que toma una señal transmitir una unidad mínima de información.

Un protocolo asíncrono es aquel que no necesita de una señal de reloj para que los dispositivos puedan distinguir una unidad mínima de información de otra y sincronizarse entre sí.

El tema de protocolos de comunicación es muy extenso y su teoría compleja y relacionada con la teoría de la información. Este tema puede variar ampliamente de una marca a otra, ya que existen empresas que desarrollan protocolos privativos que no pueden ser accedidos por dispositivos de otras marcas, por lo que el presente trabajo se limita a describir algunos protocolos comunes de encontrar en sensores y actuadores digitales, con los cuales transmiten y reciben información de señales físicas con las que interaccionan:

- Protocolo UART: siglas en inglés de *universal asynchronous receiver/transmitter*, que traducido al español significa protocolo universal de transmisión/recepción asíncrona, y se trata de un protocolo serial y como su nombre lo indica asíncrono, por lo que no necesita un canal dedicado para transmitir una señal de reloj para sincronizar la información. El título de universal significa que tanto el formato como la velocidad de transmisión pueden ser configurables. Un dispositivo UART es un circuito integrado usado para comunicación serial entre dispositivos corriendo bajo los mismos protocolos. El acondicionamiento de la señal eléctrica es controlado por un circuito externo, y este puede hacer viable

el uso de diversos dispositivos corriendo protocolos RS-232 o RS-485. Un dispositivo utilizando un protocolo UART para transmitir información es capaz de tomar un conjunto de bytes y transmitir individualmente cada uno de los bits que la información posee.

El dispositivo que se encarga de la recepción de la información es capaz de realizar el proceso inverso, transformar cada uno de los bits, que han llegado individualmente y agruparlos correctamente para reconstruir la información transmitida. La forma en que se segmenta la información en este protocolo es agregando un bit de inicio para marcar el inicio de la trama y luego enviar bit por bit hasta un número indicado por el usuario o programador de la aplicación, y al final un bit de fin. Adicionalmente se pueden agregar bits que realicen funciones específicas como el bit de paridad que permite saber si una trama de datos ha llegado corrupta.

- Las tasas de transferencias usualmente se miden en baudios, que deben ser iguales en el receptor y transmisor para que estos puedan sincronizarse y comunicarse. Si las tasas de transferencia y recepción son diferentes entre dispositivos comunicándose por este protocolo el receptor no podrá recibir la información correcta.

- Protocolo SPI: siglas en inglés para *serial peripheral interface*, es un protocolo de comunicación serial síncrono utilizado para cortas distancias, principalmente en sistemas embebidos. La comunicación es bidireccional entre dos dispositivos, uno llamado maestro y otro llamado esclavo. Ambos dispositivos pueden recibir y transmitir información, la diferencia radica que solamente el maestro puede generar la señal de reloj para la sincronización y esta debe de ser soportada por el

dispositivo llamado esclavo. Este protocolo es conocido como *four-wire protocol*, ya que posee 4 canales de comunicación específicos:

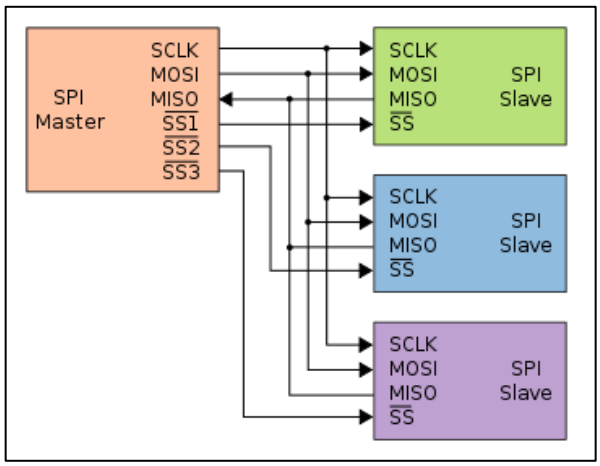
- SCLK: acrónimo para el término *serial clock* y por este canal se provee la señal de reloj necesaria para la sincronización de los datos. Por lo general es de unos cuantos mega baudios.

- MOSI: acrónimo para *master output-slave input*, y es un canal de comunicación unidireccional que siempre va del dispositivo maestro al dispositivo esclavo. En otras palabras, en este canal el dispositivo maestro puede transmitir y el dispositivo llamado esclavo puede recibir información.

- MISO: acrónimo para *master input - slave output*. Es un canal de comunicación unidireccional que siempre va del dispositivo esclavo al dispositivo maestro. En otras palabras, en este canal el dispositivo esclavo puede transmitir y el dispositivo Maestro puede recibir información.

- SS: acrónimo para *slave select,* este canal es utilizado por el dispositivo maestro para habilitar la transmisión y recepción de información en el dispositivo esclavo. Un dispositivo maestro puede tener varios de estos canales con el fin de tener varios esclavos, tal como se muestra en la figura 143; un dispositivo esclavo solamente tendrá un canal SS y estará sometido a las señales enviadas por el maestro.

Figura 143. **Dispositivo maestro controlando 3 dispositivos esclavos**



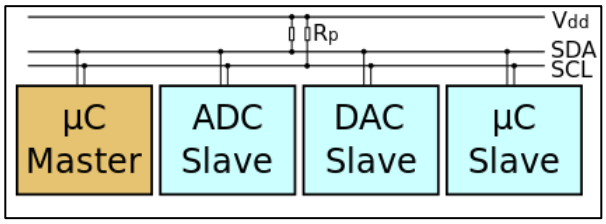
Fuente: *File:SPI three slaves daisy chained.svg*. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#/media /File:SPI_three_slaves.svg. Consulta: octubre de 2015.

- I2C: protocolo llamado *inter-integrated circuit*, pronunciado como I-cuadrado-C y es un protocolo serial multimaestro y multiesclavo desarrollado por Phillips Semiconductors (ahora NXP Semiconductors). Este protocolo solamente utiliza dos líneas de colector abierto, la línea SDA (en inglés *serial data line*) y la línea SCL (en inglés *serial clock line*), con resistencias *pull-up*.

  En este protocolo los voltajes utilizados comúnmente son 5 o 3,3 V aunque algunos sistemas permiten otros voltajes. El protocolo I2C está diseñado para 7 o 10 bits de espacio de direccionamiento el cual es utilizado por los dispositivos para reconocerse en el canal de comunicación y las velocidades se encuentran generalmente de 100 kbits/s en el modo estándar y 10 kbits/s en el modo de baja velocidad. Sin embargo, frecuencias arbitrarias son permitidas en el protocolo. Revisiones modernas del protocolo incluyen mayor cantidad

de nodos corriendo el protocolo y velocidades mayor de transmisión, 400 kbits/s en el modo de alta velocidad, 1 Mbits/s en el modo de alta velocidad plus y 3,4 Mbits en el modo de muy alta velocidad. Y número de nodos en el canal de comunicación depende del número de bits en el espacio de direccionamiento, en algunos casos alcanzando hasta 16 bits.

Figura 144. **Múltiples nodos utilizando los canales de comunicación 12C**



Fuente: *I²C*. https://en.wikipedia.org/wiki/I%C2%B2C#/media/File:I2C.svg. Consulta: octubre de 2015.

Este protocolo como se ha mencionado con anterioridad es multimaestro y multiesclavo, lo que significa que cualquier dispositivo en las líneas de comunicación puede tomar el rol de maestro o esclavo. El rol del dispositivo maestro es generar el reloj e iniciar la comunicación con los esclavos, así como llamarlos por dirección. El rol de los esclavos en la comunicación es recibir la señal de reloj enviada por el maestro y responder cuando se es llamado por dirección por el maestro. En el proceso de comunicación entre dispositivos I2C existen 4 modos potenciales de operación. En la mayoría de los casos los dispositivos utilizan solamente un rol (maestro o esclavo) en sus dos modos:

- *Maestro envía.* El dispositivo asumiendo el rol de maestro envía los datos.
- *Maestro recibe.* El dispositivo asumiendo el rol de maestro recibe datos de otro dispositivo.
- *Esclavo envía.* El dispositivo asumiendo el rol de esclavo envía datos al maestro.
- *Esclavo recibe.* El dispositivo asumiendo el rol de esclavo recibe datos del maestro.

En el proceso de transmisión, al inicio el maestro se encuentra en modo maestro envía y transmite un bits de inicio seguido de 7 bits de dirección ubicando al esclavo con el que se desea iniciar la comunicación, seguidos de un bits indicando si desea leer o escribir en el esclavo.

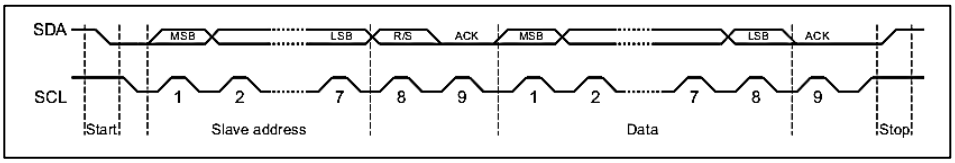
Si el esclavo existe en el canal, entonces responderá para esa dirección con un bit de acuse de recibido (*acknowledged*). El maestro continúa en su modo de envía o recibe (de acuerdo al bit que se ha transmitido) y el esclavo asume su modo complementario.

Los bytes de dirección y de información son enviados en orden empezando por el bit más significativo al menos significativo. El bit de inicio se indica mediante una transición de alto a bajo en el canal SDA con el canal SCL en alto. El bit de fin de trama se indica por una transición de bajo a alto en el canal SDA con el canal SCL en alto. El resto de las transiciones son llevadas a cabo con el canal SCL en bajo.

Si el maestro desea escribir al esclavo entonces él repetidamente envía un byte con el esclavo enviando el bit de acuse de recibido. Si el maestro desea leer del esclavo entonces repetidamente recibe un byte del

esclavo, y el maestro envía un bit de acuse de recibido después de cada byte excepto el último. En esta situación el maestro envía un bit de fin o bien un bit de inicio si desea mantener el control del canal y realizar una transferencia.

Figura 145. **Transferencia completa de información en 12C**



Fuente: *Tiva TM TM4C123GH6PM Microcontroller Datasheet*. http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: octubre de 2015.

4.6. Prácticas de laboratorio. Sistemas físicocibernéticos

A continuación, se proponen tres ejercicios prácticos a realizar para poder consolidar la información presentada en las secciones anteriores. El primero consiste en un control por lazo abierto de un motor paso a paso; utilizando una unidad digital se controlará sin retroalimentación la posición del motor, variando la corriente en las bobinas es posible manipular la resolución de las posiciones el que puede moverse el motor. El segundo consiste en la lectura de un sensor sumamente propenso a ruido y a la aplicación de un filtro digital a las mediciones tomadas. Como tercer ejercicio, se propone una simulación de un control de lazo cerrado para un motor de corriente continua, con una planta de control PID y el uso de un servomecanismo para observar los efectos de un lazo cerrado.

4.6.1. Lazo abierto. Control por micropaso de *stepper* bipolar

Para realizar esta práctica es necesario conseguir los siguientes elementos:

- Un motor paso a paso bipolar
- Dos puentes H, con control de activación (bit de *enable*)
- Un microcontrolador con dos módulos de PWM independientes
- Una fuente de poder acorde a las especificaciones del motor bipolar

Al tener un motor *stepper* bipolar, este puede ser de imanes permanentes, de reluctancia o bien un híbrido. En la práctica solo se desea saber que secuencia de polarización es correcta para el motor. Un motor bipolar fuera de su armazón electromecánica tiene cuatro cables que conectan a las bobinas del motor con el exterior.

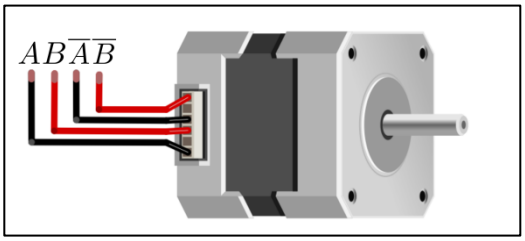
En algunos casos es posible encontrar algunos cables midiendo continuidad entre ellos para ubicar qué cables forman una bobina. En otros casos esto no es posible por la construcción interna del motor, y se pude realizar lo siguiente para encontrar una secuencia de paso para el motor.

Primero se conectan los cuatro cables al voltaje más negativo de la fuente de poder. Luego se toma arbitrariamente alguno de ellos, al cual momentáneamente se le llamará *A*, se coloca en el voltaje más positivo de la fuente y se registra el sentido al que se mueve el motor. Se vuelve a poner *A* en el voltaje más negativo, y se toma arbitrariamente un cable y se coloca en el voltaje más positivo. En caso que el motor gire en la misma dirección que la vez anterior se le llamará a este nuevo cable *B*, en caso que gire al contrario se omite el nombre del cable anterior y a este nuevo cable se le llama *A*. Se mueve

el eje del motor para dejarlo en una posición arbitraria y se repite el procedimiento para los otros dos cables restantes hasta encontrar una secuencia correcta *A* - *B*, que genere un movimiento del motor en la misma dirección dos veces consecutivas.

Una vez que se ha encontrado una pareja de cables *A*-*B*, se prueba quién de los dos cables siguientes podría formar $\bar{A}$, el cual generará un giro en la misma dirección que las veces anteriores.

Figura 146. **Motor *stepper* bipolar**



Fuente: *Fritzing*. http://fritzing.org/home/. Consulta: octubre de 2015.

Con un poco de orden, es posible encontrarla probando todas las permutaciones posibles, ya que solamente son seis.

Tabla LXXXV. **Ordenes de los cables para un paso completo**

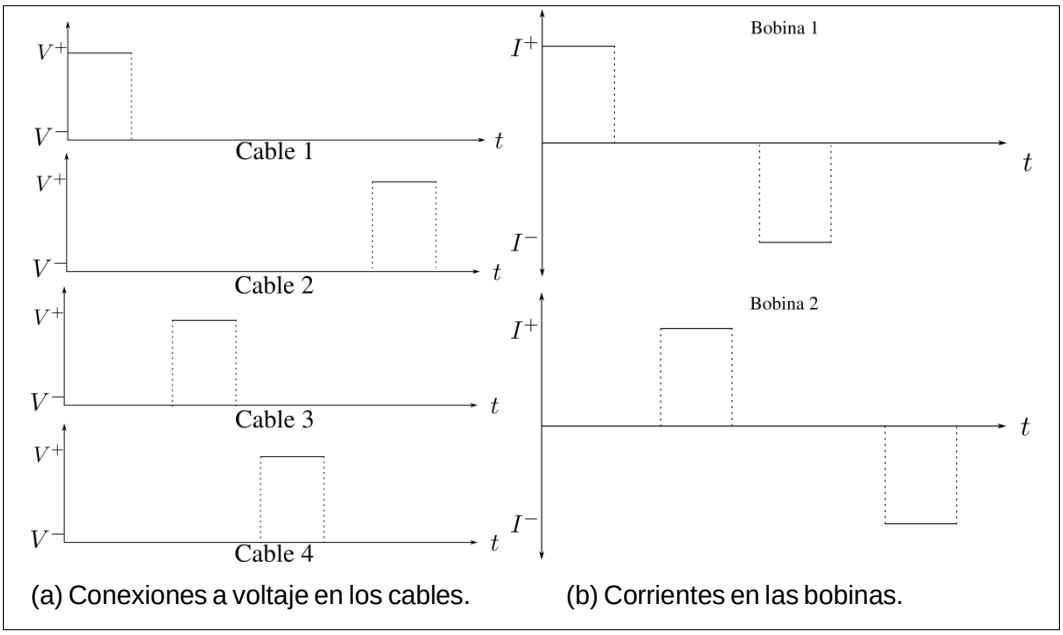
Núm.	*A*	*B*	$\bar{A}$	$\bar{B}$
1	Cable 1	Cable 2	Cable 3	Cable 4
2	Cable 1	Cable 2	Cable 4	Cable 3
3	Cable 1	Cable 3	Cable 2	Cable 4
4	Cable 1	Cable 3	Cable 4	Cable 2
5	Cable 1	Cable 4	Cable 2	Cable 3
6	Cable 1	Cable 4	Cable 3	Cable 2

Fuente: elaboración propia.

Para hacerlo por fuerza bruta solo hay que colocar cada uno los cables 1, 2, 3 y 4 al voltaje más positivo mientras el resto está en el más negativo, iterativamente en los seis órdenes mostrados en la tabla LV. Y al encontrar el orden correcto de los cables (aquel que logre mover el motor siempre en una dirección), se nombran los cables *A*, *B*, $\bar{A}$ y $\bar{B}$ respectivamente.

Por ejemplo, si la secuencia que funcionó correctamente fue cable 1, cable 3, cable 4, cable 2. El cable 1 será *A* y el cable 4 será $\bar{A}$, los cuales formaran las conexiones de una bobina en el modelo mostrado con anterioridad. El cable 3 será *B* y el cable 2 será $\bar{B}$ los cuales formaran las conexiones de una segunda bobina. Al encontrar las secuencias de esta forma se realiza un control de onda al motor.

Figura 147. **Secuencia correcta**



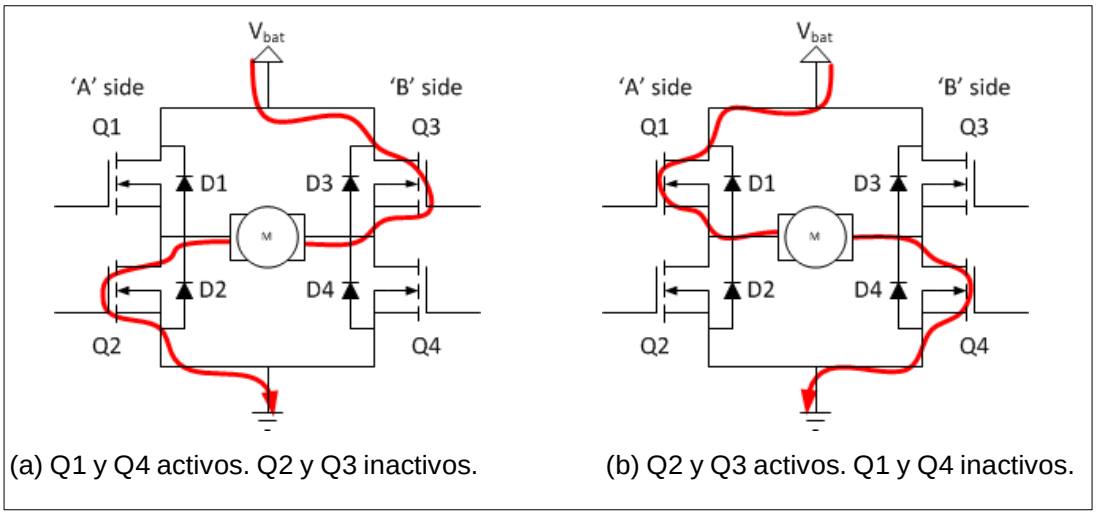
(a) Conexiones a voltaje en los cables. (b) Corrientes en las bobinas.

Fuente: elaboración propia, empleando Inkscape.

Una vez que se ha encontrado la secuencia correcta de los pasos se conectan cada una de las bobinas a un puente H, uno por bobina. Un puente H es un arreglo de cuatro transistores que permite manipular la dirección de la corriente en una bobina manipulando la configuración en el puente H tal como se muestra en la figura 148.

Al activar dos transistores y desactivar los otros dos fluye la corriente a través de la bobina en determinada dirección. Los diodos que se ven en la figura son necesarios para descargar las corrientes de retorno que se generan al operar el motor.

Figura 148. **Puente H para manipulación de la dirección de la corriente**



(a) Q1 y Q4 activos. Q2 y Q3 inactivos. (b) Q2 y Q3 activos. Q1 y Q4 inactivos.

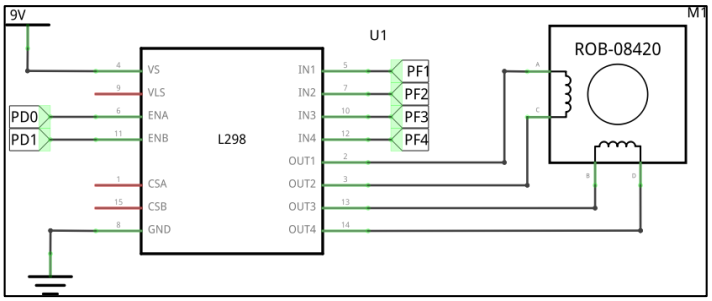
Fuente: *Circuitos modulares*. http://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/. Consulta: octubre de 2015.

El integrado L298N provee dos puentes H. Cada puente H posee 3 pines, In1, In2 y Enable. Los primeros dos permiten el flujo de corriente en una dirección u otra al tener un voltaje en alto respectivamente. El tercer pin se utiliza para el control del motor y permite o evita el flujo de corriente a través de la carga

conectada. Esta última característica es esencial para poder utilizar el control por micropaso en un motor *stepper*, utilizando una señal digital.

Como referencia, el control del motor por micropaso se hará utilizando el microcontrolador tm4c123gh6pm, como compilador C y la librería TivaWare para el control de los periféricos. El puente H a utilizar en es un integrado L298N utilizando una placa de desarrollo la cual ya provee los diodos necesarios para las corrientes de retorno, así como pines y borneras para fácil conexión. Si se cuenta con una placa de desarrollo, y se tiene la secuencia correcta se debe conectar el motor bipolar a la placa tal como se muestra en la figura 149.

Figura 149. **Conexión a *stepper* de puente H L298N**

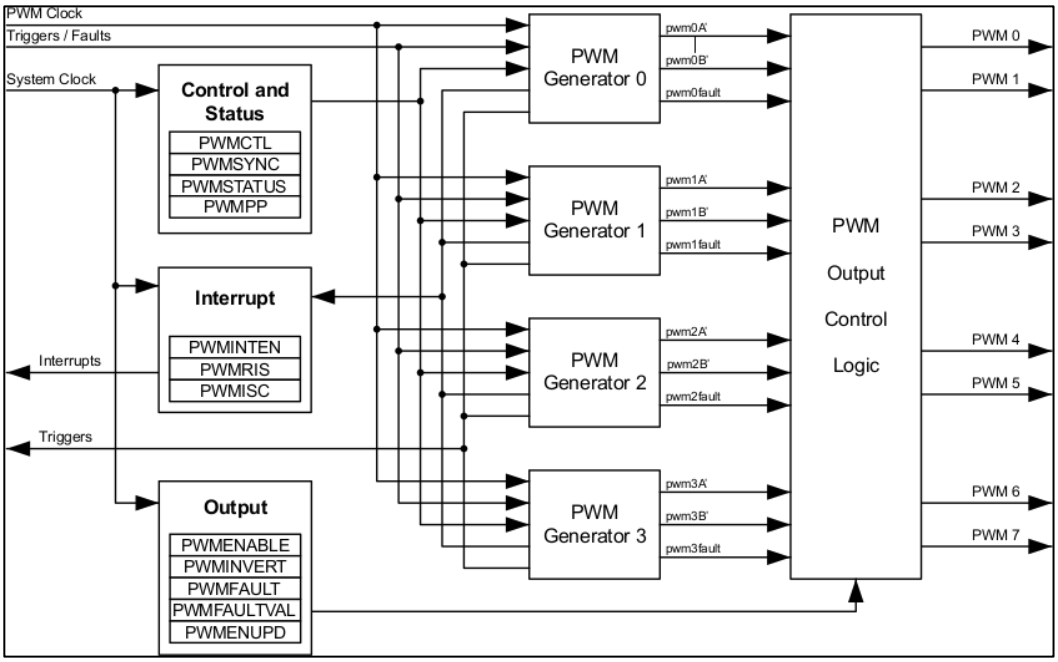


Fuente: *Fritzing*. http://fritzing.org/home/. Consulta: octubre de 2015.

En el caso de no tener una placa de desarrollo se debe de configurar el puente H de acuerdo a la hoja de datos, sin olvidar los diodos de descarga para corrientes inversas. Como una breve referencia para explicar de mejor manera el código en C a utilizar, la figura 150 muestra el diagrama de bloques del módulo de PWM en el microcontrolador TM4C123GH6PM, el cual cuenta con dos módulos de PWM y cada uno de ellos a su vez con cuatro generadores. Cada generador es capaz de producir 2 señales de PWM, cada uno de ellos

con la capacidad de variar el ancho de pulso independientemente. Es posible aplicar un prescaler de 64 veces a la señal proveniente del sistema para poder generar de esta manera utilizando un contador de 16 bits señales de baja frecuencia, usando la misma señal proveniente del reloj principal. El reloj del sistema sirve como el reloj usado para mover registros de las configuraciones, control, estado e interrupciones. La señal de reloj de PWM que se ve en la figura, es el reloj utilizado por los generadores para realizar el conteo.

Figura 150. **Módulo de PWM I**

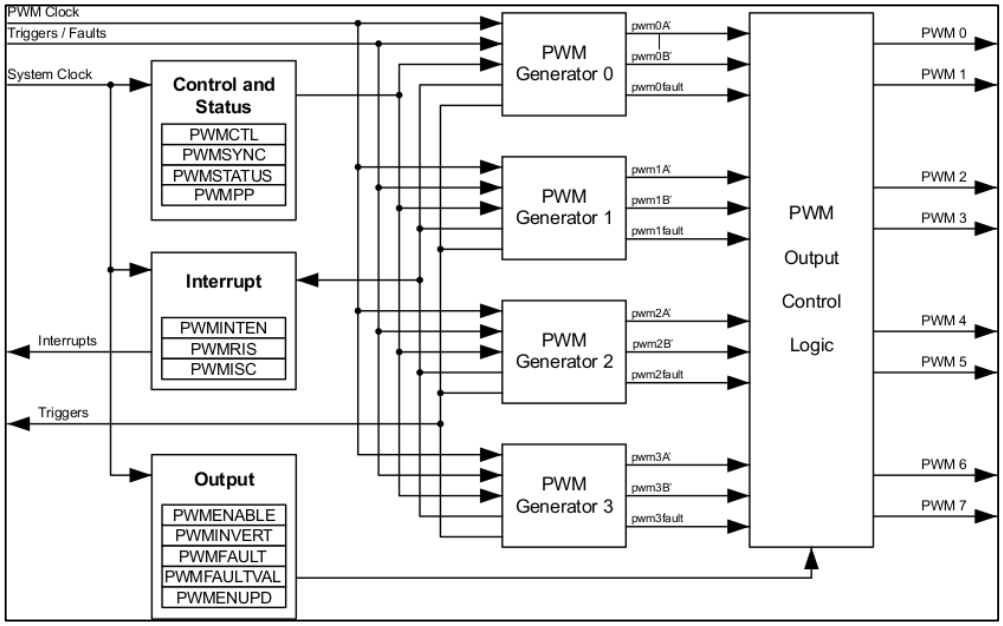


Fuente: *Tiva™ TM4C123GH6PM Microcontroller Datasheet*.
http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: octubre 2015.

Cada generador tiene registros como los que se muestran en el diagrama de bloques de la figura 151, los cuales llevan un conteo de los ciclos de reloj de PWM que han transcurrido y se almacenan en el registro PWMnCOUNT, dependiendo de la configuración del generador, DOWN o UP/DOWN, el módulo al

alcanzar el valor colocado en el registro PWMnLOAD, reiniciará el conteo o cambiará la dirección de realizarlo.

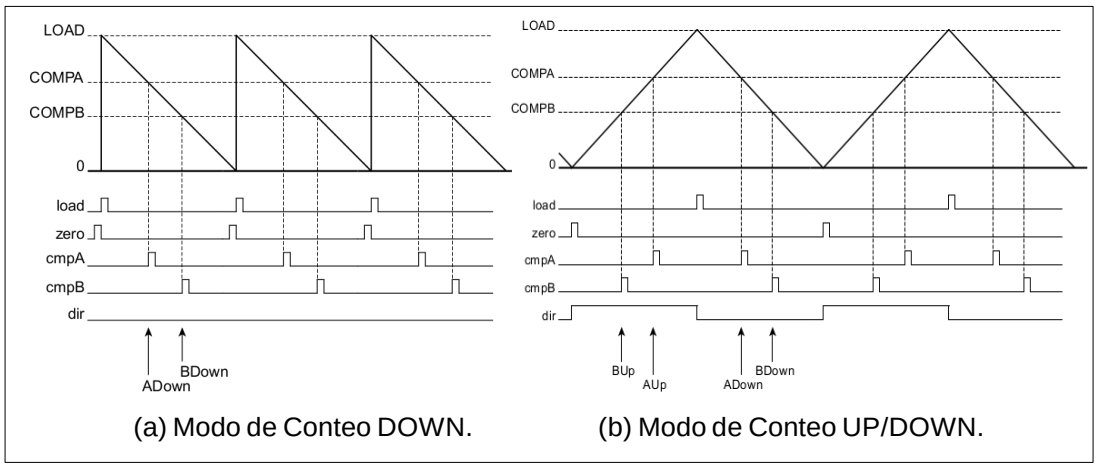
Figura 151. **Módulo de PWM II**



Fuente: *Tiva™ TM4C123GH6PM Microcontroller Datasheet*.
http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: octubre 2015.

La figura 152 muestra el comportamiento del valor del registro PWMnCOUNT dependiendo del modo de conteo, las líneas punteadas indican la forma que tendría la onda de cada una de las señales PWMA y PWMB producidas por el generador. El valor del registro PWMnCOMPA y el valor del registro PWMnCountB indican cuando cambiar de estado durante un ciclo.

Figura 152. **Modos de funcionamiento**



(a) Modo de Conteo DOWN. (b) Modo de Conteo UP/DOWN.

Fuente: *Tiva™TM4C123GH6PM Microcontroller Datasheet*.
http://www.ti.com/lit/ds/spms376e/spms376e.pdf. Consulta: octubre 2015.

El valor del registro PWMnLOAD es el número de pulsos del reloj de PWM necesarios para alcanzar el período deseado para la señal. Por ejemplo, si se activa el prescaler en el reloj de PWM, el reloj del sistema se encuentra a 80MHz, el modo de conteo se encuentra configurado como DOWN, se desea que la frecuencia de la señal PWMA en un determinado generador sea 2 500 Hz, y el ancho de pulso en alto sea un 25 % del período de la señal. El valor del reloj de PWM será el reloj principal entre 64, el cual tendrá un valor de 1,25 MHz. Por lo que para alcanzar el valor del período deseado es necesario contar 500 de estos pulsos. Como el conteo empieza en cero entonces el valor que se debe colocar en PWMnLOAD es 499. Si la señal debe de pasar el 25 % del período en alto, el valor que se debe de colocar en el registro PWMnCOMPA debe ser 124.

El valor del registro PWMnLOAD es el número de pulsos del reloj de PWM necesarios para alcanzar el período deseado para la señal. Por ejemplo, si se activa el prescaler en el reloj de PWM, el reloj del sistema se encuentra a 80 MHz, el modo de conteo se encuentra configurado como DOWN, se desea que

la frecuencia de la señal PWMA en un determinado generador sea 2500 Hz, y el ancho de pulso en alto sea un 25 % del período de la señal. El valor del reloj de PWM será el reloj principal entre 64, el cuál tendrá un valor de 1,25 MHz. Por lo que para alcanzar el valor del período deseado es necesario contar 500 de estos pulsos. Como el conteo empieza en cero entonces el valor que se debe colocar en PWMnLOAD es 499. Si la señal debe de pasar el 25 % del período en alto, el valor que se debe de colocar en el registro PWMnCOMPA debe ser 124.

Para poder lograr un control por micropaso en un motor bipolar utilizando señales digitales es necesario conocer que la bobina en el motor se opone a los cambios bruscos de corriente, una onda cuadrada de voltaje de acuerdo a su serie de Fourier se compone de una componente constante y una composición lineal de senos y cosenos, tal como la ecuación 4.104. Al entregarle la onda cuadrada a la bobina del motor las componentes de alta frecuencia serán filtradas por la bobina del motor, siendo significativa solamente la componente $pV_{máx}$, donde el ciclo de trabajo p es el porcentaje que la señal pasa en alto.

El objetivo de esta práctica es generar un control por micropaso, que consiste en generar secuencias de valores que permitan el movimiento del eje de acuerdo al diagrama de fases. El siguiente programa genera una matriz de 16£2, donde cada columna corresponde a los valores que debe tomar el comparador para la señal de PWM correspondiente a cada bobina del motor.

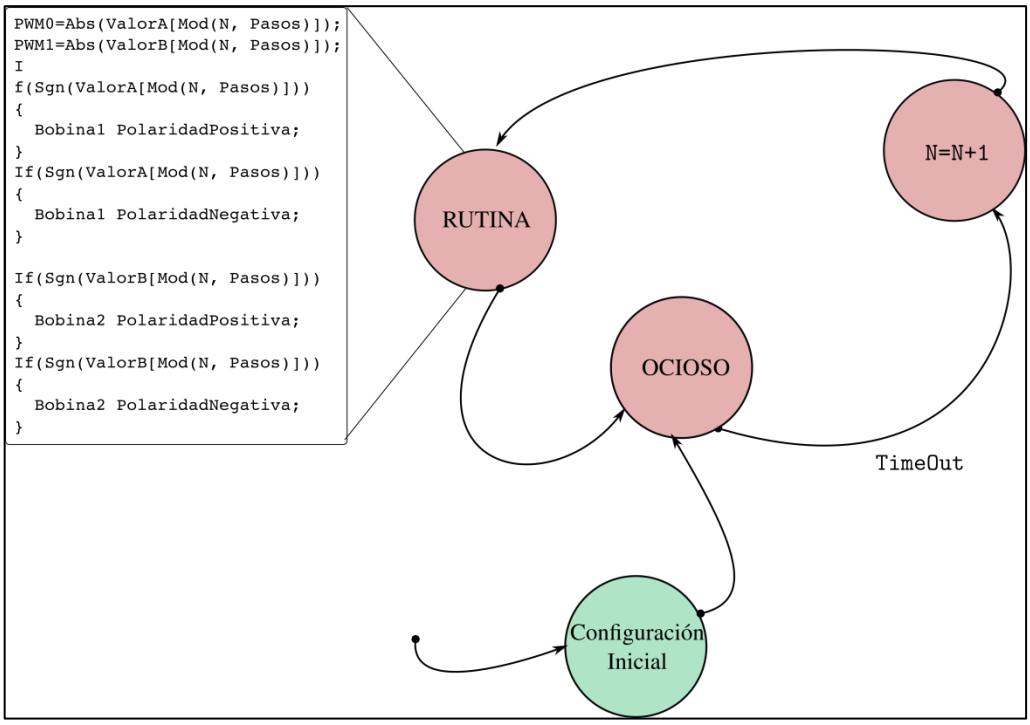
```
1 | | Steps =16;
2 | | TivaClock =80000000;
3 | | PWMfreq =2500;
4 | | PWMClock = TivaClock /64;
5 | | MaxCount =( PWMClock / PWMfreq ) – 1
6 | | N= linspace (0 ,1 , Steps );
7 | | Coils = ceil ( MaxCount *[ cos (2* %pi*N) ’, sin (2* %pi*N) ’])
8 | | disp ( Coils );
```

Cada uno de estos valores será almacenado en un arreglo, y serán leídos constantemente para asignar los valores para cada bobina, asignando un nuevo paso cada cierto tiempo. El tiempo que tardará la dinámica en dar un nuevo paso debe ser mucho mayor que el periodo de la señal de PWM, debido a que le debe de dar tiempo a la bobina de poder interpretar la señal de PWM como una componente de corriente directa $pV_{máx}$.

El arreglo contiene los valores correspondientes a ondas sinusoidales creadas en base al diagrama de fases, en el caso de tener valores negativos se invertirá la polaridad utilizando el puente H, cada cierto tiempo se avanzará un paso, el control del tiempo entre un paso y otro se llevará a cabo con temporizadores físicos propios del microcontrolador, al terminar el período del paso se incrementa en uno una variable que recorre el arreglo con los valores para poder realizar el cambio de paso.

La dinámica discreta a implementar para el control del motor por micropaso se encuentra descrita en la figura 153.

Figura 153. **Dinámica discreta de control por micropaso**



Fuente: elaboración propia, empleando Inkscape.

A continuación, se detalla el programa escrito en C para el microcontrolador TM4C123GH6PM utilizando la librería TivaWare para configurar los periféricos del microcontrolador. Primero se agregan las librerías necesarias para la declaración eficiente de variables y control de los periféricos. Además, como la librería es genérica para diversos dispositivos debe indicarse con que dispositivo se esta trabajando, al declarar la constante PART_TM$C123GH6PM las librerías sabrán el dispositivo que se está configurando con ellas.

```
# define PART_TM4C123GH6PM
# include <stdint .h>
# include <stdbool .h>
# include " inc / tm4c123gh6pm .h"
# include " inc / hw_memmap .h"
# include " inc / hw_types .h"
# include " inc / hw_gpio .h"
# include " driverlib / sysctl .h"
# include " driverlib / interrupt .h"
# include " driverlib / gpio .h"
# include " driverlib / timer .h"
# include " driverlib / pwm .h"
# include " driverlib / pin_map .h"
```

Luego se procede a declarar constantes y variables útiles durante el programa. La constante PWM_FREQUENCY será reemplazada por el precompilador por la frecuencia del reloj de PWM, la constante Stepfreq por la frecuencia asociada al período del paso y TivaClock la frecuencia del reloj principal del sistema, todas ellas en Hertz. La variable ui32Period indicará el valor al que debe contar el temporizador para lograr el período del paso.

```
/* ******** Constantes ********* */
# define PWM_FREQUENCY 2500
# define Stepfreq 2
# define TivaClock 80000000
/* ******** Variables ********* */
uint32_t ui32Period =1;
uint8_t stepN =0;
bool stepFlag = true ;
volatile uint32_t ui32Load ;
volatile uint32_t ui32PWMClock ;
```

La variable stepN lleva el control del paso, es la variable que recorre el arreglo con los valores de las corrientes entregadas a las bobinas. La variable steFlag será la que indique a la rutina principal cuando ha terminado el período del

paso y se debe pasar a un paso nuevo, esta bandera se pondrá en alto mediante una interrupción generada por el temporizador. Las variables ui32Load y ui32PWMClock, serán útiles para el ciclo de trabajo y el reloj de PWM, respectivamente.

Utilizando el programa 4.1, se pueden generar los valores que deben tomar los comparadores en los generadores PWM a una frecuencia de 2500 Hz para que el motor sea capaz de generar 16 pasos. De esta forma se declara un arreglo de 16x2 posiciones, con los valores indicados para el PWM de cada bobina.

```
# define Steps 16
int32_t Current [ Steps ][2]={{499 , 0}, {462 , 191} , {353 , 353} , {191 ,462} ,
                {0 ,499} , { -190 , 462} , { -352 , 353} , { -461 ,191} ,
                { -499 , 0}, { -461 , -190} , { -352 , -352} , { -190 , -461} ,
                {0, -499} , {191 , -461} , {353 , -352} ,{462 , -190}};
```

En la rutina principal se configura el reloj del sistema y periféricos, el reloj del sistema se configura a 80MHz y el módulo de PWM recibe una señal del reloj del sistema prescalada 64 veces.

```
// Reloj principal a 80 Mhz
SysCtlClockSet ( SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN );
// Prescaler de PWM
SysCtlPWMClockSet ( SYSCTL_PWMDIV_64 );
// Reloj a Perif é rico
SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOF );
SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOD );
// Reloj a Timer
SysCtlPeripheralEnable ( SYSCTL_PERIPH_TIMER0 );
// Reloj a PWM
SysCtlPeripheralEnable ( SYSCTL_PERIPH_PWM1 );
```

Luego se configuran los GPIO, para funcionar de acuerdo a las conexiones de la figura 152, los pines del puerto F del uno al 4 se configuran como salidas digitales para indicar la dirección del motor. Los pines cero y uno

del puerto D como salidas de PWM para poder regular las corrientes en la bobina.

```
GPIOPinTypeGPIOOutput ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 );
GPIOPinTypePWM ( GPIO_PORTD_BASE , GPIO_PIN_0 | GPIO_PIN_1 );
GPIOPinConfigure ( GPIO_PD0_M1PWM0 );
GPIOPinConfigure ( GPIO_PD1_M1PWM1 );
```

Se configura el *Timer* 0 submódulo A, para funcionar como temporizador de 32 bits de forma periódica con la frecuencia indicada por la constante Stepfreq.

```
// Configuraci ón de Timer Peri ó dico
TimerConfigure ( TIMER0_BASE , TIMER_CFG_PERIODIC );
// Configuraci ón periodo de Timer
ui32Period = ( TivaClock / Stepfreq );
TimerLoadSet ( TIMER0_BASE , TIMER_A , ui32Period -1);
```

Se configura el módulo de PWM, para trabajar a una frecuencia indicada por la constante PWM_FREQUENCY, en modo de trabajo DOWN cuya forma de generar el ancho de pulso se puede apreciar en la figura 157a.

```
ui32PWMClock = TivaClock / 64;
ui32Load = ( ui32PWMClock / PWM_FREQUENCY ) - 1;
// Configuraci ón inicial Generador 1 Modo Down y un periodo dado por PWMLoad
PWMGenConfigure ( PWM1_BASE , PWM_GEN_0 , PWM_GEN_MODE_DOWN );
PWMGenPeriodSet ( PWM1_BASE , PWM_GEN_0 , ui32Load );
```

Se le coloca el ancho de pulso en base al conteo que se realiza para lograr el periodo de PWM_FREQUENCY, el cual esta dado por la variable ui32Load, inicialmente se le coloca un ciclo de trabajo al 50 %, se habilita el funcionamiento del PWM en los pines PD0 y PD1 y se pone a funcionar el módulo de PWM.

```
// Ancho de pulso dado por output
PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_0 , ( ui32Load +1) /2 -1); // PD0
PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_1 , ( ui32Load +1) /2 -1); // PD1
// Habilita salida D0 y D1
PWMOutputState ( PWM1_BASE , PWM_OUT_0_BIT , true );// PD0
PWMOutputState ( PWM1_BASE , PWM_OUT_1_BIT , true );// PD1
// Habilita Generador 0
PWMGenEnable ( PWM1_BASE , PWM_GEN_0 );
```

Se configuran las interrupciones en el microprocesador y el temporizador. El módulo de *Timer* se configura para que las interrupciones sean generadas al terminar un conteo, y se configura para generar interrupciones, se habilitan en el procesador las interrupciones y se pone a funcionar el temporizador.

```
// Habilitar Interrupciones por TIMER0A
IntEnable ( INT_TIMER0A );
// Configura TIMER0 para generar interrupciones al terminar conteo
TimerIntEnable ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
// Sistema completo atento a interrupciones
IntMasterEnable ();
// Se pone a funcionar el TIMER0A
TimerEnable ( TIMER0_BASE , TIMER_A );
```

Es importante recordar que al suceder una interrupción entra a trabajar el vector de interrupciones (NVIC Table), por lo que se debe configurar el archivo STARTUP donde se almacenan las interrupciones a generar en un vector. En el archivo donde se almacenan las configuraciones para vector de interrupciones, se debe modificar la interrupción correspondiente al *Timer* 0, tal como se muestra en la práctica anterior:

```
IntDefaultHandler , // ADC Sequence 3
IntDefaultHandler , // Watchdog timer
Timer0AIntHandler , // Timer 0 subtimer A
IntDefaultHandler , // Timer 0 subtimer B
IntDefaultHandler , // Timer 1 subtimer A
IntDefaultHandler , // Timer 1 subtimer B
```

Se debe declarar la rutina en ese archivo como una rutina externa para que el compilador sepa que se configurará fuera del archivo STARTUP, agregando la declaración de la rutina al principio del archivo.

```
extern void Timer0AIntHandler ( void );
```

Luego de la configuración de periféricos se coloca un ciclo continuo sin fin, para albergar la dinámica discreta del control por micropaso. En el ciclo continuo se valida siempre que ya el temporizador terminara de contar el tiempo para el paso. Si ya terminó se cargan los valores absolutos en el PWM, ya que

no puede tener valores negativos, por lo que si el valor es negativo se cargará con el signo cambiado y en las bobinas se cambiarán su polaridad, acorde al signo. Los pines PF1 y PF2 serán los encargados marcar la dirección de la bobina 1 y el pin PD0 es el encargado de entregar la señal de PWM para modular su corriente. Los pines PF3 y PF4 son los encargados de la dirección de la bobina 2 y el pin PD1 de la señal PWM.

Si el bit PF1 se encuentra en alto y PF2 en bajo la corriente en la bobina, irá en una dirección, debido al comportamiento del puente H, si el bit PF2 se encuentra en bajo y PF2 en alto la corriente irá en la dirección contraria. Lo mismo sucede para los pines PF4 y PF3. Por lo que si el arreglo de pasos Current, tiene un valor negativo para la posición que se encuentra implica que la corriente debe ir al sentido contrario del valor positivo, de esa forma se revisa el signo del valor y se coloca la polaridad en la bobina.

Colocar un valor 0 en el valor de PWM puede producir un comportamiento no deseado en el motor por lo que se cambia el valor de 0 por 1, ya que esto no cambia de manera significativa el valor de la corriente y evita que el motor retroceda por una polaridad no adecuada en las bobinas. Al terminar de configurar las polaridades para cada bobina en los pines PF1, PF2, PF3 y PF4 y las señales de PWM se suma uno al valor del paso, para que al suceder la siguiente interrupción del temporizador se cargue los valores de polaridad y corriente correspondientes al siguiente paso. Ya que el vector posee valores finitos y la secuencia es periódica al alcanzar el valor máximo del vector se regresa a su posición inicial, cero.

```
1   for (;;) {
2   if( stepFlag ){
3   // Corriente en D0
4   if( Current [ stepN ][0]==0) {
5   Current [ stepN ][0]=1;
6   }
7   if( Current [ stepN ][0] >0) {
8   GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 , 2);
9   PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_0 , Current [ stepN ][0]) ;
10  }
11  else
12  {
13  GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 , 4);
14  PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_0 , -Current [ stepN ][0]) ;
15
16  }
17  // Corriente en D1
18  if( Current [ stepN ][1]==0) {
19  Current [ stepN ][1]=1;
20  }
21  if( Current [ stepN ][1] >0) {
22  GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_3 | GPIO_PIN_4 , 8);
23  PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_1 , Current [ stepN ][1]) ;
24  }
25  else
26  {
27  GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_3 | GPIO_PIN_4 , 16) ;
28  PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_1 , -Current [ stepN ][1]) ;
29  }
30  stepN ++;
31  if(stepN >= Steps )
32  { stepN =0;}
33  stepFlag = false ;
34  }
35  }
```

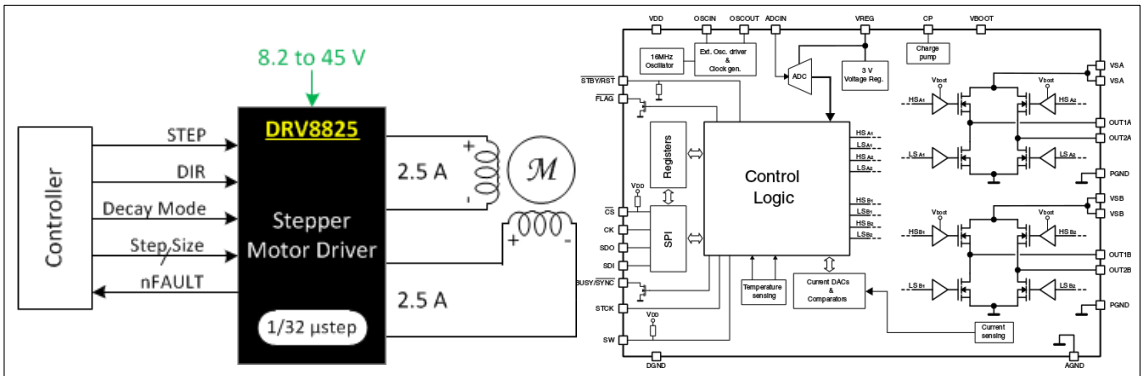
El microcontrolador debe actualizar los valores de la polaridad y PWM únicamente cuando el temporizador ha terminado el conteo. Cuando esto suceda una interrupción pondrá en alto una bandera, stepFlag, para que el programa principal sepa que debe actualizar los valores, al terminar de hacerlo colocará en bajo la bandera, hasta que suceda una nueva interrupción. En la rutina de interrupción se bajan las banderas de interrupción y se pone en alto la bandera stepFlag.

```
void Timer0AIntHandler ( void ){
```

```
// Limpiar banderas de interrupci ón
TimerIntClear ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
// Habilitar banderas de
stepFlag = true ;
}
```

Existen algunos integrados que realizan el control por micropaso. El L6470 de ST que utiliza protocolo SPI para comunicarse con la unidad de procesamiento o el DRV8825 de Texas Instruments, basado en pulsos para indicar los pasos y direcciones del motor; son ejemplos de ellos.

Figura 154. **Integrados para microstepping**



(a) Integrado DRV8825 de Texas Instruments. (b) Diagrama de bloques L6470 de ST.

Fuente: *PWM controlled high current DMOS universal motor driver*. http://www.ti.com/general/docs/datasheetdiagram.tsp?genericPartNumber= DRV8825&diagramId=SLVSA73F. http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00255075.pdf. Consulta: octubre de 2015.

4.6.2. Lectura de un sensor digital

Como práctica se propone el uso de protocolos de comunicación para la interacción entre una unidad de procesamiento y sensores digitales. Para el uso de los sensores es posible utilizar cualquier microcontrolador. Las lecturas

deben efectuarse sobre un giroscopio y un acelerómetro. El MPU9150 ofrece ambos en un empaquetado, además de un magnetómetro. La librería TivaWare ofrece una estructura muy ordenada para su manipulación y calibración, por lo que se opta por mostrar el ejercicio con una tarjeta TivaC como plataforma de desarrollo y elMPU9150.

- Lectura de MPU9150/MPU6050. Protocolos UART y I2C

En esta sección se describe el código basado en la librería TivaWare para utilizar un dispositivo MPU9150, o MPU6050 y los periféricos UART para comunicarse con un computador.

Un MPU9150 es un circuito integrado, conteniendo un acelerómetro, un giroscopio y un magnetómetro, todos ellos de 3 ejes por lo que se dice que es un sensor de 9 ejes. Su construcción esta basada en tecnología MEMS y su arquitectura la forma un sistema digital complejo, conteniendo unidades de procesamiento y periféricos.

Este sensor cuenta con 3 módulos ADC de 16 bits para digitalizar las señales del giroscopio, 3 módulos ADC de 16 bits para digitalizar las señales del acelerómetro y 3 módulos de ADC de 13 bits para digitalizar las señales del magnetómetro. Este empaquetado es un módulo multichip, consistente de dos matrices integradas en un solo empaquetado. Una de ellas alberga el giroscopio y el acelerómetro, la otra al magnetómetro *AK9875C 3-Axis* de la corporación Asahi Kasei Microdevices. El acceso a los sensores se hace mediante el protocolo I2C a una frecuencia de 400 kHz.

El proceso de acceder a los sensores usando un microcontrolador se puede tornar extenso, por lo que la lectura de las magnitudes registradas por

los sensores se hará con apoyo de la librería TivaWare. En el programa principal se agregan las librerías y la declaración **#define** PART_TM4C123GH6PM, para indicarles de que dispositivo se estará trabajando.

```
# define PART_TM4C123GH6PM
# include <stdint .h>
# include <stdbool .h>
# include " inc / hw_memmap .h"
# include " inc / hw_types .h"
# include " inc / hw_ints .h"
# include " driverlib / gpio .h"
# include " driverlib / pin_map .h"
# include " driverlib / sysctl .h"
# include " driverlib / uart .h"
# include " utils / uartstdio .h"
# include " sensorlib / i2cm_drv .h"
# include " sensorlib / ak8975 .h"
# include " sensorlib / hw_mpu9150 .h"
# include " sensorlib / mpu9150 .h"
```

Como se puede apreciar se hace uso de la sección de la librería utils y la sección sensorlib, por lo que se deben agregar de la misma manera que se hizo para la librería driverlib. La primera de ellas permite utilizar el módulo de UART con algunas funciones de entrada y salida típicas de C, como printf. Claro está, al hablar del periférico de un microcontrolador, no se tendrán todas las funcionalidades que se tienen para una computadora. La segunda, la librería sensorlib, ofrece una serie de funcionalidades para diversos sensores, en este caso resulta útil para un sensor MPU9150.

Para mayor facilidad a la hora de escribir el código se declara como una constante para el precompilador el valor del reloj principal.

```
# define clock 80000000
```

Para el uso de este dispositivo, la librería utiliza estructuras para manipular los periféricos. Basado en la documentación oficial de la librería sensorlib, se ha

modificado el ejemplo mostrado para que el programa sea capaz de enviar la información a un computador y poder desplegar gráficas.

A continuación, se muestran las instancias creadas de las estructuras tI2CMInstance y tMPU9150 que permiten el control del sensor mediante el periférico I2C.

```
tI2CMInstance I2CInst ;
tMPU9150 MPU9150Inst ;
volatile bool MPU9150Done ;
```

La instancia I2CInst alberga parámetros de configuración para el módulo I2C. La instancia MPU9150Inst alberga parámetros de configuración para el sensor. Es necesario agregar las librerías, i2cm_drv, para uso del I2C en forma de estructuras, la librería ak8975 para el uso del magnetómetro en el sensor, la librería mpu9150 para las funciones propias del sensor y la librería hw_mpu9150 para el uso de los registros en el sensor.

La variable MPU9150Done es utilizada para indicar cuando el sensor ha terminado de leer y entregar la información obtenida del entorno.

Se declara la rutina MPU9150Callback, la cual levanta una bandera indicando que las transacciones con el MPU9150 se han llevado a cabo correctamente, se ejecuta al ser llamada por un control de interrupciones otorgando el estado actual del proceso en el parámetro ui8Status.

```
void MPU9150Callback ( void * pvCallbackData , uint_fast8_t ui8Status )
{
//
// Verifica si un error ha ocurrido .
//
if( ui8Status != I2CM_STATUS_SUCCESS )
{
//
// En caso de error , colocar acciones aquí.
```

```
//
}
//
// Si la transacci ón con el MPU9150 se ha llevado a cabo .
//
MPU9150Done = true ;
}
```

Esta rutina debe ser ejecutada al momento que se levante la interrupción por hardware proveniente del módulo I2C1, que se configurará más adelante. Para ello se declara una rutina para la interrupción en el programa principal, y por supuesto en el vector de interrupciones para que al suceder el microcontrolador pueda acceder a ella, tal como se ha detallado en el capítulo anterior. Para ello se utiliza un controlador de interrupciones I2CMIntHandler, propio de la librería i2cm_drv, dentro de la rutina de interrupción. Este controlador toma como parámetro la estructura I2CInst, la cual guarda las configuraciones del módulo I2C1 y variables de control para su funcionamiento.

```
void IntHandlerI2C1 ( void ){
I2CMIntHandler (& I2CInst );
}
```

A continuación, se declara la rutina **void** MPU9150Lectura(**void**) que realizará las configuraciones para el módulo I2C y la configuración inicial del sensor MPU9150. Dentro de la rutina se declaran las variables Accel, Gyro y Magneto; las cuales almacenarán los valores de cada eje del acelerómetro, giroscopio y magnetómetro respectivamente.

```
uint32_t Accel [3] , Gyro [3] , Magneto [3];
```

Luego se habilitan las interrupciones para el módulo I2C1, y se les da una alta prioridad. La rutina **I2CMInit** , configura el módulo I2C indicado, recibe como parámetros, la dirección de memoria de una estructura capaz de albergar

parámetros de I2C que ya se encuentra definida por el tipo tI2CMInstance, en este caso I2CInst, la dirección del registro base del módulo de I2C a utilizar, en este caso el número 1, el valor de la interrupción, los valores 0xff en los parámetros los cuáles corresponden a accesos DMA, pero que aún no se encuentran construidos dentro de la librería y se coloca cualquier valor, y por último el valor del reloj con que se encuentra operando el microcontrolador. El pin PA6 en el tm4c123gh6pmes SCL del periférico I2C módulo 1 y el pin PA7 es SDA.

```
IntEnable ( INT_I2C1 );
IntPrioritySet ( INT_I2C1 , 0x00);
I2CMInit (& I2CInst , I2C1_BASE , INT_I2C1 , 0xff , 0xff , clock );
IntMasterEnable ();
```

A continuación se habilitan las interrupciones generales en el microcontrolador. Se pone en bajo la bandera MPU9150Done, para poder realizar una lectura y así estar seguros que se llevó a cabo correctamente una transacción al estar nuevamente arriba dicha bandera. De la misma forma que existe una rutina para una configuración rápida13 del módulo I2C existe una rutina para el sensor, los parámetros que recibe la rutina MPU9150Init es una dirección de memoria de una estructura que alberga declaraciones útiles para el control del sensor, de tipo tMPU9150, y en este caso ya fue declarada la instancia MPU9150Inst, una dirección de una estructura de I2C la cual se encargará de manipular el I2C que interactuá con el sensor, la dirección correspondiente al sensor, la cual es 0x68, la rutina MPU9150Callback que lleva control de las transacciones y la información que será pasada como parámetro a esta rutina.

Utilizando *polling* se espera a que todas las transacciones hayan sido procesadas.

```
MPU9150Done = false ;
```

```
MPU9150Init (& MPU9150Inst , & I2CInst , 0x68 , MPU9150Callback , 0);
while (! MPU9150Done )
{}
```

El proceso lógico a seguir es colocar en bajo la bandera MPU9150Done, darle instrucciones al sensor utilizando las funciones de la librería y esperar hasta que el sensor indique que ha terminado de procesarlas. La librería ofrece ya configuraciones del sensor que se entregan utilizando el protocolo I2C, a continuación se configura el acelerómetro para tener una escala de $\pm 4g$.

```
MPU9150Done = false ;
MPU9150ReadModifyWrite (& MPU9150Inst , MPU9150_O_ACCEL_CONFIG ,
~ MPU9150_ACCEL_CONFIG_AFS_SEL_M ,
MPU9150_ACCEL_CONFIG_AFS_SEL_4G ,
MPU9150Callback ,0);
while (! MPU9150Done )
{}
```

La rutina **MPU9150ReadModifyWrite** lee un registro del sensor, hace una conjunción bit a bit con el valor que almacena y una máscara, luego hace una disyunción con un valor y el resultado se escribe en el registro del sensor. La rutina recibe como parámetros, la dirección de memoria de una estructura de tipo tMPU9150, el registro a modificar, la máscara, el valor, la rutina que se lleva a cabo para indicar que no ha sucedido un error y la información que recibe la rutina. Luego se espera que la rutina MPU9150Callback, o la que se ha dado como parámetro, levante la bandera para indicar que se han llevado a cabo las transacciones.

La librería hw_mpu9150 ofrece constantes definidas como MPU9150_O_ACCEL_CONFIG equivalente a 0x1C, la constante MPU9150_ACCEL_CONFIG_AFS_SEL_M equivalente a 0x18 y MPU9150_ACCEL_CONFIG_AFS_SEL_4G equivalente a 0x08. Al consultar la hoja de datos del MPU9150 puede verse que lo que se hizo con la rutina fue configurar el valor que almacena el registro 0x1C

del sensor MPU9150 y modificar únicamente los bits 3 y 4 de este registro para que guarde la configuración correspondiente a la escala de $\pm 4g$. Es posible apoyarse en la hoja de datos del dispositivo que posee un mapa de los registros, para ver otras configuraciones posibles para este dispositivo.

Para efectuar la lectura del sensor se utiliza la rutina MPU9150DataRead, la cual almacenará los valores del acelerómetro, magnetómetro y giroscopio en la estructura de tipo tMPU9150 cuya dirección haya sido dada en los parámetros para esta rutina.

```
MPU9150Done = false ;
MPU9150DataRead (& MPU9150Inst , MPU9150Callback , 0);
while (! MPU9150Done )
{}
```

Utilizando las rutinas MPU9150DataAccelGetRaw, MPU9150DataGyroGet Raw y MPU9150DataMagnetoGetRaw se puede obtener la información obtenida por MPU9150DataRead. Recibe como parámetros la estructura utilizada por MPU9150DataRead y las direcciones de memoria donde se almacenarán los valores obtenidos para cada eje.

```
MPU9150DataAccelGetRaw (& MPU9150Inst , & Accel [0] , & Accel [1] ,& Accel [2]) ;
MPU9150DataGyroGetRaw (& MPU9150Inst , & Gyro [0] , & Gyro [1] , & Gyro [2]) ;
MPU9150DataMagnetoGetRaw (& MPU9150Inst , & Magneto [0] , & Magneto [1] ,& Magneto [2]) ;
```

De esta forma se puede ver la rutina MPU9150Lectura completa, donde se realizan lecturas periódicas del sensor.

```
void MPU9150Example ( void )
{
        // Declaraci ón de las variables que almacenar án las lecturas en los ejes .
        uint32_t Accel [3] , Gyro [3] , Magneto [3];
        // Habilitaci ón de las interrupciones y configuraci ón del mó dulo I2C1 .
        IntEnable ( INT_I2C1 );
        IntPrioritySet ( INT_I2C1 , 0x00);
```

```
        I2CMInit (& I2CInst , I2C1_BASE , INT_I2C1 , 0xff , 0xff , clock );
        IntMasterEnable ();
        // Inicializa el sensor MPU9150 .
        MPU9150Done = false ;
        MPU9150Init (& MPU9150Inst , & I2CInst , 0x68 , MPU9150Callback , 0);
        while (! MPU9150Done )
        {}
        // Configura el MPU9150 para un rango comprendido entre +/- 4 g.
        g_bMPU9150Done = false ;
        MPU9150ReadModifyWrite (& MPU9150 , MPU9150_O_ACCEL_CONFIG ,
                                ~ MPU9150_ACCEL_CONFIG_AFS_SEL_M ,
                                MPU9150_ACCEL_CONFIG_AFS_SEL_4G ,
                                MPU9150Callback ,0);
        while (! MPU9150Done )
        {}
        while (1)
        {
                // Realizar otra lectura del MPU9150 .
                MPU9150Done = false ;
                MPU9150DataRead (& sMPU9150Inst , MPU9150Callback , 0);
                while (! MPU9150Done )
      {}
      // Obtiene las lecturas de los sensores almacenadas en MPU9150Inst .
      MPU9150DataAccelGetRaw (& MPU9150Inst , & Accel [0] , & Accel [1] ,& Accel [2]) ;
      MPU9150DataGyroGetRaw (& MPU9150Inst , & Gyro [0] , & Gyro [1] , & Gyro [2]) ;
      MPU9150DataMagnetoGetRaw (& MPU9150Inst , & Magneto [0] , & Magneto [1] ,&
      Magneto [2]) ;
      }
}
```

Hasta el momento en esta rutina se tiene un microcontrolador que toma datos del entorno pero no interacciona con él haciéndolo un sistema cerrado, solo recibe información pidiéndola constantemente a un sensor sin entregar alguna señal al entorno, es necesario configurar algún otro protocolo que permita al entorno obtener los datos que ya han sido leído del sensor. En este ejemplo se utilizará el protocolo UART para enviar los datos obtenidos por el sensor a una computadora, para poder ver gráficas de los valores obtenidos por el sensor. Para ello se hace uso de la librería uart y uartstdio, la primera brinda declaraciones útiles para el control del periférico y la segunda permite al programador utilizar el módulo de UART de forma similar que lo haría para una salida o entrada estándar en un computador.

En la rutina principal del programa se configura el periférico de UART0 el cual en la tarjeta de desarrolla TivaC tiene conectado sus pines RX y TX (PA0 y PA1 respectivamente) a la interfaz ICDI(Integrated Circuit Debugger Interface) que al ser conectada a una computadora tendrá un puerto virtual serial que permite interaccionar con el microcontrolador.

En la rutina principal se configura el reloj principal del sistema, se le da reloj a los periféricos a utilizar, se configuran los pines respectivamente para que tomen sus respectivos roles en el programa.

```
SysCtlClockSet ( SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ );
/* ************** Configura Reloj de GPIOA ********************** */
SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOA );
/* ************** Configura Reloj de UART0 ********************** */
SysCtlPeripheralEnable ( SYSCTL_PERIPH_UART0 );
/* ************** Configura Reloj de I2C1 ********************** */
SysCtlPeripheralEnable ( SYSCTL_PERIPH_I2C1 );
/* ************ Configura pines PA7 y PA6 como I2C ************ */
GPIOPinConfigure ( GPIO_PA7_I2C1SDA );
GPIOPinConfigure ( GPIO_PA6_I2C1SCL );
/* ************ Configura PA7 como I2C DATA ******** */
GPIOPinTypeI2C ( GPIO_PORTA_BASE , GPIO_PIN_7 );
/* ************ Configura PA7 como I2C DATA ******** */
GPIOPinTypeI2CSCL ( GPIO_PORTA_BASE , GPIO_PIN_6 );

/* ************ Configura pines PA0 y PA1 como UART ************ */
GPIOPinTypeUART ( GPIO_PORTA_BASE , GPIO_PIN_0 | GPIO_PIN_1 );
```

Es posible configurar rápidamente el periférico UART0 utilizando la función UARTStdioConfig, la cual recibe como parámetros el número de periférico a utilizar, la velocidad del reloj con el cual se sincronizará el periférico con el otro dispositivo y el reloj del sistema. Esta sentencia permite que cualquier función a utilizar de la librería uartstdio, utilice el periférico nombrado en sus parámetros con su configuración indicada en esta función. Por ejemplo, utilizando UARTprintf permite escribir de forma similar que la función printf de la stdio de cualquier compilador de

C para computadoras. La rutina UARTprintf permite dar formato a algunas variables.

```
// Crea una sesi ón para ser utilizada
// por las dem ás funciones de la librer ía
UARTStdioConfig (0, 115200 , clock );
// Imprime el mensaje que recibe como pará metro .
UARTprintf (" Hola Mundo !\n");
```

El código entregará a una razón de 115 200 baudios el mensaje "Hola Mundo", el carácter '\n' es utilizado para el salto de línea en las consolas, la librería agrega el carácter '\r' de retorno de carro para ser consistentes con algunas antiguas y obsoletas convenciones en el diseño de las consolas.

Para enviar los valores obtenidos por el sensor a una computadora se utilizará esta librería para poder darle formato decimal de forma sencilla a los valores obtenidos por el sensor. A continuación de las rutinas para almacenar los valores del sensor se coloca la rutina UARTprintf para enviar por UART los valores en formato decimal.

```
UARTprintf (" Sensor :%d:%d:%d:%d:%d:%d:%d:%d:%d\n",
                    Accel [0] , Accel [1] , Accel [2] ,
                    Gyro [0] , Gyro [1] , Gyro [2] ,
                    Magneto [0] , Magneto [1] , Magneto [2]) ;
```

Se ha separado por el caracter ':' cada valor obtenido del sensor, para separarlos y un programa corriendo en una computadora que leerá la información proveída por el microcontrolador pueda procesarlos correctamente.

Se ha separado por el caracter ':' cada valor obtenido del sensor, para separarlos y un programa corriendo en una computadora que leerá la información proveída por el microcontrolador para procesarlos correctamente.

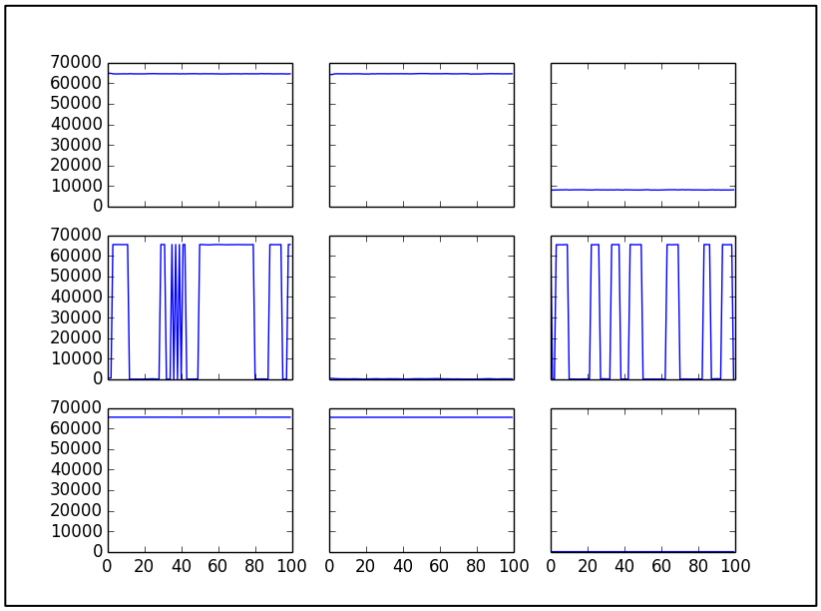
El programa 4.5 escrito en Python toma 3 000 muestras entregadas por el microcontrolador, y luego hace gráficas de las medidas tomadas, para la utilización del puerto virtual creado en la computadora se utiliza la librería serial, para la generación de gráficas la librería matplotlib; la librería numpy permite el uso de ciertas estructuras de manera similar a SCILAB oMatlab.

```
import serial
import matplotlib . pyplot as plt
import numpy as np

ser = serial . Serial ('/ dev / ttyACM0 ', 115200 , timeout =1) ; # UART
Ax =[]; Ay =[]; Az =[];
  Gx =[]; Gy =[]; Gz =[];
Mx =[]; My =[]; Mz =[];
n =0;
while n <3000:
vec =[]
  sensor =[]
data = ser . readline ()
  vec = data . split ('\r\n')
sensor = vec [0]. split (':')
  header = sensor [0];
if( header ==" Sensor " and len( sensor ) ==10) :
  Ax. append ( int ( sensor [1]) );
Ay. append ( int ( sensor [2]) );
  Az. append ( int ( sensor [3]) );
Gx. append ( int ( sensor [4]) );
  Gy. append ( int ( sensor [5]) );
Gz. append ( int ( sensor [6]) );
  Mx. append ( int ( sensor [7]) );
My. append ( int ( sensor [8]) );
  Mz. append ( int ( sensor [9]) );
n=n +1;
t=np. linspace (0, len(Ax) -1, len (Ax));
  f, (( GAx , GAy , GAz) ,(GGx , GGy , GGz ) ,(GMx , GMy , GMz ))=plt. subplots (3 ,3 , sharex
  ='col ',
        sharey ='row ');
GAx . plot (t, Ax); GAy . plot (t, Ay); GAz . plot (t, Az);
  GGx . plot (t, Gx); GGy . plot (t, Gy); GGz . plot (t, Gz);
GMx . plot (t, Mx); GMy . plot (t, My); GMz . plot (t, Mz);
plt . show ();
```

Las gráficas generadas por el programa 4.5 muestran los 16 bits en los que se almacenan el valor de cada eje del sensor. Si se coloca estático en una posición determinada se pueden ver gráficas como las mostradas por la figura 155.

Figura 155. **Gráfica generada por programa 4,5**



Fuente: elaboración propia, empleando Python, librería Matplotlib.

Tal como puede verse en la figura anterior, aparentemente se tienen oscilaciones bruscas desde el valor más alto al valor más bajo en las lecturas, sin embargo se debe recordar que el sensor es capaz de leer valores referenciados en un sistema de coordenadas local y por tanto los valores pueden ser positivos o negativos, el rango del sensor es simétrico y los valores que se almacenan en los registros lo hacen con formato, por lo que el primer bit se utiliza para indicar el signo.

En la hoja de datos se puede apreciar que los rangos para este sensor son configurables, en el programa escrito en C, se puede ver que el microcontrolador ha configurado el rango del acelerómetro para que entregue las lecturas de fuerza $x(t)$ en el intervalo simétrico [¡4g ,4g] m/s$_2$. El sensor entrega una magnitud k, expresada con 16 bits, por lo que las lecturas serán números enteros comprendidos entre 0 y 65 535, pero se debe recordar que el MSB es tomado como signo, si este bit es uno, se debe restar 2_{16} a la cantidad, si es cero solo se toman en cuenta los

$$f(k) = k - 65536(k \gg 15) \qquad \forall k \in \{0,1,\dots,65535\} \qquad \text{[Ec. 4.105]}$$

Donde $k \gg 15$ es la operación válida únicamente para enteros equivalente al corrimiento de 15 bits a la derecha de su representación en un sistema de numeración binario. Ya que $0 \leq k \leq 65535$, siendo k entero correr 15 bits de su representación binaria equivale a solamente verificar si el MSB es 1 o 0. Ya que el rango es simétrico para al modelar este sensor como una función afín g , la constante de sesgo es cero y la de sensibilidad α está dada por $\alpha = \frac{4g}{2^{15}} \approx 0{,}0011970964$, para el caso específico del acelerómetro.

$$g\big(f(k)\big) = a(k - 65536) \qquad \forall k \in \{0,1,\dots,65535\} \qquad \text{[Ec. 4.106]}$$

En la hoja de datos se puede ver que el valor por defecto para el rango del giroscopio es una velocidad angular comprendida entre [-250,250] grados por segundo, equivalente a $\left[-\frac{25}{18}\pi\frac{25}{18}\right]$ radianes por segundo y la constante de sensibilidad esta dada por $\alpha = \frac{25\pi}{18x2^{15}} \approx 1{,}3323124\text{x}10^{-4}$ por bit.

En el caso del magnetómetro el análisis es muy similar, con la diferencia que el ADC que realiza la transducción para los ejes de este sensor utiliza 13

bits en lugar de 16, sin embargo, el almacenamiento de estos valores, con 16 bits, se constrye la tabla LXXXVI.

Tabla LXXXVI. **Formato de las mediciones del magnetómetro**

Measurement data (each axis) [15:0]			Magnetic flux density [µT]
Two's complement	Hex	Decimal	
0000 1111 1111 1111	0FFF	4095	1229(max.)
|	|	|	|
0000 0000 0000 0001	0001	1	0.3
0000 0000 0000 0000	0000	0	0
1111 1111 1111 1111	FFFF	-1	-0.3
|	|	|	|
1111 0000 0000 0000	F000	-4096	-1229(min.)

Fuente: *MPU-9150 RegisterMap. Document Number RM-MPU-9150A-00. Revisión 4.2*. https://store.invensense.com/Datasheets/invensense/RM-MPU-9150A-00-v3.0.pdf. Consulta: noviembre de 2015.

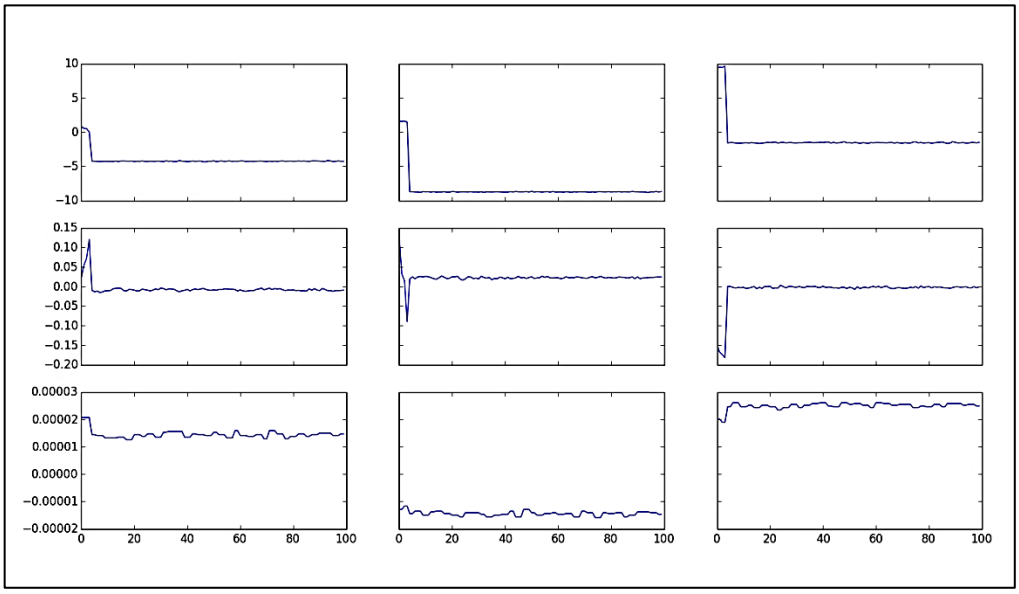
Con base en la tabla LXXXVI, se tiene que la constante de sensibilidad para el magnetómetro se obtiene al dividir el máximo valor del rango equivalente a 1.229T entre 2^{12}, teniendo como resultado $\alpha =$ 1229/4095/0,0000003 πT por bit.

El siguiente código escrito en Python, declara las constantes de sensibilidad para cada sensor y define una función que permite implementar la función afín descrita con anterioridad.

```
1|| Acc =0.0011970964;
2|| Gyr =1.3323124e -4;
3|| Mag =0.0000003;
4||
5|| def Afin (k,a,b):
6|| k= int (k)
7|| k=k -(k >> 15) * 65536;
8|| return k*a+b;
```

Al implementar esta función para cada medición obtenida se consiguen gráficas similares a las que se muestra en la figura 156.

Figura 156. **Mediciones interpretadas correctamente por programa**



Fuente: elaboración propia, empleando Python, librería Matplotlib.

Al jugar con la posición del sensor se puede observar como las mediciones van cambiando de acuerdo a la posición actual del sensor. Sin embargo, se debe notar que las mediciones arrojadas por el sensor poseen ruido que puede ser filtrado de forma digital.

- Simulación de ruido en circuito RLC y filtro de Kalman.

Como se menciona anteriormente, en la práctica las mediciones efectuadas sobre señales físicas se verán expuestas a ruido. Incluso en sistemas simples como un circuito RLC. Para tratar con este problema, se ha diseñado una serie de filtros cuyo objetivo es separar la señal original de las

perturbaciones externas. Los filtros en el análisis de señales es un tema sumamente extenso, a continuación, se propone como práctica la construcción de un filtro de Kalman, muy popular en sistemas físicos con controles digitales, y esto se debe a los increíbles resultados que se pueden obtener de él.

Antes de pasar al filtro de la información proveída por el sensor se propone simular primero un circuito RLC expuesto a ruido blanco y aplicarle el filtro de Kalman a ello para demostrar las partes que lo componen. El objetivo del filtro de Kalman es estimar los estados de una manera óptima minimizando el error cuadrático medio entre el estado real y el estimado por el filtro.

Considérese un espacio de estados lineal discreto, con una estructura muy similar al continuo de la figura 49 que representa las ecuaciones 2.373 y 2.374; al hablar de un sistema discreto cobra sentido considerar el siguiente estado, para modelar una dinámica, y deja de tenerlo considerar la derivada en el modelo. Considérese un fenómeno con una dinámica discreta cuyo modelo se puede representar por un espacio de estado lineal y discreto el cual se encuentra expuesto a perturbaciones, y sus salidas son observadas o muestreadas.

$$x_{k+1} = Ax_k + Bu_k + vk \qquad \text{[Ec. 4.107]}$$

$$yk = Cx_k + w_k \qquad \text{[Ec. 4.108]}$$

Donde $\boldsymbol{x}_k \in \mathbb{R}^n$ y sus valores son magnitudes que definen el estado actual del sistema, $\boldsymbol{x}_{k+1} \in \mathbb{R}^n$ el siguiente estado del sistema, $\boldsymbol{y}_k \in \mathbb{R}^m$la salida actual del sistema, $\boldsymbol{u}_k \in \mathbb{R}^r$ es un vector con entradas al sistema, $\boldsymbol{v}_k \in \mathbb{R}^n$ y $\boldsymbol{w}_k \in \mathbb{R}^m$perturbaciones de ruido blanco, gaussiano y de media cero. Las

matrices *A*, *B* definen de forma determinista la dinámica del sistema y la matriz *C* define las salidas del sistema.

Considérese el estado inicial del sistema $x_0 = E[x_0] = x_0$, como el valor esperado. Se consideran las perturbaciones como procesos aleatorios entre sí por tanto las covarianzas entre diferentes procesos tendrán un valor de cero. La matriz R_k se define como una matriz diagonal cuadrada de covarianzas de las perturbaciones en las salidas descrita de la siguiente forma:

$$E[w_k, w_k] = R_k \qquad \text{[Ec. 4.109]}$$

La matriz Q_k se define como una matriz diagonal cuadrada de covarianzas de las perturbaciones en el sistema, descrita de la siguiente forma:

$$E[v_k, v_k] = Q_k \qquad \text{[Ec. 4.110]}$$

Considérese el error e_k como la diferencia del valor real del estado x_k y la estimación $\tilde{x}_k$hecha.

$$e_k = x_k - \tilde{x}_k \qquad \text{[Ec. 4.111]}$$

El objetivo del filtro es minimizar el error cuadrático medio entre estos dos estados; sea P_k la matriz de covarianzas para los estados $\boldsymbol{x}$ y $\boldsymbol{x^T}$.

$$P_k = E[(x_k - \tilde{x}_k)(x_k - \tilde{x}_k)^T] = E[\boldsymbol{e}_k \boldsymbol{e}_k] \qquad \text{[Ec. 4.112]}$$

Teniendo medidas contaminadas de las salidas $y_0, y_1 \ldots y_k$ el filtro trata de encontrar los valores $\tilde{x}_{k-1}$para que P_k sea óptima. El filtro es un algoritmo que

consiste de dos pasos, predicción antes de tener la medida y corrección del estado luego de obtenida la medida.

El valor de la predicción $\tilde{x}_{k+1}$ es calculado a partir del valor más actualizado del estado, que posteriormente será corregido en la etapa de actualización o corrección del algoritmo.

$$\tilde{\boldsymbol{x}}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k \quad \text{[Ec. 4.113]}$$

El error $\tilde{\boldsymbol{e}}_{k+1}$ para la predicción es la diferencia entre el siguiente valor real xkÅ1 y la predicción efectuada.

$$\tilde{\boldsymbol{e}}_{k+1} = \boldsymbol{x}_{k+1}\tilde{\boldsymbol{x}}_{k+1} \quad \text{[Ec. 4.114]}$$

$$= Ax_k + B\boldsymbol{u}_k + \boldsymbol{v}_k - A\tilde{\boldsymbol{x}}_k - B\boldsymbol{u}_k \quad \text{[Ec. 4.115]}$$

$$= A(\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k) + \boldsymbol{V}_k \quad \text{[Ec. 4.116]}$$

La matriz de covarianza P_{k+1} para el error $\tilde{\boldsymbol{e}}_{k+1}$ esta dada:

$$\tilde{P}_{k+1} = E[(A(x_k - \tilde{x}_k) + v_k)(A(x_k - \tilde{x}_k) + v_k)^T] \quad \text{[Ec. 4.117]}$$

$$= E[(A(x_k - \tilde{x}_k) + v_k)(A(x_k - \tilde{x}_k) + v_k)^T] \quad \text{[Ec. 4.118]}$$

$$= E\left[\left(A(x_k - \tilde{x}_k)\right)\left(A(x_k - \tilde{x}_k)\right)^T + E[v_k v_k^T]\right. \quad \text{[Ec. 4.119]}$$

$$= E[A(x_k - \tilde{x}_k)(x_k - \tilde{x}_k)^T A^T] + E[v_k v_k^T] \quad \text{[Ec. 4.120]}$$

$$= AP_k A^t + Q_k \quad \text{[Ec. 4.121]}$$

El filtro realiza el cálculo para el estado siguiente con base al valor anterior del estado y una corrección basada en una aplicación lineal de la diferencia entre la última medición observada y_{k+1} y la salida del estado $\tilde{x}_{k+1}$, resultado de la predicción efectuada; la aplicación lineal es definida por la matriz K_{k+1}. El estado más actualizado que se tiene es el resultado de la predicción efectuada, y se considera al error como la medición observada y la salida estimada por el estado más actual:

$$\widehat{\boldsymbol{x}}_{k+1} = \widetilde{\boldsymbol{x}}_{k+1} + K_{k+1}[Y_{k+1} - C\widetilde{\boldsymbol{x}}_{k+1}] \quad \text{[Ec. 4.122]}$$

Si se substituye la ecuación anterior en la ecuación del error se tiene.

$$\boldsymbol{e}_{k+1} = \boldsymbol{x}_{k+1} - \widehat{\boldsymbol{x}}_{k+1} \quad \text{[Ec. 4.123]}$$

$$= \boldsymbol{x}_{k+1} - \widehat{\boldsymbol{x}}_{k+1} - K_{k+1}[Y_{k+1} - C\widetilde{\boldsymbol{x}}_{k+1}] \quad \text{[Ec. 4.124]}$$

La medición observada tiene la forma de la ecuación 4.108, por tanto el error esta dado por:

$$\boldsymbol{e}_{k+1} = \boldsymbol{x}_{k+1} - \widetilde{\boldsymbol{x}}_{k+1} - K_{k+1} \quad \text{[Ec. 4.125]}$$

$$= (I - K_{k+1}C)(x_{k+1} - \tilde{x}_{k+1}) - K_{k+1}w_{k+1} \quad \text{[Ec. 4.126]}$$

Teniendo la covarianza de la siguiente forma:

$$P_{k+1} = E\left[\left((I - K_{k+1}C)(x_{k+1} - \tilde{x}_{k+1}) - K_{k+1}w_{k+1}\right)\left((I - K_{k+1}C)(x_{k+1} - \tilde{x}_{k+1}) - K_{k+1}w_{k+1}\right)^T\right] \quad \text{[Ec. 4.127]}$$

$$= E\left[\left((I - K_{k+1}C)(X_{k+1})\right)\left((I - K_{k+1})(x_{k+1} - \tilde{x}_{k+1})^T\right)\right] + E[K_{k+1}w_{k+1}w_{k+1}^T K_{k+1}^T]$$

[Ec. 4.128]

$$= (I - K_{k+1}C)E[(x_{k+1} - \tilde{x}_{k+1})^T](I - K_{k+1})^T + K_{k+1}E[w_{k+1}w_{k+1}^T]K_{k+1}^T \quad \text{[Ec. 4.129]}$$

$$= (I - K_{k+1}C)\tilde{P}_{k+1}(I - K_{k+1}C)^T + K_{k+1}R_{k+1}K_{k+}^T \quad \text{[Ec. 4.130]}$$

Expandiendo el resultado anterior se tiene:

$$P_{k+1} = \left(\tilde{P}_{k+1} - K_{k+1}C\tilde{P}_{k+1}\right)(I - C^T K_{k+1}^T) + K_{k+1}R_{k+1}K_{k+1}^T \quad \text{[Ec. 4.131]}$$

$$= \tilde{P}_{k+1} + K_{k+1}C\tilde{P}_{k+1} - \tilde{P}_{k+1}C^T K_{k+1}^T + K_{k+1}C\tilde{P}_{k+1}C^T K_{k+1}^T + K_{k+1}R_{k+1}K_{k+1}^T$$

[Ec. 4.132]

$$= \tilde{P}_{k+1} + K_{k+1}C\tilde{P}_{k+1} - \tilde{P}_{k+1}C^T K_{k+1}^T + K_{k+1}\left(C\tilde{P}_{k+1}C^T + R_{k+1}\right)K_{k+1}^T \quad \text{[Ec. 4.133]}$$

Si se hace la matriz S_{k+1}:

$$S_{k+1} = C\tilde{P}_{k+1}C^T + R_{k+1} \quad \text{[Ec. 4.134]}$$

Se tiene para P_{k+1}:

$$P_{k+1} = \tilde{P}_{k+1} - K_{k+1}C\tilde{P}_{k+1}\tilde{P}_{k+1}C^T K_{k+1}^t + K_{k+1}S_{k+1}K_{k+1}^T \quad \text{[Ec. 4.135]}$$

El criterio a optimizar por el filtro de Kalma es el error cuadrático medio, ya que la traza de una matriz cuadrada es la suma de los elementos que componen la diagonal principal.

$$E[|\boldsymbol{e}_{k+1}|^2] = E[e_{k+1} \cdot e_{k+1}] \quad \text{[Ec. 4.136]}$$

$$= E[e_{k+1}^T e_{k+1}] \quad \text{[Ec. 4.137]}$$

$$= tr(P_{k+1}) \quad \text{[Ec. 4.138]}$$

Además se tiene la propiedad que la traza es operador lineal para matrices por tanto la traza de una suma es igual a la suma de las trazas.

$$tr(P_{k+1}) = tr\left(P_{k+1} - C\tilde{P}_{k+1} - \tilde{P}_{k+1}C^T K_{k+1}^T + K_{k+1}S_{k+1}K_{k+1}^T\right) \quad \text{[Ec. 4.139]}$$

$$= tr\left(P_{k+1}\right) - tr\left(C\tilde{P}_{k+1}\right) - tr\left(\tilde{P}_{k+1}C^T K_{k+1}^T\right) + tr(K_{k+1}S_{k+1}K_{k+1}^T) \quad \text{[Ec. 4.140]}$$

Para la derivación de las ecuaciones del filtro de Kalman se enuncian las siguientes propiedades para la traza de una matriz:

Si *X* es una matriz cuadrada su traza es igual a su transpuesta.

$$tr(X) = tr(X^T) \quad \text{[Ec. 4.141]}$$

Si el producto entre dos matrices *X* y *Y* puede ser efectuado a pesar de haber intercambiado el orden, la traza es igual para ambos productos.

$$tr(XY) = tr(XY) \quad \text{[Ec. 4.142]}$$

Si *X* y *Y* son matrices cuadradas las trazas de los siguientes productos serían iguales:

$$tr(X^T Y) = tr(XY^T) = tr(Y^T X) = tr(YX^T) = \sum_{i,j} XijYij \quad \text{[Ec. 4.143]}$$

De las propiedades anteriores se tiene para la traza de la matriz de covarianzas P_{k+1},

$$tr(P_{k+1}) = tr\left(\tilde{P}_{k+1}\right) - 2tr\left(K_{k+1}C\tilde{P}_{k+1}\right) + tr(K_{k+1}S_{k+1}K_{k+1}^T) \qquad \text{[Ec. 4.144]}$$

$$tr(P_{k+1}) = tr\left(\tilde{P}_{k+1}\right) - 2tr\left(\tilde{P}_{k+1}C^T K_{k+1}^T\right) + tr(K_{k+1}S_{k+1}K_{k+1}^T) \qquad \text{[Ec. 4.145]}$$

Se deben escoger los valores de las entradas de la matriz K para que el error cuadrático medio se reduzca al mínimo. Por tanto la matriz $K_{k+1} = k_{ij}$ que vuelve óptimo debe cumplir con:

$$\frac{\partial tr(P_{k+1})}{\partial k_{ij}} = 0, \qquad \forall k_{ij} \qquad \text{[Ec. 4.146]}$$

Por otro lado, teniendo a la matriz $X = x_{i\ j}$ y un escalar s, la matriz con entradas $\frac{\partial s}{\partial x_{ij}}$ se define como:

$$\frac{\partial s}{\partial X} = \begin{bmatrix} \frac{\partial s}{\partial x_{11}} & \frac{\partial s}{\partial x_{12}} & \cdots & \frac{\partial s}{\partial x_{1n}} \\ \frac{\partial s}{\partial x_{21}} & \frac{\partial s}{\partial x_{22}} & \cdots & \frac{\partial s}{\partial x_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial x_{m1}} & \frac{\partial s}{\partial x_{m2}} & \cdots & \frac{\partial s}{\partial x_{mn}} \end{bmatrix} \qquad \text{[Ec. 4.147]}$$

Al ser la traza de P_{k+1} un escalar, y K una matriz que busca optimizar su valor, se tiene que la matriz de derivadas debe ser igual a cero.

$$\frac{\partial tr(P_{k+1})}{\partial k_{ij}} = 0, \qquad \text{[Ec. 4.148]}$$

Es posible demostrar para la derivada matricial de una traza:

$$\frac{\partial tr(AX)}{\partial X} = \frac{\partial tr(XA)}{\partial X} = A \qquad \text{[Ec. 4.149]}$$

$$\frac{\partial tr(X^T AX)}{\partial X} = X^T(A+A^T) \qquad \text{[Ec. 4.150]}$$

Y en caso que *A* sea simétrica se tiene:

$$\frac{\partial tr(X^T AX)}{\partial X} = 2X^T A = 2(AX)^T \qquad \text{[Ec. 4.151]}$$

Por tanto se tiene para la traza de la matriz P_{k+1}

$$\frac{\partial tr(P_{k+1})}{\partial K} = \frac{\partial tr(\tilde{P}_{k+1})}{\partial K} - 2\frac{\partial tr(\tilde{P}_{k+1}C^T K_{k+1}^T)}{\partial K} + \frac{\partial tr(K_{k+1}S_{k+1}K_{k+1}^T)}{\partial K} = 0. \qquad \text{[Ec. 4.152]}$$

Las entradas de la matriz $\tilde{P}_{k+1}$ no dependen de las entradas de la matriz *K* por tanto el resultado es la matriz cero; de las ecuaciones 4.5.2, 4.149 y 4.150 se tiene una expresión para la ganancia de Kalman *K*.

$$-2\left(\tilde{P}_{k+1}C^T\right) + 2(K_{k+1}S_{k+1}) = 0 \qquad \text{[Ec. 4.153]}$$

$$K_{k+1}S_{k+1} = \tilde{P}_{k+1}C^T \qquad \text{[Ec. 4.154]}$$

$$K_{k+1} = \tilde{P}_{k+1}C^T(S_{k+1})^{-1} \qquad \text{[Ec. 4.155]}$$

Teniendo una expresión para la ganancia de Kalman es posible simplificar la expresión para la matriz de covarianzas, al multiplicar por K_{k+1}^T ambos lados 4.153, se puede simplificar la ecuación 4.134.

$$K_{k+1}S_{k+1}K_{k+1}^T = \tilde{P}_{k+1}C^T K_{k+1}^T \Longrightarrow \qquad \text{[Ec. 4.156]}$$

$$P_{k+1} = \tilde{P}_{k+1} K_{k+1} C \tilde{P}_{k+1} \quad \text{[Ec. 4.157]}$$

$$P_{k+1} = (I - K_{k+1} C) \tilde{P}_{k+1} \quad \text{[Ec. 4.158]}$$

De esta forma ya se tienen todas las expresiones necesarias para poder desarrollar el algoritmo. Las etapas de predicción y actualización se repiten iterativamente hasta tener todas muestras.

- Etapa de predicción: con base en la información generada por la iteración anterior y el modelo lineal se calcula un valor estimado del estado siguiente y su matriz de covarianzas.

    - Predicción del estado

$$\tilde{x}_{k+1} = A\tilde{x}_k + B\boldsymbol{u}_k \quad \text{[Ec. 4.159]}$$

    - Calculo de la matriz de covarianzas para la predicción

$$\tilde{P}_{k+1} = AP_k A^T + Q_k \quad \text{[Ec. 4.160]}$$

- Etapa de corrección o actualización: en esta etapa se realiza el cálculo de la ganancia de Kalman y con base a ella busca el valor del siguiente estado que minimiza el error cuadrático medio, aplicada a la medición realizada con ruido. Y se calcula la matriz de covarianzas que será usada para la siguiente iteración.

    - Cálculo de la ganancia de Kalman

$$K_{k+1} = \tilde{P}_{k+1} C^T (S_{k+1})^{-1} \quad \text{[Ec. 4.161]}$$

- Se realiza lectura del vector de magnitudes observadas y_{k+1}.
- Actualización o corrección del estado

$$\tilde{x}_{k+1} = \tilde{x}_{k+1} K_{k+1}(y_{k+1} - C\tilde{x}_{k+1}) \quad \text{[Ec. 4.162]}$$

- Actualización para la matriz de covarianzas

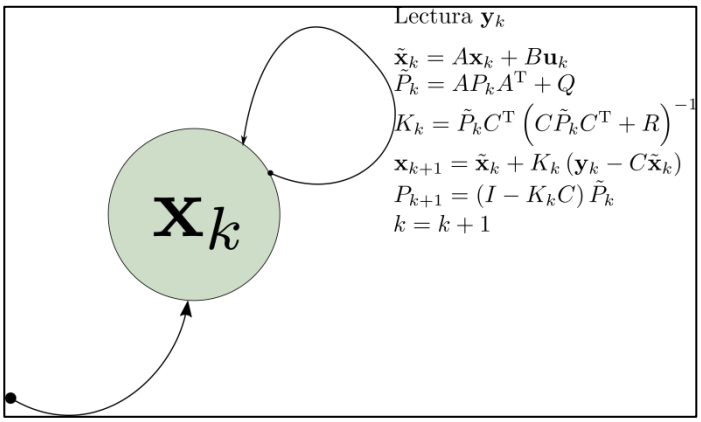
$$P_{k+1} = (I - K_{k+1}C)\tilde{P}_{k+1} \quad \text{[Ec. 4.163]}$$

Para la primera iteración se deben conocer las condiciones iniciales: el valor del estado $\mathbf{x}_0$. Las matrices de covarianza Q_k y R_k que en muchas aplicaciones se escogen como una constante para todo k, y es posible asumir para $P_0 = Q$.

Se debe tener en cuenta que la ganancia de Kalman es óptima solo si el modelo describe perfectamente el sistema real, el ruido es blanco y las covarianzas son conocidas exactamente.

La idea básica detrás del filtro es teniendo un modelo del sistema y entradas, el valor arrojado del modelo para las entradas debería ser similar al valor que se lee por un sensor, sin embargo al tener ruido en las entradas y en las lecturas, el filtro toma información estadística del sistema (las matrices de covarianzas) para estimar en base a las lecturas, la estimación con el modelo y los estados anteriores, el estado que se asemeja más al estado real del sistema.

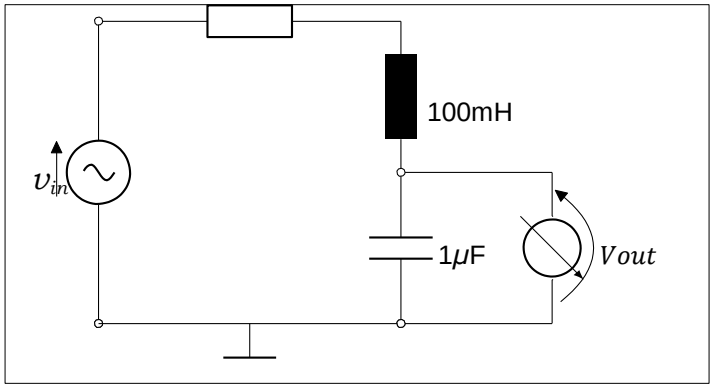
Figura 157. **Dinámica discreta de filtro de Kalman**



Fuente: elaboración propia, empleando Inkscape.

A continuación, se muestra como ejemplo la implementación del filtro a las mediciones hechas sobre el circuito RLC de la figura 163, cuyas muestras son tomadas del capacitor a una frecuencia de muestreo de 100 Hz.

Figura 158. **Circuito RLC con muestreo**



Fuente: elaboración propia, empleando Inkscape.

Si se toma como x_1 a la diferencia de potencial que se encuentra en la resistencia y x_2 a la diferencia de potencial en el capacitor, se puede escribir el siguiente espacio de estados:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & \frac{1}{RC}\\ \frac{-R}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ \frac{R}{L}\end{bmatrix} v_{in} \qquad \text{[Ec. 4.164]}$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} \qquad \text{[Ec. 4.165]}$$

Siendo *y* el valor observado, si tanto las entradas como las observaciones tienen perturbaciones en forma de ruido el estado de espacios puede escribirse de la siguiente manera:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & \frac{1}{RC}\\ \frac{-R}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ \frac{R}{L}\end{bmatrix} v_{in} + \begin{bmatrix}v_1\\ v_2\end{bmatrix} \qquad \text{[Ec. 4.166]}$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + w \qquad \text{[Ec. 4.167]}$$

Si se tiene un espacio de estados lineal de tiempo continuo, con un vector de estados x(*t*) que define un estado para cada instante *t* en un conjunto de tiempo continuo y un vector de salidas u observaciones y(*t*) también definido en el mismo conjunto de tiempo continuo. Las matrices A_c , B_c , C_c , D_c definen la dinámica del sistema y las salidas.

$$\dot{x}(t) = A_c x(t) + B_c \boldsymbol{u}(t) \qquad \text{[Ec. 4.168]}$$

$$y(t) = C_c x(t) + D_c \boldsymbol{u}(t) \qquad \text{[Ec. 4.169]}$$

Si se considera solo una cantidad contable de estados del sistema de tiempo continuo, los cuales son equidistantes en el tiempo por un período T , es posible representar las muestras del sistema de tiempo continuo en la forma:

$$\dot{x}(kT) = A_c x(kT) + B_c \boldsymbol{u}(kT) \quad \text{[Ec. 4.170]}$$

$$y(kT) = C_c x(kT) + D_c \boldsymbol{u}(kT) \quad \text{[Ec. 4.171]}$$

Donde $k \in Z$, dejando así de ser un sistema de tiempo continuo para pasar a ser un sistema de tiempo discreto. Al ser un sistema de tiempo discreto se tiene una cantidad contable de estados, cuya referencia se encuentra dada por k. Es posible representar los estados del sistema de tiempo continuo en base a su estado anterior, en una forma equivalente:

$$x_{k+1} = A_d x_{k+} B_d \boldsymbol{u}_k \quad \text{[Ec. 4.172]}$$

$$y_k = C x_{k+} D_d \boldsymbol{u}_k \quad \text{[Ec. 4.173]}$$

Donde las expresiones de los estados son equivalentes a:

$$\boldsymbol{x}_k = \boldsymbol{x}(kT) \quad \text{[Ec. 4.174]}$$

$$\boldsymbol{u}_k = \boldsymbol{u}(kT) \quad \text{[Ec. 4.175]}$$

$$\boldsymbol{y}_k = \boldsymbol{y}(kT) \quad \text{[Ec. 4.177]}$$

Si A_c no es una matriz singular, se tiene para las matrices del sistema muestreado:

$$A_d = e^{A_c T} = \mathcal{L}^{-1}(sI - A_c)\Big|t = T^{=I+A_cT+A_c^2\frac{T^2}{2}+\cdots A_c^n\frac{T^n}{n!}+\ldots} \quad \text{[Ec. 4.178]}$$

$$B_d = \left(\int_{\tau=0}^{T} e^{A_c\tau}d\tau\right)B_C = A_C^{-1}(A_d - 1)B \quad \text{[Ec. 4.179]}$$

$$C_d = C_c \quad \text{[Ec. 4.180]}$$

$$D_d = D_c \quad \text{[Ec. 4.181]}$$

Numéricamente, el espacio de estados para el circuito RLC es el siguiente:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1\,000 \\ -1\,000000 & -1\,000000 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1\,000000 \end{bmatrix} v_{in} \quad \text{[Ec. 4.182]}$$

$$y = [1\ 0]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{[Ec. 4.183]}$$

Si el muestreo se realiza a 10 kHz, se puede escribir el espacio de estados muestreados numéricamente como:

$$\begin{bmatrix} x1_{k+1} \\ x2_{k+1} \end{bmatrix} = \begin{bmatrix} 0{,}9056542 & 0{,}0009066 \\ -0{,}9065617 & -0{,}0009075 \end{bmatrix}\begin{matrix} X1_k \\ X2_k \end{matrix} + \begin{bmatrix} 0{,}0943458 \\ 0{,}9065617 \end{bmatrix} v_{in\,k} \quad \text{[Ec. 4.184]}$$

$$y_k = [1\ 0]\begin{bmatrix} x1_k \\ x2_k \end{bmatrix} \quad \text{[Ec. 4.185]}$$

De esta forma es posible predecir el siguiente estado, teniendo un estado actual, para la implementación del filtro de Kalman.

Utilizando SCILAB, se simulará un instrumento que toma muestras con ruido de un circuito RLC como se muestra en la figura 158, cuya fuente también se encuentra expuesta a ruido. Para la simulación se declaran las constantes, las matrices del sistema continuo y el período del muestreo.

```
R=1 e3; L=1e -3; C=1e -6; // Valor de la resistencia , inductancia y Capacitancia .
Ac =[0 , 1/( R*C); -R/L, -R/L] // Matriz A de sistema continuo
Bc =[0; R/L];                  // Matriz B de sistema continuo
Cc =[1 ,0];                    // Matriz C de sistema continuo
T =1/1 e4;                     // Peri ódo de muestreo
```

Luego se simula el sistema ideal, sin ruido, utilizando el comando csim, la señal de entrada será una onda de 100 Hz y se considerará para la simulación dos períodos de la señal de entrada.

```
sys = syslin ('c', Ac ,Bc ,Cc);      // Espacio de estados
t=0:T:2/f-T;                         // Vector de Tiempo
u=sin (2* %pi*f.*t);                 // Señal de entrada
xideal = csim (u, t, sys);           // Simulaci ón

[m,n]= size ( xideal );              // Tama ño del vector de simulaci ón
```

Se almacena el tamaño de la matriz resultante de la simulación en las variables m y n, que debe ser el mismo del vector de tiempo t. Luego se escriben las condiciones iniciales para el filtro de Kalman, y los valores de las desviaciones del ruido presente en las mediciones y el sistema.

```
desvObs =0.08;                       // Desviaci ón de la observaci ón
desvEstado =0.01;                    // Desviaci ón del estado
Q=( desvEstado ^2)*eye (2 ,2); // Matriz de covarianzas del sistema
R= desvObs ^2;                       // Matriz de covarianzas de la observaci ón
P=Q;                                 // Matriz de covarianzas de los estados
X =[0;0]                             // Estado inicial
```

Para realizar la etapa de predicción es necesario un modelo discreto del sistema, ya que se debe predecir las magnitudes que caracterizan un estado

siguiente teniendo las de uno actual. Para ello se utilizan las ecuaciones para darle forma a un sistema continuo discretizado. La matriz A_d de la ecuación 4.177, se puede tener su valor numérico utilizando en SCILAB la instrucción expm, o bien haciendo una iteración muy larga de su serie de Taylor.

```
Ad= expm (Ac*T);
Bd=inv (Ac)*(Ad -eye(Ad))*Bc
Cd=Cc;
```

Se declaran los vectores que almacenarán los valores observados y los obtenidos por el filtro.

```
xreal =[]
xfiltro =[]
```

Para la construcción del filtro es necesario recorrer todas las posiciones que almacenan los valores ideales y se sumarán valores aleatorios distribuidos de forma Gaussiana para así simular un entorno ruidoso. Para ello se utiliza un ciclo y en cada iteración se realiza la etapa de la predicción y actualización.

```
1  ||for k=1:n
2  ||       // Etapa de predicci ón del siguiente estado .
3  ||
4  ||       // Se agrega ruido en la señal de entradad para simular
5  ||       // un ambiente ruidoso .
6  ||       X=Ad*X+Bd *(u(k)+ grand (1, 'nor ' ,0, desvEstado ));
7  ||        // Estimaci ón de la covarianza de los estados
8  ||       P=Ad*P*Ad '+Q;
9  ||       // Ruido en la lectura .
10 ||       z= xideal (k)+ grand (1,1 , 'nor ' ,0, desvObs );
11 ||       // Estimaci ón del error .
12 ||       Y=z-Cd*X;
13 ||       // Calculo de la ganancia de Kalman
14 ||       S=Cd*P*Cd '+R;
15 ||        K=P*Cd '* inv(S);
16 ||
17 ||        // Etapa de Correci ón.
18 ||
19 ||        //Cá lculo del valor del estado siguiente .
20 ||       X=X+K*Y;
21 ||       // Se agrega el estado actual y la lectura real
```

```
22 ||         // para luego graficar .
23 ||         xfiltro =[ xfiltro ,X];
24 ||         xreal =[ xreal ,z];
25 ||         // Actualizaci ón de la covarianza de los estados ,
26 ||        // para estado siguiente .
27 ||        P=( eye (2 ,2) -K*Cd)*P;
28 || end
```

La instrucción *grand* en SCILAB genera una matriz aleatoria de valores aleatorios de acuerdo a una distribución descrita en sus parámetros. Los parámetros que toma son las filas, columnas de la matriz a generar; y dependiendo de la distribución deseada los parámetros que la caracterizan, en caso de ser una normal, parámetro ’nor’, sus parámetros son su media y su desviación estándar. La fuente es considerada con ruido, ya que se han considerado los componentes pasivos del circuito como ideales, la única perturbación en el sistema provendrá de la fuente y es por ello que la matriz *Q* se ha calculado únicamente considerando la covarianza del ruido en la fuente.

En cada iteración se realizó la etapa de predicción con base del estado actual, luego se hace un estimado de la covarianza, se toma una lectura por el instrumento, la cuál puede modelarse como la lectura ideal con ruido en el sistema. Se calcula la ganancia de Kalman, con base a la deducción desarrollada con anterioridad. Se calcula el estado siguiente para luego en base a él actualizar la matriz de covarianzas P_{k+1} y así repetir el ciclo tomando como estado actual el estado siguiente calculado en la iteración anterior.

Para graficar el resultado del filtro se utiliza el comando plot2d, y se agregan las etiquetas para diferenciar las gráficas de cada uno de los resultados.

```
plot2d ( linspace (1,n,n) ,[ xfiltro (1 ,:) ’,xideal ’,xreal ’])
set (gca () , " auto_clear ","off ")
legend ([ ’$x_ {\ mathrm { Kalman }}$’;’$x_ {\ mathrm { ideal }}$’;’$x_ {\ mathrm { ruido }}$’])
```

```
set (gca () , " auto_clear ", "off ")
```

Apreciando los resultados en la figura 159, se puede ver que el resultado a pesar del ruido en la fuente y en el instrumento de medición, se asemeja mucho a la señal ideal.

Figura 159. **Filtro de Kalman sobre circuito RLC en entorno ruidoso**

Fuente: elaboración propia, empleando Python, librería Matplotlib.

- Filtro de kalman en lectura de giroscopio y acelerómetro

Un acelerómetro puede ser utilizado para conocer la orientación actual del sistema local de coordenadas en el sensor si este se encuentra estático, considerando la gravedad como un vector hacia afuera de la superficie de la Tierra; con las proyecciones de este vector sobre cada uno de los ejes del sistema local se puede calcular la inclinación del plano que representa cada eje. Por ejemplo, si el eje *Z* del sistema local coincide con el vector de gravedad, el plano que representa el vector *Z* (donde se encuentran los ejes *X* y *Y*) tendrá un

ángulo de inclinación equivalente a cero, con respecto a la superficie de la Tierra. Las proyecciones de la gravedad g_x , g_y y g_z , en cada uno de los ejes en el sistema local *X*, *Y* y *Z* son leídos por el acelerómetro de 3 ejes y sin importar la orientación que este tenga la siguiente ecuación es cierta:

$$g^2 = g_x^2 + g_y^2 + g_z^2 \qquad \text{[Ec. 4.186]}$$

Por tanto, es posible calcular las inclinaciones de los planos representados por cada eje del sensor, respecto al plano que representa la gravedad:

$$\propto_{YZ} = artan\left(\frac{g_x}{\sqrt{g_y^2+g_z^2}}\right) \qquad \text{[Ec. 4.187]}$$

$$\propto_{ZX} = artan\left(\frac{g_x}{\sqrt{g_Z^2+g_X^2}}\right) \qquad \text{[Ec. 4.188]}$$

$$\propto_{XY} = artan\left(\frac{g_x}{\sqrt{g_X^2+g_Y^2}}\right) \qquad \text{[Ec. 4.189]}$$

Dado que el acelerómetro presenta ruido en sus lecturas, un filtro de Kalman ofrece una solución práctica para obtener mediciones más próximas a la orientación real del sensor. Para la etapa de predicción del filtro es necesario considerar un modelo lineal del siguiente estado. En esta tarea resulta útil un giroscopio, el cual obtiene lecturas de la velocidad angular de los ejes.

Figura 160. **Sistema de coordenadas local del sensor del acelerómetro y giroscopio**

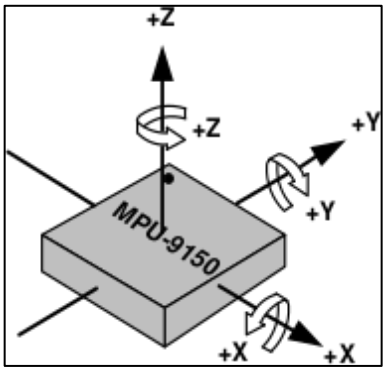


Figura: *MPU-9150 Product Specification. Document Number PS-MPU-9150A-00. Revisión 4.3*. https://store.invensense.com/Datasheets/invensense/RM-MPU-9150A-00-v3.0.pdf- Consulta: noviembre de 2015.

Utilizando las lecturas del giroscopio u_k, se puede aproximar el ángulo de inclinación a_{k+1} que tendrán los planos, luego de transcurrido un tiempo desde una lectura del ángulo a_k obtenida con el acelerómetro. Sea Δt el tiempo que ha transcurrido desde la última lectura del acelerómetro, por tanto la siguiente lectura del ángulo a_{k+1} puede ser estimada de la siguiente forma:

$$a_{k+1} \approx a_k + u_k \Delta t \qquad \text{[Ec. 4.190]}$$

La ecuación anterior muestra una situación ideal. Sin embargo, las lecturas del giroscopio presentan sesgo, siendo contante genera un error en la lectura del ángulo, que se incrementa al pasar el tiempo. Por tanto, en la estimación del ángulo a_{k+1}, se debe retirar el valor del sesgo ϵ_k de las lecturas u_k para tener una estimación confiable.

$$a_{k+1} \approx a_k + (u_k - \epsilon_k)\Delta t \qquad \text{[Ec. 4.191]}$$

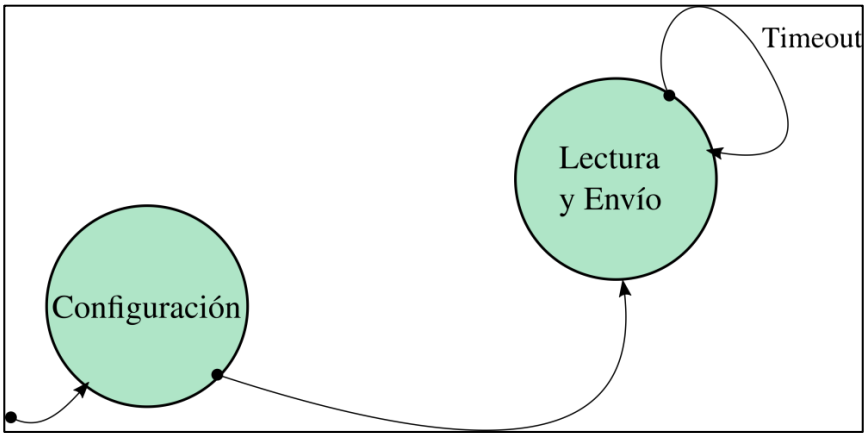
Mientras Δt sea más pequeño, la ecuación estará más cerca de la realidad. No es necesario calcular el valor del sesgo, ya que las lecturas del giroscopio, al igual que las del acelerómetro, presentan ruido y el valor puede variar en diferentes estimaciones. Se puede estimar su valor haciéndolo parte del estado actual, junto con el ángulo y el filtro utilizará las medidas del acelerómetro para estimar su valor. De esa cuenta es posible plantear el siguiente espacio de estados discreto para estimar el ángulo de inclinación en un futuro, conocido el ángulo de inclinación actual.

$$\begin{bmatrix} a_{k+1} \\ a_{\epsilon+1} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_k \\ \epsilon_k \end{bmatrix} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \quad \text{[Ec. 4.192]}$$

$$y_k = [1\ 0] \begin{bmatrix} a_k \\ \epsilon_k \end{bmatrix} \quad \text{[Ec. 4.193]}$$

Para poder utilizar el espacio de estados en un filtro de Kalman, se debe implementar en el microcontrolador una dinámica discreta que tome lecturas espaciadas por tiempos iguales Δt.

Figura 161. **Lectura y envío periódico de datos**



Fuente: elaboración propia, empleando Inkscape.

Las lecturas y el envío de datos se harán de manera periódica. Como ejemplo se presenta el código en C para la implementación de la dinámica en el microcontrolador tm4c123gh6pm. Se comienza agregando las librerías que serán útiles en el uso del sensor, el módulo de UART, el módulo de I2C y los temporizadores; además se debe definir al principio que tipo de dispositivo se está utilizando para que las librerías realicen los mapeos correctos al momento de compilar.

```
# define PART_TM4C123GH6PM

# include <stdint .h>
# include <stdbool .h>
# include " inc / hw_memmap .h"
# include " inc / hw_types .h"
# include " inc / hw_ints .h"
# include " driverlib / gpio .h"
# include " driverlib / pin_map .h"
# include " driverlib / sysctl .h"
# include " driverlib / uart .h"
# include " utils / uartstdio .h"
# include " sensorlib / i2cm_drv .h"
# include " sensorlib / ak8975 .h"
# include " sensorlib / hw_mpu9150 .h"
# include " sensorlib / mpu9150 .h"
# include " driverlib / timer .h"
```

Luego se declaran las constantes y variables que se utilizarán durante la dinámica. El reloj del microcontrolador se ha puesto a 80 MHz y para la frecuencia de muestreo se consideraron 150 muestras por segundo.

```
# define clock 80000000
# define samplefreq 150
tI2CMInstance I2CInst ;
tMPU9150 MPU9150Inst ;
volatile bool MPU9150Done = false ;
```

Para escoger una frecuencia de muestreo se debe tener en cuenta que la bandera TimeoutFlag no se active antes de terminar la lectura y envió de datos para

evitar comportamientos extraños, 150 muestras por segundo permiten al microcontrolador obtener la información del sensor usando I2C y enviar por completo los datos usando el protocolo UART.

Se declara la rutina usada por la máquina de estados que controla el protocolo I2C para la comunicación con el sensor MPU9150.

```
void MPU9150Callback ( void * pvCallbackData , uint_fast8_t ui8Status )
{
if( ui8Status != I2CM_STATUS_SUCCESS )
{
        // Si sucede un error , ejecutar el có digo de aquí.
}
// Indica que la transacci ón ha sido terminada .
MPU9150Done = true ;
}
```

Dado que la dinámica discreta, implementada en la librería, que controla el flujo de información del sensor al microcontrolador, se basa en interrupciones se debe declarar la rutina que las controla para el I2C.

```
void IntHandlerI2C1 ( void ){
I2CMIntHandler (& I2CInst );
}
```

Para poder controlar de forma precisa el tiempo se utilizan de igual forma interrupciones, que inicien la rutina de lectura de forma periódica de esa forma pueda el microcontrolador leer y enviar datos.

```
void IntHandlerTimer0A ( void ){
TimerIntClear ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
MPU9150Done = false ;
MPU9150DataRead (& MPU9150Inst , MPU9150Callback , 0);
}
```

Se han agrupado en rutinas las configuraciones tanto del módulo I2C como las del temporizador, el modulo de *Timer* que se utiliza es el Timer0 y se configura para realizar conteos periódicos.

```
void I2C1Configure ( void ){
IntEnable ( INT_I2C1 );
IntPrioritySet ( INT_I2C1 , 0x00);
I2CMInit (& I2CInst , I2C1_BASE , INT_I2C1 , 0xff , 0xff , clock );
}
void Timer0Configure ( void ){
uint32_t TimerCount ;
TimerConfigure ( TIMER0_BASE , TIMER_CFG_PERIODIC );
TimerCount =( clock / samplefreq );
TimerLoadSet ( TIMER0_BASE , TIMER_A , TimerCount -1) ;
IntEnable ( INT_TIMER0A );
IntPrioritySet ( INT_TIMER0A , 0 x00 );
TimerIntEnable ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
}
```

Una vez configurado el módulo de I2C y creadas las instancias necesarias por la librería para control del MPU9150, es posible configurar el sensor para funcionar de la forma deseada, en este caso se configura para el acelerómetro un rango de [-4*g* ,4*g*].

```
void MPU9150Configure ( void ){
      // Arranca el MPU9150 .
      MPU9150Done = false ;
      MPU9150Init (& MPU9150Inst , & I2CInst , 0x68 , MPU9150Callback , 0);
      while (! MPU9150Done )
      {}
      // Configura el MPU9150 para un rango de +/- 4 g.
      MPU9150Done = false ;
      MPU9150ReadModifyWrite (& MPU9150Inst , MPU9150_O_ACCEL_CONFIG ,~
      MPU9150_ACCEL_CONFIG_AFS_SEL_M ,
 MPU9150_ACCEL_CONFIG_AFS_SEL_4G , MPU9150Callback ,0) ;
      while (! MPU9150Done ){
      }
}
```

Se escribe la dinámica en la rutina principal de la lectura y el envío de datos en la rutina principal main, primero se configura de forma general el

dispositivo, utilizando las rutinas ya escritas para la configuración de los periféricos y el sensor. Para el módulo de UART se consideró trabajar a 230 400 baudios para tener el envío de datos lo más rápido posible. El programa no espera a que se termine de enviar, lo coloca en una cola que se está vaciando fuera del programa, sin embargo, si las lecturas son muy rápidas y el envío no se ha completado se puede desbordar la cola y provocar que no funcione adecuadamente.

```
1  || void main ( void ){
2  ||    uint32_t Accel [3] , Gyro [3] , Magneto [3];
3  ||    /* Configure System Clock */
4  ||    SysCtlClockSet ( SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
       SYSCTL_XTAL_16MHZ );
5  ||    /* Configure GPIOA Clock */
6  ||    SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOA );
7  ||     /* Configure GPIOA Clock */
8  ||    SysCtlPeripheralEnable ( SYSCTL_PERIPH_UART0 );
9  ||    /* Configure I2C1 Clock */
10 ||    SysCtlPeripheralEnable ( SYSCTL_PERIPH_I2C1 );
11 ||    /* Configure Timer0 Clock */
12 ||    SysCtlPeripheralEnable ( SYSCTL_PERIPH_TIMER0 );
13 ||    /* Configure I2C */
14 ||    GPIOPinConfigure ( GPIO_PA7_I2C1SDA );
15 ||    GPIOPinConfigure ( GPIO_PA6_I2C1SCL );
16 ||    /* Configure I2C DATA */
17 ||    GPIOPinTypeI2C ( GPIO_PORTA_BASE , GPIO_PIN_7 );
18 ||     /* Configure I2C SCLK */
19 ||    GPIOPinTypeI2CSCL ( GPIO_PORTA_BASE , GPIO_PIN_6 );
20 ||
21 ||     GPIOPinTypeUART ( GPIO_PORTA_BASE , GPIO_PIN_0 | GPIO_PIN_1 );
22 ||    UARTStdioConfig (0, 230400 , clock );
23 ||
24 ||    I2C1Configure ();
25 ||    Timer0Configure ();
26 ||    IntMasterEnable ();
27 ||     MPU9150Configure ();
28 ||
29 ||    UARTprintf (" Comenzando > \n");
30 || TimerEnable ( TIMER0_BASE , TIMER_A );
31 ||
32 ||    while (1) {
33 ||        if( MPU9150Done ){
34 ||        MPU9150DataAccelGetRaw (& MPU9150Inst , & Accel [0] , & Accel [1] ,& Accel [2]) ;
35 ||            MPU9150DataGyroGetRaw (& MPU9150Inst , & Gyro [0] , & Gyro [1] , & Gyro [2]) ;
36 ||            MPU9150DataMagnetoGetRaw (& MPU9150Inst , & Magneto [0] ,
37 ||                                        & Magneto [1] ,& Magneto [2]) ;
```

```
38 ||                    UARTprintf (" Sensor :%d:%d:%d:%d:%d:%d:%d:%d:%d\n",
39 ||                                Accel [0] , Accel [1] , Accel [2] , Gyro [0] , Gyro [1] , Gyro [2] ,
40 ||                                    Magneto [0] , Magneto [1] , Magneto [2]) ;
41 || MPU9150Done = false ;
42 || }
43 || }
44 ||}
```

En el ciclo continuo siempre verifica que el sensor ha terminado de leer los datos para poder enviarlos por UART, al terminar el envío coloca la bandera MPU9150Done en bajo para evitar volver a enviar la misma información. La rutina volverá a ejecutarse cuando el temporizador termine su conteo del tiempo y ejecute la rutina IntHandlerTimer0A donde iniciará la lectura del sensor nuevamente. Para que el código anterior funcione se deben declarar y configurar las interrupciones en el archivo que almacena el vector de interrupciones.

Una vez se tiene una dinámica discreta implementada en el microcontrolador, capaz de entregar muestras periódicas, se escribe en Python un programa que tome las muestras y pueda filtrar el ruido del sensor. Python es un lenguaje interpretado a diferencia de C que es compilado. Una explicación detallada del lenguaje queda fuera del alcance del presente trabajo, solamente se limita explicar el funcionamiento del programa que implementa el filtro como un ejemplo. Al utilizar la librería numpy, permite un uso muy intuitivo de las estructuras de datos en forma de matrices, muy similar a SCILAB.

Se declaran las librerías que permiten el uso del puerto serial, las gráficas de datos y el uso de estructuras similares a SCILAB.

```
import serial
import matplotlib . pyplot as plt
import numpy as np
```

Algunas constantes útiles en la estructura del programa. La variable dt corresponde al período de muestreo y RAD2Grad. Las variables q1 es la varianza medida sobre el sesgo del giroscopio, para estimar su valor se calculó 100 veces la pendiente que existe al dejar el giroscopio estático y calcular el ángulo acumulado por las lecturas del giroscopio. El valor de la variable q2 es la desviación estándar en las mediciones del giroscopio, multiplicado por el tiempo de esa forma se tiene una aproximación para la varianza del estado teniendo como única entrada al sistema discreto la lectura del giroscopio. Utilizando las variables q1 y q2 se construye la matriz *Q* que será útil en la implementación del filtro.

```
Acc =0.0011970964;
Gyr =1.3323124e -4;
Mag =0.0000003;
dt =1./150.;
RAD2Grad =57.295779;
q2 =((1.36442463992 e -06) **2)
q1 =((0.0955588888617) **2) *( dt **2)
```

Se construye la función que permite estimar los valores apropiados a los sensores, recibiendo como parámetros la sensibilidad y el sesgo. A pesar que se tiene sesgo en las lecturas del giroscopio se estima, se toma su como cero en la función ya que el filtro se encargará de estimar su valor más adelante.

```
def Afin (x,a,b):
        x= int (x)
        x=x -(x >> 15) * 65536;
        return x*a+b;
```

Se declara el objeto que realizará las lecturas a partir del puerto serial. Los arreglos que albergarán las lecturas, el arreglo que almacenará los valores filtrados y el número de muestras a tomar.

```
ser = serial . Serial ('/ dev / ttyACM0 ', 230400 , timeout =1) ; # UART
```

```
Ax =[]; Ay =[]; Az =[];
Gx =[]; Gy =[]; Gz =[];
Mx =[]; My =[]; Mz =[];

KangX =[];
samples =1500;
```

Se toman los valores de los 3 ejes del sensor, se aplica a cada uno de ellos la función correspondiente para convertirlos a los valores reales y se almacenan en las variables ya declaradas. Ya que el presente programa solo tiene un fin demostrativo del filtro de Kalman en las lecturas de un sensor, se calcula el ángulo de inclinación de solo un plano, en este ejemplo solo el plano que representa el eje X del giroscopio y acelerómetro, al terminar de capturar datos se hace el cálculo de la inclinación. El método numpy.artan devuelve un vector del mismo tamaño del vector dado como parámetro cuyos elementos son el resultado de la operación de arcotangente de cada elemento del vector parámetro.

```
n =0;
while n< samples :
        vec =[]
        sensor =[]
        data = ser . readline ()
        vec = data . split ('\r\n')
        print data
        sensor = vec [0]. split (':')
        header = sensor [0];
if( header ==" Sensor " and len( sensor ) ==10) :
Ax. append ( Afin ( sensor [1] , Acc ,0) );Ay. append ( Afin ( sensor [2] , Acc ,0) );Az.
append ( Afin (
        sensor [3] , Acc ,0));
Gx. append ( Afin ( sensor [4] , Gyr ,0) );Gy. append ( Afin ( sensor [5] , Gyr ,0) );Gz.
append ( Afin (
        sensor [6] , Gyr ,0) );
Mx. append ( Afin ( sensor [7] , Mag ,0) );My. append ( Afin ( sensor [8] , Mag ,0) );Mz.
append ( Afin (
        sensor [9] , Mag ,0) );
n=n +1;
AnX =np. arctan (Ax ,np. sqrt (np. power (Az ,2)+np. power (Ay ,2) ));
```

El valor de Rx es la varianza en las lecturas, la lectura se hace sobre el valor del acelerómetro, por tanto, se tomaron varias muestras y se obtuvo la desviación. Para la matriz *Q* se utilizaron los valores de las variables q1 y q2. Tal como el ejemplo en el circuito RLC la matriz *P* tiene el mismo valor que la matriz *Q* al inicio. El primer valor filtrado tendrá la primera lectura del acelerómetro. Luego se colocan los valores iniciales para el estado, y los valores de las matrices. El valor inicial del estado corresponde a la inclinación y al sesgo. El primero es obtenido a partir de la primera lectura del acelerómetro y el segundo a partir de la media calculada en conjunto con la desviación estándar en las 100mediciones. El método numpy.mat transforma una estructura de vector en un objeto que se interpreta como matriz por otros métodos de numpy

```
Rx =0.009458837762**2;
Qx=np. array ([[ q1 ,0] ,[0 , q2 ]]) ;
Px=np. array ([[ q1 ,0] ,[0 , q2 ]]) ;
KangX =[ AnX [0]]
n =1;
x=np. mat (np. array ([[ AnX [0]] ,[ -0.000208640247325]]) )
A=np. mat (np. array ([[1 , -dt ] ,[0 ,1]]) )
B=np. mat (np. array ([[ dt ] ,[0]]) )
C=np. mat (np. array ([1 ,0]) );
```

Una vez inicializados los valores de las variables a utilizar, se implementa el filtro. Utilizando algunos métodos útiles de la librería numpy, se puede operar matrices. El método numpy.dot da como resultado el producto de dos matrices, el método numpy. transpose da como resultado una estructura de datos equivalente a la transpuesta de la matriz otorgada como parámetro.

```
while (n< samples ):
x=np. dot (A,x)+Gx[n -1]* B; # Predicci ón
Px=np. dot (A,np. dot (Px ,np. transpose (A)))+Qx; # Estimaci ón para Px
y= AnX [n]-np. dot (C,x); # Estimaci ón de error en base a lectura de aceler ó metro
#Cá lculo de la ganancia de Kalman .
S=np. dot (C,np.dot (Px ,np. transpose (C)))+Rx;
K=np. dot (Px , np. transpose (C))/S;
# Etapa de correcci ón del filtro
x=x+K*y. item (0) ;
```

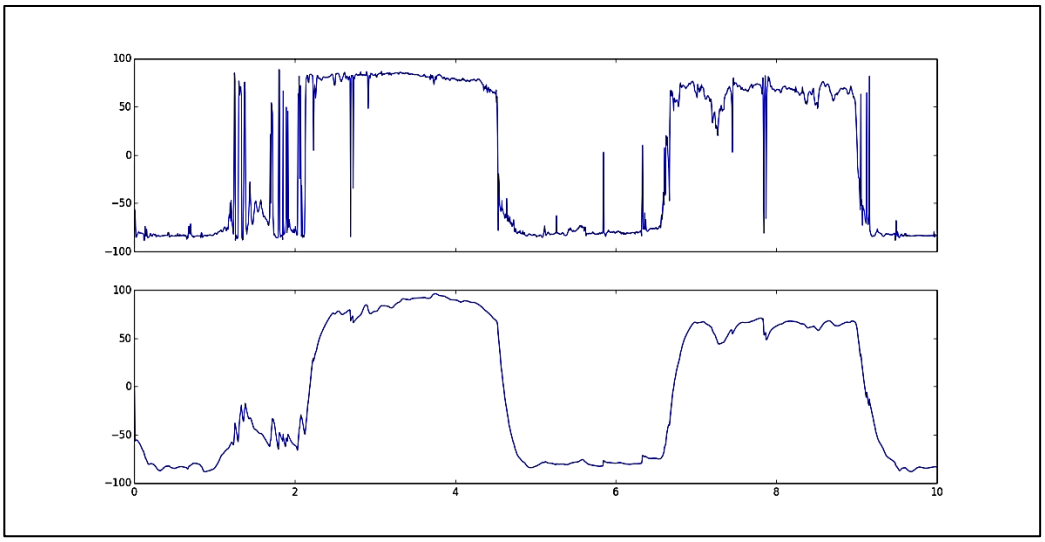
```
#Se agrega la nueva lectura filtrada al vector KangX
KangX . append ( RAD2Grad *np.dot(C,x). item (0) );
#Se actualiza la matriz Px con las nuevas covarianzas
Px=np. dot (( np. mat (np. eye (2 ,2))-np. dot (K,C)),Px);
#Se pasa a la siguiente iteraci ón.
n +=1;
```

La variable `x` contiene el valor del estado actual, el cual se sobrescribe con el valor de la predicción en cada iteración. Se calcula la matriz de covarianzas para la predicción y se estima la diferencia del valor de la predicción con el valor de la lectura que se hace sobre la inclinación calculada usando el acelerómetro. Luego se obtiene la ganancia de Kalman y con ella se hace la corrección de la lectura. Al tener estimado el estado que minimiza el error cuadrático medio con el valor real se almacena su valor (en grados para una mejor interpretación por un humano) en el vector `KangX` y se actualiza la matriz de covarianzas para ser utilizada en la siguiente iteración. Una vez terminado el ciclo se pueden comparar los resultados. Al realizar las gráficas con la librería `matplotlib`, se puede observar una sustancial diferencia entre los valores filtrados y los valores obtenidos.

```
t=np. linspace (0, len(Ax) -1, len (Ax));
f, (( GAx ) ,( GKangX ))= plt . subplots (2 ,1 , sharex =’col ’, sharey =’row ’);
AnX = AnX * RAD2Grad ;
GAx . plot (t*dt , AnX);
GKangX . plot (t*dt , KangX );
plt . show ();
```

Para comprobar los resultados del filtro se mueve el sensor de forma aleatoria durante 10 segundos (equivalente a 1 500 muestras tomadas a una frecuencia de muestreo de 150 Hz), se pueden apreciar en la figura 162, en la gráfica superior las lecturas obtenidas directamente del sensor y en la gráfica inferior el resultado del filtro.

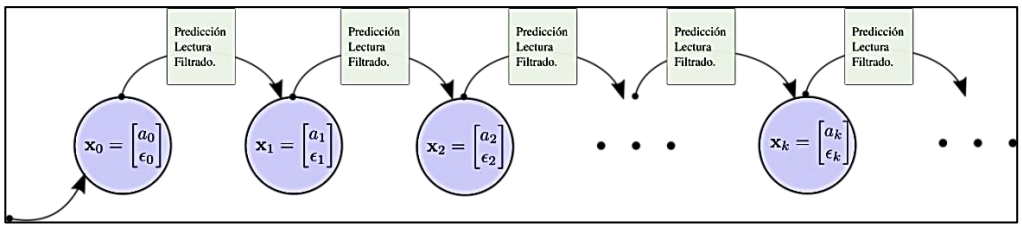
Figura 162. **Gráfica de valores leídos y valores estimados de una trayectoria aleatoria usando el giroscopio y acelerómetro de lMPU9150**



Fuente: elaboración propia, empleando Python, librería Matplotlib.

Si se desea implementar el filtro en un microcontrolador, se debe tener en consideración el tiempo que le tome a este realizar todos los cálculos del filtro, además debe tenerse en cuenta la precisión al trabajar con punto fijo o punto flotante, la implementación puede ser muy diferente al utilizar punto fijo y punto flotante; en especial al trabajar con valores muy pequeños.

Figura 163. **Dinámica discreta de filtro de Kalman**



Fuente: elaboración propia, empleando Inkscape.

4.6.3. Lazo cerrado

Simulación de un control PID de un motor DC.

El control PID es el lazo cerrado más común de encontrar en una basta variedad de sistemas controlados. En el control de procesos es común el término *set-point*, el cual es un valor típicamente modelado como una constante que un lazo cerrado desea alcanzar luego de un tiempo su planta de control manipula el sistema controlado.

Una planta de control PID basa la retroalimentación en el error $e(t)$, resultado de la diferencia entre el *set-point* y_{sp} y la señal $y(t)$ que representa la salida del sistema de control. La planta controladora PID, toma el error, calcula su integral y su derivada en relación al tiempo y devuelve una combinación lineal de estos resultados junto con el error que ha tomado para el cálculo. La planta PID puede verse como un sistema cuya entrada es el error $e(t)$ y su salida $u(t)$ esta dada por:

$$u(t) = K\left(e(t) + \frac{1}{t_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \qquad \text{[Ec. 4.194]}$$

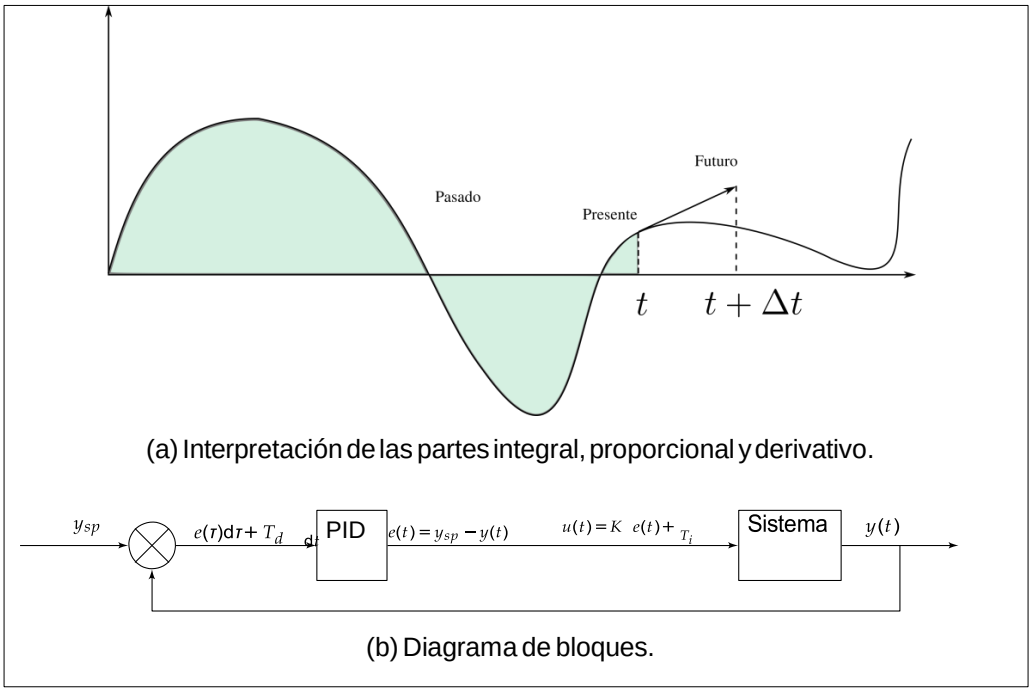
Donde K es un valor conocido como constante proporcional, el valor T_i es llamado tiempo de integración y T_d tiempo de derivación. En el sistema de control la señal $u(t)$ forma la salida del sistema controlador y al mismo tiempo la entrada del sistema controlado. Es posible escribir una expresión en el dominio de Laplace para:

$$U(s) = K(E(s) + I(s) + D(s)) \qquad \text{[Ec. 4.195]}$$

$$U(s) = K\left(E(s) + \frac{1}{sT_i}E(s) + sT_dE(s)\right) \qquad \text{[Ec. 4.196]}$$

La figura 164 muestra el diagrama de bloques de un lazo cerrado con un controlador PID, cuya entrada es un *set-point* y su salida es una variable medida en la salida del sistema controlado. La parte integral, proporcional y derivativa pueden interpretarse como el pasado, el presente y el futuro de la salida del sistema tal como se muestra en la figura 164.

Figura 164. **Control PID**



Fuente: elaboración propia, empleando Inkscape.

Un control proporcional o simplemente control P se consigue al hacer las constantes $T_i = \infty$ y $T_d = 0$, en este control el error disminuye al incrementar la ganancia *K*, sin embargo, en el sistema de control incrementa la tendencia a las

oscilaciones, tal como se muestra en la figura 164. El estado oscilatorio estable que aparece producto del control proporcional se contrarresta al introducir en el sistema el término integral. La parte integral se hace más notoria al hacer más pequeño el valor T_i , la tendencia a oscilaciones se incrementa al hacer más pequeño el valor de T_i , como se muestra en la figura 164. Los parámetros K y T_i son escogidos para que el sistema entre en estado oscilatorio. Se logra amortizar el sistema al introducir el término derivativo, sin embargo, si el valor T_d es muy grande deja de hacerlo. Como se ha mencionado, el valor derivativo en la salida $u(t)$, puede ser interpretado como una predicción (usando interpolación lineal) del estado futuro de la señal de salida, por tanto si el tiempo de derivación es muy largo es posible comprender porque no son útiles tiempos de predicción muy largos.

El control diferencial tiende a amplificar el ruido de alta frecuencia, esto se debe a la forma que tiene en el dominio de Laplace, $D(s) = sT_dE(s)$, implicando que el ruido de alta frecuencia tenderá a amplificar el error de una gran manera. En un controlador práctico es necesario limitar la ganancia de alta frecuencia de la componente derivativa. Esto se logra cambiando el término derivativo por la expresión:

$$D(s) = \frac{sK_pT_d}{1+sT_d/N}E(s) \qquad \text{[Ec. 4.197]}$$

De esta forma la ganancia para el error en altas frecuencias frecuencias podrá alcanzar a lo sumo el valor k_pN, donde N es un número que típicamente toma un valor entre 8 y 20 en las diversas implementaciones. La aproximación anterior puede ser vista como una derivada ideal sT_d que ha pasado por un filtro de primer orden con constante T_d /N.

La función de transferencia *C*(*s*), de la entrada *U*(*s*) del sistema para la salida para la planta de control estaría dada por la siguiente expresión:

$$C(s) = K\left(1 + \frac{1}{sT_i} + \frac{sT_d}{1+sT_d/N}\right) \qquad \text{[Ec. 4.198]}$$

Por tanto, si se tiene un sistema controlado con una función de transferencia *G*(*s*), para sus entradas y salidas respectivamente, la función de transferencia *H*(*s*) para el sistema de control, estaría dada por:

$$C(s) = \frac{G(s)C(s)}{1+G(s)H(s)} \qquad \text{[Ec. 4.199]}$$

Este sistema es considerado inestable cuando la función de transferencia diverge en el dominio de *s*, esto sucede en situaciones donde *G*(*s*)*H*(*s*) = -1. La estabilidad del sistema esta sujeta al criterio de Nyquist para la estabilidad.

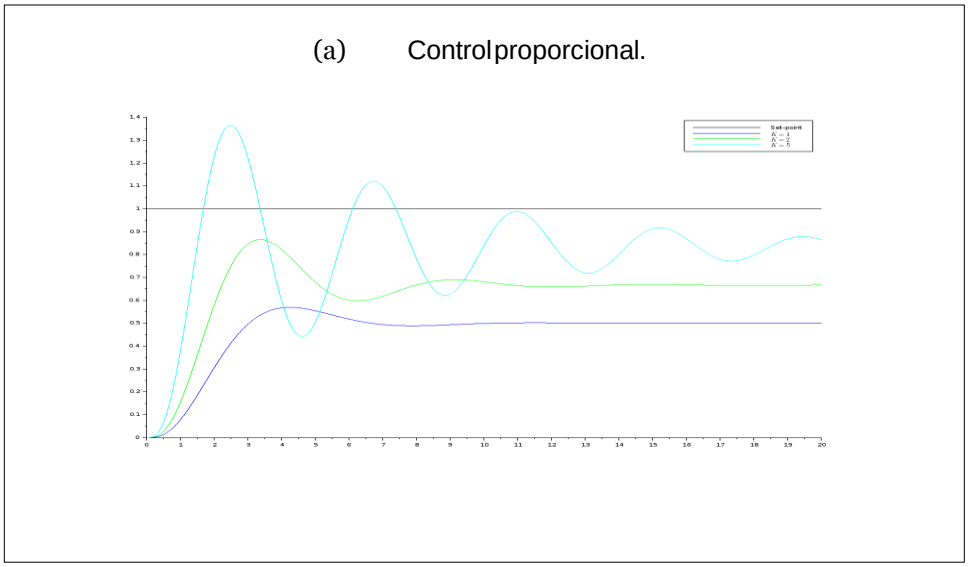
Se debe considerar que esta es una propuesta lineal para una planta de control, es posible entender muchos de sus efectos basados en teoría de sistemas lineales, algunas no-linealidades que presentan los sistemas controlados deben tomarse en cuenta. Un fenómeno conocido como *windup* aparece con la interacción de la parte integral de la planta y las saturaciones en los sistemas. Todos los actuadores que por lo general forman el sistema controlado, poseen limitaciones, por mencionar algunas: la velocidad en un motor eléctrico se ve limitada, una válvula no puede ser completamente abierta o cerrada, algunos amplificadores se saturan, entre otros.

En un control con un rango de aplicación amplio puede suceder que las variables de control alcancen su límite. Cuando esto sucede el lazo cerrado se ve roto, ya que el actuador estará en su límite independientemente de la salida de la planta. Si un sistema controlador con parte integral es utilizado el error se

continuará integrando continuamente. Esto significa que el término integral se volverá sumamente amplio. Entonces se requiere que el error tenga signo positivo por un periodo largo antes que las cosas regresen a la normalidad. Como consecuencia controladores con parte integral, tienen largos estados transitorios cuando un actuador se satura.

Otro problema que enfrentan los controladores PID es su calibración, suele ser difícil encontrar un conjunto de constantes que logren satisfacer las condiciones impuestas por la aplicación para la que se diseña el sistema de control. Que constante forman un control PID perfecto, la respuesta depende de los requerimientos de la aplicación.

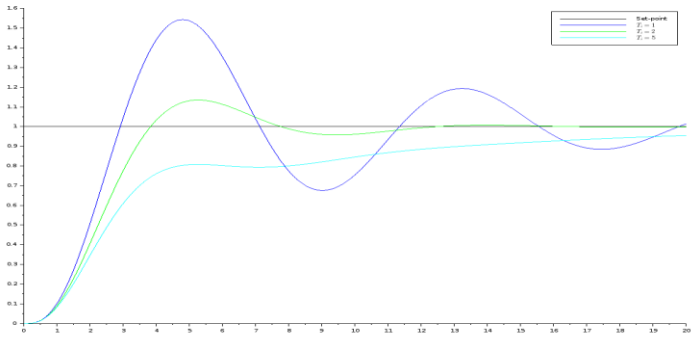
Figura 165. **Graficas de control PID**



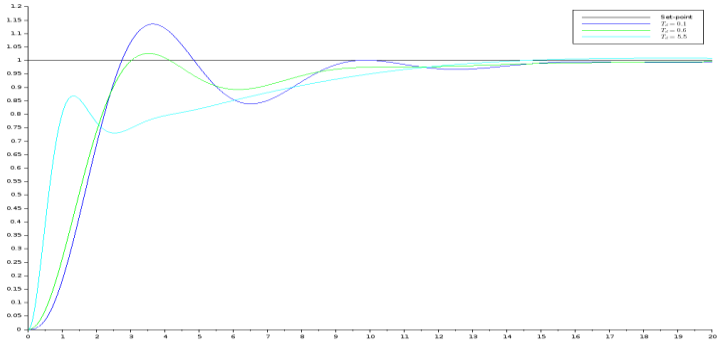
.

Continuación de la figura 165.

(b) Control proporcional integral



(c) Control proporcional, integral, derivativo.



Fuente: elaboración propia, empleando SCILAB.

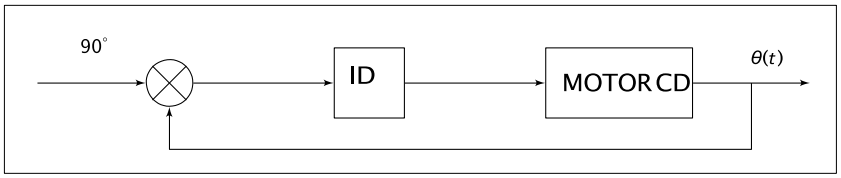
Como ejercicio práctico se proponen los siguientes enunciados:

- Considérese un motor ideal de corriente directa, modelado por las ecuaciones 4.82 y 4.83. Considérese para la inercia del motor J = 1,4 x10^{-6} kg Δm_2, una resistencia de 5,3-, una inductancia de 500¹H, debido a la estructura mecánica presenta una constante de fricción viscosa

B = 9,3 x10^{-5} kg m^2/s, considérese la constante electromotriz k_b = 0,02 V/(rad/s) y la constante del par k_m = 0,02 mNΔm/A.

Simular 10 veces un sistema de control de lazo cerrado, donde el motor representa el sistema controlado, la planta de control es un controlador PID y el *set-point* es 90°, En cada simulación se debe escoger un conjunto de constantes K, T_i y T_d distinto, mostrando los resultados en una sola gráfica. Si se toma el modelo de la planta PID tolerante a ruido de alta frecuencia tomar N = 10.

Figura 166. **Diagrama de bloques de control PID sobre motor CD**

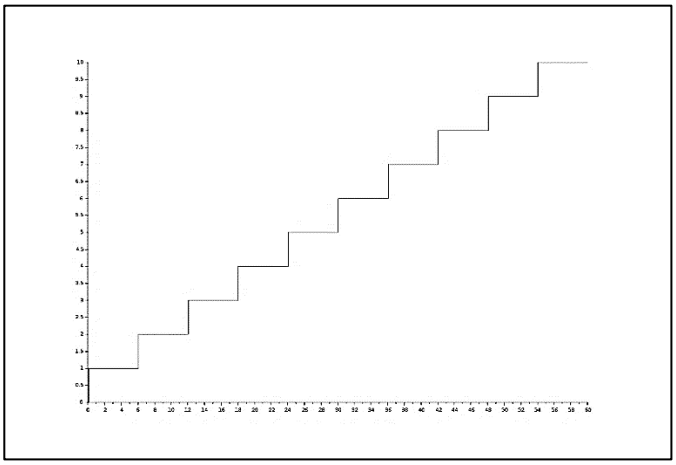


Fuente: elaboración propia, empleando Inkscape.

Utilizando un microcontrolador y un servomecanismo comercial generar gráficas que comparen el desempeño del lazo cerrado, implementado en el servomecanismo con una señal de entrada hacia el servo generada por el microcontrolador.

Para ello se debe generar una señal de entrada para el servo que permita explorar 10 posiciones consecutivas en un minuto, la señal debe generar los *set-points* de forma similar a la figura 167, con base en la forma de uso del servomotor escogido.

Figura 167. **Rampa de valores para Servo**



Fuente: elaboración propia, empleando SCILAB.

Los servomecanismos más económicos se componen de una planta de control analógica, y la posición angular del servo es medida con un potenciómetro. Desarmando la carcasa del servomecanismo es posible acceder al circuito de control, donde con un cable tipo caimán o punto de soldadura es posible leer la señal del potenciómetro utilizando un módulo de ADC del microcontrolador utilizado. Comparar en una misma gráfica los resultados obtenidos.

CONCLUSIONES

1. La humanidad está a pocos años de vivir en la IoT, por tanto, los ingenieros en las ramas de eléctrica, mecánica eléctrica y electrónica principalmente, deben de estar familiarizados con tecnologías digitales y ciencias de computación que permitan integrar su trabajo a una red de humanos y dispositivos.

2. Es importante estimular la investigación en esta área, ya que un mundo globalizado abre una ventana de oportunidad a nuevas industrias, cuyo seno se encuentra en nuevas implementaciones capaces de integrarse a las últimas tendencias.

3. Los modelos representan una parte sumamente importante en cualquier implementación y diseño de sistemas físicocibernéticos, permiten de forma estructurada y metódica predecir comportamientos y mejorar las aplicaciones.

4. El ingeniero que confunda el modelo con la implementación práctica está propenso a cometer errores que se reflejarán en el funcionamiento de un sistema.

5. El uso de dispositivos de computación permite un mayor alcance a las aplicaciones de control. Permiten procesamiento de señales, control de dispositivos, interacción con el usuario final de forma agradable y la comunicación del sistema con otros sistemas utilizando la internet como una red convergente.

RECOMENDACIONES

1. Proponer a las autoridades encargadas de la Escuela de Ingeniería de Mecánica Eléctrica, agregar como cursos optativos previos a Sistemas de Control un curso de álgebra lineal y de modelado y simulación que permita a los estudiantes profundizar más en la teoría de control.

2. Por lo amplio del contenido que representa la teoría de control, proponer a las autoridades encargadas de la Escuela de Ingeniería de Mecánica Eléctrica implementar un curso opcional que pueda dar continuidad al análisis de sistemas controlados.

3. Actualmente el curso de Robótica presente en el pénsum de Ingeniería en Electrónica, tiene como prerrequisito el curso de Electrónica 6. Las autoridades encargadas de la Escuela de Ingeniería en Mecánica Eléctrica deberían cambiar del prerrequisito de Electrónica 6 por el curso de Sistemas de Control. Permitiría a los alumnos llevar mejores bases hacia el curso de Robótica ya que el contenido del curso de Electrónica 6 se encuentra poco relacionado con esta área.

4. El estudiante debería acompañar el presente trabajo con textos paralelos relacionados con teoría de conjuntos y álgebra lineal para poder comprender de manera la simbología presente.

5. Al tutor o auxiliar de laboratorio que tome como base el presente trabajo, debe considerar que el capítulo más importante es el referente a sistemas híbridos para la construcción de sistemas de control (cuarto

capítulo), en él se consolidan todos los demás capítulos y el estudiante puede encontrar aplicaciones prácticas a la información expuesta en los anteriores. Sin embargo, no debe olvidar que los capítulos anteriores permiten al estudiante formar las bases para poder realizar los ejercicios en el capítulo 4.

BIBLIOGRAFÍA

1. ASHBY, William Ross. *An introduction to cybernetics*. London: Chapman & Hall, 1956. 295 p.

2. CATALDO, Adam. *Control algorithms* for *soft walls, research project*. Berkeley: Department of Electrical Engineering and Computer Sciences, University of Califor*n*ia, 2004. 339 p.

3. CELLIER, François. *Continuous system modeling*. New York: Springer-Verlag, 1991. 541 p.

4. CHEEVER, Erik. *Developing mathematical models of translating mechanical systems*. [en línea]. <http://lpsa.swarthmore.edu/Systems/MechTranslating /TransMechSysModel.html>. [Consulta: octubre de 2015].

5. CHOMSKY, Noam; COCKBURN, Alexander. *The golden age is in us, Noam Chomsky interviewed by Alexander Cockburn*. [en línea]. <https://chomsky.info/19940622/>. [Consulta: octubre de 2015].

6. *Condenser microphones*. [en línea]. <http://www.mediacollege.com/audio/microphones/condenser.html>. [Consulta: octubre de 2015].

7. *Damping*. [en línea]. <https://en.wikipedia.org/wiki/Damping>. [Consulta: octubre de 2015].

8. DE ROSNAY, Joel. *The macroscope: a new world scientific system*. New York: Harper & Row, 1979. 905 p.

9. EMBREE, Paul. *Algorithms* for *real-time DSP*. Brasil: Prentice-Hall, 1995. 356 p.

10. *Eviroment systems*. [en línea]. <http://en.wikipedia.org/wiki/environment_(systems)>. [Consulta: octubre de 2015].

11. FYODOROVICH, Valentin. *Turchin, the phenomenon of science: a cybernetic approach to human evolution*. New York: Columbia University, 1977. 348 p.

12. HEYLIGHEN, Francis. *Representation and change. A metarepresentational framework* for *the foundations of physical and cognitive science communication & cognition*. Belgium: Ghent, 1990. 200 p.

13. HEYLIGHEN, Francis; VRANCKX, An. *Principia cybernetica web, web dictionary of cybernetics and systems*. [en línea]. <http://pespmc1.vub.ac.be/asc /indexasc.html>. [Consulta: octubre de 2015].

14. *Invariancia galileana*. [en línea]. <https://es.wikipedia.org/wiki/Invariancia_galileana>. [Consulta: octubre de 2015].

15. InvenSense Inc. *MPU-9150 product specification*. [en línea]. <https://store.invensense.com/Datasheets/invensense/RM-MPU-91 50A-00-v3.0.pdf>. [Consulta: octubre de 2015].

16. InvenSense Inc. *MPU-9150 register map and description*. [en línea]. <https://strawberry-linux.com/pub/PS-MPU-9150A.pdf>. [Consulta: octubre de 2015].

17. KUO, Benjamin. *automatic control systems*. 7a. ed. New York: Prentice-Hall, 1994. 928 p.

18. LAGRANGE, Joseph-Louis. *Mécanique analytique, L'Institut des Sciences, Lettres et Arts, du Bureau des Longitudes*. Paris, France : Grand-Officier de la Légion-d'Honneur, 1815. 236 p.

19. LEE, Edward; SESHIA, Sanjit. *Introduction to embedded systems - a cyber-physical systems approach*. China: MáquinaPress, 2010. 978 p.

20. MALHAM, Simon. J. *An introduction to lagrangian and hamiltonian mechanics*. Edinburgh: Lecture Notes Maxwell Institute for Mathematical Sciences and School of Mathematical and Computer Sciences, 2015. 298 p.

21. *Mecánica newtoniana*. [en línea]. <http://es.wikipedia.org/wiki/ Mecánica_newtoniana>. [Consulta: octubre de 2015].

22. MORIN, David. *Introduction to classical mechanics: with problems and solutions*. New York: Cambridge University Press, 2008. 734 p.

23. MÜLLER, Gerhard. *Dissipative* for*ces in lagrangian mechanics, in classical dynamics*. Island: Department of Physics, University of Rhode Island, 2014. 128 p.

24. PIDWIRNY, Micahel. *Definitions of systems and models, in fundamentals of physical geography*. 2a. ed. Canadá: University of British Columbia, 2006. 512 p.

25. PSALTIS, Dimitrios. *Lagrangian dynamics, in theoretical mechanics II*. USA: University of Arizona, 2010. 910 p.

26. RESNICK, Robert; HALLIDAY, David. *Physics, part 1*. New York: JohnWiley & Sons Inc, 1977. 348 p.

27. SHANNON, C. E. *Communication in presence ofnoise, in reprint as classical paper*. [en línea]. <http://www.gmee.deit.univpm.it/biblioteca/sala_tecnica/scaffale_teoria/campionamento/intro_shannon.pdf>. [Consulta: octubre de 2015].

28. *System*. [en línea]. <http://en.wikipedia.org/wiki/system>. [Consulta: octubre de 2015].

29. TURCHIN, V.; HEYLIGHEN, F.; JOSLYN, C. y BOLLEN, J. *Control, principia cybernetica web.* New York: Columbia University, 1996. 411 p.

APÉNDICES

Apéndice 1. Braquistócrona, ecuación de euler y otras demostraciones del cálculo de variaciones

- Teorema B.1: ecuación de Euler-Lagrange

Sea $j\colon \mathbb{R}^{\mathrm{R}} \to \mathbb{R}$ un funcional de $u \in \mathbb{R}^{\mathrm{R}}$, de la forma

$$j(\mathrm{u}) = \int_{t_1}^{t_2} f(t, u, \ddot{\mathrm{u}})dt.$$

[Ec.A1-1]

Donde *F* es una función al menos dos veces derivable respecto todos sus parámetros, sobre todos los posibles caminos $u = u(t)$, los cuáles son al menos dos veces diferenciables sobre el intervalo $[t_1, t_2]$, con $y(a)$ y $y(b)$ fijos.

La función $q = q(t)$ que vuelve extremo el funcional J necesariamente satisface la ecuación de Euler- Lagrange sobre el intervalo $[t_1, t_2]$,,

$$\frac{\partial F}{\partial q} - \frac{\mathrm{d}}{dt}\left(\frac{\partial \mathrm{F}}{\partial q}\right)$$

[Ec. A1-2]

Demostración. Considérese la familia de funciones sobre el intervalo $[t_1, t_2]$,,

$$u(t,c) = q(t) + \epsilon\eta(t)$$

[Ec. A1-3]

Donde las funciones $\eta = \eta(t)$, son almenos dos veces diferenciables y satisfacen $\eta(a) = \eta(b) = 0$. El valor de ϵ, es un número real y la función $q = q(t)$ optimiza J . Sea entonces *Á* una función de ∅ tal que:

$$\emptyset/\epsilon) = J(q + \epsilon\eta)$$

[Ec. A1-4]

Si el funcional J tiene un máximo o mínimo local en q, entonces q es una estacionaria para J , y para todo η se tiene:

$$\emptyset'(0) = 0$$ [Ec. A1-5]

Evaluando la condición anterior directamente con la expresión equivalente en términos del funcional:

$$\emptyset'(\epsilon) = \frac{\mathrm{d}}{d\epsilon}\int_{t_1}^{t_2} F(t, q + \epsilon\eta, \ddot{u} + \in \eta)dt$$

[Ec. A1-6]

$$= \int_{t_1}^{t_2} \frac{\partial}{\partial\epsilon} F(t, q + \epsilon\eta, \ddot{u} + \in \eta)dt$$

[Ec. A1-7]

$$= \int_{t_1}^{t_2} \frac{\partial}{\partial\epsilon} F(t, q + \epsilon\eta, \ddot{u} + \in \eta)dt$$

[Ec. A1-8]

$$= \int_{t_1}^{t_2} \frac{\partial}{\partial\epsilon} F(t, u(t, \in), u(t, \in))dt$$

[Ec. A1-9]

$$= \int_{t_1}^{t_2} \left(\frac{\partial \mathrm{F}}{\partial u} \frac{\partial \mathrm{u}}{\partial \epsilon} + \frac{\partial \mathrm{F}}{\partial u} \frac{\partial \mathrm{u}}{\partial \epsilon} \right) dt$$

[Ec. A1-10]

$$= \int_{t_1}^{t_2} \left(\frac{\partial \mathrm{F}}{\partial u} \eta(t) + \frac{\partial \mathrm{F}}{\partial u} \frac{\partial \mathrm{u}}{\partial \epsilon} \eta(t) \right) dt$$

[Ec. A1-11]

$$= \int_{t_1}^{t_2} \frac{\partial \mathrm{F}}{\partial u} \eta(t) dt + \int_{t^1}^{t^2} \frac{\partial \mathrm{F}}{\partial u} \eta(t) dt$$

[Ec. A1-12]

Integrando el segundo término de la suma:

$$\int_{t^1}^{t_2} \frac{\partial \mathrm{F}}{\partial u} \eta(t) dt = \left[\frac{\partial \mathrm{F}}{\partial u} \eta(t) \right]_{t_1}^{t_2} - \int_{t^1}^{t^2} \left(\frac{\partial \mathrm{F}}{\partial u} \right) \frac{\partial \mathrm{F}}{\partial u} \eta(t) dt$$

[Ec. A1-13]

Del hecho de $\eta(a) = \eta(b) = 0$, construido de esa forma para que toda $u\,(t\,, \in)$, de tal forma que en todo $\in$ se tenga $u\,(a, \in) = u(a)\; y\; u\,(b, \in) \in u(b)$,

$$\emptyset'(\in) = \int_{t^1}^{t_2} \left(\frac{\partial \mathrm{F}}{\partial u} - \frac{\mathrm{d}}{dt} \left(\frac{\partial \mathrm{F}}{\partial u} \right) \right) \eta(t) dt$$

[Ec. A1-14]

Si se hace $\in = 0$, la condición para que q sea la estacionaria de J se tiene:

$$\int_{t^1}^{t_2} \left(\frac{\partial \mathrm{F}}{\partial q} - \frac{\mathrm{d}}{dt} \left(\frac{\partial \mathrm{F}}{\partial q} \right) \right) \eta(t) dt = 0.$$

[Ec. A1-15]

Dado que las funciones $\eta = \eta(t)$ son de elección arbitraria, se tiene que para todo $t \in [t_1, t_2], q = q(t)$ debe satisfacer necesariamente la ecuación de Euler-Lagrange.

- Teorema B.2: diferente funcional, una misma estacionaria

Considérese la curva $\boldsymbol{q} = \boldsymbol{q}(t)$, solución estacionaria para un funcional de la forma $S = \int_{t_1}^{t_2} L\,(q, q, t)dt$. Existe un funcional $S = \int_{t_1}^{t_2} L\,(q, q, t)dt$, tal que $L \neq L\ y q = q(t)$ una solución estacionaria para S.

Demostración. Considérese dos funcionales $L\ y\ \bar{L}$, que dependen del tiempo, un vector de variables dependientes $\boldsymbol{q} = \{q_1, q_2, \ldots, qn\}T$ en el tiempo y el vector de razones de cambio $\boldsymbol{q} = \{q_1,\ q_2, \ldots,\ qn\}T$ en relación al tiempo. L y $\bar{L}$ son diferentes entre sí

$$L\,(\mathrm{q}, \mathrm{q}, \mathrm{t}) = L(q, q, t) + \frac{d}{dt}\big(f(q, t)\big)$$

[Ec. A1-16]

Si se aplican las ecuaciones de Euler-Lagrange a $\bar{L}$, se obtiene:

$$0 = \frac{\mathrm{d}}{dt}\left(\frac{\partial \mathrm{L}}{\partial q;}\right)\frac{\partial \mathrm{L}}{\partial q;} \qquad \forall j \in \{I, \ldots, n\}$$

[Ec. A1-17]

$$= \frac{\mathrm{d}}{dt}\left(\frac{\partial(\mathrm{L} + \frac{df}{dt})}{\partial qj}\right) - \frac{\partial(\mathrm{L} + \frac{df}{dt})}{\partial qj} \qquad \forall j \in \{I, \ldots, n\}$$

[Ec. A1-18]

$$= \frac{\mathrm{d}}{dt}\left(\frac{\partial \mathrm{L}}{\partial q;}\right) - \frac{\partial \mathrm{L}}{\partial q;} + \frac{\mathrm{d}}{dt}\left(\frac{\partial(\frac{df}{dt})}{\partial qj}\right) - \frac{\partial(\frac{df}{dt})}{\partial qj} \qquad \forall j \in \{I, \ldots, n\}$$

[Ec. A1-19]

Hasta este punto no es fácil de ver que las ecuaciones de Euler-Lagrange para $\bar{L}$ y L son iguales, pero:

$$\frac{\mathrm{d}}{dt}\left(\frac{\partial(\frac{df}{dt})}{\partial qj}\right)-\frac{\partial(\frac{df}{dt})}{\partial qj}-\frac{\mathrm{d}}{dt}\left(\frac{\partial\left(\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{df}{\partial q_{\mathrm{i}}}q_{\mathrm{i}}\right)+\frac{df}{\partial t}\right)}{\partial qj}\right)-\frac{\partial\left(\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{df}{\partial q_{\mathrm{i}}}q_{\mathrm{i}}\right)+\frac{df}{\partial t}\right)}{\partial qj}$$

$$\forall j \in \{I,\dots,n\}$$

[Ec. A1-20]

$$=\frac{\mathrm{d}}{dt}\left(\frac{\partial f}{\partial qj}\right)-\left(\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{\partial\left(\frac{df}{\partial q_{\mathrm{i}}}q_{\mathrm{i}}\right)}{\partial qj}\right)+\frac{\partial^{2}f}{\partial qj\partial t}\right)$$

[Ec. A1-21]

$$=\left(\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{\partial^{2}f}{\partial q_{\mathrm{i}}qj}\right)+\frac{\partial^{2}f}{\partial qj\partial t}\right)-\left(\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{\partial^{2}f}{\partial q_{\mathrm{i}}\partial qj}\right)q_{\mathrm{i}}+\frac{\partial f}{\partial q:}\frac{\partial d_{\mathrm{i}}}{\partial qj}\right)+\frac{\partial^{2}f}{\partial q_{\mathrm{i}}\partial t})$$

[Ec. A1-22]

$$=\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{\partial^{2}f}{\partial q_{\mathrm{i}}qj}qt\right)+\frac{\partial^{2}f}{\partial tqj}-\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{\partial^{2}f}{\partial qj\partial q_{\mathrm{i}}}q_{\mathrm{i}}+\frac{\partial f}{\partial q:}(0)\right)-\frac{\partial^{2}f}{\partial q_{\mathrm{i}}\partial t}$$

[Ec. A1-23]

$$=\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{\partial^{2}f}{\partial q_{\mathrm{i}}qj}qt\right)+\frac{\partial^{2}f}{\partial tqj}-\sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\frac{\partial^{2}f}{\partial qj\partial q_{\mathrm{i}}}q_{\mathrm{i}}\right)-\frac{\partial^{2}f}{\partial q_{\mathrm{i}}\partial t}=0$$

$$\forall j \in \{I,\dots,n\}$$

[Ec. A1-24]

De esa cuenta,

$$= \frac{\mathrm{d}}{dt}\left(\frac{\partial L}{\partial qj}\right) - \frac{\partial L}{\partial qj} = \frac{\mathrm{d}}{dt}\left(\frac{\partial\left(L+\frac{df}{dt}\right)}{\partial qj}\right) - \frac{\partial\left(L+\frac{df}{dt}\right)}{\partial qj} \qquad \forall j \in \{I, \dots, n\}$$

[Ec. A1-25]

$$= \frac{\mathrm{d}}{dt}\left(\frac{\partial L}{\partial qj}\right) - \frac{\partial L}{\partial qj} + \frac{\mathrm{d}}{dt}\left(\frac{\partial\left(\frac{df}{dt}\right)}{\partial qj}\right) - \frac{\partial\left(\frac{df}{dt}\right)}{\partial qj} \qquad \forall j \in \{I, \dots, n\}$$

[Ec. A1-26]

$$= \frac{\mathrm{d}}{dt}\left(\frac{\partial L}{\partial qj}\right) - \frac{\partial L}{\partial qj} \qquad \forall j \in \{I, \dots, n\}$$

[Ec. A1-27]

Por tanto una curva estacionaria $\boldsymbol{q} = \boldsymbol{q}(t)$ para L será una curva estacionaria para L.

- Teorema B.3: no unicidad del lagrangiano

Sea la acción $S\{\boldsymbol{q}\} = \int_{t_1}^{t_2} L\{\boldsymbol{q}, \boldsymbol{q}, t\}\, dt$. Y sea $\boldsymbol{u} : t \to \mathbb{R}n$ un vector de variables dependientes en el tiempo. Existemás de un Lagrangiano L para el valor $S(\boldsymbol{u})$.

Demostración. Considérese dos Lagrangianos diferentes $L\, y\, L$, dependientes del tiempo, $\boldsymbol{q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{qn}\}^{\mathrm{T}}$ y $\boldsymbol{q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{qn}\}^{\mathrm{T}}$. Ya que la única condición para L y L es que sean diferentes, se puede escoger a L como:

$$L(q, qt) = L\,(q, q, t) + \frac{dg\,(q, q, t)}{dt}$$

[Ec. A1-28]

Donde g es una función cualquiera, dependiente del tiempo, $\boldsymbol{q}\ y\ \dot{\boldsymbol{q}}$ (ambos dependientes del tiempo), con la propiedad que $g\ (t_1)\ = g\ (t_2)$. Entonces sea

$$S(u) = \int_{t_1}^{t_2} \quad L(u,u,t)dt$$

[Ec. A1-29]

La acción para el Lagrangiano L . Por tanto,

$$S(u) = \int_{t_2}^{t_2} \quad L(u,u,t)\ + \frac{dg(u,u,t)}{dt}\ dt$$

[Ec. A1-30]

$$S(u) = \int_{t_1}^{t_2} \quad L(u,u,t)dt\ + \int_{t_1}^{T_2} \quad \frac{dg(u,u,t)}{dt}\ dt$$

[Ec. A1-31]

$$S(u) = S(u) + g\ (t_2)\ - g(t_1)$$

[Ec. A1-32]

$$S(u) = S(u)$$

[Ec. A1-33]

Pudiendo así obtener el mismo *S*(**u**) con dos Lagrangianos diferentes.

Fuente: elaboración propia.

Apéndice 2. **Códigos**

Código C.1: Control por micropaso para tm4c123gh6pm usando TIVAWARE

```
1 # define PART_TM4C123GH6PM
2
3 # define Circ 16
4
5 # include <stdint .h>
6 # include <stdbool .h>
7 # include " inc / tm4c123gh6pm .h"
8 # include " inc / hw_memmap .h"
9 # include " inc / hw_types .h"
10 # include " inc / hw_gpio .h"
11 # include " driverlib / sysctl .h"
12 # include " driverlib / interrupt .h"
13 # include " driverlib / gpio .h"
14 # include " driverlib / timer .h"
15 # include " driverlib / pwm .h"
16 # include " driverlib / pin_map .h"
17
18 uint32_t ui32Period =1;
19 uint8_t stepN =0;
20 bool stepFlag = true ;
21
22 # define PWM_FREQUENCY 2500
23 # define Stepfreq 2
24 # define TivaClock 80000000
25 // Variables PWM ////////////////////
26
27 volatile uint32_t ui32Load ;
28 volatile uint32_t ui32PWMClock ;
29
```

```
30 // ////////////////// Corriente
////////////////////////////////////////////////////////////
31 # ifdef Quad
32 #if Quad ==64
33 # define Steps 64
34 int32_t Current [ Steps ][2]={{499 , 0}, {499 , 48} , {499 , 97} , {499 , 144} , {499 , 190}
      ,{499 , 235} ,
35 {499 , 277} , {499 , 316} , {352 , 499} , {316 , 499} , {277 , 499} , {235 , 499} ,
36 {190 , 499} , {144 , 499} , {97 , 499} , {48 , 499} , {0, 499} , { -49 , 499} ,
37 { -98 , 499} , { -145 , 499} , { -191 , 499} , { -236 , 499} , { -278 , 499} ,
38 { -317 , 499} , { -499 , 352} , { -499 , 316} , { -499 , 277} , { -499 , 235} ,
39 { -499 , 190} , { -499 , 144} , { -499 , 97} , { -499 , 48} , { -499 , 0}, { -499 , -49} ,
40 { -499 , -98}, { -499 , -145} , { -499 , -191} , { -499 , -236} , { -499 , -278}
41 { -499 , -317} , { -353 , -499} , { -317 , -499} , { -278 , -499} , { -236 , -499} ,
42 { -191 , -499} , { -145 , -499} , { -98 , -499} , { -49 , -499} , {0, -499} ,
43 {48 , -499} , {97 , -499} , {144 , -499} , {190 , -499} , {235 , -499} ,
44 {277 , -499} , {316 , -499} , {352 , -353} , {499 , -317} , {499 , -278} ,
45 {499 , -236} , {499 , -191} , {499 , -145} , {499 , -98}, {499 , -49}};
46 # elif Quad ==32
257
258 C. CÓDIGOS.
47 # define Steps 32
48 int32_t Current [ Steps ][2]={{499 , 0}, {499 , 97} , {499 , 190} , {499 , 277} , {352 , 499}
      , {277 , 499} ,
49 {190 , 499} , {97 , 499} , {0, 499} , {-98 , 499} , { -191 ,499} , { -278 , 499} ,
50 { -499 , 352} , { -499 , 277} , { -499 , 190} , { -499 ,97} , { -499 ,0} , { -499 , -98} ,
51 { -499 , -191} , { -499 , -278} , { -353 , -499} , { -278 , -499} , { -191 , -499} ,
52 { -98, -499} , {0, -499} , {97 , -499} , {190 , -499} , {277 , -499} , {352 , -353} ,
53 {499 , -278} , {499 , -191} , {499 , -98}};
54 # elif Quad ==16
55 # define Steps 16
56 int32_t Current [ Steps ][2]={{499 , 0}, {499 , 190} , {352 , 499} , {190 , 499} , {0, 499} , {
      -191 , 499} ,
57 { -499 , 352} , { -499 , 190} , { -499 , 0}, { -499 , -191} , { -353 , -499} , { -191 , -499} ,
58 {0, -499} , {190 , -499} , {352 , -353} , {499 , -191}};
```

```
59 # elif Quad ==8
60 # define Steps 8
61 int32_t Current [ Steps ][2]={{499 , 0}, {352 , 499} , {0, 499} , { -499 , 352} , { -499 , 0},
{ -353 , -499} , {0, -499} , {352 , -353}};
62 # else
63 # define Steps 4
64 int32_t Current [ Steps ][2]={{499 , 499} , { -499 , 499} ,{ -499 , -499} ,{499 , -499}};
65 # endif
66 # elif Circ
67 #if Circ ==64
68 # define Steps 64
69 int32_t Current [ Steps ][2]={{499 , 0}, {497 , 49} , {490 , 98} , {478 , 145} , {462 , 191} ,
      {441 , 236} , {415 , 278} ,
70 {386 , 317} , {353 , 353} , {317 , 386} , {278 , 415} , {236 , 441} , {191 , 462} , {145 ,
      478} ,
71 {98 , 490} , {49 , 497} , {0, 499} , { -48 , 497} , { -97 , 490} , { -144 , 478} , { -190 , 462} ,
72 { -235 , 441} , { -277 , 415} , { -316 , 386} , { -352 , 353} , { -385 , 317} , { -414 , 278} ,
73 { -440 , 236} , { -461 , 191} , { -477 , 145} , { -489 , 98} , { -496 , 49} , { -499 , 0},
74 { -496 , -48} , { -489 , -97} , { -477 , -144} , { -461 , -190} , { -440 , -235} , { -414 , -277} ,
75 { -385 , -316} , { -352 , -352} , { -316 , -385} , { -277 , -414} , { -235 , -440} , { -190 , -
      461} ,
76 { -144 , -477} , { -97 , -489} , { -48 , -496} , {0, -499} , {49 , -496} , {98 , -489} , {145 , -
      477} ,
77 {191 , -461} , {236 , -440} , {278 , -414} , {317 , -385} , {353 , -352} , {386 , -316} ,
78 {415 , -277} , {441 , -235} , {462 , -190} , {478 , -144} , {490 , -97} , {497 , -48}};
79 # elif Circ ==32
80 # define Steps 32
81 int32_t Current [ Steps ][2]={{499 , 0}, {490 , 98} , {462 , 191} , {415 , 278} , {353 , 353}
      , {278 , 415} , {191 , 462} , {98 , 490} ,
82 {0, 499} , { -97 , 490} , { -190 , 462} , { -277 , 415} , { -352 , 353} , { -414 , 278} , { -461 ,
      191} ,
83 { -489 , 98} , { -499 , 0}, { -489 , -97} , { -461 , -190} , { -414 , -277} , { -352 , -352} , { -
      277 , -414} ,
84 { -190 , -461} , { -97 , -489} , {0, -499} , {98 , -489} , {191 , -461} , {278 , -414} , {353 , -
      352} ,
```

```
85 {415 , -277} , {462 , -190} , {490 , -97}};
86 # elif Circ ==16
87 # define Steps 16
88 int32_t Current [ Steps ][2]={{499 , 0}, {462 , 191} , {353 , 353} , {191 , 462} , {0, 499} , {
     -190 , 462} , { -352 , 353} , { -461 , 191} ,
89 { -499 , 0}, { -461 , -190} , { -352 , -352} , { -190 , -461} , {0, -499} , {191 , -461} , {353 ,-
     352} ,{462 , -190}};
90 # elif Circ ==8
91 # define Steps 8
92 int32_t Current [ Steps ][2]={{499 , 0}, {353 , 353} , {0, 499} , { -352 , 353} , { -499 , 0},{
     -352 , -352} , {0, -499} , {353 , -352}};
93 # else
94 # define Steps 4
95 int32_t Current [ Steps ][2]={{499 , 0}, {0, 499} ,{ -499 , 0} ,{0 , -499}};
96 # endif
97 # endif
98
99 int main ( void ) {
100 // Reloj principal a 80 Mhz
101 SysCtlClockSet ( SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL |
     SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN );
102 // Prescaler de PWM
103 SysCtlPWMClockSet ( SYSCTL_PWMDIV_64 );
104
105 // Reloj a Perif é rico
106 SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOF );
107 SysCtlPeripheralEnable ( SYSCTL_PERIPH_GPIOD );
259
108 // Reloj a Timer
109 SysCtlPeripheralEnable ( SYSCTL_PERIPH_TIMER0 );
110 // Reloj a PWM
111 SysCtlPeripheralEnable ( SYSCTL_PERIPH_PWM1 );
112
113 // Configuraci ón de pines
```

```
114 GPIOPinTypeGPIOOutput ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 |
      GPIO_PIN_3 | GPIO_PIN_4 );
115 GPIOPinTypePWM ( GPIO_PORTD_BASE , GPIO_PIN_0 | GPIO_PIN_1 );
116 GPIOPinConfigure ( GPIO_PD0_M1PWM0 );
117 GPIOPinConfigure ( GPIO_PD1_M1PWM1 );
118
119 // Configuraci ón de Timer Peri ó dico
120 TimerConfigure ( TIMER0_BASE , TIMER_CFG_PERIODIC );
121 // Configuraci ón periodo de Timer
122 ui32Period = ( TivaClock / Stepfreq );
123 TimerLoadSet ( TIMER0_BASE , TIMER_A , ui32Period -1);
124
125 ui32PWMClock = TivaClock / 64;
126 ui32Load = ( ui32PWMClock / PWM_FREQUENCY ) - 1;
127 // Configuraci ón inicial Generador 1 Modo Down y un periodo dado por PWMLoad
128 PWMGenConfigure ( PWM1_BASE , PWM_GEN_0 , PWM_GEN_MODE_DOWN );
129 PWMGenPeriodSet ( PWM1_BASE , PWM_GEN_0 , ui32Load );
130
131 // Ancho de pulso dado por output
132 PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_0 , ( ui32Load +1) /2 -1); // PD0
133 PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_1 , ( ui32Load +1) /2 -1); // PD1
134 // Habilita salida D0 y D1
135 PWMOutputState ( PWM1_BASE , PWM_OUT_0_BIT , true );// PD0
136 PWMOutputState ( PWM1_BASE , PWM_OUT_1_BIT , true );// PD1
137 // Habilita Generador 0
138 PWMGenEnable ( PWM1_BASE , PWM_GEN_0 );
139
140
141 IntEnable ( INT_TIMER0A );
142 TimerIntEnable ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
143 IntMasterEnable ();
144
145 TimerEnable ( TIMER0_BASE , TIMER_A );
146
147 for (;;) {
```

```
148
149 if( stepFlag ){
150 // Corriente en D0
151 if( Current [ stepN ][0]==0) {
152 Current [ stepN ][0]=1;
153 }
154 if( Current [ stepN ][0] >0) {
155 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 , 2);
156 PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_0 , Current [ stepN ][0]) ;
157 }
158 else
159 {
160 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_1 | GPIO_PIN_2 , 4);
161 PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_0 , -Current [ stepN ][0]) ;
162
163 }
164 // Corriente en D1
165 if( Current [ stepN ][1]==0) {
166 Current [ stepN ][1]=1;
167 }
168 if( Current [ stepN ][1] >0) {
169 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_3 | GPIO_PIN_4 , 8);
170 PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_1 , Current [ stepN ][1]) ;
171 }
172 else
173 {
174 GPIOPinWrite ( GPIO_PORTF_BASE , GPIO_PIN_3 | GPIO_PIN_4 , 16) ;
175 PWMPulseWidthSet ( PWM1_BASE , PWM_OUT_1 , -Current [ stepN ][1]) ;
176 }
177 stepN ++;
178 if(stepN >= Steps )
260 C. CÓDIGOS.
179 { stepN =0;}
180 stepFlag = false ;
181 }
```

```
182 }
183 return 0;
184 }
185
186 void Timer0AIntHandler ( void ){
187 TimerIntClear ( TIMER0_BASE , TIMER_TIMA_TIMEOUT );
188 stepFlag = true ;
```

Fuente: elaboración propia.