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

DISEÑO ELECTRÓNICO DE CONTROL PARA ROBOTS USANDO TECNOLOGÍA OPEN SOURCE, PLATAFORMA ARDUINO

Gabriel Fernando Montenegro Ortiz
Asesorado por el Ing. Enrique Edmundo Ruiz Carballo

Guatemala, julio de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO ELECTRÓNICO DE CONTROL PARA ROBOTS USANDO TECNOLOGÍA OPEN SOURCE, PLATAFORMA ARDUINO

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

GABRIEL FERNANDO MONTENEGRO ORTÍZ
ASESORADO POR EL ING. ENRIQUE EDMUNDO RUIZ CARBALLO

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, JULIO 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. José Aníbal Silva de los Ángeles
EXAMINADOR	Ing. Byron Odilio Arrivillaga Méndez
EXAMINADOR	Ing. Marvin Marino Hernández Fernández
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

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:

DISEÑO ELECTRÓNICO DE CONTROL PARA ROBOTS USANDO TECNOLOGÍA OPEN SOURCE, PLATAFORMA ARDUINO

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 23 de febrero de 2012.

Gabriel Fernando Montenegro Ortíz

Guatemala, 25 de Enero de 2016

Ing, Carlos Eduardo Guzmán Salarzar
Coordinador de Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Apreciable Ing. Guzmán

El motivo de la presente es para informarle que he revisado el Trabajado de Graduación titulado: **"DISEÑO ELECTRÓNICO DE CONTROL PARA ROBOTS USANDO TECNOLOGÍA OPEN SOURCE, PLATAFORMA ARDUINO"**, desarrollado por el estudiante Gabriel Fernando Montenegro Ortíz con carné No. 2004-12829 de la carrera de Ingeniería Electrónica, en calidad de asesor, me permito dar como aprobado y a la vez indicar que el autor y mi persona somos responsables del contenido de dicho trabajo.

Sin otro particular, me suscribo de usted

Atentamente,

Enrique Edmundo Ruiz Carballo
Asesor
Colegiado 2225

Enrique E. Ruiz Carballo
Ingeniero Electricista
Col. No. 2225

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 21. 2016.
Guatemala, 4 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: DISEÑO ELECTRÓNICO DE CONTROL PARA ROBOTS USANDO TECNOLOGÍA OPEN SOURCE, PLATAFORMA ARDUINO, del estudiante Gabriel Fernando Montenegro Ortíz, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

Ing. Carlos Eduardo Guzmán Salazar
Coordinador Área Electrónica

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

sro

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 21. 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; GABRIEL FERNANDO MONTENEGRO ORTÍZ Titulado: DISEÑO ELECTRÓNICO DE CONTROL PARA ROBOTS USANDO TECNOLOGÍA OPEN SOURCE, PLATAFORMA ARDUINO , 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, 14 DE ABRIL 2016.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 344.2016

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: **DISEÑO ELECTRÓNICO DE CONTROL PARA ROBOTS USANDO TECNOLOGÍA OPEN SOURCE, PLATAFORMA ARDUINO,** presentado por el estudiante universitario: **Gabriel Fernando Montenegro Ortiz,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, julio de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

/gdech

ACTO QUE DEDICO A:

Dios	Por haberme dado la fortaleza y el entendimiento para lograr todas las metas que me he propuesto.
Mis padres	José Gabriel Montenegro Paiz y Marta Lidia Ortiz Sánchez, por todo el apoyo, cariño y comprensión a lo largo de mi vida.
Mi hermana	Silvia Montenegro Ortiz, por haber compartido conmigo tantas experiencias y siempre estar a mi lado.
Mi abuelo	José Adrián Montenegro (q. e. p. d.), por haber sido mi modelo a seguir desde pequeño y enseñarme todos esos valores que han forjado mi forma de ser.
Mis abuelas	María Luisa Sánchez (q. e. p. d.) y Andrea Paiz, por haberme apoyado y motivado a ser una mejor persona.
Mis primos	Karla y Ana Lucía Arroyo, Luis, Claudia y Sandra Montenegro y Jackeline Paredes, por todo su cariño y apoyo en todos los momentos que hemos compartido.

Mis tíos Alfredo Montenegro y Amanda Montenegro, por brindarme su apoyo y consejos a lo largo de mi vida que me han ayudado a tomar mejores decisiones.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por permitirme ser parte de la cultura universitaria y fomentar los valores para ser una persona de éxito y autosuficiente.
Facultad de Ingeniería	Por brindarme las facilidades para realizar mis estudios de Ingeniería en un ambiente agradable.
Escuela de Mecánica Eléctrica	Por darme las herramientas necesarias para adquirir los conocimientos que me han permitido desarrollarme como profesional.
Mi asesor	Dr. Enrique Ruiz, por haberme guiado en la elaboración de este trabajo de graduación y compartir conmigo el amplio conocimiento que posee.
Mi novia	Silvia Gómez, por todo el cariño, comprensión y apoyo durante el tiempo de conocernos y ser uno de los pilares más importantes en mi vida.
Mis amigos	Por todas esas enseñanzas y apoyo durante todos estos años.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
A	Amperios
bps	Baudios por segundo
gdl	Grado de libertad
Hz	Hertz
KB	Kilo Bytes
MHz	Mega Hertz
mA	Miliamperios
mm	Milímetros
VDC	Voltaje en corriente continua
V	Voltios

GLOSARIO

ADC Analog to Digital Conversion (conversión analógica a digital). Consiste en la transcripción de señales analógicas en señales digitales.

Arduino Plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo.

ASCII American Standard Code for Information Interchange (Código estándar estadounidense para el intercambio de información). Utiliza 7 bits para representar los caracteres.

BSD Berkeley Software Distribution (distribución de software Berkeley) es un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

Byte Conjunto de información formado por 8 bits.

EEPROM Electrically Erasable Programmable Read-Only Memory (ROM programable y borrable eléctricamente). Es un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente.

FOSS	*Free and open source* software (software libre y de código de abierto) es el que está licenciado de tal manera que los usuarios pueden estudiar, modificar y mejorar su diseño mediante la disponibilidad de su código fuente.
GND	Tierra. Es la referencia del voltaje (0 voltios) utilizada en la energización de componentes electrónicos.
GNU GPL	Licencia que garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software.
Hardware	Conjunto de componentes físicos de un sistema.
LANC	Logic Application Control Bus System or Local Application Control Bus System, también conocido como Control-L es un protocolo de Sony utilizado para sincronizar cámaras de video.
Led	Light Emisor Diode (diodo emisor de luz) es un componente optoelectrónico pasivo –diodo– que emite luz.
Open source	Código abierto. Software distribuido y desarrollado libremente.

PWM	Pulse Width Modulation (modulación por ancho de pulsos). Es una técnica donde se modifica el ciclo de trabajo de una señal periódica.
Software	Conjunto de componentes lógicos de un sistema.
Software libre	Por elección manifiesta de su autor, puede ser copiado, estudiado, modificado, utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras
SPI	Serial Peripheral Interface. Estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos
TTL	Transistor-transistor logic (lógica transistor a transistor). Es una tecnología de construcción de circuitos electrónicos digitales en donde los niveles lógicos están definidos como 5v 1 lógico y 0v 0 lógico.
UART	Universal Asynchronous Receiver-Transmitter (transmisor-receptor asíncrono universal), es el dispositivo que controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo.

RESUMEN

Arduino es una plataforma de hardware libre que se ha vuelto muy popular en la década de 2000, y en este trabajo de investigación se muestra el diseño de un programa para el control de un robot. Se empieza describiendo el concepto de software libre, sus características, ventajas y aplicaciones relacionadas con Arduino.

Existe una amplia variedad de modelos de circuitería (placas) de Arduino, las cuales se explican brevemente, para posteriormente centrarse en el modelo Arduino UNO, la cual tiene mucha versatilidad y es una de las más utilizadas para realizar este tipo de proyectos. Adicionalmente, se hace una introducción a las conexiones físicas y configuración básica para conectarse a un sistema de control, y la conexión serial hacia una computadora para poder interactuar con el usuario y expandir las funcionalidades desde el entorno de desarrollo.

Asimismo, se detalla el funcionamiento básico de un robot, qué características debe tener según la aplicación, restricciones físicas y una breve descripción de sus componentes, entre ellos: motores, sensores y partes mecánicas, así como la interconexión con dichos componentes.

Por último, se mencionan las instrucciones básicas para la programación en el entorno de desarrollo, una descripción de los componentes (motores, módulos de RF, sensores), propuestos para el ejemplo del diseño de control de un robot y el código del programa con comentarios sobre la estructura de este, que deben tomar en cuenta para dicho diseño de control, considerando la

flexibilidad que se tiene para utilizar este entorno de desarrollo en aplicaciones más complejas que exceden los objetivos de este trabajo de investigación.

OBJETIVOS

General

Realizar el código de un programa para el sistema de control de un robot utilizando la plataforma Arduino, que permita mostrar la versatilidad, escalabilidad y ventajas del código abierto y de las plataformas de hardware libre.

Específicos

1. Presentar una introducción al software y hardware libre, mostrando las diferencias entre software gratis y software libre. Las características del código abierto y los requisitos que un código debe cumplir para ser considerado abierto.

2. Mencionar las características básicas de una placa Arduino, sus componentes y configuraciones posibles para conectarla con los dispositivos que se requieran, haciendo énfasis en la comunicación serial.

3. Presentar una breve descripción de un sistema robótico incluyendo sus componentes, restricciones y configuraciones para operarlo. Adicionalmente, mencionar la configuración básica de un circuito de puente H para el sistema de control de los motores necesarios en el sistema robótico.

4. Realizar un código de programación para la plataforma Arduino, donde se ejemplifique el manejo de un sistema robótico por medio de módulos electrónicos que interactúan con la placa Arduino.

INTRODUCCIÓN

Existen muchas variantes a la hora de querer utilizar un sistema para controlar robots u otro tipo de aplicaciones que requieran conexión con la computadora para el control de dicho sistema y conectarse con dispositivos como motores, para realizar trabajos mecánicos definidos previamente. Arduino (plataforma de hardware libre basada en una placa con un microcontrolador y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring apoyado en Visual Basic, Java, C, Flash) ha adquirido mucha popularidad a partir del año 2000 debido a la flexibilidad y variantes que ofrece tanto a nivel de hardware, como por la escalabilidad y el potencial del software con el que se programa.

Arduino se puede utilizar para desarrollar objetos interactivos autónomos, robots, transmisión de información o puede ser conectado a la computadora y mediante el software adecuado, realizar una gran variedad de funciones. Las placas se pueden fabricar a mano por el diseñador, quien tiene la capacidad de patentar su circuitería y software. El entorno de desarrollo integrado libre se puede descargar gratuitamente y funcionar en entornos como internet. Al ser open-hardware su diseño y su distribución son libres.

Este trabajo de graduación se enfoca en el diseño de control de un robot controlado inalámbricamente, con sensores de medición para controlar velocidad, para ello se utilizará la tecnología Arduino que permite el control en doble vía, retroalimentación, transmisión de audio y video, con el fin de mostrar una alternativa de un sistema de control simple utilizando una plataforma económica, para que las empresas y los centros de estudio puedan utilizarla en

Guatemala, aprovechando el bajo costo, la documentación disponible y la capacidad del desarrollador de patentar el hardware y software de control. Las aplicaciones que se realizan con Arduino llegan a ser muy complejas dependiendo de las acciones que se quieran hacer, lo cual excede el objetivo de este trabajo de graduación, pero hay mucha documentación, la cual junto con este trabajo, puede servir de guía para realizarlas.

1. TECNOLOGÍA ARDUINO, *OPEN SOURCE*

1.1. Conceptos básicos

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

Arduino puede detectar el medio ambiente recibiendo señales en sus entradas de una variedad de sensores y e influir sus alrededores controlando luces, motores y otros actuadores.

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8, por su sencillez y bajo costo que permiten el desarrollo de múltiples diseños. Por otro lado, el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque (*boot loader*) que corre en la placa.

Una de sus aplicaciones es el desarrollo de objetos interactivos autónomos. También puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente desde la página web de Arduino.

Los diseños de hardware de referencia están disponibles bajo una licencia de código libre, y cualquier persona está en la libertad de adaptarlo a sus necesidades. Open-hardware puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia. Está diseñado por artistas, diseñadores, aficionados y cualquier persona interesada en crear objetos o entornos interactivos.

El proyecto Arduino recibió una mención honorífica en la categoría de Comunidades Digital en el Prix Ars Electrónica de 2006. El equipo Arduino es: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, y David Mellis.

1.2. Historia de la tecnología Arduino y la tecnología *open source*

Código abierto (*open source*) es el término con el que se conoce al software distribuido y desarrollado libremente. Este tiene un punto de vista más orientado a los beneficios prácticos de compartir el código, que a las cuestiones éticas y morales las cuales destacan en el llamado software libre.

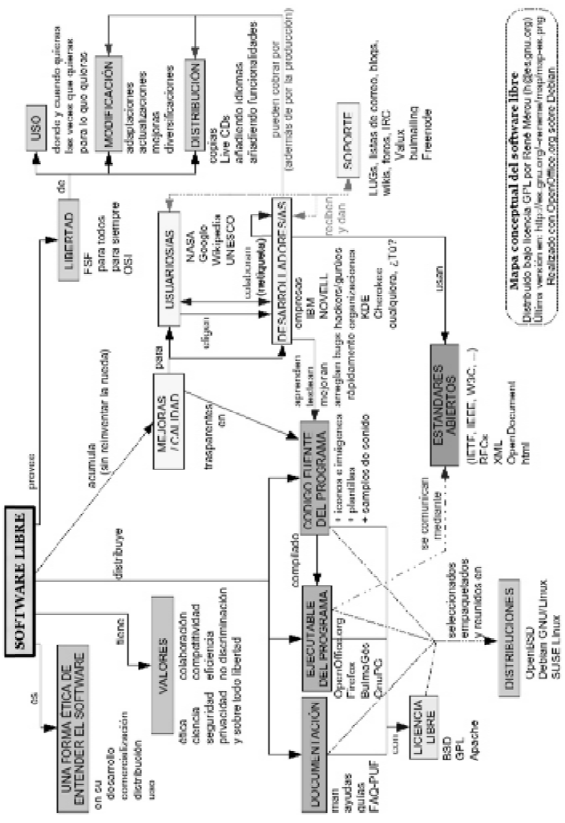
1.2.1. 1.2.1 Historia del software libre y de código abierto

La historia, actualmente, se remonta a inicios de los años 1980, época en la que la mayoría de software era privativo y surgió la necesidad, por parte de algunos programadores, de crear proyectos que impulsaran la creación de software libre. Cabe mencionar que antes, cuando las primeras computadoras nacieron (y por ende los primeros programas informáticos), el software tenía un modelo de desarrollo cooperativo, similar al de otras ciencias como la física; esto empezó a cambiar en los años 1960 y 1970, cuando nacieron las primeras compañías que privatizaron su código.

Es importante señalar que el software libre y de código abierto, no debe confundirse con el llamado *freeware*; el software libre y de código abierto suele ser gratuito, lo que puede llevar a confusión. El FOSS (acrónimo en inglés para *free and open source software*), también puede ser comprado y vendido. La confusión es aún mayor en países de habla inglesa por la ambigüedad de la palabra *free* que significa tanto libertad como gratuidad.

En la figura 1 se observa una línea de tiempo del software libre y código abierto desde su concepción.

Figura 1. **Mapa conceptual del software libre**



Fuente: GNU España. http://es.gnu.org/-reneme/map/map-es.png.

Consulta: 23 de mayo de 2014.

1.2.2. Libertades del software libre

De acuerdo con tal definición, el software es libre si garantiza las siguientes libertades:

- Libertad 0: ejecutar el programa con cualquier propósito (privado, educativo, público, comercial, entre otros).
- Libertad 1: estudiar y modificar el programa (para lo cual es necesario poder acceder al código fuente).
- Libertad 2: copiar el programa de manera que se pueda ayudar a cualquier persona que lo requiera.
- Libertad 3: mejorar el programa y hacer públicas las mejoras, de forma que se beneficie toda la comunidad.

Es importante señalar que las libertades 1 y 3 obligan a que se tenga acceso al código fuente. La libertad 2 hace referencia a la libertad de modificar y redistribuir el software libremente, licenciado bajo algún tipo de licencia de software libre que beneficie a la comunidad.

Ciertos teóricos usan este cuarto punto (libertad 3) para justificar parcialmente las limitaciones impuestas por la licencia GNU GPL frente a otras licencias de software libre; sin embargo, el sentido original es más libre, abierto y menos restrictivo que el que le otorga la propia GNU GPL.

La licencia GNU GPL posibilita la modificación, redistribución del software, pero únicamente bajo esa misma licencia. Y añade, que si se reutiliza en un mismo programa código A licenciado bajo licencia GNU GPL y código B licenciado bajo otro tipo de licencia libre, el código final C, independientemente de la cantidad y calidad de cada código A y B debe de estar bajo la licencia

GNU GPL. En la práctica esto hace que las licencias de software libre se dividan en dos grandes grupos, aquellas que pueden ser mezcladas con código licenciado bajo GNU GPL (y que inevitablemente desaparecerán en el proceso, al ser el código resultante licenciado bajo GNU GPL), y las que no lo permiten al incluir mayores u otros requisitos que no contempla ni admite la GNU GPL y que, por lo tanto, no pueden ser enlazadas ni mezcladas con código gobernado por la licencia GNU GPL.

Esta situación de incompatibilidad, que podría ser resuelta en la próxima versión 3.0 de la licencia GNU GPL (en desarrollo), causa en estos momentos graves perjuicios a la comunidad de programadores de software libre, que muchas veces no pueden reutilizar o mezclar códigos de dos licencias distintas, pese a que las libertades teóricamente lo deberían permitir. En el sitio web de la FSF (Free Software Foundation) hay una lista de licencias que cumplen las condiciones impuestas por la GNU GPL y otras que no, y en el sitio web de la OSI está la lista completa de las licencias de software libre actualmente aprobadas.

El término software no libre se emplea para referirse al software distribuido bajo una licencia de software más restrictiva que no garantiza estas cuatro libertades. Las leyes de la propiedad intelectual reservan la mayoría de los derechos de modificación, duplicación y redistribución para el dueño del *copyright*; el software dispuesto bajo una licencia de software libre rescinde específicamente la mayoría de estos derechos reservados.

La definición de software libre no contempla el asunto del precio; un eslogan frecuentemente usado es "libre como en libertad de expresión no como en cerveza gratis" (aludiendo a la ambigüedad del término inglés "*free*"), y es habitual ver a la venta CD de software libre como distribuciones GNU/Linux. Sin

embargo, en esta situación, el comprador del CD tiene el derecho de copiarlo y redistribuirlo.

El software gratis puede incluir restricciones que no se adaptan a la definición de software libre, por ejemplo, puede no incluir el código fuente, prohibir explícitamente a los distribuidores, recibir una compensación a cambio, entre otros. Para evitar la confusión, algunas personas utilizan los términos "libre" (Libre software) y "gratis" (Gratis software) para evitar la ambigüedad de la palabra inglesa "*free*". Sin embargo, estos términos alternativos son usados únicamente dentro del movimiento del software libre, aunque están extendiéndose poco a poco hacia el resto del mundo.

Otro grupo de personas prefieren el uso del término *open source software* (software de código abierto, también llamado de fuentes abiertas). La principal diferencia entre los términos "*open source*" y "*free software*" es que este último tiene en cuenta los aspectos éticos y filosóficos de la libertad, mientras que el primero se basa únicamente en los aspectos técnicos.

1.2.3. Tipos de licencias

Una licencia es aquella autorización formal con carácter contractual que un autor de un software da a un interesado para ejercer "actos de explotación legales". Puede haber tantas licencias como acuerdos concretos se den entre el autor y el licenciatario. Desde el punto de vista del software libre, existen distintas variantes del concepto o grupos de licencias:

- Las libertades definidas anteriormente están protegidas por licencias de software libre, de las cuales una de las más utilizadas es la Licencia Pública General GNU (GPL). El autor conserva los derechos de autor

(*copyright*), y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que no sea imposible crear un producto con partes no licenciadas GPL: el conjunto tiene que ser GPL.

- Licencias estilo BSD: se utilizan en software distribuido junto a los sistemas operativos BSD. El autor, bajo tales licencias, mantiene la protección de *copyright* únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario. Son muy permisivas, tanto que son fácilmente absorbidas al ser mezcladas con la licencia GNU GPL con quienes son compatibles.

Puede argumentarse que esta licencia asegura verdadero software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al software, y que puede decidir, incluso redistribuirlo como no libre.

Existen otras opiniones orientadas a destacar que este tipo de licencia no contribuye al desarrollo de más software libre, por ejemplo las licencias estilo MPL y derivadas. Estas licencias son de software libre y tienen un gran valor porque fueron el instrumento que empleó Netscape Communications Corp. para liberar Netscape Communicator 4.0 y empezar ese proyecto tan importante para el mundo del software libre: Mozilla. Se utilizan en gran cantidad de productos de software libre de uso cotidiano en todo tipo de sistemas operativos.

La MPL es software libre y promueve eficazmente la colaboración evitando el efecto viral de la GPL (si se usa código licenciado GPL, el desarrollo final

tiene que estar licenciado GPL). Desde un punto de vista del desarrollador, la GPL presenta un inconveniente en este punto, y lamentablemente mucha gente se cierra en banda ante el uso de dicho código. No obstante, la MPL no es tan excesivamente permisiva como las licencias tipo BSD. Estas licencias son denominadas de *copyleft* débil.

La NPL (luego la MPL) fue la primera licencia nueva después de muchos años, que se encargaba de algunos puntos que no fueron tenidos en cuenta por las licencias BSD y GNU. En el espectro de las licencias de software libre se la puede considerar adyacente a la licencia estilo BSD, pero perfeccionada.

Hay que hacer constar que el titular de los derechos de autor (*copyright*) de un software bajo licencia *copyleft* puede también realizar una versión modificada bajo su *copyright* original, y venderla bajo cualquier licencia que desee, además de distribuir la versión original como software libre. Esta técnica ha sido usada como un modelo de negocio por una serie de empresas que realizan software libre (por ejemplo MySQL); esta práctica no restringe ninguno de los derechos otorgados a los usuarios de la versión *copyleft*. También podría retirar todas las licencias de software libre anteriormente otorgadas, pero esto obligaría a una indemnización a los titulares de las licencias en uso.

1.2.4. Diferencia entre software *open source* y software libre

Aunque en la práctica el software Open Source y el software libre comparten las mismas licencias, la FSF (Free Software Foundation) opina que el movimiento Open Source es filosóficamente diferente del movimiento del software libre y apareció en 1998 con un grupo de personas, entre los que cabe destacar a Eric S. Raymond y Bruce Perens, que formaron la Open Source Initiative (OSI). Buscaban darle mayor relevancia a los beneficios prácticos del

compartir el código fuente, e interesar a las principales casas de software y otras empresas de la industria de la alta tecnología en el concepto.

Estos defensores ven que el término *open source* evita la ambigüedad del término inglés *free* en *free software*. El nombre *open source* fue acuñado por Christine Peterson del *think tank* Foresight Institute y se registró para actuar como marca registrada para los productos de software libre. Mucha gente reconoce el beneficio cualitativo del proceso de desarrollo de software cuando los desarrolladores pueden usar, modificar y redistribuir el código fuente de un programa.

El movimiento del software libre hace especial énfasis en los aspectos morales o éticos del software, viendo la excelencia técnica como un producto secundario deseable de su estándar ético mientras que el movimiento de código abierto ve la excelencia técnica como el objetivo prioritario, siendo la compartición del código fuente un medio para dicho fin. Por tal motivo, la FSF (Free Software Foundation) se distancia tanto del movimiento de código abierto como del término *open source*.

Puesto que la OSI solo aprueba las licencias que se ajustan a la OSD (*Open Source Definition*), la mayoría de la gente lo interpreta como un esquema de distribución, e intercambia libremente *open source* con software libre. Aun cuando existen importantes diferencias filosóficas entre ambos; especialmente en términos de las motivaciones para el desarrollo y el uso de este software, raramente suelen tener impacto en el proceso de colaboración.

Aunque el término *open source* elimina la ambigüedad de libertad frente a precio (en el caso del inglés), introduce una nueva: entre los programas que se ajustan a la Open Source Definition, que dan a los usuarios la libertad de

mejorarlos, y los programas que simplemente tiene el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente.

Muchas personas creen que cualquier software que tenga el código fuente disponible es *open source*, puesto que lo pueden manipular (un ejemplo es el paquete de software gratuito Graphviz, inicialmente no libre, pero que incluía el código fuente, aunque luego AT&T le cambió la licencia).

1.2.5. Breve historia de Arduino

El proyecto empezó en Ivrea, Italia (el lugar de la compañía de computadoras Olivetti), en el 2005, con el fin de crear un dispositivo para controlar proyectos interactivos de diseño construidos por estudiantes, que tuviera un menor costo que otros sistemas de prototipos disponibles en esa época. Para mayo de 2011, más de 300 000 unidades de Arduino existen en el mundo. Los fundadores Massimo Banzi y David Cuartielles nombraron el proyecto “Arduin of Ivrea” el cual es el personaje histórico principal del pueblo. “Arduino” es un primer nombre italiano masculino, que significa amigo fuerte. La versión en inglés de dicho nombre es Hardwin

.

El proyecto Arduino es una bifurcación de la plataforma de código abierto Wiring. Wiring fue creado por el artista y programador colombiano Hernando Barragán como una tesis de maestría en el Interaction Desing Institute Ivrea bajo la supervisión de Massimo Banzi y Casey Reas. Además, Wiring está basado en Processing y su entorno de desarrollo integrado creado por Casey Reas y Ben Fry.

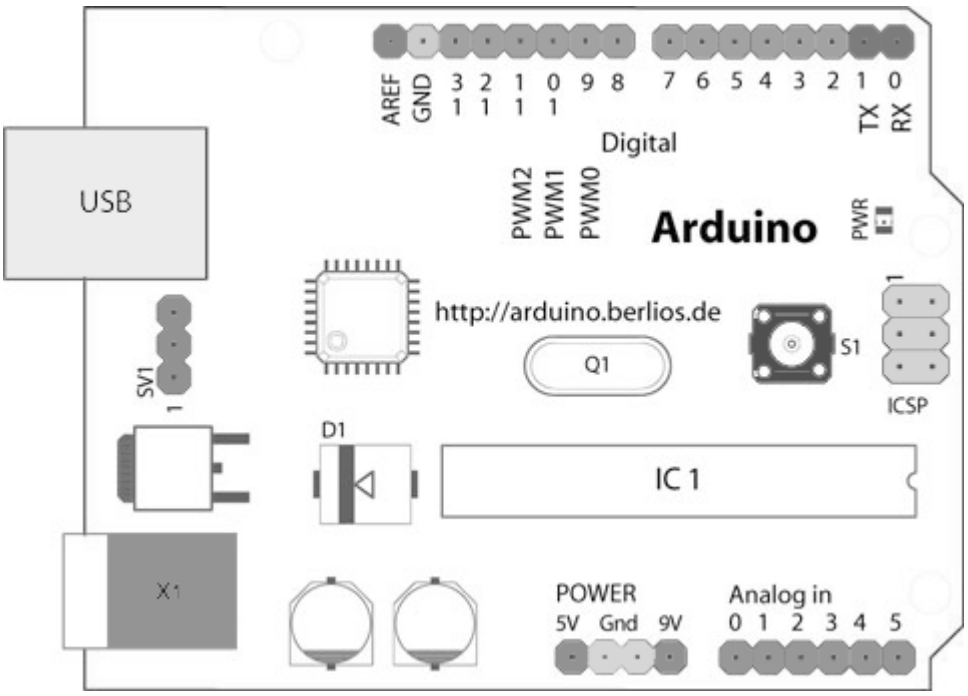
1.3. Estructura básica de los componentes

Arduino está pensado para proveer el hardware necesario para interconectarse con los componentes externos comúnmente más utilizados por aplicaciones que requieren procesamiento de las señales.

1.3.1. Introducción a la placa Arduino

Al observar la placa de arriba hacia abajo, este es un bosquejo de cómo está conformada la placa (las partes de la placa con las cuales se interactúa normalmente están resaltadas):

Figura 2. **Componentes básicos de una placa Arduino**



Fuente: Placa Arduino. http://arduino.berlios.de.
Consulta: 10 de junio de 2014.

Empezando en el sentido de las manecillas del reloj desde la parte central superior:

- *Analog reference pin* (AREF)
- *Digital ground* (GND)
- *Digital pins* (13 a 2)
- *Digital pins* (0-1 Serial In/Out - TX/RX) – estas terminales no pueden ser utilizadas como lectura o escritura digital (digitalRead and digitalWrite) si ya se están utilizando para comunicación serial (Serial.begin).
- *Reset button* (S1)
- *In-circuit serial programmer* (ICSP)
- *Analog in pins* (0 a 5)
- *Power and ground pins* (power 5V, ground Gnd, power 9V)
- *External power supply in* de 9 a 12 VDC (X1)
- *Toggles external power* y USB *power* (SV1) - *jumper* en pines más cercanos a la fuente de poder deseada.
- USB (utilizada para cargar programas a la placa y para la comunicación serial entre la placa y la computadora. Además puede ser utilizada para energizar la placa)

1.3.1.1. Pines digitales

Adicionalmente a las funciones específicas mencionadas anteriormente, los pines digitales en una placa Arduino pueden ser utilizadas para entradas y salidas de propósito general mediante los comandos pinMode( ), digitalRead( ) y digitalWrite( ). Cada terminal tiene una resistencia interna de polarización la cual puede ser encendida y apagada utilizando el comando digitalWrite( ) (con el valor “High” o “Low” respectivamente) cuando dicha terminal es configurada como entrada. La corriente máxima por cada pin es de 40 mA.

- Serial: 0 (RX) y 1 (TX). Utilizados para recepción (RX) y transmisión (TX) de datos TTL en serie. En el Arduino Diceimila, estos pines están conectados a los pines correspondientes del Chip Serial FTDI USB-to-TTL. En el Arduino BT, estos están conectados a los pines correspondientes del módulo Bluetooth WT11. En el Arduino Mini y Arduino LilyPad estos pines están destinados para el uso con el módulo serial TTL externo (mini-USB Adapter)
- Interrupciones internas: 2 y 3. Estos pines pueden ser configurados para disparar una interrupción en un valor bajo (0 lógico), un flanco ascendente o descendente, o un cambio de estado (lógico). Se recomienda leer sobre la función attachInterrupt( ) para más detalles.
- PWM: 3,5,6,9,10 y 11. Provee una salida PWM de 8 bits con la función analogWrite( ). En placas que tienen un ATmega8, la salida PWM se encuentra disponible solo en las terminales 9, 10 y 11.
- BT Reset: 7. (Solo para Arduino BT). Terminal conectada a la línea de reset del módulo Bluetooth.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estos pines soportan comunicación SPI (Serial Peripheral Interface), la cual, aunque es soportada por el hardware, no está incluida actualmente en el lenguaje Arduino.
- LED : 13. En el Diecimila y LilyPad existe un LED integrado que está conectado a la terminal digital 13. Cuando el pin tiene un valor alto, el led se enciende, cuando el pin tiene un valor bajo, el led se apaga.

1.3.1.2. Pines análogos

Adicionalmente a las funciones específicas mencionadas anteriormente, las terminales análogas soportan una conversión análoga a digital (ADC) de 10 bits de resolución, utilizando la función analogRead( ). La mayoría de estas

entradas análogas pueden ser utilizadas también como terminales digitales: entrada análoga 0 como pin digital 14 hasta la entrada análoga 5 como pin digital 19. Las entradas análogas 6 y 7 (presentes en la placa Mini y en la BT) no pueden ser utilizadas como pines digitales.

- I^2C: 4 (SDA) y 5 (SCL). Soporta comunicación I^2C (TWI) utilizando la librería Wire (se puede encontrar mayor documentación en el sitio web de Wiring)

1.3.1.3. Pines de alimentación

La mayoría de placas tiene dos diferentes valores de voltajes de entrada (con excepción del modelo Diecimila, el cual soporta hasta tres valores diferentes).

- VIN (a veces llamado "9V"): El voltaje de entrada a la placa Arduino cuando está siendo alimentado por una fuente de poder externa (diferente a los 5 voltios de la conexión USB o de otra fuente de poder regulada). Se puede energizar directamente este pin o si se suministra el voltaje mediante un conector, acceder a dicho conector mediante este pin. Tomar en cuenta que diferentes placas aceptan diferentes rangos de voltajes de entrada, por favor referirse a la documentación específica de la placa para mayor información. Además, tomar en cuenta que LilyPad no tiene terminal de VIN y solo acepta una entrada regulada.

- 5V: La fuente de poder regulada utilizada para energizar el microcontrolador y otros componentes de la placa. Este voltaje puede venir ya sea de la terminal VIN por medio de un regulador integrado en la placa o puede ser suministrado via USB o alguna otra fuente regulada de 5V.

- 3v3: Una fuente de voltaje de 3.3V generada por el chip integrado FTDI (solo Diecimila).
- GND. Terminales de tierra.

1.3.1.4. Otros pines

Adicionalmente la placa Arduino tiene otras funcionalidades para lo cual existen algunos pines que se pueden configurar de la siguiente forma:

- AREF: voltaje de referencia para las entradas análogas. Utilizado con la función analogReference.
- Reset: al colocar en estado bajo esta línea se resetea el microcontrolador (solo Diecimila).

1.3.2. Microcontroladores

Los microcontroladores Arduino Diecimila, Arduino Duemilanove y Arduino Mega están basados en Atmega168, Atmega 328 y Atmega1280.

Los valores operativos de voltaje y corriente no tienen una gran variación entre los microcontroladores, siendo principalmente la capacidad de memoria y procesamiento la diferencia más notoria. En la tabla I se observan estas características para los tres tipos de microcontroladores mencionados.

Tabla I. **Especificaciones técnicas de microcontroladores de placas Arduino**

	Atmega168	Atmega328	Atmega1280
Voltaje operativo	5 V	5 V	5 V
Voltaje de entrada recomendado	7-12 V	7-12 V	7-12 V
Voltaje de entrada límite	6-20 V	6-20 V	6-20 V
Pines de entrada y salida digital	14 (6 proporcionan PWM)	14 (6 proporcionan PWM)	54 (14 proporcionan PWM)
Pines de entrada analógica	6	6	16
Intensidad de corriente	40 mA	40 mA	40 mA
Memoria flash	16KB (2KB reservados para el bootloader)	32KB (2KB reservados para el bootloader)	128KB (4KB reservados para el bootloader)
SRAM	1 KB	2 KB	8 KB
EEPROM	512 bytes	1 KB	4 KB
Frecuencia de reloj	16 MHz	16 MHz	16 MHz

Fuente: Proyecto Arduino. *Hojas de especificaciones de microcontroladores.* http://playground.arduino.cc/Main/ArduinoCoreHardware. Consulta: 10 de junio de 2014.

1.4. Aplicaciones generales y modelos de placas

Las aplicaciones que ofrece Arduino son múltiples, y depende del uso, creatividad y necesidad de los usuarios. Mediante sensores se pueden crear aplicaciones sencillas enfocadas a la docencia para estudiantes de electrónica, proyectos más elaborados para la industria o incluso sistemas dirigidos simplemente al ocio. Es muy utilizado también en los entornos artísticos para crear obras más elaboradas, dada su facilidad de programación.

El núcleo del equipo de desarrollo de Arduino está formado por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, David Mellis y Nicholas Zambetti.

Debido a las posibilidades que presentan sus entradas, Arduino permite utilizar una gran variedad de sensores, como se ha dicho. Por ejemplo se puede utilizar para aplicaciones como controlar un relay, leer la posición de un potenciómetro, o como controlar el giro de un servomotor.

1.4.1. Patch Pduino

Pduino nace de la fusión de Pure Data y Arduino. Ambos de código abierto, permiten trabajar el uno con el otro de una manera gráfica e intuitiva. Cargando el firmware de Pure Data (PD) a la placa Arduino se puede acceder a ella mediante el lenguaje de programación gráfico. Así mismo, observar que se tiene todo el esquema físico de la placa, con los puertos, pines digitales y analógicos, entradas y salidas en una interfaz gráfica, gracias a PD.

1.4.2. Minibloq

Es un entorno gráfico de programación que puede generar código nativo de Arduino y escribirlo directamente en la memoria *flash* de la placa. No necesita por lo tanto, ni de un firmware específico en la placa Arduino ni de conexión en tiempo de ejecución. Tiene un modo donde permite visualizar el código generado, el cual también puede ser copiado y pegado en el Arduino-IDE, para los usuarios que intentan hacer el pasaje de una herramienta gráfica a la programación en sintaxis C/C++. Minibloq es de uso libre y sus fuentes también están disponibles gratuitamente. Una característica importante es que puede correr también en la XO (OLPC), mediante Wine.

1.4.3. LilyPad Arduino

Es una placa con microcontrolador diseñado para prendas y e-textiles. Puede utilizarse con complementos similares como fuentes de alimentación, sensores actuadores unidos por hilo conductor. La placa está basada en el ARmega168V (la versión de bajo consumo del ATmega168), o el ATmega328V. El LilyPad Arduino ha sido diseñado y desarrollado por Leah Buechley y SparkFun Electronics.

El LilyPad Arduino se puede programar con el software de Arduino versión 0010 o superior. Así mismo, programarse con versiones anteriores, pero todas las funciones relacionadas con el tiempo estarán deshabilitadas (ira la mitad de rápido de lo que debería). El ATmega168V o ATmega328V del Arduino LilyPad viene precargado con un gestor de arranque *bootloader*, que permite cargar nuevo código sin necesidad de un programador por hardware externo. Se comunica utilizando el protocolo STK500 original.

Respecto a la energización de la placa, esta se realiza por medio de la conexión USB o con una fuente externa. Si se utiliza una fuente externa, debe proporcionar entre 2,7 y 5,5 voltios. Esta puede ser un transformador o una batería.

El LilyPad Arduino es un círculo de aproximadamente 50mm (2") de diámetro. la placa en sí mide 8 mm (1/32") de grosor (aproximadamente 3 mm (1/8") donde tiene ubicada los componentes).

1.4.4. Arduino Nano

Es una pequeña y completa placa basada en el ATmega328 (Arduino Nano 3.0) o ATmega168 (Arduino Nano 2.x) que se usa conectándola a un protoboard. Tiene más o menos la misma funcionalidad que el Arduino Duemilanove, pero con una presentación diferente. No posee conector para alimentación externa, y funciona con un cable USB Mini-B en vez del cable estandar. El nano fue diseñado y está siendo producido por Gravitech.

El Arduino Nano puede ser alimentado usando el cable USB Mini-B, con una fuente externa no regulada de 6-20V (pin 30), o con una fuente externa regulada de 5V (pin 27). La fuente de alimentación es seleccionada automáticamente a la de mayor voltaje.

El chip FTDI FT232RL que posee el Nano solo es alimentado si la placa está siendo alimentada usando el cable USB. Como resultado, cuando se utiliza una fuente externa (no USB), la salida de 3.3V (la cual es proporcionada por el chip FTDI) no está disponible y los pines 1 y 0 parpadearán si los pines digitales 0 o 1 están a nivel alto.

El ATmega168 posee 16KB de memoria *flash* para almacenar el código (de los cuales 2KB son usados por el *bootloader*); el ATmega 328 posee 32KB, (también con 2 KB usados por el *bootloader*). El Atmega168 posee 1KB de SRAM y 512 bytes de EEPROM (puede ser leída y escrita con la librería EEPROM); el ATmega328 posee 2 KB de SRAM y 1KB de EEPROM.

El Arduino Nano tiene algunos métodos para la comunicación con un PC, otro Arduino, u otros microcontroladores. El ATmega168 y el ATmega328 poseen un módulo UART que funciona con TTL (5V)el cual permite una comunicación vía serie, la cual está disponible usando los pines 0 (RX) y 1 (TX). El microcontrolador FTDI FT232RL en la placa, hace de puente a través del USB para la comunicación serial y los controladores FTDI (incluidos con el software de Arduino) proveen al PC de un puerto com virtual para el software en el PC. El software Arduino incluye un monitor serial que permite visualizar en forma de texto los datos enviados desde y hacia la placa Arduino. Los leds RX y TX en la placa parpadearán cuando los datos se estén enviando a través del microcontrolador FTDI y la conexión USB con la PC (pero no para la comunicación directa a través de los pines 0 y 1).

El Arduino Nano puede ser programado con el software de Arduino. El ATmega168 o ATmega328 del Arduino Nano vienen preprogramados con un *bootloader* que permite subir el código al Arduino sin la necesidad de un programador externo. Se comunica usando el protocolo STK500 original. También se puede programar el microcontrolador usando un programador ICSP (In-Circuit Serial Programming, Pogramación Serie En-Circuito).

En vez de necesitar pulsar un botón físico de reset, el Arduino Nano ha sido diseñado de tal manera que permite ser reiniciado por el software de la PC a la que está conectado. Una de las líneas de control de flujo por hardware

(DTR) del microcontrolador FT232RL está conectada a la línea de reset del ATmega168 o ATmega328 por medio de un condensador de 100 nanofaradios. Cuando esta línea se pone a nivel bajo, la línea de reset se mantiene a nivel bajo el suficiente tiempo para causar el reinicio del microcontrolador. El software de Arduino usa esta capacidad para permitir cargar código en la placa pulsando simplemente el botón *upload* en el entorno software de Arduino. Esto significa que el tiempo de espera del *bootloader* es más pequeño, ya que el tiempo en el que se encuentra a nivel bajo el DTR puede ser coordinado bien con el inicio de la carga del código.

Esta configuración tiene otras implicaciones. Cuando el Nano se conecta a un PC que funciona con Mac OS X o Linux, se resetea cada vez que se hace la conexión con el software (a través del USB). Durante el siguiente medio segundo, más o menos, el *bootloader* está corriendo en el Nano. Como el *bootloade*r ha sido para programado para ignorar cualquier dato erróneo (cualquier dato que no sea la carga de nuevo código), ignorará los primeros bytes que se reciban justo después de hacer la conexión. Si un *sketch* cargado en la placa recibe algún tipo de configuración o algún otro tipo de dato importante nada más iniciarse, asegurarse que el software con el que se comunique, espere al menos un segundo antes de enviar datos para que no sean ignorados por el *bootloader*.

2. DISEÑO BÁSICO DE CONTROL POR MEDIO DE ARDUINO

2.1. Envío de información de control

Arduino se comunica básicamente de dos formas con dispositivos externos. La primera es con la computadora (vía serial), para cargar el código y envío de comandos que se necesiten según el objetivo del programa. La segunda forma es por medio de sensores que envían datos o información necesaria para que el programa actúe de acuerdo a condiciones preestablecidas.

2.1.1. Comunicación serial

Se utiliza para la comunicación entre la placa Arduino y un ordenador u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie (también conocido como UART o USART). Se comunica a través de los pines digitales 0 (RX) y 1 (TX), así como con el ordenador mediante USB. Por lo tanto, si se utilizan estas funciones, no se pueden usar los pines 0 y 1 como entrada o salida digital.

Se puede utilizar el monitor del puerto serie incorporado en el entorno de desarrollo de para comunicarse con la placa Arduino seleccionando la misma velocidad en baudios utilizada en la función Begin.

La placa Arduino Mega tiene tres puertos seriales adicionales: serial1 en los pines 19 (RX) y 18 (TX), serial2 en los pines 16 (RX) y 17 (TX), serial3 en los pines 15 (RX) y 14 (TX). Para utilizar estos pines para comunicarse con la

computadora, es necesario un adaptador USB a serial, ya que no están conectados al adaptador USB-Serie de la placa Arduino Mega. Para usarlos para comunicarse con un dispositivo serie externo TTL, se debe conectar el pin TX de la placa al pin RX del dispositivo, el RX de la placa al pin TX del dispositivo, y el GND de Arduino Mega a tierra del dispositivo. (No se debe conectar estos pines directamente a un puerto serie RS232, que operan a +/- 12V debido a que esto puede dañar la placa Arduino).

2.1.1.1. Función Begin

Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Las velocidades de conexión a una computadora pueden ser: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200. Sin embargo, se pueden especificar otras velocidades para comunicarse a través de los pines 0 y 1 con un componente que requiere una velocidad de transmisión diferente a los mencionados

A continuación se observa la sintaxis de la función Begin, donde el parámetro velocidad se refiere a los baudios utilizados para la conexión y el tipo de datos es *long*.

Figura 3. **Sintaxis función Begin**

```
void conexionserial1( ) {
  Serial.begin (9600);    //  conexión establecida en 9600 bps
}
```

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

2.1.1.2. Función End

Finaliza la conexión de la comunicación serial, permitiendo a los pines RX and TX ser usados como entradas o salidas digitales.

La sintaxis de la función es: serial.end( )

2.1.1.3. Función Read

Esta función se utiliza para leer los datos del puerto serial, el resultado de su ejecución devuelve el primer byte recibido por el puerto serie, de lo contrario devuelve -1 si no hay datos disponibles. El tipo de datos de la función es *int*.

La sintaxis de la función es: serial.read( )

2.1.1.4. Función Available

Devuelve el número de bytes (caracteres) disponibles para ser leídos por el puerto serie. Se refiere a datos ya recibidos y disponibles en el búfer de recepción del puerto (que tiene una capacidad de 128 bytes)

La sintaxis de la función es: serial.available( )

2.1.1.5. Función Print

Imprime los datos al puerto serie como texto ASCII. Este comando puede tomar muchas formas. Los números son impresos mediante un juego de caracteres ASCII para cada dígito. Los valores de tipo *float* (valores con punto decimal) son impresos en forma de dígitos ASCII con dos decimales por

defecto. Los valores tipo *byte* se envían como un único carácter. A continuación se muestran varios ejemplos de cómo son enviados los caracteres:

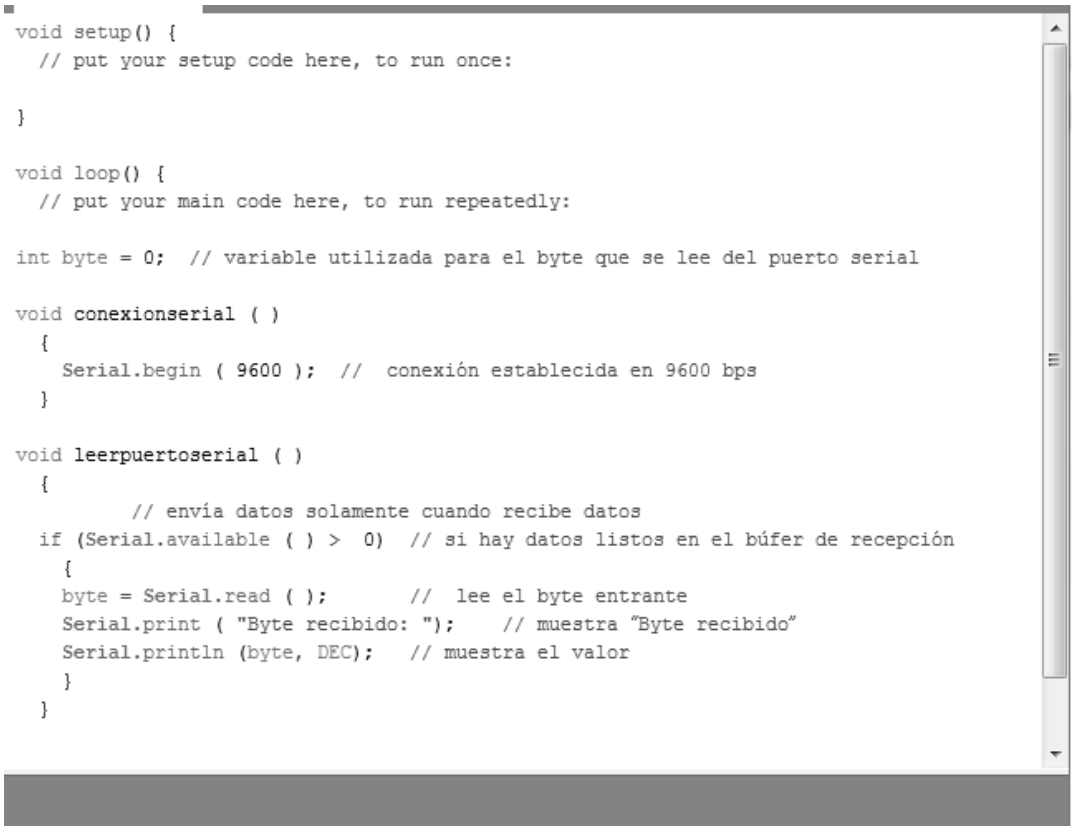
- Serial.print (5) muestra " 5 "
- Serial.print (3.1415) muestra " 3.14 "
- Serial.print (byte (60)) muestra " < " (cuyo código ASCII es 60)
- Serial.print('N') imprime " N "
- Serial.print(" prueba ") imprime " prueba "

Un segundo parámetro opcional especifica la base (formato) a usar; los valores permitidos son de tipo *byte*, bin (binarios o base 2), oct (octales o base 8), dec (decimales o base 10), hex (hexadecimales o base 16). Para números de coma flotante, este parámetro especifica el número de posiciones decimales a usar. Por ejemplo:

```
Serial.print ( 60, BYTE ) muestra " < "
Serial.print ( 78, BIN ) muestra " 1001110 "
Serial.print ( 78, OCT ) muestra " 116 "
Serial.print ( 5, DEC ) muestra " 5 "
Serial.print ( 10, HEX ) muestra " A "
Serial.println ( 3.1415, 0 ) muestra " 3 "
Serial.println ( 3.1415, 2 ) imprime " 3.14 "
Serial.println ( 3.1415, 4 ) imprime " 3.1415 "
```

La sintaxis de la función es: serial.print (valor, formato). Valor es el número que se muestra, de cualquier tipo dependiendo de formato (si se especifica). La figura 4 muestra un ejemplo que imprime en pantalla el valor que recibe del puerto serial.

Figura 4. **Programa que lee datos del puerto serial**



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

int byte = 0;  // variable utilizada para el byte que se lee del puerto serial

void conexionserial ( )
  {
    Serial.begin ( 9600 );  //  conexión establecida en 9600 bps
  }

void leerpuertoserial ( )
  {
          // envía datos solamente cuando recibe datos
  if (Serial.available ( ) >  0)  // si hay datos listos en el búfer de recepción
    {
    byte = Serial.read ( );       //  lee el byte entrante
    Serial.print ( "Byte recibido: ");    // muestra "Byte recibido"
    Serial.println (byte, DEC);   // muestra el valor
    }
  }
```

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

2.1.1.6. Función Flush

Esta función se encarga de vaciar el búfer de entrada de datos en serie. Es decir, cualquier llamada a la función Serial.read o Serial.available devolverá solo los datos recibidos después la llamada más reciente a Serial.flush.

La sintaxis es: Serial.flush( ) y no lleva ningún parámetro ni devuelve ningún valor.

2.1.1.7. Función Write

Escribe datos binarios en el puerto serie. Estos datos se envían como un byte o una serie de bytes. Para enviar los caracteres que representan los dígitos de un número se debe utilizar la función Print.

La sintaxis de la función puede ser de las siguientes formas:

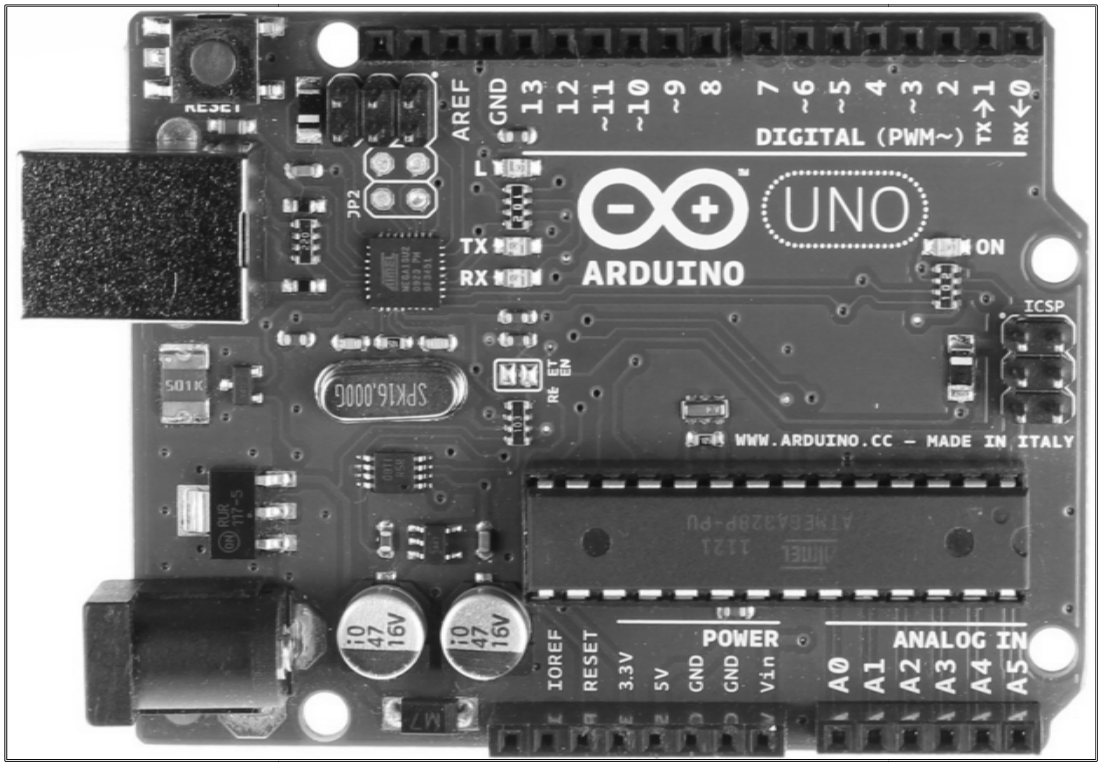
- Serial.write (val) – val se envía como un solo byte.
- Serial.write (str) – str es una cadena que se envía como una serie de bytes.
- Serial.write (buf, len) – buf es un arreglo que se envía como una serie de bytes y len es la longitud del búfer.

2.2. Placa base de Arduino Uno y estructura de componentes

Una de las placas de circuitería más utilizadas por su versatilidad y simplicidad a la hora de la interconexión dispositivos externos como sensores y motores entre otros es la placa Arduino Uno. Esta se ha popularizado en el entorno educativo debido a las características del microcontrolador que incluye.

La figura 5 muestra la imagen de la versión comercial de esta placa, la cual es ampliamente vendida a nivel mundial.

Figura 5. **Placa Arduino Uno**



Fuente: http://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpg.
Consulta: 15 de junio de 2014.

Arduino Uno es una placa electrónica basada en el ATmega328. Cuenta con 14 pines de entradas / salidas digitales (de los cuales 6 pueden ser utilizados como salidas PWM), 6 entradas analógicas, un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, unos pines para ICSP, y un botón de reinicio. Contiene todo lo necesario para apoyar al microcontrolador, solo se conecta a un ordenador con un cable USB o se energiza con un adaptador AC-DC o la batería para empezar.

Uno difiere de todas las placas anteriores en que no utiliza el microcontrolador USB FTDI a serie. Por el contrario, cuenta con la Atmega16U2 (Atmega8U2 hasta la versión R2) programado como un convertidor de USB a serie. La segunda revisión de la placa Uno tiene una resistencia que conecta la línea 8U2 HWB a tierra, haciendo más fácil poner en modo DFU (Device Firmware Update o Actualización del Firmware del dispositivo). La tercera revisión de la placa tiene las siguientes nuevas características:

- 1.0 pinout: añade pines SDA y SCL que se encuentran cerca al pin AREF y dos nuevos pines colocados cerca del pin de RESET, el IOREF que permite a los componentes adaptarse a la tensión proporcionada por la placa. En el futuro, estos componentes serán compatibles tanto con la tarjeta que utiliza el regulador, que operan con 5V y con el Arduino Due que funciona con 3.3V. El segundo es un pin que no está conectado, que se reserva para usos futuros.
- Circuito de reset más robusto.
- El Atmega 16U2 reemplaza al 8U2.

Uno es llamado así con motivo del lanzamiento de Arduino 1.0, la cual se convirtió en la versión de referencia de Arduino hasta la fecha.

2.2.1. Resumen de características

Se debe tener en cuenta que las características de la placa Arduino Uno no difieren demasiado en relación a otras placas de circuitería digitales. En la siguiente tabla se muestran los valores nominales para el correcto funcionamiento de la placa.

Tabla II. **Valores nominales de placa Arduino Uno**

Microcontrolador	ATmega328
Voltaje de operación	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límites)	6-20V
Pines digitales de E/S	14 (de los cuales 6 proveen salida PWM)
Pines de entradas analógicas	6
Corriente DC por pin E/S	40 mA
Corriente DC por pin 3.3V	50 mA
Memoria flash	32 KB (ATmega328) de los cuales 0.5 KB son usados por gestor de arranque
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Frecuencia del reloj	16 MHz

Fuente: Proyecto Arduino. *Hojas de especificaciones de microcontroladores.* http://playground.arduino.cc/Main/ArduinoCoreHardware. Consulta: 10 de junio de 2014.

2.2.2. Energización

Arduino UNO puede ser energizada vía conexión USB o por medio de alguna fuente de poder externa. La fuente de poder es seleccionada automáticamente. En caso se utilice una fuente de poder externa, esta puede

ser un adaptador de AC/DC conectado a la energía comercial de 120 voltios o de una batería

En el caso que se utilice un adaptador, se conecta mediante un *plug* de 2.1 mm (con polaridad positiva en el centro del mismo) que se coloca en la terminal de poder de la placa. Los cables de la batería pueden ser insertados en los pines de GND y VIN del conector de poder.

La placa puede operar con una fuente externa de 6 a 20 voltios. Sin embargo, si la placa es energizada con menos de 7 voltios, el pin de 5V puede proveer menos de los cinco voltios y la placa puede presentar un comportamiento inestable en cuanto al voltaje. Si por el contrario, se energiza la placa con más de 12 voltios, el regulador de voltaje se puede sobrecalentar y dañar la placa. El rango recomendado de voltaje es de 7 a 12 voltios.

Los pines de poder son los siguientes:

- Pin VIN**:** el voltaje de entrada de la placa Arduino Uno cuando se está utilizando una fuente de poder externa (que no sean los 5 voltios desde una conexión USB u otra fuente regulada de voltaje). Se puede conectar directamente el voltaje a través de este pin, o, si se está energizando la placa por medio de un conector de alimentación, se puede conectar dicho conector por medio de este pin.

- Pin 5V: este pin tiene una salida de 5V regulados provenientes del regulador propio de la placa. Como se mencionó anteriormente, se puede energizar la placa, ya sea con un conector de alimentación de DC (7 – 12V), por medio del conector USB (5V), o mediante el pin VIN. Si se alimenta la placa mediante los pines de alimentación 5V y 3.3V, el voltaje

de alimentación no pasa por el regulador, y por lo tanto se corre riesgo de dañar la placa. No es aconsejable.

- Pin 3V3**:** una salida de 3.3 voltios generada por el regulador integrado en la placa. El máximo consumo de corriente es de 50 mA.

- Pin GND: pines de tierra.

2.2.3. Memoria

El microcontrolador ATmega328 tiene 32 KB (0.5 KB usados por el gestor de arranque). Además tiene 2 KB de SDRAM y 1KB de EEPROM (la cual puede ser leída y escrita desde la librería de EEPROM).

2.2.4. Terminales de entrada y salida

Cada uno de los 14 pines digitales en la placa Uno puede ser utilizado como entrada o salida, dependiendo de la configuración que se realice mediante las funciones pinMode, digitalWrite y digitalRead. Estos pines operan con 5 voltios y cada uno puede proveer o recibir una corriente máxima de 40 mA, con una resistencia interna de polarización (desconectada por defecto) de 20–50 KOhms. Adicionalmente, algunas terminales se pueden configurar para tareas especializadas:

- Serial: 0 (RX) y 1 (TX) utilizados para recepción (RX) y transmisión (TX) de datos TTL en forma serial. Estos pines están conectados a los pines del chip ATMmega8U2 USB – TTL serial correspondiente.
- Interrupciones externas: terminales 2 y 3. configurados para disparar (activar) una interrupción en un nivel bajo lógico, un flanco ya sea

ascendente o descendente o un cambio de estado lógico. Para mayores detalles se puede consultar la función attachInterrupt.

- PWM: terminales 3, 5, 6, 9, 10, y 11 las cuales pueden ser configuradas como salidas PWM de 8 bits con la función analogWrite.
- SPI: terminales 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estos pines soportan comunicación SPI (Serial Peripheral Interface) utilizando la librería SPI.
- Led: 13**.** Existe un led integrado a la placa que está conectado al pin 13. Cuando el valor del pin es alto (1 lógico) el led se enciende, cuando el valor del pin es bajo (0 lógico), se apaga.

La placa Uno tiene 6 entradas analógicas que se etiquetan A0 hasta A5, cada una de las cuales provee una resolución de 10 bits (1024 valores diferentes). Por defecto, la escala va de tierra (0V) a 5V, pero es posible cambiar el límite superior del rango utilizando el pin AREF y la función analogReference. Adicionalmente algunos pines tienen funciones especializadas, por ejemplo la terminal A4 o SDA y terminal A5 o SCL**,** las cuales soportan comunicación TWI utilizando la librería *Wire*.

Hay algunos otros pines en la placa que tienen otras funciones:

- AREF: voltaje de referencia para entradas analógicas. Utilizado con analogReference.
- Reset**.** Al colocar esta línea en un valor bajo se resetea el microcontrolador. Normalmente se utiliza para agregar un botón de *reset* y se colocar en placas auxiliares para poder reiniciarlo.

2.2.5. Comunicación

Arduino Uno tiene varias facilidades para comunicarse con una computadora, otra placa Arduino u otros microcontroladores. El microcontrolador ATmega328 provee comunicación serial UART TTL (5V), la cual está disponible en los pines digitales 0 (RX) y 1 (TX). Un ATmega16U2 en la placa canaliza esta comunicación serial, sobre USB, y aparece como un puerto COM virtual en el software de la computadora. El *firmware* "16U2" utiliza los controladores COM estándar de USB y no se necesitan controladores externos. Sin embargo, en ambiente Windows se requiere un archivo de tipo inf. El software Arduino incluye un monitor serial el cual permite el envío de datos simples en formato de texto hacia y desde la placa Arduino. Los LED de TX y RX parpadean cuando haya transmisión/recepción de datos por medio del dispositivo USB – serial y la conexión USB a la computadora (pero no para la comunicación serial en los pines 0 y 1).

La librería SoftwareSerial permite la comunicación serial en cualquiera de los pines digitales de la placa Uno. El microcontrolador ATmega328 soporta comunicación I2C (TWI) y SPI. El software de Arduino incluye la librería Wire para simplificar el uso del bus I2C.

2.2.6. Tipos de programación soportados

Arduino Uno puede ser programado con el software de Arduino, el cual puede ser descargado del sitio web oficial. El entorno de desarrollo oficial es Arduino IDE. El microcontrolador ATmega328 viene preprogramado con un gestor de arranque que permite cargar código nuevo sin utilizar un programador externo de hardware. Dicho gestor de arranque se comunica utilizando el protocolo original STK500. También se puede omitir el gestor de

arranque y programar el microcontrolador a través de la terminal ICSP (In-Circuit Serial Programming).

El código fuente para el *firmware* ATmega16U2 (o 8U2 en las versiones rev1 y rev2) ya viene incluido por defecto y se encuentra cargado con un gestor de arranque DFU, el cuál puede ser activado de las siguientes maneras:

- En las placas con versión Rev1 se conecta el *jumper* en la parte trasera de la placa (cerca del mapa con la forma de Italia), para resetear el 8U2.
- En las placas con versión Rev2 o posteriores existe un resistor que conecta el pin 8U2/16U2 HWB a tierra, haciendo más fácil colocarlo en modo DFU.

Se puede utilizar el software Atmel's FLIP (Windows) o DFU programmer (Mac OS X y Linux) para cargar el nuevo *Firmware*. O también se puede utilizar el pin ISP con un programador externo (sobrescribiendo el gestor de arranque DFU).

La placa Arduino Uno tiene la característica que para el reset automático (software) en vez de requerir presionar físicamente el botón antes de cargar información en la placa, esta permite ser reseteada por software desde una computadora conectada a la placa. Una de las líneas de control de flujo de hardware (DTR) del ATmega8U2/16U2 está conectada a la línea de reset del ATmega328 por medio de un capacitor de 100 nanofaradios. Cuando esta línea es conectada a tierra, la línea de reset tiene un valor lógico de 0 el tiempo suficiente para reiniciar el microcontrolador. El software Arduino usa esta capacidad para permitir cargar el código a la placa simplemente presionando la opción en el programa. Esto significa que el gestor de arranque puede tener un

tiempo más corto sin conexión, mientras el DTR que se encuentra conectado a tierra puede ser bien coordinado con el inicio de la carga.

Esta configuración tiene otras implicaciones. Cuando la placa Uno está conectada, ya sea a la computadora corriendo Mac OS X o Linux, esta se resetea cada vez que se realiza una conexión por software vía USB. Durante el medio segundo siguiente, el gestor de arranque está corriendo en la placa Uno. Mientras tanto se encuentra programado para ignorar datos erróneos (cualquier dato que no sea la carga de nuevo código), por lo cual analizará los primeros bytes enviados a la placa después de abierta la conexión. Si la placa recibe una configuración "que corre solo una vez", hay que asegurar que el software con el cual se comunica espere alrededor de un segundo después de abrir la conexión y antes de enviar los datos.

La placa Uno contiene una opción para deshabilitar el autoreset (reset automático).

2.2.7. Protección sobrecorriente USB

La placa Arduino Uno tiene un "resettable polyfuse" el cual actúa como un termistor no lineal y regresa a estado de conducción luego que la corriente es removida. Este fusible protege los puertos USB de la computadora de sobrecorrientes y corto circuitos. Aunque muchas computadoras contienen su propia protección interna, el fusible provee una capa extra de protección. Si se aplican más de 500 mA al puerto USB, el fusible automáticamente romperá la conexión hasta que el corto o sobrecorriente sea removido.

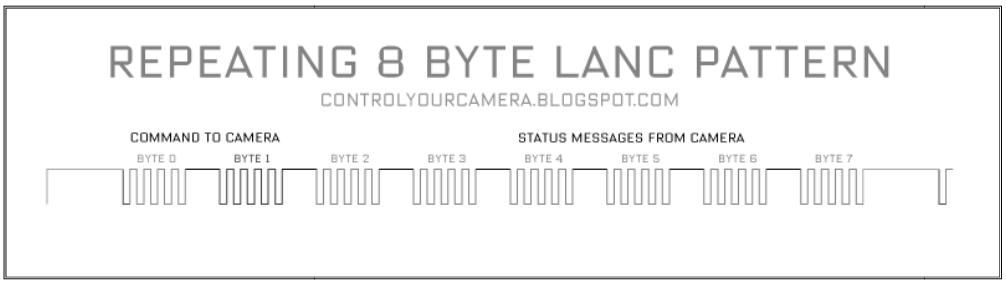
2.2.8. Características físicas

El tamaño máximo de la placa Uno son de 2,7 pulgadas de largo por 2,1 pulgadas de ancho, con el conector USB y de poder (energización) extendiéndose un poco más de las dimensiones mencionadas. Cuatro agujeros para tornillos permiten que la placa sea sujetada a alguna superficie o base.

2.3. Envío de información de audio y video a placa base

Arduino ofrece una gran cantidad de opciones a la hora de conectar un aparato de video para los usos que se requieran. Un protocolo bastante utilizado es el desarrollado por SONY llamado “Bus de control de aplicación local”, por sus siglas en inglés LANC, Local Application Control Bus. LANC se trata de un colector serial abierto de dos vías de 9,600 baudios con lógica invertida. La línea LANC se encuentra en un estado lógico alto (5 voltios) y se cambia a un estado lógico bajo (0 voltios) para enviar comandos o información de control.

Figura 6. **Patrón de bus de control de la aplicación local (LANC)**

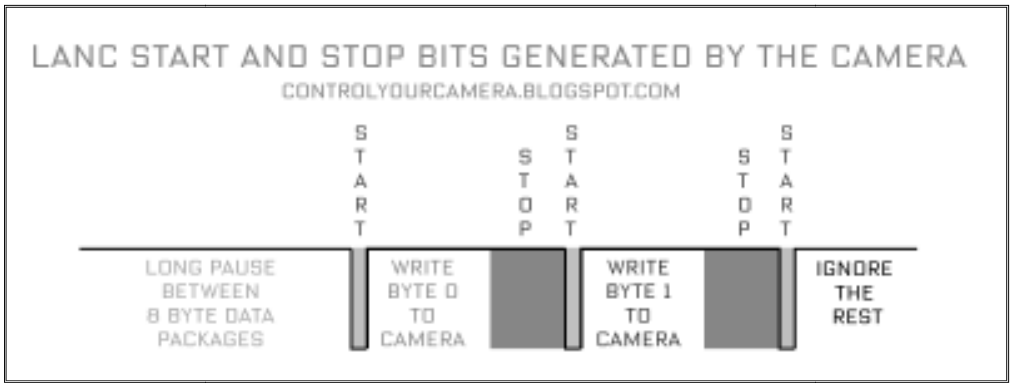


Fuente: Martin Koch. http://controlyourcamera.blogspot.com/2011/02/arduino-controlled-video-recording-over.html. Consulta: 3 de agosto de 2014.

La trama de datos es de 8 bytes, seguida por una pausa mayor a los 5 ms hasta el final de la trama actual. Luego se encuentra otra trama de 8 bytes y otra pausa y así sucesivamente. Existe también una pequeña pausa entre cada byte individual y cada byte es precedido por un bit de inicio.

LANC es una conexión serial de un solo cable y debido a esto la sincronización es crítica. La cámara solo puede recibir comandos durante la primera parte del patrón de la señal. Esto quiere decir que la cámara recibe los primeros 4 bytes y envía información de estado en los últimos 4 bytes. Únicamente los dos primeros bytes son necesarios para controlar una cámara de video, el resto puede ignorarse.

Figura 7. **Bits de inicio y final generados por una cámara, usando LANC**

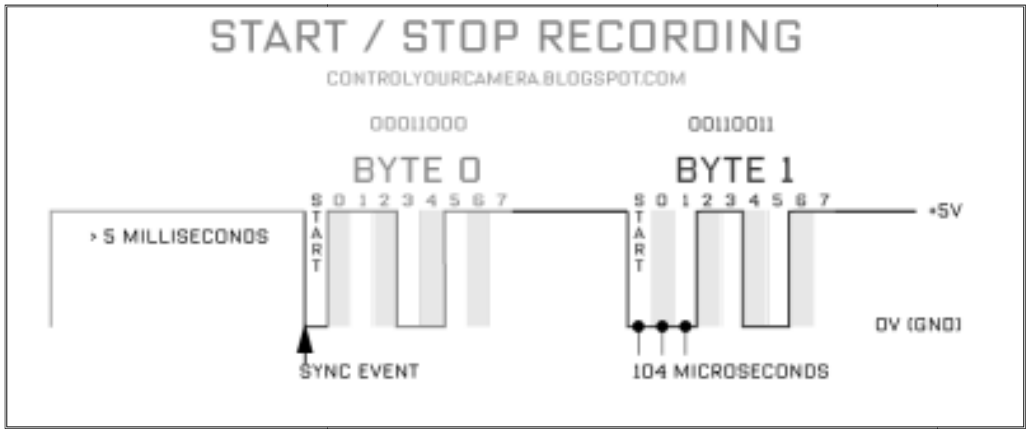


Fuente: Martin Koch. http://controlyourcamera.blogspot.com/2011/02/arduino-controlled-video-recording-over.html. Consulta: 3 de agosto de 2014.

Para enviar un comando a la cámara, este tiene que ser sincronizado a la señal LANC desde la cámara. La cámara coloca un bit de inicio antes y un bit

de finalización después de cada byte. El primer byte del comando enviado desde el controlador (Arduino) debe ser trasmitido exactamente entre el bit de inicio que sigue la pausa larga entre los paquetes de datos de 8 bytes y el bit de finalización siguiente. Luego el segundo byte debe ser enviado a la cámara. Después de enviar estos dos bytes, la señal LANC ya no debe modificarse y debe regresarse al valor bajo (+5V debido a que es lógica inversa).

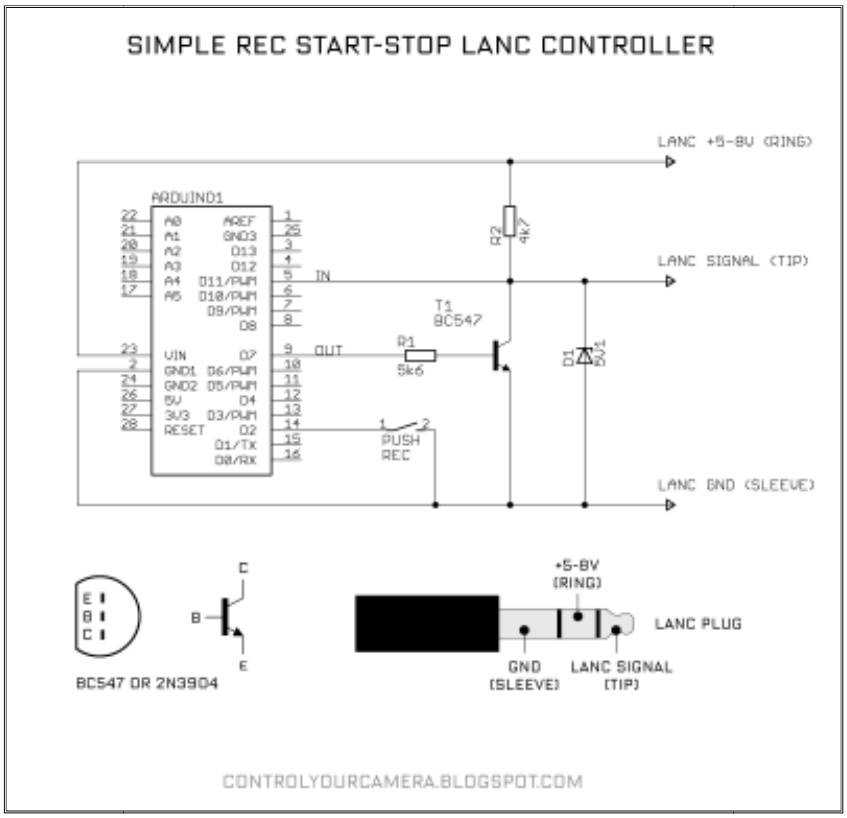
Figura 8. **Período de grabación de la cámara**



Fuente: Martin Koch. http://controlyourcamera.blogspot.com/2011/02/arduino-controlled-video-recording-over.html. Consulta: 3 de agosto de 2014.

Los comandos para iniciar y detener la grabación de video son 18h y 33h (en formato hexadecimal) o 00011000 y 00110011 (en formato binario). Los bytes deben de enviarse con el bit menos significativo (más a la derecha) de primero, por ejemplo para el número 00110011 se debe enviar en la siguiente forma: 11001100. Tomar en cuenta que la señal LANC baja tiene un valor de +5v y la señal alta es 0V (lógica inversa). Los comandos LANC deben ser repetidos 4 veces para que puedan ser aceptados por la cámara.

Figura 9. **Controlador simple de grabación usando LANC**



Fuente: Martin Koch. http://controlyourcamera.blogspot.com/2011/02/arduino-controlled-video-recording-over.html. Consulta: 3 de agosto de 2014.

Debido a que se tiene que escribir y leer mediante un solo cable, se requiere de una interfaz. En la figura 9 se muestra un circuito para lectura/escritura simple desarrollado por Ariel Rocholls, para que este proceso pueda realizarse independientemente.

2.4. Programación vía Flash y Java utilizando comandos de Arduino

Existen 4 componentes principales que son necesarios para que Arduino logre comunicarse mediante el lenguaje de programación Flash:

- El código que se corre en Arduino, los datos son enviados mediante el puerto serial virtual USB.
- Un servidor serial a *socket* que toma los datos del puerto serial y los envía a través del *socket* de red. Está diseñado para ser utilizado con cualquier cliente de software, por ejemplo Flash y con cualquier microcontrolador.
- Una librería Flash que es un conjunto de comandos preestablecidos que aceptan e interpretan los datos enviados desde el *socket* de red.
- Código Flash del usuario que no es más que los comandos personalizados que se encargan de ejecutar lo que el usuario requiere de acuerdo a los datos recibidos. Algunos recursos tienen códigos de prueba mediante los cuales se puede aprender de los ejemplos incluidos en los códigos.

2.4.1. Código a correr en Arduino

Se puede realizar desde cero utilizando comandos mencionados en el Manual de usuario de Arduino en la sección “Comunicación Serial” o se puede utilizar un código de muestra ya preestablecido.

Existen dos juegos de códigos que contienen todo lo necesario para cargar la información a Arduino, y en los cuales después de cierto tiempo se pueden enviar comandos de forma serial para cambiar el funcionamiento del código:

- Firmdata: contiene una base muy completa de código para que, prácticamente se puedan utilizar todos los comandos para realizar lo que se requiera de Arduino. Se está desarrollando un estándar que pueda tener soporte para muchas librerías de software incluyendo Flash 9 (ActionScript 3).
- Arduino: código Flash desarrollado por Stephen Wilson. En pocas palabras, es una versión simple de Firmdata, pero no tiene soporte para Flash.

2.4.2. Serial to Socket Server

La forma de conexión de una placa Arduino con su entorno externo para la comunicación serial, necesita un servidor que sea capaz de poder interpretar y procesar los datos que se envían desde el puerto serial, dependiendo de la aplicación que se esté utilizando. A continuación se mencionan algunas alternativas de servidores que pueden ser utilizados para la conexión con una placa Arduino Uno:

- Serproxy: simple, rápido y robusto. No tiene compatibilidad con Mac OS X 10.6.
- TinkerProxy: basado en Serproxy, actualizado para trabajar con Mac 10.6. Permite el envío y recepción de datos vía serial.
- Servidor Serial SS6: utiliza Java, lo cual lo hace independiente de la plataforma. Pero desafortunadamente, Java no soporta la comunicación vía puerto serial sin que antes se instalen manualmente los complementos necesarios. Esto lo vuelve más complicado y bastante lento establecer la comunicación.

- Arduino2Flash: reemplazo para serproxy en Java para los usuarios que han tenido problema en Mac y OS X. Versión aún no comprobada al 100 % de su funcionamiento.
- NETLab Toolkit Hub: servidor robusto que soporta la comunicación con Arduino (via Firmdata), OSC, XBee y otros productos. Puede ser utilizado con el propio código del usuario que se conecta vía *sockets* (acepta conexiones múltiples simultaneas que acceden al mismo puerto serial), o también puede ser utilizado con las herramientas de arrastrar y pegar del NETLab Toolkit flash. Compatible con Mac y PC.

2.4.3. Flash Socket Library

Las librerías son necesarias para poder incluir el conjunto de instrucciones de la aplicación que se utilice y entre las más utilizadas se encuentran las siguientes:

- Flash CS3 *socket* binario: esquema que muestra como conectar flash a Arduino utilizando un *socket* binario y TinkerProxy. (Desarrollado por Brett Forsyth). Cabe mencionar que aunque no ha sido diseñado como librería, su código es corto y puede ser fácilmente utilizado como un punto de partida.
- Arduino flash AS3 con comunicación con la librería Messenger: desarrollada por Kasper Kamperman. Ejemplo basado en la clase de puerto serial de Tinker. Funciona en conjunto con el ejemplo básico de comunicación de la librería Messenger
- Glue: código Arduino de muestra utilizado con Firmdata.
- Arduino: herramientas flash (Beltran Berrocal). Utiliza Serproxy. Incluye "'Arduino.as Actionscript Class v1.0 (30-12-2005)", una aplicación flash

de muestra que utiliza Arduino.as Actionscript Class v1.0 e información sobre el manejo de la comunicación flash de múltiples bytes.

- Flash Arduino IO: comunicación flash de Kasper Kamperman. Basado en AS2 “Arduino.as Actionscript Class v1.0” y que utiliza Serproxy. Es una implementación clara y efectiva. Adicionalmente se puede encontrar un tutorial en vídeo (que explica la versión revisada) en Vimeo. Incluye monitor de entradas y salidas configurables en Arduino con flash. También contiene fuentes, documentación y el servidor Serialproxy. Existen dos versiones: la versión revisada flash Arduino IO 2 y la versión anterior flash Arduino IO.

Existen algunas excepciones que utilizan la librería “Arduino.as Actionscript Class v1.0”, pero su implementación es muy específica para aplicaciones especiales que utilizan esta librería como base del proyecto. A continuación se describen tres de estas excepciones:

- Servidor serial SS6: por Dan O’Sullivan. Provee una librería simple “SS6 Flash Library” que se ha vuelto bastante utilizada. Además incluye código flash de muestra y código Arduino de muestra.
- Componente serial MIDAS: por Tek-Jin Nam & Ji-Dong Yim en el departamento de diseño industrial KAIST. Este provee componentes flash (trabaja con AS2.0 y AS3.0 en Windows y Mac) para realizar la conexión entre flash y Arduino mediante puertos seriales. Incluye además software de instalación y códigos flash de muestra.
- Herramientas NETLab: una colección de componentes flash (AS3) que funcionan con Arduino y otros sistemas mediante el concentrador NETLab Toolkit. Las herramientas pueden ser configuradas y utilizadas sin código (solo arrastrar y pegar), pero también tienen una interfaz de código para aplicaciones más sofisticadas. Estas herramientas pueden

trabajar en una configuración de flujo de datos, esto es, la salida de una herramienta puede servir como entrada de otra herramienta.

2.4.4. Código del usuario y código de muestra

Hay una extensa mayoría de manuales y tutoriales que se pueden consultar a la hora de escribir el código fuente para la interfaz entre flash y Arduino, pero generalmente, la mayoría de las personas optan por modificar ejemplos que contienen las instrucciones básicas como definir el puerto de la computadora a la que está conectada la placa, la velocidad de conexión, las funciones básicas para detectar cuando se presione una tecla e identificarla desde la aplicación de la computadora y enviar esa información mediante el código a la placa y procesarla adecuadamente.

El IDE (entorno de desarrollo integrado) de Arduino está escrito en Java y puede comunicarse con el puerto serial mediante la librería RXTX Java. Esta librería es bastante similar a la extensión API Java Communications. Internamente el IDE recuerda el puerto y la tasa de baudios que se utilizó la última vez. Desafortunadamente la implementación interna no puede ser considerada una API (interfaz de programación de aplicaciones) pública que pueda utilizarse confiablemente; por lo que el usuario deberá recordar el puerto COM que la placa Arduino está utilizando.

Esta interfaz de muestra presume que el usuario ya sabe cómo compilar y correr el código Java. Hay una variedad de recursos en Internet para aprender a programar en Java. Los programadores de Java más avanzados utilizan un IDE Java para crear los programas, por ejemplo, el IDE Eclipse.

2.5. Estructura del diseño de control

Para configurar el entorno de desarrollo y comenzar a realizar proyectos de Flash que se integren con la plataforma Arduino es necesario como se mencionó anteriormente: una placa Arduino, el software y los *drivers* para el sistema operativo correspondiente, el servidor (en este caso se utilizará TinkerProxy), un editor para escribir el código ActionScript (el cual se cargará en la placa Arduino), y por último un compilador flash / ActionScript.

2.5.1. Servidor TinkerProxy

TinkerProxy es una aplicación de proxy local que servirá como puente de comunicación entre flash y Arduino. Hay un número de proxies posibles para realizar este puente, pero debido a varias ventajas que se mencionan posteriormente se utilizará TinkerProxy. Este es un puerto de serproxy de código abierto y tiene un buen soporte para flash.

Se deben realizar algunas modificaciones al archivo de configuración, generalmente llamado “serproxy.cfg” (serproxy.exe en el entorno Windows) para ajustar algunos parámetros como el puerto al cual se conecta la placa Arduino y la velocidad a la cual se transferirán los datos.

Se necesita actualizar el comando serial_device (serial_device1=/dev/tty.usbserial-A6004osh) con el puerto serial al cual está conectada la placa Arduino. Esta actualización se puede realizar de dos formas. La forma más fácil de realizar dicha actualización es abrir el entorno de desarrollo de Arduino (asegurarse que la placa Arduino esté conectada), y luego seleccionar la opción Arduino > Tools > Serial Port.

También se puede correr el siguiente comando: ls -l / dev / y encontrar la información sobre el puerto serial conectado a Arduino en el resultado de dicho comando (será la entrada usbserial).

Luego de encontrar el puerto serial correcto, hay que actualizar el archivo de configuración con dicha información. Por ejemplo: serial_device1=/dev/cu.usbserial-A700dZgM.

A continuación, se debe especificar el puerto COM al que Arduino está asociado (en ambiente Windows, generalmente es el puerto COM 3) y cambiar la velocidad de conexión del puerto serial. Como primera opción se puede establecer en 9,600 baudios. Esta es la velocidad a la que Arduino y TinkerProxy se comunican (9,600 bytes por segundo). Se puede establecer esta velocidad a un valor más alto, pero es recomendable empezar con un valor bajo en orden de asegurar que si hay algún problema en la comunicación entre la placa y el servidor, estos no sean debido a la tasa de baudios.

Finalmente, se necesita actualizar el parámetro net_port. Para el valor de este parámetro, se elige el puerto que se desea que proxy escuche (se conecte). Por ejemplo: net_port1=5331.

A continuación, en la figura 10 se muestra un archivo de configuración típico “serproxy.osx.cfg” (entorno Mac).

Figura 10. **Configuración de archivo serproxy.osx.cfg**

```
# Config file for serproxy
# See serproxy's README file for documentation

# Transform newlines coming from the serial port into nils
# true (e.g. if using Flash) or false
newlines_to_nils=false

serial_device1=/dev/cu.usbserial-A700dZgM

comm_ports=1

comm_baud=9600
comm_databits=8
comm_stopbits=1
comm_parity=none

timeout=300

net_port1=5331
```

Fuente: Mike Chambers. http://www.mikechambers.com/blog/2010/08/04/getting-started-with-flash-and-arduino/. Consulta: 5 de agosto de 2014.

Luego de las modificaciones en el archivo de configuración, ya es posible correr TinkerProxy (la placa Arduino debe estar conectada a la computadora). Si no hay ningún problema, la salida que muestra el programa debería mostrar la información que se muestra en la figura 11.

Figura 11. **Resultado de conexión exitosa con TinkerProxy**

```
1  Serproxy 0.2.0 - Tinker.it
2  Based on code by (C)1999 Stefano Busti, (C)2005 David A. Mellis
3
4  Waiting for clients
```

Fuente: Mike Chambers. http://www.mikechambers.com/blog/2010/08/04/getting-started-with-flash-and-arduino/. Consulta: 5 de agosto de 2014.

De no mostrarse esta información, revisar nuevamente el archivo de configuración.

3. CONTROL ROBÓTICO POR MEDIO DE ARDUINO

3.1. Estructura básica de un robot

Un robot es una entidad virtual o mecánica artificial (puede referirse tanto a mecanismos físicos como a sistemas virtuales de software). La independencia creada en sus movimientos hace que sus acciones sean la razón de un estudio razonable y profundo en el área de la ingeniería. Puede estar formado por una estructura mecánica, sistema de accionamiento, sistema sensorial, sistema de control y elementos terminales.

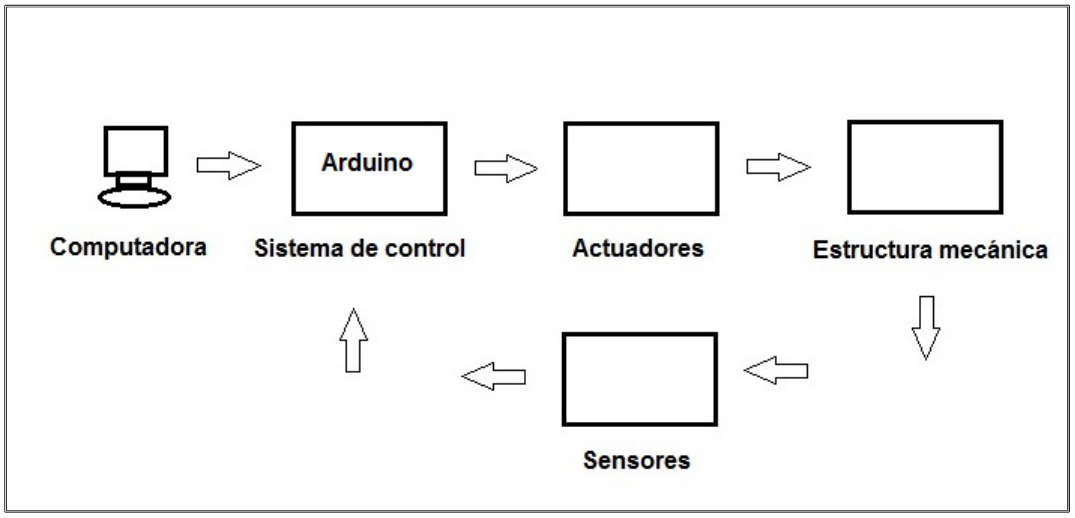
Mecánicamente, un robot está formado por una serie de elementos o eslabones unidos mediante articulaciones que permiten un movimiento relativo entre cada dos eslabones consecutivos. La constitución física de la mayor parte de los robots industriales guarda cierta similitud con la anatomía del brazo humano, por lo que en ocasiones, para hacer referencia a los distintos elementos que componen el robot, se usan términos como: cuerpo, brazo, codo y muñeca.

Los componentes básicos de un robot son:

- Estructura mecánica (los eslabones, base, actuadores, entre otros). Esto exige mucha masa, para proporcionar la suficiente rigidez estructural para asegurar la exactitud mínima bajo las cargas útiles variadas.
- Actuadores: los motores, los cilindros y articulaciones del robot. Esto también podría incluir los mecanismos para una transmisión, entre otros.

- Sistema de control encargado de transmitir la información que envía el usuario al circuito que realiza el movimiento o acción del robot.
- Sensores utilizados para la retroalimentación necesaria en caso de ser requerida por el funcionamiento específico para el cual está destinado el robot.

Figura 12. **Sistema básico de control utilizando Arduino**



Fuente: elaboración propia, empleando el programa Microsoft Paint 2007.

El movimiento de cada articulación puede ser de desplazamiento, de giro, o de una combinación de ambos. De este modo son posibles los seis tipos diferentes de articulaciones. Cada uno de los movimientos independientes que puede realizar cada articulación con respecto a la anterior, se denomina grado de libertad.

El número de grados de libertad del robot viene dado por la suma de los grados de libertad de las articulaciones que lo componen. Por ejemplo, si las articulaciones empleadas fueran de rotación y prismáticas solamente, con un

solo con grado de libertad cada una, el número de grados de libertad del robot suele coincidir con el número de articulaciones de que se compone, en este caso, 2 articulaciones – 2 grados de libertad.

El empleo de diferentes combinaciones de articulaciones en un robot, da lugar a diferentes configuraciones, con características a tener en cuenta tanto en el diseño y construcción del robot como en su aplicación. Dependiendo de la aplicación, por ejemplo, posicionar un objeto en un punto en el espacio la combinación de tres articulaciones es la más importante. Para posicionar y orientar un cuerpo de cualquier manera en el espacio son necesarios seis parámetros, tres para definir la posición y tres para la orientación, si se pretende que un robot posicione y oriente de cualquier modo en el espacio, se precisara al menos seis grados de libertad.

En la práctica, a pesar de ser necesarios los seis grados de libertad comentados para tener total libertad en el posicionado y orientación del extremo del robot, muchos robots industriales cuentan con solo cuatro o cinco grados de libertad, por ser estos suficientes para llevar a cabo las tareas que se encomiendan.

Existen también casos opuestos en los que se precisan más de seis grados de libertad para que el robot pueda tener acceso a todos los puntos de su entorno. Así, si se trabaja en un entorno con obstáculos, el dotar al robot de grados de libertad adicionales permite acceder a posiciones y orientaciones de su extremo a las que, como consecuencia de los obstáculos, no hubieran llegado con seis grados de libertad. Otra situación frecuente es dotar al robot de un grado de libertad adicional que le permita desplazarse a lo largo de un carril aumentando así el volumen de su espacio al que puede acceder. Cuando

el número de grados de libertad del robot es mayor que los necesarios para realizar una determinada tarea se dicen que el robot es redundante.

3.1.1. Condiciones básicas

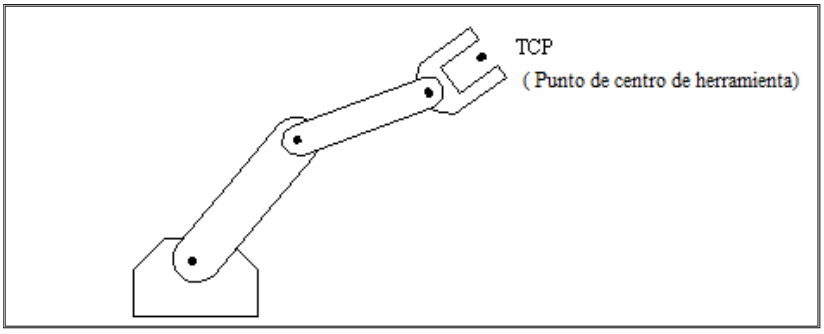
Los eslabones y articulaciones: los eslabones son los miembros estructurales sólidos de un robot, y las articulaciones son los acoplamientos movibles entre ellos.

El grado de libertad (gdl): cada articulación en el robot introduce un grado de libertad. Cada gdl puede ser un deslizador, un actuador rotatorio o de otro tipo. Los robots tienen 5 o 6 grados de libertad típicamente. 3 de los grados de libertad permiten el posicionamiento en tres dimensiones del espacio, mientras otros 2 o 3 se usan para la orientación del efector del extremo. 6 grados de libertad son bastante para permitir al robot alcanzar todas las posiciones y orientaciones en tres dimensiones del espacio. 5 gdl requiere una restricción a dos dimensiones en el espacio, el resto limita las orientaciones. Normalmente se usan 5 gdl por ocuparse de herramientas como los soldadores del arco.

La orientación eslabón: básicamente, si la herramienta se sostiene a una posición fija, la orientación determina qué dirección puede apuntarse. El rollo, diapasón y guiñada son los elementos de la orientación comunes usadas.

El punto de centro de herramienta (TCP por sus siglas en inglés) se localiza en el robot, y se usa al referirse a la posición del mismo, así como el punto focal de la herramienta. (Por ejemplo, el TCP podría estar en la punta de una antorcha de la soldadura) El TCP puede especificarse en el plano cartesiano, cilíndrico, esférico, coordenadas que dependen del robot.

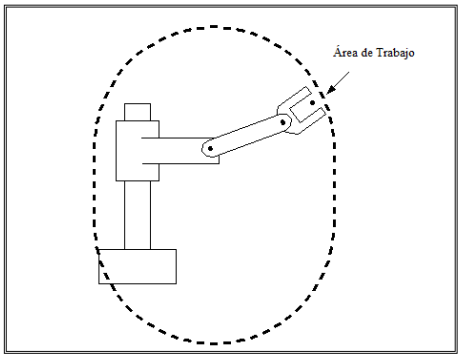
Figura 13. **Punto de centro de herramienta**



Fuente: Universidad de Guadalajara. *Libro de Robótica.* http://docentes.uto.edu.bo/gguzmanm/wp-content/uploads/LIBRO_DE_ROBOTICA.pdf. Consulta: 20 de marzo de 2015.

El robot tiende a tener una geometría fija, y limitada. El área de trabajo es el límite de posiciones en espacio que el robot puede alcanzar. Para un robot cartesiano (como una grúa arriba) el área de trabajo podría ser un cuadrado, para los robots más sofisticados esta área podría tener una forma esférica.

Figura 14. **Área de trabajo de un sistema robótico**



Fuente: Universidad de Guadalajara. *Libro de Robótica.* http://docentes.uto.edu.bo/gguzmanm/wp-content/uploads/LIBRO_DE_ROBOTICA.pdf. Consulta: 20 de marzo de 2015.

Al hablar de velocidad, se refiere a la máxima que es lograble por el TCP, o por las articulaciones individuales. Este número no es exacto en la mayoría de los robots, y variará encima del área de trabajo como la geometría del robot cambia (y de los efectos dinámicos). El número reflejará la velocidad más segura máxima posible. Algunos robots permiten la tasa máxima de velocidad para ser aprobado, pero debe tenerse un gran cuidado.

La carga útil indica la masa máxima que el robot puede alcanzar antes de cualquier inconveniente o pérdida dramática de exactitud. Es posible exceder la carga útil máxima, y que aún el robot funcione, pero esto no es aconsejable por la inestabilidad que ocasiona. Cuando el robot está acelerando rápidamente, la carga útil debe estar menos de la masa máxima. Esto es afectado por la habilidad de agarrar la parte firmemente, así como la estructura del robot, y el actuador. El extremo de brazo al laborar con herramienta debe ser considerado parte de la carga útil. La carga útil siempre se especifica como un valor máximo, esto puede estar antes del fracaso, o más normalmente, antes de la pérdida de la estabilidad.

Adicionalmente un sistema robótico está influenciado por varias consideraciones estáticas y dinámicas, entre las que podemos mencionar:

- Estáticas

    - Gravedad, la cual puede causar desviación descendente del brazo y sistemas de apoyo.
    - Repercusión negativa, cantidades notables de lentitud que causa errores en la posición
    - El trabajo de la articulación cuando se usan partes rotatorias largas en un sistema de esfuerzos se tuercen bajo la carga.

- La temperatura puede modificar las dimensiones en el manipulador.

- Dinámicas

  - Aceleración, la puede hacer que las fuerzas inerciales desvíen las partes estructurales. Estos aspectos normalmente pueden llegar a ser problemas cuando un robot tiene un movimiento muy preciso, limitado o cuando un camino continuo a seguir es esencial.
  - Torque, el cual se va produciendo cuando las partes rotatorias se van moviendo en forma circular y en cada instante producen una fuerza proporcional a la distancia del centro de la articulación.

3.2. Estructura de interfaces en placa Arduino para control robótico

La conexión de la placa Arduino con el sistema de control robótico que incluye los motores y sensores, entre otros, se realiza por medio de la terminal PWM para poder optimizar el tiempo en que se encuentran encendidos los componentes externos y de esta forma controlar la potencia utilizada.

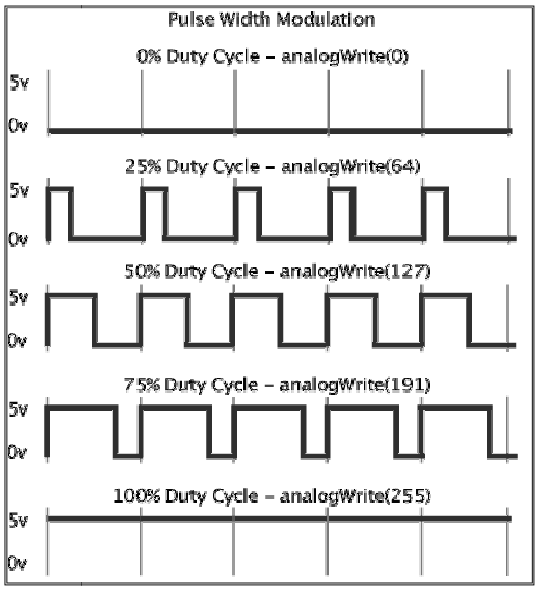
3.2.1. PWM

La modulación por ancho de pulso (PWM = Pulse Width Modulation) es una técnica para simular una salida analógica con una salida digital. El control digital se usa para crear una onda cuadrada, una señal que conmuta constantemente entre encendido y apagado. Este patrón de encendido-apagado puede simular voltajes entre 0 (apagado) y 5 voltios (encendido) simplemente variando la proporción de tiempo entre un valor y otro. A la duración del tiempo de encendido (ON) se le llama ancho de pulso (*pulse*

width). Para variar el valor analógico se cambia o modula ese ancho de pulso. Si se repite este patrón de encendido-apagado lo suficientemente rápido, por ejemplo, con un led el resultado es como si la señal variara entre 0 y 5 voltios controlando el brillo del led.

En la figura 15, las líneas representan un periodo regular. Esta duración o periodo es la inversa de la frecuencia del PWM. En otras palabras, con la placa de Arduino la frecuencia PWM puede llegar a ser bastante próxima a 500 Hz, lo que equivale a periodos de 2 milisegundos cada uno. La llamada a la función analogWrite debe ser en la escala desde 0 a 255, siendo 255 el 100 % del ciclo (siempre encendido), el valor 127 será el 50 % del ciclo (la mitad del tiempo encendido) y el valor 0 será el 0 % del ciclo.

Figura 15. **Esquema de modulación de ancho de pulso, PWM**



Fuente: Universidad de Guadalajara. *Libro de Robótica.* http://docentes.uto.edu.bo/gguzmanm/wp-content/uploads/LIBRO_DE_ROBOTICA.pdf. Consulta: 20 de marzo de 2015.

3.2.2. Función analogWrite

Escribe un valor analógico (PWM) en un pin. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor. Después de llamar a la función analogWrite, el pin generará una onda cuadrada estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función analogWrite (o una llamada a las funciones digitalRead o digitalWrite en el mismo pin). La frecuencia de la señal PWM será de aproximadamente 490 Hz.

En la mayoría de las placas Arduino (aquellas con el ATmega168 o ATmega328), se podrá generar señales PWM en los pines 3, 5, 6, 9, 10, y 11. En la placa Arduino Mega, se puede llevar a cabo con los pines desde el 2 hasta el pin 13. Las placas Arduino más antiguas que poseen el microcontrolador ATmega8 solo pueden utilizar la función analogWrite con los pines 9, 10 y 11 y no hace faltar configurar el pin como salida.

La función analogWrite no tiene ninguna relación con los pines de entrada analógicos ni con la función analogRead, su sintaxis es la siguiente: analogWrite(pin, valor).

En la figura 16, se muestra un ejemplo de un: pulso en el pin 6 cuyo ciclo de trabajo (PWM) será proporcional a la tensión leída en otro pin. Para fines de muestra se supone un potenciómetro que se encargue de leer la señal en el pin 3 (simulando una entrada análoga de un sensor).

Figura 16. **Programa utilizando PWM**

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

int var1 = 6;   //Salida digital en pin 6 donde se puede conectar el LED para
      // ver la señal PWM
int var2 =  3;  // Entrada análoga en pin 3 simulando una señal de un sensor
int var3 = 0; // Variable que guarda el valor del dato leído en el pin 3

void setup ( )        // Función para definir pin 6 como salida
{
  pinMode (var1, OUTPUT) ;  // Declarar pin 6 como salida
}
void ejpwm        // Función para leer valor pin 3 y guardarlo
            // en la variable var3
{
var3 = analogRead ( var2 ) ;    // Lee el valor en pin 3 y lo guarda en var3
analogWrite ( var1, var3 / 4 ) ;  // Debido a que el rango de analogWrite es
// 0 a 255 y de analogRead es 0 a 1023
// se realiza el ajuste al dividir dentro de 4
}
```

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

Las señales PWM generadas en los pines 5 y 6 poseerán ciclos de trabajo superiores a lo esperado. Esto es así, porque para esos dos pines se utiliza el mismo temporizador que se utiliza en las funciones millis() y delay(). Este efecto se notará mucho más en ciclos de trabajo bajos (por ejemplo, de 0 a 10), y puede ser que, aunque se configuren esos pines con una señal de ciclo de trabajo cero no llegue a ser verdaderamente 0.

3.3. Aplicación de sistema Arduino para control de un robot

Los motores son parte esencial para el funcionamiento de un sistema robótico, ya sean motores de corriente directa, servomotores o de algún otro tipo. Es importante considerar el sistema de control que se utilizará para la conexión entre la placa Arduino y los motores, debido a que los requerimientos de corriente son bastante elevados y el microcontrolador de la placa Arduino no es capaz de suministrar la corriente para este tipo de dispositivos. Por tal motivo es necesario utilizar un controlador que suministre la corriente necesaria y también se pueda comunicar con el controlador por medio de señales digitales.

3.3.1. Control de motores

La placa Arduino Uno no puede gestionar directamente motores de corriente continua, dado que la corriente máxima que es capaz de proporcionar en sus pines de salida es de unos escasos 20 mA. Por lo tanto, se necesita un controlador de motores que sea capaz de soportar la carga de los motores. Dicho controlador será gestionado a su vez, por la placa Arduino. El controlador debe soportar la potencia suficiente para los motores que se utilizarán.

Una buena opción es el controlador en doble puente H basado en el conocido driver L298.

3.3.2. Alimentación

Se debe tomar en cuenta la alimentación para el controlador de potencia y demás dispositivos que se conectarán a la placa Arduino debido a la cantidad de potencia (corriente) que consumen generalmente estos dispositivos.

3.3.3. Conexiones físicas

Las conexiones físicas son muy importantes, tanto si se arma el proyecto en protoboard o en placa impresa. El orden y la simplicidad de las conexiones son claves para el posterior análisis a la hora de algún inconveniente y resolver problemas o para que en el futuro no sea tan complicado poder sacar un diagrama o esquema de dicho proyecto.

3.3.4. Sensores

Son dispositivos encargados de obtener retroalimentación del equipo que se requiera (el robot en este caso); esta información es de gran utilidad para poder tener un mejor control del comportamiento del sistema. La retroalimentación puede estar dada en un valor digital o análogo dependiendo del tipo de sensor y las necesidades que se tengan. A continuación, se mencionan algunos tipos de sensores de proximidad los cuales son comúnmente utilizados en sistemas de robots.

3.3.4.1. Sensores infrarrojos

Son dispositivos electrónicos capaces de medir la radiación electromagnética infrarroja de los cuerpos en su campo de visión. Todos los cuerpos reflejan una cierta cantidad de radiación, esta resulta invisible para la vista del hombre, ya que se encuentran en el rango del espectro justo por debajo de la luz visible. Existe una variedad de sensores infrarrojos, por ejemplo se pueden clasificar según el tipo de señal emitida.

- Sensores reflexivos: presentan una cara frontal en la que se encuentran tanto el led como el fototransistor. En esta configuración el sistema mide

la radiación que proviene del reflejo de luz emitida por el led. Esta configuración es sensible a la luz del ambiente, lo cual puede llegar a influir en medidas inexactas, por lo cual es recomendable utilizar filtros de longitud de onda y, así mismo trabajar en ambientes de luz controlada. Un último aspecto a considerar es el coeficiente de reflectividad del objeto.

- Sensores modulados: tienen el mismo principio que los sensores reflexivos mencionados anteriormente. La diferencia radica en que lo sensores modulados emiten una señal modulada, reduciendo en gran medida la influencia de la iluminación ambiental. Típicamente estos sensores son utilizados en la detección de presencia de objetos, en la medición de distancias, en la detección de obstáculos aún teniendo cierta variación en la iluminación del trayecto hacia dicho obstáculo.

- Sensores de barrido: este tipo de sensor realiza un barrido horizontal de la superficie reflectante mediante la emisión de señales moduladas, lo cual permite mejorar la exactitud independientemente de la luz incidente, el color o la reflectividad de los objetos. Típicamente estos sensores se colocan de tal forma que sean parte de un dispositivo de desplazamiento perpendicular al eje de exploración del sensor para poder cubrir toda el área necesaria.

3.3.4.2. Sensores capacitivos

La función de estos sensores consiste en indicar un cambio de estado basado en la variación de la intensidad de un campo eléctrico. Estos sensores pueden detectar objetos metálicos o no metálicos midiendo el cambio en la capacitancia, la cual depende de la constante dieléctrica del material a detectar,

la masa, el tamaño y la distancia hasta la superficie sensible del detector. Los detectores capacitivos consisten de un oscilador RC el cual dependiendo del cambio de capacitancia debido a la influencia del objeto a detectar hace que el sistema entre en oscilación. Para poder calibrar este punto de oscilación se utiliza un potenciómetro el cual controla la realimentación del oscilador y, por lo tanto, la distancia de actuación en algunos materiales puede llegar a regularse por medio de dicho potenciómetro. Luego, la señal de salida de este oscilador alimenta un amplificador el cual a su vez lleva la señal a la etapa de salida.

Cuando un material conductor se acerca a la cara activa del detector, el objeto actúa como un capacitor. Cabe mencionar que el cambio de la capacitancia es significativo en una distancia grande. Por otro lado, si se aproxima un material no conductor, se produce una pequeña variación en la constante dieléctrica y, por lo tanto el incremento de la capacitancia es muy pequeño comparado con el cambio que produciría un material conductor.

Existen normativas para los sensores capacitivos referentes a los tipos o grados de protección (NEMA y DIN). Para los normas NEMA se pueden mencionar el tipo1 (propósito general), el tipo2 (hermético a gotas), el tipo 3 (resistencia a la intemperie, instalación en el exterior) entre otros. La norma DIN establece el grado de protección IP que se compone de dos dígitos (el primer dígito indica la protección contra sólidos y el segundo dígito indica la protección con el agua)

3.3.4.3. Sensores ultrasónicos

Se utilizan como detectores de proximidad de objetos a distancias establecidas por sus características específicas. Básicamente el funcionamiento de estos sensores es emitir un sonido y medir el tiempo que la señal (sonido)

emitida tarda en regresar al sensor. El sensor envía el sonido el cuál se refleja (una parte) en el objeto y esta parte reflejada regresa hacia el sensor (eco producido) y posteriormente se transforme en una señal eléctrica la cual estará directamente relacionada con el tiempo que se tarde la señal en volver al sensor como se mencionó anteriormente. Estos sensores trabajan solamente en el aire, y son capaces de detectar objetos de diferentes formas, colores, superficies y materiales (sólidos y líquidos) siempre que sean deflectores de sonido.

Una ventaja de este tipo de sensores es que no necesita contacto físico con el objeto, tiene la posibilidad de detectar objetos frágiles como por ejemplo pintura fresca o cualquier material (sin importar el color) al mismo alcance sin ajustar algún factor de corrección. Algunos sensores pueden llegar a tener una precisión de 6 mm. Por otro lado, una desventaja de estos sensores son las falsas alarmas y las posibles zonas ciegas (zona comprendida entre el lado sensible del detector y el alcance mínimo en el que ningún objeto puede detectarse de una forma certera) que pueda tener dicho sensor.

3.4. Análisis por medio de simulador del control robótico

Un simulador robótico o simulador de robótica es utilizado para crear aplicaciones integradas para un robot sin depender físicamente de la máquina real. Con esto se pueden ahorrar costos y tiempo. En algunos casos, estas aplicaciones pueden ser cargadas en el sistema del robot real sin modificaciones. Existen varios tipos aplicaciones de simuladores, desde aplicaciones que permiten a los robots interactuar con un mundo de objetos creados por el programador, como también aplicaciones binarias o computacionales las cuales tiene preestablecidas rutinas y funciones

determinadas y donde a diferencia de la anterior los robots no tienen la libertad de acoplarse a diferentes escenarios o aprender de los errores.

4. PROGRAMACIÓN

El hardware de la placa Arduino por sí solo no puede realizar las funciones requeridas por la aplicación, tampoco tiene la habilidad de tomar decisiones o mostrar datos de retroalimentación. Todas estas opciones se realizan por medio del programa (software) que se corre en el microcontrolador. El lenguaje de programación utilizado en las placas Arduino se llama Processing y está basado en el lenguaje C.

4.1. Estructura de programación por medio de Arduino

Los microcontroladores dependen de un cliente en una computadora para desarrollar y compilar programas. El software utilizado en el cliente de la computadora es conocido como: entorno de desarrollo integrado o IDE, por sus siglas en inglés. El lenguaje Arduino como se ha explicado en los capítulos anteriores está basado en el lenguaje C y el IDE Arduino como varios entornos de programación provee cierta ayuda a la hora de cometer errores en la programación.

El software de programación de Arduino (IDE) se puede descargar desde la página web oficial (www.arduino.cc) y es compatible con las plataformas PC, Mac y Linux. En dicha página existe una sección de instrucciones paso a paso para poder descargar e instalar el programa en cualquier plataforma disponible. Luego de que se complete la instalación, se puede proceder a conectar la placa Arduino a la computadora por medio de un cable USB. Como todo dispositivo USB que se conecta a una PC por primera vez, hay que instalar los drivers

correspondientes, como se detalla en la guía que se encuentra en el sitio web oficial de Arduino.

El entorno de programación de Arduino es simple. Al conectar una placa Arduino por primera vez, se requiere especificar el tipo de placa que se está utilizando y el puerto serial al cual está conectada en la computadora.

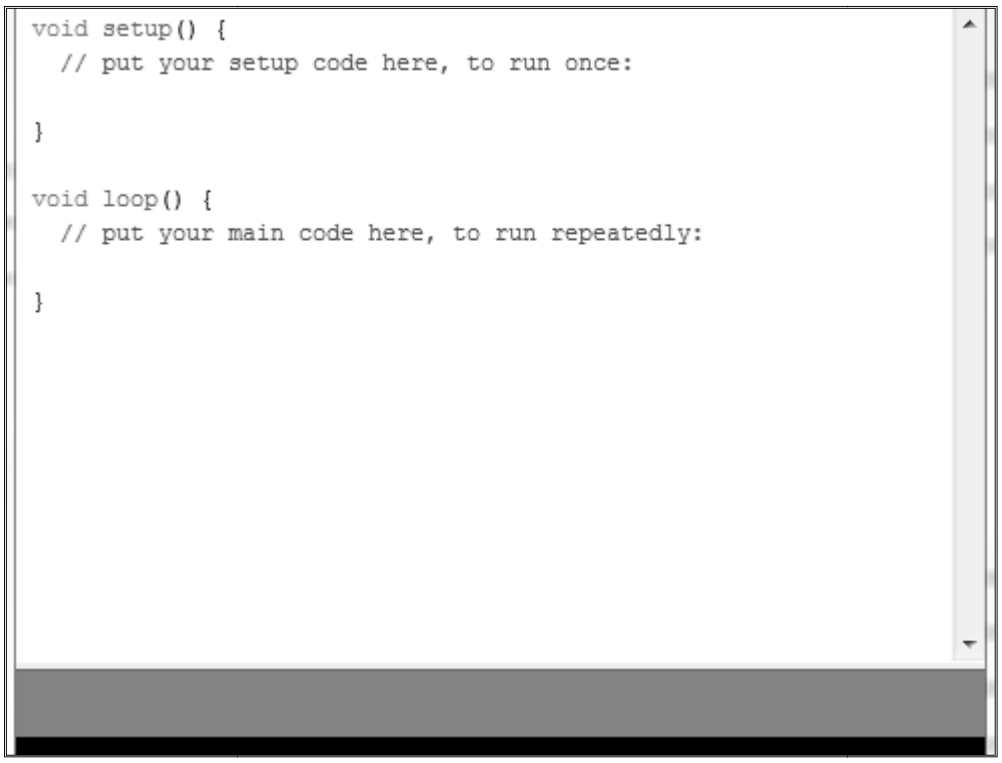
Luego de escribir el código de un programa o de cargar un código existente, se debe compilar el programa, lo cual es llamado Esquema en terminología de ambiente Arduino. Esto prepara el código para descargarlo en la placa Arduino y el cual se ejecutará una vez que la transferencia hacia la placa se haya completado.

Como se mencionó en capítulos anteriores, Arduino soporta el uso de librerías, los cuales son repositorios de código que extienden la funcionalidad de la programación. Las librerías permiten reutilizar código sin tener que escribirlo o copiarlo en cada uno de los programas. En la instalación básica de Arduino se encuentran varias librerías de propósito general. En adición a estas librerías se pueden descargar otras desde las páginas web de soporte de Arduino y de páginas de terceros que publican código para librerías Arduino.

Un buen ejemplo de una librería que se utilizará en la mayoría de programas destinados para el control de robots es la librería Servo. Esta librería permite conectar uno o más servos a los pines digitales de entrada/salida de la placa Arduino. La librería Servo viene incluida en la instalación estándar del IDE Arduino, por lo que añadirla al esquema es tan simple como seleccionar la opción esquema > importar librería > servo en el IDE Arduino.

A nivel de estructura, los esquemas de Arduino son lineales y bastante fáciles de leer e interpretar. Todos los esquemas tienen al menos dos partes, llamadas setup y loop. Estas son llamadas funciones y tienen la siguiente forma:

Figura 17. **Sintaxis funciones Setup y Loop**



Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

Los paréntesis ( ) son utilizados en caso se requieran argumentos opcionales en la función (datos a ser utilizados por esa función específicamente). En el caso de las funciones setup y loop, no se utilizan argumentos pero los paréntesis son necesarios por la estructura del entorno de programación.

Las llaves { } definen el principio y fin de la función. El código contenido dentro de las llaves conforma el bloque de código. Aún en el caso de que no haya código dentro de las llaves, son necesarias por la sintaxis del entorno de programación.

La palabra *void* que precede ambas funciones se utilizar para indicar al compilador que la función no retornará ningún valor después de que se procese.

Las funciones Setup y Loop siempre son requeridas en cualquier esquema de programación Arduino que se realice, en caso contrario el IDE reportará un error al compilar el esquema y no se podrá procesar. Estas funciones tienen el objetivo de definir el hardware Arduino como por ejemplo los puertos a utilizar (función Setup) y repetir infinitamente su contenido media vez la placa Arduino esté operando (función Loop).

Adicionalmente muchos esquemas Arduino tienen una sección de declaración global al inicio del esquema, entre otras cosas para poder definir variables, constantes que se utilizarán en todo el programa y para indicarle al IDE las librerías externas que se desean utilizar, por ejemplo la librería Servo.

Arduino utiliza variables para guardar información mientras el esquema se está ejecutando. La plataforma soporta varias clases de datos, de diferente tipo y tamaño. Entre las más comunes se encuentran: *integer* (número entero en el rango de -32,768 a 32,767), *unsigned integer* (número entero positivo en el rango de 0 a 65,535), byte (número entero positivo en el rango de 0 a 255), *boolean* (valor de falso o verdadero), *float* (número con punto decimal, con hasta 15 decimales), *string* (cadena de texto).

Para utilizar una variable, primero es necesario declararla. Esto se puede realizar al inicio del esquema o en cualquier punto dentro del esquema, la diferencia entre declarar una variable global (al inicio del esquema) o en otro punto dentro del esquema es el uso que se le puede dar a dicha variable, ya sea en cualquier parte del esquema si es una variable global (declara al inicio del esquema) o solamente en la función dentro la cual se declaró si es una variable local (declarada en otro punto dentro del esquema).

Para declarar una variable, al igual que en otros entornos de programación, se requiere colocar el tipo de variable al inicio, seguida del nombre. Opcionalmente se puede colocar el signo = seguido del valor inicial que se le quiera asignar a dicha variable. Los nombres de variables deben contener solamente letras del alfabeto, números enteros positivos o el carácter guión bajo "_". El nombre no puede empezar con un número y no puede contener espacios. Por último el programa distingue entre mayúsculas y minúsculas en los nombres de las variables por lo cual "var1" es diferente a "Var1".

Las terminales o pines de la placa Arduino están referenciadas por número. Existen dos secuencias: una para los pines análogos y otra para los pines digitales. Los pines análogos son referenciados como Ax (donde x es el número de pin). Los pines digitales son referenciados en los esquemas solo por números, aunque también puede utilizar Dx (donde x es el número de pin) para evitar confusiones con los pines análogos.

Los pines digitales son considerados automáticamente como entradas por defecto, lo que significa que están configurados para leer un valor, en vez de establecer un valor. Para cambiar el modo de un pin digital se utiliza la función pinMode (PinNumber, Direction), donde PinNumber es el número de pin que se

utilizará y Direction es el modo, ya sea OUTPUT o INPUT (se utilizan mayúsculas debido a que estas palabras están definidas como constantes, y por convención las constantes se escriben todas mayúsculas para diferenciarlas de las variables).

Figura 18. **Ejemplo básico para controlar un servomotor**

```
#include <Servo.h>      // Utilizar la librería Servo

Servo servo1; // Se crea un objeto de tipo Servo para controlar // el servomotor

void setup ( )
{
servo1.attach(10) // Se indica que el servomotor está conectado al //pin 10
}

void loop ( )
{
servo1.write(0);  // Se utiliza para mover el servomotor a la
// posición 0
  delay(500);     // Tiempo de retardo para ejecutar próxima
// instrucción
  servo1.write(180);    // Se utiliza para mover el servomotor a la
          // posición 180
  delay(500);     // Tiempo de retardo para ejecutar próxima
// instrucción
}
```

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

La primera línea, #include <Servo.h> indica al IDE que se utilizará la librería Servo, la cual es el nombre de una clase. Con una clase se pueden crear múltiples copias o instancias de un objeto sin tener que duplicar código de programación. Tomar en cuenta que Servo es el nombre de la clase que se utiliza y servo1 es el nombre del objeto el cual se creó.

Al igual que en otros lenguajes de programación, Arduino tiene la facilidad de crear funciones definidas por el usuario para poder reutilizar código. La estructura es igual que la que se mencionó para las funciones *setup* y *loop*. Para el ejemplo anterior se puede crear una función para mover el servomotor a la posición 0 y así utilizarla cada vez que se quiera realizar dicho movimiento sin tener que volver a escribir las líneas de código. En la figura 19 se muestra la sintaxis para esta función.

Figura 19. **Sintaxis de función para mover un servomotor**

```
void posicion0 ( )
{
  servo1.write(0);        //Mover servomotor a posición 0
  delay(500);          //Retardo de 500 ms)
}
```

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

Para utilizar una función solo se necesita escribir el nombre de la función seguida de paréntesis con los parámetros opcionales, si los hubiera y por último ";". Para llamar a la función "posicion0", la sintaxis es la siguiente: posicion0 (0); .

4.1.1. Estructuras de control de flujo

Las estructuras de control de flujo son utilizadas para indicarle al esquema qué condiciones se deben de cumplir para poder realizar una instrucción o tarea. La estructura de control de flujo más ampliamente utilizada es el comando *if*. Este comando realiza una validación de si existe o no una condición y de acuerdo al resultado de la validación ejecuta o no las instrucciones del programa.

En la figura 20 se muestra la sintaxis de la función If:

Figura 20. **Sintaxis función If**

```
if (condición)
  {
            // Valor verdadero de la validación: se cumple la condición
  }
else
  {
           // Valor falso de la condición: no se cumple la condición
  }
```

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

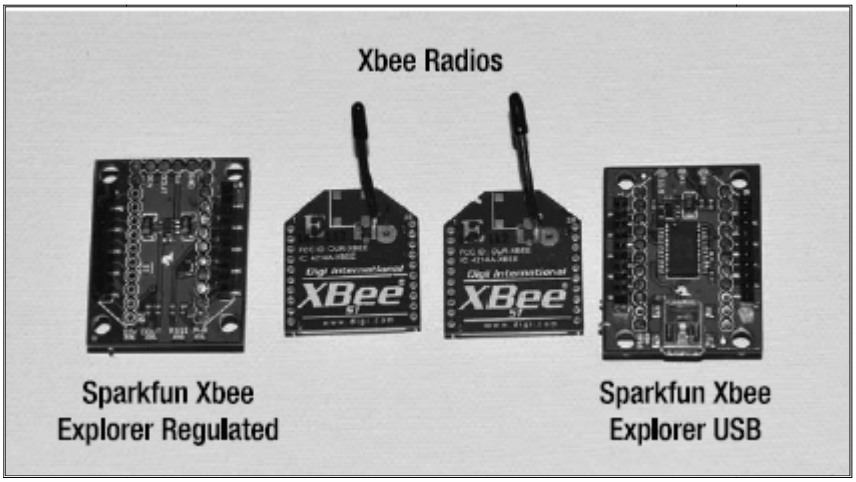
4.2. Sintaxis y ejemplo de control, video y audio

Según se ha descrito en las páginas anteriores, dependiendo de la utilidad del proyecto puede ser necesario o no que el robot sea completamente autónomo en una o varias tareas específicas, o que se limite a realizar estrictamente lo que el usuario le ordene mediante instrucciones en el programa (esquema). Por ejemplo, para el caso de este trabajo se requerirá poder mostrar un robot que pueda seguir instrucciones de movimiento y mostrar una retroalimentación por medio de una cámara que se activará a la placa de Arduino.

Para la comunicación de la cámara de vídeo hacia la placa de Arduino y con fines de simplificación en el diseño se utilizará un protocolo llamado Xbee, el cual es utilizado para crear una conexión serial entre la placa Arduino y el sistema de control (en este caso una PC). El funcionamiento detallado de Xbee supera los alcances de este trabajo, pero en pocas palabras Xbee es un

transmisor/receptor a una frecuencia de 2.4 GHz que utiliza dos radios, uno conectado al sistema de control y otro conectado a la placa Arduino que controla los movimientos del robot. Al utilizar la configuración del transmisor Xbee de dos radios, se puede simplificar para efectos de programación como que la placa Arduino estuviera directamente conectada al sistema de control (computadora personal). Se pueden enviar y recibir valores utilizando la conexión serial del computador o de otro dispositivo previamente programado.

Figura 21. **Placa de módulos Xbee**



Fuente: Sparkfun Electronics. https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide. Consulta: 10 de abril de 2015.

4.2.1. Configuración de Xbee

Existe una gran variedad de radios Xbee, entre estos los de Serie 1, 2.5 y 900 MHz (todos estos ofrecen velocidades elevadas de transmisión de datos seriales, mayores a 115,200 bps) con módulos de consumo de potencia altos y bajos según sea el uso que se requiera.

Para este caso se utilizará en el diseño un módulo Xbee Serie 1 de bajo consumo (modelo Sparkfun WRL-08665), el cual tiene todas las funciones que se requieren y lo único que hay que realizar es conectar el radio Xbee en la placa del adaptador Sparkfun.

Como se muestra en la figura 21, lo que se necesita para utilizar los módulos Xbee son: dos radios Xbee, la placa Xbee Explorer Regulated para la conexión con la placa Arduino, la placa Xbee Explorer USB para la conexión con el sistema de control, el software de programación XCTU Xbee (que se puede descargar gratuitamente de la página www.digi.com) y un cable mini USB.

Al utilizar los módulos serie 1 para transmisión de datos simple, no es necesario reprogramar dichos módulos debido a que por defecto traen la configuración de la velocidad de transferencia de datos serial a 9600 bps. Si se requiere utilizar una velocidad diferente o cambiar la configuración de los radios Xbee para operar en un canal privado, se necesitará realizarlo por medio del software de programación XCTU que se mencionó anteriormente.

4.2.2. Comunicación serial utilizando Xbee

La sintaxis para utilizar los módulos Xbee se muestra en la figura 22. Se debe tomar en cuenta que la lógica del programa para la comunicación serial con el módulo Xbee no cambia con respecto a la que se utilizaría con cualquier otro módulo lo cual es una gran ventaja en cuanto a pureza y simplicidad en el código del programa.

Figura 22. **Programa de comunicación serial utilizando módulo Xbee**

```
int entrada_serial = 0;       //variable del byte a leer del puerto serial
void setup() {
Serial.begin(9600);         //configuración de puerto serial y velocidad
pinMode(13, OUTPUT);
}
void loop() {
if (Serial.available() > 0) {     //enviar datos solo cuando se reciba algo
entrada_serial = Serial.read();   //leer el byte de entrada del puerto serial
Serial.println(entrada_serial, BYTE); //imprimir byte recibido en puerto serial
digitalWrite(13, HIGH);       //encender LED en pin 13
delay(1000);          //retard 1 segundo
}
else {
digitalWrite(13, LOW);      //apagar LED en pin 13
}
```

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

La placa Xbee Explorer Regulated que se conecta a la placa Arduino debe de conectarse del pin DOUT de dicha placada hacia el pin DP0 (RX) de la placa Arduino y el pin DIN de la placa Xbee Explorer Regulated hacia el pin DP1 (TX) de la placa Arduino. En caso de utilizar una placa Arduino que soporte programación FTDI para conversión de señales seriales a USB, basta con conectar la placa Xbee Explorer en el puerto FTDI de la placa Arduino.

4.3. Prototipo del programa

El código que se presenta a continuación tiene como objetivo mostrar cómo se puede utilizar una placa Arduino en conjunto con el hardware de un robot, para realizar el control de dicho robot mediante un mando remoto (un control de *joystick* para este programa), utilizando los módulos de radios Xbee.

El uso de estos módulos de radios Xbee se debe a la flexibilidad que tienen para conectarse a la computadora y enviar instrucciones y tener control de la placa Arduino a distancia y desde la computadora, según los requerimientos, lo cual le da un nuevo alcance a las funciones y objetivos que puedan realizarse con el robot.

Figura 23. **Programa para control de robot utilizando Arduino**

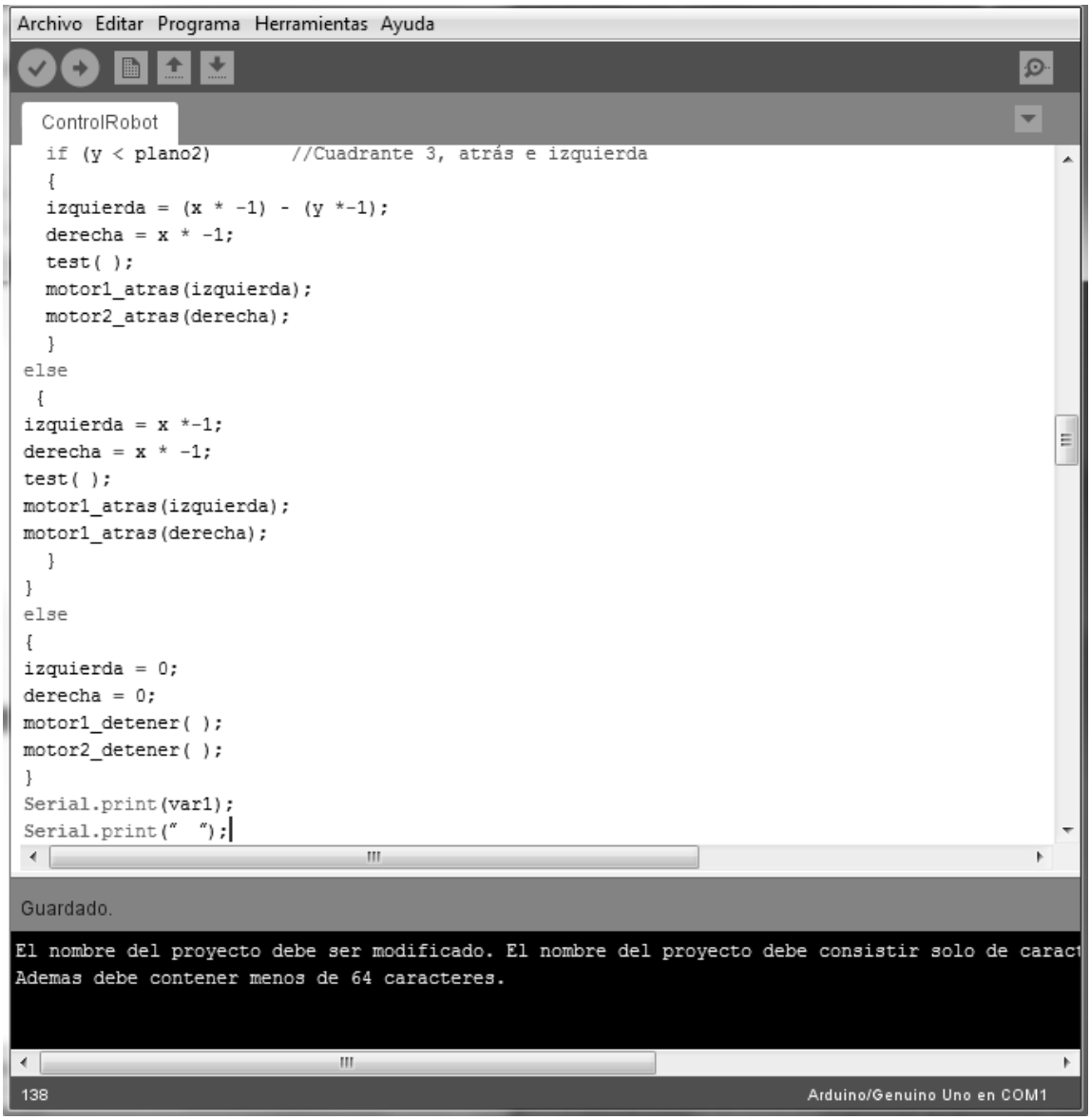
Continuación de la figura 23.

Continuación de la figura 23.

Continuación de la figura 23.

Continuación de la figura 23.

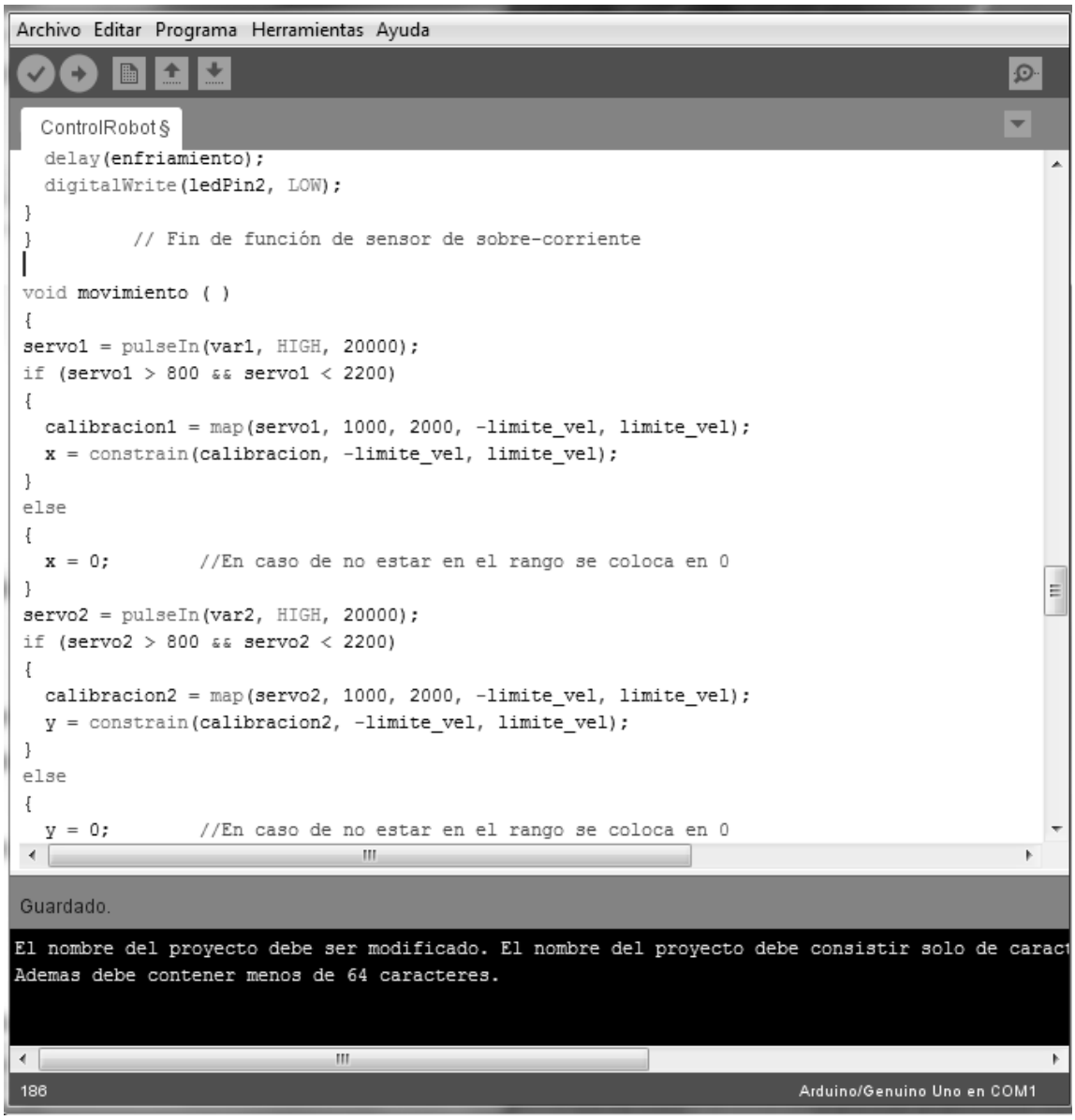
Continuación de la figura 23.



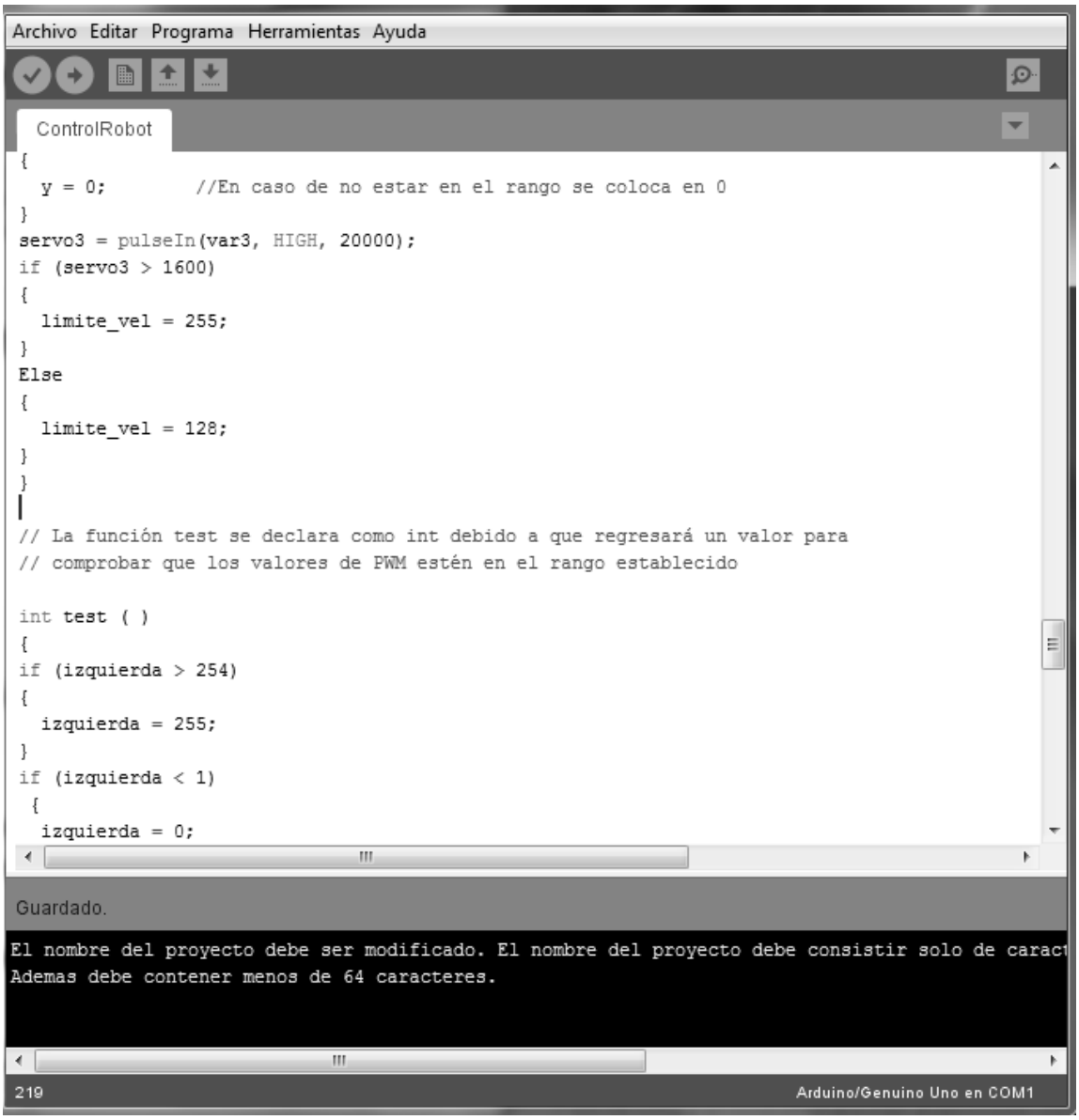
Continuación de la figura 23.

Continuación de la figura 23.

Continuación de la figura 23.



Continuación de la figura 23.



Continuación de la figura 23.

Continuación de la figura 23.

Continuación de la figura 23.

Fuente: elaboración propia, empleando el programa Arduino IDE 1.6.6.

CONCLUSIONES

1. La diferencia entre un software gratis y uno libre es muy importante debido a los costos y escalabilidad a la hora de utilizar un código de programación y se deben respetar las normas establecidas por la licencia GNU GPL.

2. Arduino es una plataforma de hardware libre, lo que la hace tener una gran ventaja en cuanto a la flexibilidad del diseño de acuerdo a las necesidades específicas de la aplicación.

3. La carga útil en el diseño de un sistema robótico es de suma importancia, ya que define el límite de masa en el cual el sistema puede seguir funcionando con la precisión y estabilidad necesarios para cumplir las funciones de forma adecuada.

4. Se utilizaron esquemas básicos de circuitería para motores y sensores con el objetivo de mostrar la facilidad y compatibilidad que ofrecen las placas Arduino, para conectarse con componentes externos.

5. El código de programación del sistema de control del robot que se utilizó tiene la flexibilidad de poder ser complementado con la interconexión a una computadora para obtener funciones adicionales, por ejemplo el control desde un servidor remoto.

RECOMENDACIONES

1. Motivar a las personas en general, y en especial a los estudiantes de Ingeniería Electrónica a investigar el desarrollo de proyectos utilizando la plataforma Arduino, la cual ofrece muchas ventajas en cuanto a documentación, flexibilidad y escalabilidad comparado con otras plataformas de hardware.

2. Elaborar proyectos utilizando Arduino, aprovechando que es una placa de bajo costo y existe una gran variedad de versiones de hardware. Incluso se pueden realizar personalizaciones para crear una placa Arduino específica según la aplicación que se requiera.

3. Reforzar e incluir más prácticas de laboratorio utilizando Arduino y otras plataformas de código abierto y hardware libre en los cursos de Electrónica de la Facultad de Ingeniería, y apoyar la actualización de aplicaciones y proyectos que sean factibles para los estudiantes a nivel económico y académico.

4. Fomentar la constante actualización de nuevas tecnologías de plataformas de código abierto y hardware libre, que ayuden el desarrollo de una unidad de investigación que tenga la finalidad de realizar propuestas de proyectos innovadores y de bajo costo que puedan ser utilizadas comercialmente para el beneficio de la sociedad.

BIBLIOGRAFÍA

1. Arduino. *AVR Code.* [en línea]. <http://www.arduino.cc/playground/Main/AVR>. [Consulta: 3 de agosto de 2014].

2. _____.*Board Setup and Configuration.* [en línea]. <http://playground.arduino.cc/Main/ArduinoCoreHardware>. [Consulta: 10 de junio de 2014].

3. _____.*EEPROM Library.* [en línea]. <http://arduino.cc/en/Reference/EEPROM>. [Consulta: 10 de junio de 2014].

4. _____.*Ethernet Library.* [en línea]. <http://arduino.cc/en/Reference/Ethernet>. [Consulta: 15 de junio de 2014].

5. _____.*Interfacing Arduino to other languages.* [en línea]. <http://www.arduino.cc/playground/Main/InterfacingWithSoftware>. [Consulta: 5 de agosto de 2014].

6. _____.*Manipulación de puertos.* [en línea]. <http://arduino.cc/en/Reference/PortManipulation>. [Consulta: 10 de junio de 2014].

7. _____.*Processing Language Comparison.* [en línea]. <http://arduino.cc/en/Reference/Comparison?from=Main.ComparisonProcessing>. [Consulta: 5 de agosto de 2014].

8. _____.*Serial.* [en línea]. <http://arduino.cc/en/Reference/Serial>. [Consulta: 15 de junio de 2014].

9. BLUM, Jeremy. *Exploring Arduino: Tools and Techniques for Engineering Wizardry.* 1a ed. John Wiley & Sons Inc., 2013. 384 p.

10. FRY, Ben. *Processing Reference.* [en línea]. <https://processing.org/reference/>. [Consulta: 10 de junio de 2014].

11. KOCH, Martin. *Arduino controlled video recording using the LANC port*. [en línea]. <http://controlyourcamera.blogspot.com/2011/02/arduino-controlled-video-recording-over.html>. [Consulta: 10 de junio de 2014].

12. MARGOLIS, Michael. *Make an Arduino-Controlled Robot*. Maker Media Inc., 2012. 256 p.

13. MCCOMB, Gordon. *Robot Builder's Bonanza.* 4a ed. McGraw-Hill, 2011. 685 p.

14. MCROBERTS, Michael. *Beginning Arduino.* 1a ed. Apress, 2010. 472 p.

15. MONK, Simon. *Programming Arduino Getting Started with Sketches.* 1a ed. McGraw-Hill Education TAB, 2011. 176 p.

16. OXER, Jonathan. *Practical Arduino: Cool Projects for Open Source Hardware (Technology in Action)* 2010 ed. Apress, 2009. 456 p.

17. Sparkfun Electronics. XBee Shield Hookup Guide. [en línea]. <https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide>. [Consulta: 10 de abril de 2015].

18. Universidad de Guadalajara. *Robótica.* [en línea]. <http://docentes.uto.edu.bo/gguzmanm/wp-content/uploads/LIBRO_DE_ROBOTICA.pdf>. [Consulta: 20 de marzo de 2015].

19. WARREN, John-David. *Arduino Robotics*. Apress. 2011. 581 p.