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

# EVALUACIÓN DE EFICIENCIA DE UN ALGORITMO PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN

**Luis Gabriel Catalán Soto**
Asesorado por el Ing. Guillermo Antonio Puente Romero
e Ing. Héctor Francisco Galeros Juárez

Guatemala, mayo de 2014

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA


FACULTAD DE INGENIERÍA

# EVALUACIÓN DE EFICIENCIA DE UN ALGORITMO PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

**LUIS GABRIEL CATALÁN SOTO**

ASESORADO POR EL ING. GUILLERMO ANTONIO PUENTE ROMERO
ING. HÉCTOR FRANCISCO GALEROS JUÁREZ
AL CONFERÍRSELE EL TÍTULO DE

**INGENIERO EN ELECTRÓNICA**

GUATEMALA, MAYO DE 2014

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA


## NÓMINA DE JUNTA DIRECTIVA

| | |
|---|---|
| DECANO | Ing. Murphy Olympo Paiz Recinos |
| VOCAL I | Ing. Alfredo Enrique Beber Aceituno |
| VOCAL II | Ing. Pedro Antonio Aguilar Polanco |
| VOCAL III | Inga. Elvia Miriam Ruballos Samayoa |
| VOCAL IV | Br. Walter Rafael Véliz Muñoz |
| VOCAL V | Br. Sergio Alejandro Donis Soto |
| SECRETARIO | Ing. Hugo Humberto Rivera Pérez |

## TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

| | |
|---|---|
| DECANO | Ing. Murphy Olympo Paiz Recinos |
| EXAMINADOR | Ing. Romeo Neftalí López Orozco |
| EXAMINADORA | Inga. Ingrid Salomé Rodríguez de Loukota |
| EXAMINADOR | Ing. José Anibal Silva de los Angeles |
| SECRETARIO | Ing. Hugo Humberto Rivera Pérez |

# HONORABLE TRIBUNAL EXAMINADOR

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

**IMPLEMENTACIÓN Y EVALUACIÓN DE UN ALGORITMOS PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN**

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Escuela de Mecánica Eléctrica, con fecha 28 de junio de 2013.

**Luis Gabriel Catalán Soto**

Guatemala,17 de marzo de 2014.

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

Ingeniero Guzmán:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **"EVALUACIÓN DE EFICIENCIA DE UN ALGORITMO PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN"**, desarrollado por el estudiante Luis Gabriel Catalán Soto con carné No. 2005-11869, ya que considero que cumple con los requisitos establecidos, por lo que el autor y mi persona somos responsables del contenido y conclusiones del mismo.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

Ing. Guillermo Antonio Puente Romero
ASESOR
Colegiado 5898

Guillermo A. Puente R.
INGENIERO ELECTRONICO
COL. # 5898

Guatemala, 24 de marzo de 2014.

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

Ingeniero Guzmán:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **"EVALUACIÓN DE EFICIENCIA DE UN ALGORITMO PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN"**, desarrollado por el estudiante Luis Gabriel Catalán Soto con carné No. 2005-11869, ya que considero que cumple con los requisitos establecidos, por lo que el autor y mi persona somos responsables del contenido y conclusiones del mismo.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

Ing. Héctor Francisco Galeros Juárez
ASESOR
Colegiado 9662

Héctor F. Galeros Juárez
INGENIERO ELECTRONICO
Col. No. 9662

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Ref. EIME 15. 2014
Guatemala, 25 de MARZO 2014.

Señor Director
Ing. Guillermo Antonio Puente Romero
Escuela de Ingenierìa Mecànica Elèctrica
Facultad de Ingenierìa, USAC.

Señor Director:

**Me permito dar aprobación al trabajo de Graduación titulado: EVALUACIÓN DE EFICIENCIA DE UN ALGORITMO PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN, del estudiante Luis Gabriel Catalán Soto, 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

**UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**

**FACULTAD DE INGENIERIA**

REF. EIME 15. 2014.

**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; LUIS GABRIEL CATALÁN SOTO titulado: EVALUACIÓN DE EFICIENCIA DE UN ALGORITMO PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN, procede a la autorización del mismo.**

**Ing. Guillermo Antonio Puente Romero**

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

**GUATEMALA, 7 DE ABRIL 2,014.**

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 226.2014

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: **EVALUACIÓN DE EFICIENCIA DE UN ALGORITMO PARA REDUCCIÓN DE ERRORES POR RUIDO BLANCO Y LA TOLERANCIA DE COMPONENTES ELECTRÓNICOS EN AMPLIFICADORES DE INSTRUMENTACIÓN,** presentado por el estudiante universitario **Luis Gabriel Catalán Soto,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Murphy Olympo Paiz Recinos
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, 15 de mayo de 2014

/gdech

## ACTO QUE DEDICO A:

**Mis padres** María Lucila Soto y Luis Fernando Catalán, por todo su esfuerzo que realizaron para brindarme lo mejor y por el cariño mostrado durante toda mi vida.

# AGRADECIMIENTOS A:

| | |
|---|---|
| **La Universidad de San Carlos de Guatemala** | Por haberme abierto las puertas, brindado conocimiento y sabiduría. |
| **Mis amigos de la Facultad** | Por haberme brindado su apoyo. |
| **Andrea de la Cruz** | Por ser una importante influencia en la elaboración de este trabajo de graduación. |

# ÍNDICE GENERAL

# ÍNDICE DE ILUSTRACIONES

## FIGURAS



## TABLAS

# LISTA DE SÍMBOLOS

| Símbolo | Significado |
|---|---|
| **OpAmp** | Amplificador Operacional |
| ∅ | Conjunto vacío |
| **I** | Corriente |
| **dB** | Decibeles |
| **Mod** | Función módulo |
| $\mathbb{R}^{\mathbf{n}}$ | Espacio Vectorial de Dimensión n |
| ∫ | Integral |
| ∩ | Intersección entre conjuntos |
| **(a,b)** | Intervalo abierto entre a y b |
| **[a,b]** | Intervalo cerrado entre a y b |
| **MCU** | Microcontrolador |
| $\mathbb{R}$ | Números reales |
| ∀ | Para todo |
| ∈ | Pertenece a |
| **Vee** | Pin de voltaje negativo para el Opamp |
| **Vcc** | Pin de voltaje positivo para el Opamp |
| **CMRR** | Relación de Rechazo en Modo Común |
| **σ** | Sigma |
| Σ | Suma |
| **CPU** | Unidad Central de Procesamiento |
| ∪ | Unión entre conjuntos |
| **V** | Voltaje |
| **Vo** | Voltaje de salida |

# GLOSARIO

**Amplificador operacional** Circuito Integrado el cual se utiliza como amplificador diferencial y además idealmente tiene una impedancia de entrada, ganancia y ancho de banda infinito, impedancia de salida nula, tiempo de respuesta nula y ningún ruido, es también llamado OpAmp.

**Impedancia** Medida de oposición que presenta un circuito a una corriente cuando se aplica un voltaje.

**Modo común** Se refiere al voltaje que existe entre las terminales diferenciales de un conjunto de amplificadores operacionales configurados como amplificadores de instrumentación.

**Proceso estocástico** Se refiere a un proceso el cual incluye un componente de incertidumbre respecto a su valor en el tiempo.

**Pseudoaleatorios** Variable generada a través de un proceso determinístico la cual no presenta ningún patrón o regularidad desde un punto de vista estadístico.

| | |
|---|---|
| **Seguidor de voltaje** | Configuración de un amplificador operacional el cual proporciona en su salida la misma tensión que a la entrada, además que permite adaptar impedancias entre circuitos. |
| **Simulación** | Es la utilización de un modelo matemático el cual intenta encontrar soluciones analíticas a problemas que permiten la predicción del comportamiento de un sistema de un conjunto de parámetros y condiciones iniciales. |
| **Variable analógica** | Tipo de variable la cual puede tomar valores continuos. |
| **Variable digital** | Tipo de variable la cual toma valores discretos, específicamente 0 y 1. |
| **Variables Independientes e Idénticamente Distribuidas (IID)** | Variables aleatorias que están definidas en el mismo espacio de probabilidad, poseen la misma función de distribución de probabilidad, y son independientes estadísticamente entre sí. |

# RESUMEN

El siguiente trabajo de graduación consiste en un análisis de los efectos del ruido blanco y las tolerancias en resistencias en amplificadores de instrumentación con dos o tres amplificadores operacionales

El trabajo está estructurado en cinco capítulos cuyo contenido se describe a continuación:

El primer capítulo contiene la información referente a los conceptos básicos acerca de los amplificadores de instrumentación, se describe a detalle su pieza fundamental, el amplificador operacional, además de una pequeña reseña histórica de los mismos. Se desarrolla la función de transferencia del amplificador de instrumentación con dos y tres amplificadores operacionales.

En el segundo capítulo se desarrolla una descripción de las fuentes de error en instrumentación, como caso particular se caracteriza el ruido blanco aditivo y la tolerancia en las resistencias. Además se presentan los fundamentos teóricos de la teoría axiomática de probabilidades.

El tercer capítulo corresponde a una descripción del método de MonteCarlo, se da una reseña histórica del mismo y finaliza con una simulación del amplificador de instrumentación considerando las fuentes de error.

En cuarto capítulo se describen las piezas fundamentales de los microcontroladores, así como las familias PIC, ARM y AVR. Se presenta un estudio acerca de la segmentación del mercado actual de los mismos, además

se realiza una implementación de un algoritmo para la reducción de los efectos producidos por la incorporación de ruido blanco y tolerancia en las resistencias.

Finalmente en el quinto capítulo se discuten los resultados obtenidos de las simulaciones de cada microcontrolador, y se realiza un análisis en base a la eficiencia, para seleccionar determinado microcontrolador.

# OBJETIVOS

## General

Evaluar la eficiencia de implementar un algoritmo para la reducción de errores provocados por ruido blanco y la tolerancia de componentes electrónicos en amplificadores de instrumentación.

## Específicos

1. Presentar información sobre las bases teóricas de los amplificadores de instrumentación y procesos que se desarrollarán.

2. Caracterizar las diversas fuentes de error en instrumentación.

3. Modelar y simular las fuentes de error y proponer un algoritmo para obtener mediciones fiables.

4. Presentar fundamentos e implementar un algoritmo en un microcontrolador de las familias Microchip, ARM y Atmel.

5. Evaluar la eficiencia del algoritmo y seleccionar un microcontrolador adecuado.

# INTRODUCCIÓN

En el ámbito de la ingeniería electrónica, específicamente en el área de diseño de circuitos, el encargado de diseño generalmente se encontrará que en la mayoría de casos las condiciones de idealidad no se cumplen, ya que los componentes, las señales, las mediciones y las herramientas estarán sujetos a incertidumbre, y esto afectará el desempeño y funcionalidad inicial de los circuitos que inicialmente se planteó.

Los amplificadores de instrumentación son ampliamente utilizados en ámbitos de industria, medicina, electrónica de consumo, automóviles, etc. Por lo que su uso toma especial relevancia en el diseño de circuitos, ya que debido a sus características como lo son alta impedancia de entrada, baja impedancia de salida, y alto rechazo al modo común, estos son utilizados para la medición de voltajes en el orden de los microvoltios, siendo esta la razón principal por la que deberían de entrar en consideración métodos para la reducción de errores, como es el caso de algún algoritmo que lo permita.

Un algoritmo para la reducción de errores consiste en un conjunto de instrucciones contenidas en un microcontrolador, microprocesador, etc. Las cuales permiten discriminar el ruido y los errores de tipo estadístico de la señal que se desea medir, y así obtener una señal más exacta y precisa.

Las fuentes de error consideradas se refieren al ruido blanco, que se refiere a perturbaciones las cuales se encuentran en todo el espectro de frecuencias, su valor esperado es cero y además su varianza es constante y no correlacionada en algún período distinto al de observación. Además la

tolerancia en componentes electrónicos se refiere al rango de valores máximo permitido por los fabricantes y su desviación respecto al valor teórico, tomando en cuenta imperfecciones, temperatura, humedad, potencia, etc.

Este proyecto de graduación se realizó con el interés de observar los efectos de no considerar un sistema ideal; segundo, si es posible reduciendo la tolerancia de los componentes lograr el mismo resultado, ya que esto significaría la reducción de costos al utilizar componentes no tan precisos y por último, se observó la necesidad de utilizar algoritmos o algún tipo de procesamiento para obtener una señal aceptable en la salida del sistema.

# 1. AMPLIFICADORES DE INSTRUMENTACIÓN

Son un circuito que consiste en una determinada configuración de amplificadores operacionales y resistencias. El concepto de amplificador operacional surgió del desarrollo extensivo de la electrónica analógica para la computación durante los años 1 940. Estos reciben su nombre debido a su capacidad para realizar operaciones matemáticas tales como la suma, integración y diferenciación. Luego, con la adición de los amplificadores logarítmicos y el concepto de retroalimentación, se amplió el espectro de funciones a la multiplicación y la división.

Estas características permiten a los circuitos con amplificadores operacionales simular ecuaciones diferenciales, como por ejemplo las que describen trayectorias del vuelo de un aeroplano, y demás fenómenos físicos.

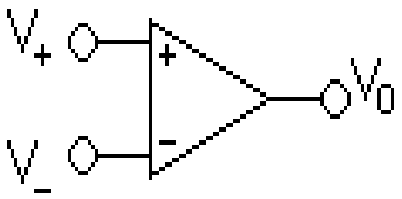
Esto provocó que un sin número de computadoras analógicas fueran usadas durante la segunda guerra mundial con el fin de producir armas más inteligentes capaces de predecir la posición de algún blanco en un intervalo de tiempo, de modo que se pudieran determinar las condiciones óptimas de disparo para asegurar con una alta probabilidad de acertar en el blanco.

A esto se debió el éxito de los amplificadores operacionales ya que podían ser utilizados para programar una computadora analógica de manera rápida cambiando las ganancias de los mismos, convirtiéndoles en una herramienta indispensable para la artillería militar y además para la investigación académica.

## 1.1. El amplificador operacional

En la figura 1 se presenta el símbolo esquemático generalmente utilizado para describir el amplificador operacional, sin considerar los voltajes de polarización.

Figura 1. **Símbolo del amplificador operacional**




Fuente: PÉREZ, Miguel; et al. *Instrumentación electrónica*. p. 49.

La función que desarrolla el amplificador operacional es la de amplificar con una ganancia de tensión $A_v$ de la tensión diferencial $v_{ent}$ según la ecuación 1.

$$v_o = A_v v_{ent} \quad (1)$$

### 1.1.1. Características ideales

La consideración de idealidad del operacional queda definida por los valores extremos de los diversos parámetros que lo definen, los cuales consisten en:

- Resistencia de entrada infinita, lo que obliga a que la corriente de entrada por cualquiera de sus dos entradas sea siempre nula.

- Resistencia de salida nula, lo que implica capacidad de proporcionar corriente sobre cualquier carga.

- Ganancia de tensión $A_v$ infinita, lo que significa que si su salida es finita, la tensión diferencial de entrada deberá ser nula.

Como consideración adicional, se debe de tener en cuenta que la tensión de salida siempre se encuentra limitada por las tensiones de alimentación y no se podrán sobrepasar en ningún caso; con lo que, si el operacional está alimentado entre un valor positivo $v_{cc}$ y un valor negativo $-v_{ee}$, se cumplirá la ecuación 2, la cual indica que la tensión $v_o$ siempre estará comprendida entre estos límites, es decir:

$$-v_{ee} \leq v_o \leq v_{cc} \tag{2}$$

Las consideraciones de idealidad permiten un estudio simplificado del operacional y de los circuitos construidos en torno a él, además existen muchas aplicaciones de los amplificadores operacionales tanto de tipo analógico como digital y una multitud de circuitos en los que se combinan con otros bloques para realizar funciones mucho más complejas.

### 1.1.2. Parámetros reales de los amplificadores operacionales

Un amplificador operacional en general está compuesto por tres etapas:

- Etapa de entrada tipo diferencial que aporta una buena parte de la ganancia total del circuito. En esta se encuentran:
  - Corrientes de polarización
  - Tensión de desviación de entrada

- Tensión de desviación de salida

- Etapa intermedia que se encarga fundamentalmente de la adaptación de niveles ya que el acoplamiento entre etapas se hace en corriente continua.

- Etapa de salida que proporciona la potencia necesaria para manejar pequeñas cargas y que incorpora protecciones.

## 1.2. Amplificadores de instrumentación

El amplificador de instrumentación es un amplificador diferencial de tensión de precisión con un circuito optimizado para su trabajo en los ámbitos más hostiles, caracterizados por grandes fluctuaciones de la temperatura e intenso ruido eléctrico. Además, estos amplificadores especiales deben ser capaces de trabajar con sensores de resistencia interna apreciable no simétrica, sobre los que el ruido eléctrico inducido y/o conducido tiene una gran influencia y además que ofrecen señales eléctricas muy débiles, es por esta razón la cual son ampliamente utilizados.

### 1.2.1. Particularidades del amplificador de instrumentación

Para ser efectivo, un amplificador de instrumentación debe ser capaz de amplificar señales del orden de los microvoltios y a la vez rechazar tensiones de modo común del orden de voltios. Esto presupone que el amplificador de instrumentación tenga un elevado rechazo al modo común CMRR del orden de 80 o 100 dB "(El decibelio es una unidad logarítmica, que representa la relación entre la magnitud estudiada y la de referencia)"[1] a la vez que una gran ganancia

[1] *Decibelio*. <http://es.wikipedia.org/wiki/Decibelio.> [Consulta: 10 de marzo de 2014.]

diferencial $A_d$. A continuación expondrán las configuraciones de circuitos de amplificadores de instrumentación sobre los cuales se efectuará el análisis y simulación posteriormente.

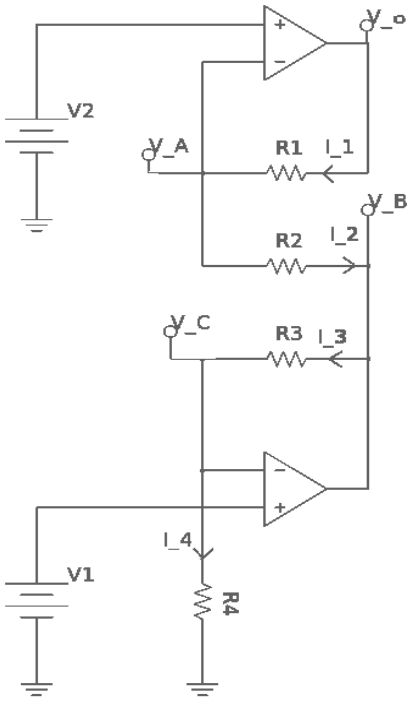
### 1.2.2. Configuraciones de circuitos

Existen varias configuraciones de circuitos en las cuales un amplificador de instrumentación puede ser formado, mediante determinadas conexiones de amplificadores operacionales conservando sus características de impedancia de entrada muy alta, impedancia de salida muy baja y además la etapa diferenciadora de voltaje. A continuación se presenta las estructuras más comunes de configuraciones de amplificadores de instrumentación.

#### 1.2.2.1. Amplificadores de instrumentación de dos operacionales

El amplificador de instrumentación de dos operacionales, como el que se muestra en la figura 2, es una configuración que presenta impedancias de entrada alta e igual lo que permite que la fuente de señal pueda tener una impedancia interna alta y/o desequilibrada. La principal desventaja de este circuito es que su margen de entrada para la tensión de modo común es una función de la ganancia del mismo.

Figura 2. **Amplificador de Instrumentación con dos operacionales**




Fuente: PÉREZ, Migue; et al. *Instrumentación electrónica*. p. 102.

Para su análisis se supone una entrada de voltaje diferencial para el sistema, de tal modo que se cumpla la condición mostrada en la ecuación

$$v_2 > v_1 \tag{3}$$

Aplicando el método de nodos para obtener las ecuaciones para cada corriente, con las direcciones propuestas en el diagrama anterior, el desarrollo posterior se muestra en las ecuaciones 4 - 16:

$$I_1 = \frac{v_0 - v_1}{R_1} \tag{4}$$

$$I_2 = \frac{v_A - v_B}{R_2} \quad (5)$$

$$I_3 = \frac{v_B - v_C}{R_3} \quad (6)$$

$$I_4 = \frac{v_C}{R_2} \quad (7)$$

Considerando características ideales del amplificador operacional:

$$I' = I_1 = I_2 \quad (8)$$

$$I'' = I_3 = I_4 \quad (9)$$

$$v_C = v_1 \quad (10)$$

$$v_A = v_2 \quad (11)$$

Resolviendo el sistema:

$$I'' = \frac{v_B - v_1}{R_3} = \frac{v_1}{R_4} \quad (12)$$

$$v_B = \left(\frac{R_3}{R_4} + 1\right) v_1 \quad (13)$$

$$I' = \frac{v_0 - v_2}{R_1} = \frac{v_2 - v_B}{R_2} \quad (14)$$

$$\frac{v_0 - v_2}{R_1} = \frac{v_2 - \left(\frac{R_3}{R_4} + 1\right) v_1}{R_2} \quad (15)$$

Finalmente se obtiene, la ecuación 16 que describe la función de transferencia del amplificador de instrumentación:

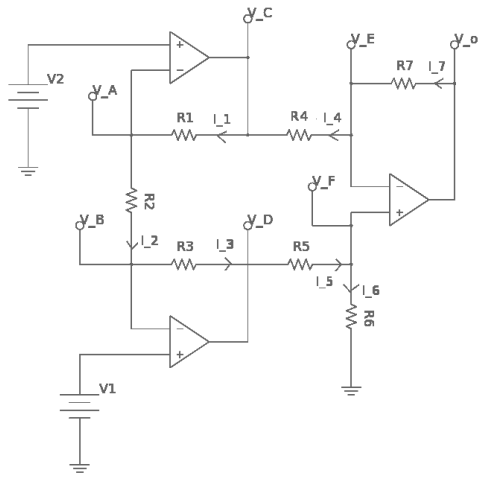
$$v_0 = v_2 \left(\frac{R_1}{R_2} + 1\right) - v_1 \left(\frac{R_1}{R_2}\right)\left(\frac{R_3}{R_4} + 1\right) \quad (16)$$

#### 1.2.2.2. Amplificadores de instrumentación de tres operacionales

El amplificador de instrumentación de tres operacionales consiste en un circuito que integra las ventajas del amplificador de instrumentación de dos operacionales y además reduce en buena medida sus desventajas. Este circuito utiliza dos amplificadores operacionales configurados como seguidores de voltaje, ya que debido a su alta impedancia dos etapas nos permite aislar en buena medida los efectos producidos por las impedancias provenientes del circuito el cual se está censando, finalmente se obtiene un voltaje diferencial al efectuar una resta ponderada entre ambas terminales.

Por estas razones se dice que un amplificador de este tipo consiste en dos etapas: la de entrada y la diferencial. En la figura 3 se presenta el diagrama esquemático de este circuito.

Figura 3. **Amplificador de instrumentación con tres operacionales**




Fuente: PÉREZ, Migue; et al. *Instrumentación electrónica*. p. 49.

Para el análisis se toman los supuestos anteriores, de forma que se tiene la condición de la ecuación 17:

$$v_2 > v_1 \tag{17}$$

Utilizando el método de nodos para obtener las corrientes, se muestra el desarrollo en las ecuaciones 18 - 39:

$$\mathrm{I}_1 = \frac{\mathrm{v_c} - \mathrm{v_A}}{\mathrm{R}_1} \tag{18}$$

$$\mathrm{I}_2 = \frac{\mathrm{v_A} - \mathrm{v_B}}{\mathrm{R}_2} \tag{19}$$

$$\mathrm{I}_3 = \frac{\mathrm{v_B} - \mathrm{v_D}}{\mathrm{R}_3} \tag{20}$$

$$\mathrm{I}_4 = \frac{\mathrm{v_E} - \mathrm{v_C}}{\mathrm{R}_4} \tag{21}$$

$$\mathrm{I}_5 = \frac{\mathrm{v_D} - \mathrm{v_F}}{\mathrm{R}_5} \tag{22}$$

$$\mathrm{I}_6 = \frac{\mathrm{v_F}}{\mathrm{R}_6} \tag{23}$$

$$\mathrm{I}_7 = \frac{\mathrm{v_0} - \mathrm{v_E}}{\mathrm{R}_7} \tag{24}$$

Considerando los amplificadores operacionales como ideales:

$$\mathrm{I}_1 = \mathrm{I}_2 = \mathrm{I}_3 = \mathrm{I}' \tag{25}$$

$$\mathrm{I}_4 = \mathrm{I}_7 = \mathrm{I}'' \tag{26}$$

$$\mathrm{I}_5 = \mathrm{I}_6 = \mathrm{I}''' \tag{27}$$

$$\mathrm{v_E} = \mathrm{v_F} = \mathrm{v}' \tag{28}$$

$$\mathrm{v}_2 = \mathrm{v_A} \tag{29}$$

$$\mathrm{v}_1 = \mathrm{v_B} \tag{30}$$

Resolviendo el sistema para las ecuaciones de los nodos:

$$I''' = \frac{v_D - v}{R_5} = \frac{v'}{R_6} \tag{31}$$

$$v' = \left(\frac{R_6 + R_5}{R_6}\right) v_D \tag{32}$$

Haciendo las sustituciones necesarias:

$$I' = \frac{v_C - v_2}{R_1} = \frac{v_2 - v_1}{R_2} \tag{33}$$

$$v_C = \left(\frac{R_1 + R_2}{R_2}\right) v_2 - \left(\frac{R_1}{R_2}\right) v_1 \tag{34}$$

$$I' = \frac{v_2 - v_1}{R_2} = \frac{v_1 - v_D}{R_3} \tag{35}$$

$$v_D = \left(\frac{R_2 + R_3}{R_3}\right) v_1 - \left(\frac{R_3}{R_2}\right) v_2 \tag{36}$$

Finalmente se obtiene:

$$I'' = \frac{v' - v_C}{R_4} = \frac{v_0 - v'}{R_7} \tag{37}$$

$$v_0 = \left(\frac{R_4 + R_7}{R_4}\right) v' - \left(\frac{R_7}{R_4}\right) v_C \tag{38}$$

La respuesta del sistema está dada por la ecuación 39:

$$v_0 = \left[\left(\frac{R_6}{R_4 R_3}\right)\frac{(R_4 + R_7)(R_2 + R_3)}{R_5 + R_6} + \frac{R_1 R_7}{R_2 R_4}\right] v_1 - \left[\left(\frac{R_3 R_6}{R_2 R_4}\right)\frac{R_4 + R_7}{R_5 + R_6} + \left(\frac{R_7}{R_2 R_4}\right)(R_1 + R_2)\right] v_2 \tag{39}$$

# 2. FUENTES DE ERROR EN INSTRUMENTACIÓN

En este capítulo se estudiarán las técnicas para conseguir que los sistemas continúen funcionando correctamente, a pesar de fallos en su hardware o bien errores de software. Este tipo de sistemas se denominan sistemas tolerantes a fallos.

## 2.1. Conceptos generales sobre tolerancia a fallos

Los conceptos relacionados con la tolerancia a fallos tienen cada vez más importancia; esto se debe a la proliferación de los sistemas de cómputo y procesamiento de datos en tiempo real y además el uso de estos cada vez en más ámbitos. Algunas de las aplicaciones de los estos sistemas resultan lo suficientemente críticas como para protegerlas contra potenciales fallos. En esos entornos, un funcionamiento incorrecto del sistema resultaría catastrófico y podría causar importantes perjuicios. Un tipo de aplicaciones que deben ser especialmente tolerantes a fallos, son aquellas en las que se prevea un funcionamiento continuo durante largo tiempo sin posibilidad de intervención.

En general, se definen tres conceptos fundamentales en el ámbito de la tolerancia a fallos, estos se resumen en la figura 4:

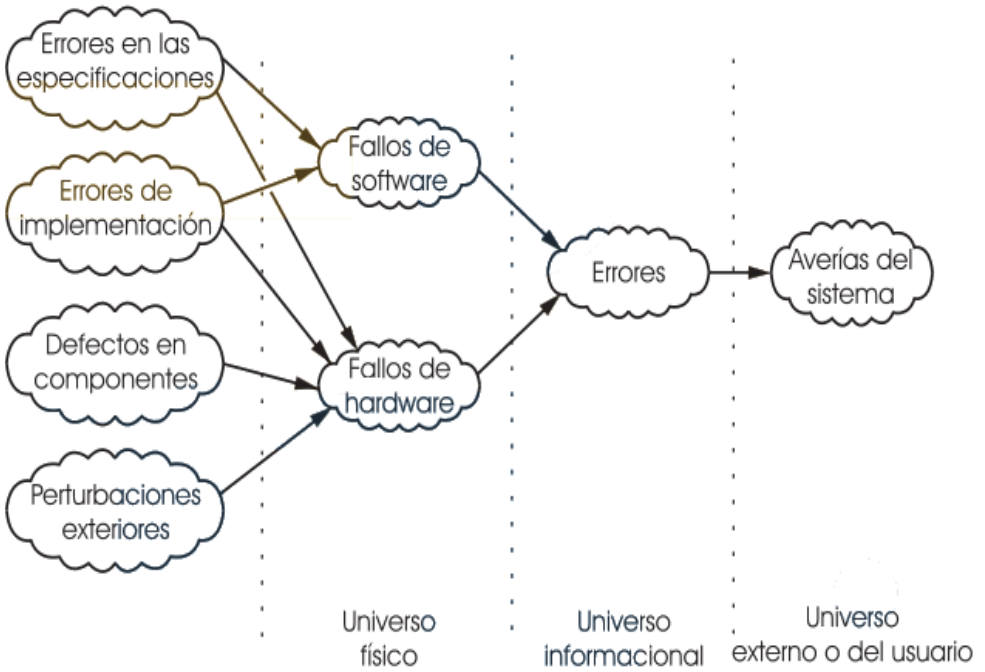
- Fallo se refiere a cualquier defecto, físico o lógico, en cualquier componente, hardware o software, de un sistema. Dentro de esta categoría se incluyen los contactos accidentales entre conductores eléctricos, cortes en los mismos, defectos en los componentes, variaciones en el funcionamiento de los elementos electrónicos

provocadas por perturbaciones externas tales como la temperatura, ondas electromagnéticas, etc. Un fallo se enmarca en el universo físico.

- Un error es la manifestación o el resultado de un fallo. Dicho de otra forma, un error es la consecuencia de un fallo desde el punto de vista de la información. Los errores se enmarcan dentro del llamado universo informacional.

- Una avería es producida por un error tal que cause un funcionamiento incorrecto del sistema desde el punto de vista externo. Las averías se producen en el universo externo o universo del usuario.

Figura 4. **Esquema general de fuentes de error**




Fuente: IBÁÑEZ, Javier. *Sistemas tolerantes a fallos*. <http://www.infor.uva.es/~bastida/Arquitecturas%20Avanzadas/Tolerant.pdf.>. Consulta: 29 de agosto de 2013.

Además cabe mencionar que los errores y fallos que se presentan en estos sistemas poseen a su vez características temporales, está la se le llama latencia y se refiere al tiempo en el cual se produce hasta que se manifiesta.

A continuación se presenta el análisis sobre las fallas, específicamente en los defectos en los componentes y a las variaciones en el funcionamiento de los elementos electrónicos debidas a perturbaciones externas, ya que estos son los más importantes que conciernen al agente encargado del diseño de circuitería de hardware.

### 2.1.1. Caracterización de los fallos

Los fallos pueden caracterizarse atendiendo a varios criterios como se enlistan a continuación, estos se encuentran en forma resumida en la figura 5:

- Las causas de los fallos puede ser múltiples, como se mencionó con anterioridad, especificaciones incorrectas en el momento del diseño, fallos en el proceso de implementación, defectos en los componentes, perturbaciones externas, etc.

- La naturaleza de los fallos específicamente se puede clasificar según la parte del sistema que falla: software o hardware, dentro del hardware, el fallo puede tener naturaleza analógica o digital.

- En cuanto a la duración, los fallos pueden ser permanentes, que se caracterizan por continuar indefinidamente en el tiempo si no se toma alguna acción correctora, intermitentes, repetidos, aleatorios, o transitorio.

- La extensión de un fallo indica si solo afecta a un punto localizado o si afecta a la globalidad del hardware, del software o de ambos.

- La variabilidad de los fallos pueden ser determinados, si su estado no cambia con el tiempo, incluso aunque cambie la entrada u otras condiciones, o indeterminados, cuyo estado puede cambiar cuando varíen algunas de las condiciones.

Figura 5. **Caracterización de los fallos**


Fuente: IBÁÑEZ, Javier. *Sistemas tolerantes a fallos*. <http://www.infor.uva.es/~bastida/Arquitecturas%20Avanzadas/Tolerant.pdf.>. Consulta: 29 de agosto de 2013.

## 2.2. Fundamentos de probabilidad

Adicionalmente a los fundamentos técnicos, es necesario proporcionar los elementos necesarios bajo los cuales teóricamente es posible la simulación, ya que su potencia yace en la teoría axiomática de probabilidades que se presenta a continuación.

### 2.2.1. Eventos y probabilidad

Se tiene a un conjunto no vacío $\Omega$. Un $\sigma - \text{álgebra } \mathrm{F}$ en el conjunto $\Omega$ que es una familia de subconjuntos que cumplen:

A. El conjunto vacío $\emptyset$ pertenece a la $\sigma - \text{álgebra } \mathrm{F}$.

B. Si $\mathrm{A} \in \mathrm{F}$, entonces su complemento también pertenece a $\mathrm{F}$.

C. Si una secuencia de conjuntos $A_1, A_2, \ldots$ pertenece a $\mathrm{F}$, entonces la unión de $A_1 \cup A_2 \cup \ldots$ también pertenece a F.

Sabido que F es una $\sigma - \text{álgebra}$ en el conjunto $\Omega$ se define una medida de probabilidad P que es una función:

$$\mathrm{P}: \mathrm{F} \rightarrow [0,1] \tag{40}$$

Tal que:

A. $\mathrm{P}(\Omega) = 1$

B. Si $\mathrm{A}_1, \mathrm{A}_2, \ldots$ son conjuntos disjuntos ($\mathrm{A_i} \cap \mathrm{A_j} \neq 0, \forall \mathrm{i} \neq \mathrm{j}$) que pertenece a $\mathrm{F}$ entonces $\mathrm{P}(\mathrm{A}_1 \cup \mathrm{A}_2 \cup \ldots) = \mathrm{P}(\mathrm{A}_1) + \mathrm{P}(\mathrm{A}_2) + \cdots$.

Se le llama espacio de probabilidad al conjunto formado por $\{\Omega, \mathrm{F}, \mathrm{P}\}$. Los conjuntos pertenecientes a $\mathrm{F}$ son llamados eventos. Un evento A se dice que es casi seguro que ocurra cuando $\mathrm{P}(\mathrm{A}) = 1$.

### 2.2.2. Variables aleatorias

Si $\mathrm{F}$ es una $\sigma-\text{álgebra}$ en $\Omega$, entonces una función $\xi:\Omega\to\mathbb{R}$ se dice que es $\mathrm{F}-\text{medible}$ si:

$$\{\xi\in \mathrm{B}\}\in \mathrm{F} \tag{41}$$

Para cualquier conjunto Borel $B\in B(\mathbb{R})$. Si $\{\Omega,\mathrm{F},\mathrm{P}\}$ es un espacio de probabilidad, entonces la función $\xi$ es llamada una variable aleatoria.

Además para toda variable aleatoria $\xi:\Omega\to\mathbb{R}$ tiene asociada una medida de probabilidad:

$$\mathrm{P}_{\xi}(\mathrm{B})=\ \mathrm{P}[\xi\in \mathrm{B}] \tag{42}$$

En los $\mathbb{R}$ definidos en la $\sigma-\text{álgebra}$ del conjunto Borel $\mathrm{B}\in \mathrm{B}(\mathbb{R})$, se le llama a $\mathrm{P}_{\xi}$ la distribución de $\xi$. De forma análoga la función $\mathrm{F}_{\xi}:\mathbb{R}\to[0,1]$, está definida por:

$$\mathrm{F}_{\xi}(\mathrm{x})\ =\ \mathrm{P}(\xi\leq \mathrm{x}) \tag{43}$$

Es llamada la función de distribución acumulada de $\xi$.

Para el caso de densidad de probabilidad, si existe una distribución $f_{\xi}:\mathbb{R}\to\mathbb{R}$ tal que para cualquier conjunto Borel $\mathrm{B}\subset\mathbb{R}$, se cumple:

$$P(\xi\in B)=\int_{B}\ f_{\xi}(x)dx \tag{45}$$

Entonces se dice que $\xi$ es una variable aleatoria con una distribución continua absoluta y $\mathrm{f}_{\xi}$ es su función de densidad. Si es que existe una secuencia finita o infinita de pares distintos de números reales $\mathrm{X}_1, \mathrm{X}_2, \ldots$ tal que para cualquier conjunto Borel $\mathrm{B} \subset \mathbb{R}$.

$$\mathrm{P}(\xi \in \mathrm{B}) = \sum_{\mathrm{x_i} \in \mathrm{B}} \mathrm{P}[\xi = \mathrm{X_i}] \tag{46}$$

Entonces se dice que $\xi$ tiene una distribución discreta con valores $\mathrm{X}_1, \mathrm{X}_2, \ldots$ y peso $\mathrm{P}(\xi = X_i)$.

La definición conjunta de varias variables $\xi_1, \ldots, \xi_\mathrm{n}$ es una medida de probabilidad $\mathrm{P}_{\xi_1,\ldots,\xi_\mathrm{n}}$ en los $\mathbb{R}^n$ tal que:

$$\mathrm{P}_{\xi_1,\ldots,\xi_\mathrm{n}}(\mathrm{B}) = \mathrm{P}[(\xi_1, \ldots, \xi_\mathrm{n}) \in \mathrm{B}] \tag{47}$$

Para cualquier conjunto Borel $\mathrm{B}$ en $\mathbb{R}^n$. Si es que existe una función Borel $\mathrm{f}_{\xi_1,\ldots,\xi_\mathrm{n}} : \mathbb{R} \to \mathbb{R}^\mathrm{n}$ tal que:

$$\mathrm{P}[(\xi_1, \ldots, \xi_\mathrm{n}) \in \mathrm{B}] = \int_B \mathrm{f}(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}) \mathrm{dx}_1 \ldots \mathrm{dx}_\mathrm{n} \tag{48}$$

Para cualquier conjunto Borel $\mathrm{B}$ en $\mathbb{R}^\mathrm{n}$, entonces $\mathrm{f}_{\xi_1,\ldots,\xi_\mathrm{n}}$es llamada densidad conjunta de $\xi_1, \ldots, \xi_\mathrm{n}$, además se deben de cumplir las condiciones de integrabilidad, se dice que una variable aleatoria $\xi : \Omega \to \mathbb{R}$ es integrable cuando:

$$\int_\Omega |\xi| dP < \infty \tag{49}$$

En otras palabras la suma de las variables aleatorias a lo largo de todo el espacio de probabilidades es convergente, entonces:

$$E(\xi) = \int_{\Omega} \xi dP \tag{50}$$

Existe y es llamada la esperanza de $\xi$. Por último una variable aleatoria $\xi: \Omega \longrightarrow \mathbb{R}$ es llamada integrable al cuadrado si es que:

$$\int_{\Omega} |\xi|^2 \mathrm{dP} < \infty \tag{51}$$

Entonces la varianza de $\xi$ puede ser definida por:

$$var(\xi) = \int_{\Omega} \left(\xi - \mathrm{E}(\xi)\right)^2 dP \tag{52}$$

## 2.3. Ruido

Las señales incorporan niveles de ruido y, además, se va añadiendo más ruido a lo largo de la cadena de tratamiento de los datos, de tal forma, que este se convierte en un parámetro muy importante en sistemas de instrumentación.

El ruido es una forma de onda que varía con el tiempo de manera impredecible, es decir, este es un proceso estocástico. Una señal o forma de onda acompañada de ruido se le llama corrupta o contaminada. El sistema debe ser capaz de distinguir la señal del ruido y con ello reproducir una señal sin distorsión parecida a la original.

### 2.3.1. Caracterización del ruido blanco

El ruido blanco es una señal aleatoria con una densidad espectral de potencia uniforme. En otras palabras, una señal que contiene una potencia igual en cualquier frecuencia con ancho de banda fijo.

#### 2.3.1.1. Propiedades estadísticas del ruido blanco

Dentro del estudio de las series temporales en estadística, a menudo es útil definir un proceso de ruido blanco igualmente dentro del dominio temporal. Las definiciones presentadas acá son hechas para los procesos en tiempo discreto y con valores continuos de igual forma son validad para el caso continuo.

Un proceso $\epsilon_t$ está calificado como ruido blanco independiente si cumple con:

$$\mathrm{E}[\,\epsilon_t\,] = 0 \tag{53}$$

$$\mathrm{E}[\,\epsilon_t^2\,] = \sigma^2 \tag{54}$$

$$\mathrm{E}[\,\epsilon_t \epsilon_\tau\,] = 0 \tag{55}$$

En palabras la definición consiste en $\epsilon_t$ y $\epsilon_\tau$ son procesos independientes, o bien su covarianza es 0, además que el valor esperado ruido es 0 y su varianza es igual a $\sigma^2$.

Un proceso $\epsilon_t$ está calificado como ruido blanco gaussiano si es un ruido blanco independiente y cumple con:

$$\epsilon_t \sim \mathrm{N}\,(0, \sigma^2) \tag{56}$$

## 2.4. Caracterización de la tolerancia en resistencias

Para la caracterización de la tolerancia en las resistencias se han tomado dos muestras de más de 30 resistencias cada una con diferentes tolerancias, se determinó su distribución por medio de una prueba $X^2$ y se llegó a la conclusión de que se distribuyen de forma normal ver apéndice A.

# 3. SIMULACIÓN DE FUENTES DE ERROR

Para la simulación de las fuentes de error lo más conveniente es realizarla por métodos numéricos, siendo el más ampliamente utilizado el Método de Monte Carlo debido a la facilidad de implementación y su versatilidad, ya que su potencia yace en generación de números aleatorios, o bien pseudoaleatorios, y de la teoría axiomática de probabilidades.

## 3.1. Método de Montecarlo

Es un método numérico que permite resolver problemas matemáticos mediante la simulación de variables aleatorias. Este método tiene como origen en 1949, en el que apareció el artículo titulado *The Monte Carlo Method*. La creación de este método suele ligarse a los matemáticos norteamericanos J. von Neumann y S. Ulam. En la Unión Soviética los primeros artículos dedicados al Método de Montecarlo aparecieron en 1955 y 1956.

En primer lugar este método permite simular cualquier proceso cuya marcha depende de factores aleatorios. En segundo lugar, en muchos problemas matemáticos, que no tienen la menor relación con cuestiones aleatorias, se puede crear un modelo probabilístico artificial e incluso más de un modelo que permita resolver estos problemas.

El Método de Monte Carlo proporciona soluciones aproximadas a una gran variedad de problemas matemáticos posibilitando la realización de experimentos con muestreos de números pseudoaleatorios en una computadora. El Método es aplicable a cualquier tipo de problema, ya sea

estocástico o determinista, dicho en otras palabras, el cual considera o no incertidumbre.

La simulación de Monte Carlo es un método que permite calcular un parámetro $\theta = \mathrm{E}[\mathrm{h}(\mathrm{X})]$ donde $\mathrm{X} = (\mathrm{X}_1, \dots, \mathrm{X}_\mathrm{d})$ es un vector aleatorio perteneciente a $\mathbb{R}^\mathrm{d}$, $\mathrm{h}(\mathrm{X})$ es una función $\mathbb{R}^\mathrm{d} \to \mathbb{R}$ y $\mathrm{E}[\mathrm{h}(\mathrm{X})] < \infty$.

El método consiste en simular $\mathrm{X}_1, \dots, \mathrm{X}_\mathrm{n}$, cuyos valores son independientes e idénticamente distribuidos (i.i.d.) y estimar un determinado parámetro, según la ecuación 57:

$$\widehat{\theta_\mathrm{n}} = \frac{\mathrm{h}(\mathrm{X}_1) + \cdots + \mathrm{h}(\mathrm{X}_\mathrm{n})}{\mathrm{n}} \qquad (57)$$

Además por la Ley de los Números Grandes (la ley de los números grandes nos dice que en el límite probabilístico en el que el tamaño de una muestra tienda a infinito, el valor esperado del valor esperado de un número infinito de submuestras será el parámetro poblacional) se tiene que $\theta_\mathrm{n} \to \theta$ cuando $\mathrm{n} \to \infty$ con probabilidad de uno.

## 3.2. Generación de variables aleatorias

La habilidad de generar variables aleatorias uniformes en el intervalo $[0,1]$ es de importancia fundamental porque generalmente son la base para generar otras variables aleatorias. A las variables aleatorias uniformes en el intervalo de $[0,1]$ se les llama números aleatorios.

Los números aleatorios presentan diversas características que se debe tener en cuenta. Estos deben ser:

- Uniformemente distribuidos
- Estadísticamente independientes
- Reproducibles
- De periodo largo, es decir que se generen la mayor cantidad.
- Generados a través de un método rápido.
- Generados a través de un método que no requiera mucha capacidad de almacenamiento de computadora.

Para generarlos comúnmente se utiliza el método de congruencia lineal, este produce una secuencia de enteros, $Z_1, Z_2, \ldots$ entre cero y $m - 1$, conforme la relación recursiva mostrada en la ecuación 58:

$$Z_i = (aZ_{i-1} + c) \bmod m \tag{58}$$

Donde $m, a, c$ y $Z_0$ son números enteros no negativos, $Z_0$ es la semilla, $a$ el multiplicador, $c$ el incremento y $m$ el módulo.

Para generar los números pseudoaleatorios $U_1, \ldots, U_n, \ldots$, sea $U_i = Z_i / m$. Se puede notar que $U_i \in (0,1)$ para todo $i$.

### 3.3. Simulación de ruido blanco

Para la simulación del ruido blanco se toma en cuenta los supuestos de mencionados en el capítulo 2, de esta manera se generará una componente de ruido que se distribuirá normalmente con media igual a 0, varianza constante igual a $\sigma^2$.

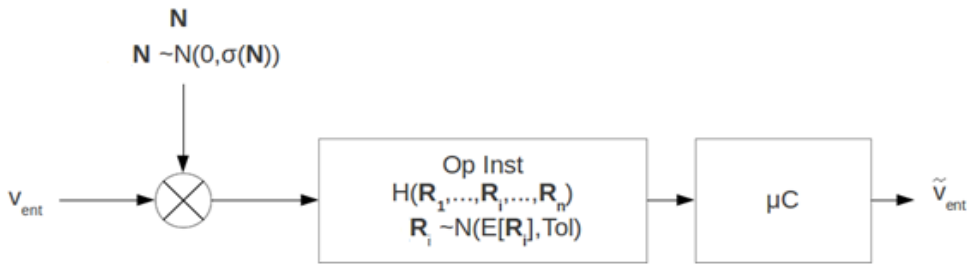
## 3.4. Simulación de tolerancia en resistencias

Para la simulación de la tolerancia en resistencias, se tomaron muestras de varias resistencias a diferentes tolerancias, esto para obtener su distribución la cual es Gaussiana, para tal efecto se utilizó una prueba $X^2$. Cabe resaltar que este resultado era de esperarse tomando en consideración la ley de los números grandes. Los datos se encuentran en el apéndice C.

## 3.5. Descripción general del sistema

La figura 6 muestra el sistema a simular, en este se observa un valor llamado $v_{ent}$ al cual de modo aditivo se contaminará con una señal de ruido $N$ distribuido normalmente, el cual entrará a un amplificador de instrumentación que tiene una función de transferencia $H$ el cual depende de las resistencias $R_1, R_2, \dots, R_n$ al igual que el ruido estas tienen una distribución normal con media el valor teórico esperado, y desviación estándar su tolerancia. Luego, estas señales entrarán a un microcontrolador el cual tendrá la función de realizar algún algoritmo el cual sea capaz de reducir las perturbaciones antes mencionadas y recuperar la señal de entrada $v_{ent}$.

Figura 6. **Descripción sistema**




Fuente: elaboración propia.

En las tablas I, II, III y IV se presentan los resultados de las simulaciones, se utilizó el paquete estadístico R, ver apéndice B, con distintas combinaciones entre tamaño de muestra, tolerancia de las resistencias y ruido blanco en los voltajes teóricos. Además se presentan los intervalos de confianza de la salida, y una prueba de hipótesis que determina si el valor de la media al valor teórico 2V y 3V para la salida considerando dos y tres operacionales respectivamente al 5 % de significancia. Referirse al apéndice C para para visualizar las simulaciones en términos de densidad de probabilidad.

Tabla I. **Amplificador de instrumentación con dos operacionales, n=30**

| Tol % | Error % | $E[V_0]$ | Intervalo de Confianza 95% | | p-value |
|---|---|---|---|---|---|
| 1 | 1 | 2,000 | 1,993 | 2,009 | 0,936 |
| 1 | 5 | 3,430 | -1,879 | 8,738 | 0,586 |
| 1 | 10 | 3,290 | -7,724 | 1,.304 | 0,812 |
| 10 | 1 | 2,057 | 1,968 | 2,146 | 0,203 |
| 10 | 5 | 4,839 | 0,551 | 9,126 | 0,186 |
| 10 | 10 | -1,164 | -11,541 | 9,213 | 0,538 |

Fuente: elaboración propia.

Tabla II. **Amplificador de instrumentación con dos operacionales, n = 1 000**

| Tol % | Error % | E[$V_0$] | Intervalo de Confianza 95% | | p-value |
|---|---|---|---|---|---|
| 1 | 1 | 2,035 | 1,861 | 2,210 | 0,692 |
| 1 | 5 | 2,000 | 1,993 | 2,007 | 0,991 |
| 1 | 10 | 1,579 | 0,702 | 2,456 | 0,347 |
| 10 | 1 | 1,992 | 1,823 | 2,170 | 0,930 |
| 10 | 5 | 1,705 | 0,806 | 2,603 | 0,520 |
| 10 | 10 | 3,222 | 1,491 | 5,154 | 0,157 |

Fuente: elaboración propia.

Tabla III. **Amplificadores de instrumentación con tres operacionales 1, n = 1 000**

| Tol % | Error % | E[$V_0$] | Intervalo de Confianza 95% | | p-value |
|---|---|---|---|---|---|
| 1 | 1 | 3,008 | 2,992 | 3,025 | 0,312 |
| 1 | 5 | 5,119 | -2,869 | 13,108 | 0,592 |
| 1 | 10 | 4,824 | -11,777 | 21,424 | 0,824 |
| 10 | 1 | 3,046 | 2,917 | 3,175 | 0,476 |
| 10 | 5 | 6,814 | 0,811 | 12,817 | 0,204 |
| 10 | 10 | -1,299 | -15,746 | 13,148 | 0,548 |

Fuente: elaboración propia.

Tabla IV. **Amplificadores de instrumentación con tres operacionales 2, n = 1 000**

| Tol % | Error % | E[$V_0$] | Intervalo de Confianza 95% | | p-value |
|---|---|---|---|---|---|
| 1 | 1 | 3,054 | 2,792 | 3,316 | 0,687 |
| 1 | 5 | 3,003 | 2,990 | 3,015 | 0,667 |
| 1 | 10 | 2,359 | 1,043 | 3,676 | 0,340 |
| 10 | 1 | 3,007 | 2,739 | 3,274 | 0,962 |
| 10 | 5 | 2,535 | 1,183 | 3,886 | 0,499 |
| 10 | 10 | 5,260 | 2,488 | 8,032 | 0,110 |

Fuente: elaboración propia

En vista de los resultados obtenidos se observa que estadísticamente no se rechaza la hipótesis nula (debido a que el valor del p-value es mayor que las significancia utilizadas, es posible rechazar la hipótesis nula, este valor representa una medida de probabilidad de la verosimilitud de la hipótesis sobre la cual se efectuó la prueba) de que el valor esperado de la respuesta del sistema con dos y tres operacionales es igual al valor teórico con las significancias usuales 1 %, 5 %, 10 %, además se observa que para cada sistema la tolerancia en cierta medida produce una variación con respecto al valor esperado bastante pequeña, viéndose reflejado de igual forma en los intervalos de confianza, a diferencia del efecto que produciría un ambiente ruidoso. En los capítulos siguientes se reducirá este efecto.

# 4. MICROCONTROLADORES

Son computadores digitales integrados en un chip que cuentan con un microprocesador o unidad de procesamiento central, una memoria para almacenar el programa, una memoria para almacenar datos y puertos de entrada/salida. A diferencia de los microprocesadores de propósito general, como los que se usan en los computadores PC, estos son unidades autosuficientes y más económicas.

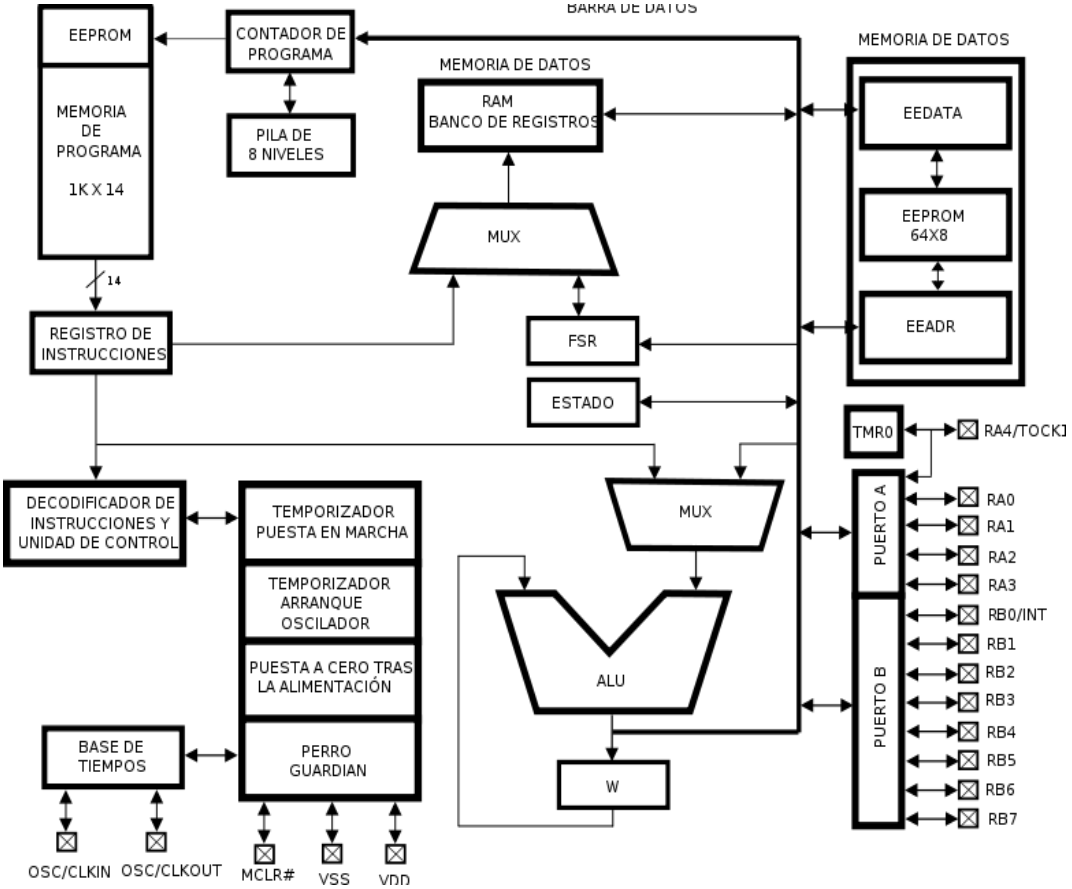
El funcionamiento de los microcontroladores está determinado por el programa almacenado en su memoria. Este puede escribirse en distintos leguajes de programación. Además, la mayoría de los microcontroladores actuales pueden reprogramarse repetidas veces.

Por las características mencionadas y su alta flexibilidad, los microcontroladores son ampliamente utilizados como el cerebro de una gran variedad de sistemas embebidos que controlan máquinas, componentes de sistemas complejos, como aplicaciones industriales de automatización y robótica, domótica, equipos médicos, sistemas aeroespaciales, e incluso dispositivos de la vida diaria como automóviles, hornos de microondas, teléfonos y televisores.

## 4.1. Características de los microcontroladores (µC)

Las principales características de los µC se muestran en la figura 7, además se presenta una breve descripción de las partes más importantes:

Figura 7. **Estructura de un microcontrolador**




Fuente: *Estructura del microcontrolador*. <http://es.wikiversity.org/wiki/Archivo:PIC_arquitecturadibujo.svg.>. Consulta: 12 de marzo de 2014.

- Unidad de Procesamiento Central (CPU): generalmente 4, 8, 16, 32 y hasta 64 bits con arquitectura Harvard o de von Neumann. Esta se encarga de direccionar la memoria de instrucciones, recibir la instrucción en curso, su decodificación y la ejecución de dicha instrucción, así como la búsqueda de los operandos y el almacenamiento del resultado.

- Memoria de programa: es una memoria ROM (*Read-Only Memory*), EPROM (*Electrically Programable ROM*), EEPROM (*Electrically Erasable/Programable ROM*) o Flash que almacena el código del programa.

- Memoria de datos: es una memoria RAM (*Random Access Memory*) que típicamente puede ser de 1, 2, 4, 8, 16, 32 kilobytes.

- Generador del reloj: usualmente un cristal de cuarzo de frecuencias que genera una señal oscilatoria de entre 1 a 40 MHz, o también resonadores o circuitos RC, además internamente algunos microcontroladores pueden tener un PLL (*Phase-Locked Loop*) que a su vez incrementará o disminuirá la frecuencia del oscilador.

- Interfaz de Entrada/Salida: puertos paralelos, seriales (*UART's, Universal Asynchronous Receiver/Transmitter*), $I^2C$ (*Inter-Integrated Circuit*), Interfaces de Periféricos Seriales (SPI's, *Serial Peripheral Interfaces*), Red de Area de Controladores (CAN, *Controller Area Network*), USB (*Universal Serial Bus*).

- Conversores Análogo-Digitales (A/D's, *Analog-to-Digital*) para convertir un nivel de voltaje en un cierto pin a un valor digital manipulable por el programa del microcontrolador.

- Moduladores por Ancho de Pulso (PWM, Pulse-Width Modulation) para generar ondas cuadradas de frecuencia fija pero con ancho de pulso modificable.

- Las interrupciones son señales que se generan internamente en el microcontrolador que detienen la ejecución normal del programa para ejecutar alguna subrutina de respuesta al evento. Una vez ejecutada la subrutina de interrupción la ejecución del programa continúa en el punto en que se encontraba antes de generarse la interrupción.

## 4.2. Proceso de desarrollo de aplicaciones

El proceso de desarrollo de una aplicación basada en microcontroladores se compone de las siguientes etapas principales, las cuales se explican en más detalle en las siguientes subsecciones.

- Desarrollo de software, en esta etapa corresponde la escritura y compilación/ensamblaje del programa que regirá las acciones del µC y los sistemas periféricos conectados a este.

- Programación del µC, en esta etapa el código de máquina correspondiente al programa desarrollado en la etapa anterior se descarga en la memoria del µC.

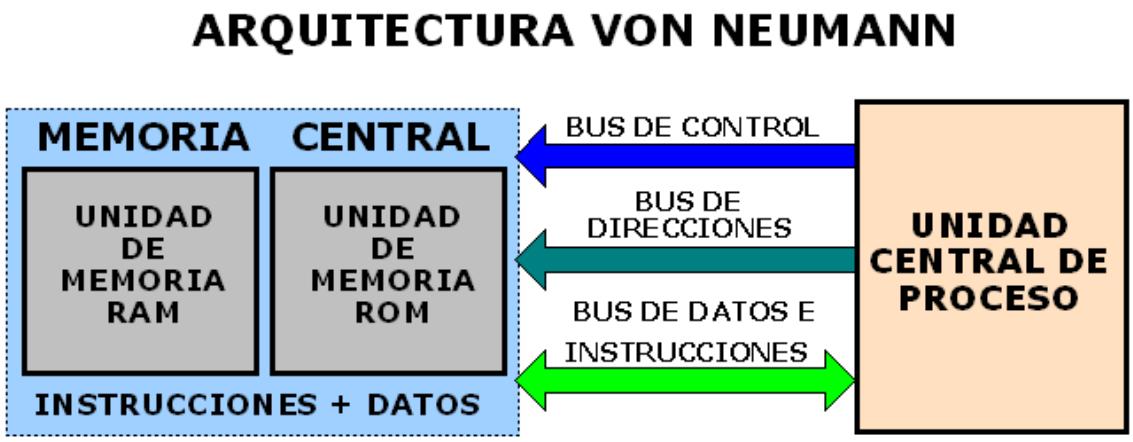
- Prueba y verificación, el µC debe conectarse al circuito base y someterse a pruebas para verificar el funcionamiento correcto del programa.

## 4.3. Arquitectura

Según la forma en que se distribuyen los buses de datos y direcciones se pueden nombrar dos distintos tipos de arquitectura:

- La arquitectura de von Neumann, tal como se muestra en la figura 8, se caracteriza por disponer de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único direcciones, datos y control.

Figura 8. **Arquitectura Von Neumann**

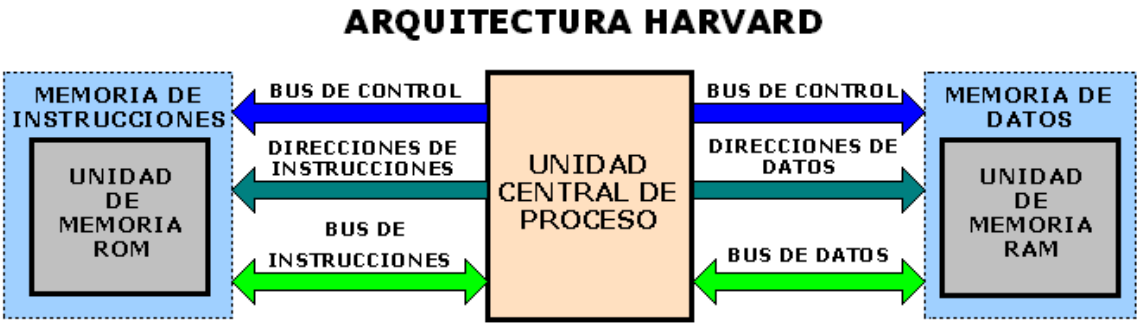



Fuente: *Arquitectura Von Neumann*.
<http://perso.wanadoo.es/pictob/imagenes/vonneumann.gif.> Consulta: 12 de marzo de 2014.

- La arquitectura Harvard, tal como se muestra en la figura 9, dispone de dos memorias independientes una, que contiene solo instrucciones y otra, solo datos. Ambas disponen de sus respectivos sistemas de buses

y es posible realizar operaciones de acceso de lectura o escritura simultáneamente en ambas memorias.

Figura 9. **Arquitectura Harvard**




Fuente: *Arquitectura Harvard*. <http://perso.wanadoo.es/pictob/imagenes/harvard.gif.> Consulta: 12 de marzo de 2014.

Además se pueden encontrar tres orientaciones en cuanto a la arquitectura tomando en cuenta la funcionalidad de los procesadores actuales:

CISC (Computadores de Juego de Instrucciones Complejo), este juego de instrucciones dispone de más de 80 instrucciones máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos ciclos para su ejecución. Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.

RISC (Computadores de Juego de Instrucciones Reducido), en estos procesadores el repertorio de instrucciones máquina es muy reducido y las instrucciones son simples y, generalmente, se ejecutan en un ciclo. La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del

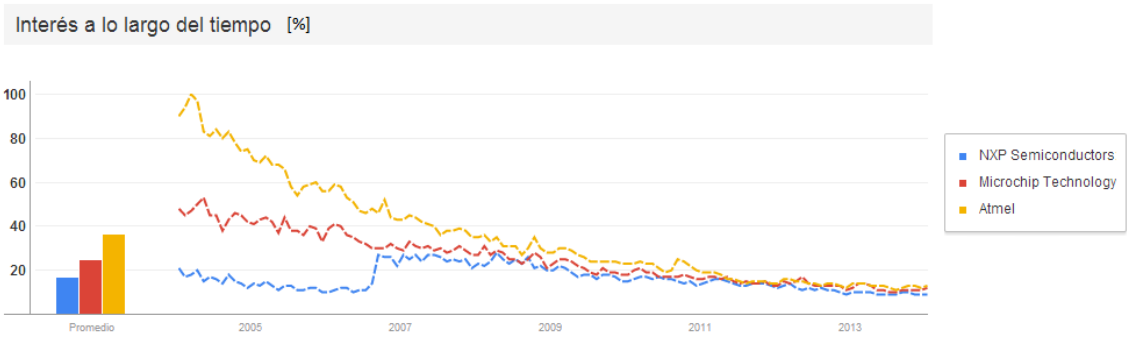
procesador. Siendo esta la tendencia actual de los microcontroladores para aplicaciones industriales.

SISC (Computadores de Juego de Instrucciones Específico), este tipo de microcontroladores están destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es “específico”, o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista.

## 4.4. Familias de microcontroladores

Actualmente existen varias familias de fabricantes de microcontroladores, por lo que se efectúo una búsqueda de tendencias tomando en consideración el interés sobre determinadas familias de microcontroladores bajo el criterio de persistencia del interés a lo largo del tiempo, para realizar una selección en base a este criterio, en la figura 10 se presenta los resultados.

Figura 10. **Tendencias de búsqueda de marcas de microcontroladores**




Fuente: *Microcontroller Trends*. <www.google.com/trends/explore#q=%2Fm%2F0gp7yz%2C%20%2Fm%2F0c5zfd%2C%20%2Fm%2F01521p&cmpt=q.> Consulta: 22 de enero de 2014.

### 4.4.1. Microcontrolador PIC

Los microcontroladores PIC son una familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instruments.

El PIC original se diseñó para ser usado con la nueva CPU de 16 bits CP16000. Siendo en general una buena CPU, esta tenía malas prestaciones de E/S, luego se lanzó el PIC de 8 bits que se desarrolló en 1 975 para mejorar el rendimiento del sistema quitando peso de E/S a la CPU. El PIC utilizaba microcódigo simple almacenado en ROM para realizar estas tareas este se trata de un diseño RISC que ejecuta una instrucción cada 4 ciclos del oscilador.

En 1985, dicha división de microelectrónica de General Instruments se convirtió en una filial y el nuevo propietario canceló casi todos los desarrollos, que para esas fechas la mayoría estaban obsoletos. El PIC, sin embargo, se mejoró con EPROM para conseguir un controlador de canal programable.

Hoy en día multitud de PICs vienen con varios periféricos incluidos como lo son módulos de comunicación serie, UARTs, núcleos de control de motores, etc., y con memoria de programa desde 512 a 32 000 palabras (una palabra corresponde a una instrucción en ensamblador, y puede ser 12, 14 o 16 bits, dependiendo de la familia específica de PICmicro).

Actualmente esta familia de microcontroladores cuenta con las características siguientes:

- Núcleos de CPU de 8/16 bits con Arquitectura Harvard modificada

- Memoria Flash y ROM disponible desde 256 bytes a 256 kilobytes
- Puertos de E/S (típicamente 0 a 5,5 voltios)
- Temporizadores de 8/16 bits
- Tecnología Nanowatt para modos de control de energía
- Periféricos serie síncronos y asíncronos: USART, AUSART, EUSART
- Conversores analógico/digital de 8,10 y 12 bits
- Comparadores de tensión
- Módulos de captura y comparación PWM
- Controladores LCD
- Periférico MSSP para comunicaciones I²C, SPI, y I²S
- Memoria EEPROM interna con duración de hasta un millón de ciclos de lectura/escritura
- Periféricos de control de motores
- Soporte de interfaz USB
- Soporte de controladores Ethernet, CAN, LIN, Irda, etc.

Con respecto a las gamas de MCU's PIC en la actualidad se pueden diferenciar tres gamas, la básica *Linebase*, la de medio rango *Mid Range* y la de alto desempeño *High Performance*.

Además de estas tres gamas se pueden encontrar los dsPIC's cuya producción comenzó en gran escala a finales de 2004. Son los primeros con bus de datos inherente de 16 bits, incorporan todas las posibilidades de los anteriores y añaden varias operaciones de DSP implementadas en hardware, como multiplicación con suma de acumulador (*multiply-accumulate*, o MAC), *barrel shifting*, *bit reversion* o multiplicación 16x16 bits.

En 2007, Microchip Technology lanza los microcontroladores de 32 bits con una velocidad de procesamiento de 1.5 DMIPS/MHz con capacidad HOST

USB. Estos MCUs permiten un procesamiento de información increíble con un núcleo de procesador de tipo M4K.

### 4.4.2. Microcontrolador ARM

El diseño del microcontrolador ARM comenzó en 1 983 como un proyecto de desarrollo en la empresa Acorn Computers. Roger Wilson y Steve Furber lideraban el equipo, cuya meta era, originalmente, el desarrollo de un procesador avanzado, pero con una arquitectura similar a la del MOS 6502. El equipo terminó el diseño preliminar y los primeros prototipos del procesador en 1985, al que llamaron ARM1. La primera versión utilizada comercialmente se bautizó como ARM2 y se lanzó en 1986.

La arquitectura del ARM2 posee un bus de datos de 32 bits y ofrece un espacio de direcciones de 26 bits, junto con 16 registros de 32 bits. Uno de estos registros se utiliza como contador de programa, aprovechándose sus cuatro bits superiores y los dos inferiores para contener los flags de estado del procesador.

El ARM2 es probablemente el procesador de 32 bits útil más simple del mundo, ya que posee sólo 30 000 transistores. Su simplicidad se debe a que no está basado en microcódigo, ya que este sistema suele ocupar en torno a la cuarta parte de la cantidad total de transistores usados en un procesador y a que, como era común en aquella época, no incluye caché. Gracias a esto, su consumo en energía es bastante bajo.

Su sucesor, el ARM3, incluye una pequeña memoria caché de 4 kb, lo que mejora los accesos a memoria repetitivos. A finales de los años 80, Apple Computer comenzó a trabajar con Acorn en nuevas versiones del núcleo ARM.

En los noventas, el trabajo anterior derivó en el ARM6 presentado en 1991. Apple utilizó el ARM 610, basado en el ARM6, como procesador básico para su innovador PDA, el Apple Newton. Por su parte, Acorn lo utilizó en 1994 como procesador principal en su RiscPC. El núcleo mantuvo su simplicidad a pesar de los cambios: en efecto, el ARM2 tiene 30 000 transistores, mientras que el ARM6 solo cuenta con 35 000. La idea era que el usuario final combinara el núcleo del ARM con un número opcional de periféricos integrados y otros elementos, pudiendo crear un procesador completo a la medida de sus necesidades.

La mayor utilización de la tecnología ARM se alcanzó con el procesador ARM7TDMI, con millones de unidades en teléfonos móviles y sistemas de videojuegos portátiles. DEC licenció el diseño, lo cual generó algo de confusión debido a que ya producía el DEC Alpha, y creó el StrongARM. Con una velocidad de reloj de 233 MHz, este procesador consumía solo 1 W de potencia.

Luego esta tecnología pasó posteriormente a manos de Intel, como fruto de un acuerdo jurídico, que la integró en su línea de procesadores Intel i960 e hizo más ardua la competencia. Freescale (una empresa que derivó de Motorola en el 2004), IBM, Infineon Technologies, OKI, Texas Instruments, Nintendo, Philips, VLSI, Atmel, Sharp, Samsung y STMicroelectronics también licenciaron el diseño básico del ARM. El diseño del ARM se ha convertido en uno de los más usados del mundo, desde discos duros hasta juguetes. Hoy en día, cerca del 75 % de los procesadores de 32 bits poseen este chip en su núcleo.

### 4.4.3. Microcontroladores AVR

La gama de microcontroladores de la empresa Atmel son llamados AVR (*Advanced Virtual RISC*) cuya arquitectura fue concebida por dos estudiantes en el Norwegian Institute of Technology, Alf-Egil Bogen y Vegard Wollan, posteriormente refinada y desarrollada en Atmel Norway, la empresa subsidiaria de Atmel, donde los diseñadores originales trabajaron en estrecha colaboración con los autores de compiladores de IAR Systems para asegurar que el conjunto de instrucciones proporcionado este optimizado para la compilación más eficaz de los lenguajes de programación de alto nivel. Este microcontrolador fue diseñado para la ejecución de programas escritos en código C compilado.

Los microcontroladores AVR se clasifican en seis grandes grupos:

- ATtinyAVR, la serie más pequeña la cual contienen de 0,5 a 16 kB de memoria de programa, y un conjunto limitado de periféricos.

- ATmegaAVR, esta serie contienen de 4 a 512 kB de memoria de programa, además de un conjunto de periféricos amplia.

- ATxmega, esta serie contiene de 16 a 384 kB de memoria de programa, además de prestaciones amplias referentes a los periféricos.

- ATmegaAVR con características especiales no encontradas en el resto de los miembros de las familias, tales como, controlador LCD, controlador USB, PWM, CAN, etc.

- FPSLIC se refieren a microcontroladores AVR con la inclusión de FPGA (*Field Programmable Gate Array*).

- AVRs de 32 bits

## 4.5. Diferencias entre microcontroladores

Al momento de realizar alguna aplicación es necesario tomar en cuanto algunas de sus características más relevantes, ya que esto impactará en la relación costo-beneficio del mismo, a continuación se presentan algunos criterios para efectuar esta selección:

- Velocidad de procesamiento de datos, este se debe tener en cuenta cuando el tiempo es un factor crítico, por lo que se debe asegurar la selección de un dispositivo suficientemente rápido para tal propósito.

- Ancho de palabra/Precisión de los datos, en ambos casos el criterio de diseño debe ser seleccionar el microcontrolador de menor ancho de palabra que satisface los requerimientos de la aplicación. Usar un microcontrolador de 4 bits supondrá una reducción en los costes importante, mientras que uno de 8 bits puede ser el más adecuado si el ancho de los datos es de un byte. Los microcontroladores de 16 y 32 bits o bien de punto flotante, debido a su elevado coste, deben reservarse para aplicaciones que requieran altas prestaciones

- Entradas/Salidas, un criterio suficiente para determinar las necesidades de E/S del sistema sería conocer de manera conveniente el diagrama de bloques del mismo, de tal forma que sea sencillo identificar la cantidad y tipo de señales a controlar. Una vez realizado este análisis puede ser necesario añadir periféricos externos o cambiar a otro microcontrolador más adecuado a ese sistema.

- Consumo, ya que algunos productos que incorporan microcontroladores están alimentados con baterías. Lo más conveniente en un caso como éste puede ser que el microcontrolador esté en estado de bajo consumo pero que despierte ante la activación de una señal de interrupción y ejecute el programa adecuado para procesarla.

- Memoria, en cuanto a la cantidad de esta necesaria se debe hacer una estimación de cuánta memoria volátil y no volátil es necesaria y si es conveniente disponer de memoria no volátil modificable.

- Diseño de la placa, esta determinará el tipo de empaquetamiento que se utilizará e impactará en la selección de un microcontrolador concreto que condicionará el diseño de la placa de circuitos.

- Lenguaje de Programación, en general esta se efectuará en un lenguaje de alto nivel (C, Pascal, Basic), su elección dependerá de los requerimientos del programador ya que estos afectarán el tamaño del código final al ser uno más óptimo que el resto.

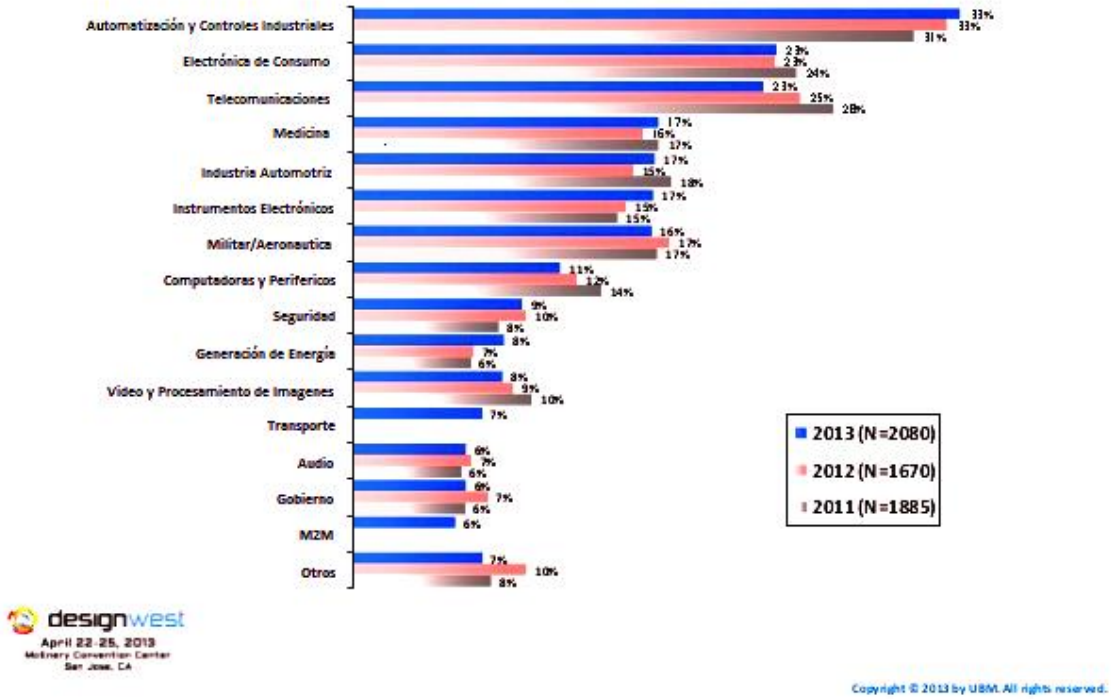
- Hardware de Programación, es necesario determinar si el hardware es específico para la programación, o bien si existe algún circuitos programadores universales que te permiten programar varias marcas/modelos con distinta cantidad de pines sin hacer ningún cambio.

## 4.6. Mercado actual de los microcontroladores

Para determinar la segmentación del mercado actual de microcontroladores se consultó la encuesta por la empresa United Business

Media, UBM, realizada a nivel mundial[2]. La figura 11 muestra la segmentación del mercado tomando en cuenta el sector en el cual se utilizan los microcontroladores.

Figura 11. **Comparativa de uso de microcontroladores según sector**



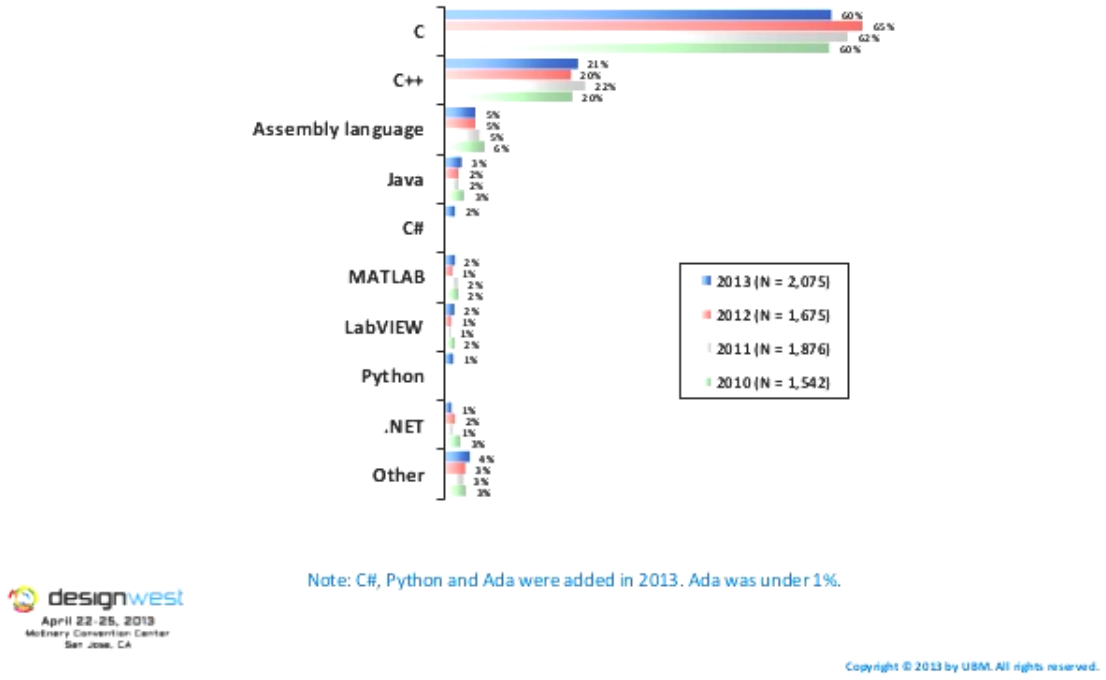

Fuente: 2013 *Embedded Market Study*. <http://image.slidesharecdn.com/2013 embeddedmarketstudyfinal-130328190555-phpapp01/95/slide-10-638.jpg?cb=1364515685.> Consulta: 3 de febrero de 2014.

Posteriormente se revisó las tendencias en cuanto a lenguaje de programación utilizado, ya que, si bien teóricamente es posible realizar gran cantidad de optimizaciones utilizando código assembler, en la realidad es

[2] *Embedded Market Study*. <http://www.slideshare.net/MTKDMI/2013-embedded-market-study-final.> [Consulta: 3 de febrero de 2014].

bastante impráctico y representa costos elevados en cuanto al tiempo de desarrollo, de igual forma lenguajes de programación como BASIC los cuales al momento de realizar la compilación, su resultado no es tan bueno, por tanto pueden producir incrementos en el tamaño de la memoria flash del microcontrolador a elegir. Según la encuesta realizada por la compañía antes mencionada la cual se muestra en la figura 12, el lenguaje de programación de mayor uso a nivel mundial es C, por tanto la programación se realizará en este lenguaje de modo que sea competitivo.

Figura 12. **Comparativa lenguajes de programación más utilizados**




Fuente: 2013 *Embedded Market Study.* <http://image.slidesharecdn.com/2013embeddedmarketstudyfinal-130328190555-phpapp01/95/slide-20-638.jpg?cb=1364515685.> Consulta: 3 de febrero de 2014.

Para la selección de microcontroladores se tomaron los circuitos impresos fabricados por la empresa ETT en modo *stamp*, debido a la vasta oferta de microcontroladores actuales disponibles y de sus diversas características igualmente amplias, se consideró adoptar una selección en base a los modelos presentados por la electrónica FUTURLEC[3] cuyo costo era el menor para las tres distintas marcas seleccionadas. En la tabla V se presenta una comparativa de las características más relevantes para los distintos modelos de microcontroladores utilizados.

Tabla V. **Comparativa entre microcontroladores**

| Marca | MCU | Costo | Flash | Registros | Vel. Máx. | Modo Programación | E/S |
|---|---|---|---|---|---|---|---|
| Atmel | ATMega128 | $22,90 | 128kb | 8 bits | 16MHz | SPI & JTAG | 53 |
| Microchip | PIC18F87 | $32,90 | 128kb | 8/16 bits | 10MHz | ICD2 & PICKit | 65 |
| NXP ARM | LPC2119 | $29,90 | 128kb | 16/32 bits | 58,98MHz | RS232 & JTAG | 46 |

Fuente: elaboración propia.

Además se presenta la lista de los compiladores más utilizados, para estos fabricantes:

- Keil C Compiler[4]
- Micro IDE C Compiler[5]
- WinAVR C Compiler[6]

[3] *Futurlec Electronics*. <http://www.futurlec.com.> Consulta: 4 de febrero de 2014.

[4] *Keil uVision*. <http://www.keil.com/dd/chip/3646.html.> Consulta: 4 de febrero de 2014.

[5] *Micro IDE*. <http://www.micro-ide.com/.> Consulta: 4 de febrero de 2014.

[6] *WinAVR*. <http://winavr.sourceforge.net/.> Consulta: 4 de febrero de 2014.

- AVR Studio IDE[7]
- CSS PIC C Compiler[8]
- Mikroelektronika C Compiler[9]
- Microchip PIC C Compiler[10]

## 4.7. Implementación de algoritmo

Tomando como punto de partida la discusión realizada en el capítulo 3, al efectuar la simulación de las fuentes de error, estadísticamente es posible realizar una aproximación del valor real tomando su valor esperado; además debido a la naturaleza aleatoria de la señal a muestrear es válido considerar una medida de dispersión, como lo es la desviación estándar a modo de tener una idea de que tanto se logró acercarse al valor teórico.

El algoritmo implementado consiste en una serie de 100 mediciones, separadas por un período $T$, estas mediciones contienen una componente de tendencia que se extraerá mediante un promedio móvil y se determinará su dispersión.

[7] *AVR Studio IDE.* <http://www.atmel.com/dyn/products/tools_card.asp?family_id=607&family_name=AVR+8%2DBit+RISC+&tool_id=2725.> Consulta: 4 de febrero 2014.

[8] *CSS Compiler*. <http://www.ccsinfo.com/content.php?page=compilers.> Consulta: 4 de febrero de 2014.

[9] *MicroC Compiler*. <http://www.mikroe.com/mikroc/pic/.> Consulta: 4 de febrero de 2014.

[10] *MPLAB XC Compiler*. <http://www.microchip.com/pagehandler/en_us/devtools /mplabxc/.> Consulta: 4 de febrero de 2014.

Figura 13. **Diagrama de flujo de algoritmo**


Fuente: elaboración propia.

En el apéndice D, E, F, se presenta el código utilizado para los programas PICC (CSS Compiler), WINAVR y Keil uVision, utilizando el lenguaje de programación C, para todos los casos. Estos microcontroladores obtuvieron sus datos en base a la simulación efectuada en los capítulos anteriores, tomando en cuenta las limitaciones físicas, tal como lo son la saturación de los amplificadores de instrumentación, las pérdidas de información mediante la cuantización, al igual que el hecho que no se trabajó utilizando punto flotante para las operaciones, esto con el afán de maximizar el tiempo de ejecución.

## 4.8. Resultados

En las tablas VI, VII, se muestran los resultados obtenidos luego de simular las fuentes de error y su recuperación mediante el algoritmo propuesto, considerando las condiciones de saturación de los amplificadores operacionales, estando el voltaje de entrada a cada convertidor A/D limitado al

Vdd máximo de cada modelo de microcontrolador, existiendo en este caso dos tipos, para Vdd = 3.3V (LPC2119) y Vdd = 5V (Atmega128 y PIC18F8722), para el voltaje diferencial de 1V entre 0 y 5V, el tamaño de la muestra utilizada fue de 100 datos simulados.

Tabla VI. **Resultados de algoritmo para un ADC de 10bits con Vdd=5V**

| Tolerancia [%] | Error [%] | Voltaje de Salida [V] | |
|---|---|---|---|
| | | 2 OpAmp | 3 OpAmp |
| 1 | 1 | 1,860±0,288 | 2,378±0,327 |
| 1 | 5 | 2,451±0,562 | 2,568±0,581 |
| 1 | 10 | 2,578±0,381 | 2,651±0,396 |
| 10 | 1 | 2,217±0,244 | 2,827±0,254 |
| 10 | 5 | 2,49±0,391 | 2,661±0,386 |
| 10 | 10 | 2,368±0,61 | 2,456±0,483 |

Fuente: elaboración propia.

Tabla VII. **Resultados de algoritmo para un ADC de 10bits con Vdd=3.3V**

| Tolerancia [%] | Error [%] | Voltaje de Salida [V] | |
|---|---|---|---|
| | | 2 OpAmp | 3 OpAmp |
| 1 | 1 | 2,392±0,327 | 2,827±0,323 |
| 1 | 5 | 2,451±0,562 | 2,568±0,581 |
| 1 | 10 | 2,578±0,381 | 2,651±0,396 |
| 10 | 1 | 2,803±0,244 | 3,291±0,248 |
| 10 | 5 | 2,651±0,395 | 2,719±0,405 |
| 10 | 10 | 2,465±0,464 | 2,524±0,483 |

Fuente: elaboración propia.

En la tabla VIII se presentan los errores en los resultados obtenidos.

Tabla VIII. **Errores de resultado**

| Vdd [V] | Tolerancia [%] | Error [%] | Error Resultado 2 OpAmps [%] | Coef. Variabilidad 2 OpAmps[%] | Error Resultado 3 OpAmps [%] | Coef. Variabilidad 3 OpAmps [%] |
|---|---|---|---|---|---|---|
| 5 | 1 | 1 | 7 % | 15,48 % | 20,73 % | 13,75 % |
| 5 | 1 | 5 | 22,55 % | 22,93 % | 14,4 % | 22,62 % |
| 5 | 1 | 10 | 28,9 % | 14,78 % | 11,63 % | 14,94 % |
| 5 | 10 | 1 | 10,85 % | 11,01 % | 5,77 % | 8,98 % |
| 5 | 10 | 5 | 24,5 % | 15,7 % | 11,3 % | 14,51 % |
| 5 | 10 | 10 | 18,4 % | 25,76 % | 18,13 % | 19,67 % |
| 3,3 | 1 | 1 | 19,6 % | 13,67 % | 5,77 % | 11,43 % |
| 3,3 | 1 | 5 | 28,4 % | 22,59 % | 11,63 % | 21,88 % |
| 3,3 | 1 | 10 | 32,8 % | 15,06 % | 10,67 % | 14,93 % |
| 3,3 | 10 | 1 | 40,15 % | 8,7 % | 9,7 % | 7,54 % |
| 3,3 | 10 | 5 | 32,55 % | 14,9 % | 9,37 % | 14,9 % |
| 3,3 | 10 | 10 | 23,25 % | 18,82 % | 15,87 % | 19,14 % |

Fuente: elaboración propia.

# 5. EVALUACIÓN EFICIENCIA DE ALGORITMO

A partir de los resultados obtenidos en el capítulo anterior se determinó una selección de microcontrolador y de configuración de amplificador de instrumentación que presentara menor dispersión, a la vez mayor exactitud, con respecto al valor teórico esperado.

- Se observa que según la tabla VIII existe una preferencia con respecto a la configuración de amplificador de instrumentación a utilizar con referencia al voltaje de polarización (Vdd) del microcontrolador, encontrándose que se obtiene una mejor estimación cuando se utiliza la configuración de dos amplificadores operacionales y un microcontrolador con un Vdd=5V, en caso de que el Vdd=3.3V el circuito óptimo resulta en la implementación de una configuración de tres amplificadores operacionales. Ahora bien, tomando en consideración el criterio de consumo y de costos de circuitería, lo óptimo sería minimizar la cantidad de amplificadores operacionales involucrados, por tanto es recomendable utilizar la configuración del amplificador de instrumentación con dos amplificadores operacionales.

- Considerando la desviación estándar producto de las perturbaciones provocadas al sistema, se observa que la influencia del error es mucho mayor que la de la tolerancia de las resistencias. Resultado que persistió desde la función de transferencia del amplificador de instrumentación. Con respecto al coeficiente de variabilidad es posible considerar la implementación de resistencias de una mayor tolerancia, si el ambiente no es tan hostil, o bien, se sabe de antemano que el circuito se

encontrará en un ambiente poco ruidoso, o bien tomar las consideraciones necesarias como blindajes, filtros, etc., ya que con esto se reducen los costos de circuitería. Además se podría compensar con una constante que reduzca el error de medición por medio de una implementación por software, si persiste la condición observada de mayor precisión.

- Con respecto a la selección de microcontrolador, debido a que el algoritmo implementado únicamente utiliza los periféricos ADC y UART, sería imposible seleccionar un microcontrolador en base a un criterio de maximización de alguna propiedad o prestación, como lo son ancho de palabra, memoria, velocidad de procesamiento de datos, entradas y salidas, etc., por lo que se optó a elegir el de menor precio que es el microcontrolador AVR, aunque este tuvo su contraparte de un curva de aprendizaje del lenguaje de programación mucho más lenta con respecto a los otros dos microcontroladores.

- Finalmente, en las tablas IX, X y XI se muestran los costos de la implementación de cada circuito, por lo que en base a este análisis el óptimo sería elegir el microcontrolador AVR con la configuración de dos amplificadores operacionales con resistencias de 10 %, a sabiendas que la curva de aprendizaje para la implementación del algoritmo planteado será mucho mayor que el resto de microcontroladores.

Tabla IX. **Costo de implementación en AVR**

| | |
|---|---|
| Atmega128 | $22,90 |
| 4*Resistencias 10 % | $0,40 |
| 2*OpAmp LM741 | $0,44 |
| **Total** | **$23,74** |

Fuente: elaboración propia.

Tabla X. **Costo de implementación ARM**

| | |
|---|---|
| LPC2119 | $29,90 |
| 7*Resistencias 10 % | $0,70 |
| 3*OpAmp LM741 | $0,66 |
| **Total** | **$31,26** |

Fuente: elaboración propia.

Tabla XI. **Costo de implementación en PIC**

| | |
|---|---|
| PIC18F8722 | $32,90 |
| 4*Resistencias 10 % | $0,40 |
| 2*OpAmp LM741 | $0,44 |
| **Total** | **$33,74** |

Fuente: elaboración propia.

# CONCLUSIONES

1. Debido a la naturaleza estocástica de las perturbaciones de las fuentes de error en electrónica, la teoría de probabilidades resulta útil para su caracterización, previendo solidez sobre los diversos análisis que se pueden llevar a cabo.

2. Las técnicas de simulación de MonteCarlo permiten romper con los supuestos de idealidad de los componentes electrónicos.

3. Las técnicas de simulación de MonteCarlo permiten distorsionar las señales mediante fuentes de error, tal como lo es el ruido blanco.

4. El diseño de un algoritmo de reducción de fuentes de error de tipo estadístico se simplifica al realizar pruebas de hipótesis y experimentos simulados.

5. El diseño de circuitería está ligado a la implementación de un algoritmo eficiente y no deben considerarse como procesos separados.

6. En el mercado actual de microcontroladores, efectuar un análisis costo beneficio implica la selección de una circuitería adecuada que minimice costos, un lenguaje de programación y un algoritmo eficiente para la solución de determinado problema, y una curva de aprendizaje lo suficientemente corta para ser competitivo.

# RECOMENDACIÓNES

1. Utilizar software libre para la reducción de costos a largo plazo como lenguaje de alto nivel para compilación, por ejemplo para el caso de AVR[11], se puede utilizar avrdude, para GNU ARM tools[12], o bien SDCC[13]

2. La implementación de otros algoritmos de promedio móvil como suavización exponencial.

3. La implementación de algún filtro para obtener tendencia tal como el Hodrick-Prescott.

4. Utilizar el microcontrolador AVR de Atmel si se tiene proyectado la elaboración de varios proyectos en un largo plazo.

5. Utilizar el microcontrolador PIC de Microchip para proyectos de corto plazo y recreacionales, ya que debido a sus prestaciones se reduce el tiempo de desarrollo.

[11] *Avrdud*e. <http://nongnu.org/avrdude.> Consulta: 11 de febrero de 2014.

[12] *Gcc ARM*. <http://launchpad.net/gcc-arm-embedded.> Consulta: 11 de febrero de 2014.

[13] *SDCC*. <http://sdcc.sourceforge.net.> Consulta: 11 de febrero de 2014.

# BIBLIOGRAFÍA


1. *AVR GCC Tutorial*. [en línea] <http://www.8051projects.net/files/public/1242392126_2569_FT22383_avr_tutorial.pdf.> [Consulta: 11 de Febrero de 2014.]

2. BERGER, Marsha. *Scientific computing.* [en línea] <http://www.cs.nyu.edu/courses/fall06/G22.2112-001/MonteCarlo.pdf.> [Consulta: 29 de Agosto de 2013.]

3. COUGHLIN, Robert; DRISCOLL, Frederick. *Amplificadores operacionales y circuitos integrados lineales*. 3a ed. México: Prentice Hall, 1999. 516 p.

4. HAUGH, Martin. *Monte Carlo simulations.* [en línea]. <http://www.columbia.edu/~mh2078/MCS04.html.> [Consulta: 03 de Noviembre de 2013.]

5. IBAÑEZ, Javier. *Sistemas tolerantes a fallos*. [en línea] <http://www.infor.uva.es/~bastida/Arquitecturas%20Avanzadas/Tolerant.pdf.> [Consulta: 29 de agosto de 2013.]

6. LEE, T. H. IC Op-Amps through the ages, history of the operational amplifiers. [en línea]. <http://webapps.calvin.edu/~pribeiro/courses/engr332/Handouts/ho18opamp.pdf.> [Consulta: 18 de agosto de 2013.]

7. MARONE, José. *Microcontroladores atmel* [En línea]. <http://www.exa.unicen.edu.ar/catedras/tmicrocon/Material/3_Overview_Microcontroladores_ATMEL.pdf.> [Consulta: 22 de enero de 2014.]

8. METROPOLIS, N. *The beggining of the MonteCarlo Method*. [en línea]. <http://library.lanl.gov/cgi-bin/getfile?00326866.pdf.> [Consulta: 22 de junio de 2013.]

9. PAPOULIS, Athanasios. *Probability, random variables, and stochastic processes.* 2a ed. Estados Unidos: McGraw-Hill, 1984.

10. PÉREZ, Miguel; et al. *Instrumentación electrónica*. 2a ed. España: Thomson, 2004. 862 p.

11. SALINAS, Eduardo. *Microcontroladores.* [en línea] <http://electricidad.utpuebla.edu.mx/Manuales%20de%20asignatura/4to%20cuatrimestre/Microcontroladores.pdf.> [Consulta: 18 de diciembre de 2013.]

12. SEDRA, Adel; SMITH, Kenneth. *Microelectronic circuits*. a ed. Estados Unidos: Oxford University Press, 2004. 1 283 p.

13. SÓBOL, I. M. *Método de Montecarlo*. 2a ed. Perú: MIR, 1983, 78 p.

14. TAUB, Herbert; SCHILLING, Donald. *Principles of communication systems.* 2a ed. Estados Unidos: McGraw-Hill, 1991. 759 p.

15. TOSINI, Marcelo; MARONE, José; GOÑI, Enrique. *Arquitectura y organización de un microcontrolador genérico*. [en línea] <http://www.exa.unicen.edu.ar/catedras/tmicrocon/Material/1_introduccion_a_los_ucontroladores.pdf.> [Consulta: 18 de diciembre de 2013.]

16. TORRES-TORRITI, Miguel. *Tutorial de microcontroladores PIC.* [en línea] <http://web.ing.puc.cl/~mtorrest/downloads/pic/tutorial_pic.pdf.> [Consulta: 18 de diciembre de 2013.]

17. VALLEJO, Horacio. *Microcontroladores atmel*. [en línea] <http://www.clubse.com.ar/DIEGO/NOTAS/2/nota18.htm.> [Consulta: 22 de enero de 2014.]

18. WILLIANSON, Andy. *Random vectors.* [en línea] <http://andywilliamson.org/_/wp-content/uploads/2010/04/White-Noise.pdf.> [Consulta: 22 de junio de 2013.]

# APÉNDICES

## APÉNDICE A: MUESTRA DE RESISTENCIAS

Tabla I. **Muestra de Resistencias de 1kΩ con 1 % de Tolerancia**

| Resistencias [kΩ] | | | | | |
|---|---|---|---|---|---|
| 0,991 | 0,991 | 0,991 | 0,991 | 0,992 | 0,993 |
| 0,993 | 0,993 | 0,993 | 0,994 | 0,994 | 0,994 |
| 0,995 | 0,995 | 0,995 | 0,992 | 0,996 | 0,996 |
| 0,994 | 0,994 | 0,992 | 0,994 | 0,994 | 0,997 |
| 0,998 | 0,992 | 0,992 | 0,994 | 0,995 | 0,992 |
| 0,996 | | | | | |

Fuente: elaboración propia.

Tabla II. **Muestra de Resistencias de 1kΩ con 10 % de Tolerancia**

| Resistencias [kΩ] | | | | | |
|---|---|---|---|---|---|
| 0,975 | 0,985 | 0,985 | 0,985 | 0,985 | 0,990 |
| 0,992 | 0,992 | 0,992 | 0,992 | 0,994 | 0,994 |
| 0,995 | 0,995 | 0,996 | 1,000 | 1,002 | 1,002 |
| 1,002 | 1,004 | 1,016 | 1,027 | 1,007 | 1,009 |
| 1,011 | 0,997 | 0,997 | 0,998 | 0,998 | 0,999 |
| 0,985 | 0,986 | 0,986 | 0,986 | 0,990 | 0,993 |
| 0,993 | 0,993 | 0,993 | 0,994 | 1,004 | 1,007 |

Fuente: elaboración propia.

## APÉNDICE B: CÓDIGO EN R PARA SIMULACIÓN DEL SISTEMA

```
# El siguiente código simula la cantidad de experimentos diferentes indicados por la variable muestra

# Ingreso de Valores
resistencia <- 1000
muestra <- 100
tolerancia <- 1
tolerancia_v1 <- 1
tolerancia_v2 <- 1
valor_v1 <- 1
valor_v2 <- 0
significancia <- 5

# Generación de Distribuciones para Variables
# Al ser números pseudoaleatorios se hace una preselección del valor que generará la distribución posteriormente, este se toma de una distribución uniforme en el rango establecido por la tolerancia
set.seed(runif(n=1,min=resistencia - resistencia*tolerancia/100,max=resistencia + resistencia*tolerancia/100))
# Se genera una distribución uniforme con media el valor de resistencia teórico, y con desviación estandár la tolerancia especificada.
r1 <- rnorm(n=muestra,sd=resistencia*tolerancia/100,m=resistencia)
set.seed(runif(n=1,min=resistencia - resistencia*tolerancia/100,max=resistencia + resistencia*tolerancia/100))
r2 <- rnorm(n=muestra,sd=resistencia*tolerancia/100,m=resistencia)
set.seed(runif(n=1,min=resistencia - resistencia*tolerancia/100,max=resistencia + resistencia*tolerancia/100))
```

```
r3 <- rnorm(n=muestra,sd=resistencia*tolerancia/100,m=resistencia)
set.seed(runif(n=1,min=resistencia - resistencia*tolerancia/100,max=resistencia
+ resistencia*tolerancia/100))
r4 <- rnorm(n=muestra,sd=resistencia*tolerancia/100,m=resistencia)
set.seed(runif(n=1,min=resistencia - resistencia*tolerancia/100,max=resistencia
+ resistencia*tolerancia/100))
r5 <- rnorm(n=muestra,sd=resistencia*tolerancia/100,m=resistencia)
set.seed(runif(n=1,min=resistencia - resistencia*tolerancia/100,max=resistencia
+ resistencia*tolerancia/100))
r6 <- rnorm(n=muestra,sd=resistencia*tolerancia/100,m=resistencia)
set.seed(runif(n=1,min=resistencia - resistencia*tolerancia/100,max=resistencia
+ resistencia*tolerancia/100))
r7 <- rnorm(n=muestra,sd=resistencia*tolerancia/100,m=resistencia)
# Se generan distribuciones para un voltaje determinado más su componente
de ruido dada por sigma cuadrado.
set.seed(runif(n=1,min=as.integer(v1 - v1*tolerancia_v1/100),max=as.integer(v1
+ v1*tolerancia_v1/100)))
v1 <- rnorm(n=muestra,sd=tolerancia_v1, m=valor_v1)
set.seed(runif(n=1,min=abs(as.integer(v2 -
v2*tolerancia_v2/100)),max=as.integer(v2 + v2*tolerancia_v2/100)))
v2 <- rnorm(n=muestra,sd=tolerancia_v2, m=valor_v2)
# Genera gráfico con las respectivas distribuciones de resistencias.
colors <- c("black","blue","yellow","green","red","gray","purple")
leg <- c("R1","R2","R3","R4","R5","R6","R7")
png("resistencias.png")
plot(density(r1),main="Densidad de Probabilidad de
Resistencias",xlab="Resistencia [Ohms]",ylab="Densidad de Probabilidad")
lines(density(r2),col=colors[2])
lines(density(r3),col=colors[3])
```

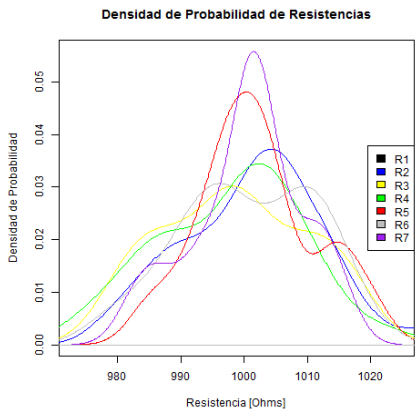
```
lines(density(r4),col=colors[4])
lines(density(r5),col=colors[5])
lines(density(r6),col=colors[6])
lines(density(r7),col=colors[7])
legend(“right”,,leg,colors)
dev.off()


# Función de Transferencia del Sistema con 2 Operacionales
opamp2 <- (r1/r2 + 1) *v1 - (r1/r2) * (r3/r4 + 1)*v2


# Función de Transferencia del Sistema con 3 Operacionales
opamp3 <- ((r6/(r4*r3))*((r4+r7)*(r2+r3))/(r5+r6) + ((r1*r7)/(r2*r4)))*v1 -
(((r3*r6)/(r2*r4))*((r4+r7)/(r5+r6)) + ((r7/(r2*r4)) * (r1+r2)))*v2


# Valores Teóricos
vt2 <- valor_v2
vt1 <- valor_v1
set.seed(resistencia)
r <- rnorm(n= muestra, sd = resistencia*tolerancia/100, m=resistencia)
opampt2 <- ((r/r) + 1) *vt1 - (r/r) * ((r/r) + 1)*vt2
opampt3 <- ((r/(r*r))*((r+r)*(r+r))/(r+r) + ((r*r)/(r*r)))*vt1 - (((r*r)/(r*r))*((r+r)/(r+r)) +
((r/(r*r)) * (r+r)))*vt2


# Gráfico Respuesta del Sistema con 2 Operacionales
png(“respuestas2.png”)
```

```
plot(density(opamp2),main="Densidad de Probabilidad de Respuestas del Sistema",xlab="Voltaje [V]",ylab="Densidad de Probabilidad")
dev.off()


# Gráfico Respuesta del Sistema con 3 Operacionales
png("respuestas3.png")
plot(density(opamp3),main="Densidad de Probabilidad de Respuestas del Sistema",xlab="Voltaje [V]",ylab="Densidad de Probabilidad")
dev.off()
# Genera un archivo con la media y la desviación estándar de cada variable
a<-matrix( c("Dato", "R1:","R2:","R3:","R4:","R5:","R6:","R7:","V1:","V2:","OpAmp2:","OpAmp3:", "Media",mean(r1),mean(r2),mean(r3),mean(r4),mean(r5),mean(r6),mean(r7),mean(v1), mean(v2),mean(opamp2),mean(opamp3), "Desv. Std.",sd(r1),sd(r2),sd(r3),sd(r4),sd(r5),sd(r6),sd(r7),sd(v1),sd(v2),sd(opamp2),sd(opamp3)),nrow = 12,
ncol = 3)
write.table(a, "resultados.csv",sep=",")


# Intervalos de Confianza par alas medias de ambos sistemas
leftopamp2 <- mean(opamp2)-qnorm(1-significancia/(2*100))*sd(opamp2)/sqrt(muestra)
rightopamp2 <- mean(opamp2)+qnorm(1-significancia/(2*100))*sd(opamp2)/sqrt(muestra)
leftopamp3 <- mean(opamp3)-qnorm(1-significancia/(2*100))*sd(opamp3)/sqrt(muestra)
```

```
rightopamp3 <- mean(opamp3)+qnorm(1-significancia/(2*100))*sd(opamp3)/sqrt(muestra)
# Realiza pruebas de hipotesís, para conprobar si las medias teóricas y las medidas simuladas son iguales
t.test(opamp2,mu=mean(opampt2),conf.level=1-significancia/100)
t.test(opamp3,mu=mean(opampt3),conf.level=1-significancia/100)


# Guarda en un archivo todos los números aleatorios generados
a<-matrix(
c(“R1”,r1,”R2”,r2,”R3”,r3,”R4”,r4,”R5”,r5,”R6”,r6,”R7”,r7,”v1”,v1,”v2”,v2,”opamp2”
,opamp2,”opamp3”,opamp3) , ncol = 11, nrow = muestra+1)
write.table(a, “resultados.csv”, sep=”,”,append=TRUE)
```

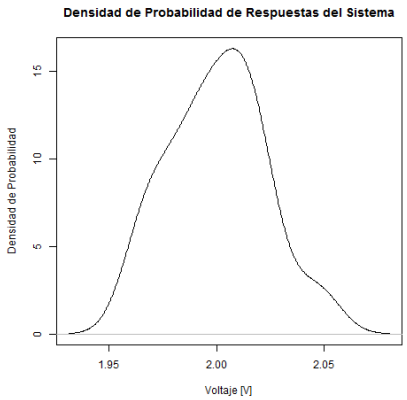
# APÉNDICE C: GRÁFICOS DE RESPUESTAS DE SIMULACIONES

## Densidad de Probabilidad de Resistencias, n = 30, R = (1,00±0,01) kΩ, Ruido = 1 %



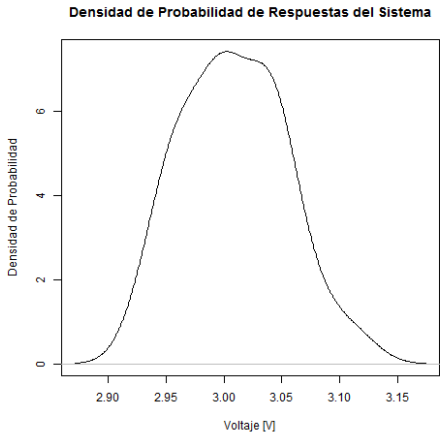

Fuente: elaboración propia.

## Densidad de Probabilidad de Respuesta del Sistema con dos operacionales, n = 30, R = (1,00±0,01) kΩ, Ruido = 1 %



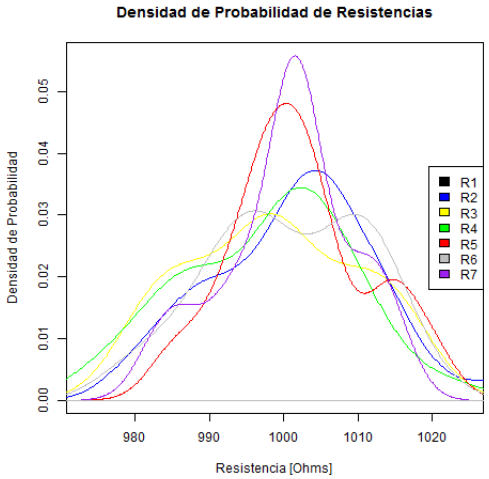

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuesta al Sistema con tres Operacionales, n = 30, R = (1,00±0,1) kΩ, Ruido = 1 %**



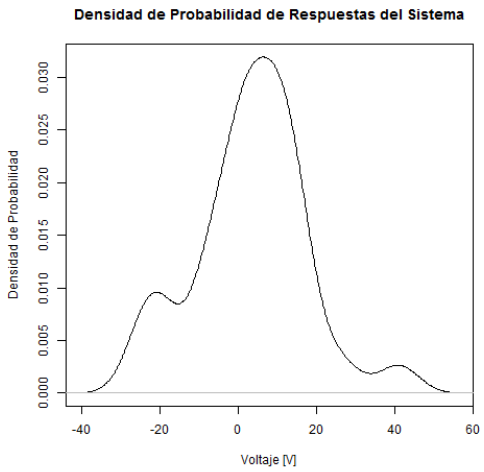

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 30, R = (1,00±0,01) kΩ, Ruido = 5 %**



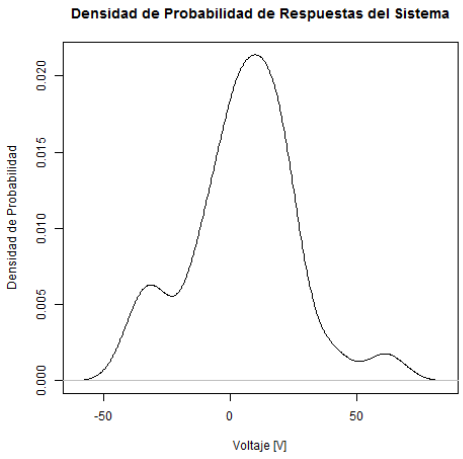

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuesta del Sistema con dos Operacionales, n = 30, R = (1,00±0,01) kΩ, Ruido = 5 %**



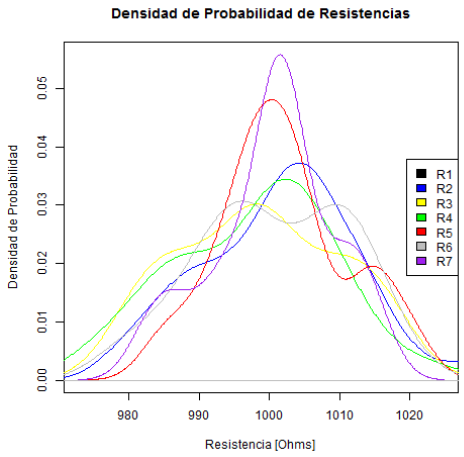

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres operacionales, n = 30, R = (1,00±0,01) kΩ, Ruido = 5 %**



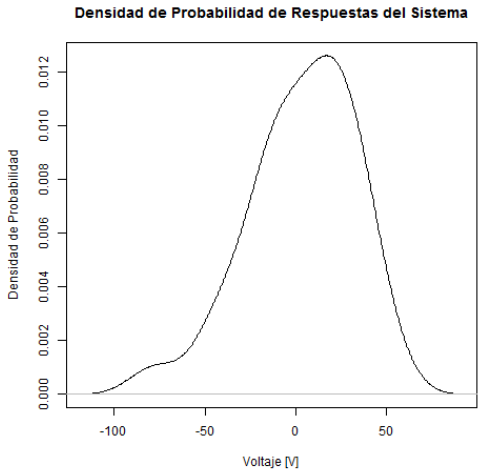

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 30, R = (1,00±0,01) kΩ, Ruido = 10 %**



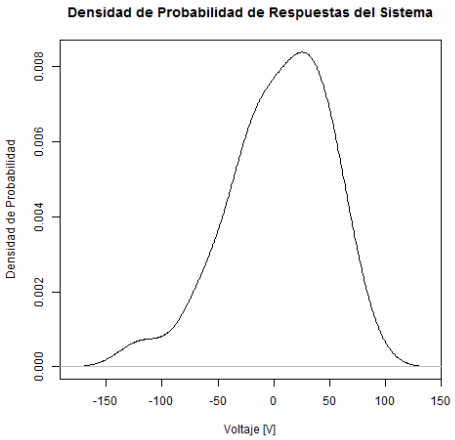

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos operacionales, n = 30, R = (1,00±0,01) kΩ, Ruido = 10 %**



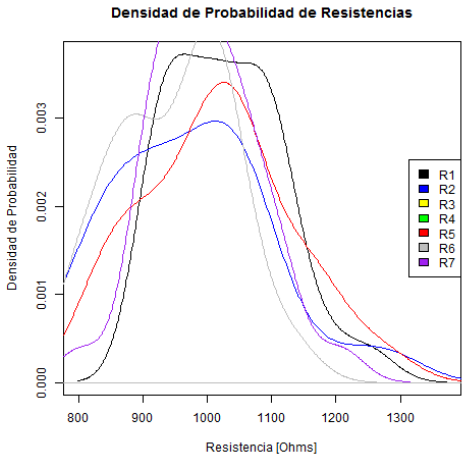

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuesta del Sistema con tres Operacionales, n = 30, R = (1,00±0,01) kΩ, Ruido = 10 %**



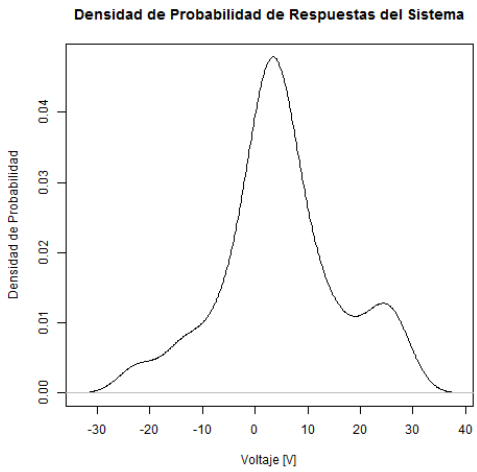

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 30, R = (1,0±0,1) kΩ, Ruido = 1 %**



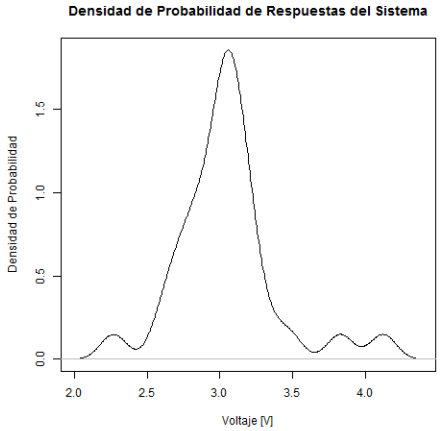

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 30, R = (1,0±0,1) kΩ, Ruido = 1 %**



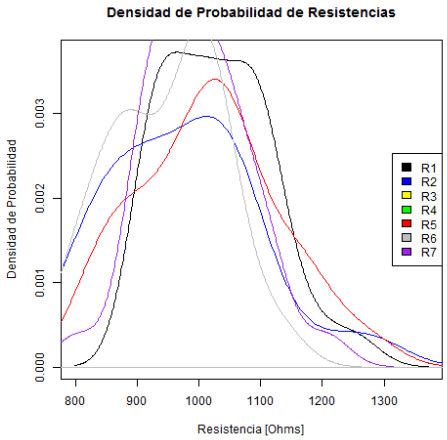

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres Operacionales, n = 30, R = (1,0±0,1) kΩ, Ruido = 1 %**



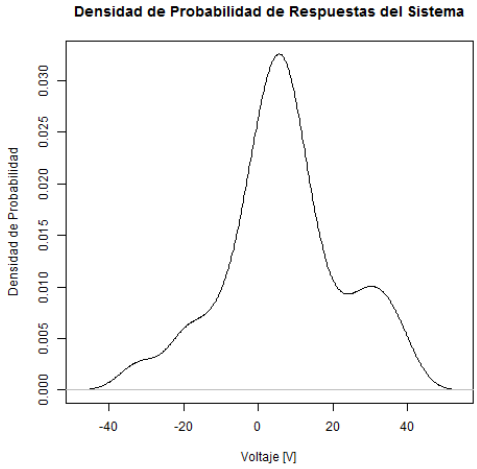

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 30, R = (1,0±0,1) kΩ, Ruido = 5 %**



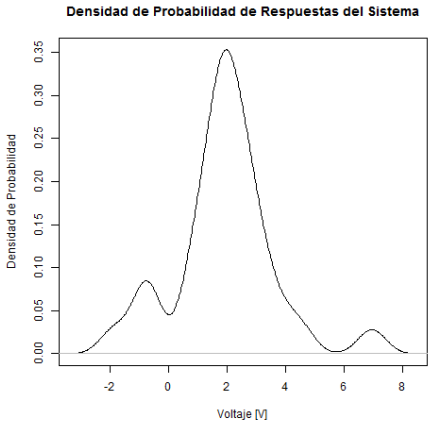

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 30, R = (1,0±0,1) kΩ, Ruido = 5 %**



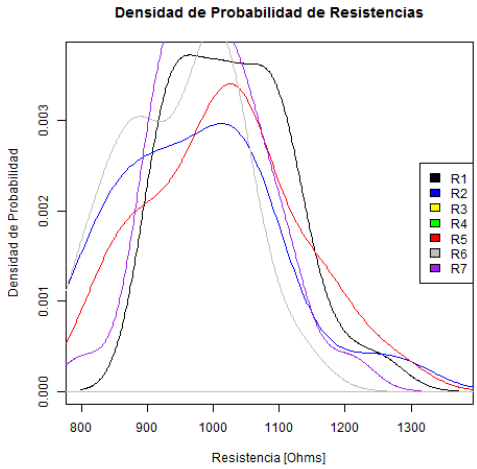

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres Operacionales, n = 30, R = (1,0±0,1) kΩ, Ruido = 5 %**



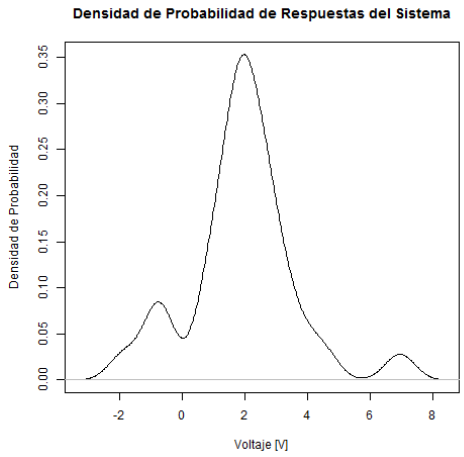

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 30, R = (1,0±0,1) kΩ, Ruido = 10 %**



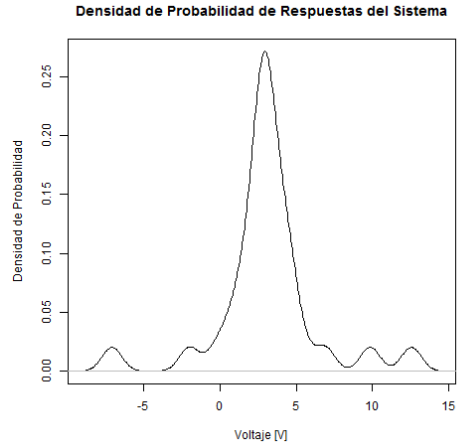

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 30, R = (1,0±0,1) kΩ, Ruido = 10 %**



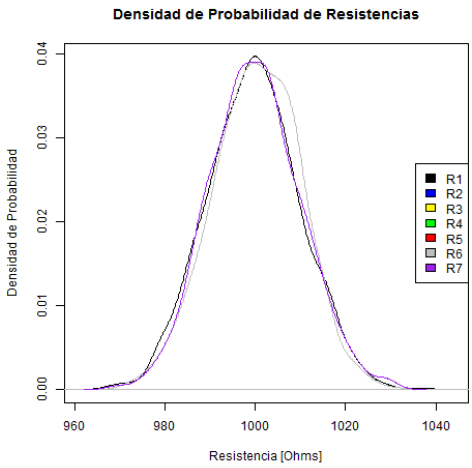

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuesta del Sistema con tres operacionales, n = 30, R = (1,0±0,1) kΩ, Ruido = 10 %**



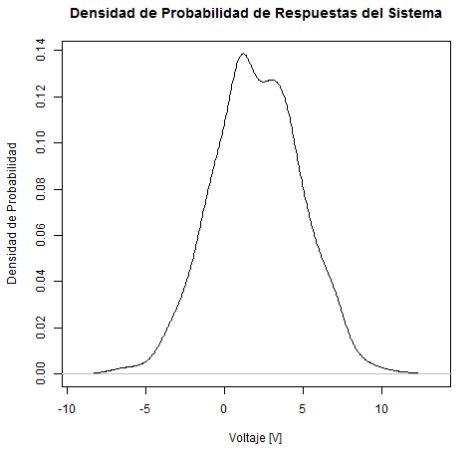

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencia, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 1 %**



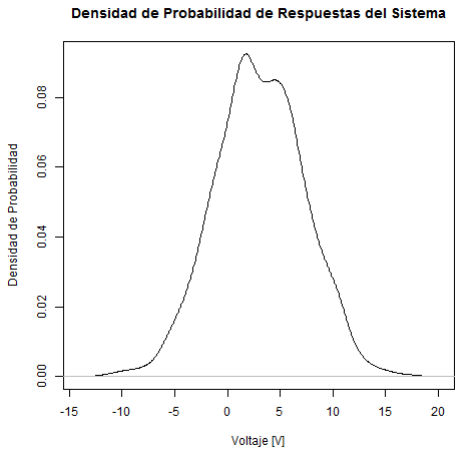

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuesta del Sistema con dos operacionales, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 1 %**



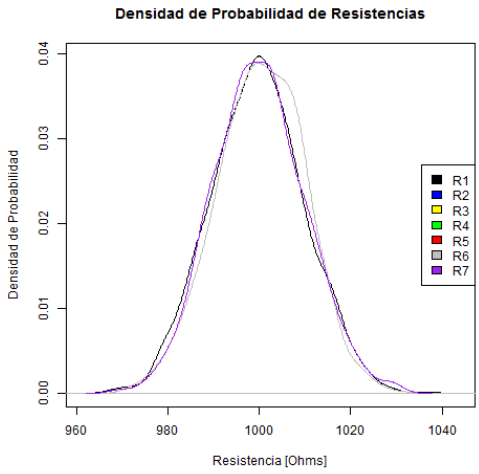

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres operacionales, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 1 %**



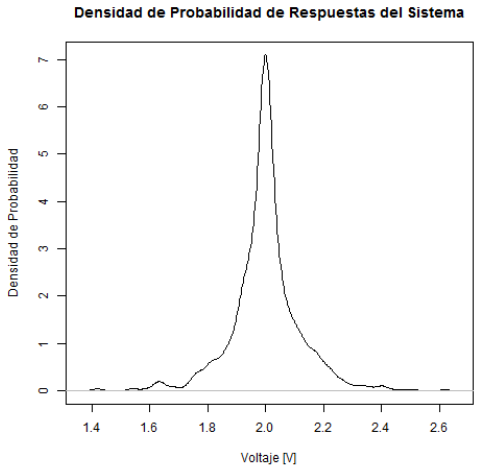

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 5 %**



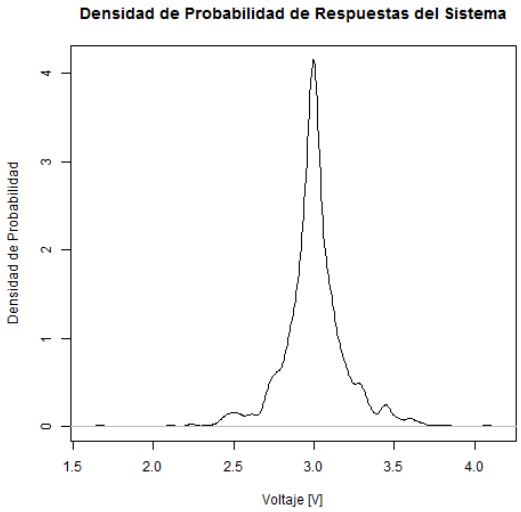

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 5 %**



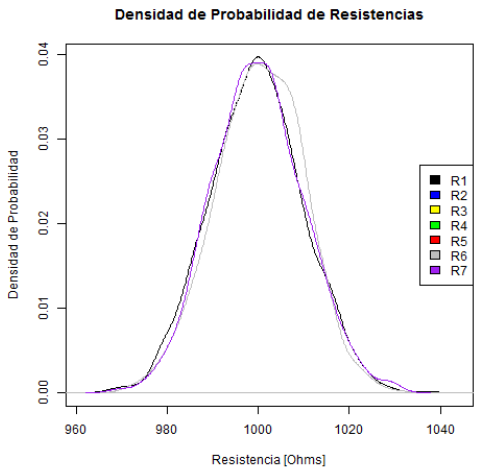

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres Operacionales, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 5 %**



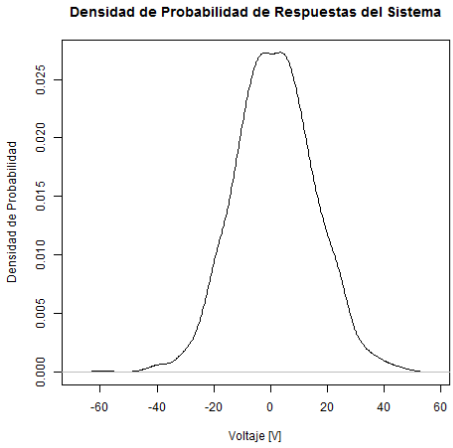

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 10 %**



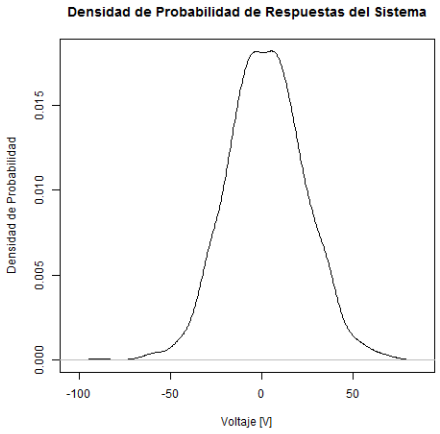

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 10 %**



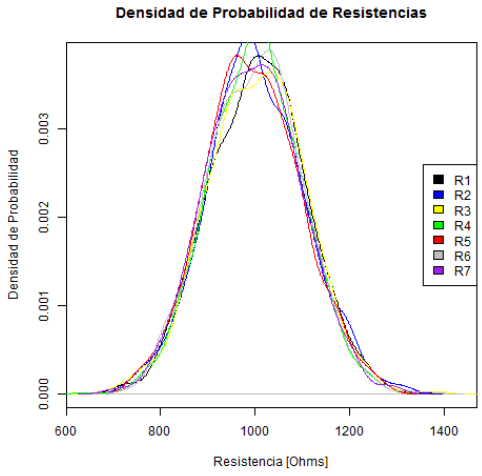

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres Operacionales, n = 1 000, R = (1,00±0,01) kΩ, Ruido = 10 %**



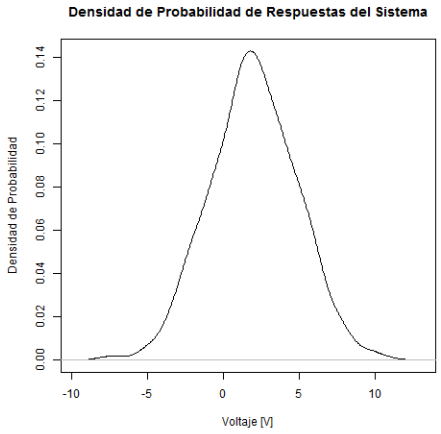

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 1 000, R = (1,0±0,1) kΩ, Ruido = 1 %**



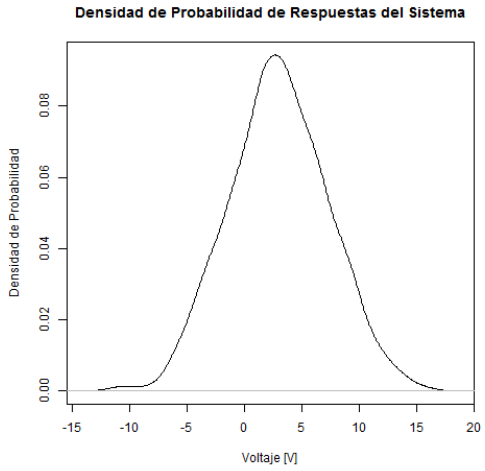

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 1 000, R = (1,0±0,1) kΩ, Ruido = 1 %**



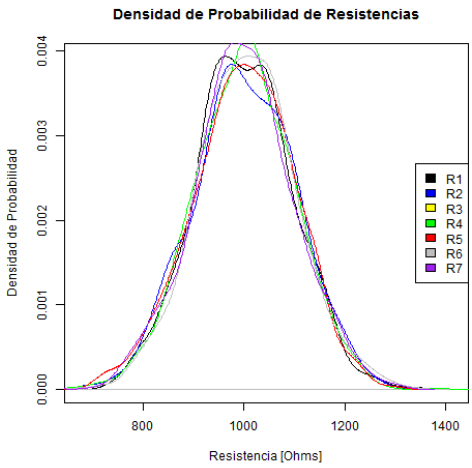

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres Operacionales, n = 1000, R = (1,0±0,1) kΩ, Ruido = 1 %**



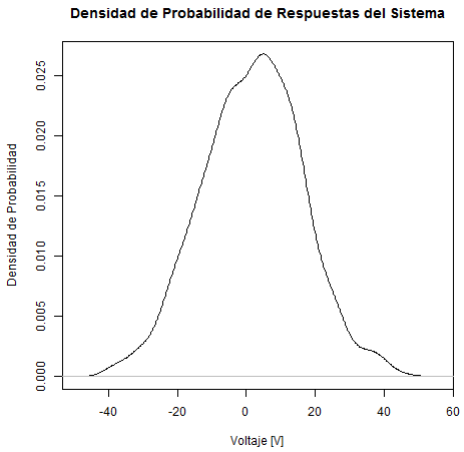

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 1 000, R = (1,0±0,1) kΩ, Ruido = 5 %**



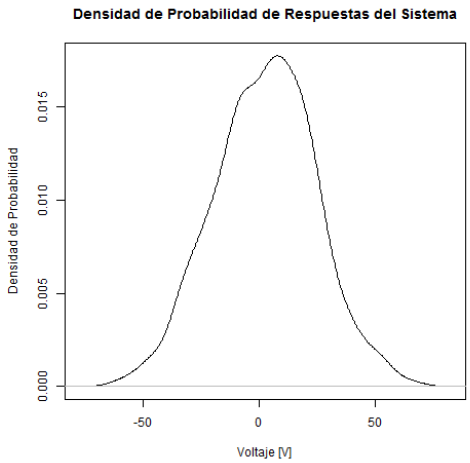

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 1 000, R = (1,0±0,1) kΩ, Ruido = 5 %**



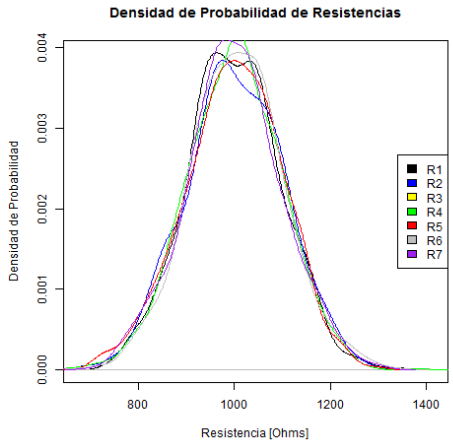

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres Operacionales, n = 1 000, R = (1,0±0,1) kΩ, Ruido = 5 %**



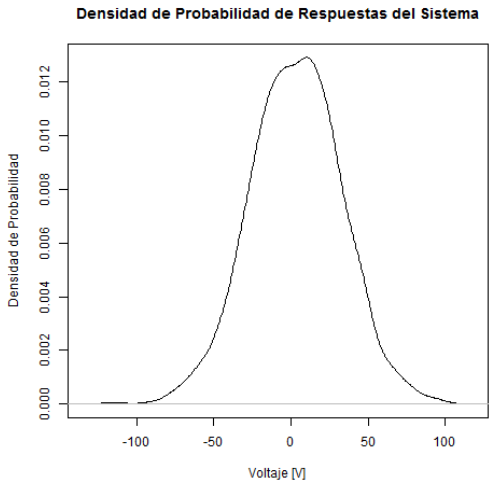

Fuente: elaboración propia.

**Densidad de Probabilidad de Resistencias, n = 1000, R = (1,0±0,1) kΩ, Ruido = 10 %**



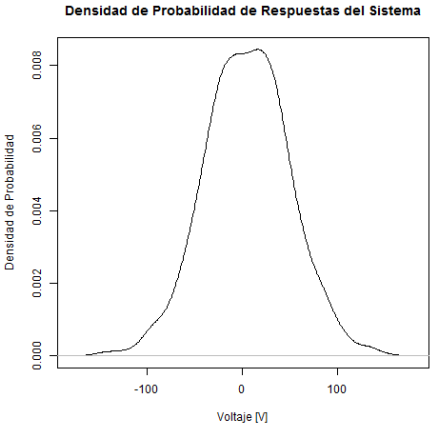

Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con dos Operacionales, n = 1 000, R = (1,0±0,1) kΩ, Ruido = 10 %**




Fuente: elaboración propia.

**Densidad de Probabilidad de Respuestas del Sistema con tres Operacionales, n = 1 000, R = (1,0±0,1) kΩ, Ruido = 10 %**




Fuente: elaboración propia.

# APÉNDICE D: CÓDIGO EN PICC PARA RECUPERACIÓN DE SEÑAL ORIGINAL

# El siguiente código obtiene la componente de tendencia de los datos en base a una serie de tiempo, bajo una ventana de tiempo de 1 seg.

```
#include <18f8722.h>
#DEVICE ADC=10
#include <stdio.h>
#use delay (clock=1000000)
#use rs232 (BAUD=9600, BITS=8, STOP=1, UART1)

#define  N       100            // Tamaño de la muestra
#define  ESPERA   1000/N            // Ventana de Muestreo 1seg
#define  SUBN     20            // Submuestra para obtener tendencia

unsigned long muestras[N], tenden[N];

void muestreo(unsigned long milisec);
unsigned long desviacionm(unsigned long * datos, unsigned long media);
void tendencia(unsigned long * datos, unsigned long * tend, unsigned char m);
unsigned long promedio(unsigned long * datos);
unsigned long convvolt(unsigned long muestras);

void main(void){
  unsigned long resmedia, resdesvm;
  unsigned char entero, decimal;
```

```
  // Inicializacion de ADC
  setup_adc(ADC_CLOCK_INTERNAL);
  setup_adc_ports( ALL_ANALOG );
  set_adc_channel(0);

  while(1){
    muestreo(ESPERA);                                  // Muestreo
// Determinación de Tendencia
tendencia(muestras,tenden,SUBN);
// Determinación de Media
resmedia = promedio(tenden);
// Determinación de Desviación
resdesvm = desviacionm(tenden, resmedia);

    // Conversión a Voltaje
    resmedia = convvolt(resmedia);
    resdesvm = convvolt(resdesvm);

    // Impresión de Resultados
    entero  = resmedia / 1000;
    decimal = resmedia % 1000;
    printf("Valor Aproximado: (%u.%03u",entero, decimal);
    putc (241);              // Signo más menos
    entero  = resdesvm / 1000;
    decimal = resdesvm % 1000;
    printf("%u.%03u) V\n",entero, decimal);
  }
```

```
  setup_adc(ADC_OFF);

}

void muestreo(unsigned long milisec){
  unsigned int i;
  i = N;
  do{
    // Lectura en ADC
    *(muestras + N-i)= read_adc();

    i -= 1;
    delay_ms(milisec);
  }while (i);
}

void tendencia(unsigned long * datos, unsigned long * tend, unsigned char
m){
  unsigned int i,j;
  unsigned long temp;

  // Efectúa un promedio móvil de una submuestra de la muestra original
  for (i=0; i+m<N; i++){
    temp = 0;
    for (j=0; j<m; j++)
      temp += *(datos + j + i);
    *(tend + i) = temp/m;
  }
}
```

```
unsigned long promedio(unsigned long * datos){
  unsigned char i;
  unsigned long prom;

  // Realiza un promedio de toda la muestra luego del promedio móvil
  for (i=0; i<N - SUBN -1; i++)
    prom += *(datos + i);
  prom /= (N - SUBN - 1);
  return prom;
}

unsigned long desviacionm(unsigned long * datos, unsigned long media){
  unsigned char i;
  unsigned long srq, desviacion;

    // Obtiene la desviación muestral
    srq = 0;
    for (i = 0; i < N; i++)
      srq += (*(datos+i) - media)^2;
    srq /= (N-1);

    for (i = 1; i < 255; i++)
      if (i*i > 2*srq+1)
        return(desviacion);
  return(0);

}
```

```
unsigned long convvolt(unsigned long muestras){
  double conversion;

    // Conversión a voltaje de los valores cuantizados
    // Factor   5000/1024
    conversion = muestras * 125;
    conversion /= 256;

    return ( (unsigned long) conversion);

}
```

# APÉNDICE E: CÓDIGO EN WINAVR PARA RECUPERACIÓN DE SEÑAL ORIGINAL

// El siguiente código obtiene la componente de tendencia de los datos en base a una serie de tiempo, bajo una ventana de tiempo de 1 seg.

```
#define F_CPU 1000000
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <stdbool.h>

#define BAUDRATE 19200
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)

#define  N       100            // Tamaño de la muestra
#define  ESPERA   1000/N            // Ventana de Muestreo 1seg
#define  SUBN     20             // Submuestra para obtener tendencia

uint16_t muestras[N], tenden[N];

void muestreo(uint16_t milisec);
uint16_t desviacionm(uint16_t * datos, uint16_t media);
void tendencia(uint16_t * datos, uint16 * tend, uint8_t m);
uint16_t promedio(uint16_t * datos);
uint16_t convvolt(uint16_t muestras);
```

```
void usart_init(uint16_t ubrr);
char usart_getchar( void );
void usart_putchar( char data );
void usart_pstr (char *s);
unsigned char usart_kbhit(void);

int usart_putchar_printf(char var, FILE *stream)
static FILE mystdout = FDEV_SETUP_STREAM(usart_putchar_printf,
NULL, _FDEV_SETUP_WRITE);

void main(void){
  uint16_t resmedia, resdesvm;
  uint8_t entero, decimal;
   // define some local variables

   uint8_t myvalue;

  ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
  ADMUX  |= (1 << REFS0);
  ADMUX  |= (1 << ADLAR);
  ADCSRA |= (1 << ADEN);
  ADCSRA |= (1 << ADSC);

   // configuracion de stdio stream

   stdout = &mystdout;

  // Inicialización de UART
   usart_init(UBRRVAL);
```

```
  while(1){
    muestreo(ESPERA);                              // Muestreo
// Determinación de Tendencia
tendencia(muestras,tenden,SUBN);
// Determinación de Media
resmedia = promedio(tenden);
// Determinación de Desviación
resdesvm = desviacionm(tenden, resmedia);

    // Conversión a Voltaje
    resmedia = convvolt(resmedia);
    resdesvm = convvolt(resdesvm);

    // Impresión de Resultados
    entero  = resmedia / 1000;
    decimal = resmedia % 1000;
    prshortf("Valor Aproximado: (%u.%03u",entero, decimal);
    prshortf("%c",241);              // Signo más menos
    entero  = resdesvm / 1000;
    decimal = resdesvm % 1000;
    prshortf("%u.%03u) V\n",entero, decimal);
  }
}

void muestreo(uint16_t milisec){
  uint16_t i;
  uint16_t read_adc;
  i = N;
```

```
do{
      // Lectura en ADC
  read_adc = ADCL;

  /*shift from low level to high level ADC, from 8bit to 10bit*/
  read_adc += (ADCH<<8);

      *(muestras + N-i)= read_adc;

      i -= 1;
      _delay_ms(milisec);

   }while (i);
}

void tendencia(uint16_t * datos, uint16_t * tend, uint8_t m){
   uint16_t i,j;
   uint16_t temp;

   // Efectúa un promedio móvil de una submuestra de la muestra original
   for (i=0; i+m<N; i++){
      temp = 0;
      for (j=0; j<m; j++)
         temp += *(datos + j + i);
      *(tend + i) = temp/m;
   }
}
```

```
uint16_t promedio(uint16_t * datos){
  uint8_t i;
  uint16_t prom;

  // Realiza un promedio de toda la muestra luego del promedio móvil
  for (i=0; i<N - SUBN -1; i++)
    prom += *(datos + i);
  prom /= (N - SUBN - 1);
  return prom;
}

uint16_t desviacionm(uint16_t * datos, uint16_t media){
  uint8_t i;
  uint16_t srq, desviacion;

    // Obtiene la desviación muestral
    srq = 0;
    for (i = 0; i < N; i++)
      srq += (*(datos+i) - media)^2;
    srq /= (N-1);

    for (i = 1; i < 255; i++)
      if (i*i > 2*srq+1)
        return(desviacion);
  return(0);

}
```

```
uint16_t convvolt(uint16_t muestras){
   uint16_t conversion;

      // Conversión a voltaje de los valores cuantizados
      // Factor   5000/1024
      conversion = muestras * 125;
      conversion /= 256;

      return (conversion);

}

int usart_putchar_printf(char var, FILE *stream) {
    if (var == '\n') usart_putchar('\r');
    usart_putchar(var);
    return 0;
}

void usart_init( uint16_t ubrr) {
    // Configura tasa de baudios
    UBRRH = (uint8_t)(ubrr>>8);
    UBRRL = (uint8_t)ubrr;
    // Habilita transmisión y recepción
    UCSRB = (1<<RXEN)|(1<<TXEN);
    // 8 bits de datos, 1 bit de parada
    UCSRC = (1<<URSEL)|(3<<UCSZ0);
}
```

```
void usart_putchar(char data) {
    // Espera que el buffer este vacío
    while (!(UCSRA & (_BV(UDRE))));
    // Empieza transmisión
    UDR = data;
}

char usart_getchar(void) {
    // Espera por dato en el buffer
    while (!(UCSRA & (_BV(RXC))));
    // Envía caracter
    return UDR;
}

unsigned char usart_kbhit(void) {
    //regresa un caracter diferente de cero en la version que utiliza polled
    unsigned char b;
    b=0;
    if(UCSRA & (1<<RXC)) b=1;
    return b;
}

void usart_pstr(char *s) {
    // Bucle a través de todo el string
    while (*s) {
        usart_putchar(*s);
        s++;
    }
}
```

```
// handler utilizado por printf
int usart_putchar_printf(char var, FILE *stream) {
    // cambia \n por \r para la terminal br@y++
    if (var == '\n') usart_putchar('\r');
    usart_putchar(var);
    return 0;
}
```

# APÉNDICE F: CÓDIGO EN KEIL ARM PARA RECUPERACIÓN DE SEÑAL ORIGINAL

// El siguiente código obtiene la componente de tendencia de los datos en base a una serie de tiempo, bajo una ventana de tiempo de 1 seg.

```
#include <LPC211X.h>
#include <stdio.h>

#define  N        100              // Tamaño de la muestra
#define  ESPERA   1000/N           // Ventana de Muestreo 1seg
#define  SUBN     20               // Submuestra para obtener tendencia

unsigned int muestras[N], tenden[N];

void muestreo(unsigned int milisec);
unsigned int desviacionm(unsigned int * datos, unsigned int media);
void tendencia(unsigned int * datos, unsigned int * tend, unsigned char m);
unsigned int promedio(unsigned int * datos);
unsigned int convvolt(unsigned int muestras);
void SER_init (void);
int SER_putChar (int uart, int c);
char SER_getChar (int uart);
int SER_getChar_nb (void);
int SER_BufferReady (void);
void SER_putString (unsigned char *s);
void SER_getString (unsigned char *s);
void adc_init(void);
void delay(unsigned long int count1);
```

```
void main(void){
   unsigned int resmedia, resdesvm;
   unsigned char entero, decimal;

  // Configuración P0.27 -> ADC0
  PINSEL1 &= ~0x00C00000;
  PINSEL1 |=  0x00400000;

  SER_init();

   while(1){
      muestreo(ESPERA);                              // Muestreo
// Determinación de Tendencia
      tendencia(muestras,tenden,SUBN);
// Determinación de Media
resmedia = promedio(tenden);
// Determinación de Desviación
      resdesvm = desviacionm(tenden, resmedia);
      // Conversión a Voltaje
      resmedia = convvolt(resmedia);
      resdesvm = convvolt(resdesvm);

      // Impresión de Resultados
      entero  = resmedia / 1000;
      decimal = resmedia % 1000;
      printf("Valor Aproximado: (%u.%03u",entero, decimal);
      printf("%c",241);              // Signo más menos
```

```
      entero  = resdesvm / 1000;
      decimal = resdesvm % 1000;
      printf("%u.%03u) V\n",entero, decimal);
    }
}

void muestreo(unsigned int milisec){
   unsigned int i;
   unsigned char read_adc;

   i = N;
  PCONP               |= 0x00000100;
  AD0CR               &= 0xFFFFFF00;
  AD0CR               |=  1;

   do{
// PDN = 1 = Active ADC Module
          AD0CR                |= 0x00200000;
// Inicio de la Conversion
AD0CR                  |= 0x01000000;
// Mientras termina la conversion
          while (!(AD0GDR & 0x80000000));
          read_adc      = AD0GDR;
//Obtener dato aplicando mascara
          read_adc =   (read_adc >> 6) & 0x03FF;
//Detiene Conversion
AD0CR          &=      ~(0x07000000);

//PDN = 0 = Deactivate ADC Module
```

```
AD0CR          &=  ~(0x00200000);

      // Lectura en ADC
      *(muestras + N-i)= read_adc;

      i -= 1;
      delay(milisec*1000);
   }while (i);
}

void tendencia(unsigned int * datos, unsigned int * tend, unsigned char m){
   unsigned int i,j;
   unsigned int temp;

   // Efectúa un promedio móvil de una submuestra de la muestra original
   for (i=0; i+m<N; i++){
      temp = 0;
      for (j=0; j<m; j++)
         temp += *(datos + j + i);
      *(tend + i) = temp/m;
   }
}

unsigned int promedio(unsigned int * datos){
   unsigned char i;
   unsigned int prom;

   // Realiza un promedio de toda la muestra luego del promedio móvil
   for (i=0; i<N - SUBN -1; i++)
```

```
    prom += *(datos + i);
  prom /= (N - SUBN - 1);
  return prom;
}

unsigned int desviacionm(unsigned int * datos, unsigned int media){
  unsigned char i;
  unsigned int srq, desviacion;

    // Obtiene la desviación muestral
    srq = 0;
    for (i = 0; i < N; i++)
      srq += (*(datos+i) - media)^2;
    srq /= (N-1);

    for (i = 1; i < 255; i++)
      if (i*i > 2*srq+1)
        return(desviacion);
  return(0);

}

unsigned int convvolt(unsigned int muestras){
  unsigned int conversion;


    // Conversión a voltaje de los valores cuantizados
    // Factor   5000/1024
    conversion = muestras * 125;
```

```
    conversion /= 256;

    return (conversion);

}


// Inicializar UART
void SER_init (void) {

/* Resetea los pines P0.0,P0.1 por medio de una operacion And*/
   PINSEL0 &= 0xFFFFFFF0;
/* Enable RxD0 and TxD0, RTS y CTS             */
   PINSEL0 |= 0x00000005;
/* 8 bits, no Parity, 1 Stop bit, DLAB = 1*/
   U0LCR       = 0x00000083;
/* 9600 Baud Rate @ 14.75MHz VPB Clock, DLL = 96*/
   U0DLL       = 0x00000060;
/* DLM = 0                                      */
   U0DLM               = 0x00000000;
/* FR = 1, DIVADDVAL = 0, MULVAL = 1 */
U0FDR          = 0x10;
/* DLAB = 0                                     */
U0LCR          = 0x00000003;
}

// Escribe caracter en puerto serial
int SER_putChar (int uart, int c) {
   while (!(U0LSR & 0x20));
```

```
    return (U0THR = c);
}

// Lee caracter del puerto serial
char SER_getChar (int uart) {
    while (!(U0LSR & 0x01));
    return (U0RBR);
}

// Lee caracter del puerto serial
int SER_getChar_nb (void) {
    if (U0LSR & 0x01) return (U0RBR);
  else return 0;
}

// Verifica si el buffer está listo
int SER_BufferReady (void) {
    if (U0LSR & 0x01) return (1);
  else return 0;
}

// Escribe caracteres al puerto serial
void SER_putString (unsigned char *s) {

  while (*s != 0) {
   SER_putChar(*s++);
  }
}
//  Lectura de String por puerto serialRead string from Serial Port
```

```
void SER_getString (unsigned char *s) {
   char c;
// Esperar el primer Byte o salir por TimeOut
  while (1){
   c = SER_getChar();
   if (c != 0x0A){
          *s++ = c;
  }else{
                *s++ = c;
                c = 0;
                *s = c;
          return;
          }
  }
}

//Inicialización ADC
void adc_init(void){

  PINSEL1      &= ~0x00300000;
//Configuracion ADC0
  PINSEL1      |= 0x00100000;              .0
// Limpia todos los bits de control
  AD0CR              &= 0x00000000;
// ADC Clock = VBP(PCLK) / 7
  AD0CR              |= 0x00000600;
// Busrt = 1 = Conversion Continue
  AD0CR              |= 0x00010000;
// CLKS = 000 = 10Bit : 11 Cycle Clock Conversion
```

```
  AD0CR                 &= 0xFFF1FFFF;
// TEST[1:0] = 00 = Normal Mode
AD0CR                   &= 0xFF3FFFFF;
// EDGE = 0 = Conversion on Falling Edge
  AD0CR                 &= 0xF7FFFFFF;

}

void delay(unsigned long int count1){

  while(count1 > 0){
          count1--;
  }
}
```