SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

SEP
Institutos Tecnológicos

Subsecretaría de Educación Superior
Dirección General de Educación Superior Tecnológica
Instituto Tecnológico de la Laguna

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

" Implementación de un electrocardiógrafo con trasmisión inalámbrica y tratamiento de la señal con Matlab y Octave "

POR

Ing. Juan Salvador García García

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

DIRECTOR DE TESIS

Dr. Francisco Flores García

ISSN: 0188-9060

RIITEC: (04)-TMCIE-2014

Torreón, Coahuila. México
Junio, 2014

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
Instituto Tecnológico de la Laguna

"2014, Año de Octavio Paz

Torreón, Coah., 29/Mayo/2014
Dependencia. DEPI/CPCIE
Oficio: DEPI/CPCIE/032/2014
Asunto: Autorización de impresión de tesis.

C. JUAN SALVADOR GARCÍA GARCÍA
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA.
PRESENTE

Después de haber sometido a revisión su trabajo de tesis titulado:

"Implementación de un electrocardiógrafo con transmisión inalámbrica y tratamiento de señales con Matlab y Octave"

Habiendo cumplido con todas las indicaciones que el jurado revisor de tesis hizo, se le comunica que se le concede la autorización con número de registro **RITEC: (04)-TMCIE-2014**, para que proceda a la impresión del mismo.

ATENTAMENTE
EDUCACIÓN TECNOLÓGICA FUENTE DE INNOVACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO de la Laguna
División de Estudios de Posgrado e Investigación

DR. JOSÉ LUIS MEZA MEDINA
Jefe de la División de Estudios de Posgrado e Investigación
del Instituto Tecnológico de la Laguna

c.c.p. HSR

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
Instituto Tecnológico de la Laguna

"2014, Año de Octavio Paz

Torreón, Coah., 28/Mayo/2014

DR. JOSE LUIS MEZA MEDINA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Por medio de la presente, hacemos de su conocimiento que después de haber sometido a revisión el trabajo de tesis titulado:

"Implementación de un electrocardiógrafo con transmisión inalámbrica Y tratamiento de señal con Matlab y Octave"

Desarrollado por el **C. Salvador García García**, con número de control **M1213020** y habiendo cumplido con todas las correcciones que se le indicaron, estamos de acuerdo que se le conceda la autorización de la fecha de examen de grado para que proceda a la impresión de la misma.

ATENTAMENTE
EDUCACIÓN TECNOLÓGICA FUENTE DE INNOVACIÓN

Dr. Francisco Flores García
Asesor/Director

Dr. María I. Cepeda Rubio
Comité Tutorial

Dr. Francisco Valdés Perezgasga
Comité Tutorial

M.C. Sergio F. Salas Huerta
Comité Tutorial

AGRADECIMIENTOS

Quiero agradecer a todas las personas que me ayudaron en este largo camino. Al Instituto Tecnologico de La Laguna, una gran casa de estudios, donde pasé muchos años de mi vida, y a todo su cuerpo docente, que hicieron de mí un gran profesionista y sin duda una mejor persona.

A todos los excelentes maestros del posgrado, ¡Qué gran experiencia fue compartir con ellos!, a todos los recuerdo con mucho cariño admiración y respeto.

Agradezco al Dr. Francisco Flores García por haber confiado en mi persona, por la paciencia y por la dirección de este trabajo. Muy especial agradecimiento Al Dr. Mario Francisco Jesús Cepeda Rubio por cada consejo, el apoyo y el ánimo que me brindó en todo momento. Al Dr. Francisco Valdés Perezgasga por su paciencia y por la atenta lectura de este trabajo. Además, por sus comentarios en todo el proceso de elaboración de la Tesis y sus atinadas correcciones.

Hoy culmino una meta más en mi camino, una de las más importantes, la cual, no hubiera sido posible sin el apoyo de muchos Ángeles que Dios puso en mi camino. Dedico este trabajo a todos ellos, por ser parte importante de mi vida: mi familia, mi novia y mis amigos.

A mi madre Mayela García Ortega, un temple de belleza interior y exterior, un ejemplo para cualquier mujer, un ideal de persona. No conozco mujer más bondadosa que ella, que fortuna de tenerla conmigo, a ella le debo todo lo bueno que tengo y que soy, así que este logro también es suyo.

A mi padre David García Mora, gracias por tu apoyo, por ser mi mejor amigo, por la orientación que me has dado, por iluminar mi camino y darme la pauta para poder realizarme en mis estudios y mi vida. Agradezco los consejos sabios que en el momento exacto has sabido darme para no dejarme caer y enfrentar los momentos difíciles, por ayudarme a tomar las decisiones que me ayuden a balancear mi vida y sobre todo gracias por el amor tan grande que me das.

Gracias a ambos por hacerme ver que los límites se los pone uno mismo, por enseñarme que las cosas se deben hacer lo mejor posible y con humildad, pero sobre todo por darme la oportunidad de ser su hijo.

Gracias a mis hermanos, David Alejandro García García, Daniel Eduardo García García y Ángel de Jesús García García que con su amor me han enseñado a salir adelante. Gracias por su paciencia, gracias por preocuparse por su hermano "sándwich" el de en medio,

gracias por compartir sus vidas, por apoyarme a no abandonar mis sueños, por escucharme, por su cariño pero sobre todo, gracias por estar en otro momento tan importante en mi vida.

A mi novia Marcela García Ezquerra, mi mejor amiga, la mujer con la cual deseo estar toda mi vida, el impulso que motiva mis pasos y me hace querer ser mejor cada día y superarme tanto en el ámbito personal como profesional, simplemente el amor de mi vida que me acompañó en esta aventura que significó la maestría y que, de forma incondicional, entendió mi ausencia y mis malos momentos.

A mi mejor amigo Jesús Cruz "Chuy", por hacer mucho más placenteros mis ratos libres y por todo su apoyo incondicional.

Gracias a cada uno de mis profesores a lo largo de todos estos años, porque cada uno contribuyo de manera importante para poder alcanzar este logro. Como no recordar a los Ingenieros "Cobos". "Estrada", "Ramón", "Marín", "Garay", "Panchito", "Salas", Juan Sifuentes, el Dr. Dzul y el Ing. "Lino", quien nos recuerda siempre que el integrado 741 nos puede sacar de grandes "apuros", Pero sobre todo, un gran agradecimiento al profesor y Maestro en Ciencias David García Mora, quien es mi ejemplo a seguir, debido a que no solo como padre ha sido una excelente persona, si no como catedrático. Agradezco infinitamente cada una de sus agradables e interesantes clases impartidas, ya que debido a ellas, descubrí ese gran amor que tengo por mi carrera.

Gracias a todos aquellos que no están aquí, pero que me ayudaron a que este gran esfuerzo se volviera realidad

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

RESUMEN:

Este documento ilustra un método de bajo costo para la adquisición de la señal ECG para el almacenamiento y procesamiento usando una interfaz gráfica de usuario basada en MATLAB, OCTAVE y PYTHON. La señal ECG es muestreada y después de la digitalización con ayuda de un microcontrolador se convierten los datos de la señal ECG aun formato RS232. Estos datos se transmiten a un ordenador personal de forma inalámbrica gracias a los módulos comunicación RF Xbee. Los datos del ECG se reconstruyen a partir de los establecidos por una fórmula de conversión de datos digitales. Por último, mediante las GUIS creadas anteriormente, se realiza un análisis en línea de los datos de la señal electrocardiográfica para calcular diferentes características como frecuencia cardiaca y amplitud, mostrando también la gráfica de la señal ECG.

ABSTRACT:

This report illustrates a low cost method for ECG signal acquisition and processing for storage using a Graphical User Interface based on MATLAB, OCTAVE AND PYTHON. The ECG signal is sampled and then digitized using a microcontroller. Data from the ECG signal are translated to a RS232 format. This data are transmitted to a PC wirelessly using Xbee RF communication modules. The ECG data is re-built from the digital data set by a conversion formula. Finally, using the GUIs previously created, an analysis is performed on the data from electrocardiographic signal to calculate several features like heart rate and amplitude, also displaying a graph of the ECG signal.

1. ANTECEDENTES DEL PROBLEMA, PLANTEAMIENTO DEL PROBLEMA, OBJETIVOS, HIPÓTESIS Y JUSTIFICACIÓN DEL PROYECTO.

1.1ANTECEDENTES DEL PROBLEMA

Al momento de empezar con cualquier proyecto de aplicación, una de las cuestiones principales suele ser enlazar la parte teórica, como algoritmos de tratamiento de datos, sistemas de control automático, etc., con el mundo real.

Hasta no hace mucho tiempo, una de las maneras de adquirir datos físicos, provenientes de algún sistema, consistía en obtener un sistema comercial de adquisición de datos (DAQ), tales como los que ofrece la compañía National Instrument, con costos relativamente altos.

Con el aumento del uso de microcontroladores como dispositivos programables, además de las computadoras personales (PC) como sistemas de adquisición de datos, estos han ganado popularidad en la última década o más. Los microcontroladores son la opción preferida en sistemas de adquisición de datos de bajo costo debido a su bajo consumo de energía, programación, y alta velocidad [1].

La adquisición de datos ECG y tratamiento automatizado, ha sido uno de los sectores clave de la investigación en las últimas décadas. La importancia de la monitorización del ECG posee una amplia gama de aplicación en las áreas como la salud, el deporte, la milicia, los programas espaciales y los servicios de atención domiciliaria. Con el avance de las tecnologías de comunicación, apoyadas por el bajo el bajo consumo de energía, el diseño de chips de procesamiento inteligente, es posible monitorear a un paciente desde un punto remoto [1].

Los proyectos "Control en tiempo real de robots manipuladores" y "Sistema de adquisición de datos por RF utilizando MATLAB (MathWorks, Natick, Massachusetts, EUA) y el microcontrolador ATMEGA 328 (Atmel Corporate, San Jose, California, Estados Unidos)" desarrollados durante los periodos de Residencia Profesional y Tesis Profesional, dieron como resultado un sistema de adquisición de datos versátil y sumamente económico.

El presente Proyecto de Investigación se realizó bajo la supervisión del Dr. Francisco Flores García en los Laboratorios de Instrumentación del Instituto Tecnológico de la Laguna, en la ciudad de Torreón Coahuila. Así pues, en este proyecto se realizó la mejora del sistema electrónico para adquisición de datos creado anteriormente y con base en él, se han implementado programas para realizar la adquisición, transmisión alámbrica e inalámbrica de señales ECG, así como el procesamiento y análisis de los datos adquiridos mediante MATLAB y los programas de código libre con un futuro muy prometedor

llamados OCTAVE (John Wesley Eaton, Universidad de Wisconsin-Madison) y PYTHON (Python Software Foundation, Delaware, Estados Unidos).

1.2. PLANTEAMIENTO DEL PROBLEMA

Este proyecto de mejora, pretende resolver la reducida capacidad de adaptación a las necesidades de los usuarios y los altos costos de software y hardware, que son problemas comunes en los sistemas comerciales de adquisición de datos de ECG y que afectan al presupuesto de hospitales, médicos, estudiantes y de pequeñas empresas.

1.3. OBJETIVOS DE LA INVESTIGACIÓN GENERAL Y ESPECÍFICOS.

Objetivo General

Desarrollar un electrocardiógrafo (ECG) digital económico utilizando MATLAB, OCTAVE, PYTHON y el Microcontrolador PIC16F877A.

Objetivos específicos

1. Diseño del Electrocardiógrafo
2. Diseño del sistema del sistema electrónico de adquisición de datos para la señal electrocardiográfica utilizando el microcontrolador PIC16F877A
3. Configuración de los módulos de radiofrecuencia X-Bee
4. Transmisión y recepción de datos por medio de los módulos X-Bee utilizando MATLAB, OCTAVE y PYTHON
5. Graficación de la señal ECG en tiempo real utilizando MATLAB, OCTAVE y PYTHON
6. Análisis de la señal ECG

1.4. JUSTIFICACIÓN, IMPACTO SOCIAL, TECNOLÓGICO, ECONÓMICO Y AMBIENTAL. VIABILIDAD DE LA INVESTIGACIÓN.

1.4.1 Justificación

Este proyecto, propone diseñar un electrocardiógrafo (ECG) digital de bajo costo. Este diseño pretende lograr avances en la investigación en el área de electrónica, específicamente en el curso de Adquisición de Datos, para ofrecerse en complemento o como alternativa al *software* o interfaz (comercial).

El desarrollo de este tipo de herramientas ha demostrado su pertinencia al enfrentar al futuro profesional ante situaciones reales donde se interactúa directamente con los equipos físicos o con paquetes de *software* sofisticado al cual no tienen acceso usualmente los estudiantes en sus hogares.

1.4.2 Impacto social

Este proyecto será de gran ayuda para estudiantes de medicina que se encuentran realizando sus prácticas y deseen analizar diferentes patologías del corazón mediante el simulador incluido. Así se evita la necesidad de tener, en un momento preciso, un paciente mostrando una patología definida. Servirá también a los médicos que requieran de equipos portátiles donde puedan realizar diagnósticos rápidos sin la necesidad de tener que pedir al paciente que realice sus estudios antes de acudir a él.

1.4.3 Impacto tecnológico

El proyecto propone realizar un prototipo que tenga como objetivo, avanzar en la investigación en el área de electrónica, específicamente en el ambiente de adquisición de datos y diseño de electrocardiógrafos. No se prevé generación de tecnología, solo una nueva aplicación de algo que ya existe, aunque podría tener impacto en los proyectos de investigación.

1.4.4 Impacto económico

Los electrocardiógrafos comerciales tienen un valor que oscila entre los doce mil a quince mil (pesos) y el equipo que se pretenden diseñar tendrá un valor aproximado de cuatro mil pesos ó menos. Dentro del impacto económico se prevé que la mayoría de los médicos, estudiantes de medicina, ingeniería o posgrado, tengan fácil acceso a este equipo

1.4.5 Impacto ambiental

No se prevé ningún impacto ambiental positivo, debido a que al terminar la vida útil del dispositivo sus residuos electrónicos nunca podrán degradarse.

1.4.6 Viabilidad de la investigación

Este proyecto es viable porque se cuenta con:

Los conocimientos, las instalaciones y los recursos necesarios para llevarlo a cabo.

El tiempo de desarrollo es relativamente corto, los costos de los materiales son bajos y es un producto de gran utilidad.

2. MARCO TEÓRICO

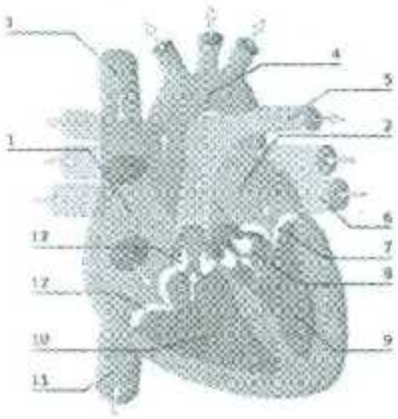
2.1 Electrocardiografía (ECG)

2.1.1 El corazón

El corazón humano es esencialmente una bomba muscular compleja cuya función es mantener la sangre en movimiento a través del cuerpo y distribuirla a todas las células, mediante un sistema de conductos.

El corazón de un ser humano adulto pesa aproximadamente unos 300 gramos y es del tamaño de un puño. Consiste de cuatro cámaras: dos aurículas (derecha e izquierda) y dos ventrículos (derecho e izquierdo); Las cavidades derechas e izquierdas están separadas una de otra por válvulas. Así también, existen válvulas que separan las venas y las arterias que se unen al corazón.

La válvula tricúspide separa la aurícula y el ventrículo derechos; la válvula mitral separa la aurícula y el ventrículo izquierdos. Note que las válvulas pulmonar y aórtica separan a los ventrículos derecho e izquierdo de las arterias pulmonares y aorta, respectivamente. La Figura 1 muestra el esquema del corazón humano.



1) Aurícula Derecha, 2) Aurícula Izquierda, 3) Vena Cava Superior, 4) Arteria Aorta, 5) Arteria Pulmonar, 6) Venas Pulmonares, 7) Válvula Mitral, 8) Válvula Aórtica, 9) Ventrículo Izquierdo, 10) Ventrículo Derecho, 11) Vena Cava Inferior, 12) Válvula Tricúspide, 13) Válvula Pulmonar.

Figura 1. Esquema del corazón humano [2].

2.1.2 Circulación de la sangre en el corazón

La sangre circula a través del cuerpo y regresa al corazón vía el sistema corporal de venas, para ingresar en la aurícula derecha; esta sangre sin oxígeno acarrea el bióxido de carbono y es llamada sangre venosa. De la aurícula derecha la sangre pasa hacia ventrículo derecho mediante la válvula

tricúspide, que permanece abierta hasta que el ventrículo derecho se contrae; esto es, cuando el músculo del ventrículo empuja a la sangre hacia la arteria pulmonar. Cuando la contracción del ventrículo se ha completado, la válvula pulmonar se cierra para evitar el reflujo de sangre hacia el ventrículo.

La sangre de la arteria pulmonar pasa hacia los pulmones, donde descarga el bióxido de carbono y absorbe el oxígeno por la acción de los capilares. Esta sangre re-oxigenada, sangre arterial, regresa al corazón a través de las venas pulmonares y fluye hacia la aurícula izquierda. Enseguida pasa a través de la válvula mitral hacia el ventrículo izquierdo y cuando este se contrae, impulsa a la sangre hacia la arteria aorta, a través de la válvula aórtica. La sangre que sale por la aorta circula por las ramificaciones hacia todo el cuerpo, en la Figura 2 se puede apreciar lo anterior.

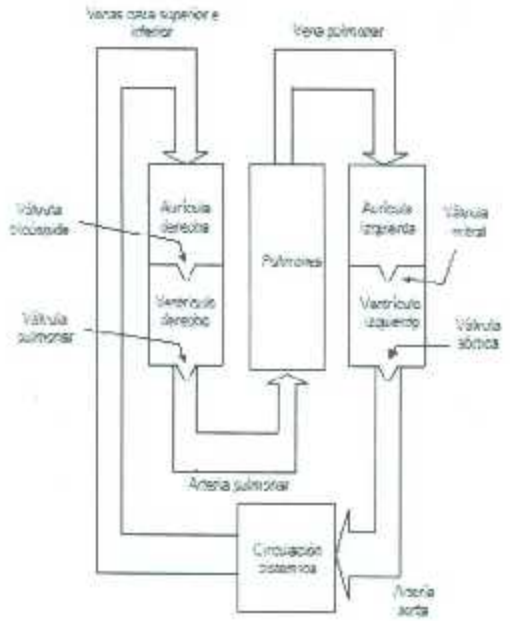


Figura 2. Circulación de la sangre en el corazón [2].

2.1.3 Funcionamiento del corazón

El funcionamiento del corazón, como bomba, se divide en dos fases:

Primera fase:

Sístole es el término usado para describir la contracción rítmica del corazón, específicamente de los ventrículos; es decir, la acción por la cual la sangre pasa del corazón hacia la arteria pulmonar y aorta. La presión de la sangre en esta fase es de alrededor de 120 mmHg, en el ventrículo izquierdo y de 25 mmHg, en el derecho.

Segunda fase:

Diástole es el término empleado para describir la dilatación de las cavidades ventriculares del corazón, durante la cual, el músculo ventricular se encuentra relajado y las cavidades auriculares llenas de sangre. Durante la fase diastólica las válvulas aórtica y pulmonar están cerradas, para evitar que la sangre de cualquiera de estas arterias regrese a los ventrículos.

El músculo cardíaco está constituido por dos tipos de tejido y cada uno de ellos actúa de manera diferente:

Músculo Cardíaco Ordinario:

Forma la mayor parte de los músculos del corazón y se trata de un músculo involuntario que actúa automáticamente. Es del tipo estriado y tiene miofibrillas que contienen filamentos de actina y miosina. Tiene como función llevar a cabo el mecanismo de contracción.

Músculo Cardíaco Especial O Autónomo:

Actúa en forma espontánea e independiente de la voluntad. Su función es la de estimular al músculo cardíaco ordinario. La excitación de cualquier parte de este músculo se transmite a todo el corazón provocando contracciones rítmicas de sístole y diástole en el músculo ordinario. Este sistema de músculo cardíaco especializado también se denomina "red de conducción cardíaca" [2].

2.1.4 Sistema de conducción cardiaco

El sistema de conducción cardíaco comprende:

Nodo Seno-Auricular (nodo S-A)

Se localiza en la aurícula derecha, cerca de la entrada de la vena cava superior. Es una región de células (de forma oval plana) que mide alrededor de 5x15 mm de área y 2mm de espesor. Cuando el corazón funciona normalmente, el estímulo automático generado por esta región controla la frecuencia cardíaca, por eso se le denomina también marcapaso.

Nodo Auriculo-Ventricular (nodo A-V)

Esta situado en la aurícula derecha, cerca de la entrada de las venas coronarias. Es de estructura reticular y tiene forma oval plana midiendo 1x6x3 mm. En su parte superior tiene una ramificación que se extiende hacia la aurícula izquierda, y la parte inferior se ramifica hacia abajo, hacia los ventrículos, conformando el Haz de His. El impulso eléctrico generado en el nodo Seno-Auricular es conducido hacia el nodo Auriculo-Ventricular, con un retraso promedio de .1segundo.

Haz De His (Haz A-V)

Este haz se bifurca en dos ramas, derecha e izquierda, mismas que están ubicadas a lo largo del septum, hacia cada uno de los ventrículos. Estas ramas se diseminan en ramillas más finas distribuidas en el endocardio de los ventrículos. Los extremos de esas ramas finas son llamados Fibras De Purkinje y conducen los impulsos eléctricos hacia todas las células musculares de los ventrículos [2].

En la Figura 3 se muestra lo anterior.

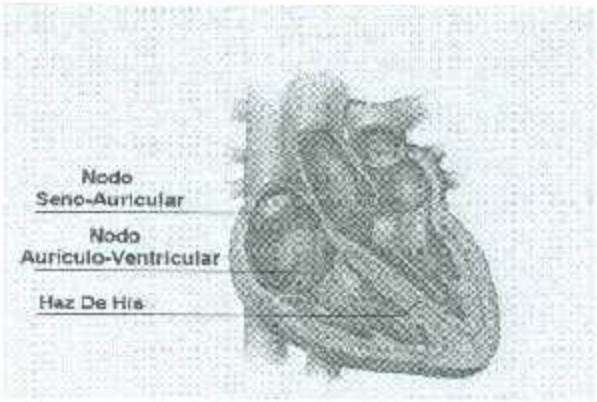


Figura 3. Sistema de conducción cardíaco [2].

2.1.5 Electrofisiología de las células cardíacas

Desde el punto de vista eléctrico, en el corazón se pueden distinguir dos tipos de células:

1. **Células automáticas o de respuesta lenta**, que suelen formar parte del sistema de conducción cardíaco.
2. **Células de trabajo o musculares o de respuesta rápida**, representadas por los miocitos.

Las células de respuesta lenta, además de conducir el impulso eléctrico poseen la propiedad de generarlo en forma espontánea.

Las células de respuesta rápida necesitan un estímulo externo que las active. En la Figura 4 se observa que las células de respuesta lenta poseen un potencial de reposo inestable que de forma automática va despolarizándose y al alcanzar el potencial de umbral generan un potencial de acción que va a transmitirse a las células vecinas. Por el contrario, las células de respuesta rápida poseen un potencial de reposo estable, necesitan un estímulo externo que lo sitúe en el potencial umbral para posteriormente, siguiendo la "ley del todo o nada", generar un potencial de acción que hará contraerse al miocito [2].

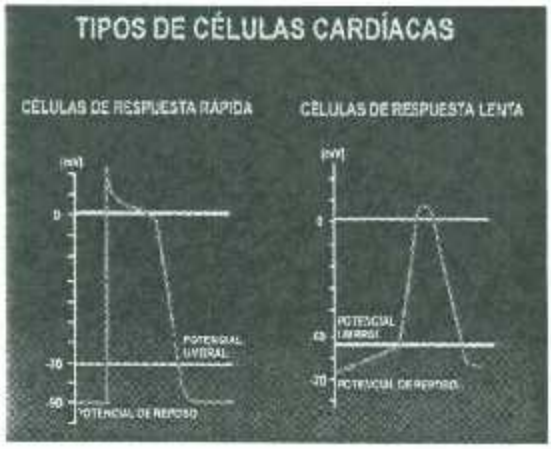


Figura 4. Tipo de células cardíacas [2].

2.1.6 Potenciales de acción de las células cardíacas

El fenómeno eléctrico generado durante el proceso de conducción del estímulo del músculo especializado hacia el músculo ordinario, puede ser registrado en forma de un electrocardiograma (ECG) que refleja la propagación eléctrica de la despolarización y la repolarización en las cámaras.

Estos fenómenos de despolarización y repolarización se llevan a cabo en distintos tiempos y velocidades, dependiendo del nivel en el que se encuentren en el sistema de conducción; en consecuencia, la suma total de cada una de estas propagaciones da origen al registro que conocemos como ECG (véase Figura 5).

La velocidad de propagación en las fibras del nodo A-V es de aproximadamente 0.01 m/seg. Después de ingresar en el nodo, la velocidad de conducción en las fibras A_V es todavía baja de 0.1 m/seg. Las fibras de Purkinje transmiten alrededor de 1.5 a 4.0 m/seg, en consecuencia transmiten casi de inmediato el impulso cardíaco a través de todo el sistema ventricular.

El tiempo transcurrido entre la transmisión de un impulso desde el haz A-V hasta las fibras de Purkinje es de alrededor de .03 seg. La masa ventricular tiene una velocidad de conducción de entre 0.4 y 0.5 m/seg. La frecuencia cardiaca en reposo es de unos 72 latidos por minuto (800 ms) [2] [GUYTON].

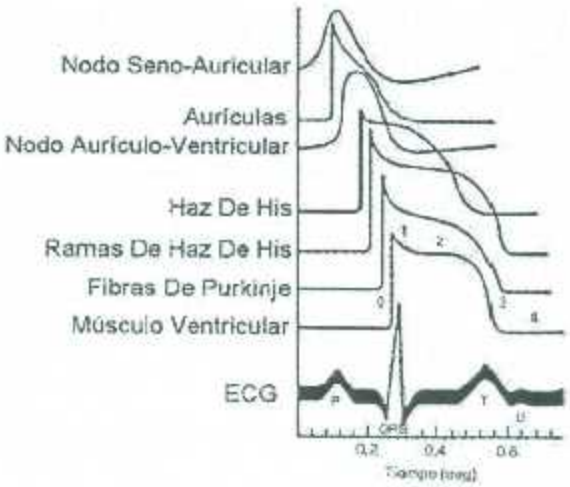


Figura 5. Potenciales de acción de las células (cardiacas) [2].

2.1.7 Nomenclatura de las ondas del electrocardiograma

Onda P

Representa la despolarización de las aurículas. Tiene una morfología redondeada; su duración es menor de 100ms y su voltaje no excede los 2,5mV.). Es positiva en todas las derivaciones salvo en la aVR del plano frontal que es negativa, y en la derivación V1 del plano horizontal.

Onda Q

Es la deflexión negativa inicial resultante de la despolarización ventricular, que precede una onda R. La duración de la onda Q es de 0.010 0.020 s no supera normalmente 0.30 s.

Onda R

Es la primera deflexión positiva durante la despolarización ventricular.

Onda S

La segunda deflexión negativa durante la despolarización ventricular.

Complejo QRS

Es la marca más característica de la señal electrocardiográfica. Representa la llegada de la señal de activación a ambos ventrículos. Su duración es de 80 a 100ms.

Onda T

Es la Deflexión lenta producida por la repolarización ventricular.

Onda U

Es una onda habitualmente positiva, de escaso voltaje, que se observa sobre todo en las derivaciones precordiales y que sigue inmediatamente a la onda T. El origen exacto de la onda U Se desconoce, aunque algunos postulan que se debe a la repolarización de los músculos papilares. Hay patologías asociadas a la onda U, como por ejemplo, cuando esta es negativa está relacionada a isquemia del miocardio.

Las porciones del electrocardiograma entre las deflexiones se denominan **segmentos**, y las distancias entre ondas se denominan **intervalos**. El ECG puede ser dividido en los siguientes segmentos e intervalos:

Segmento S-T

Es un periodo de inactividad que separa la despolarización ventricular de la repolarización ventricular. Este segmento es normalmente isoeléctrico y va desde el final del complejo QRS hasta el comienzo de la onda T.

Intervalo R-R

Es la distancia que existe entre dos ondas RR sucesivas. En un ritmo sinusal este intervalo debe mantenerse prácticamente constante, la medida de él dependerá de la frecuencia cardiaca que tenga el paciente.

Intervalo P-P

Es la distancia que existe entre dos ondas P sucesivas. Al igual que el intervalo RR, el intervalo PP debe ser muy constante y su medida depende de la frecuencia cardiaca.

Intervalo P-R

Representa el retraso fisiológico que sufre el estímulo que viene de las aurículas a su paso por el nodo auriculo-ventricular. Éste se mide desde el comienzo de la onda P hasta el inicio de la onda Q ó de la onda R. Su duración debe estar comprendida entre los 120 y 200ms.

Intervalo QRS

Este mide el tiempo total de despolarización ventricular. Se mide desde el comienzo de la inscripción de la onda Q ó R hasta el final de la onda S. Los valores normales de este intervalo se encuentran entre 0.06 y 0.10s.

Intervalo Q-T

Se extiende desde el comienzo del complejo QRS hasta el final de la onda T y representa la sístole eléctrica ventricular, o lo que es lo mismo, el conjunto de la despolarización y la repolarización de los ventrículos. Su duración estará entre 320 y 400 ms.

En la Figura 6 se muestra lo explicado anteriormente.

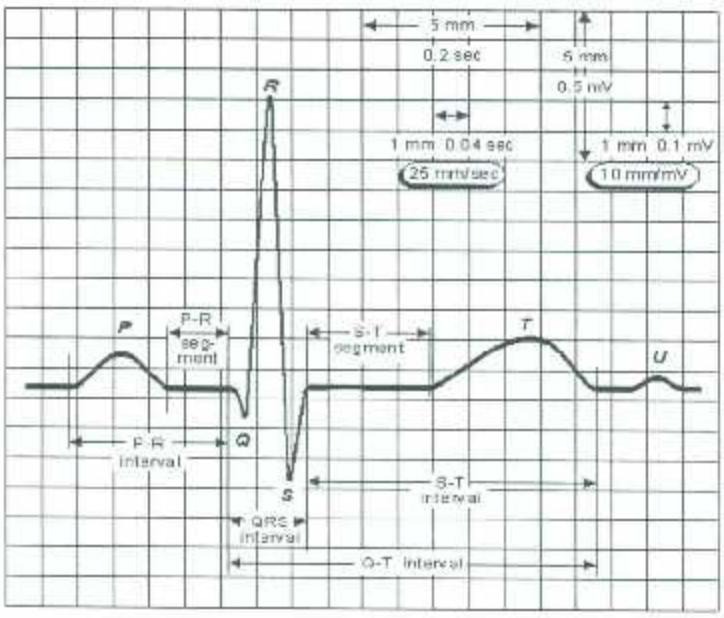


Figura 6. ECG [2].

2.1.8 Derivaciones electrocardiográficas

El objetivo fundamental de un ECG es registrar las señales eléctricas del corazón en la superficie del cuerpo, mediante un proceso completamente no invasivo. La información así obtenida permite a los médicos cardiólogos determinar el estado funcional del corazón.

Las derivaciones electrocardiográficas son disposiciones específicas de los electrodos sobre el cuerpo humano y se conocen simplemente como "derivaciones" y en la práctica clínica se utilizan un número de hasta doce derivaciones estándar, clasificadas según el modo de registrarlas, en derivaciones del plano frontal y derivaciones del plano horizontal. Las derivaciones del plano frontal pueden ser bipolares o monopolares [2].

Derivaciones Bipolares ó de Einthoven

El primer registro exitoso del ECG fue realizado por el fisiólogo Willem Einthoven, quien ideó un instrumento usando un galvanómetro conectado a los brazos de un sujeto para medir las variaciones de energía que resultaban en la sístole cardíaca. Einthoven utilizó tres derivaciones llamadas DI, DII y DIII. Las derivaciones bipolares creadas por Wilhen Einthoven registran la diferencia de potencial eléctrico que se produce entre dos puntos del cuerpo. La Figura 7 muestra el triángulo de Einthoven.

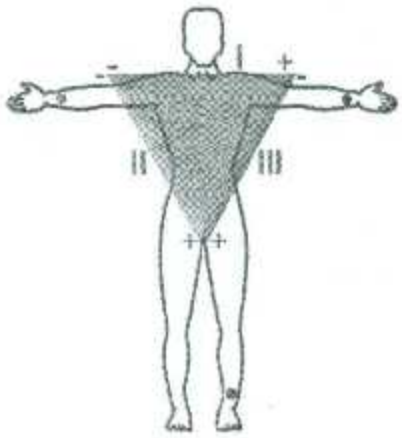


Figura 7. Triángulo de Einthoven [2].

Para su registro, las derivaciones bipolares utilizan 3 electrodos ubicados en: Brazo derecho RA, Brazo izquierdo LA, Pierna Izquierda LL. Las tres derivaciones son denominadas DI, DII y DIII

DI: Registra la diferencia de potencial entre el brazo izquierdo (polo positivo) y el brazo derecho (polo negativo).

DII: Registra le diferencia de potencial que existe entre la pierna izquierda (polo positivo) y el brazo derecho (polo negativo).

DIII: Registra la diferencia del potencial que existe entre la pierna izquierda (polo positivo) y el brazo izquierdo (polo negativo).

Se puede notar, el método de medición es simple pero, la desventaja consiste en que es imposible medir el voltaje absoluto de cada arista. Cada una de estas derivaciones se miden mediante el empleo de un amplificador diferencial, la Figura 8 ofrece esquemas que ilustran cada caso.

Figura 8. Derivaciones Bipolares ó de Einthoven [2].

Derivaciones Unipolares de Wilson

Frank Norman Wilson (1890-1952) y colegas (Macleod, y Barker) investigaron sobre los potenciales unipolares de electrocardiografía, siendo estos medidos con referencia a un terminal de potencial absoluto estándar llamado "Terminal Central De Wilson", (TCW), el cual se formaba conectando resistencias de 5kΩ en cada de las extremidades llevadas a un punto común.

Wilson sugirió que los potenciales unipolares deben medirse con respecto a este terminal y registrar el potencial absoluto total en un punto del cuerpo. Este método pretende corregir la ambigüedad de las derivaciones bipolares de Einthoven. La configuración de Wilson se conecta a un aparato de registro del que sale el electrodo explorador, el cual toma el potencial absoluto (V): Brazo derecho (VR), Brazo izquierdo (VL), Pierna izquierda (VF). La aplicación de estas derivaciones hace posible medir los potenciales eléctricos de todos los puntos, usando el tórax como el estándar.

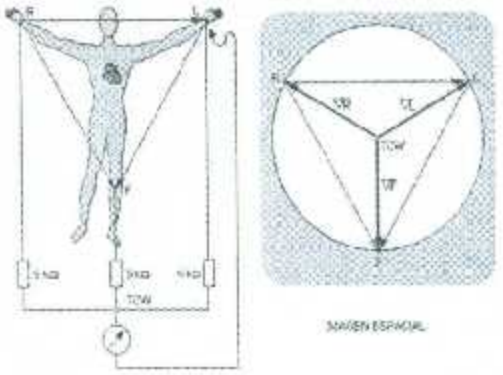


Figura 9. Derivaciones Unipolares de Wilson [2].

Derivaciones Aumentadas De Goldberg

Goldberger modificó el sistema de derivaciones de Wilson y consiguió aumentar la amplitud de la onda hasta en un 50%; de aquí que estas derivaciones se llamen aVR, aVL, aVF, donde la a significa "ampliada ó aumentada" [2].

Las derivaciones son :

1) aVR: Brazo derecho (+) y Brazo izquierdo − Pierna Izquierda (-).
2) aVL: Brazo izquierdo (+) y Brazo derecho + Pierna Izquierda (-)
3) aVF: Pierna izquierda (+) y Brazo derecho − Brazo izquierdo (-)

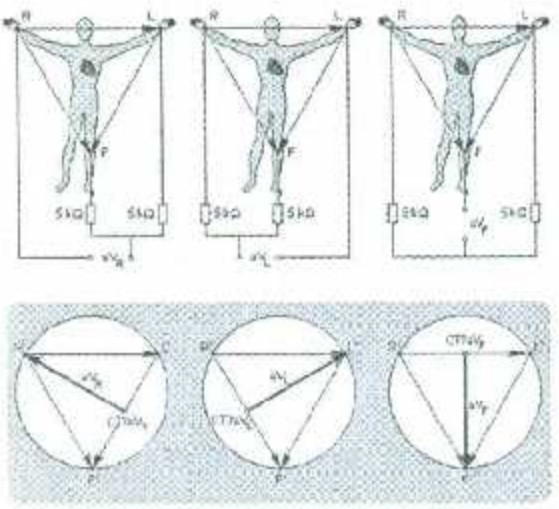


Figura 10. Implementación de las derivaciones aumentadas de Goldberg [2].

Derivaciones del plano horizontal ó precordiales

Son derivaciones verdaderamente monopolares ó unipolares, pues comparan la actividad del punto en que se coloca el electrodo a nivel precordial (electrodo explorador), contra la suma de los tres miembros activos o Terminal Central (LL + LA + RA, que da como resultado cero).

La localización precordial de los electrodos es la siguiente (véase Figura 11):

- **V1**: intersección del 4to espacio intercostal derecho con el borde derecho del esternón.
- **V2**: intersección del 4to espacio intercostal izquierdo con el borde izquierdo del esternón
- **V3**: a mitad de distancia entre V2 y V4.
- **V4**: intersección del 5to espacio intercostal izquierdo y línea medio clavicular.
- **V5**: intersección del 5to espacio intercostal izquierdo y línea axilar anterior.
- **V6**: Intersección del 5to espacio intercostal izquierdo y la línea axilar anterior.

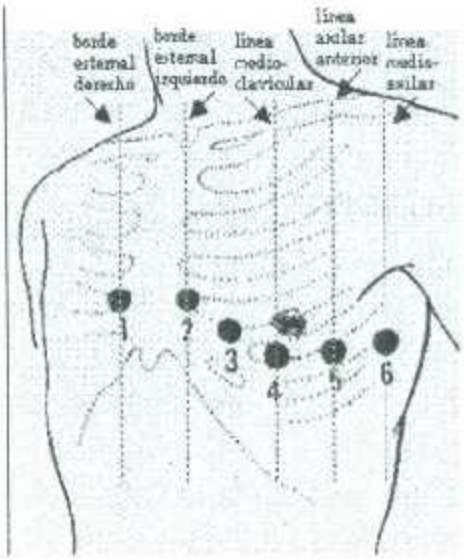


Figura 11. Localización precordial de los electrodos [2].

En la Tabla 1 se muestra el modo en que se obtiene cada una de las derivaciones específicas y su cálculo [2]:

Tabla 1. Derivaciones específicas y su cálculo.

Derivación	Tipo	Cálculos
I	Extremidad	LA - RA
II	Extremidad	LL - RA
III	Extremidad	LL - LA
aVR	Aumentada	RA - (LA+LL)/2
aVL	Aumentada	LA - (RA+LL)/2
aVF	Aumentada	LL - (RA + LA)/2
V1	Precordial	V1-(RA+LA+LL)/3
V2	Precordial	V2-(RA+LA+LL)/3
V3	Precordial	V3-(RA+LA+LL)/3
V4	Precordial	V4-(RA+LA+LL)/3
V5	Precordial	V5-(RA+LA+LL)/3
V6	Precordial	V6-(RA+LA+LL)/3

La Figura 12 muestra un esquema de los oscilogramas correspondientes a las doce derivaciones estudiadas en los apartados anteriores. Las amplitudes en mV pico a pico, con ±5% de incertidumbre son DI=1.00; D2=1,25; D3=0.25; aVR=1.12; aVL=0.37 y aVF=0.75.

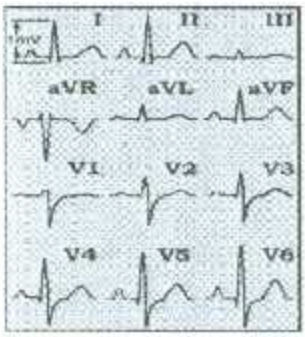


Figura 12. Oscilogramas correspondientes a las doce derivaciones [2].

Durante un estudio electrocardiográfico, la colocación correcta de los electrodos es de suma importancia y para evitar confusiones al colocarlos, se ha asignado a cada uno de ellos un color específico, convencionalmente aceptado por la comunidad clínica. La Tabla 2 indica el código de colores y su posición en el cuerpo humano [2].

Tabla 2. Código de colores y posición de electrodos en el cuerpo humano.

Posición Electrodos	Color
Mano derecha	
Pie derecho	
Mano izquierda	
Pie izquierdo	
V1	
V2	
V3	
V4	
V5	
V6	

2.1.9 El electrocardiógrafo

El electrocardiógrafo es un equipo que permite medir y visualizar la diferencia de potencial eléctrico generada por el corazón, entre dos puntos específicos del cuerpo. La magnitud de los potenciales electrocardiográficos está en el orden de los milivoltios (mV) y las componentes de frecuencia de dicha señal están en el rango de 0 a 200 Hertz (Hz). Sin embargo, para fines de diagnóstico médico, es suficiente que el aparato responda en un ancho de banda de 0.1 a 100 Hz.

Los electrocardiógrafos se clasifican según su funcionamiento, su sistema de despliegue, su sistema de alimentación y su número de canales, en la Figura 13 se explica lo anterior.

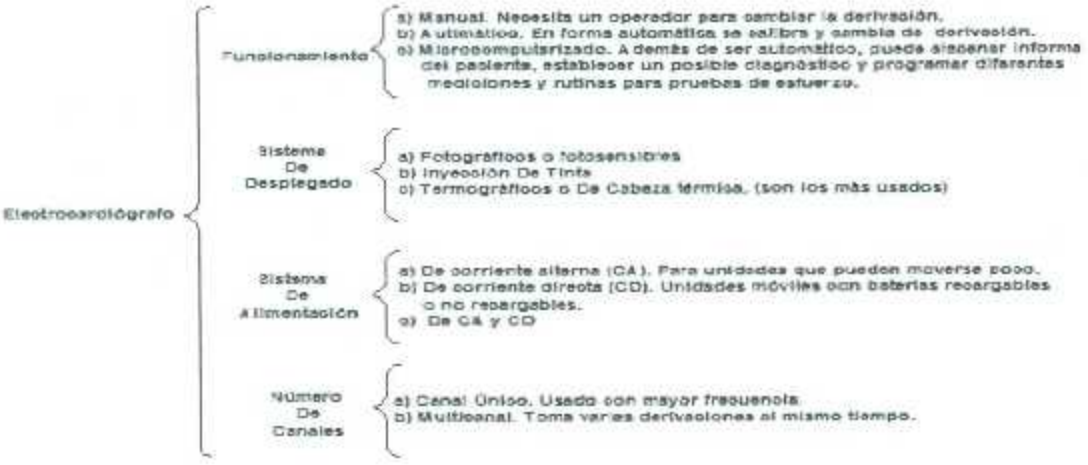


Figura 13. Cuadro sinóptico sobre los ECG [2].

Por último, en la Figura 14 se muestra un diagrama a bloques de cómo está compuesto un electrocardiógrafo.

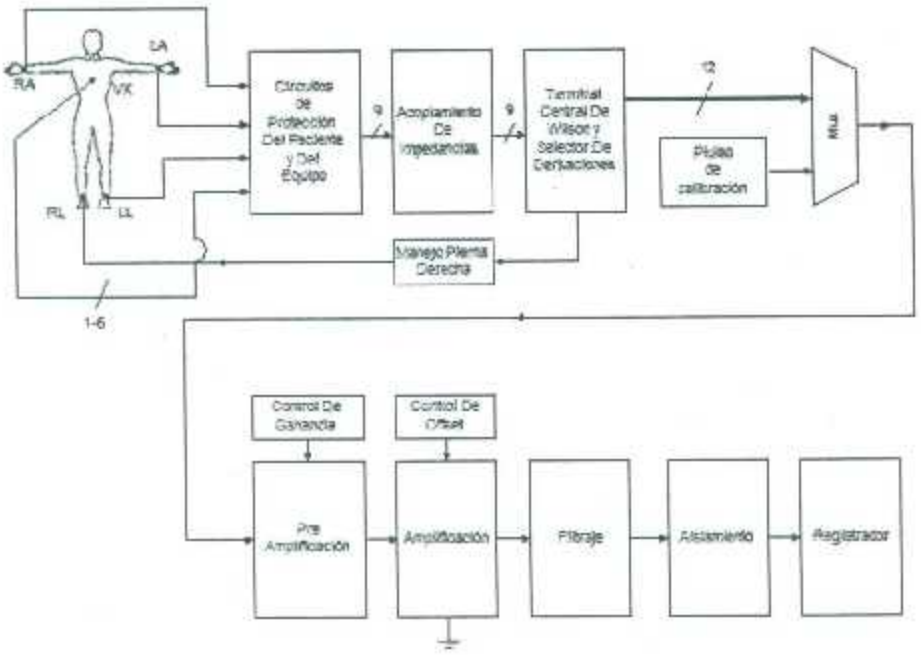


Figura 14. Diagrama a bloques de un electrocardiógrafo [2].

2.2 Descripción del Microcontrolador PIC16F877A

El microcontrolador PIC16F877 (Microchip Technology Inc., Chandler, Arizona, Estados Unidos) pertenece a una gran familia de microcontroladores de 8 bits. Microchip ha dividido sus microcontroladores en tres grandes subfamilias de acuerdo al número de bits de su bus de instrucciones (véase Tabla 3) [3].

Tabla 3. Subfamilias de los microcontroladores de Microchip.

Subfamilia	Bits del bus de instrucciones	nomenclatura
Base - Line	12	PIC12XXX y PIC14XXX
Mid – Range	14	PIC16XXX
High - End	16	PIC17XXX y PIC18XXX

En la Figura 15 se muestra una serie de los diferentes empaquetados del Microcontrolador PIC16F877 [3].

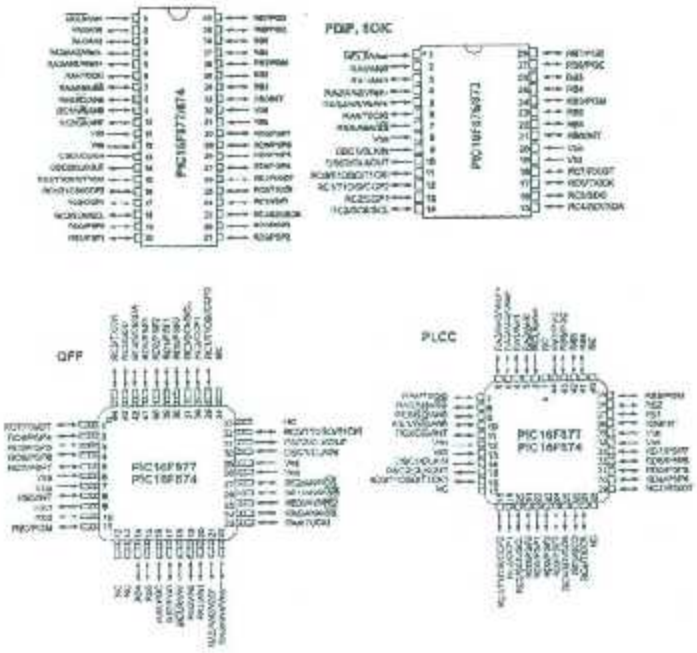


Figura 15. Empaquetados Microcontrolador PIC16F877A [3].

2.2.1 Características generales del PIC16F877A

La siguiente es una lista de las caracteristicas que comparte el PIC16F877 con los dispositivos más cercanos de su familia [3]:

- Sólo 35 instrucciones que aprender
- Todas las instrucciones se ejecutan en un ciclo de reloj, excepto los saltos que requieren dos
- Frecuencia de operación de 0 a 20 MHz (DC a 200 nseg de ciclo de instrucción)
- Hasta 8k x 14 bits de memoria Flash de programa
- Hasta 368 bytes de memoria de datos (RAM)
- Hasta 256 bytes de memoria de datos EEPROM
- Hasta 4 fuentes de interrupción
- Stack de hardware de 8 niveles
- Lectura/escritura de la CPU a la memoria flash de programa
- Rango de voltaje de operación de 2.0 a 5.5 volts
- Alta disipación de corriente de la fuente: 25mA

Periféricos

- Timer0: Contador/Temporizador de 8 bits con pre-escalador de 8 bits
- Timer1: Contador/Temporizador de 16 bits con pre-escalador
- Timer2: Contador/Temporizador de 8 bits con pre-escalador y post-escalador de 8 bits y registro de periodo.
- Dos módulos de Captura, Comparación y PWM
- Convertidor Analógico/Digital: de 10 bits, hasta 8 canales
- Puerto Serie Síncrono (SSP)

- Puerto Serie Universal (USART/SCI).
- Puerto Paralelo Esclavo (PSP): de 8 bits con líneas de protocolo

2.2.2 Descripción de la CPU

La CPU es la responsable de la interpretación y ejecución de la información (instrucciones) guardada en la memoria de programa. Para operar sobre la memoria de datos además, si se van a realizar operaciones lógicas o aritméticas, requieren usar la Unidad de Lógica y Aritmética (ALU, por sus siglas en inglés).

La ALU controla los bits de estado (Registro STATUS), los bits de este registro se alteran dependiendo del resultado de algunas instrucciones [3].

2.2.3 Organización de la memoria del PIC

Los PIC tienen dos tipos de memoria: Memoria de Datos y Memoria de programa, la memoria de datos a su vez se divide en

Registros de Propósito Especial (SFR): Son localidades asociadas específicamente a los diferentes periféricos y funciones de configuración del PIC y tienen un nombre específico asociado con su función.

Registro de Propósito General (GPR): Son memoria RAM de uso general

Toda la memoria de datos está organizada en 4 bancos numerados (0, 1, 2 y 3) y está implementada en RAM estática. Para seleccionar un banco se debe hacer uso de los bits del registro STATUS denominados IRP, RP1 y RP0. En las posiciones más bajas de cada banco se encuentran los SFR, y arriba de éstos se encuentran los GPR.

En la Figura 16 se muestra los bancos de memoria del PIC.

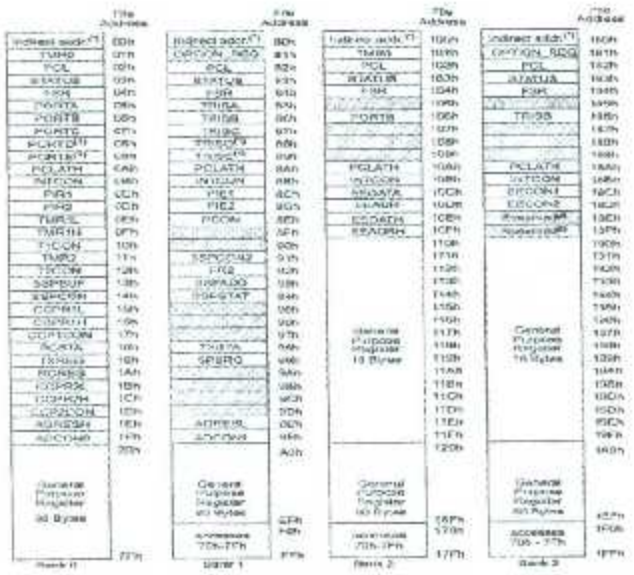


Figura 16. Bancos de memoria del PIC [3].

2.3. MATLAB

2.3.1 Descripción de MATLAB

MATLAB ha sido aceptado universalmente como una de las más potentes plataformas de procesamiento de datos. Su conectividad con muchos lenguajes de programación avanzados como C y Java y la disponibilidad de una amplia gama de cajas de herramientas hace que sea popular entre la comunidad científica y de investigación [1].

MATLAB es un lenguaje de alto funcionamiento para computación técnica, Este integra computación, visualización, y programación, en un entorno fácil de usar donde los problemas y las soluciones son expresados en la más familiar notación matemática [4]. Los usos más familiares de MATLAB son:

- Matemáticas y Computación
- Desarrollo de algoritmos
- Modelamiento y simulación
- Análisis de datos, exploración y visualización
- Gráficas científicas e ingenieriles
- Desarrollo de aplicaciones, incluyendo construcción de interfaces graficas de usuario

2.3.2 Partes de MATLAB

El sistema MATLAB consiste de cuatro partes principales:

Entorno de desarrollo:

Es el conjunto herramientas y módulos que ayudan a usar las funciones y archivos de MATLAB. Muchas de esas herramientas son interfaces graficas de usuario. Esto incluye, el escritorio de MATLAB, la ventana de comandos, el historial de comandos, un editor y un depurador, navegadores para revisión de la ayuda, el espacio de trabajo o *workspace* y los archivos.

La libreria de funciones matemáticas: esta es una gran colección de algoritmos computacionales que van desde funciones eleménteles como la suma, la función seno y coseno, y la aritmética de números complejos hasta funciones mucho más sofisticadas como inversas de matrices, auto valores de matrices, funciones de *bessel*, y transformadas radiadas de Fourier.

El lenguaje MATLAB:

Es un lenguaje de alto nivel para matrices con sentencias para control de flujo, creación de funciones y estructuras de datos, funciones de entrada/salida y algunas características de programación orientada por objetos. Este lenguaje permite tanto la programación a pequeña escala para la creación rápida de programas, como programación a larga escala para la realización de aplicaciones complejas.

Gráficas:

MATLAB cuenta con módulos extensivos para la visualización de vectores y matrices en forma de graficas, así como para realizar comentarios e impresión de estas gráficas. MATLAB incluye funciones de alto nivel para la visualización de datos en dos y tres dimensiones, procesamiento de imágenes, animación, y creación de gráficos de presentación. MATLAB también incluye funciones de bajo nivel que permiten personalizar completamente la apariencia de los gráficos así como construir interfaces graficas de usuario para las aplicaciones.

Interfaz gráfica (GUIDE)

Una interfase gráfica es el vinculo entre el usuario y un programa computacional, constituida generalmente por un conjunto de comandos o menús, instrumentos y métodos por medio de los cuales el usuario se comunica con el programa durante las operaciones que se desean realizar, facilitando la entrada y salida de datos e información.

MATLAB nos permite realizar GUIs de una manera sencilla usando una herramienta llamada GUIDE (*GUI Development Environment*). La forma de implementar las GUI con MATLAB es crear los objetos y definir las acciones que cada uno va realizar [5].

2.4. OCTAVE

2.4.1 Descripción de OCTAVE

OCTAVE es un lenguaje de alto nivel inicialmente desarrollado para operaciones numéricas. Proporciona una interfaz de línea de comandos para resolver problemas numéricos tanto lineales como no lineales [6].

Otra característica importante de este software, es que es de distribución gratuita y se apega a la filosofía GNU, esto es poder tener acceso al programa y al código fuente del programa para modificarlo (si nos interesa), sin tener que pagar ni por su uso, ni por su obtención, además de poder hacer cuantas copias se quieran e instalaciones en diferentes máquinas, también, sin tener que pagar [6].

Incluye herramientas para resolver problemas numéricos comunes de álgebra lineal, búsqueda de raíces de ecuaciones no lineales, integración de funciones ordinarias, manipulación de polinomios, integración de ecuaciones diferenciales algebraicas y generación de gráficas (usando GNU *Plot*).

Su tipo de datos fundamental es la matriz; integra compatibilidad con números complejos e incluye una potente y extensiva librería de funciones matemáticas, que puede ser extendida mediante funciones definidas por el usuario.

La sintaxis de operación en línea de comandos y para la creación de scripts es muy similar a MATLAB. Esto se hace escribiendo los comandos a utilizar y el software responderá a cada uno de ellos mostrando los resultados a través de la pantalla.

2.4.2 Descripción de Cygwin

Existen varias versiones de OCTAVE y PYTHON, todas disponibles en forma gratuita en Internet. La versión original de cada una de ellas se utiliza en LINUX (Red Hat, Raleigh, Carolina del Norte), a pesar de ello existen varias versiones para Windows y para ello se utiliza el programa Cygwin.

Cygwin es una colección de herramientas desarrollada por *Cygnus Solutions* (Red Hat, Raleigh, Carolina del Norte) para proporcionar un comportamiento similar a los sistemas Unix en Microsoft Windows. Aunque los programas portados funcionan en todas las versiones de Windows, su comportamiento es mejor en Windows NT, Windows XP y Windows Server 2003 [7].

Indispensable para ejecutar algunas aplicaciones desarrolladas para Linux y Unix, Cygwin proporciona una compatibilidad aceptable y un buen rendimiento, siendo muy útil en escenarios con plataformas mixtas.

2.5. PYTHON

2.5.1 Descripción de PYTHON

Básicamente, Python es un lenguaje de programación de alto nivel, interpretado y multipropósito. En los últimos años su utilización ha ido constantemente creciendo y en la actualidad es uno de los lenguajes de programación más empleados para el desarrollo de *software*.

Python puede ser utilizado en diversas plataformas y sistemas operativos, entre los que podemos destacar los más populares, cómo Windows (en este caso), Mac OS X y Linux.

Pero, además, Python también puede funcionar en *smartphones*. Nokia desarrolló un intérprete de esté lenguaje para su sistema operativo *Symbian* (Symbian Ltd, Southwark, Londres) [8].

¿Tiene Python un ámbito específico? Algunos lenguajes de programación sí que lo tienen. Por ejemplo, PHP fue ideado para desarrollar aplicaciones *Web*. Sin embargo, esté no es el caso de Python. Con este lenguaje podemos desarrollar software para aplicaciones científicas, para comunicaciones de red, para aplicaciones de escritorio con interfaz gráfica de usuario (GUI, por sus siglas en inglés), para crear juegos, para *smartphones* y por supuesto, para aplicaciones *web* [8].

2.6. Comparativa MATLAB, OCTAVE y PYTHON

Antes de empezar, es importante destacar que MATLAB es una herramienta muy capaz y en algunos campos (aún) no tiene rival. El gran problema de MATLAB es el coste de las licencias junto a algunas inconsistencias de su lenguaje. El coste de una licencia de MATLAB al día de hoy es de aproximadamente 36000 pesos mexicanos (2000€) más los toolboxes que se requieran (18000$ c/u) más 90000$ para el compilador que nos permitirá comercializar nuestro trabajo [4].

Si bien existen varias alternativas a MATLAB tales como Scilab, FreeMat y OCTAVE, el uso de PYTHON para sustituir MATLAB va ganando adeptos. A continuación se muestra una comparativa de cada una estas herramientas.

OCTAVE: es el lenguaje más compatible con MATLAB pero tiene la "desventaja" de estar protegido con licencia GPL. por lo que, sin entrar en detalle, deberás facilitar el código de tu aplicación (comercial o no) siempre que se te solicite. Por otro lado, OCTAVE carece de una GUI IDE, lo que quiere decir es que carece de interfaz gráfica para el desarrollo de código y lo hace menos atractivo que el MATLAB [6].

FreeMat: a pesar de ser más o menos compatible con MATLAB posee licencia GPL también y una interfaz gráfica muy básica.

Scilab: pese a ser de licencia similar a la GPL pero francesa (CECILL) y no es compatible de forma directa con MATLAB, aunque existe conversores. Sin embargo tiene la ventaja de tener una IDE muy similar a la de MATLAB.

A pesar de que las tres opciones de arriba salvan los costes de la licencia de uso de MATLAB, todas tienen las mismas limitaciones técnicas. Además ninguna de ellas perfectamente compatible con MATLAB y las comunidades que hay detrás de ellas son relativamente pequeñas.

PYTHON: por otro lado, está cubierto por la licencia BSD por lo que se permite su comercialización sin ningún tipo de restricción (no tendrás que facilitar el código si no lo deseas) y sus paquetes son gratuitos [8]. En otras características que posee se encuentran:

- Es un lenguaje de programación bien diseñado, fácil de leer y escalable por lo que tu código será útil para pequeñas pruebas como para usos industriales.
- La integración con otras herramientas y lenguajes es casi siempre posible.
- Multiplataforma, se puede ejecutar tanto en Linux como en MacOS como en Windows.
- Existe una cantidad de información y documentación en internet incomparable con el resto de alternativas.

En resumen, para todos los usuarios que desean "migrar" de MATLAB a otras plataformas, las mejores opciones para esto (en lo personal), serían OCTAVE y PYTHON, que son las que se utilizarán durante este proyecto de investigación. Algo interesante será el observar el rendimiento de cada uno de los programas mencionados y destacar cada una de sus ventajas y desventajas.

2.7. Adquisición de datos

2.7.1 Descripción de Adquisición de Datos

La adquisición de datos o adquisición de señales, consiste en la toma de muestras del mundo real (sistema analógico) para generar datos que puedan ser manipulados por un ordenador u otro equipo electrónico (sistema digital). Se requiere una etapa de acondicionamiento, que adecúa la señal a niveles compatibles con el elemento que hace la transformación a señal digital. El elemento que hace dicha transformación es el módulo de digitalización o tarjeta de Adquisición de Datos (DAQ) [9].

La adquisición de datos se inicia con el fenómeno físico o la propiedad física de un objeto (objeto de la investigación) que se desea medir. Esta propiedad física o fenómeno podría ser el cambio de temperatura o la temperatura de una habitación, la intensidad o intensidad del cambio de una fuente de luz, la presión dentro de una cámara, la fuerza aplicada a un objeto, o muchas otras cosas. Un eficaz sistema de adquisición de datos puede medir todas estas diferentes propiedades o fenómenos.

Un sensor es un dispositivo que convierte una propiedad física o fenómeno en una señal eléctrica correspondiente medible, tal como tensión, corriente, el cambio en los valores de resistencia o condensador, etc. La capacidad de un sistema de adquisición de datos para medir los distintos fenómenos depende de los transductores para convertir las señales de los fenómenos físicos medibles en la adquisición de datos por hardware. Los transductores son

sinónimo de sensores en los sistemas de adquisición de datos. Hay transductores específicos para diferentes aplicaciones, como la medición de la temperatura, la presión, o el flujo.

Las señales pueden ser digitales ó analógicas en función del transductor utilizado. El acondicionamiento de señales suele ser necesario si la señal desde el transductor no es adecuado para la tarjeta DAQ que se utiliza. La señal puede ser amplificada o reducida, o puede requerir de filtrado. Varios otros ejemplos de acondicionamiento de señales podrían ser el divisor de tensión, el aislamiento, la linealización, etc.

En ocasiones el sistema de adquisición es parte de un sistema de control, y por lo tanto, la información recibida se procesa para obtener una serie de señales de control. En la Figura 17 se puede ver los bloques que componen el sistema de adquisición de datos:

Figura 17. Esquema de bloques de un sistema de adquisición de datos [9].

Como se puede apreciar, los bloques principales son los siguientes:

- Transductor
- El acondicionamiento de señal
- El convertidor analógico-digital
- La etapa de salida (interfaz con la lógica)

El transductor es un elemento que convierte la magnitud física que se va a medir en una señal de salida (normalmente tensión o corriente) que puede ser procesada por nuestro sistema.

El acondicionamiento de señal es la etapa encargada de filtrar y adaptar la señal proveniente del transductor a la entrada del convertidor analógico / digital.

El convertidor analógico/digital es un sistema que presenta en su salida una señal digital a partir de una señal analógica de entrada, (normalmente de tensión) realizando las funciones de cuantificación y codificación

La etapa de salida es el conjunto de elementos que permiten conectar el sistema de adquisición de datos con el resto del equipo, y puede ser desde una serie de buffers digitales incluidos en el circuito convertidor, hasta una interfaz RS-232(en este caso utilizada) para conectar a un ordenador o estación de trabajo [9].

2.8. Los módulos Xbee

2.8.1 Descripción de los módulos XBee

Los módulos XBee son soluciones integradas que brindan un medio inalámbrico para la interconexión y comunicación entre dispositivos. Estos módulos utilizan el protocolo de red llamado IEEE 802.15.4 para crear redes *FAST POINT-TO-MULTIPOINT* (punto a multipunto); o para redes *PEER-TO-PEER* (punto a punto). Fueron diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible. Por lo que básicamente XBee es propiedad de Digi basado en el protocolo Zigbee [10].

Zigbee Es una alianza y un estándar de redes *MESH* de eficiencia energética y de costos. XBee utiliza el estándar Zigbee y lo agrega y envuelve en un pequeño empaque elegante (véase Figura 18), en términos simples, los XBee son módulos inalámbricos fáciles de usar [10].

Figura 18. Módulo Xbee [10].

2.8.2 Modo de comunicación

Actualmente, los dispositivos ZigBee de Digi (Digi International, Minnetonka, Estados Unidos) tienen 2 maneras para comunicarse. Esta el modo de transmisión serial

transparente (*Modo AT*), que ha sido utilizada en el proyecto y el modo de comunicación API. La comunicación AT se asemeja a lo que sería una transmisión a través de un puerto serial, ya que el dispositivo se encarga de crear la trama y el dato que llegue a la terminal TX será enviado de forma inalámbrica, por lo cual se considera como el modo más sencillo para utilizar estos nodos, su principal desventaja es que para enviar información a distintos nodos es necesario entrar constantemente al modo configuración para cambiar la dirección de destino.

Por otro lado, el modo conocido como *API*, un microcontrolador externo se debe encargar de crear una trama especifica al tipo de información que se va a enviar, este modo es recomendado para redes muy grandes donde no se puede perder tiempo entrando y saliendo del modo configuración de los dispositivos. Para redes con topología en malla éste es el modo a utilizar [10].

2.9. Filtros activos

Un filtro, es un circuito eléctrico que tiene la habilidad de "elegir" y dejar pasar ciertas componentes de frecuencia de una señal eléctrica, además de rechazar a las restantes.

Los circuitos de filtro son muy importantes en electrónica y pueden clasificarse según diferentes criterios:

2.9.1 Clasificación por el tipo de componentes

Los filtros se pueden clasificar en pasivos y activos (véase Figura 19). Los filtros pasivos son todos aquellos que están constituidos únicamente por componentes pasivos; es decir, componentes que no requieren de polarización adicional para procesar la señal, tales como resistencias (R), capacitores (C) e inductores (L).

Por el contrario, los filtros activos utilizan además de resistencias, capacitores e inductores, otros componentes activos que requieren de ser polarizados tales como diodos (D) transistores (T), transistores de efecto de campo (FET), amplificadores operacionales (OPAMP), etc [11].

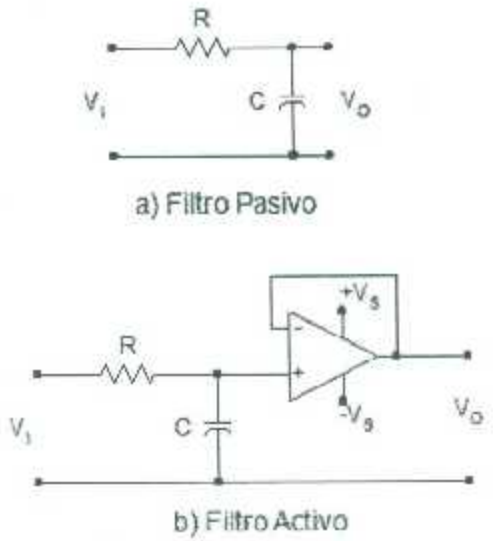


Figura 19. Filtros Pasivos y Activos [11].

En general, los filtros pasivos tienen la ventaja de ser sencillos y económicos pero tienen la gran desventaja de alterar sus propiedades de filtraje ante las perturbaciones de la carga, ya que ésta forma parte del circuito que determina la frecuencia de corte del filtro. Por esta razón no se recomienda conectarlos en cascada

En contraste, los filtros activos tienen la ventaja de poseer una alta o moderada impedancia de entrada y una baja impedancia de salida, lo que los capacita para tener características de filtraje estables ante perturbaciones de la carga y además poder colocarlos en cascada, sin alterar sus propiedades. Las desventajas principales que tienen los filtros activos son la necesidad de ser polarizados y su relativo alto costo.

2.9.2 Clasificación por el tipo de la respuesta a la frecuencia

Los filtros se pueden clasificar también por el tipo de respuesta que ofrecen ante la frecuencia. Estos son, filtros Pasa Bajas, Pasa Altas, Pasa Banda y de Rechazo De Banda.

La Figura 20 presenta las respuestas ideales de los filtros mencionados. La región ó regiones de la respuesta de frecuencia para la cual los filtros presentan una salida con magnitud A_0, se denomina el "paso de banda" del filtro; contrariamente, la región o regiones que presentan una respuesta con magnitud de cero es llamada "banda suprimida" [11].

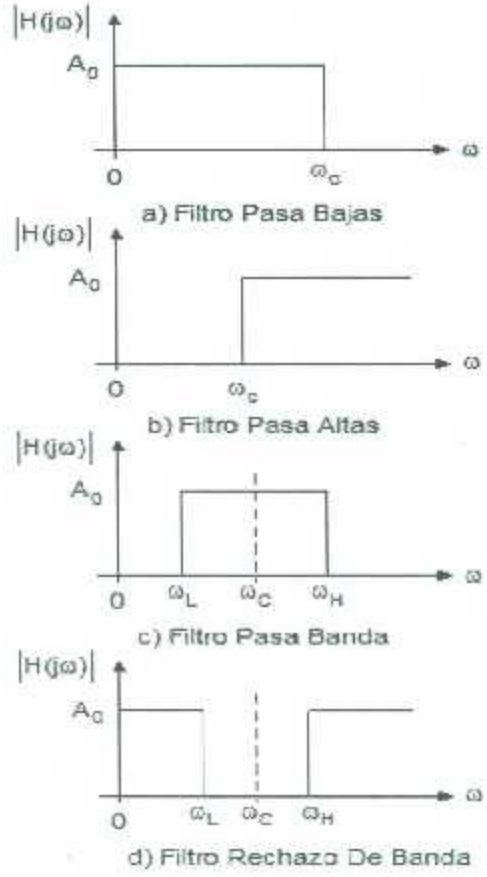


Figura 20. Clasificación por el tipo de la respuesta a la frecuencia [11].

2.9.3 Clasificación por la forma de la respuesta a la frecuencia

Los filtros, además, se pueden clasificar con base en la forma que tiene su respuesta a la frecuencia, en Filtros *Butterworth*, Filtros *Chebyshev*, Filtros De Respuesta Elíptica y Filtros *Bessel* (véase Figura 21).

Los filtros Butterworth se caracterizan por poseer una respuesta máximamente plana en su paso de banda, con una atenuación moderada de la salida a razón de -20n decibeles por década, en la región de supresión de su banda; n es el orden del filtro [11].

A Diferencia, los filtros *Chebyshev* se caracterizan por presentar fluctuaciones cosenoidales en la región de su paso de banda, pero la pendiente de la atenuación es mucho más pronunciada que la de los *Butterworth*, con el mismo orden de filtro [11].

Los filtros *Bessel* y de respuesta elíptica, además de presentar fluctuaciones en la región de paso de banda, presentan fluctuaciones en la región de banda suprimida, sin embargo las pendientes de atenuación son superiores a las de los filtros anteriores [11].

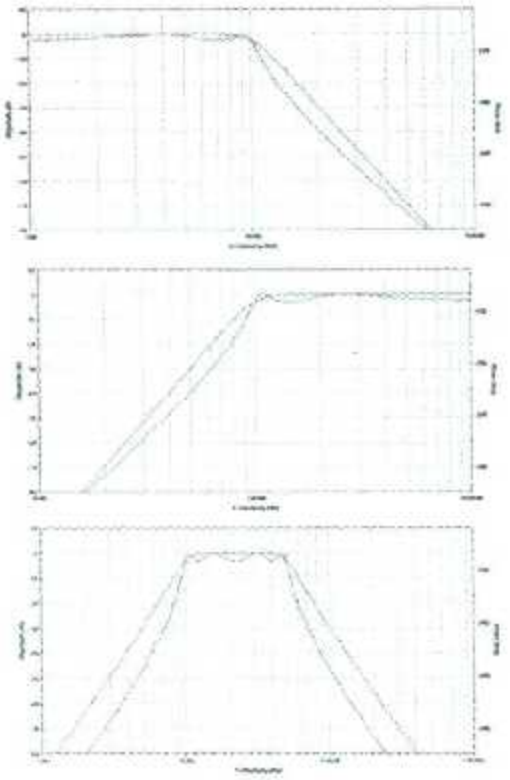


Figura 21.Clasificación por la forma de la respuesta a la frecuencia [11].

2.9.4 Respuesta a la frecuencia

Al estudiar los filtros y sus propiedades, es fundamental que el estudiante haya comprendido el concepto de "Respuesta A La Frecuencia". La respuesta a la frecuencia se refiere a "la manera en cómo un filtro va cambiando su capacidad de amplificación ó atenuación, según sea la frecuencia (o frecuencias) de la señal aplicada en su entrada".

En la práctica, lo que se hace para comprobar cómo responde un sistema ante una frecuencia particular, es inyectar en su entrada una señal senoidal con la frecuencia de interés, ω, y una amplitud de referencia, X (véase Figura 22).

Seguidamente, se mide la amplitud de la señal de salida, Y, y el ángulo de fase, Φ, y se anota la ganancia (G=Y/X), para esa frecuencia particular. Si el analista desea conocer la característica de respuesta a la frecuencia, para un rango particular de frecuencias, puede "barrer" la frecuencia de la señal de entrada desde una frecuencia de inicio, ω1, hasta una frecuencia final, ω2, y obtener las gráficas de la magnitud de la ganancia y de la fase, ambas, contra la frecuencia (véase Figura 23) [11].

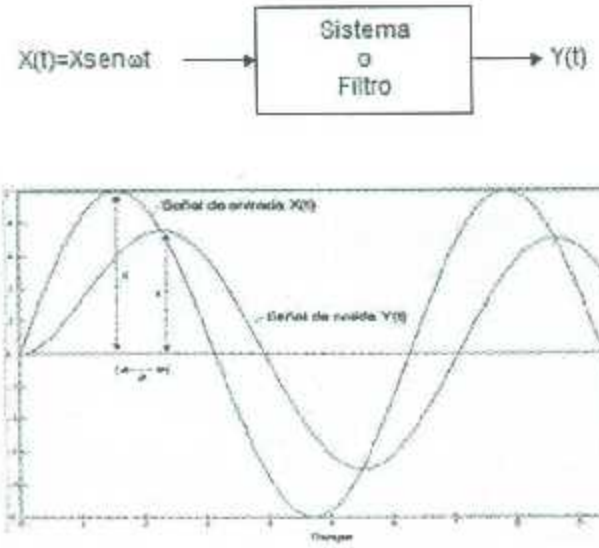


Figura 22. Respuesta de un sistema ante una frecuencia particular [11].

Para realizar las pruebas de respuesta a la frecuencia se utiliza siempre la señal senoidal debido a que esta señal es elemental y contiene solamente una componente de frecuencia. Esto, evidentemente garantiza que la señal que se obtiene a la salida del sistema es, efectivamente, la respuesta del sistema ante la frecuencia particular de la entrada [11].

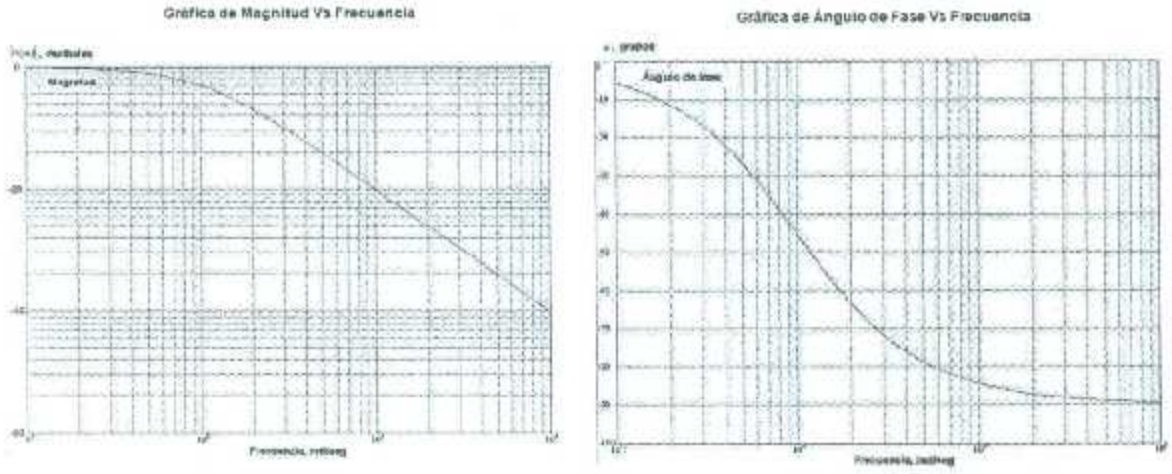


Figura 23. Graficas de Magnitud vs Frecuencia y Fase vs Frecuencia de un sistema en particular [11].

3. IMPLEMENTACIÓN DEL ECG DIGITAL CON MATLAB, OCTAVE Y PYTHON

3.1 Descripción de actividades

3.1.1 Graficación en tiempo real utilizando MATLAB

La primera parte que conforma al proyecto de investigación corresponde al diseño de los graficadores para la señal ECG. En seguida se describe la elaboración de cada uno de ellos.

Tomando en cuenta el Anexo C y el subtema 2.3, se crea una interfaz gráfica (GUI) en MATLAB la cual posee la siguiente estructura:

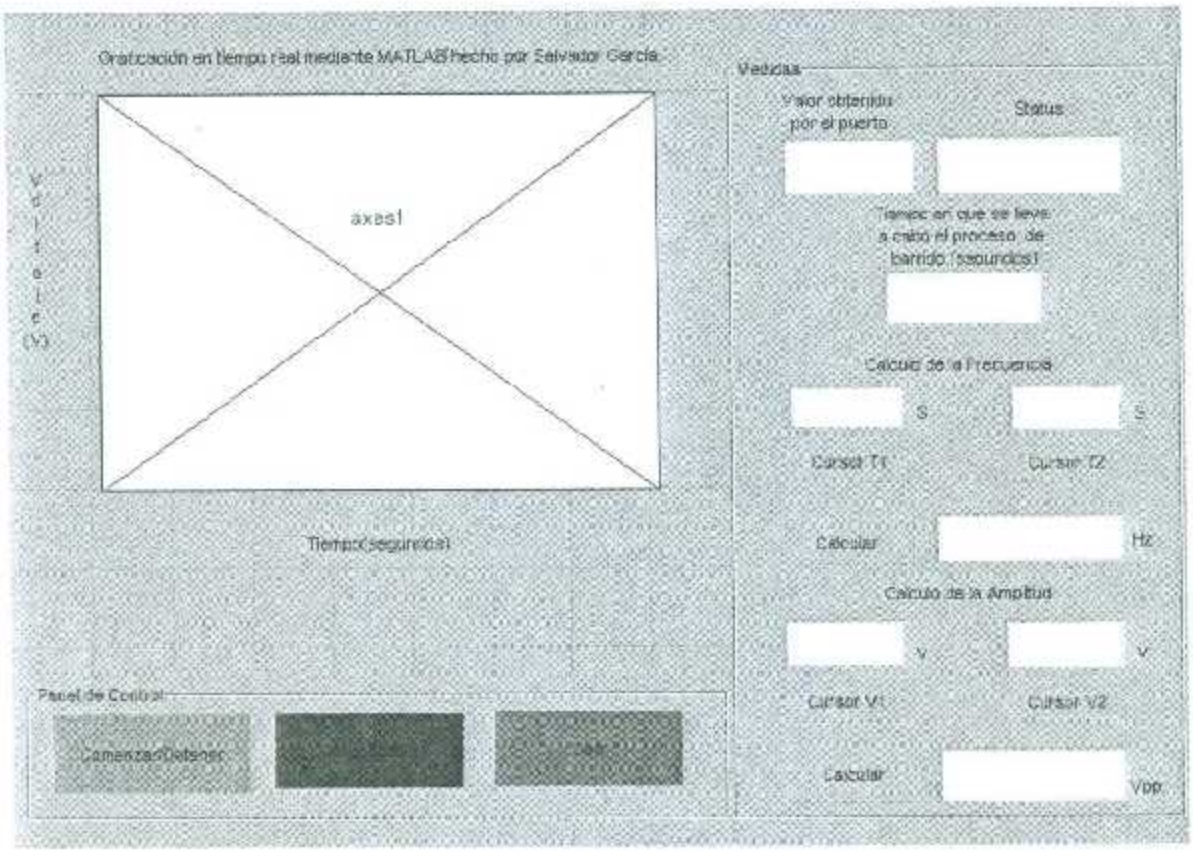


Figura 24. Interfaz gráfica creada en MATLAB.

Como se puede observar en la Figura 24, La GUI posee un Panel de control con 3 botones (Comenzar/Detener, Limpiar Gráfico y Salir), diversas ventanas para observar información de interés tales como el valor obtenido por el puerto, amplitud, frecuencia y por último y más importante, la ventana de graficación.

Ya que se ha dado la estructura anterior, lo siguiente es ingresar el código a cada "widget" como se estudió en el Anexo C. A continuación se muestra el script junto con su respectiva explicación.

El programa esta divido en tres bloques para su mejor entendimiento. Cabe resaltar que el programa principal se encuentra "dentro" del widget del botón "toogle" llamado "Comenzar/Detener". El primer bloque consiste en la inicialización del puerto serie, se asignan propiedades como el "Baudrate", el tamaño y bit de parada entre otros. En el siguiente bloque se procede a la creación del objeto línea, la declaración de diversas variables que se explicarán posteriormente, se habilitan los cursores y las propiedades del gráfico como por ejemplo, los límites del eje *X* y él *Y* y la cuadrícula de la gráfica. Todo esto se puede observar en la Figura 25.

```
%Captura de datos mediante MATLAB hecho por Salvador Garcia
 delete(instrfind({'Port'},{'COM20'})); %ajustar puerto serie
 %Creación del objeto serie
  s=serial('COM20');
  set(s,'Baudrate',9600); % se configura la velocidad a 9600 Baudios
  set(s,'StopBits',1); % se configura bit de parada a uno
  set(s,'DataBits',8); % se configura que el dato es de 8 bits, debe estar entre 5 y 8
  set(s,'Parity','none'); % se configura sin paridad
  set(s,'Terminator','CR');% caracter con que finaliza el envio 'CR/LF'
  set(s, 'FlowControl', 'none'); % control de flujo "ninguno"
  fopen(s);
%Creacion del objeto linea
tmax = 10; % tiempo de captura 10s
rate = .13; % resolucion (resultado experimental)
lf=0;
t=0;
ylim([0 6]);
set(gca,'XTick',0:1:10)
xlim([0 10])
grid on
hold on
%==========Habilitar cursores==================
datacursormode on
%==============================================
d=1;
l(d) = line('Color','g','LineWidth',2);
i=1;
y = zeros(1,tmax*rate);
%Graficación
```

Figura 25. Primer y segundo bloque del código de la GUI creada para graficación en tiempo real hecha en MATLAB.

El tercer y último bloque es el más importante, en el, se realiza la graficación de los valores obtenidos por el puerto serial. Todo el proceso está dentro de la sentencia while en modo true; el primer paso es inicializar un cronometro mediante las instrucciones *tic* y *toc*, estas funciones se encargarán de que cada iteración dentro de la sentencia sea ejecutada durante cada segundo. Enseguida, se verifica si se ha pulsado el botón toogle Comenzar/Detener, si es así, debería haberse obtenido un valor de "uno" y con ello proseguir al siguiente paso, de lo contrario el programa permanecería estático esperando una condición verdadera. A continuación, se procede a la lectura del puerto y la conversión del valor obtenido a un formato de número entero para ser manipulado sin ningún problema por el programa, si el número procesado fuera mayor a 1024 (dado que el convertidor A/D del microcontrolador tiene una resolución de diez bits y su escala estará dentro del rango de 0 a 1024), la ventana de "*status*" informará que se ha producido un error debido a que no se realizo la conversión correctamente y se debe presionar nuevamente el botón "Comenzar". Por otra parte, si se realizó una conversión satisfactoria, se mostrará en la ventana de información el valor obtenido por el puerto, el status "listo y graficando", se efectuará la conversión de la escala del convertidor del microcontrolador (0 a 1024), a un rango que vaya desde cero a cinco y con ello pueda ser interpretado cada valor en términos de voltaje para su mejor análisis, posteriormente se almacenará cada dato procesado en la variable vector "*y*" de tamaño *rate*tmax* (variables que serán explicadas).

Lo siguiente es la creación de un objeto línea "dinámico" para dar el efecto de graficación a través del tiempo, esto se logra de la siguiente manera. Durante cada ciclo, mediante la instrucción *linspace* establecemos que nuestro objeto línea declarado anteriormente, comprenderá un espacio de cero a la variable límite final (*lf*) con un número de puntos (*i*) dentro del espacio del objeto línea. Realizado lo anterior, mediante la instrucción *set* se define cada valor que poseerá el objeto línea tanto en el eje *X* como en el *Y* en cada vuelta del ciclo while. Enseguida, con ayuda del comando *drawnow* se "dibujará" la acción anterior y en cada vuelta se actualizará la gráfica.

El siguiente paso es incrementar el contador (*i*) y modificar el límite final del objeto línea, esto se logra incrementando *lf* más la variable establecida como *rate*, la cual, será un número elegido experimentalmente y será la proporción en la que el objeto línea se desplaza a través del eje *X* para cumplir con el tiempo de barrido establecido por la variable *tmax* (en este caso 10 segundos). Cabe recordar, que el porqué de establecer un espacio de *tmax*rate* para el vector (*y*) es debido a que cuando graficamos en MATLAB ó cualquier otro programa, las dimensiones tanto del eje *X* como del *Y* deben ser exactamente del mismo tamaño para la primera iteración y así evitar errores de compilación.

En el momento que el cronometro (*tic toc*) llegué a los diez segundos, se desplegará en una ventana de información el tiempo en que se llevó a cabo el proceso de barrido, se deberá limpiar el graficador, reiniciar el contador (*i*), crear un nuevo objeto línea, reiniciar el límite

final (*if*) y reiniciar el cronometro para comenzar con un nuevo tiempo de barrido. Todo lo anterior se puede apreciar en la Figura 26.

```
while true %t<tmax
    t = toc;%conteo en segundos (cada iteración se realizará en 1 segundo)
bandera=get(handles.UpDown,'Value');
if bandera==1
    try% leer del puerto serie
     %m4 = fscanf(s,'%d');
     m=fread(s,100);%400
     m2=char(m(11:15));
     m3=str2num(m2);
     m4=m3(1)*1000+m3(2)*100+m3(3)*10+m3(4);
     if m4>1024
         set(handles.status, 'String','error,press again start..');
         break;
     end
     set(handles.status, 'String','ready and plotting..');
     set(handles.Muestra, 'String',m4);
     y(i)=(m4*5)/1024;
     x = linspace(0,lf,i);
     set(l(d),'YData',y(1:i),'XData',x);
     drawnow;
     i = i+1;
     lf=lf+rate;
     if t>tmax
      t2=floor(t);
      set(handles.Tproceso, 'String',t2);
      i=1;
      t=0;
      cla;
      lf=0;
      d=d+1;
      l(d) = line('Color','g','LineWidth',2);
      tic %comienza nuevamente el cronometro
     end
    catch
```

Figura 26. Tercer bloque del código de la GUI creada para graficación en tiempo real hecha en MATLAB.

Cálculo de la frecuencia y la amplitud.

Es de vital importancia que cuando se adquiere y procesa una señal se debe conocer tanto la amplitud como la frecuencia para su posterior análisis, es por esta razón que dentro de la GUI realizada (véase figura 24) se han agregado controles para conocer dichas medidas. A continuación se explica el proceso de medición.

Para conocer estos valores es relativamente sencillo, solo se requiere utilizar cursores y conocer las coordenadas exactas dentro la grafica para calcular las medidas. El primer paso es habilitarlos mediante la función *datacursormode on* (véase figura 25), después, en cada

widget de los botones "Cursor T1, T2, V1 y V2" se agregan propiedades a los cursores, se obtiene información general de ellos cada vez que se pulsen en el objeto figura con la instrucción *getCursorInfo ( )* guardándose en una variable llamada *c_info,* enseguida se obtiene la posición del eje *X* y *Y* de la estructura anterior (*c_info.Position*) que está dada en un arreglo de 1X2 en donde la posición uno (1) de este mismo vector corresponde a la coordenada del eje *X* y la posición dos (2) al eje *Y,* en la Figura 27 se muestra lo anterior.

```
% --- Executes on button press in T1.
function T1_Callback(hObject, eventdata, handles)
% hObject    handle to T1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
dcm_obj = datacursormode(Serial3);
set(dcm_obj,'DisplayStyle','datatip',...
'SnapToDataVertex','off','Enable','on')
c_info = getCursorInfo(dcm_obj);
a=c_info.Position;
```

Figura 27. Código de los "widgets" de los botones "Cursor T1, T2, V1 y V2".

Ya que se obtienen las coordenadas, para calcular la frecuencia se requiere que tanto del cursor T1 como del T2 se obtenga el eje *X* del vector (*c_info.Position*), lo siguiente sería realizar una resta entre el valor de los dos cursores (*T2-T1*) y al resultado sacar su inverso (1/T2-T1) para obtener la frecuencia y ser mostrado en la ventana de información, el código anterior se muestra en la Figura 28.

```
% --- Executes on button press in calcularf.
function calcularf_Callback(hObject, eventdata, handles)
% hObject    handle to calcularf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
b1=eval(get(handles.DISPT2,'String'));
a1=eval(get(handles.DISPT1,'String'));
dp=b1-a1;
fr=1/dp;
set(handles.frecu, 'String',fr);
```

Figura 28. Código para calcular la frecuencia.

Por último, para conocer la amplitud se siguen los mismos pasos que para la frecuencia solo que en este caso se obtiene de los cursores V1 y V2 el eje *Y* del vector (*c_info.Position*), se realiza una resta entre ellos (*V1-V2*) y se muestran los resultados en la ventana de información, todo el proceso se muestra en la Figura 29.

```
% --- Executes on button press in calculary.
function calculary_Callback(hObject, eventdata, handles)
% hObject    handle to calculary (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
z1=eval(get(handles.DISPV1,'String'));
w1=eval(get(handles.DISPV2,'String'));
dv=z1-w1;
set(handles.volt, 'String',dv);
```

Figura 29. Código para calcular la amplitud.

Pruebas para la Interfaz Gráfica (Prototipo para adquisición de datos):

Terminada la interfaz gráfica, lo siguiente es realizar algunas pruebas, para ello, se debe tener en cuenta el ANEXO E, los subtemas 2.2, 2.7, 2.8 y tomar las siguientes consideraciones.

La Figura 30 esquematiza conceptualmente el prototipo para la adquisición de datos.

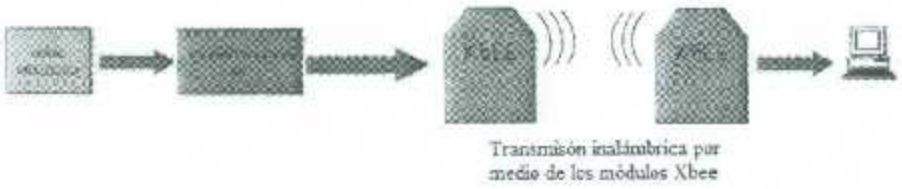


Figura 30. Esquema del prototipo para la adquisición de datos.

Bloque del microcontrolador

En la Figura 31, vemos un circuito esquemático de este bloque, quien comanda todos las acciones: gestión de datos, comunicación con la PC (bloque comunicación serie), y el bloque ADC.

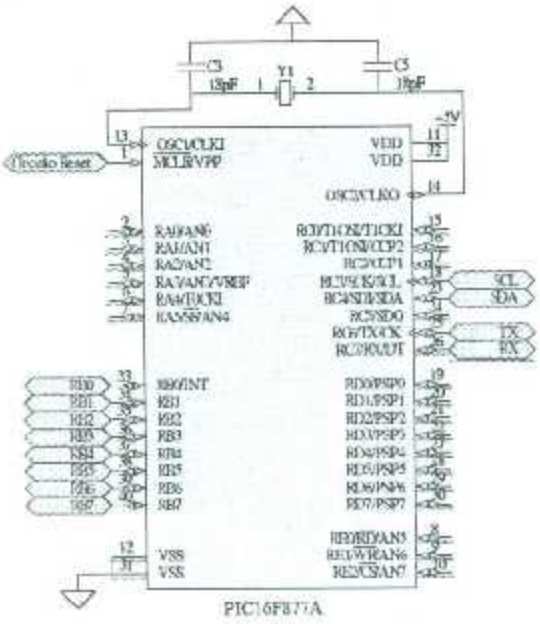


Figura 31. Circuito esquemático del bloque del microcontrolador.

Programa residente en el microcontrolador

En la Figura 32 se presenta un diagrama de flujo del programa, el cuál, gobierna todas las acciones del prototipo. Este mismo fue realizado en lenguaje ensamblador mediante el programa MPLAB y por último fue grabado en la memoria de programa del microcontrolador.

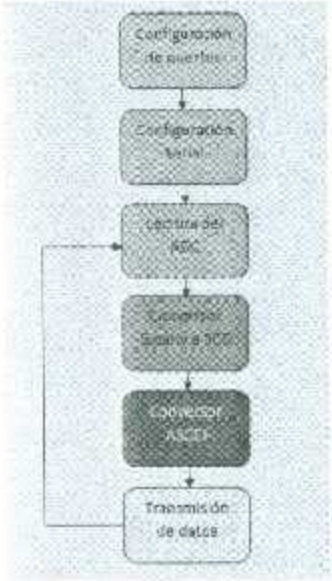


Figura 32. Programa residente en el microcontrolador.

Bloque de comunicación inalámbrica por medio de los módulos Xbee.

El módulo (véase Figura 33), se alimenta con 3.3 ó 5 V, se hace la conexión a tierra y se conectan las líneas de transmisión de datos por medio del UART (TXD y RXD) para comunicarse con el microcontrolador.

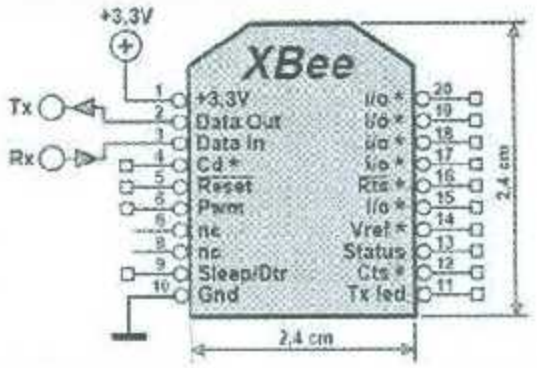


Figura 33. Bloque de comunicación inalámbrica por medio de los módulos Xbee.

Resultados de las pruebas del prototipo.

Para realizar las pruebas, se ingresa una señal sinusoidal previamente acoplada (el circuito de acoplamiento se explicara posteriormente), de frecuencia de 1.2 Hz y amplitud de 1 Vpp, estos valores de voltaje se convierten a un formato de serie (RS232) y se transmiten inalámbricamente a la PC por medio de los módulos de transmisión RF (Xbee). La interfaz gráfica desarrollada en MATLAB realiza la adquisición para su posterior procesamiento y análisis. En la Figura 34 se muestra los resultados.

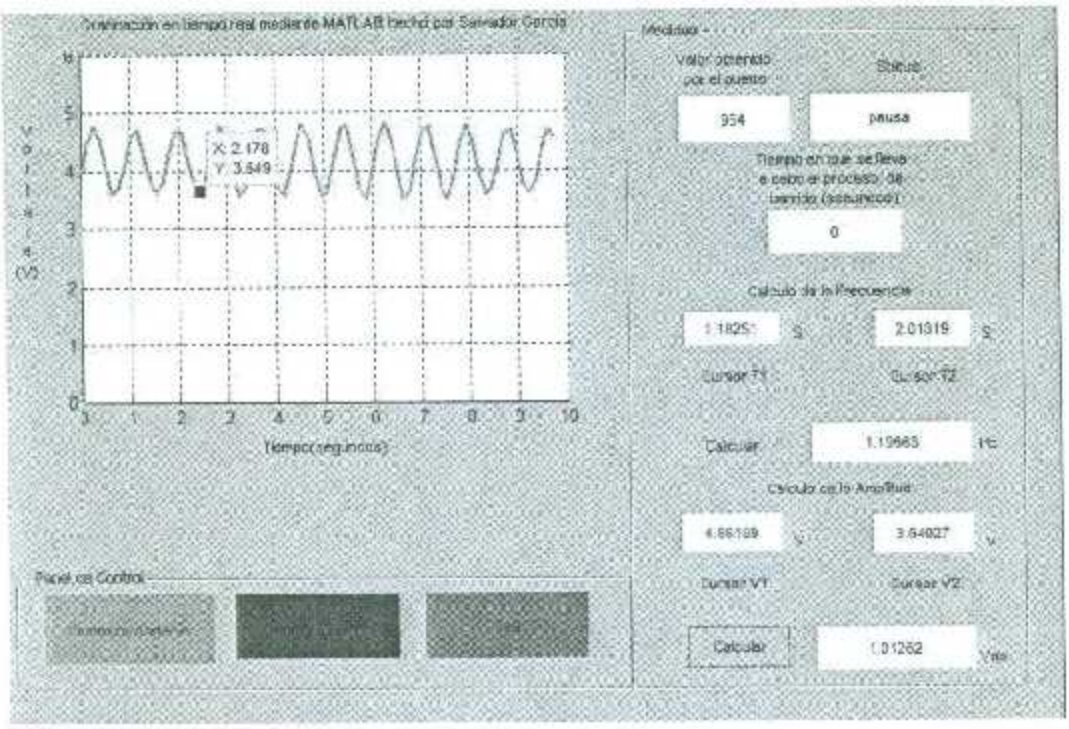


Figura 34. Resultados de la interfaz gráfica diseñada para graficación en tiempo real hecha en MATLAB.

3.1.2 Graficación en tiempo real utilizando OCTAVE

Como se podrá recordar, parte de los objetivos del proyecto de investigación es desarrollar un ECG digital que tenga la posibilidad de ser funcional con diferentes programas y ser compatible en diferentes sistemas operativos. El siguiente graficador se desarrolló utilizando OCTAVE, tomando en cuenta los ANEXOS A, B y C y el subtema 2.4 el programa se explica a continuación.

Es importante mencionar que el programa esta divido en 3 bloques principales, en el primero de ellos se cargan las librerías necesarias para el programa, estos son el paquete *zenity* para la pseudo interfaz gráfica (véase ANEXO C), la herramienta *instrument-control* para comunicación serial y el paquete *graphics_toolkit fltk*, que como su nombre lo indica, se utiliza para graficación. A continuación, se coloca una barra de progreso con ayuda del paquete zenity para inicializar el programa principal y una vez llegado al cien por ciento se abre el archivo *"figurastart.m"* (que se explicará posteriormente), este mismo, inicia un objeto figura para mostrar la ventana de presentación del programa. Inmediatamente, aparecerá una ventana de información que indica si se desea comenzar con la graficación en tiempo real y si la condición es verdadera, se ejecutará el archivo *"gtr_zenity2.m"* (que se explicará posteriormente), de lo contrario, se cierra el objeto figura y todo lo que se haya ejecutado anteriormente. Enseguida se muestra el código y los resultados en las Figuras 35 y 36.

```
#=======================LIBRERIAS_REQUERIDAS=========================
graphics_toolkit fltk
pkg load zenity
pkg load instrument-control
more off
#=======================INICIALIZANDO_PROGRAMA=======================
x=zenity_progress('Progreso','auto-close');
#
zenity_progress(x,0,'Despertando al programador');
#pause(1.5);
pause(1);
#
zenity_progress(x,12.5,'Haciendo un cafe');
pause(1);
#
zenity_progress(x,25,'Cargando librerias');
pause(1);
#
zenity_progress(x,37.5,'Verificando puertos');
pause(1);
#
zenity_progress(x,50,'Tomando asiento');
pause(1);
#
zenity_progress(x,62.5,'Enlazando a Zenity y Octave');
```

```
pause(1);
#
zenity_progress(x,75,'Creando interfaz');
pause(1);
#
zenity_progress(x,87.5,'Falta poco!');
pause(1);
#
zenity_progress(x,99.9,'Listo, Bienvenido a Graficacion en Tiempo Real con Octave hecho por Salvador Garcia!');
pause(1);
#
zenity_progress(x,100);
#==========================PROGRAMA_PRINCIPAL==========================
#Abrimos objeto figura
figurastart
pause(1.5)
#
z=zenity_message ('Deseas comenzar?', 'question');
#
if z==0
#
  gtr_zenity2
#
 else
```

clf

close all

clear all

clc

endif

Resultados:

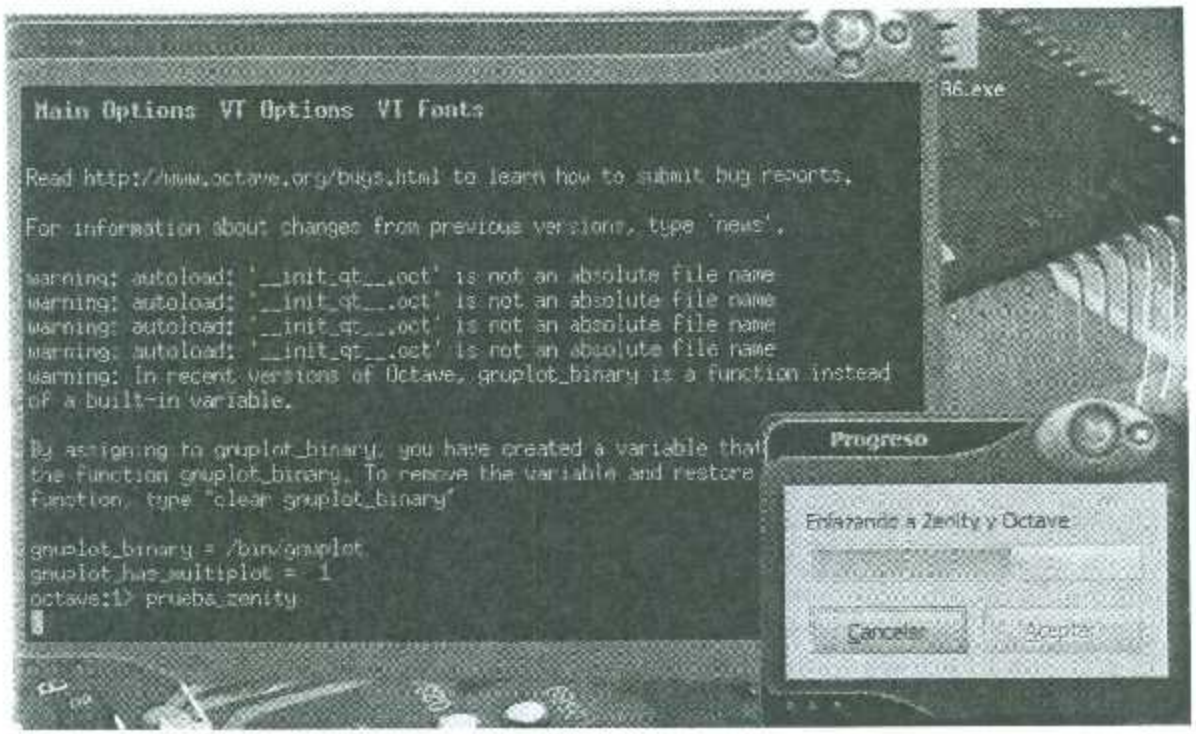


Figura 35. Barra de progreso para inicializar la interfaz gráfica en OCTAVE (primer bloque).

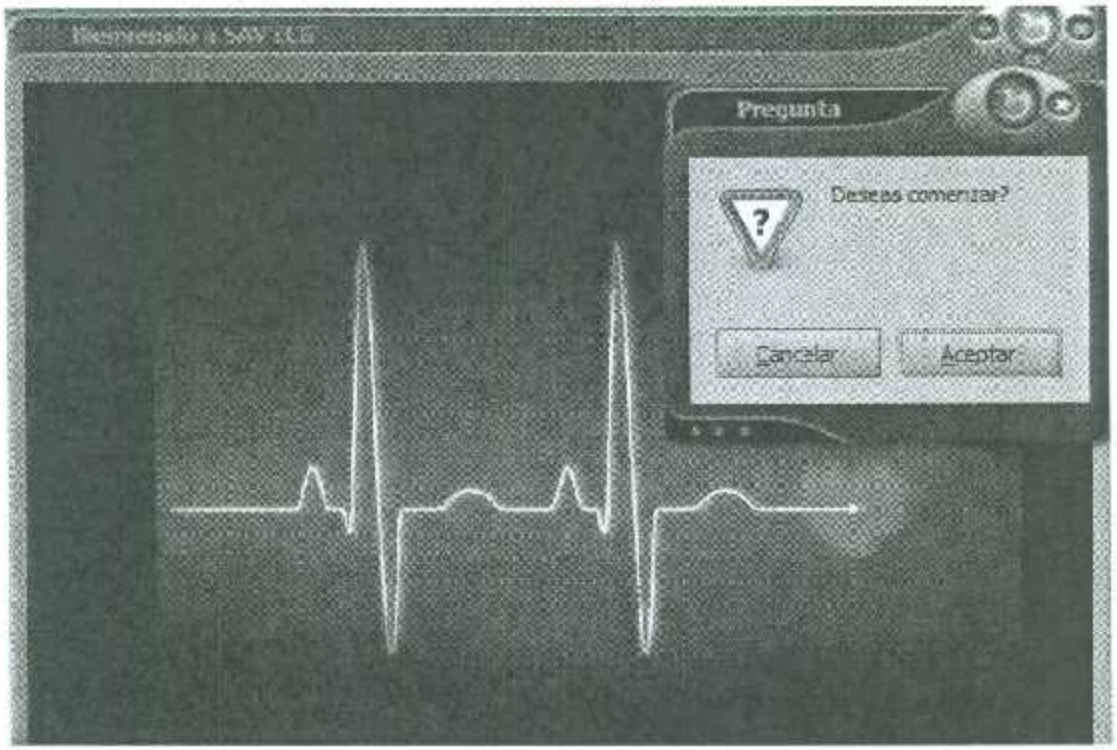


Figura 36. Pantalla de inicio para Interfaz Gráfica diseñada para graficación en tiempo real hecha en OCTAVE (segundo bloque).

Archivo *figurastart*

El archivo *"figurastart.m"* es el segundo bloque, este se encarga de mostrar la portada de presentación del programa principal ya descrita anteriormente (véase Figura 36). El programa es sumamente sencillo, en el se definen caracteristicas de la figura como el color de fondo, el título de la portada y la imagen a mostrar en este caso "ecg.jpg".

```
graphics_toolkit fltk

figure(1)

set (gcf, 'color', 'black')

set (gcf, 'name', 'Bienvenido a SAV ECG')

set (gcf, 'numbertitle', 'off')

#

imshow("ecg.jpg")
```

Archivo *gtr_zenity2*

Por último se presenta el tercer bloque y más importante, dentro de él se realiza la graficación en tiempo real de la señal analógica adquirida. Como se vio en el subtema 2.6, OCTAVE es el lenguaje más compatible con MATLAB esto debido a que su programación posee prácticamente la misma estructura y funciones (véase ANEXO B). Así pues, el núcleo del programa para la adquisición de datos es muy similar a lo que se hizo en MATLAB (véase subtema 3.1.1) solo teniendo presente algunas consideraciones. Lo primero a realizar es habilitar el puerto serie y añadir varias características como el puerto, el "Baudrate", el tamaño del byte, la paridad y el bit de parada. Es importante resaltar que cuando se trabaja en Linux, en este caso Cygwin (véase subtema 2.4.2) los puertos son nombrados "ttyS" y al número de puerto en Windows se le resta uno, por ejemplo, si se tuviera el dispositivo Xbee conectado al puerto "COM20" en Windows, para ser identificado en Linux seria "ttyS19".

Lo siguiente es la declaración de varios contadores que se van a utilizar (*c, i y t*), un vector de almacenamiento (*y*) de tamaño *rate*tmax* que como se recordara también fue utilizado en MATLAB, la creación del objeto línea, los limite del eje *X* y *Y*, el limite final del objeto línea *lf* y las características de la gráfica como la cuadricula, el color y titulo. Hecho lo anterior, se procede a la graficación.

Se comienza entonces inicializando el cronometro *tic toc* (ya visto en MATLAB) para que cada iteración se realice cada segundo, el siguiente paso es iniciar la sentencia while especificando que el proceso durará hasta que el cronometro sea mayor al límite de tiempo establecido como *tmax* (en este caso 10 segundos). A continuación, se empieza con la lectura del puerto mediante la instrucción *char(srl_read(s,100))*, después, el valor obtenido es transformado de una cadena de caracteres a una cadena de números enteros para poder ser manipulada fácilmente y se elige una localidad del vector (*n(5)*). Enseguida, se establecen nuevamente las características de la gráfica debido a que en OCTAVE en cada vuelta del ciclo cuando se habla de graficación, es necesario refrescar todos los objetos por motivos de evitar la saturación del buffer y perdida de datos.

Ahora, se realiza la misma técnica que se utilizo en la GUI de MATLAB (véase subtema 3.1.1), se almacena en el vector "*y*" cada valor que se ha obtenido por el puerto y se efectúa la conversión de su escala que va desde 0 a 1024 a un rango que vaya desde cero a cinco y con ello pueda ser interpretado cada valor en términos de voltaje para su mejor análisis.

Posteriormente, se definen los límites del eje *X* y *Y* y mediante la instrucción *linspace* establecemos que nuestro objeto línea declarado anteriormente, comprenderá un espacio de cero a la variable límite final (*lf*) con un número de puntos (*i*) dentro del espacio del objeto línea, esto para dar un efecto dinámico durante el tiempo de graficación. Realizado lo anterior, se establece el valor del objeto línea en el eje *X* y *Y* durante cada espacio de tiempo con la instrucción *set(l(d),'YDATA',y(1:c),'XDATA',x)* y se grafica todo lo anterior por medio del comando *drawnow*. El siguiente paso es incrementar los contadores (*c* y *i*) y

modificar el límite final del objeto línea, esto se logra incrementando *lf* más la variable establecida como *rate*, la cual como se podrá recordar, será un número elegido experimentalmente y será la proporción en la que el objeto línea se desplaza a través del eje *X* para cumplir con el tiempo de barrido establecido por la variable *tmax*. Es importante tener en cuenta nuevamente que las dimensiones tanto del eje *X* como del *Y* deben ser exactamente del mismo tamaño para la primera iteración y así evitar errores de compilación, en el siguiente código se muestra lo anterior.

```
#=======================ABRIMOS_PUERTO_SERIAL==============

#puerto ttyS19

s=serial("/dev/ttyS19",9600);

srl_baudrate(s, 9600);

srl_bytesize(s, 8);

srl_parity(s, "N");

srl_stopbits(s, 1);

#=======================VARIABLES_CONTADORES================

c=1;

i=1;

d=1;

tmax = 10; # tiempo de captura 10

rate = 0.137; # proporcion (resultado experimental)

#=================VECTOR_DE_ALMACENAMIENTO=================

y =zeros(1,rate*tmax);

#=======================CREACIÓN_OBJETO_LINEA===============

l(d) = line('Color','g','LineWidth',2);

#

#Limite final del objeto linea

lf=0;
```

```
#=======Inicializando_figuras=============================================

clf(1)

close(1)

figure(2)

set (gcf, 'color', [0.5 0.5 0.5])#color gris!!

set (gcf, 'name', 'SAV ECG by: Salvador Garcia')

set (gcf, 'numbertitle', 'off')

t=0;

tic

while t<tmax

t=toc;

try

#-------Lectura_Puerto=========================================================

    m=char(srl_read(s,100)); #50-500

    n=str2num(m);

    n(5);  #n(2)

#=======Gráficacion=======================================================

    title({'Graficacion en tiempo Real con Octave, hecho por Salvador Garcia';'Lectura del Puerto';n(5);'Tiempo del proceso';t})

    xlabel('Tiempo (s)')

    ylabel('Voltaje (V)')

    axis on

    grid on

    hold on

    l(d) = line('Color','g','LineWidth',2);
```

```
y(1,c)=(n(5)*5)/1024;
x = linspace(0,lf,i);
ylim([0 6]);#[0 1024]
xlim([0 10])
set(gca,'XTick',0:1:10)
set(l(d),'YDATA',y(1:i),'XDATA',x)
drawnow
i = i+1;
c = c+1;
lf=lf+rate;
catch
;
end_try_catch
#
endwhile
```

Medición de la amplitud y la frecuencia:

En el momento en que el cronometro (*tic toc*) llegue a los diez segundos, se cerrara tanto la sentencia while como el objeto serial y se comenzará con la medición de la amplitud y la frecuencia. Para ello, es sumamente sencillo solo se requiere de la instrucción *ginput()* la cuál, en el momento en que se selecciona una coordenada con el cursor en el objeto figura (en este caso la grafica) y se ingresa la tecla ENTER, esta función proporciona la coordenadas seleccionadas en un vector de 1X2 donde la posición uno (1) del arreglo antes mencionada pertenece al eje *X* y la posición dos (2) al eje *Y*. Conociendo lo anterior, el programa desplegara varios mensajes, el primero de ellos será para la medición de la frecuencia, pidiendo la posición de los cursores *T1* y *T2* y una vez que se obtiene la información de los datos deseados, se realiza una resta entre el valor de estos cursores (T2-T1) y se saca el inverso del resultado anterior (*1/T2-T1*) para así obtener la frecuencia.

Por otra parte, para obtener la amplitud de la señal adquirida, se realiza el mismo procedimiento que se utilizó para la frecuencia, solo teniendo en cuenta que la información que se desea de los cursores V1 y V2 es la posición (2) del vector de la función *ginput()*

que pertenece a la coordenada del eje *Y*. Ya que se han obtenido los valores deseados de los cursores, solo es necesario restar el cursor V1 menos el cursor V2 (V1-V2) y con ello obtener la amplitud.

Por último, aparecerá un mensaje información con ayuda de zenity y preguntará si deseamos cerrar o reiniciar la aplicación. Si esta acción es verdadera, se cerrará y limpiará automáticamente toda la aplicación, por otro lado, si la acción es falsa, se abrirá nuevamente el archivo *prueba_zenity.m* para así volver al programa principal. Todo esto se puede observar en el siguiente código.

```
#==========MEDIDAS==========

 disp('Calculo de la Frecuencia:')

 disp('Posicione el cursor T1')

 t1=ginput()

 disp('Posicione el cursor T2')

 t2=ginput()

 f=1/(t2-t1);

 disp('La Frecuencia es:')

 f

#

 disp('Calculo de la Amplitud:')

 disp('Posicione el cursor V1')

 [a,v1]=ginput();

 v1

 disp('Posicione el cursor V2')

 [b,v2]=ginput();

 v2

 dv=v1-v2;

 disp('La Amplitud es:')
```

```
dv
#
x=zenity_message ('Deseas cerrar(cancelar) o reiniciar(aceptar) la aplicacion', 'question');
if x==1
cla
clc
close all
clear all
else
cla
clc
prueba_zenity
endif
```

Resultados de las pruebas del prototipo.

Para realizar las pruebas, se ingresa la misma señal sinusoidal utilizada en MATLAB previamente acoplada con los mismos valores tanto de amplitud como de frecuencia (1.2 Hz y 1 Vpp), Los resultados se muestra en la Figura 37.

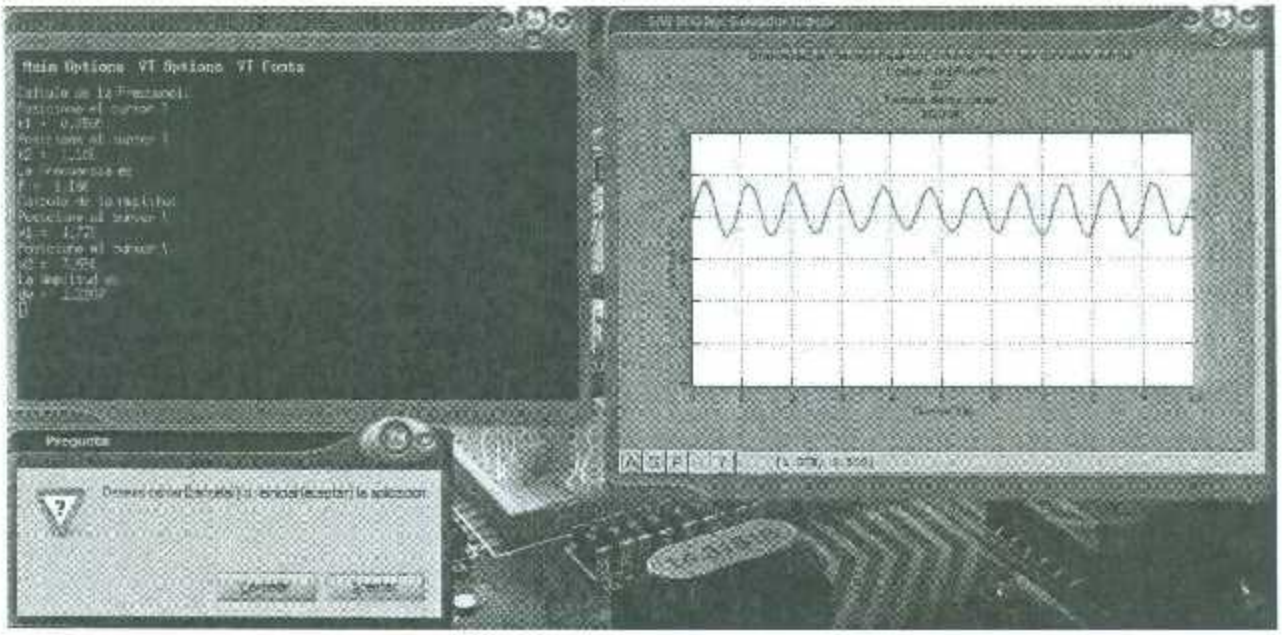


Figura 37. Resultados de la Interfaz Gráfica para graficación en tiempo real hecha en OCTAVE.

3.1.3 Graficación en tiempo real utilizando PYTHON

El último graficador que se desarrolló para el ECG fue utilizando PYTHON debido a que además de ser un programa bien diseñado, fácil de leer y escalable, está cubierto por la licencia BSD por lo que se permite su comercialización sin ningún tipo de restricción y sus paquetes son gratuitos. Tomando en cuenta los ANEXOS A y D y el subtema 2.5 el programa se explica a continuación.

Antes de comenzar, es importante señalar que aunque la programación en PYTHON es muy diferente a la de OCTAVE y MATLAB, si se sigue el algoritmo de graficación que se formuló para los dos programas anteriores, el código poseerá prácticamente la misma estructura. Nuevamente para propósito de aprendizaje, el programa se ha dividido en tres bloques principales.

En el primer bloque, se procede a establecer las librerías necesarias para poder trabajar, entre ellas pyplot para graficación, la librería serial que, como su nombre lo indica, es utilizada para la comunicación RS232, pylab y numpy para graficación y operaciones matemáticas y por último y de las más importantes, Tkinter para el desarrollo de la interfaz gráfica. A continuación se muestra el código.

from matplotlib import pyplot as plt

import serial

import pylab

import numpy as np

import time

*from Tkinter import **

import Tkinter as Tk

En el segundo bloque se definen varias funciones importantes, esto debido a que cuando se diseña una interfaz gráfica en PYTHON con Tkinter (véase Anexo D), se necesita primero visualizar lo que se desea hacer en la práctica, es decir, para este caso se necesita tres botones de control, uno llamado "start" para comenzar la graficación, uno llamado "stop" para detener el proceso y otro llamado "quit" para salir de la interfaz, además de dos ventanas de información para conocer los valores obtenidos por el puerto, la amplitud y la frecuencia y un objeto figura para observar la graficación, de esta manera el esquema de la GUI quedaria como se muestra en la Figura 38.

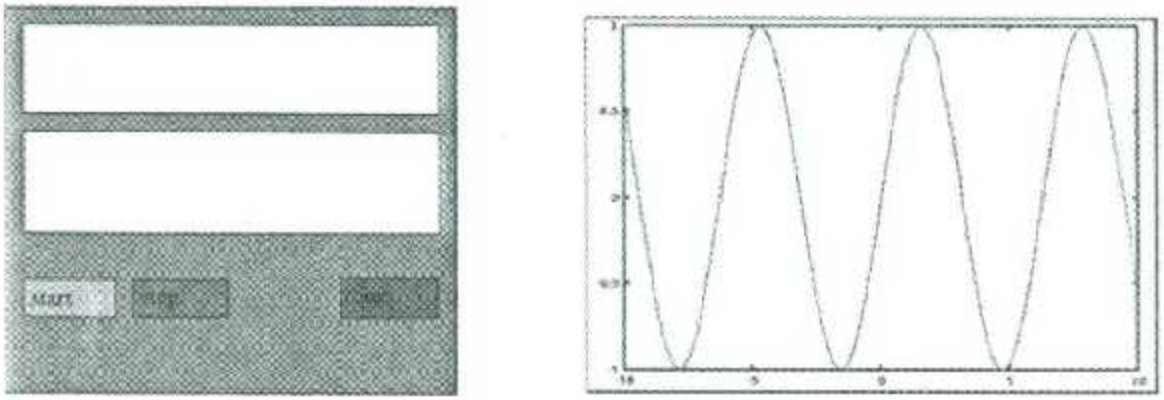


Figura 38. Esquema de la GUI a realizar en PYTHON.

Conociendo lo anterior, ya se puede definir las funciones del programa.

Función *root* (Tkinter).

La primera función a establecer será conocida como "*root*", en ella, se establecen las propiedades y características que se desean en la interfaz gráfica, dicho de otra manera, será el "esqueleto" ó "cascarón" de la GUI. Se comienza entonces con el nombre y el tamaño de la interfaz, utilizando los comandos *root.title* y *root.minsize*. Después, se declara una variable booleana que se llamará "*active_stat*" que se utilizará posteriormente, se define una etiqueta con cierto título, dos cajas de texto que servirá para mostrar la información del

puerto serie, la amplitud y la frecuencia y por último y más importante, las características de los tres botones de control, entre esas propiedades se deberá establecer, texto, fondo, posición y el comando o función a ejecutar cuando se presione cada uno de ellos, es necesario resaltar que para el botón *"quit"* solo se necesita agregar en las propiedades de comando de este mismo la instrucción *root.quit* para que realice dicha acción. El programa se muestra a continuación.

```
root = Tk.Tk()

root.title("RTP with Python by Ing.Salvador Garcia")

root.minsize(300,300)

root.geometry("500x650")#ANCHOXALTO

#

active_stat = Tk.BooleanVar(root)

active_stat.set(True)

#

w = Label(root, text="Panel de control para graficacion en tiempo real con Python\nhecho por: Salvador Garcia\n\nLectura del Puerto:")

w.pack(side=Tk.TOP)

#

textbox=Tk.Text(root,height=15,width=50)#textbox=Tk.Text(root)

textbox.pack(side=Tk.TOP)

#

w2 = Label(root, text="Mediciones (Frecuencia y Amplitud):\n")

w2.pack(side=Tk.TOP)

#

textbox2=Tk.Text(root,height=15,width=50)#textbox=Tk.Text(root)

textbox2.pack(side=Tk.TOP)

#
```

Tk.Button(root,text='Start',activebackground='green',command=start).pack(side=Tk.LEFT
)

Tk.Button(root, text='Stop',activebackground='red', command=stop).pack(side=Tk.LEFT)

Tk.Button(root,text='Quit',activebackground='blue',command=root.quit).pack(side=Tk.RI
GHT)

root.mainloop()

Función *start*

Esta será la encargada de realizar el algoritmo de graficación, el procedimiento es el siguiente. Primero que nada, se inicializará el puerto serie por medio de la instrucción *ser = serial.Serial('/dev/ttyS19', 9600)* estableciendo únicamente el "Baudrate" dado que las demás características como el bit de parada, la paridad, entre otros ya son asignados por "default", enseguida se activa la animación para graficación utilizando el comando *pylab.ion()*, se declara una variable llamada *dt* y varios contadores que serán utilizados (*i* ,*t* y *c*), se define un vector de tamaño *rate* (que será explicado posteriormente), además del rango de espacio para el objeto línea en el eje *X* y el número de incrementos en este mismo utilizando la ya conocida instrucción *linspace*, se crea un objeto figura con diversas características como título, etiquetas de los ejes y cuadriculado, por último, se define e inicializa el objeto línea.

La siguiente parte de la función comprende la graficación como tal, esta se encuentra como en todos los programas anteriores dentro de una sentencia while; antes de comenzar, es necesario explicar algunas cosas importantes. Debido a que PYTHON no posee una función como el cronometro *tic toc* (utilizada en MATLAB y OCTAVE) que nos permita que cada vuelta del ciclo se realice cada segundo, es necesario utilizar otra estrategia. Lo primero es conocer cuánto tiempo (en segundos) le toma a PYTHON realizar cada iteración, para ello, se dispone de una función llamada time (de la librería time) que nos permite saber el tiempo de duración de cada proceso. Ya que se tiene la información del tiempo gracias a la instrucción anterior, es posible ajustar el número de iteraciones en el ciclo while con ayuda de una variable (en este caso *tmax*) para que el proceso se realice durante el tiempo deseado que para este caso se requiere de ochenta iteraciones que equivalen a diez segundos.

Volviendo al tema, para inicializar el ciclo se pide al usuario una condición, en la que si la variable booleana *active_stat* (declarada anteriormente para la GUI), posee un valor verdadero (active_stat.get(True)), inicialice la graficación, de no ser así, cierre el objeto serial y figura, se rompa el ciclo y devuelva el valor por default de la variable *active_stat*. Dicho lo anterior, si se obtuvo un valor verdadero, se comienza con la lectura del puerto y se hace la conversión de una cadena de caracteres a una de números enteros para poder

manipular fácilmente el dato y con ello cambiar su escala de 0 a 1024 por un rango que vaya desde cero a cinco, debido a que como ya se habia mencionado se requiere de un rango que pueda ser interpretado en términos de voltaje, por último este valor es almacenado en el vector (*y*).

El siguiente paso es muy sencillo, para crear el efecto de dinamismo en la grafica se utiliza un método casi similar a lo que se ha estado realizando en todos los programas, solo que a diferencia de MATLAB y OCTAVE, en PYTHON, el número de iteraciones es proporcional al número total de puntos en un tiempo "n" elegido (en este caso diez segundos), por esta razón la variable *rate* será igual al valor que posee la variable *tmax* y la función *linspace* tendrá un rango que va desde cero a diez con un número total de ochenta puntos *(x = np.linspace(0.0,10.0, rate),* cumpliéndose todo el trazo en diez segundos.

Realizado lo anterior, utilizando el comando *set* (como ya se ha visto anteriormente) esta función asigna el valor que tendrá el objeto línea en el eje *X* durante un espacio de tiempo determinado utilizando el valor de la localidad actual en el vector de almacenamiento (*y*), después, será dibujado lo anterior en la grafica por la instrucción *draw* y se incrementaran los contadores (*i y t*). Por último, solo resta colocar la información del puerto serie en el cuadro de texto, agregando algunas propiedades como la cantidad de texto en el cuadro y el salto de línea, el código se muestra a continuación.

```
def start():

ser = serial.Serial('/dev/ttyS19', 9600)

pylab.ion() #activo animacion

dt = 1

t=0;

tmax=80;#10s->80iteraciones

rate=tmax;#el numero de iteraciones es proporcional al numero total de puntos en un
tiempo "n" elegido

x = np.linspace(0.0,10.0, rate)#190(0-20)

y = np.zeros(rate)

i=0;

c=0;

fig = plt.figure()
```

```
fig.canvas.set_window_title("RTP with Python by Ing.Salvador Garcia")
plt.suptitle('Graficador en tiempo real hecho por Salvador Garcia')
plt.ylabel('Voltaje')
plt.xlabel('S')
plt.ylim((0, 6))
plt.xlim((0, 10))
plt.grid(True)
line,=plt.plot(x,y,lw=2,c= 'r',ls='-')#line,=plt.plot(x,y,lw=2,c= 'r',ls='-',marker='.')
#
t0 = time.time()
while (t<tmax):
        if active_stat.get():
                m=ser.read(100)
                a=list(m) #convertimos a lista
#
                b=m.split() #convertimos de lista str a lista chr
#
                d=map(int,b[:]) #convertimos de lista chr a list int
#
#               print d[15] #dato limpio y listo para usarse
#
                y[i]= d[15]*5/float(1024)
#               print y[i]
                #timesub.set_xlim((0,i))#((i-20,i-5)),((i-200,i-1))i-200,i-1))
                line.set_ydata(y)
```

```
            plt.draw()
            i=i+dt;
            t=t+1;
            #time.sleep(1);
            textbox.insert(Tk.INSERT,"%d\n" % (d[15]))#textbox.insert(Tk.END,
"%d\n" % (i))
            textbox.delete(30.0,Tk.END)#textbox.delete(2.0,Tk.END)
            textbox.mark_set(Tk.INSERT, 1.0)
        else:
            active_stat.set(True)
            plt.close(fig)
            ser.close()
            break
root.update()
```

Medición de la amplitud y la frecuencia:

En el momento en que la sentencia while llegue a los diez segundos, se detendrá el proceso y se informará en cuanto tiempo se llevo a cabo, enseguida en la segunda ventana de información se desplegará un mensaje para efectuar las mediciones de frecuencia y amplitud a la señal adquirida. Para realizar lo anterior es muy sencillo, utilizando la instrucción *ginput()* (previamente implementada en OCTAVE), en el momento en que se selecciona una coordenada con el cursor en el objeto figura (en este caso la grafica) proporciona la coordenadas seleccionadas en un vector de 1X2 donde la posición uno (1) del arreglo antes mencionada pertenece al eje *X* y la posición dos (2) al eje *Y*, solo que a diferencia de OCTAVE, en PYTHON se requiere de un número determinado de "*clicks*" en el mouse (en este caso se optó por dos clicks), en lugar de la tecla ENTER.

Es por esta razón, que el procedimiento para obtener la frecuencia y la amplitud es muy similar a lo que se realizo en MATLAB y OCTAVE, se obtiene los datos de los cursores, se realizan las operaciones ya echas anteriormente (*T2-T1, 1/T2-T1* y *V2-V1*) y se despliega la información en la caja de texto número dos, el código explicado se muestra a continuación.

```
ser.close();
```

```
print time.time() - t0, "segundos que llevo el proceso"

textbox2.insert(Tk.INSERT,"Calculo de la Frecuencia\nColoque cursor
T1:\n")#textbox.insert(Tk.INSERT,"%d\n" % (d[15]))

pts = ginput(2) # espera por dos clicks para confirmar

int1, int2 = pts

t1=int1[0]

textbox2.insert(Tk.INSERT,"%f Hz\n" % (t1))

textbox2.insert(Tk.INSERT,"Coloque cursor T2:\n")

pts = ginput(2) # espera por dos clicks para confirmar

int1, int2 = pts

t2=int1[0]

textbox2.insert(Tk.INSERT,"%f Hz\n" % (t2))

dt=t2-t1;

f=1/float(dt);

textbox2.insert(Tk.INSERT,"La Frecuencia es: %f Hz\n\n" % (f))

#

textbox2.insert(Tk.INSERT,"Calculo de la Amplitud\nColoque cursor V1:\n")

pts = ginput(2) # espera por dos clicks para confirmar

int1, int2 = pts

v1=int1[1]

textbox2.insert(Tk.INSERT,"%f V\n" % (v1))

textbox2.insert(Tk.INSERT,"Coloque cursor v2:\n")

pts = ginput(2) # espera por dos clicks para confirmar

int1, int2 = pts

v2=int1[1]
```

textbox2.insert(Tk.INSERT,"%f V\n" % (v2))

dv=float(v1-v2);

textbox2.insert(Tk.INSERT,"La Amplitud es: %f Vpp\n\n" % (dv))

#

Función *stop*

Finalmente se crea la función más sencilla, en ella solo es necesario cambiar el valor de la variable *active_stat* a una condición falsa para detener todo el proceso, enseguida se muestra el código.

def stop():

active_stat.set(False)

Resultados

A continuación, en la Figura 39 se muestra el programa ejecutándose utilizando el prototipo de adquisición de datos y el circuito de acoplamiento diseñados, ingresando la misma señal sinusoidal utilizada en MATLAB y OCTAVE con los mismos valores tanto de amplitud como de frecuencia (1.2 Hz y 1 Vpp).

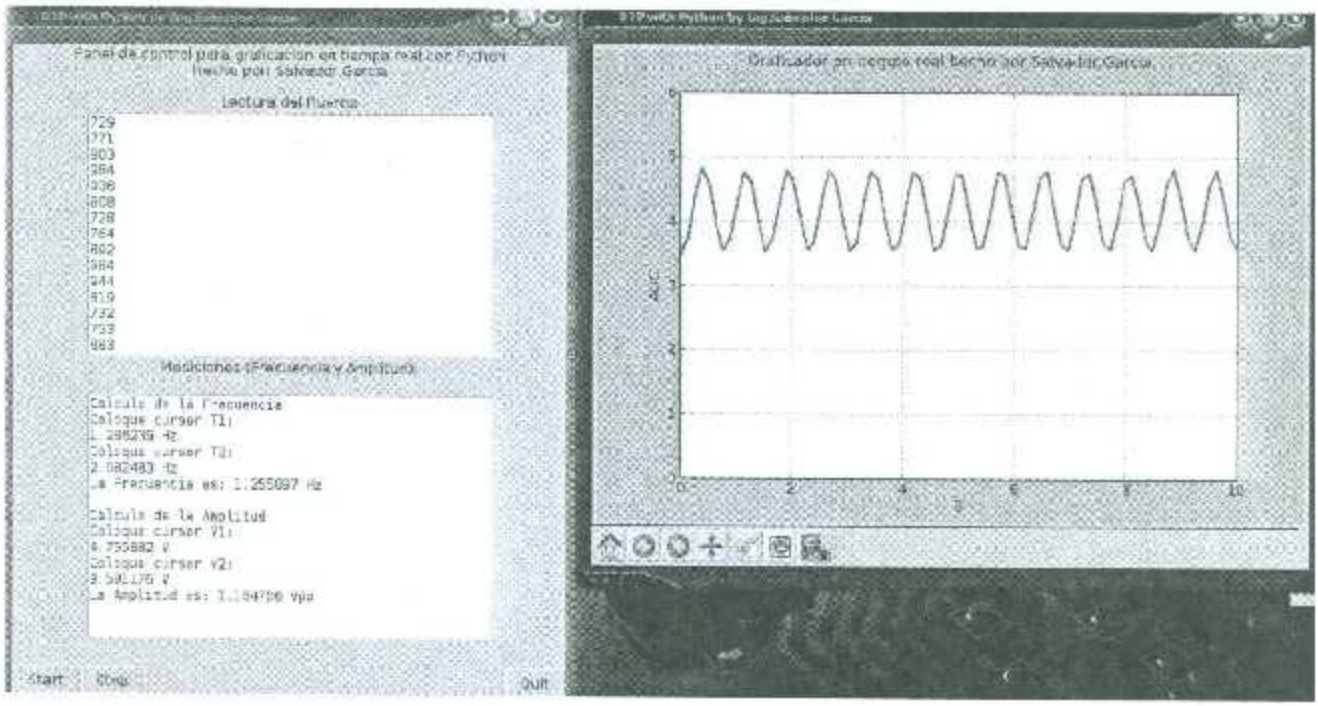


Figura 39. Resultados de la Interfaz Gráfica para graficación en tiempo real hecha en PYTHON.

3.1.4 Diseño y desarrollo del electrocardiógrafo

La siguiente parte del proyecto de investigación corresponde al diseño del electrocardiógrafo (ECG) para ser visualizado en los graficadores hechos en MATLAB, OCTAVE y PYTHON y con ello analizar la señal cardiaca. Teniendo en cuenta el subtema 2.1 se elaboran los siguientes circuitos.

Circuito de protección del paciente y del equipo

La Figura 40 muestra el circuito de protección del paciente y del equipo. Como se podrá notar, la señal bioeléctrica, detectada por el electrodo, llega al equipo mediante un cable coaxial blindado.

La función de la resistencia R1X es esencialmente, actuar como dispositivo limitador de la corriente que fluye a través del cable de entrada. Asimismo, los diodos D1X y D2X, colocados en anti-paralelo, aseguran que cualquier señal de CD o CA, cuya amplitud exceda 600 mV, quedará acoplada a tierra.

Este arreglo protege la entrada de los circuitos amplificadores del equipo, contra descargas electrostáticas y de voltajes muy altos, que se pueden presentar durante un procedimiento de desfibrilación cardíaca.

Además de proteger los circuitos de entrada del equipo, el arreglo de la resistencia y los diodos protege al paciente contra corrientes de retorno que pueden provenir del amplificador o de la circuitería externa asociada (otros equipos).

La resistencia R1X en conjunto con el capacitor C1X (más las capacitancias de los diodos D1X y D2X), forman un filtro pasa bajas que previene contra oscilaciones transitorias y rechaza ruido de alta frecuencias.

Un detalle importantísimo que se incluye en este circuito radica en la inclusión de un amplificador de ganancia unitaria que alimenta una señal equivalente a la del biopotencial detectado, hacia el blindaje del cable coaxial del electrodo.

Esta conexión hace que el potencial del blindaje sea aproximadamente igual al del cable central y, en consecuencia, minimiza los efectos de ruido debidos a las corrientes de fuga existentes entre la capacitancia distribuida del cable interior y el blindaje. En este caso, las resistencias R2 Y R3 colocan el potencial del blindaje al 99% del potencial del cable interior y estabilizan al amplificador. Se requieren 3 circuitos idénticos al mostrado, para implementar el bloque de entrada de un electrocardiógrafo [2].

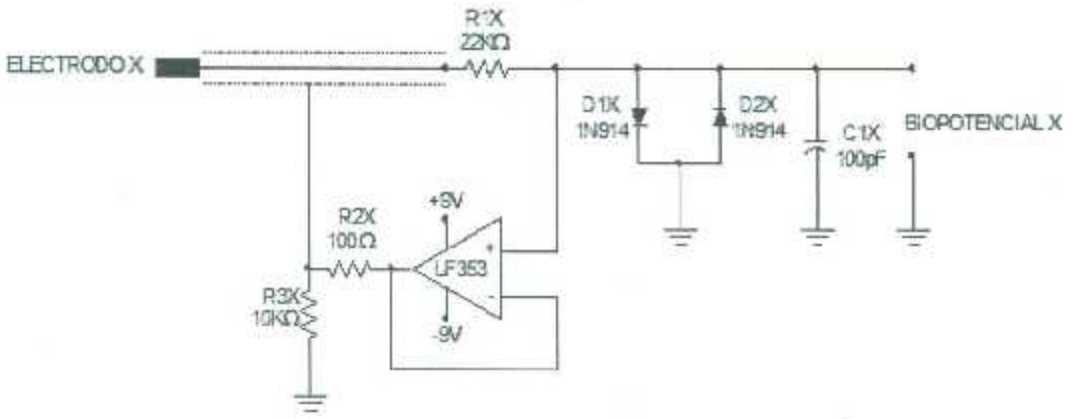


Figura 40. Circuito de protección del paciente y del equipo [2].

Circuitos de acoplamiento de impedancias para los potenciales Bipolares RA, LA, LL y manejo de pierna derecha, RL

Las señales de salida provenientes de los circuitos de protección del paciente y del equipo, se acoplan inmediatamente hacia la entrada de amplificadores de ganancia unitaria, a base de amplificadores operacionales LF353 (véase Figura 41).

El LF353 está fabricado con tecnología J-FET en sus entradas y es muy recomendado para aplicaciones biomédicas porque posee una ultra alta impedancia de entrada (en el orden de 10^{12} Ω), elevada razón de cambio (13V/μs) y una razón de rechazo en modo común típica de 100 dB.

La función de estos amplificadores en primera instancia, es presentar una elevada impedancia de entrada mucho mayor que la impedancia de la interfase electrodo-electrolito-piel. Esta condición garantizará que los efectos de carga sobre la fuente de biopotencial serán minimizados. Por otra parte, la baja impedancia de salida de los amplificadores ofrecerá la posibilidad de alimentar adecuadamente a la circuitería de generación de los potenciales de las diferentes derivaciones, así como el potencial de la terminal central de Wilson, TCW (véase Figura 42).

El circuito que se muestra en la Figura 41 en la parte inferior, se denomina "Circuito De Manejo De potencial De Pierna Derecha, RL". Este circuito tiene como misión inyectar, al cuerpo del paciente, una corriente cuya dinámica se oponga y cancele el efecto de las corrientes acopladas capacitivamente al cuerpo humano.

Recuerde que las señales de corriente alterna de 50/60 Hertz que son transportadas por las líneas de alimentación superponen señales de esa frecuencia al cuerpo. También los

balastros y las lámparas fluorescentes dan origen a interferencias cuyas frecuencias rondan 1KHz y sus amplitudes pueden opacar fácilmente a las del biopotencial.

La estrategia para eliminar estos ruidos de interferencia es alimentar al cuerpo del paciente con una versión similar a la de la señal interferente, desfasada 180°. Por esta razón, el circuito de manejo de pierna derecha toma una muestra del potencial de modo común, que se manifiesta en el terminal central de Wilson (véase Figura 42), y lo realimenta al cuerpo, con fase invertida [2].

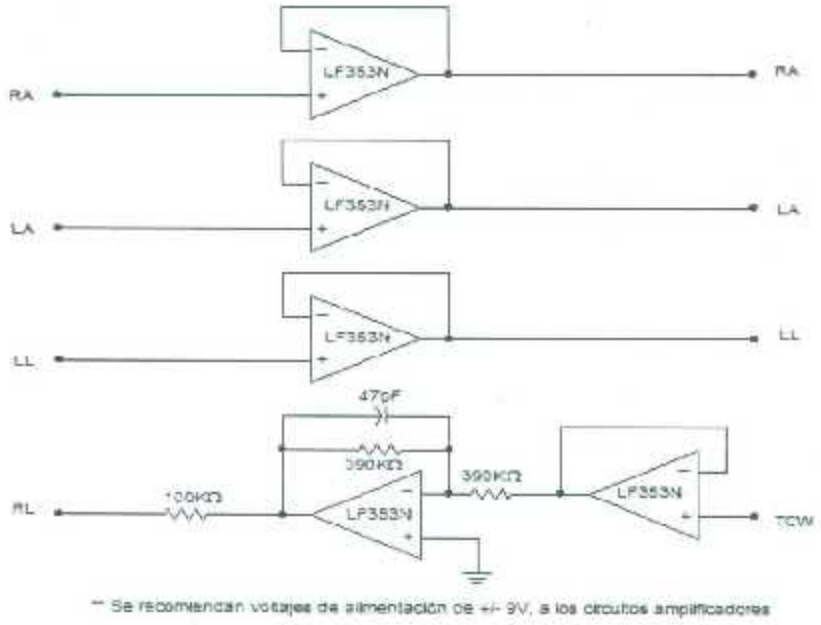


Figura 41. Circuitos de acoplamiento de impedancias para los potenciales Bipolares RA, LA, LL y manejo de pierna derecha, RI. [2].

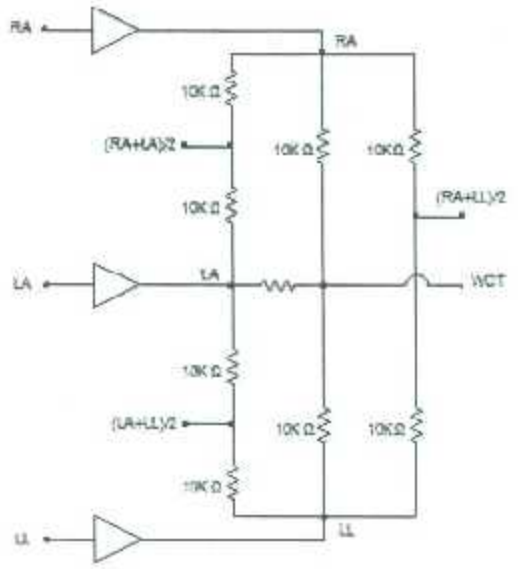


Figura 42. Terminal central de Wilson [2].

Circuito de derivaciones bipolares y preamplificación

Recordando el subtema 2.1.8, para obtener las derivaciones bipolares DI, DII y DIII, se requiere saber la diferencia de potencial entre los tres electrodos ubicados en Brazo Derecho (RA), Brazo Izquierdo (LA) y Pierna Izquierda (LL). Es por esta razón que se utiliza el amplificador de instrumentación AD620N, que además de ser un amplificador diferencial, posee características tales como alta impedancia de entrada, gran capacidad de rechazo a señales comunes y una muy alta inmunidad al ruido, haciéndolo perfecto para el diseño del electrocardiógrafo. El proceso de conexión puede observarse en la Figura 43 [2].

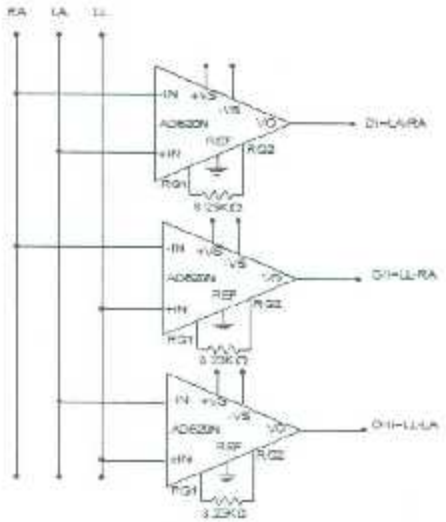


Figura 43. Circuito de derivaciones bipolares y preamplificación [2].

Circuitos de amplificación y selección de derivaciones

La siguiente etapa del ECG consiste en la selección de las derivaciones, esto se logra conectando las salidas obtenidas de cada amplificador de instrumentación a tres de las entradas de un multiplexor, además de una cuarta entrada conectada a un tren de pulsos de 7mVpp (utilizando un timer 555) para calibración de la señal ECG. Es importante recordar que en el MUX analógico las entradas que no sean utilizadas deben colocarse a tierra y se requiere de un DIP Switch para las líneas de selección y poder elegir la derivación deseada. Una vez elegida la derivación ó los pulsos de calibración, estos son amplificados y se manda hacia la última etapa (véase Figura 44) [2].

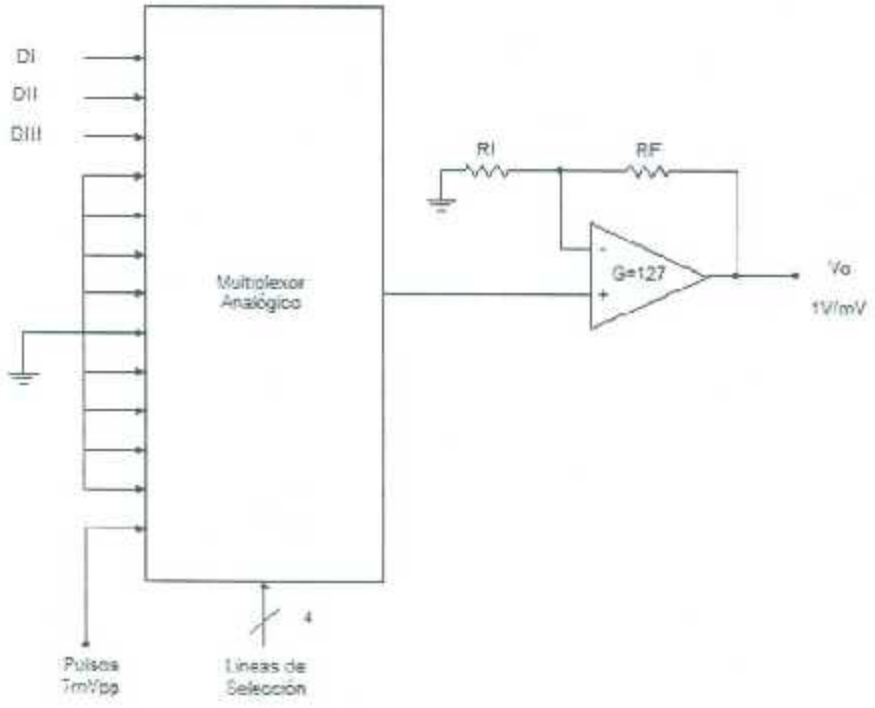


Figura 44. Circuitos de amplificación y selección de derivaciones [2].

Circuitos de filtraje

La etapa final es una de las más importantes en el diseño del ECG, debido a que en ella, se elimina todos los posibles ruidos e interferencias que puedan presentarse distorsionando y dificultando el análisis de la señal cardiaca.

Como se observo anteriormente, las señales de corriente alterna de 50/60 Hertz que son transportadas por las líneas de alimentación superponen señales de esa frecuencia al cuerpo. También los balastros y las lámparas fluorescentes dan origen a interferencias cuyas frecuencias rondan 1KHz y sus amplitudes pueden opacar fácilmente a las del biopotencial [2].

Teniendo en cuenta lo anterior y conociendo que la señal cardiaca promedio está entre los 40 lpm (0.6 Hz) y 100 lpm (1.6 Hz), se diseña un filtro pasa banda *Butterworth* de cuarto orden cuyas frecuencias de corte sean de .5Hz a 200Hz, además de dos filtros notch de 60Hz y 120Hz (véase ANEXO F).

Filtro pasa banda de .5Hz a 200Hz

Ya estudiado el subtema 2.9 y el ANEXO F, se puede observar que el filtro pasa banda está compuesto por un filtro pasa altas Butterworth de cuarto orden de 0.5Hz y un pasabajas Butterworth de 4 orden de 200Hz.

La Figura 45 muestra el diagrama del filtro pasa banda y la Figura 46 su respuesta a la frecuencia.

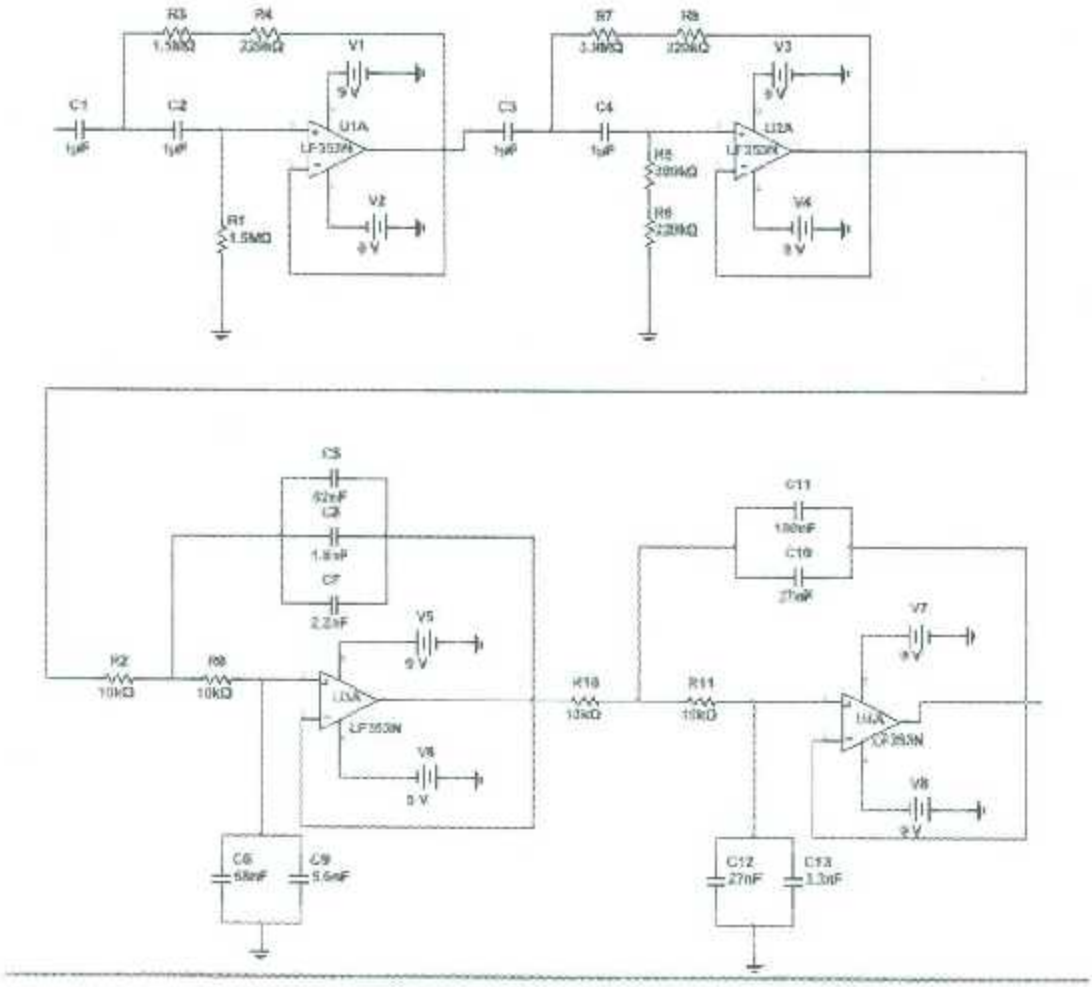


Figura 45. Circuito esquemático para el Filtro pasa banda de 0.5Hz a 200Hz.

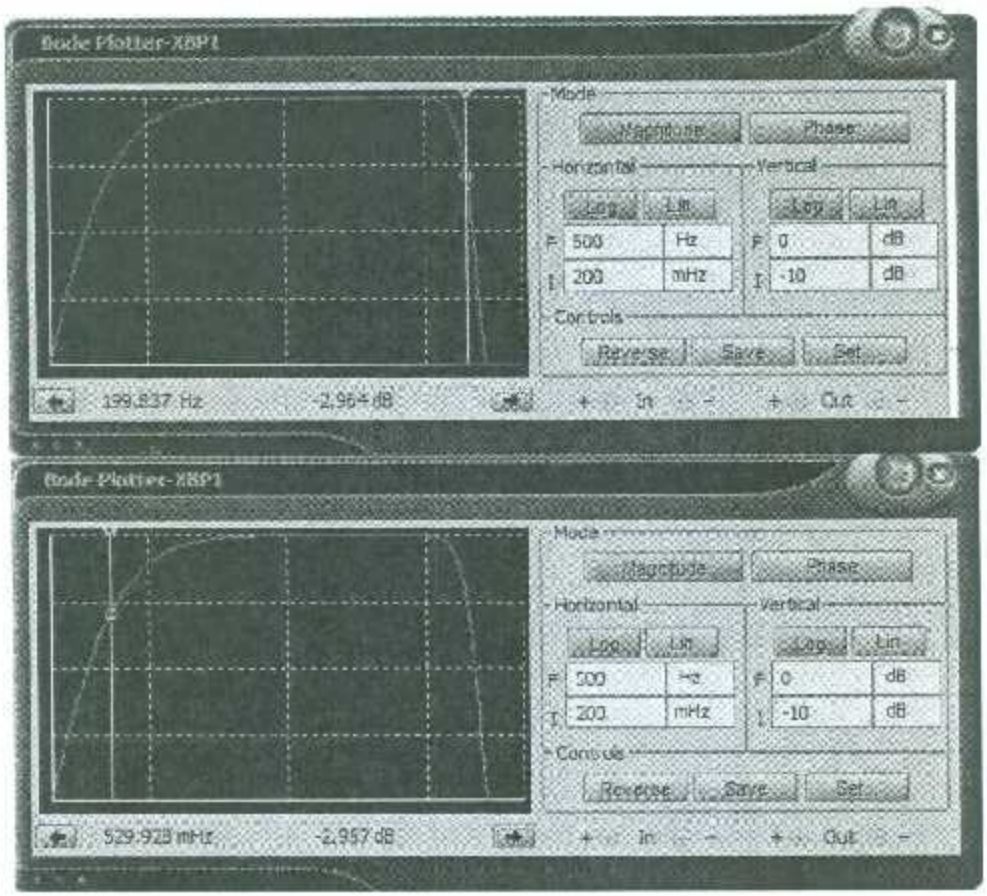


Figura 46. Respuesta a la frecuencia del filtro pasa banda.

Filtros notch de 60 Hz y 120Hz

En este caso, las frecuencias de 60Hz y 120Hz que son generadas por la línea de potencia hacen que el circuito del ECG se vea expuesto a ruido ambiental que proviene de las lámparas fluorescentes y otros dispositivos que emiten ruido a través de ondas de 60 Hz y 120 Hz. Los filtros notch se encargarán de rechazar exclusivamente estos ruidos para entregar a la salida una señal completamente pura de distorsiones.

Dicho lo anterior y recordando una vez más el subtema 2.9 y el ANEXO F, se diseñan los filtros. En las Figuras 47 y 48 se muestran los diagramas de los filtros y en las Figuras 49 y 50 su respuesta a la frecuencia.

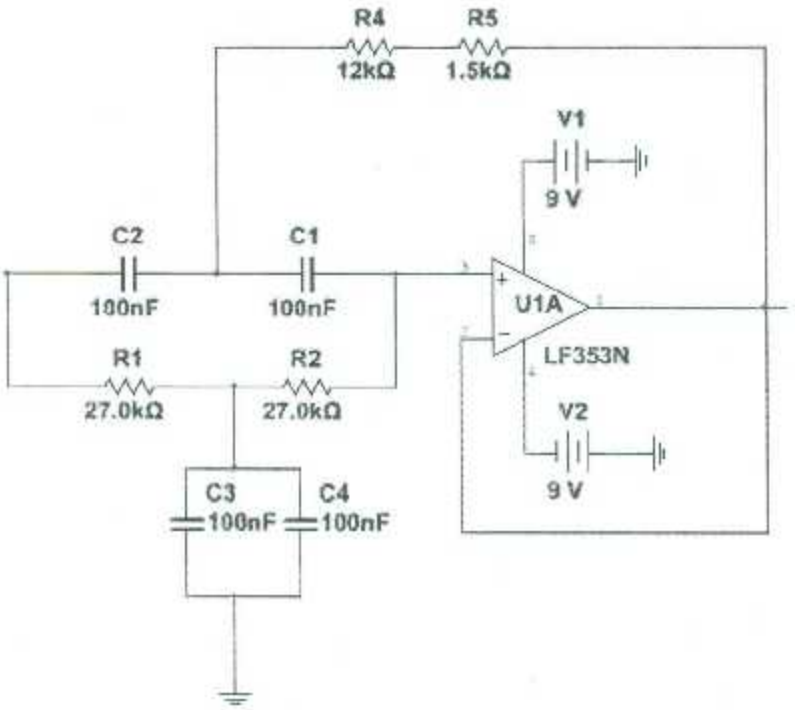


Figura 47. Circuito esquemático para el filtro notch de 60 Hz.

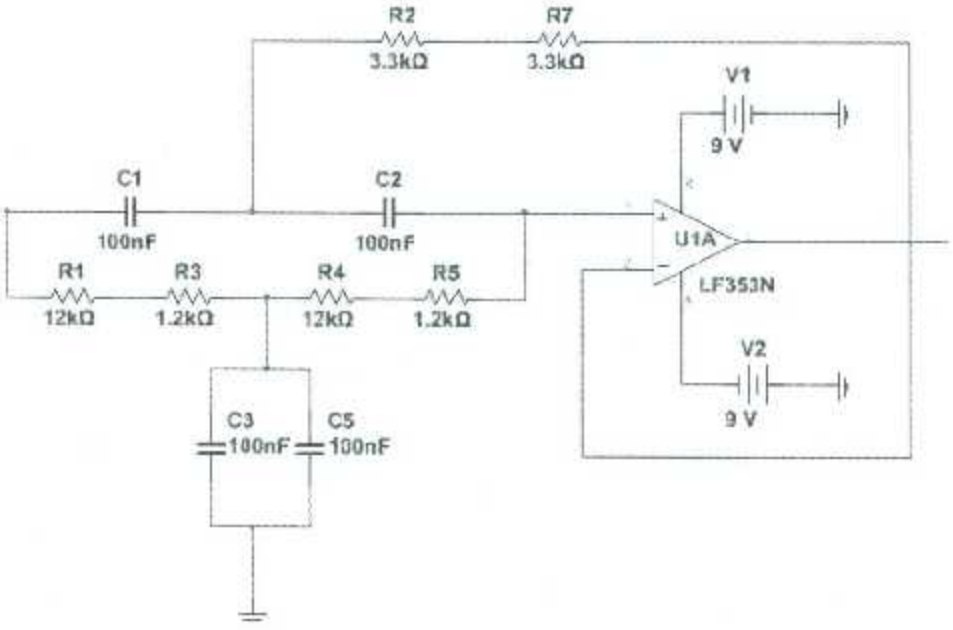


Figura 48. Circuito esquemático para el filtro notch de 120 Hz.

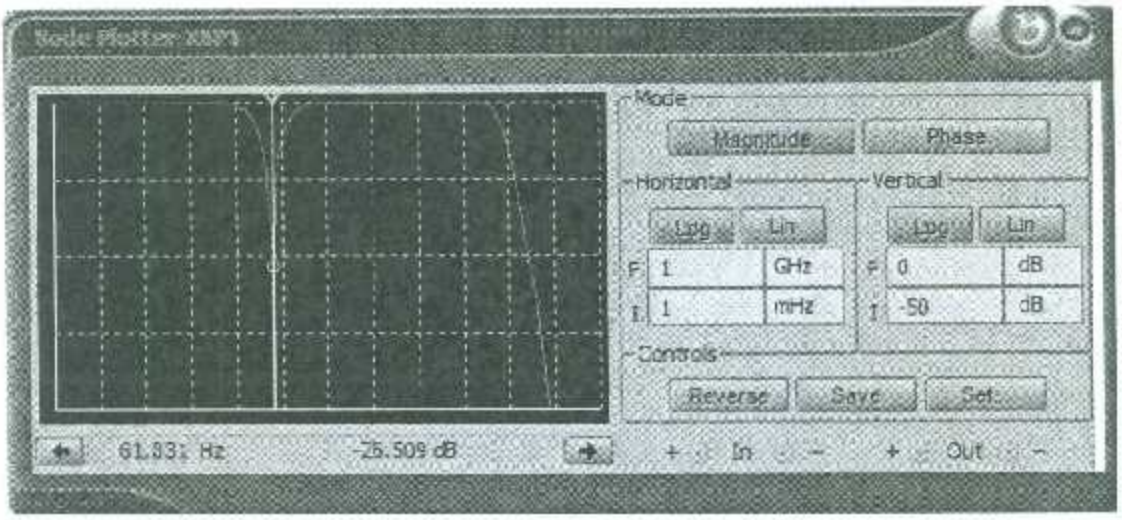


Figura 49. Respuesta a la frecuencia para el filtro notch de 60 Hz.

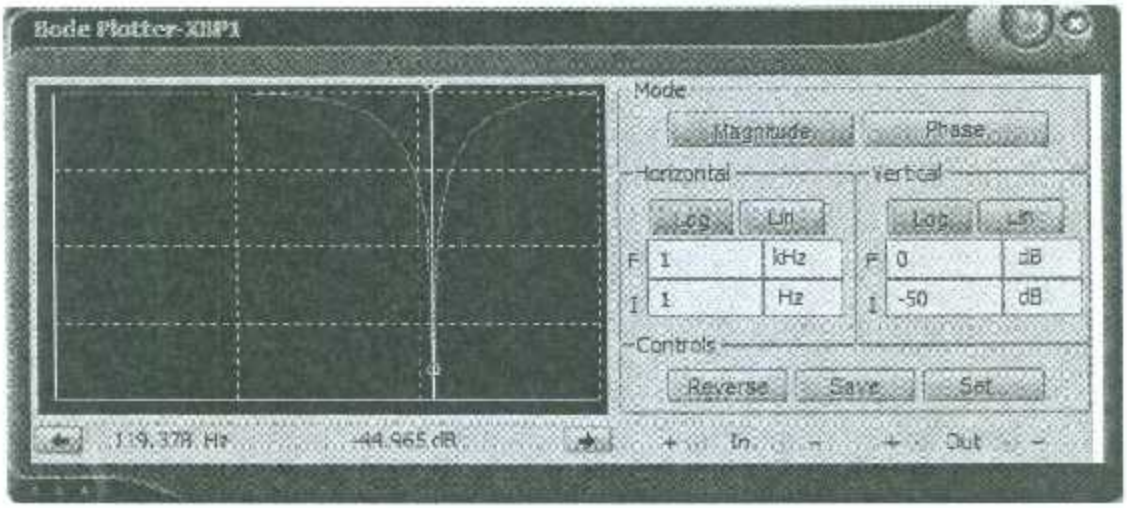


Figura 50. Respuesta a la frecuencia para el filtro notch de 120 Hz.

Prueba de la etapa de filtraje

Para verificar esta etapa, en el programa Multisim (National Instruments Corporation, Austin, Texas, Estados Unidos) se puede descargar y acceder a una base de datos de una señal ECG, la idea es mezclar señales de 60Hz, 120Hz y 600Hz con la señal cardiaca para simular el ruido al que se enfrentará el equipo y probar así el circuito de filtraje (véase Figura 51).

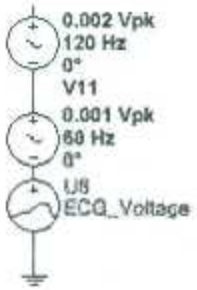


Figura 51. Mezcla de la señal ECG y ruido a diferentes frecuencias para la prueba.

En la Figura 52 se muestra el diagrama de la etapa de filtraje para las pruebas y en las Figuras 53, 54 y 55 los resultados.

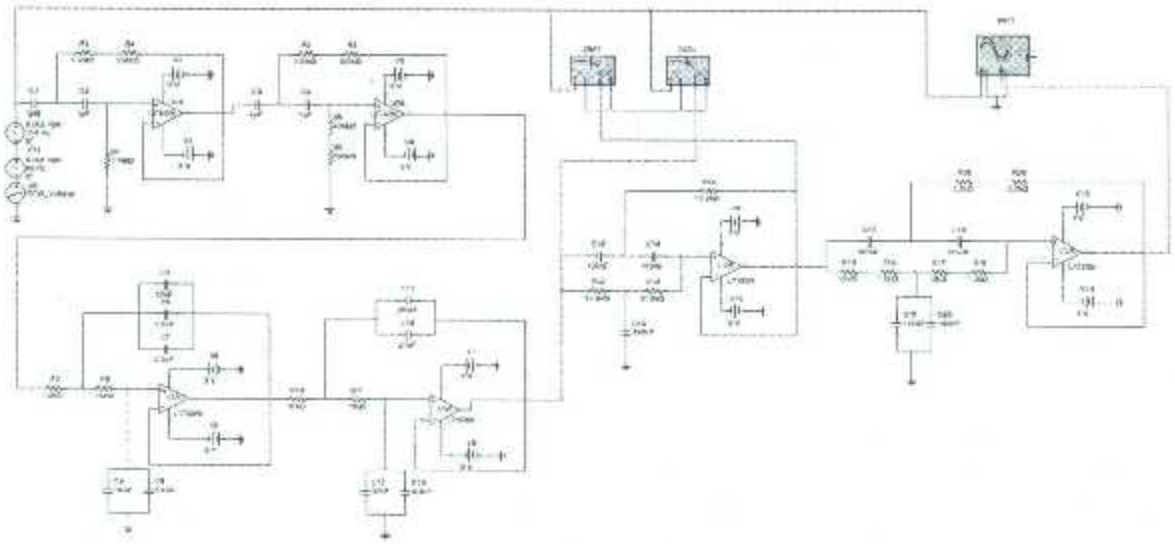


Figura 52. Circuito esquemático de la etapa de filtraje.

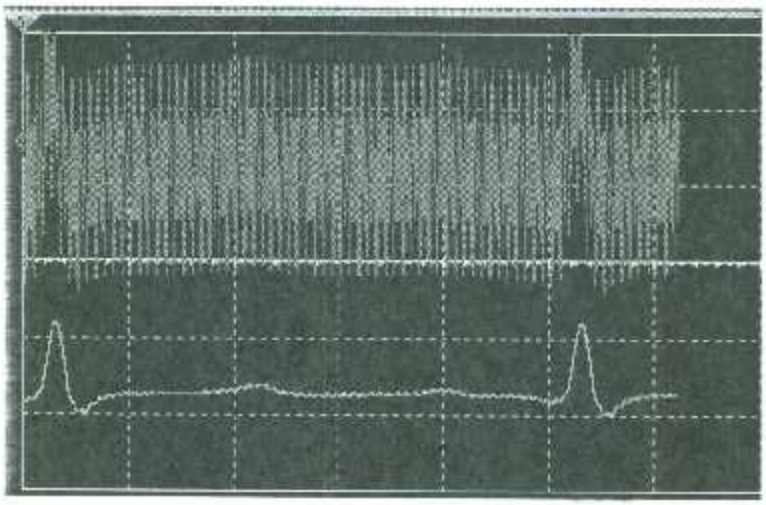


Figura 53. Mezcla de la señal ECG con ruido de 60 Hz y 120 Hz junto a salida filtrada vista en el osciloscopio.

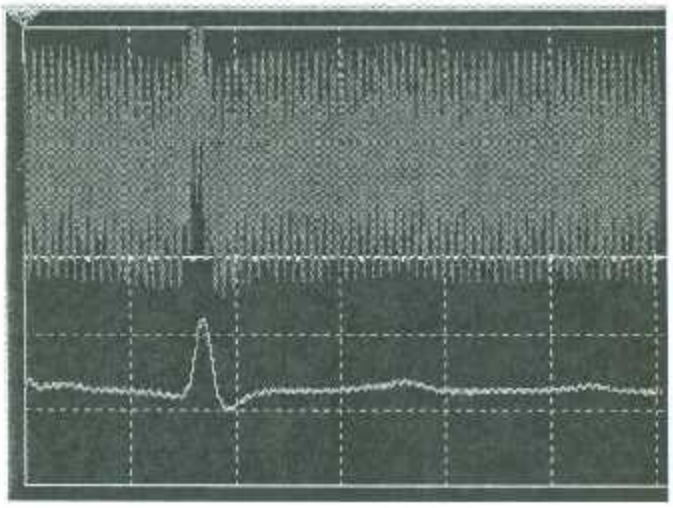


Figura 54. Mezcla de la señal ECG con ruido de 60 Hz y 600 Hz junto a salida filtrada vista en el osciloscopio.

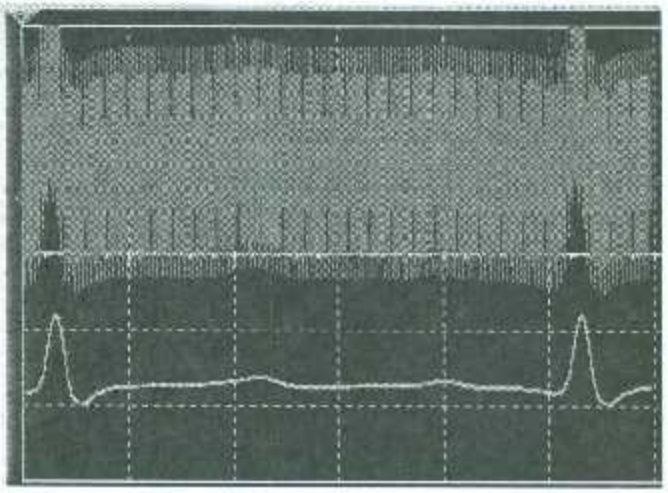


Figura 55. Mezcla de la señal ECG con ruido de 120 Hz y 600 Hz junto a salida filtrada vista en el osciloscopio.

Armado del electrocardiógrafo.

Diseñado el ECG, el diagrama debió a ver quedado como se muestra en la Figura 56.

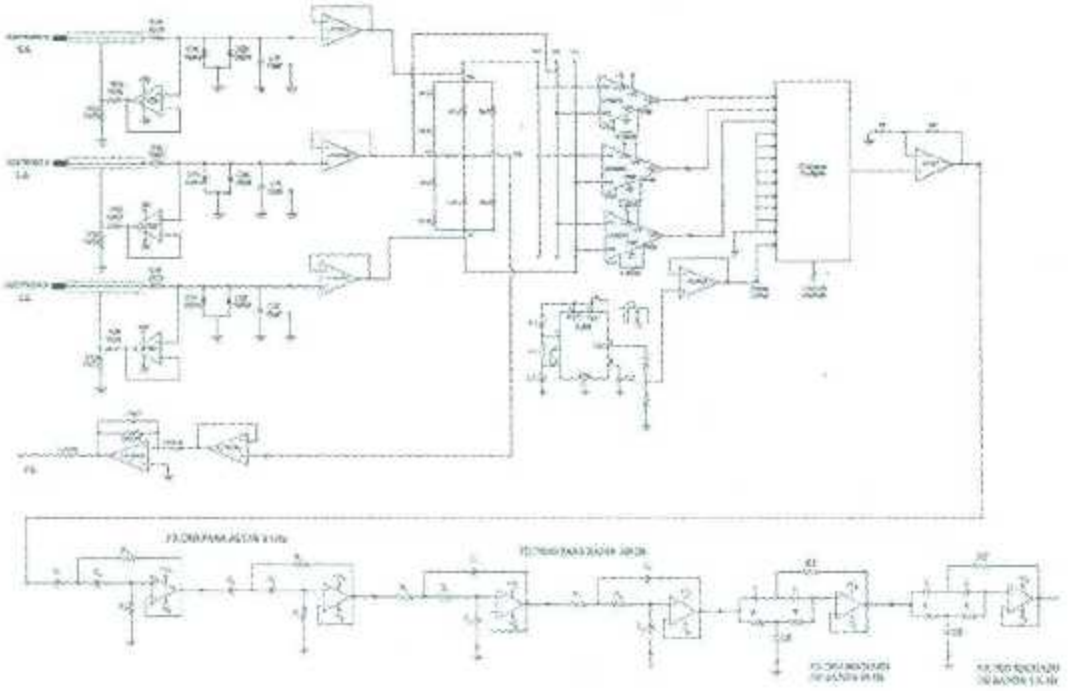


Figura 56. Circuito esquemático del ECG.

Lo siguiente es armar el prototipo, realizar el circuito impreso, elegir un sujeto de prueba X y visualizar en el osciloscopio los primeros resultados, en las Figuras 57, 58, 59, 60, 61 y 62 se muestran todos los procesos mencionados anteriormente.

Circuitos impresos del ECG

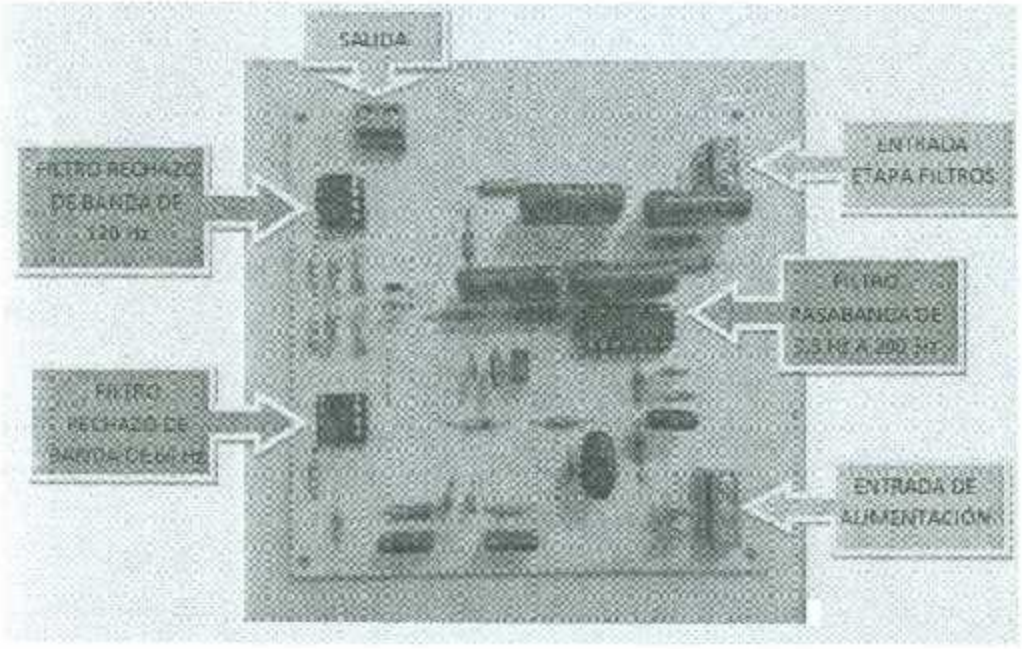


Figura 57. Circuito impreso de la etapa de filtraje.

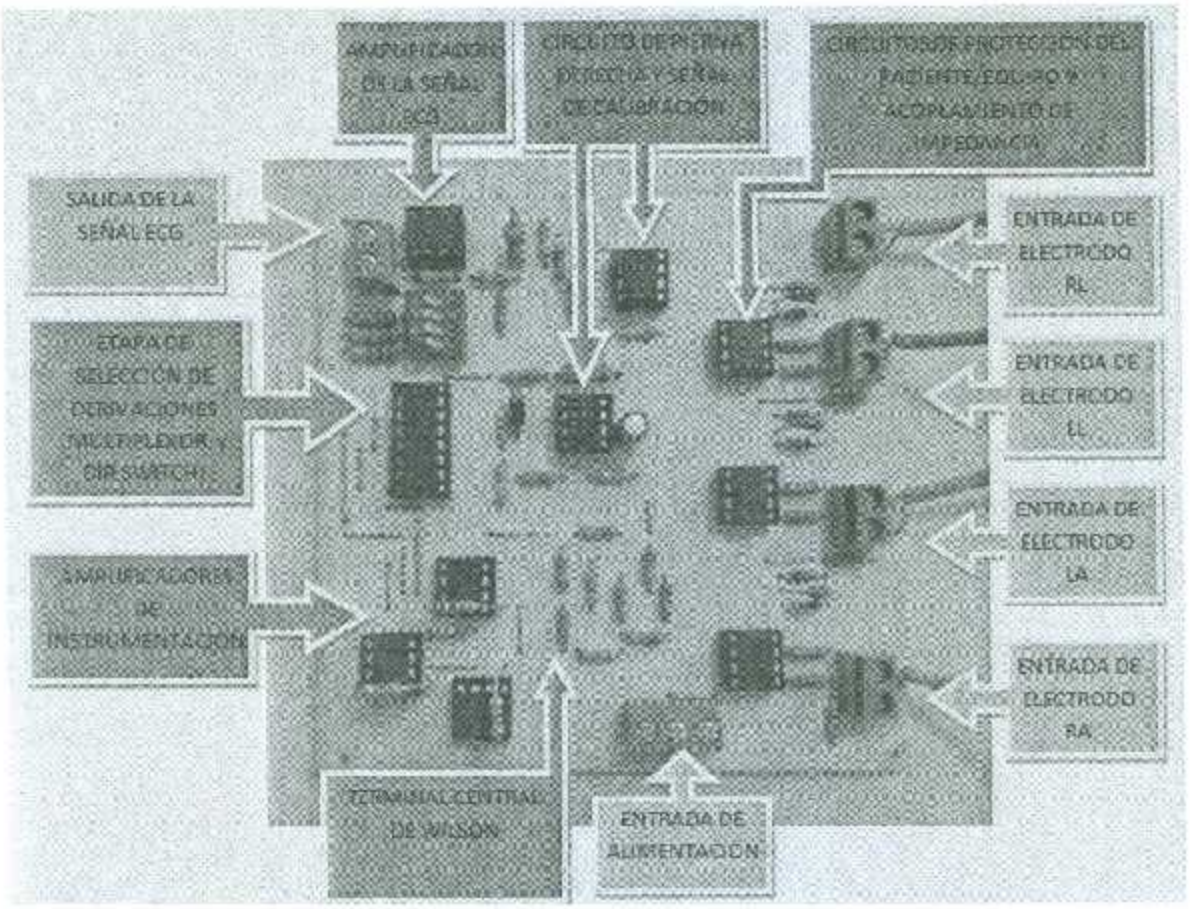


Figura 58. Circuito impreso de la etapa ECG.

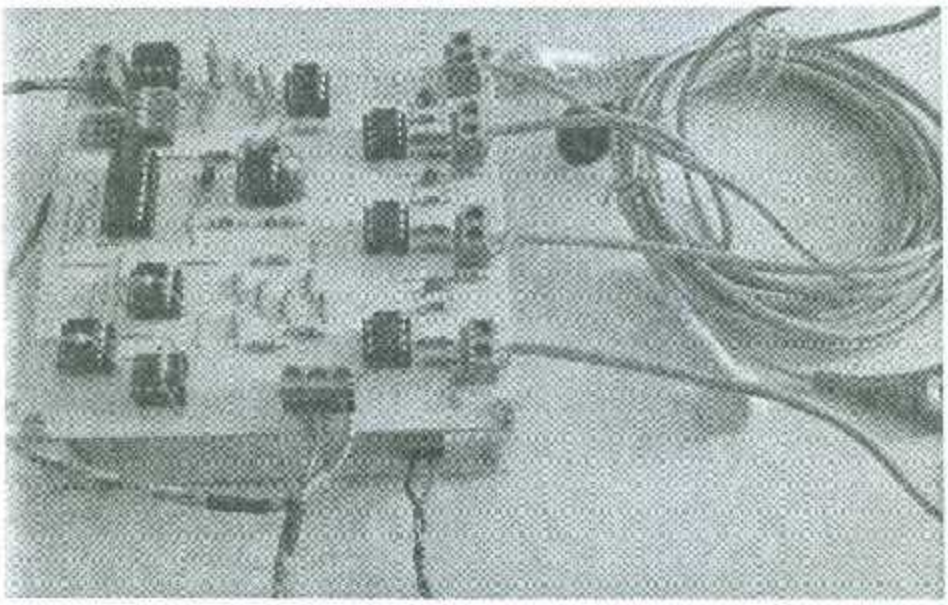


Figura 59. Etapas de filtraje y ECG montadas, listas para realizar pruebas.

Prototipo en funcionamiento (impresiones del osciloscopio)

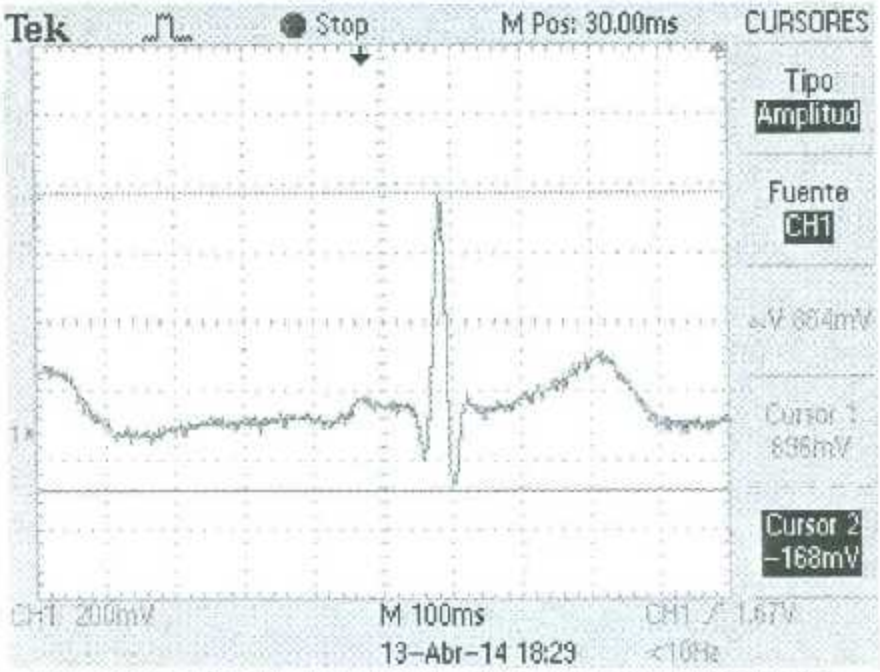


Figura 60. Impresión del osciloscopio de la derivación DI del sujeto de prueba X.

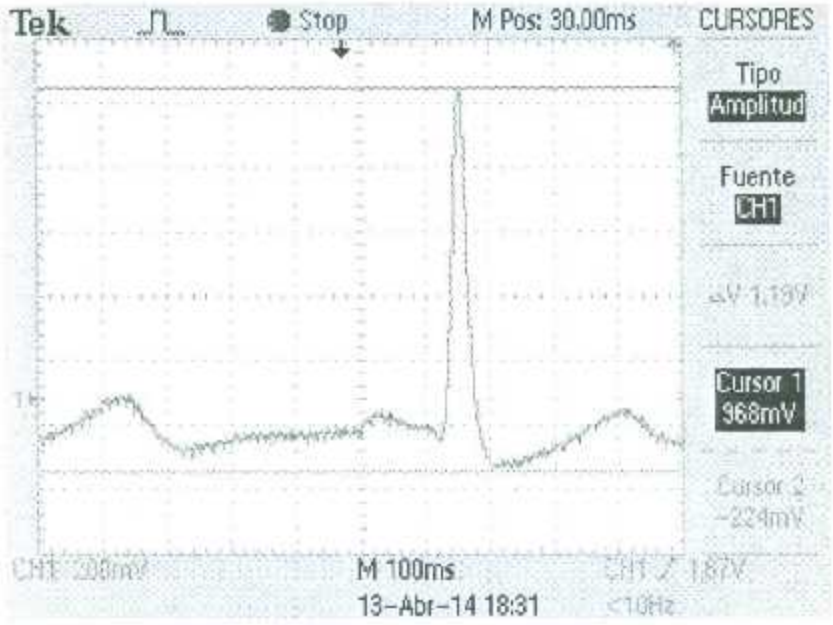


Figura 61. Impresión del osciloscopio de la derivación DII del sujeto de prueba X.

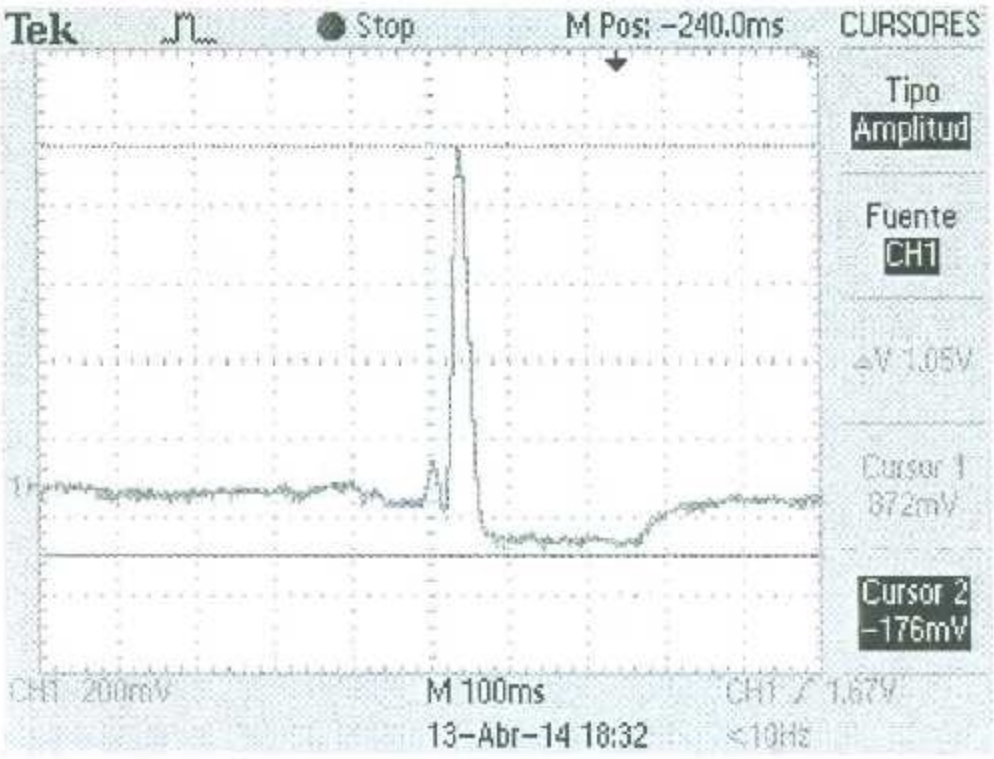


Figura 62. Impresión del osciloscopio de la derivación DIII del sujeto de prueba X.

Como se pudo apreciar, el prototipo funciona correctamente cumpliéndose así el objetivo. Cabe resaltar, que la calidad de los electrodos es de vital importancia para el buen funcionamiento del ECG, por esta razón, se requiere de electrodos que posean entre sus características una baja impedancia.

3.1.5 Implementación del ECG digital: graficación y análisis de la señal cardiaca

Acoplamiento de la señal cardiaca al sistema de adquisición de datos.

Para poder introducir la señal ECG al ADC del microcontrolador y con ello visualizar los datos en la computadora, es necesario diseñar un circuito de acoplamiento, esto es debido a que dicha señal presenta tanto picos de voltaje positivos como negativos que indiscutiblemente dañarían al integrado. Así pues, se parte de la siguiente gráfica de transferencia general para el circuito de acoplamiento (véase Figura 63).

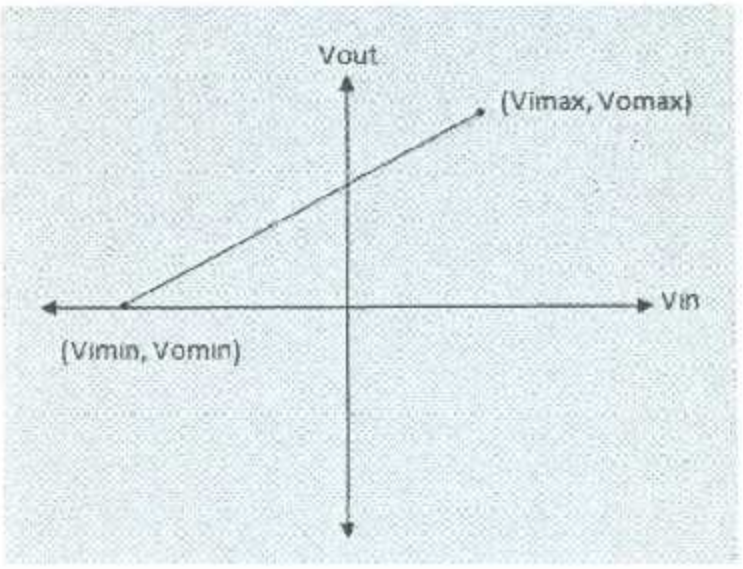


Figura 63. Gráfica de transferencia general para el circuito de acoplamiento.

La ecuación de la recta asociada a dicha línea es:

$$Vout - Vomin = \frac{Vomax - Vomin}{Vimax - Vimin}(Vin - Vimin) \quad (1)$$

Donde:

Vout es el voltaje de salida del circuito de acoplamiento

Vin es el voltaje de entrada del circuito de acoplamiento

Vomax es el voltaje máximo que puede salir del circuito de acoplamiento

Vomin es el voltaje mínimo que puede salir del circuito de acoplamiento

Vimax es el voltaje máximo que puede entrar al circuito de acoplamiento

Vimin es el voltaje mínimo que puede entrar al circuito de acoplamiento

Ya que se realizaron pruebas con el ECG y dado que el máximo valor de voltaje que se puede obtener de la señal cardiaca es de 1.19 Vpp de la derivación DII (véase subtema 3.1.4), se proponen entonces valores para sustituirlos en la ecuación (1) suponiendo que la señal de entrada Vin pudiera exceder el valor de 1.19 Vpp y tomando en cuenta que la señal de salida Vout debe de estar dentro del rango de cero a cinco volts dando como resultado un margen seguro para el buen funcionamiento del circuito de acoplamiento y evitar así daños al microcontrolador. A continuación se muestra el proceso.

Propuesta de valores de entrada y salida para el circuito de acoplamiento

$Vimin = -3$ $\quad Vomin = 0$

$Vimax = 3$ $\quad Vomax = 5$

Sustitución de valores en la ecuación (1)

$$Vout = \frac{5-0}{3-(-3)}[Vin-(-3)]+0$$

$$Vout = \frac{5}{6}Vin + \frac{15}{6}$$

$$Vout = 0.833Vin + 2.5 \qquad (2)$$

Circuito de acoplamiento

Conocido el resultado anterior, el siguiente paso es diseñar el circuito de acoplamiento. Como se puede apreciar en la ecuación (2), el circuito se puede implementar con amplificadores operaciones, el primero de ellos estará compuesto por un amplificador en configuración seguidor con un voltaje de entrada de 2.5v, el segundo amplificador se encontrará en configuración inversor con una ganancia variable que este dentro del rango de cero a uno logrando con ello variar la amplitud de la señal de salida y por último, un sumador para la adición de las dos señales anteriores. El diagrama esquemático se muestra a continuación.

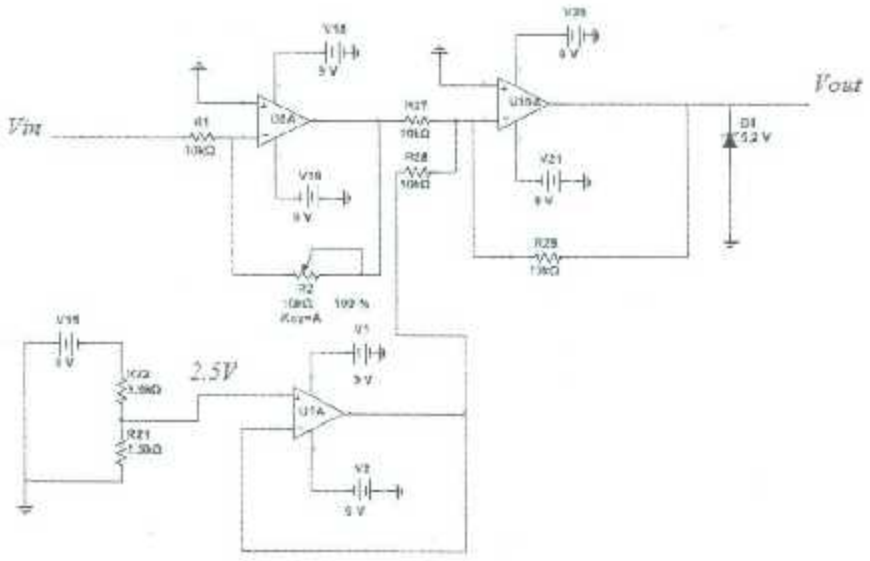


Figura 64. Diagrama esquemático del circuito de acoplamiento.

Como se puede observar en la Figura 64, la salida (*Vout*) del circuito de acoplamiento posee un diodo zener de 5.2V con propósitos de protección y con ello evitar cualquier daño al microcontrolador por sobre picos de voltaje.

Prueba del circuito de acoplamiento

Para probar el circuito, se ingresa una señal sinusoidal de amplitud de 1.19Vpp similar a la de la derivación DII y se ajusta el potenciómetro del amplificador inversor al 100% es decir, con una ganancia unitaria. Enseguida se muestran los resultados.

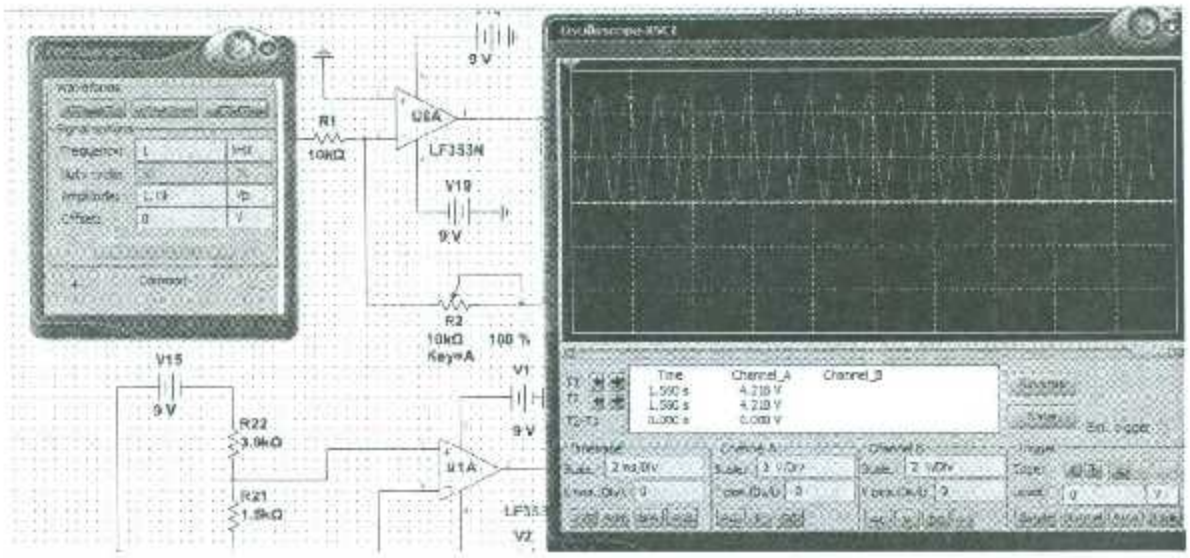


Figura 65. Prueba del circuito de acoplamiento.

Como se pudo apreciar en la Figura 65, la señal sinusoidal fue perfectamente acoplada a la salida del circuito montándose en una señal de 2.5V y con una amplitud de salida no mayor a 5Vpp.

Prototipo terminando: graficación y análisis de la señal ECG en MATLAB, OCTAVE y PYTHON.

En las Figuras 66 y 67 se muestra el impreso del circuito acoplamiento (diseñado en el apartado anterior) y el sistema de adquisición de datos (véase subtema 3.1.1).

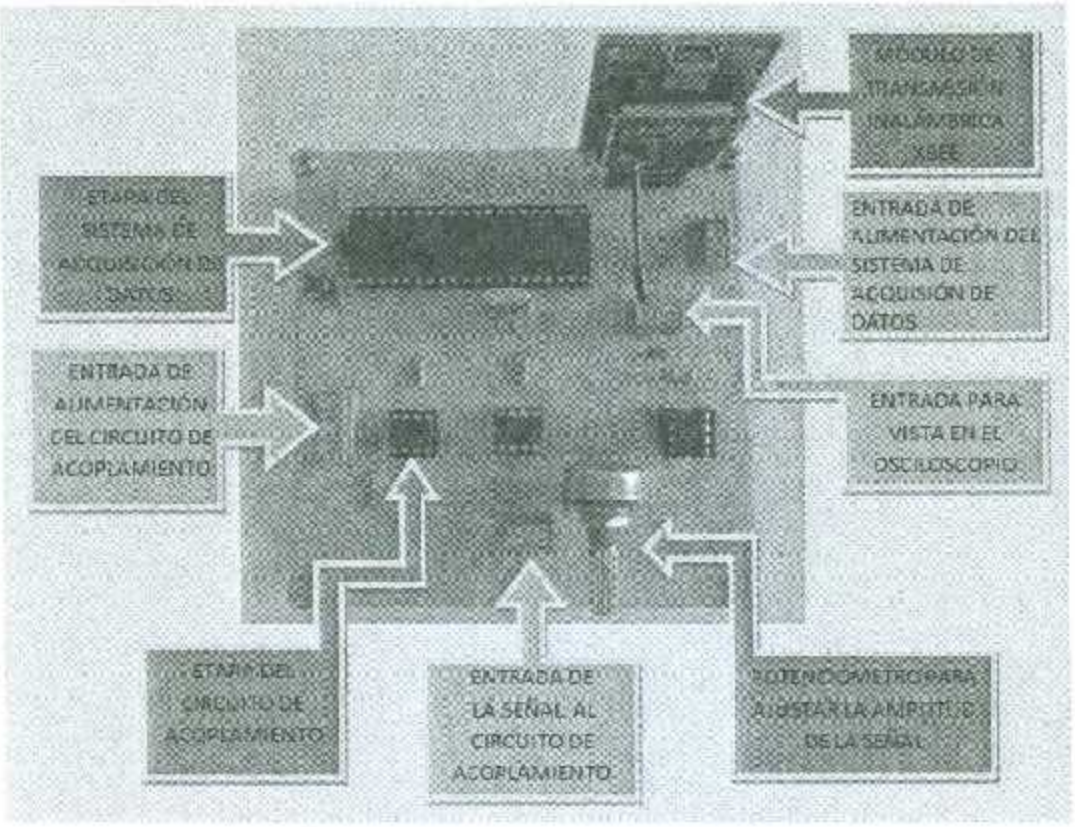


Figura 66. Circuito impreso de la etapa de acoplamiento y adquisición de datos.

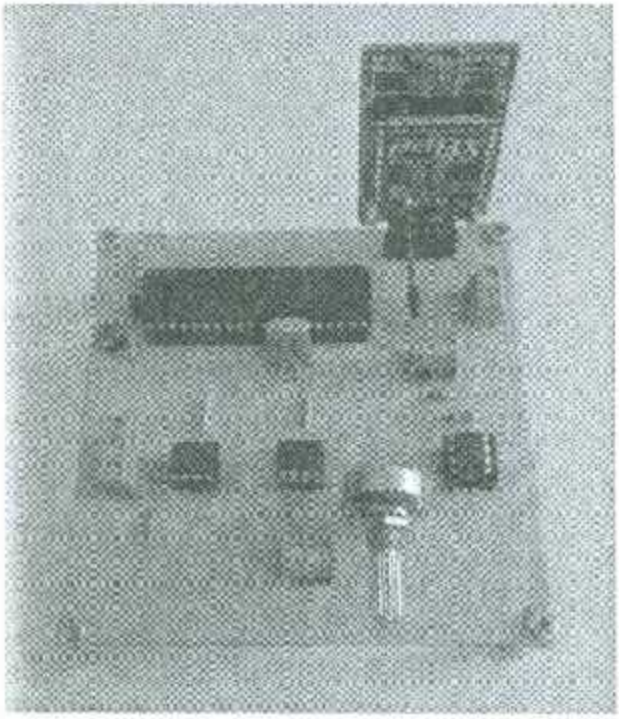


Figura 67. Circuito impreso de la etapa de acoplamiento y adquisición de datos vista de otro ángulo.

Terminadas todas las etapas, es momento de montar el ECG digital en un solo prototipo como se muestra en la Figura 68.

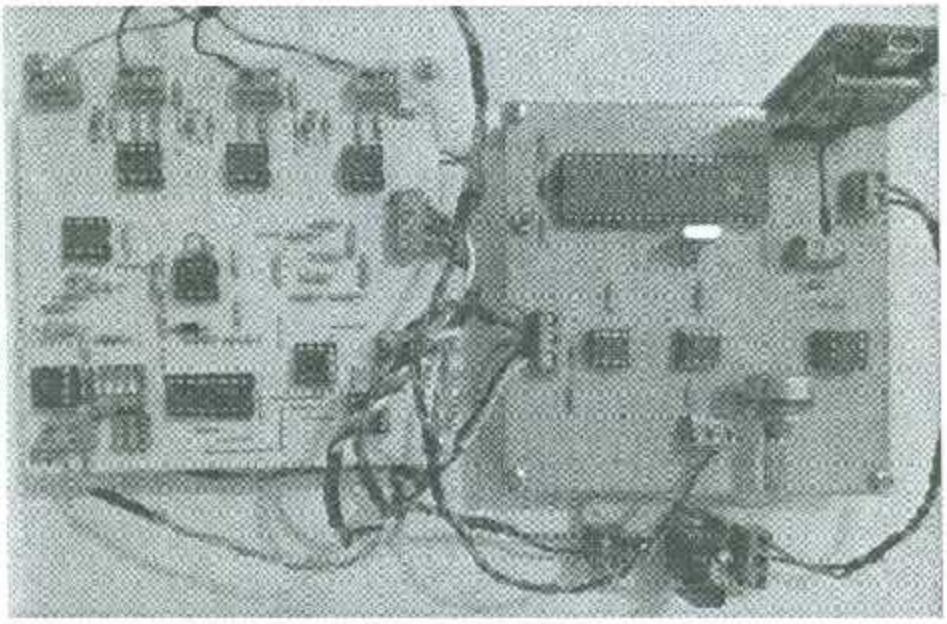


Figura 68. Prototipo de ECG Digital terminado.

El siguiente paso es conseguir un sujeto de prueba (en este caso un servidor) y utilizar los graficadores diseñados en MATLAB, OCTAVE y PYTHON para el posterior análisis de la señal ECG. Antes de comenzar, es importante agregar a los graficadores creados la capacidad de obtener la Frecuencia Cardiaca en unidades de latidos por minutos (lpm) para propósitos clínicos, esto se logra multiplicando la frecuencia de la señal cardiaca obtenida por sesenta ($lpm = Fx60$) este proceso se muestra en las Figuras 69,70 y 71.

```
% --- Executes on button press in calcularf.
function calcularf_Callback(hObject, eventdata, handles)
% hObject    handle to calcularf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
b1=eval(get(handles.DISPT2,'String'));
a1=eval(get(handles.DISPT1,'String'));
dp=b1-a1;
fr=1/dp;  %frecuencia
fc=fr*60; %frecuencia cardiaca
set(handles.frecu, 'String',fc);
```

Figura 69. Obtención de la Frecuencia Cardiaca en unidades de latidos por minuto en el código de MATLAB.

```
#==============MEDIDAS=====================
disp('Calculo de la Frecuencia Cardiaca:')
disp('Posicione el cursor T1')
t1=ginput()
disp('Posicione el cursor T2')
t2=ginput()
f=1/(t2-t1);
fc=f*60;
disp('La Frecuencia Cardiaca es:')
fc
```

Figura 70. Obtención de la Frecuencia Cardiaca en unidades de latidos por minuto en el código de OCTAVE.

```
pts = ginput(2) # espera por dos clicks para confirmar
int1, int2 = pts
t1=int1[0]
textbox2.insert(Tk.INSERT,"%f S\n" % (t1))
textbox2.insert(Tk.INSERT,"Coloque cursor T2:\n")
pts = ginput(2) # espera por dos clicks para confirmar
int1, int2 = pts
t2=int1[0]
textbox2.insert(Tk.INSERT,"%f S\n" % (t2))
dt=t2-t1;
f=1/float(dt);
fc=f*60; #frecuencia cardiaca
textbox2.insert(Tk.INSERT,"La Frecuencia Cardiaca es: %f lpm\n\n" % (fc))
```

Figura 71. Obtención de la Frecuencia Cardiaca en unidades de latidos por minuto en el código de PYTHON.

Hecho lo anterior, de la Figura 72 a la 80 se muestran los resultados.

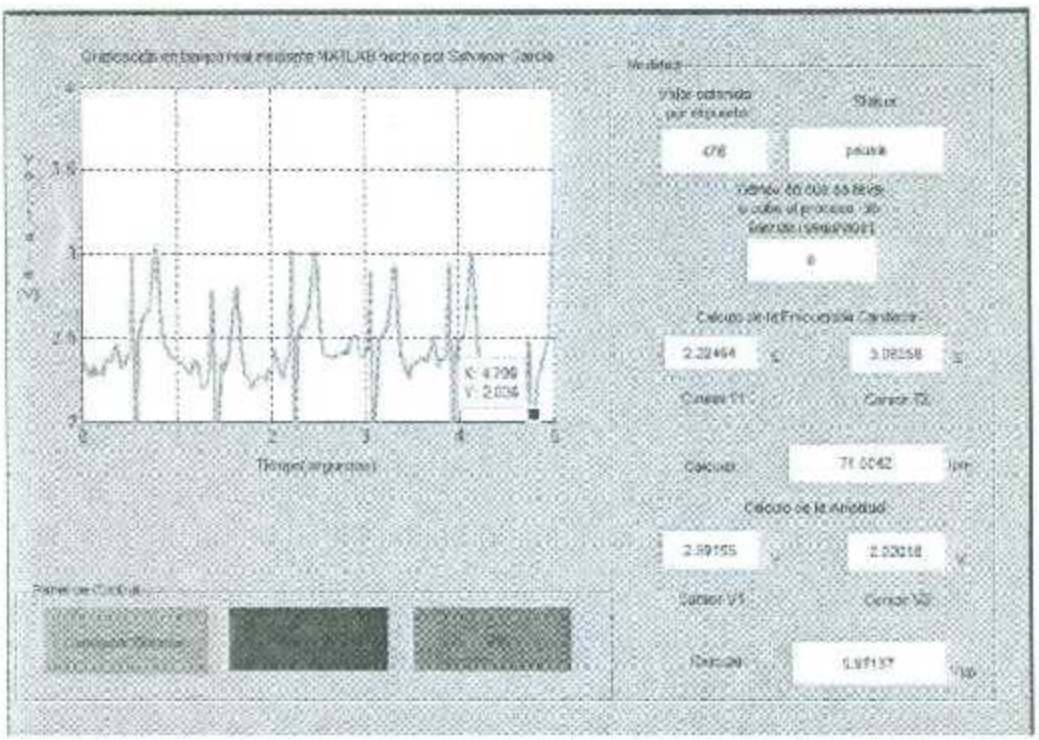


Figura 72. Electrocardiógrafo Digital implementado en MATLAB (Derivación DI).

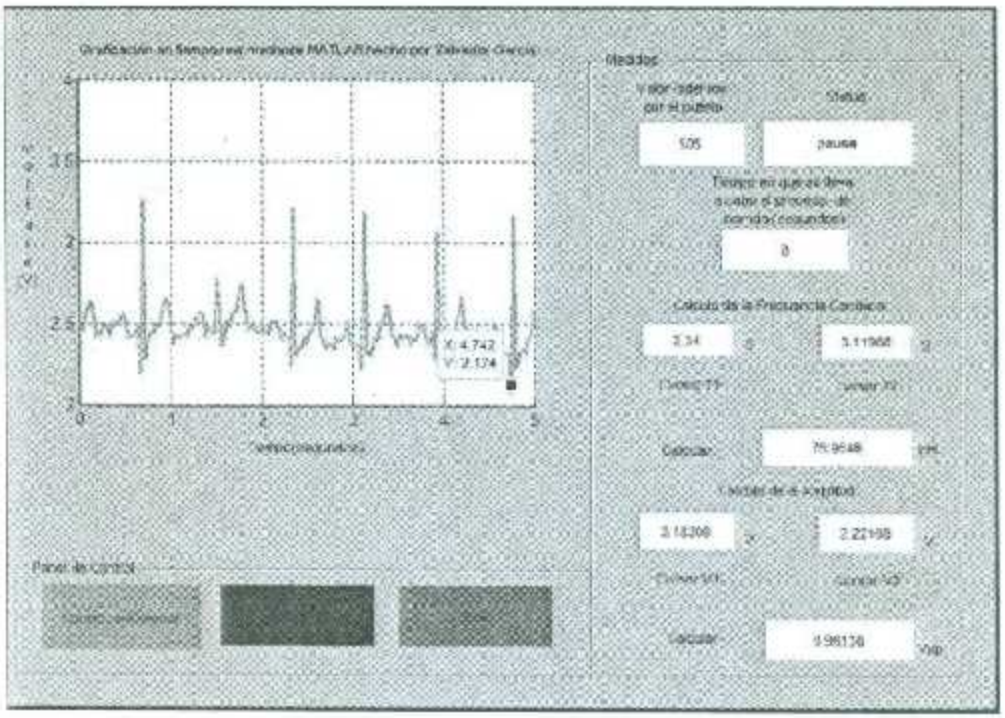


Figura 73. Electrocardiógrafo Digital implementado en MATLAB (Derivación DII).

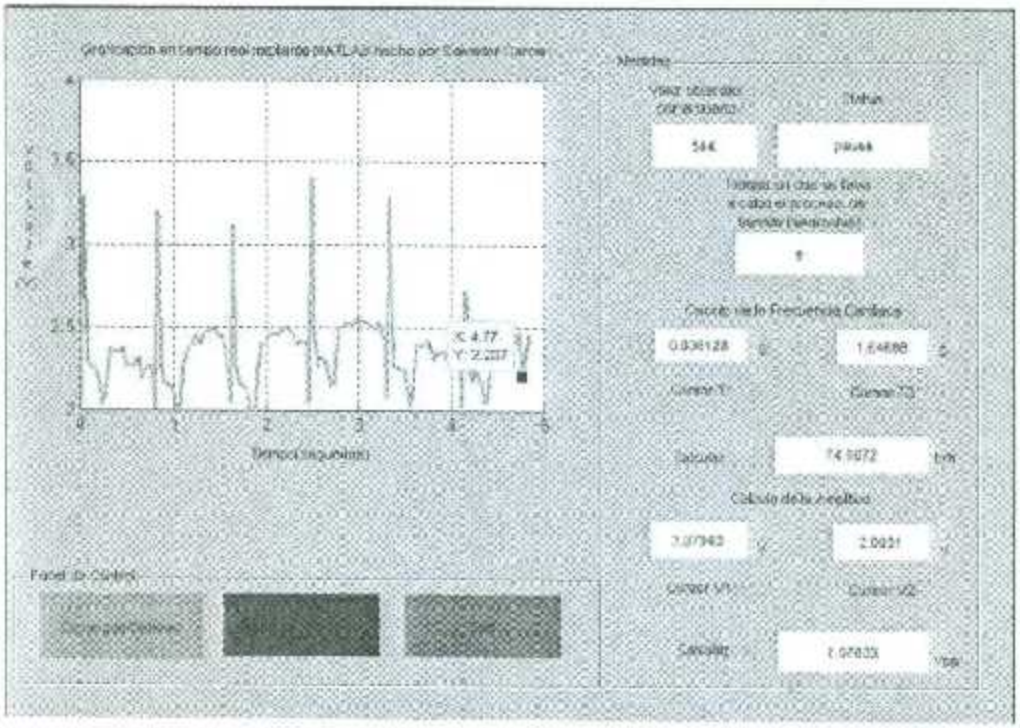


Figura 74. Electrocardiógrafo Digital implementado en MATLAB (Derivación DIII).

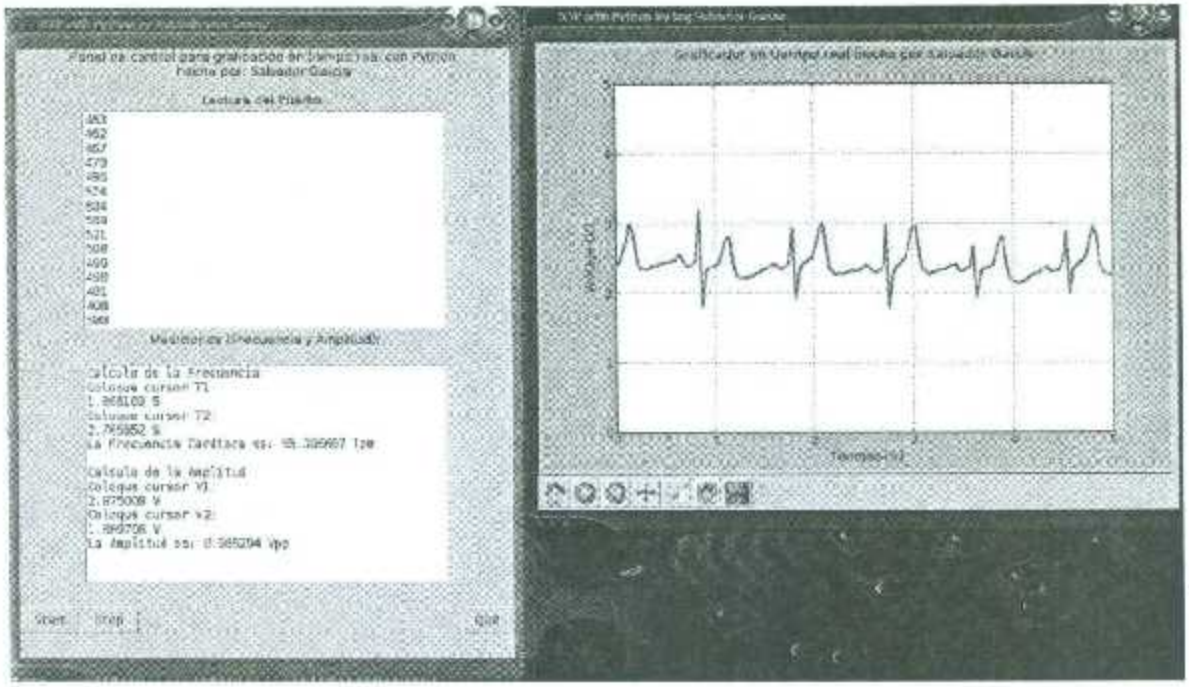


Figura 75. Electrocardiógrafo Digital implementado en PYTHON (Derivación DI).

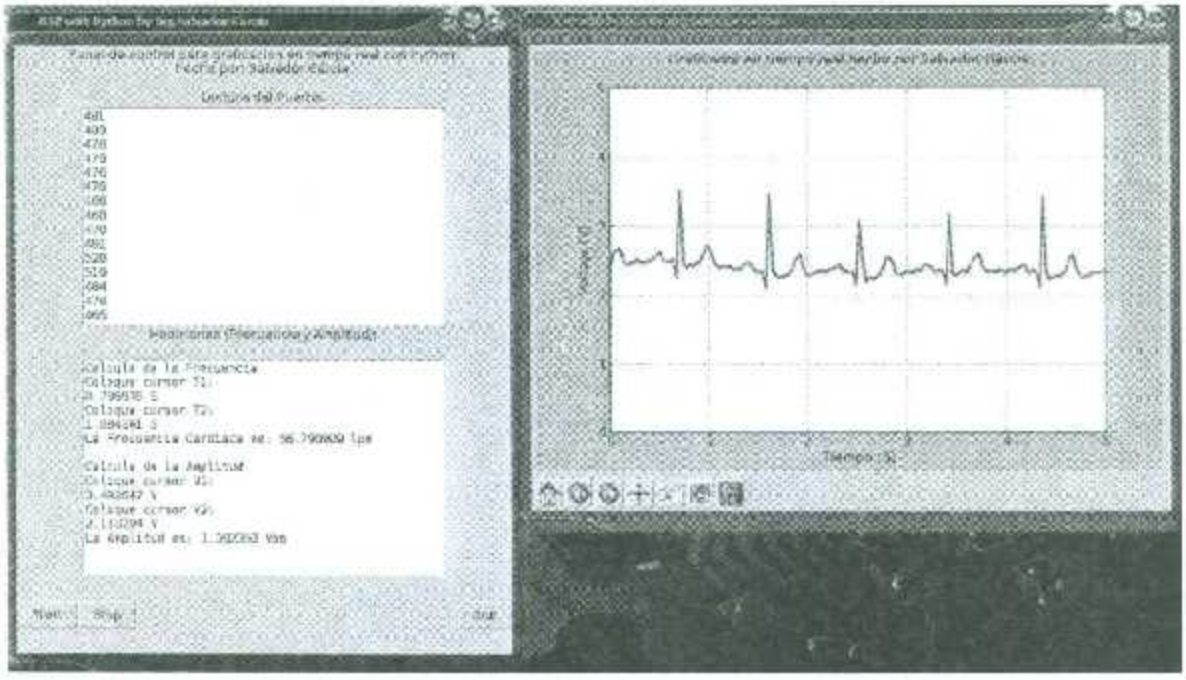


Figura 76. Electrocardiógrafo Digital implementado en PYTHON (Derivación DII).

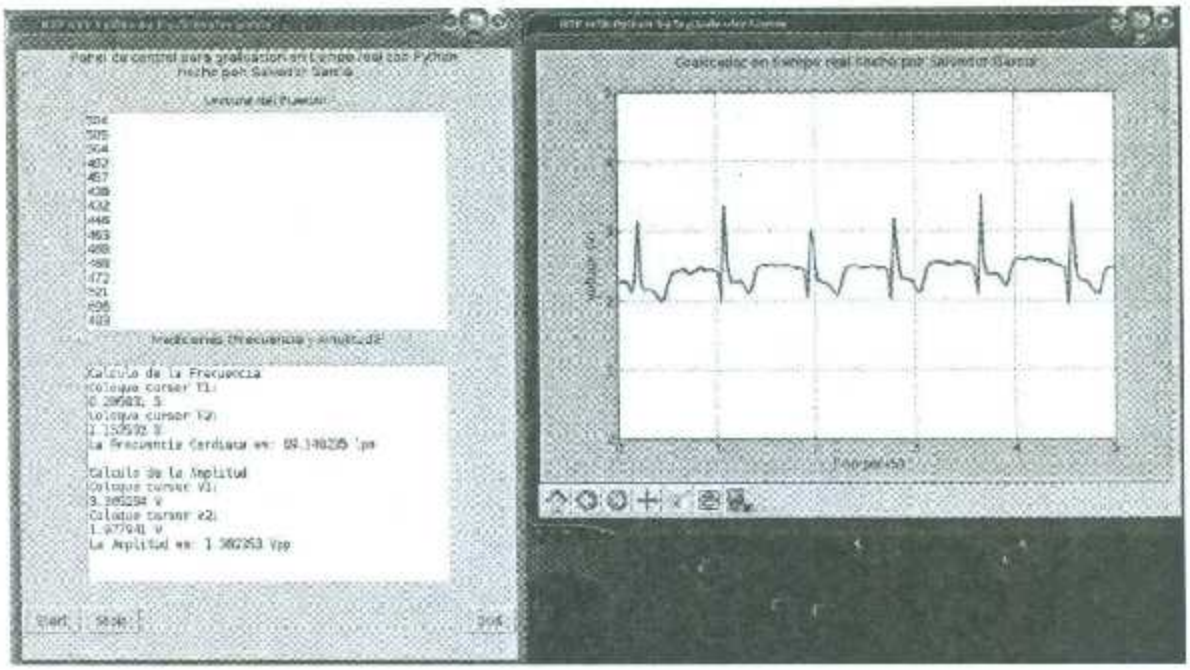


Figura 77. Electrocardiógrafo Digital implementado en PYTHON (Derivación DIII).

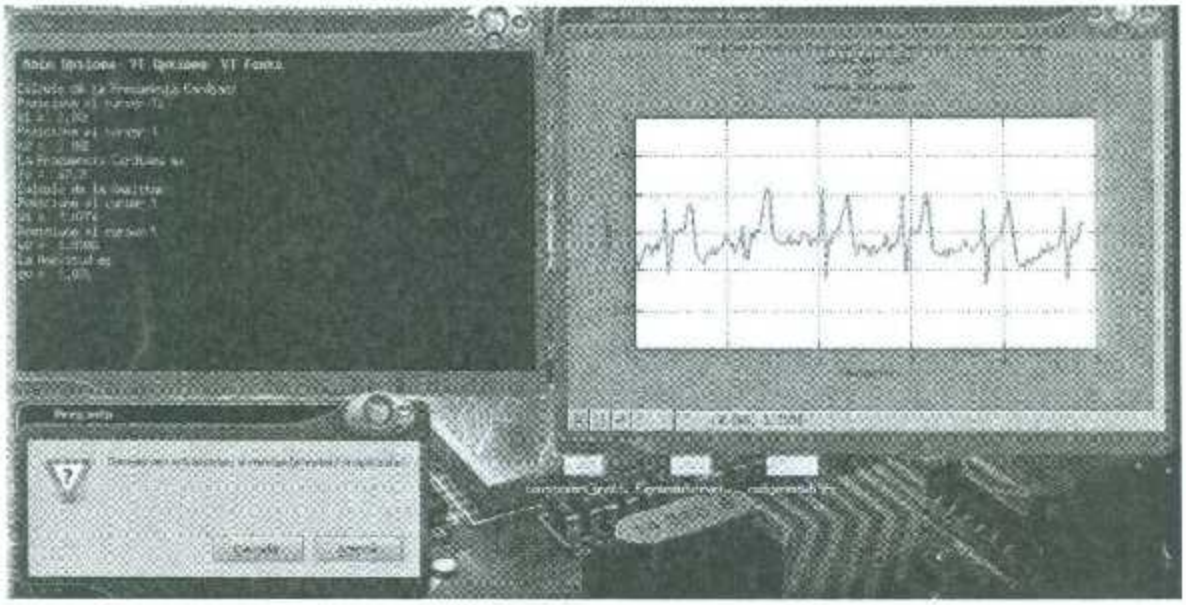


Figura 78. Electrocardiógrafo Digital implementado en OCTAVE (Derivación DI).

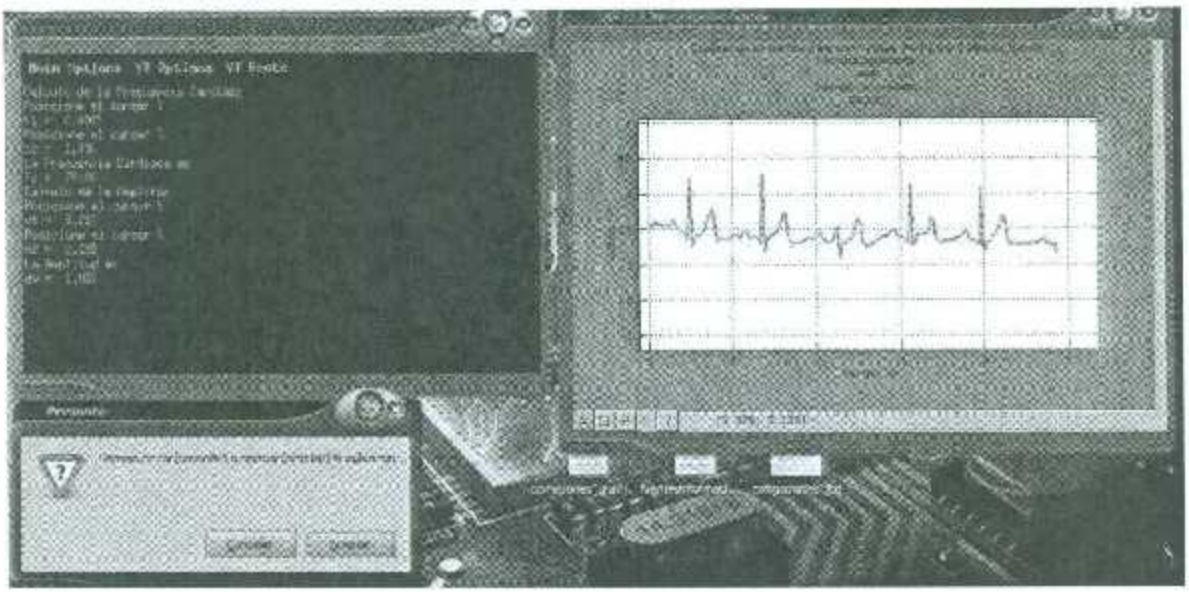


Figura 79. Electrocardiógrafo Digital implementado en OCTAVE (Derivación DII).

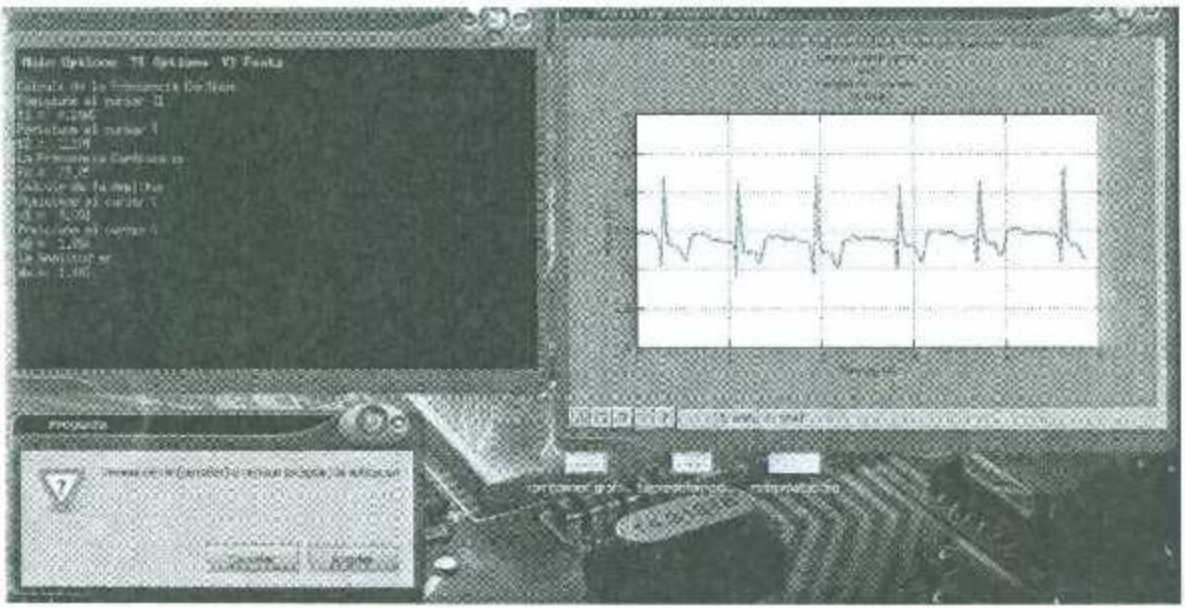


Figura 80. Electrocardiógrafo Digital implementado en OCTAVE (Derivación DIII).

Como se pudo observar en las Figuras anteriores, se han obtenido resultados muy favorables en cada uno de los graficadores. Si bien cada uno de ellos posee sus diferencias, en todos se aprecia la señal cardiaca y se puede realizar la medición tanto de la frecuencia como de la amplitud.

Para realizar un análisis sencillo, se ha optado por la medición frecuencia cardíaca (FC) ya que es uno de los parámetros no invasivos más utilizado en el análisis y en la valoración de la actividad cardiaca. En una persona sana, en estado basal, los latidos se producen con una frecuencia más o menos estable; sin embargo como es natural puede existir cierta variabilidad (en milisegundos).

La variabilidad de la frecuencia cardíaca está relacionada con la influencia del Sistema Nervioso Autónomo o Vegetativo, que es quien "controla" la actividad del Nodo Sinusal, pudiendo indicar la predominancia del Sistema Nervioso Simpático o Parasimpático [18].

Los aumentos de variabilidad de la frecuencia cardíaca están ligados a un predominio del sistema nervioso parasimpático o vagal, mientras que una reducción de la variabilidad está relacionada con el predominio del sistema nervioso simpático [18].

En la Figura 81, de los experimentos anteriores, se puede observar una señal cardiaca con una frecuencia cardiaca aproximada de 71 lpm. Del dato anterior se podría concluir que el sujeto de prueba (realizando un análisis sencillo) presenta un ritmo cardiaco promedio y sano ya que según estudios, los niños de 10 años o más y adultos (incluso ancianos) poseen un FC de 60 a 100 latidos por minuto [19].

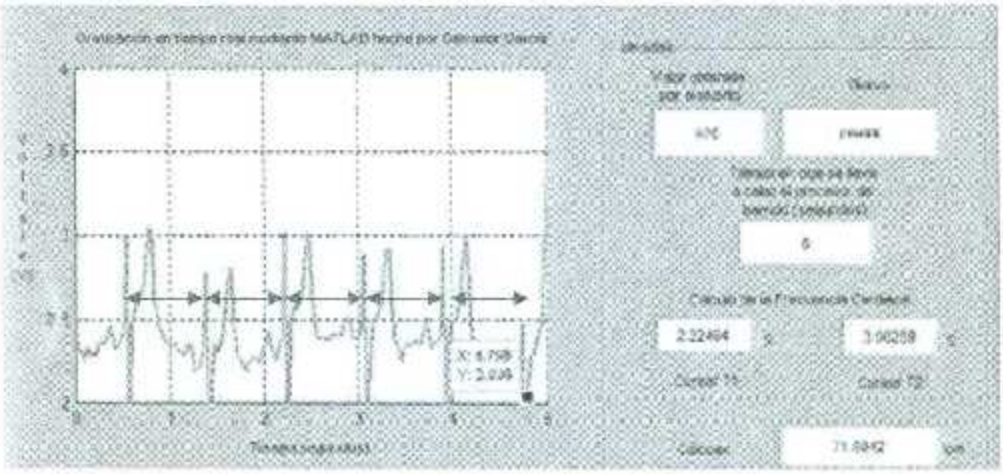


Figura 81. Análisis de la frecuencia cardiaca en MATLAB.

Otro punto importante que se pudiera obtener de este análisis es que si por algún motivo la frecuencia cardíaca en reposo del paciente se mantuviera continuamente alta pueden ser indicio de un problema (taquicardia). Por otro lado, sería importante consultar respecto a frecuencias cardíacas en reposo que estén por debajo de los valores normales (bradicardia).

Por supuesto, es necesario que un médico calificado revise al sujeto de prueba para dar una valoración clínica y un diagnostico mucho más completo.

CONCLUSIONES:

En este apartado se incluyen las conclusiones acerca de las actividades desarrolladas durante el periodo del proyecto de investigación, agosto de 2013 a junio de 2014. Se han clasificado las conclusiones en varios rubros:

Conclusiones de orden técnico:

1.- Se considera que se ha cumplido con todos los objetivos para desarrollar interfaces de *hardware* y *software* para el procesamiento y análisis de la señal ECG, las cuales, según las pruebas realizadas han resultado exitosas.

2.- Se dedicó gran cantidad de tiempo al estudio y desarrollo de programas para la graficación en tiempo real de señales analógicas para su posterior análisis. De ahí, se pueden sacar las siguientes conclusiones.

- MATLAB sin duda alguna es una potente herramienta y debido a ello se obtuvieron excelentes resultados para la adquisición y análisis de la señal ECG. Sin embargo, el costo de sus licencias y algunas restricciones al momento de realizar programación son limitantes importantes que no lo hacen ser un mejor *software*.

- OCTAVE a diferencia de su contraparte, es un programa que aunque carece de interfaz gráfica y sus herramientas para graficación son más sencillas y menos potentes que las de MATLAB, da la enorme ventaja de ajustar la programación a las necesidades del usuario (incluyendo sistema operativo) y con ello convertirlo en desarrollador para crear librerías propias dando resultados muy favorables para este proyecto.

- PYTHON, aunque su lenguaje es totalmente diferente al de MATLAB y OCTAVE, no posee problemas de licencias, su programación es versátil y multipropósito con una gran gama de librerías haciéndolo una herramienta muy buena para la graficación en tiempo real de la señal ECG con resultados muy buenos. No obstante, los inconvenientes de PYTHON se presenta dentro de su propia ventaja, es decir, al ser modularizado, su programación es sumamente "quisquillosa" creando constantes conflictos al momento de compilar si no se tiene la experiencia necesaria.

- Considero que el programa que se desempeño mejor durante las pruebas del ECG fue sin lugar a duda MATLAB debido a los pocos problemas de saturación del buffer y del graficador al momento de adquirir la señal. Si bien OCTAVE y

PYTHON aún no superan a su competidor, son programas que poseen un futuro muy prometedor y dentro de poco estarán a la par.

3.- El desarrollo de un ECG requiere de paciencia y análisis minucioso de cada elemento y etapas que lo compone para así obtener un buen funcionamiento del dispositivo.

4.- Es de vital importancia la calidad de los electrodos al momento de realizar las pruebas con un electrocardiógrafo, ya que de eso dependerán los buenos resultados.

Conclusiones de orden personal:

Considero que el desarrollo de un ECG portátil y de bajo costo resolverá la reducida capacidad de adaptación a las necesidades de los usuarios (hospitales, médicos, estudiantes y de pequeñas empresas) y los altos costos de *software* y *hardware*, que son problemas comunes en los sistemas electrocardiográficos comerciales y con ello incursionar en el mercado nacional, incluso internacional.

Conclusiones hacia la institución:

1.- En lo personal me ha parecido que la Maestría en Eléctrica debe de apoyar con mejores servicios de equipo de laboratorio y lugares de trabajo para que así el alumno se desempeñe adecuadamente durante el desarrollo de su proyecto de investigación incrementado la eficacia y las oportunidades de desarrollo para tesistas y profesores investigadores.

2.- Hacer lo posible por que los proyectos estén vinculados con la empresa, para que así los investigadores aporten su capital intelectual para ayudar a concebir las mejoras específicas que requieren estas mismas, que entre otras cosas podrían traducirse en el lanzamiento de nuevos productos patentados con beneficios para los investigadores, las instituciones universitarias y las empresas. Con ello, al egresar los estudiantes no se limitarán a ingresar al mercado laboral, sino que propondrán mejoras a lo que ya se hace en la industria, al no repetir los procesos ya existentes, sino creando otros mejores que permitan a México ser más competitivo frente a otros países.

ANEXO A

Instalación de Cygwin y OCTAVE

La instalación de estos programas es relativamente sencilla, solo basta con seguir los siguientes pasos

1. Descargar el instalador del siguiente enlace: http://cygwin.com/setup.exe
2. Crear la carpeta "C:\Cygwin"
3. Copiar el archivo setup.exe y se coloca en la carpeta creada anteriormente
4. Ejecutar
5. Cuando solicite el programa donde descargara los paquetes, seleccionar "Install from internet" (Véase Figura 82) [12].

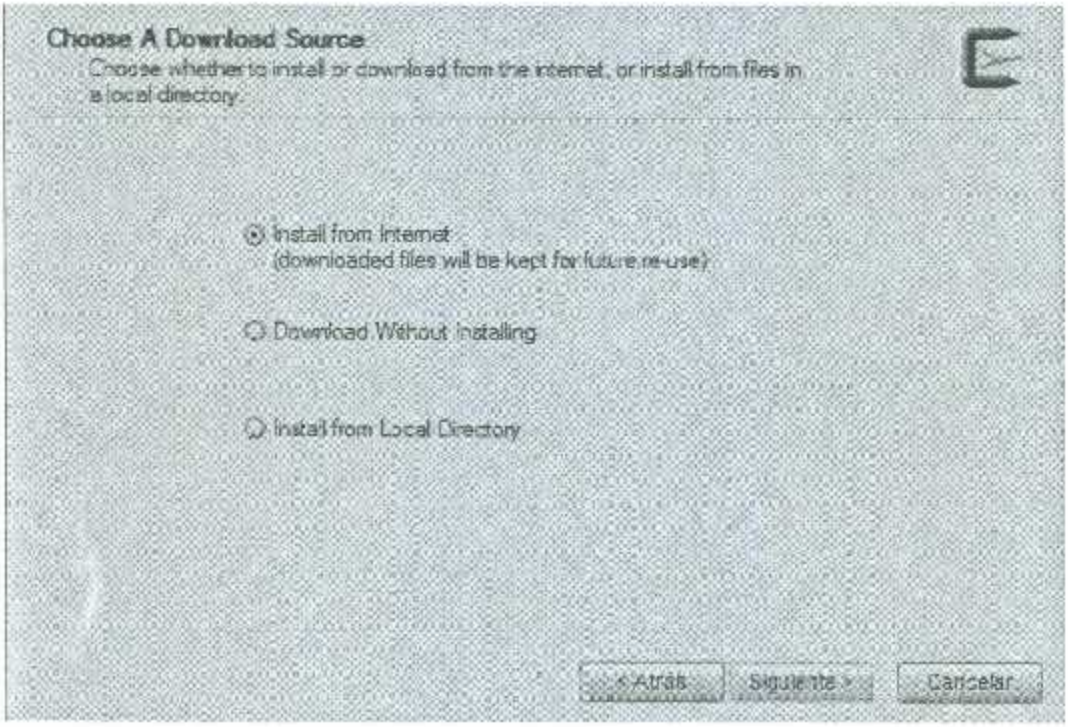


Figura 82. Descarga de paquetes en Cygwin

6. Para la ruta de instalación se selecciona, "C:/Cygwin" (Véase Figura 83).

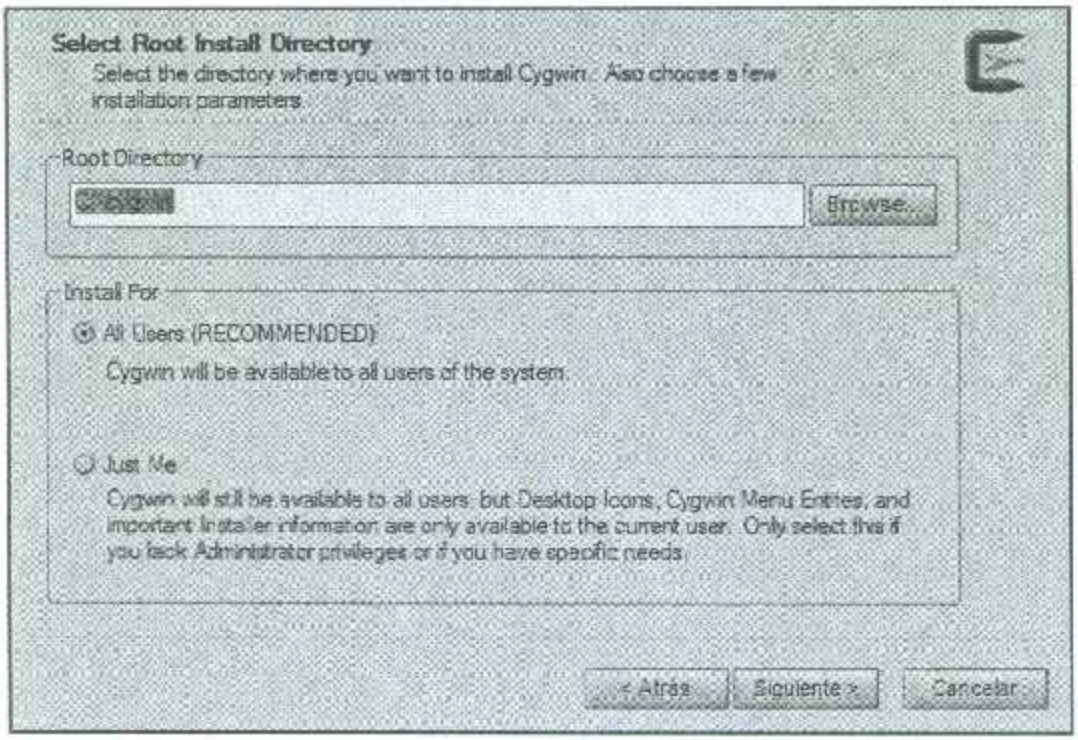


Figura 83. Ruta de instalación Cygwin.

7. Como directorio de paquetes también se elige "C:/Cygwin" (Véase Figura 84).

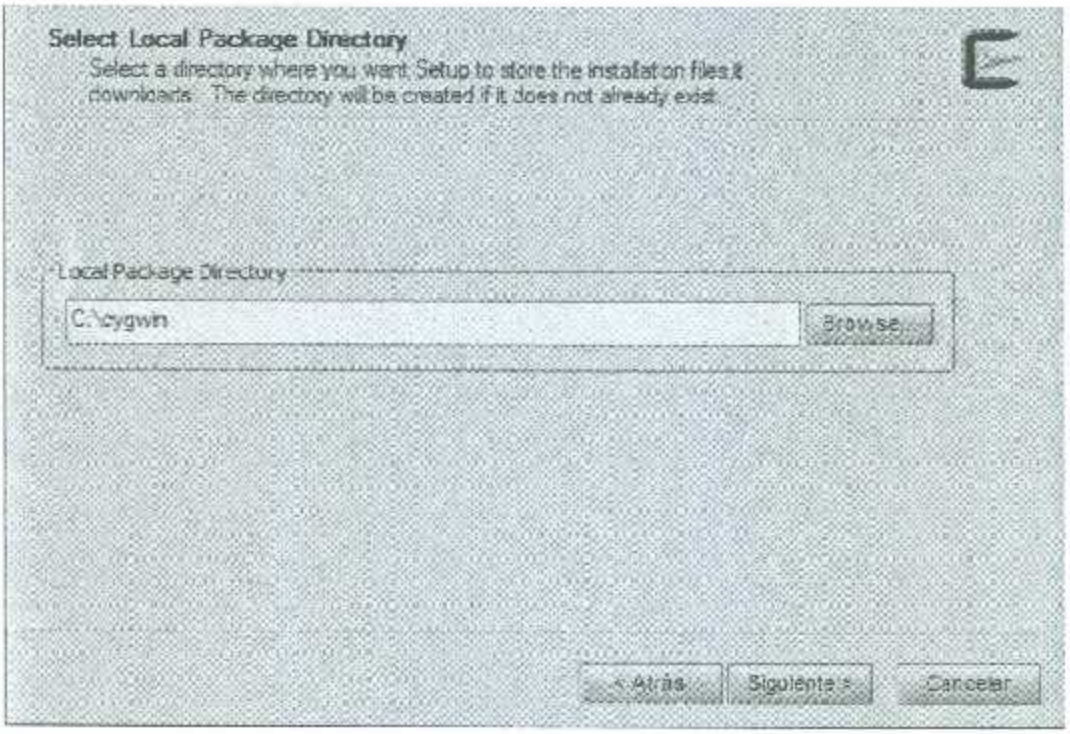


Figura 84. Directorio de paquetes.

8. En el siguiente paso se elige la casilla "Direct Connection" (Véase Figura 85).

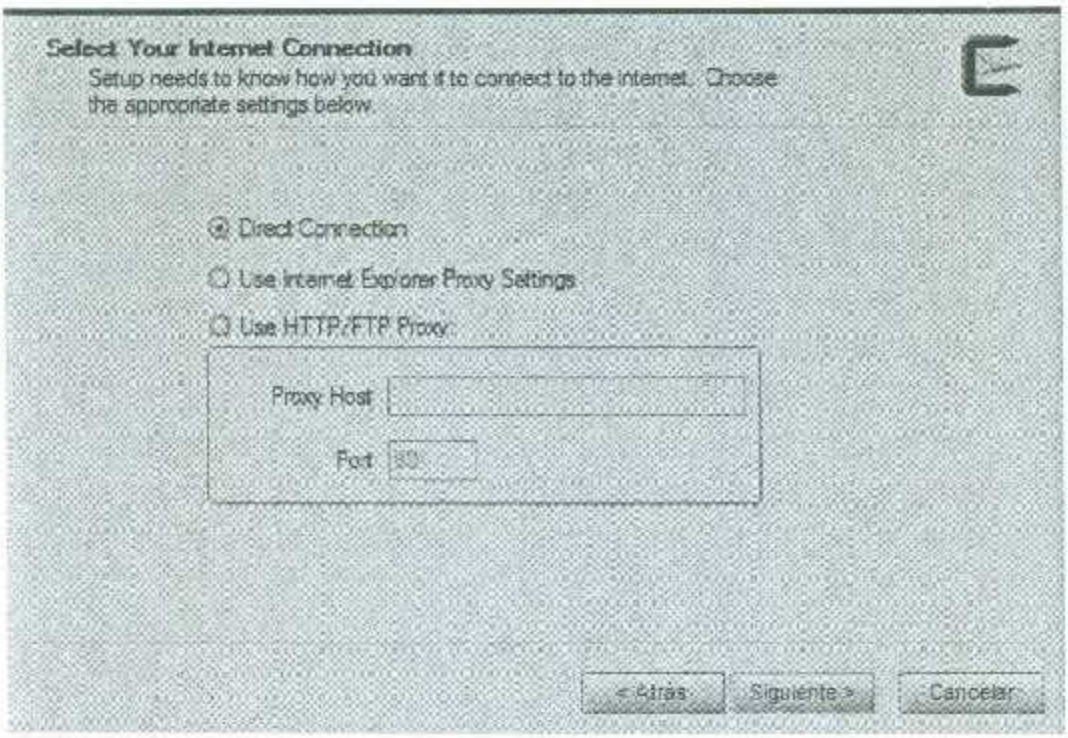


Figura 85. Casilla "Direct Connection".

9. Se elige un servidor. Se sugiere el primero en la lista por proximidad [12] (Véase Figura 86).

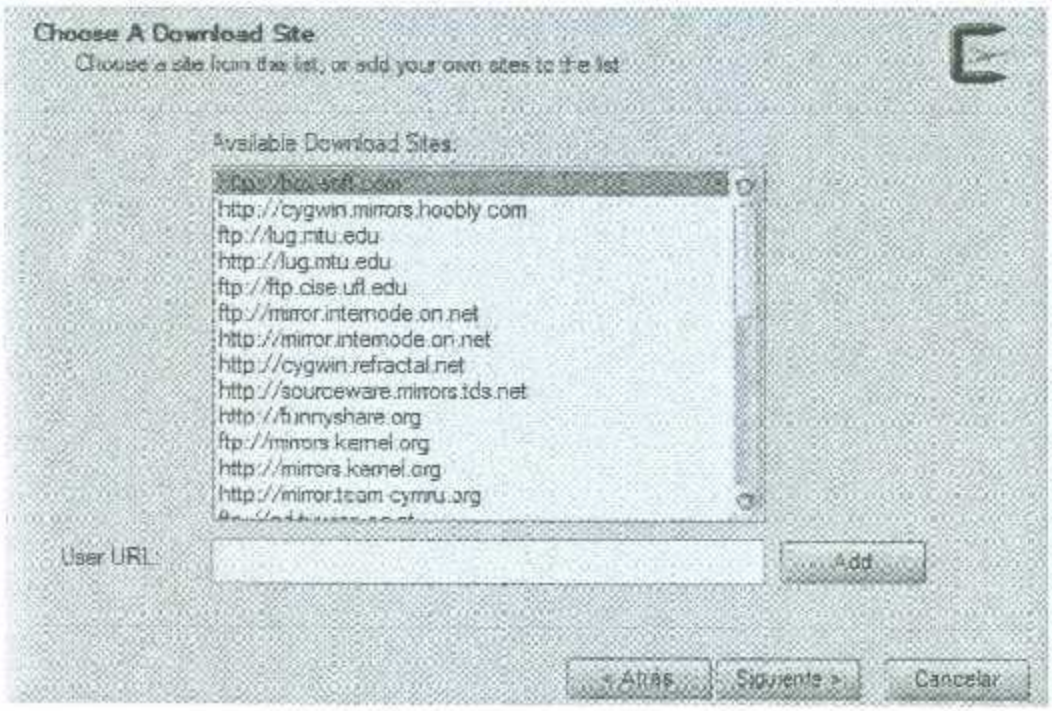


Figura 86. Elección de servidor de descarga.

10. Tras unos segundos, aparecerá una lista con todos los paquetes que se pueden instalar. En este ejemplo, se desea instalar el programa OCTAVE. Para ello, en la casilla de búsqueda simplemente se escribe "octave", y enseguida se mostrara una lista de todos los componentes necesarios para el programa (Véase Figura 87).

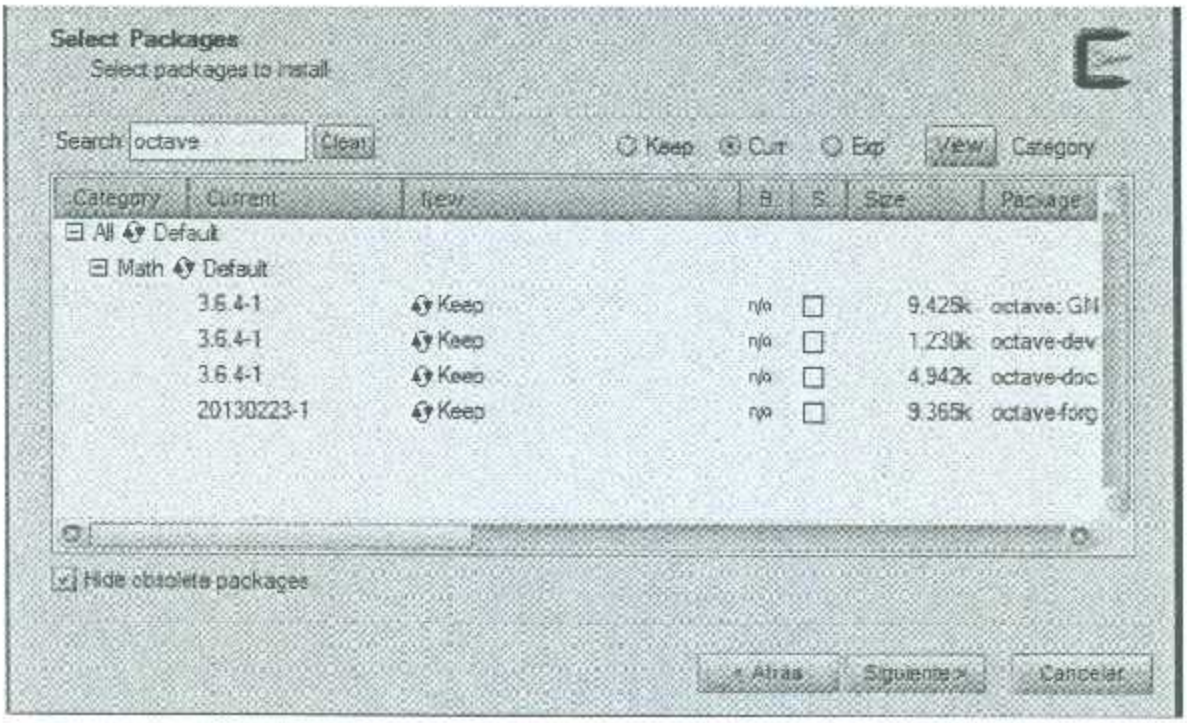


Figura 87. Selección de paquetes.

11. Se da "click" en siguiente, y se descargaran e instalaran tanto los paquetes básicos como los que hemos seleccionado.

12. Una vez terminado el proceso de descarga e instalación, se pregunta si se desea un acceso directo en el escritorio (Véase Figura 88).

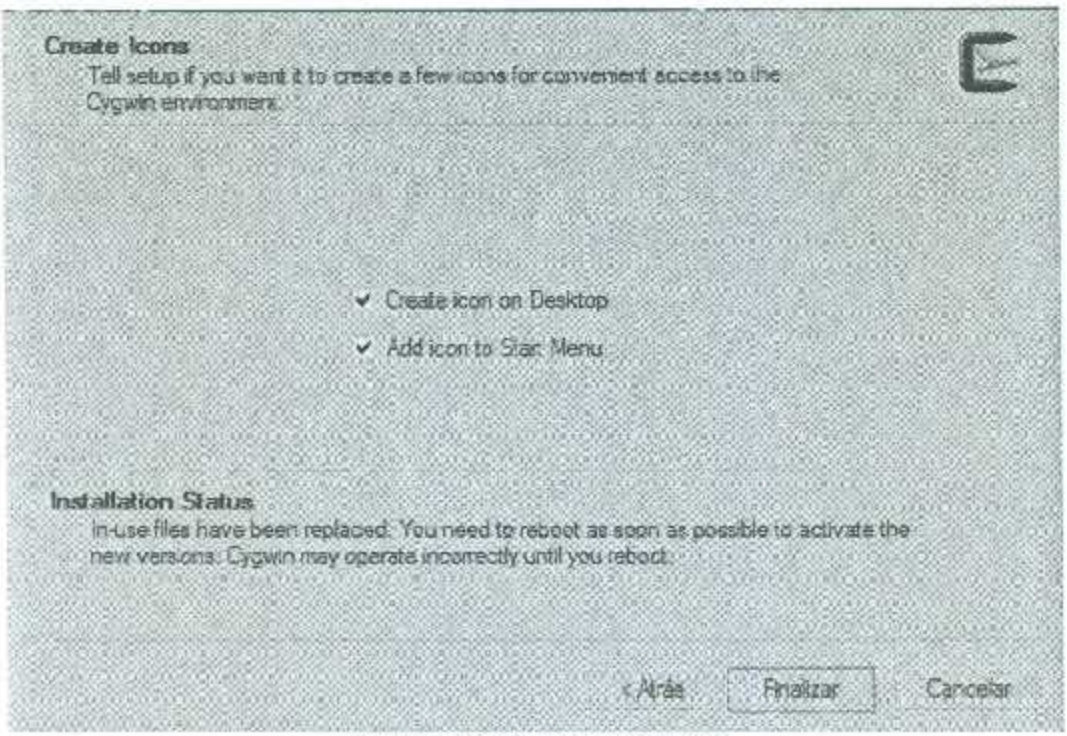


Figura 88. Creación de acceso directo.

Ahora ya se tiene instalado Cygwin y se puede utilizar la consola de Windows como si fuera de Linux para poder ejecutar el programa OCTAVE. Si se ha elegido "crear el icono de acceso directo en el escritorio", solo basta con dar doble "click" sobre él (Véase Figura 89).

Figura 89. Instalación de Cygwin y Octave satisfactoriamente.

ANEXO B

Aspectos importantes para la programación en OCTAVE

El uso de programas de Linux bajo Windows es complicado. En realidad, no es posible de correr programas de Linux directamente, ya que se necesitaría de demasiadas funciones del sistema operativo, es por ese motivo que se utiliza la herramienta Cygwin.

El soporte básico para las ventanas de Linux se llama X11. Encima de esta base, se construyeron las herramientas para el dibujo/funcionamiento de los elementos como botones, listas, desplegables, etc. Hay dos librerías grandes: GTK+ y Qt. En este caso se utilizo GTK+ [12].

Proveer los elementos de ejecución bajo Windows, lo hace el paquete Cygwin (como ya se mencionó anteriormente). Al instalar Cygwin, por defecto, se instala solamente una mínima parte. Para que funcione X-Windows, es preciso habilitar, durante la instalación de Cygwin, al paquete completo de X11 (esto se puede realizar, utilizando los pasos del ANEXO A).

Para correr programas con interfaz gráfica, en este caso OCTAVE, es preciso arrancar X11 antes, mediante el comando 'startxwin.bat' como se muestra en la Figura 90.

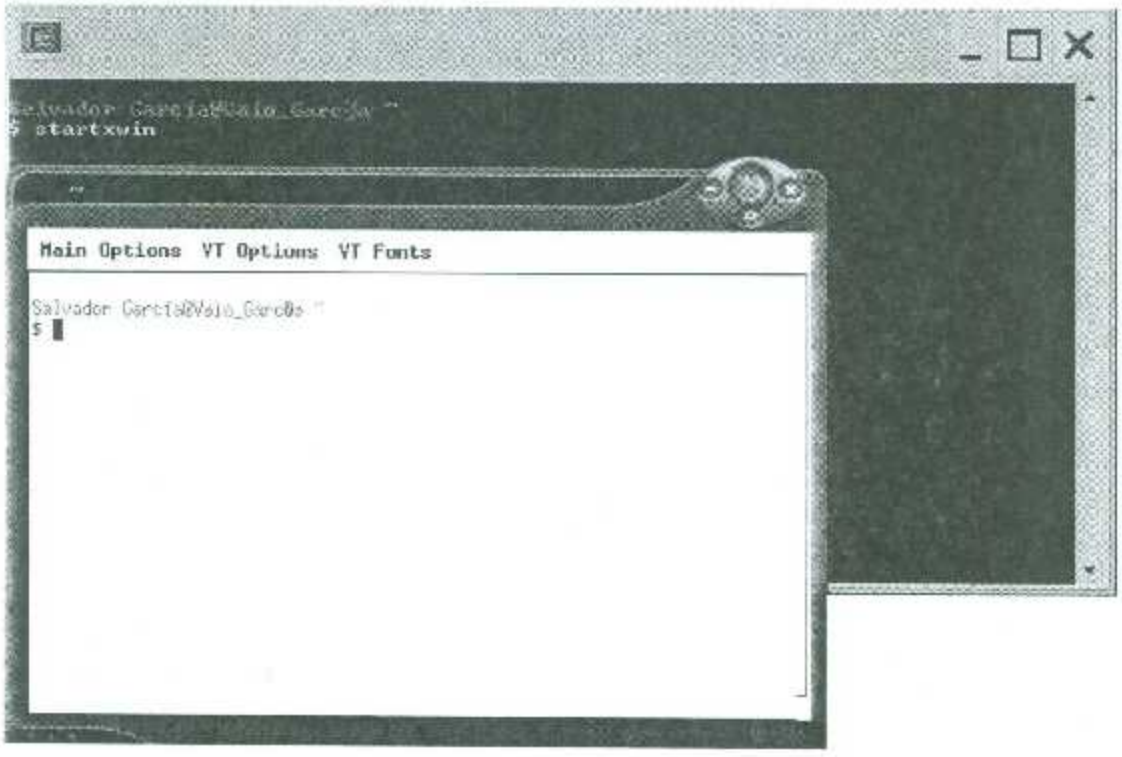


Figura 90. Inicio del servidor X11.

Enseguida de haber arrancado el soporte X11, se escribe el comando "octave" e inmediatamente se inicializara el paquete OCTAVE (véase Figura 91).

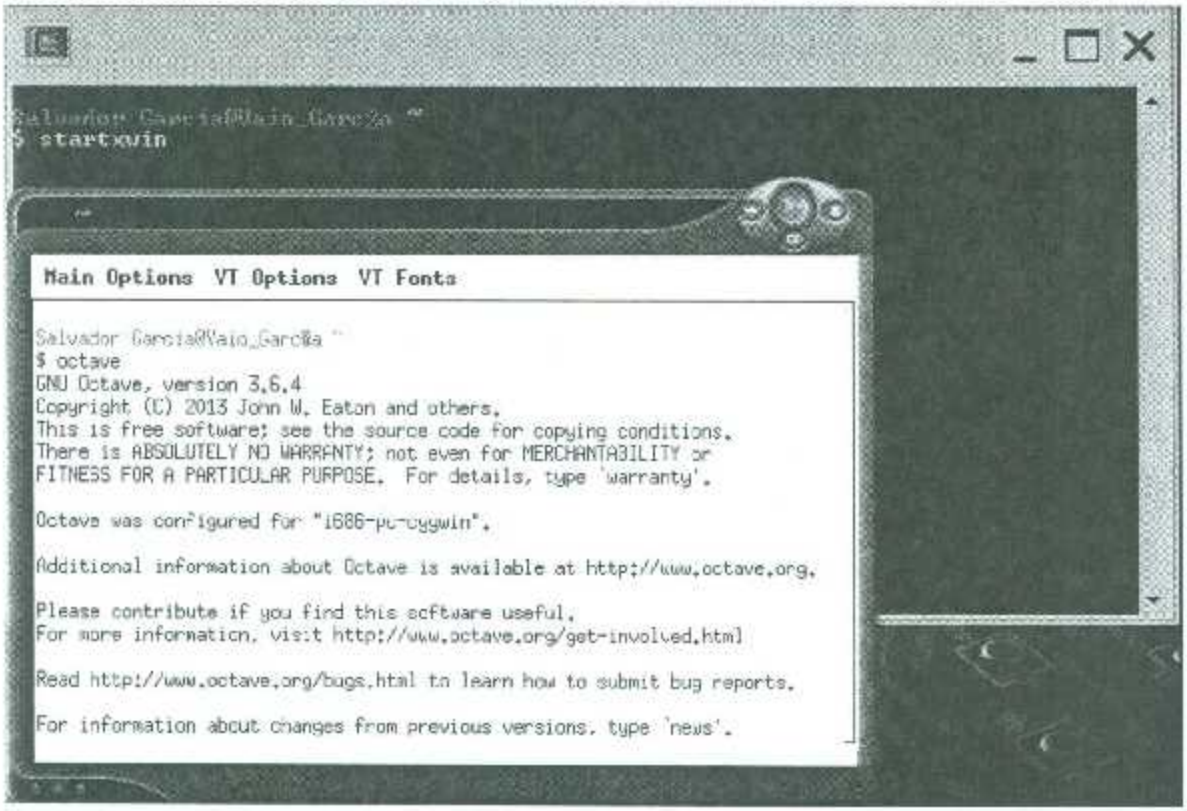


Figura 91. Inicialización del paquete Octave.

Ejecutando OCTAVE por primera vez

La presente guía tiene como objetivo, mostrar las características básicas y necesarias para comenzar a programar en el paquete OCTAVE. Sin embargo, si desea mayor información, existe un manual on-line en inglés en la dirección http://octave.org/doc/octave_toc.html.

Cuando el software se inicia, se muestra una ventana donde se puede escribir los comandos que necesitemos. Además, Octave puede graficar nuestros datos y resultados en otra ventana.

Para salir de Octave bastará con escribir **"exit"** y presionar **ENTER,** Lo anterior se muestra en la Figura 92.

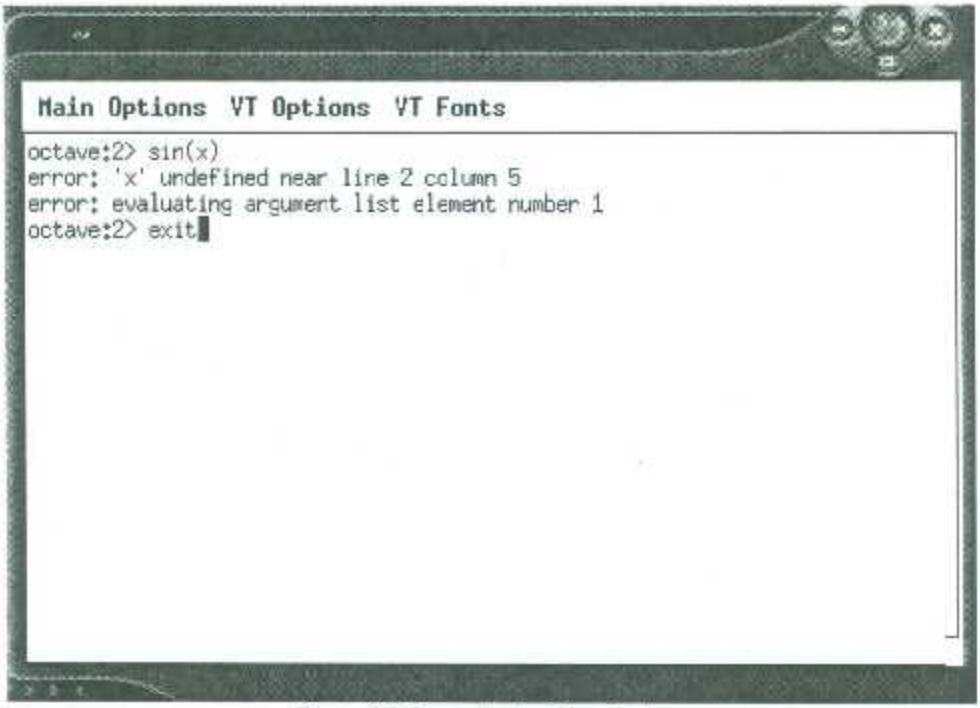


Figura 92. Comando "exit" en Octave.

Tipos de Datos en OCTAVE

Todas las versiones de Octave incluyen una serie de tipos de datos predefinidos, que incluye reales, complejos, escalares, matrices cadenas de caracteres y tipos de datos estructurados. Un tipo de datos no sólo define al dato sino que además define que operaciones pueden hacerse sobre él y con otros tipos de datos.
Es posible definir nuevos tipos de datos escribiendo un poco de código en C++. Estos tipos de datos pueden ser cargados al iniciarse Octave o durante la ejecución de Octave. Esto no será necesario en nuestro caso [13].

Tipos de Datos Numéricos

En este apartado veremos las más importantes, constantes numéricas y matrices.

Constantes Numéricas:

Una constante numérica puede ser un escalar, un vector o una matriz o un número imaginario. La forma más simple de una constante numérica, un escalar, es un número simple que puede ser un entero, una fracción decimal, un número en notación científica o un número complejo. Todas las constantes numéricas en Octave son representadas internamente como números de punto flotante en doble precisión (los números complejos se representan como dos valores flotantes). Un número de punto flotante es lo que se conoce como real. Así algunos valores reales que tienen el mismo valor numérico serán, por ejemplo, **105 ENTER, 1.05e+2 ENTER, 1050e-1 ENTER** [13].

Estos valores se pueden ingresar directamente escribiéndolos y Octave nos mostrará el formato más simple del valor ingresado.

Una constante numérica compleja se escribe como una expresión de la siguiente forma: **3 + 4i ENTER, 3.0 + 4.0i ENTER, 0.3e1 + 40e-1i ENTER.** La única condición es que no haya un espacio entre el número y la **i** que representa el valor de la raíz cuadrada de -1.

Matrices

Es muy sencillo definir una matriz de valores en Octave. El tamaño de la matriz es determinado automáticamente, así la expresión **a = [1, 2; 3, 4] ENTER**, resulta en la matriz [13]:

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Nótese que los elementos de la matriz se separan con coma (,) y las filas con punto y coma (;). Los elementos de la matriz pueden ser expresiones arbitrarias siempre que las dimensiones mantengan coherencia, así la expresión **[a , a] ENTER**, será

$$a = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{bmatrix}$$

Variables

Una variable permite asignar un nombre a un valor que será utilizado luego. El nombre de una variable consistirá de caracteres, dígitos y caracteres de subrayados, pero no está permitido que empiecen con un dígito o con doble caracter de subrayado. Los nombres de variable no deben superar los 30 caracteres. Las mayúsculas y minúsculas son consideradas como diferentes caracteres. Como un ejemplo, Son nombre de variables válidos **X , x15, altura** [13].

El comando **clear** *<variable>* **ENTER** borra la variable, la variable deja de estar definida.

Expresiones

Las expresiones son los ladrillos con los que se construyen todo en Octave. Una expresión se evalúa en un valor, que se puede imprimir, comparar, pasar como parámetro a una función o asignar a una variable con una operación de asignación.

Una expresión puede servir como sentencia. Una sentencia puede contener una o más expresiones que incluyan variables, constantes, matrices, o combinaciones de ellas [13].

Expresiones Índices

Una expresión índice permite recuperar ó referenciar elementos de una matriz o un vector. Los índices pueden ser escalares, vectores, rangos o el carácter especial dos puntos (:) que se utiliza para seleccionar la filas o columnas enteras. Los vectores utilizan un solo índice mientras que las matrices utilizan dos [13].

Por ejemplo, dada la matriz **A = [1,2;3,4;5,6] ENTER**, la siguiente expresión **A(2,1)** selecciona (y retorna su valor) el elemento de la segunda fila y la primera columna (el **3**), mientras que las siguientes expresiones seleccionan la primera fila de la matriz **A(1,1:2)** ó **A(1,:)**, la segunda fila de la matriz **A(2,1:2)** ó **A(2,:)** y la segunda columna **A(1:3,2)** ó **A(:,2)**.

Nótese que para seleccionar una columna entera se puede utilizar un rango (1:2 en el ejemplo) ó el carácter especial dos puntos (:)

Operadores Aritméticos

Los operadores aritméticos son elementos que permiten construir expresiones más complejas partiendo de expresiones sencillas [13].

Adición: x + y: Si ambos operandos son matrices el número de filas y columnas debe coincidir. Si un operando es escalar su valor es sumado a cada elemento del otro operando.

Adición elemento por elemento: x .+ y : Este operador suma dos matrices elemento a elemento. Es equivalente a x +y.

Sustracción: x – y

Sustracción elemento a elemento: x .-y

Multiplicación Matricial: x * y : El número de columnas de x debe coincidir con el número de filas de y.

Multiplicación elemento a elemento: x .* y : Si ambos operandos son matrices deben coincidir en dimensión.

División a derecha: x / y : Este concepto es equivalente a efectuar el producto del inverso de y transpuesta por x transpuesta, transpuesto.

División a derecha elemento a elemento: x ./ y

División a Izquierda: x \ y : El concepto es equivalente multiplica el inverso de x por y.

División a Izquierda elemento por elemento: x .\ y : Cada elemento de y es dividido por el correspondiente de x.

Potencia: x ^ y ó x ** y : Si x e y son escalares devuelve x elevado a la potencia y.

Potencia elemento por elemento: x .^ y ó x . y :** Si los dos operandos son matrices las dimensiones deben coincidir.

Negación: -x

Suma unaria: +x : Este operador no tiene efecto sobre el operando.

Conjugado: x' : Para argumentos reales este operador es lo mismo que el operador Transpuesta. Para operandos complejos calcula el conjugado.

Transpuesta: x.'

Operadores de Comparación

Los operadores de comparación, como su nombre lo indica, comparan los valores numéricos con respecto a la igualdad. Todos los operadores de comparación devuelven un valor 1 si la operación es verdadera ó 0 si es falsa. Para matrices los operadores trabajan elemento por elemento. Así por ejemplo, **[1, 2; 3, 4] == [1, 3; 2, 4] ENTER** dará un resultado de unos y ceros elemento por elemento [1,0;0,1]. Si un operador fuera un escalar y el otro una matriz, el escalar se comparará con cada elemento de la matriz [13].

x < y es verdadero si x es menor que y
x <= y es verdadero si x es menor o igual que y
x == y es verdadero si x es igual a y
x >= y es verdadero si x es mayor o igual que y
x > y es verdadero si x es mayor que y
x != y es verdadero si x es distinto que y
x ~= y es verdadero si x es distinto que y
x <> y es verdadero si x es distinto que y

Expresiones Booleanas

Una expresión booleana es una combinación de expresiones de comparación usando operadores "o" (||), "y" (&&) y "no" (!), junto con paréntesis que controlan el orden de las operaciones lógicas. La veracidad de una expresión booleana es computada combinando la veracidad de las expresiones componentes elementales. Un valor cero es considerado falso y un valor distinto de cero es considerado verdadero [13].

x && y: es verdadero si x e y son verdaderos.
x || y: es verdadero si x o y son verdaderos.
!x: es verdadero si x es falso.

Expresiones de Asignación

Una expresión de asignación es una expresión que almacena un nuevo valor en una variable. Por ejemplo, para asignar un valor 1 a la variable z se utiliza la siguiente expresión **z = 1 ENTER**.

Luego de que esta expresión es ejecutada la variable z tiene un valor 1, sin importar el valor que tenía z antes. Así el signo igual (=) se conoce como operador de asignación (y suele leerse como "se vuelve").

Es importante notar que las variables pueden cambiar de tipo durante la ejecución de un programa, en un momento ser un valor numérico y luego ser una matriz. Esta técnica si bien es válida no es recomendada.

Asignar un escalar a un conjunto de índices de una matriz hará que los elementos de la matriz se vuelvan el valor escalar. Así, **a(:, 2) = 5 ENTER**, hará que todos los elementos de la segunda columna de la matriz a sean 5 [13].

Operadores de Incremento

Los operadores de incremento incrementan ó decrementan el valor de una variable en 1. El operador de incremento se escribe como dos signos más seguidos sin espacio entre ellos (++) y el de decremento como dos signos menos seguidos sin espacio entre ellos(--). La forma de utilizarlos será antes o después de variables que ya se hayan definido, por ejemplo, **entradas++**, **y--**, **--altura**, **++índice** [13].

Así, **++x** puede escribirse también como **x=x+1**. Que se lee como **x se vuelve el valor que tiene la variable x más uno.**

Sentencias

Una sentencia es una expresión simple o un conjunto de sentencias de control de flujo anidadas. Ejemplos de sentencias simples son [13]:

a=1; ENTER
b=32; ENTER
c= a+b; ENTER
d= sqrt(c + (a*b)) ENTER

Sentencias de Control de Flujo

La sentencia **IF**

La sentencia **if** (si condicional) es la sentencia que permite tomar una decisión en Octave. Existen tres formas básicas, la más sencilla es [13]:

if (<condición>)
{parte-entonces}
endif

Donde *<condición>* es una expresión que controla que lo que se va a hacer. Si *<condición>* es verdadera, las sentencias de la *{parte-entonces}* se ejecutarán. Si *<condición>* es falsa no se ejecutarán las sentencias de la parte entonces. Por ejemplo, en el siguiente fragmento de código, si x es menor que dos, se sumará uno a x.

```
if (x< 2) ENTER
  x++; ENTER
endif ENTER
```

La segunda forma del **if** es:
if (<condición>)
{parte-entonces}
else
{parte-no-entonces}
endif

Si *<condición>* es verdadera se ejecutan las sentencias de la *{parte-entonces}*, si *<condición>* es falsa se ejecutan las sentencias de la *{parte-no-entonces}*. Así, por ejemplo:

```
if (rem (x, 2) == 0)
  x++;
else
  x--;
endif
```

Ese fragmento hace que si el resto de dividir x por 2 (que se escribe rem(x,2)) es igual a cero (==0) se sumará 1 a x (x++;) y sino se le restará 1 a x (x--;)

La tercera forma del **if** es la siguiente:

if (<condición>)
{parte-entonces}
elseif (<condición>)
{parte-entonces-elseif}
elseif (<condición>)
{parte-entonces-elseif}
elseif (<condición>)

{parte-entonces-elseif}
else
{parte-no-entonces}
endif

Donde cada *<condición>* se chequea y cuando una *<condición>* es verdadera se ejecuta su correspondiente *{parte entonces}*. Si ninguna es verdadera la última *{parte-no-entonces}* será ejecutada. Puede haber muchas partes **elseif**.

```
if (rem (x, 2) == 0)
  x++;
elseif (rem (x, 3) == 0)
  x = x+5;
else
  x--;
endif
```

En este caso si x es divisible por 2 se le sumará uno a su valor, si es divisible por tres se le sumará cinco a su valor y si no, se le restará 1.

La sentencia **WHILE**

En programación un loop significa que una parte de un programa se ejecutará más de una vez en forma sucesiva. La sentencia **while** es la forma más sencilla de hacer loops en Octave. Lo que hace un while es repetir un grupo de sentencias, mientras que una condición determinada sea verdadera [13].

while (condición)
cuerpo
endwhile

Lo primero que sucede es la verificación la *condición*. Si la *condición* es verdadera se ejecuta el *cuerpo*. Una vez ejecutado el *cuerpo*, se vuelve a verificar la *condición* y mientras siga siendo verdadera se vuelve a ejecutar el *cuerpo* y haste que la *condición* no es verdadera. Si la *condición* es inicialmente falsa el *cuerpo* no se ejecuta nunca. El siguiente bloque de código genera los 10 primeros números de la sucesión de fibbonacci.

```
fib = ones (1, 10);
i = 3;
while (i <= 10)
  fib (i) = fib (i-1) + fib (i-2);
  i++;
endwhile
```

La sentencia **FOR**

El **for** es más conveniente cuando queremos contar la cantidad de veces que iteramos en el loop. La estructura de un for es como sigue:

for nombre-variable = expresión
  cuerpo
endfor

La parte *nombre-variable* puede tener varias formas. La forma más común es el nombre de una variable simple. El operador de asignación (=) que figura en el for funciona de la siguiente manera. En vez de asignar la *expresión* a la *nombre-variable* de una vez, trabaja por columnas. Si la *expresión* es un rango, un vector o un escalar, el valor de *nombre-variable* será asignado cada vez que el cuerpo se ejecute. Como puede verse en el siguiente código que genera los 10 primeros números de la sucesión de fibonacci.

```
fib = ones (1, 10);
for i = 3:10
  fib (i) = fib (i-1) + fib (i-2);
endfor
```

El rango 3:10 se va asignando a la variable i, hasta que no quedan valores.
Cualquier sentencia for se puede traducir como while y viceversa.

La sentencia **BREAK**

La sentencia **break** hace que la ejecución "salte" fuera del ciclo while o for que la contiene [13].

```
while (div*div <= num)
  if (rem (num, div) == 0)
    break;
  endif
  div++;
endwhile
```

La sentencia **CONTINUE**

La sentencia **continue** saltea el resto del ciclo que la incluye y empieza con el siguiente ciclo de iteración inmediatamente [13].

```
vec = round (rand (1, 10) * 100);
i = 0;
for x = vec
  if (rem (x, 2) != 0)
    continue;
  endif
  i++
```

endfor

La sentencia **INPUT**

La sentencia **input** permite el ingreso de un valor numérico.

input("Ingrese un valor ");

Si se agrega el argumento "s" el valor ingresado se considera una cadena de caracteres.

La sentencia **DISP**

La sentencia **disp** permite mostrar en pantalla el argumento que utiliza.

disp("Ingrese un valor ");

Archivos de Funciones

Los programas complicados en Octave pueden ser simplificados definiendo funciones. Las funciones pueden ser directamente definidas en la línea de comando o en archivos externos y pueden ser llamadas como funciones predefinidas [13].

Definiendo funciones

La forma más simple de definición de una función es la siguiente:
function nombre
  cuerpo
endfunction

Un *nombre* válido de función debe obedecer a las mismas reglas que un nombre válido de una variable. El *cuerpo* de la función consiste en un conjunto de sentencias Octave. En el cuerpo de la función se define que es lo que la función debe hacer. Normalmente, será necesario pasarle alguna información a la función que definimos. La estructura general para pasar parámetros a una función en Octave es:

function nombre (arg-list)
  cuerpo
endfunction

donde *arg-list* es una lista de argumentos separados por comas. Cuando la función es llamada los nombre de los argumentos son utilizados para contener los valores de los argumentos durante esa llamada.

Devolución de valores desde una función

En la mayoría de los casos también se desea que alguna información regrese desde las funciones que se definan. La forma general de hacer esto se logra:

```
function ret-val = name (arg-list)
  cuerpo
endfunction
```

Donde *ret-val* es el nombre de la variable que contendrá el valor que la función retornará. Esta variable debe estar definida antes del fin del cuerpo de la función.
Las variables usadas en el cuerpo de la función son locales a la función (no existen fuera de ella). Las variables nombradas en la *arg-list* y *ret-val* también son locales a la función.
Por ejemplo, esta sería una función que computa el valor promedio de una serie de valores almacenados en un vector [13].

```
function retval = avg (v)
  retval = sum (v) / length (v);
endfunction
```

Retorno de Múltiple Valores desde una función

A diferencia de muchos otros lenguajes, Octave permite que una función retorne más de un valor. La sintaxis para definir funciones que devuelvan múltiples valores será [13]:

```
function [ret-list] = nombre (arg-list)
  cuerpo
endfunction
```

donde *ret-list* será una lista de variables separadas por comas que guardarán los valores devueltos por la función. Por ejemplo:

```
function [max, idx] = vmax (v)
  idx = 1;
  max = v (idx);
  for i = 2:length (v)
    if (v (i) > max)
      max = v (i);
      idx = i;
    endif
  endfor
endfunction
```

Retornando desde una función

El cuerpo de una función puede contener una sentencia **return**. Esta sentencia hace que la ejecución vuelva al resto del programa. La sintaxis es [13]:

return

El comando return simplemente permite salir fácilmente del cuerpo de una función.

Archivos de Funciones (.m)

Excepto para programas muy sencillos no es práctico tener que definir todas las funciones que necesitamos utilizar cada vez que las necesitemos. En lugar de ello, la idea es guardar las funciones en un archivo que sea fácil de utilizar, ya sea para agregar funciones nuevas o modificar las que ya hemos creado y luego utilizarlas desde Octave.

Octave no requiere que las funciones sean cargadas desde un archivo para utilizarlas, simplemente hay que poner las definiciones de las funciones en algún sitio donde Octave pueda encontrarlas.

Cuando Octave encuentra un identificador que no ha sido definido, primero verifica que no sea una variable o función que ya haya sido usada. Si no la encuentra allí, busca en la lista de directorios que se encuentra almacenada en la variable predefinida **LOADPATH**, buscando archivos **".m"** que tengan el mismo nombre que el identificador. Así por ejemplo si Octave encuentra el identificador **"promedio"** en un programa y no es una variable, buscará por el archivo **promedio.m** en la lista de directorios definidos en LOADPATH. Una vez que Octave encuentra el archivo, el contenido del mismo es leído. Si se define sólo una función dentro de ese archivo esta función se ejecuta. El lugar por defecto en donde OCTAVE espera que se encuentren los archivos es **<directorio de instalación>/octave_files**, entonces por ejemplo si instalamos Octave con las opciones por defecto este lugar sería **"c:/Archivos de programas/GNU Octave 2.1.36/octave_files"** [13].

Instalación y Carga de Paquetes en OCTAVE:

Como en todo lenguaje de "alto nivel", el programa OCTAVE cuenta con una gran cantidad de herramientas ó paquetes que agregan "extra funcionalidades" al software y ayudan al programar en el desarrollo de sus algoritmos, por nombrar algunos, se puede encontrar [13]:

Fuzzy-logic-toolkit: Herramienta para algoritmos de lógica difusa

Gnuplot: Herramienta para graficación

Geometry:

Video: Herramienta para procesamiento de video

Audio: Herramienta para procesamiento de audio

Civil-engineering

Octave por "default", cuenta con una lista de paquetes ya instalados, para observar cuales son estos, solo basta con escribir el comando "pkg list" y se desplegara automáticamente la lista de todos ellos, como se muestra en la Figura 93.

```
octave:2> pkg list
Package Name          | Version | Installation directory
----------------------+---------+-----------------------
          actuarial   |   1.1.0 | /usr/share/octave/packages/actuarial-1.1.0
              audio   |   1.1.4 | /usr/share/octave/packages/audio-1.1.4
          benchmark   |   1.1.1 | /usr/share/octave/packages/benchmark-1.1.1
                bim   |   1.1.1 | /usr/share/octave/packages/bim-1.1.1
            bioinfo   |   0.1.2 | /usr/share/octave/packages/bioinfo-0.1.2
                cgi   |   0.1.0 | /usr/share/octave/packages/cgi-0.1.0
  civil-engineering   |   1.0.7 | /usr/share/octave/packages/civil-engineering-1.0.7
       combinatorics  |   1.0.9 | /usr/share/octave/packages/combinatorics-1.0.9
      communications  |   1.1.1 | /usr/share/octave/packages/communications-1.1.1
             control  |   2.4.2 | /usr/share/octave/packages/control-2.4.2
      data-smoothing  |   1.3.0 | /usr/share/octave/packages/data-smoothing-1.3.0
           dataframe  |   0.9.1 | /usr/share/octave/packages/dataframe-0.9.1
        econometrics  |   1.1.1 | /usr/share/octave/packages/econometrics-1.1.1
              engine  |   1.0.9 | /usr/share/octave/packages/engine-1.0.9
           financial  |   0.4.0 | /usr/share/octave/packages/financial-0.4.0
             fl-core  |   1.0.0 | /usr/share/octave/packages/fl-core-1.0.0
                fpl   |   1.3.3 | /usr/share/octave/packages/fpl-1.3.3
 fuzzy-logic-toolkit  |   0.4.2 | /usr/share/octave/packages/fuzzy-logic-toolkit-0.4.2
                ga    |  0.10.0 | /usr/share/octave/packages/ga-0.10.0
           general    |   1.3.2 | /usr/share/octave/packages/general-1.3.2
      generate_html   |   0.1.5 | /usr/share/octave/packages/generate_html-0.1.5
          geometry    |   1.6.0 | /usr/share/octave/packages/geometry-1.6.0
           gnuplot    |   1.0.1 | /usr/share/octave/packages/gnuplot-1.0.1
               gsl    |   1.0.8 | /usr/share/octave/packages/gsl-1.0.8
             ident    |   1.0.7 | /usr/share/octave/packages/ident-1.0.7
             image    |   2.0.0 | /usr/share/octave/packages/image-2.0.0
 informationtheory    |   0.1.8 | /usr/share/octave/packages/informationtheory-0.1.8
 instrument-control   |   0.2.0 | /home/Salvador García/octave/instrument-control-0.2.0
       integration    |   1.0.7 | /usr/share/octave/packages/integration-1.0.7
                io    |   1.2.0 | /usr/share/octave/packages/io-1.2.0
              irsa    |   1.0.7 | /usr/share/octave/packages/irsa-1.0.7
    linear-algebra    |   2.2.0 | /usr/share/octave/packages/linear-algebra-2.2.0
              lssa    |   0.1.2 | /usr/share/octave/packages/lssa-0.1.2
           mapping    |   1.0.7 | /usr/share/octave/packages/mapping-1.0.7
         mechanics    |   1.3.1 | /usr/share/octave/packages/mechanics-1.3.1
     miscellaneous    |   1.2.0 | /usr/share/octave/packages/miscellaneous-1.2.0
```

Figura 93. Lista de paquetes instalados en Octave.

Hay dos formas de instalar un nuevo paquete en octave, esta la forma manual y vía internet.

Para la primera forma, Suponiendo que un paquete está disponible en el archivo llamado "imagen 1.0.0.tar.gz" y este mismo se encuentra en el directorio raíz de octave, lo único que hay que hacer para instalarlo, es escribir el comando [13]:

pkg install image-1.0.0.tar.gz

Si el paquete se ha instalado correctamente no se imprimirá en el sistema, pero si se produce un error durante la instalación, se informara. Si hay una versión diferente del

paquete y ya está instalado será eliminado antes de instalar el nuevo paquete. Esto hace que sea fácil de actualizar la versión de un paquete.

Para verificar que el paquete se ha instalado correctamente escribimos nuevamente, "pkg list" y se podrá observar la lista actualizada como se muestra en la Figura 94 [13].

```
pkg list
-| Package Name  | Version | Installation directory
-| --------------+---------+-----------------------
-|        image *|   1.0.0 | /home/jwe/octave/image-1.0.0
```

Figura 94. Comprobación de paquetes en Octave.

La segunda forma de instalación es vía internet, y solo es necesario el siguiente comando [13]:

pkg install -forge *nombre_del_paquete*

Si la instalación se ha efectuado correctamente, se obtendrá la misma respuesta por parte de octave que si se hubiera llevado a cabo manualmente.

Además, es posible eliminar un paquete del sistema mediante el siguiente comando de desinstalación [13]:

pkg uninstall nombre_del_paquete

Si el paquete se elimina correctamente no será impreso en el sistema, pero si se produce un error, se informará

Para cargar un paquete y utilizar las funciones proporcionadas por este mismo, es necesario el comando [13]:

pkg load nombre_del_paquete

También es posible cargar todos los paquetes instalados a la vez con la palabra clave 'all'. Como se muestra a continuación [13].

pkg load all

Edición y ejecución de un archivo.m en OCTAVE.

Para finalizar este anexo, se pondrá en práctica lo estudiado anteriormente con un pequeño ejemplo.

Para este ejercicio se desea crear un "script" que grafique una señal sinusoidal, con un periodo de tiempo que vaya desde -10 a 10 con intervalos de 0.1. A continuación se muestran los pasos.

1. Ejecutar OCTAVE como se ha visto anteriormente.

2. Abrir un archivo de texto en "Bloc de Notas", guardar el documento con el nombre "sinu" (para este ejemplo), en el directorio raiz del programa OCTAVE y cambiar su extensión de ".txt" a ".m" como se muestra en la Figura 95.

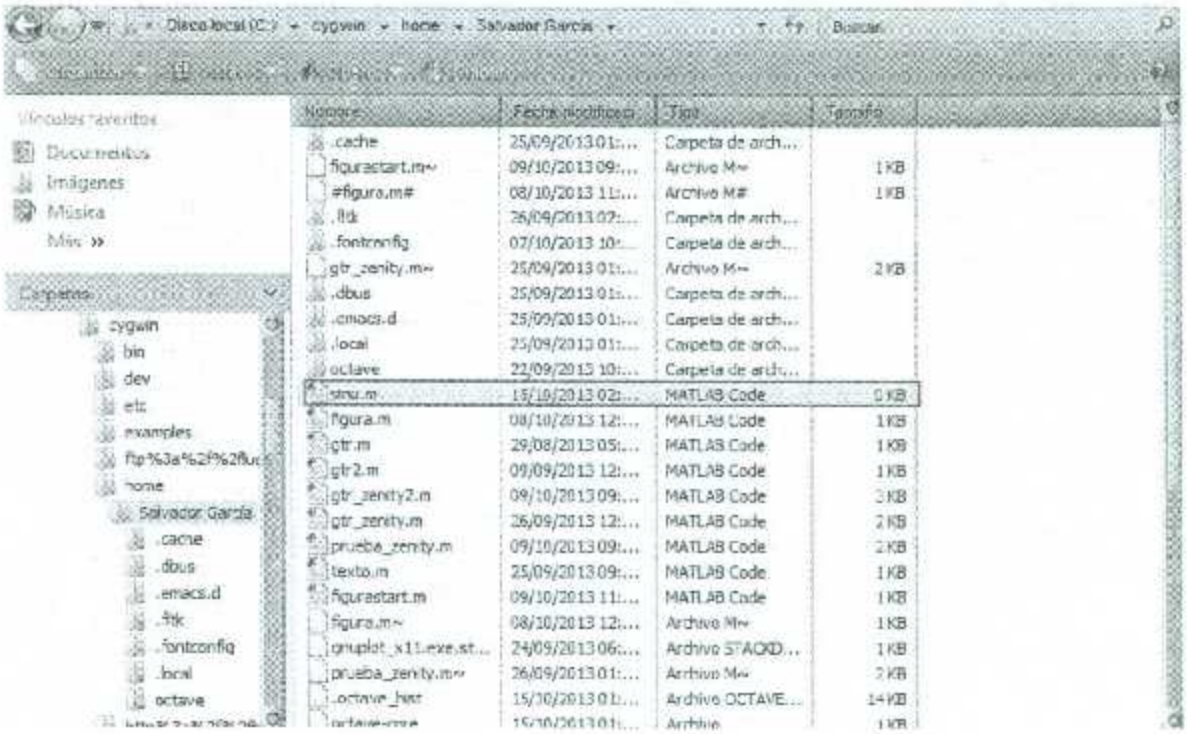


Figura 95. Cambio de extensión de archivo .txt a .m.

Nota: Si no conoce el directorio raíz de OCTAVE, en la pantalla de comandos escribir: *"pwd"* y automáticamente se desplegara la dirección del programa.

3. En la ventana de comandos escribir "edit sinu" y con esto se abrirá la ventana de edición de archivos Cygwin/X X para OCTAVE, como se muestra en la Figura 96.

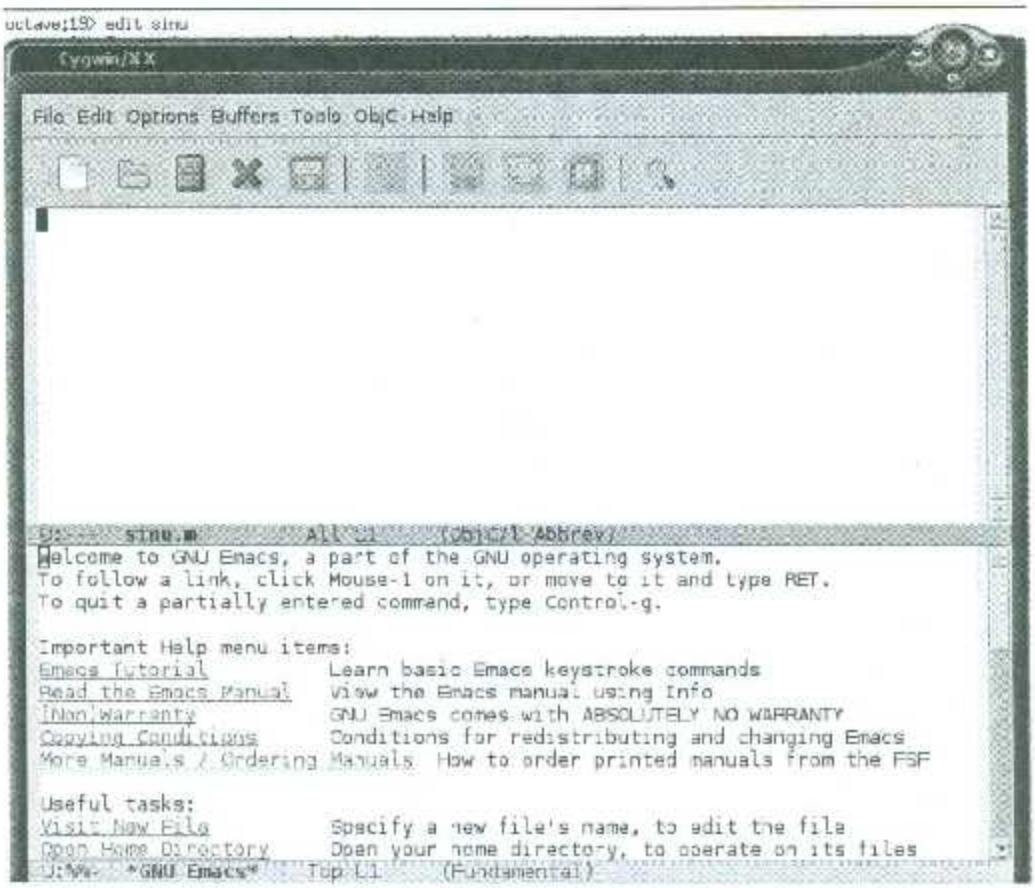


Figura 96. Ventana de edición de archivos en Octave.

4. Ahora que se ha creado el archivo de edición exitosamente, solo resta escribir el programa. Primero se cargan las librerías ó paquetes a utilizar, para este ejemplo, se requiere de herramientas de graficación, es por esto que se agregan los paquetes "GNUPLOT" y "PLOT". Enseguida se crea una variable, en este caso se llamara "X" y será de la siguiente manera:

   X=-10:0.1:10

   Esto le indica al programa que la variable "X" es un vector que va desde -10 a 10 con incrementos de 0.1.

   Hecho lo anterior, se requiere de otra variable contra la cual se graficaran los valores del vector anterior, para el ejemplo se pide que sea la función seno, esto quedaría de la siguiente forma:

   Y=SIN(X)

Por último, se utiliza la función "plot" de la siguiente manera

PLOT(X,Y)

Todo este procedimiento se puede observar en la Figura 97.

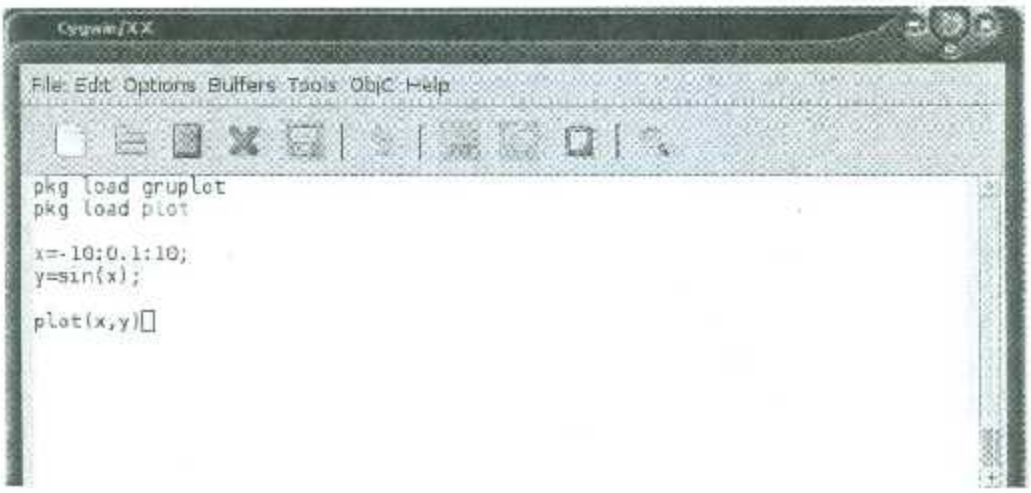


Figura 97. Código visto en ventana de edición.

5. Finalmente, se guarda el "script" creado anteriormente y en la ventana de comando se escribe el nombre del mismo archivo, en este caso, "sinu". Si todo se ha realizado correctamente, se obtendrá los resultados de la Figura 98.

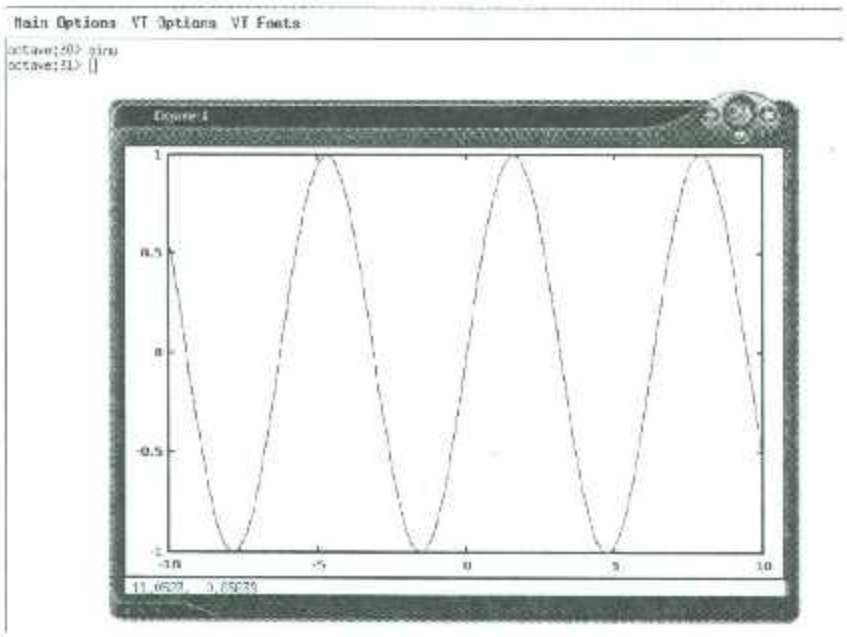


Figura 98. Resultados obtenidos del script creado.

ANEXO C

Creación de Interfaces Gráficas en MATLAB y OCTAVE.

Debido a que la programación de MATLAB y OCTAVE es práctimicante similar y además, hay una mayor cantidad de información y ayuda para el primero, se omitirá la parte de conceptos básicos para MATLAB y en su lugar se tratara el tema de Interfaces Gráficas (GUI), ya que es la parte en la que más difieren estos dos programas.

Una interfase es una de las partes más importantes de cualquier programa puesto que determinan que tan factible y preciso será el desempeño del programa ante los comandos que el usuario pretenda ejecutar. Aunque un programa sea muy poderoso, si se manipula por medio de una interface pobremente elaborada, tendrá poco valor para un usuario inexperto. Es por esto que las interfaces gráficas tienen una gran importancia para usuarios inexpertos o avanzados de cualquier programa ya que facilita su uso.

Creando una GUI en MATLAB

MATLAB nos permite realizar GUIs de una manera sencilla usando una herramienta llamada GUIDE (GUI Development Environment).

La forma de implementar las GUI con MATLAB es crear los objetos y definir las acciones que cada uno va realizar. Al usar GUIDE para crear una GUI obtendremos dos archivos: un archivo FIG que contiene la descripción de los componentes que contiene la interface y un archivo M que contiene las funciones y los controles del GUI asi como el callback.

Un callback se define como la acción que lleva a cabo un objeto de la GUI cuando el usuario lo active. Para ejemplificarlo, suponga que en una ventana existe un botón que al presionarlo ejecutará una serie de instrucciones, a ese conjunto de instrucciones se le conoce como la función del callback.

Iniciando GUIDE

Para empezar a crear una GUI en MATLAB usamos GUIDE, ya sea que escribamos guide en la ventana de comandos de MATLAB o lo ejecutamos desde el menú principal FILE → New → GUI como se muestra en la Figura 99 [14].

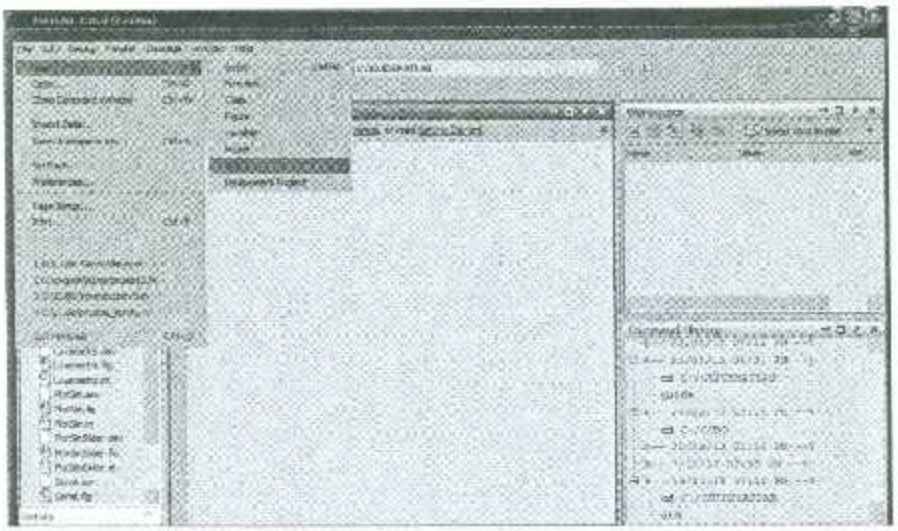


Figura 99. Creación de una GUI en MATLAB.

Una vez hecho lo anterior MATLAB se mostrará una ventana con opciones para la creación de la GUI o si se desea abrir una existente. Si se selecciona "nueva GUI", entonces el área de la figura aparecerá mostrando en la parte superior los menús y opciones de GUIDE. En la parte izquierda se aprecian los diferentes controles y en la parte central el área de diseño donde pondremos controles a usar. (Se puede hacer que los controles aparezcan con su nombre mediante el menú File→Preferences...→Guide y seleccionando luego Show names in component palette). Todo esto se puede observar en la Figura 100 [14].

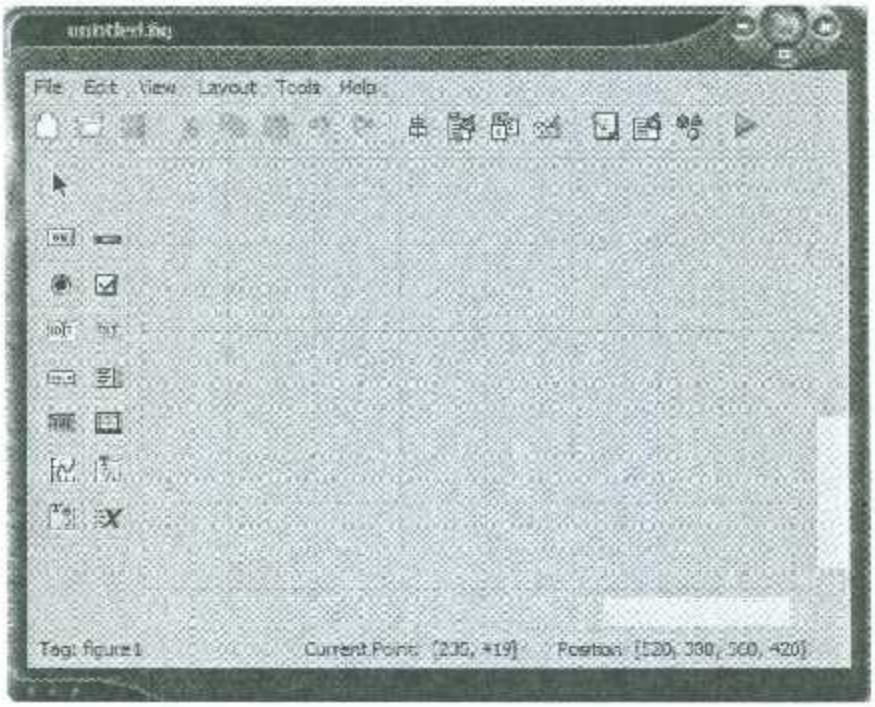


Figura 100. Ventana para la creación de una GUI.

Partes de GUIDE

Se mencionara las partes más importantes de GUIDE que servirán para realizar un ejemplo representativo. La Figura 101 muestra la lista de los controles con sus nombres. Cada botón cuenta con una serie de propiedades que explicaremos más adelante. En la parte superior de la Figura 100 se encuentra la barra de herramientas que contiene entre otras las siguientes: Alineación de objetos, Editor de menú, Editor del orden del tabulador, Editor de archivo M, Inspector de propiedades, Navegador de objetos y Activación de GUI. Se describirá brevemente algunas de ellas que son necesarias para empezar a generar GUIs [14].

Inspector de propiedades. Cada control cuenta con diferentes propiedades y es aquí donde podremos cambiar: el color, el nombre, el tag, el valor, el callback, entre otros.

Activación de Figura. Una vez que se haya terminado de diseñar, presionamos este botón para activar la figura y poder probar la GUI.

Botón de presión (Push Button). Crea un botón.

Botón de selección (Radio Button). Crea un botón circular.

Área de texto (Edit Text). Crea un campo para captura de texto.

Ejes (Axes). Crea un área para gráficas.

Marco (Frame). Crea un marco que puede contener otros controles.

Texto estático (Static Text). Crea un letrero fijo.

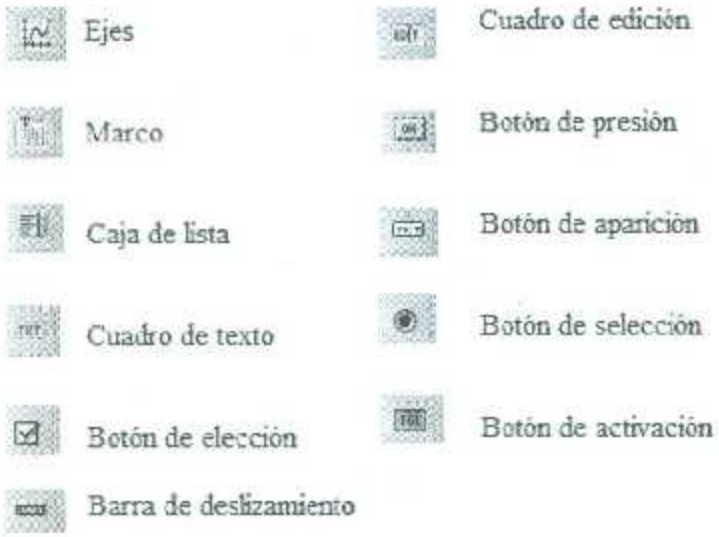


Figura 101. Lista de controles con sus nombres.

Propiedades de los controles.

Para entender las propiedades de un control primero se crea un botón y luego activamos el inspector de propiedades (véase Figura 102). Los distintos elementos se pueden crear en el área de diseño con solo seleccionarlos y arrastrarlos a donde se desea que queden colocados [14].

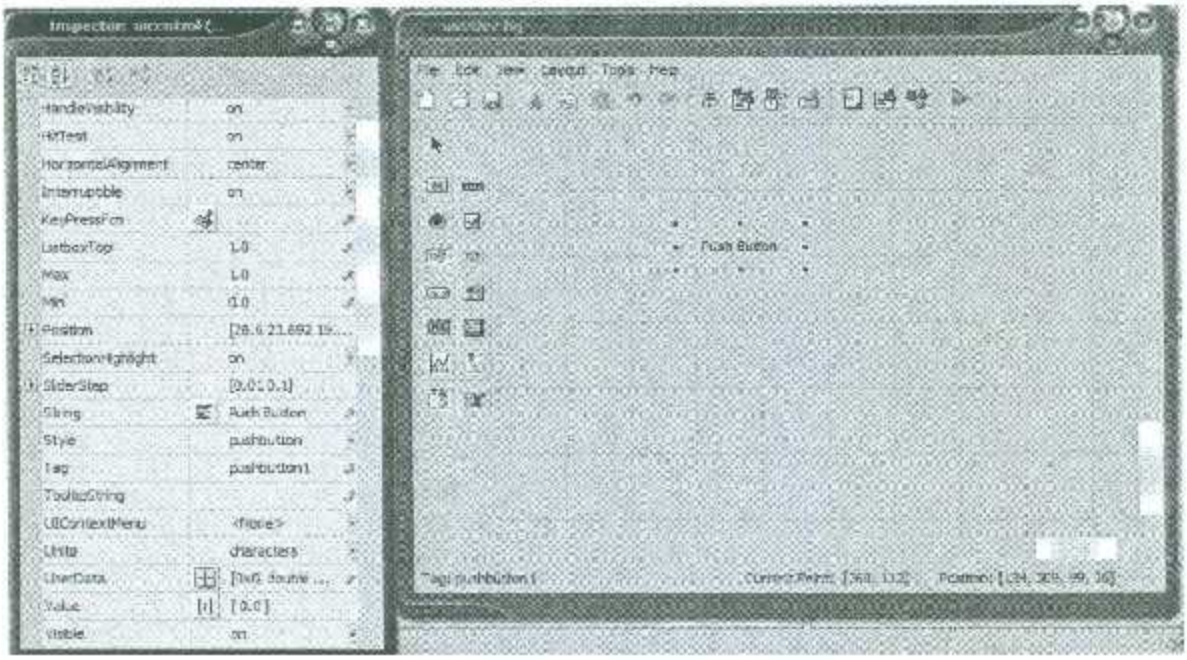


Figura 102. Inspector de propiedades.

Como se ha mencionado, las propiedades varian dependiendo del control a usar. A continuación se explican las más comunes:

BackgroundColor. Cambia el color del fondo del control

Callback. La propiedad más importante del control, ya que le dice al control que hacer cuando este se active

Enable. Activa o desactiva un control.

String. En el caso de botones, cajas de texto, texto estático; es el texto que se muestra en el control

Tag. Otra de las propiedades más importantes ya que con ésta es posible regresar datos o identificar al control.

Ejemplo (Conversor de temperaturas).

Con el propósito de dominar el tema anterior, se realizara un conversor de temperatura Celsius-Kelvin-Fahrenheit. Para empezar, se ejecuta Guide como se ha visto y colocamos cuatro StaticText como se muestra en la Figura 103, dándoles el tamaño indicado en la Figura.

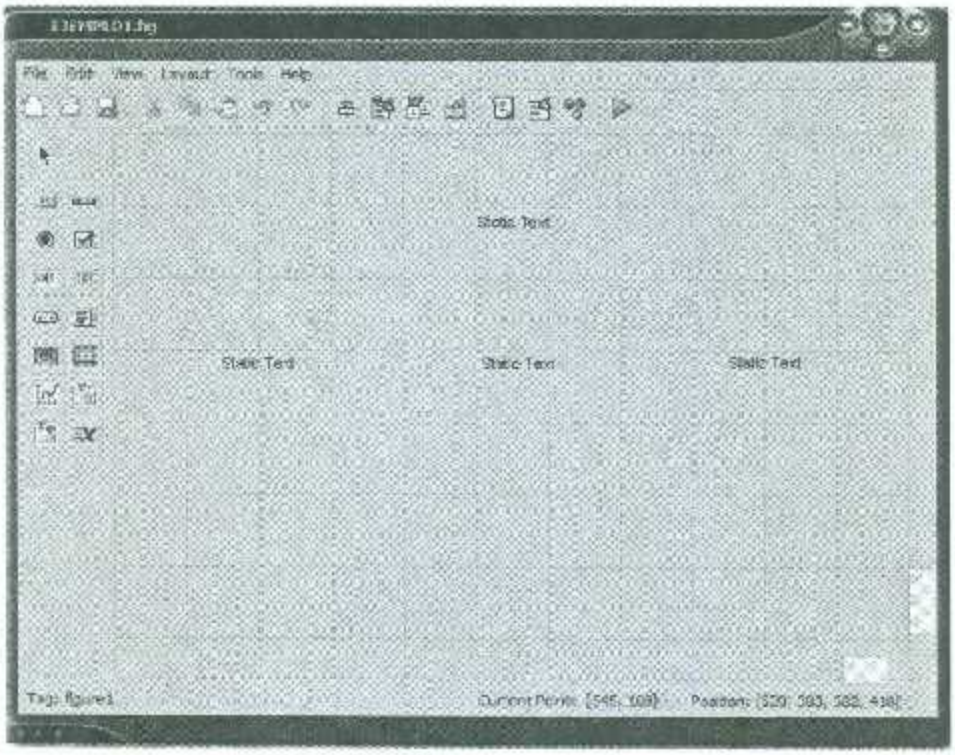


Figura 103. Colocación de StaticText en la GUIDE.

En el caso de cada una de ellas modificamos la propiedad **String** con el **Property Inspector** en el siguiente orden:

- Calculadora de Temperaturas
- Celsius
- Kelvin
- Fahrenheit

En la Figura 104 se puede apreciar el estado actual de la GUI, además de la ventana del Inspector de Propiedades con el cambio del último Static Text a Fahrenheit.

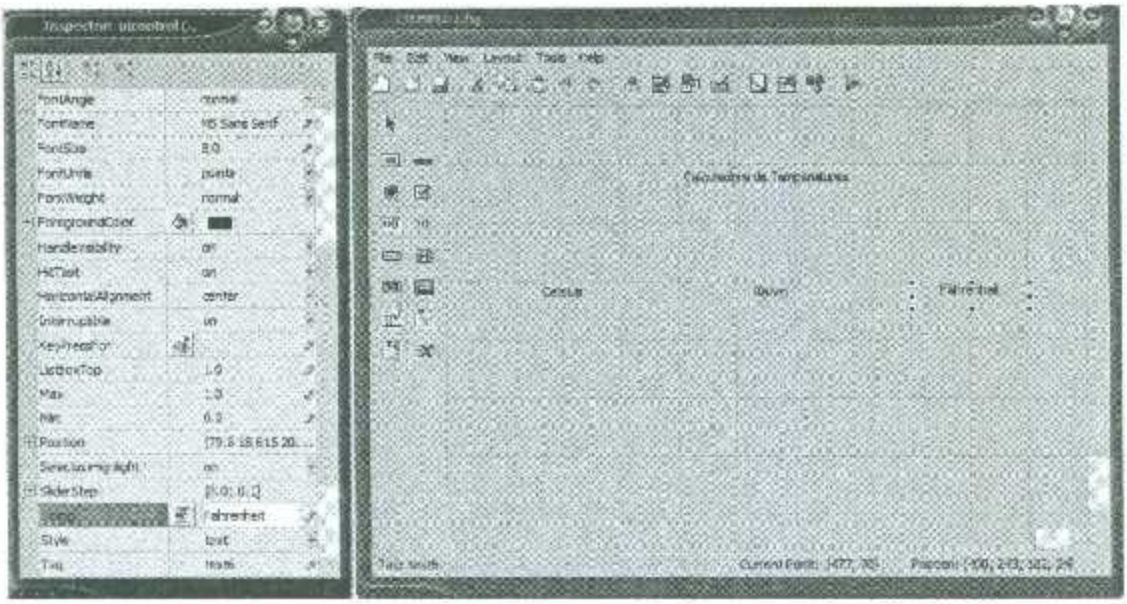


Figura 104. Cambio de nombre de Static Text.

Debajo de las etiquetas Celsius, Kelvin y Fahrenheit se agregan tres **Edit Text** y cuatro **Push Button** y se modifica la propiedad **String** para que los Edit Text queden en blanco, quitando todo texto de la propiedad String, y en los Push Button cambiamos la propiedad String a "Calcula" para los tres primeros y "Salir" para el último. Al mismo tiempo, se cambia la propiedad Tag de los Push Button a "Boton_Celsius", "Boton_Kelvin" y "Boton_Fahrenheit", respectivamente. Para el botón "Salir" le cambiamos su **Tag** a "Boton_Salir", para dejar ahora la interfase como se muestra en la Figura 105.

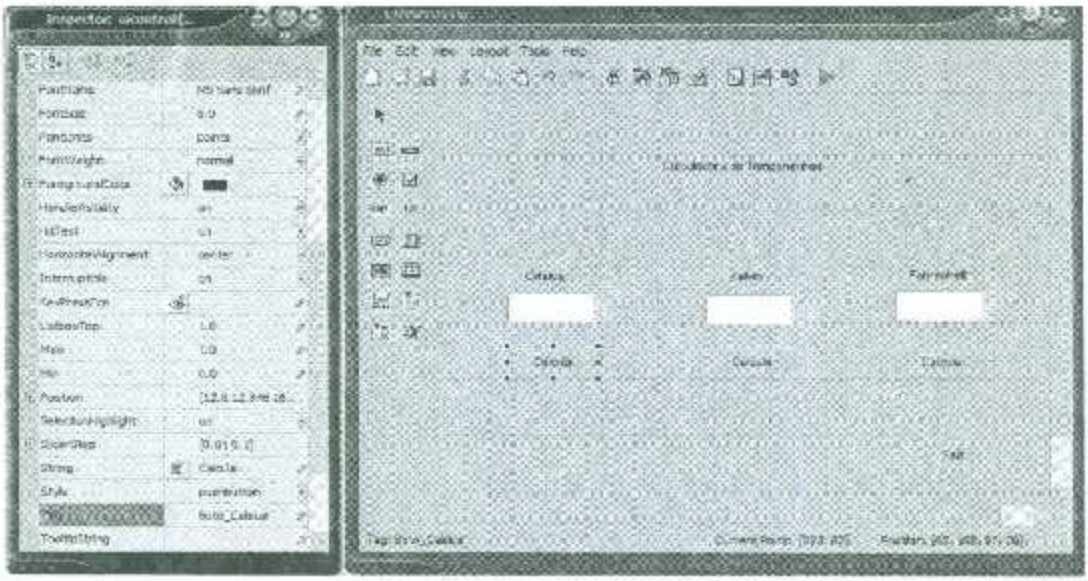


Figura 105. Cambio de propiedades Tag en los botones de la GUIDE.

Ahora es necesario definir las variables que se van a usar en el callback, y para ello se debe modificar en la propiedad Tag de los Edit Text de acuerdo con el siguiente orden.

- Celsius
- Kelvin
- Fahrenheit

Para realizar esto seleccionamos el Edit Text y modificamos la propiedad Tag con los datos anteriores para cada uno de los Edit Text. Esto se aprecia en la Figura 106 para Fahrenheit.

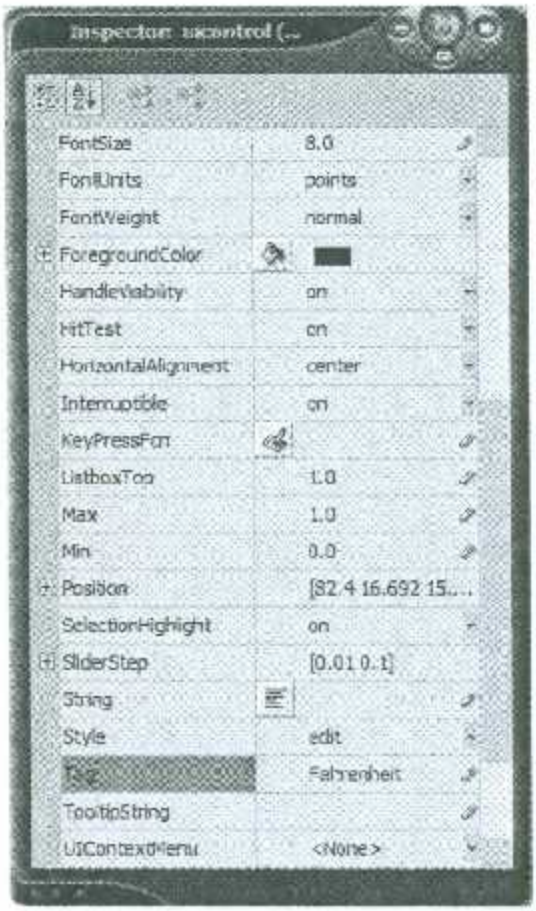


Figura 106. Inspector de propiedades de los Edit Text.

Una vez hecho lo anterior guardamos nuestra interfase con File→Save As... y ponemos el nombre de "EJEMPLO1"; esto nos abrirá una nueva ventana del editor de MATLAB en donde pondremos el código necesario. Ahora tenemos dos archivos: EJEMPLO1.fig y EJEMPLO1.m.

Primero se agrega el código al botón de Salir, y para ello se selecciona la opción callback del control a modificar (en este caso Salir) y se escribe "close(gcbf)". En la Figura 107 se observa el código agregado para el botón Salir.

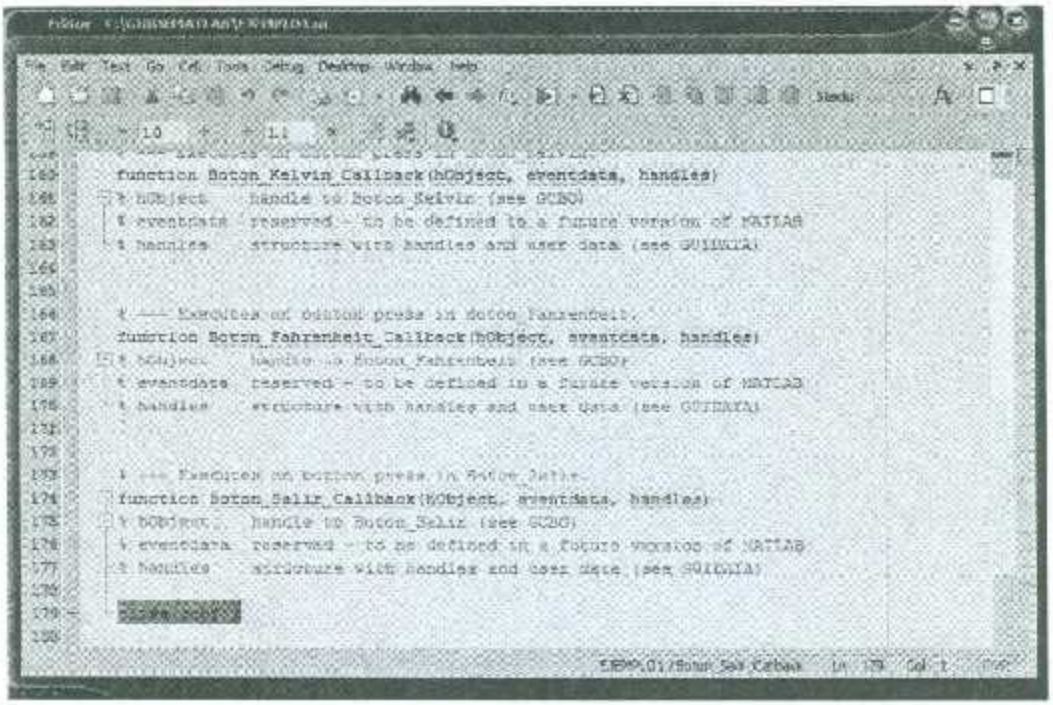


Figura 107. Código agregado para el botón Salir.

En el control de Calcula debajo de Celsius pondremos el código para convertir de Celsius a Kelvin y Fahrenheit, y para ello tenemos el siguiente algoritmo:

1. Se obtiene el dato de Celsius
2. Se realiza la conversión de Celsius a Kelvin
3. Se realiza la conversión de Celsius a Fahrenheit
4. Desplegar resultados

Basándose en el algoritmo, se debe recordar las formulas para realizar dichas conversiones:

Celsius a Kelvin.

$$K = (Celsius + 273.5)$$

Celsius a Fahrenheit.

$$F = (1.8 * Celsius) + 32$$

Tomando en cuenta lo anterior, en el editor de archivos-m se modifica el **callback** del botón "Calcula" debajo de Celsius que corresponde a Boton_Celsius y se escribe el código necesario. Para leer el dato se utiliza:

micelsius = eval (get (handles.Celsius, 'String'));

y para convertir de Celsius a Kelvin y Fahrenheit se utiliza:

mikelvin = micelsius +273.15;

mifahrenheit = 1.8 * micelsius + 32;

y para escribir en los Edit Text de Kelvin y Fahrenheit se usa:

set (handles.Kelvin, 'String' , mikelvin);

set (handles.Fahrenheit, 'String', mifahrenheit);

La Figura 108 muestra cómo queda el código del botón Calcula correspondiente a Celsius.

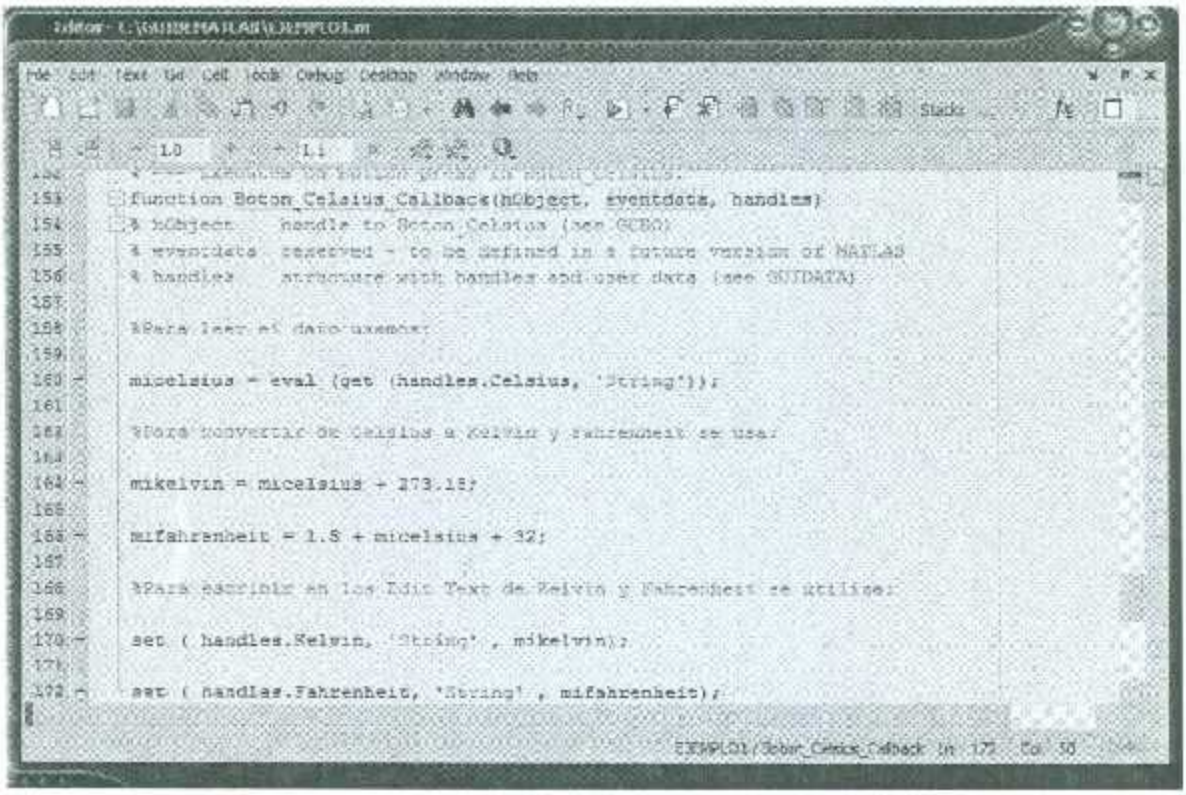


Figura 108. Código de botón "Calcula Celsius".

Como siguiente acción pondremos el código para convertir de Kelvin a Celsius y Fahrenheit el cual se observa en la Figura 109, el algoritmo es muy similar al anterior y las formulas usadas son:

Kelvin a Fahrenheit.

$$F = \frac{K - 273.15}{5} * 9 + 32$$

Kelvin a Celsius.

$$C = K - 273.15$$

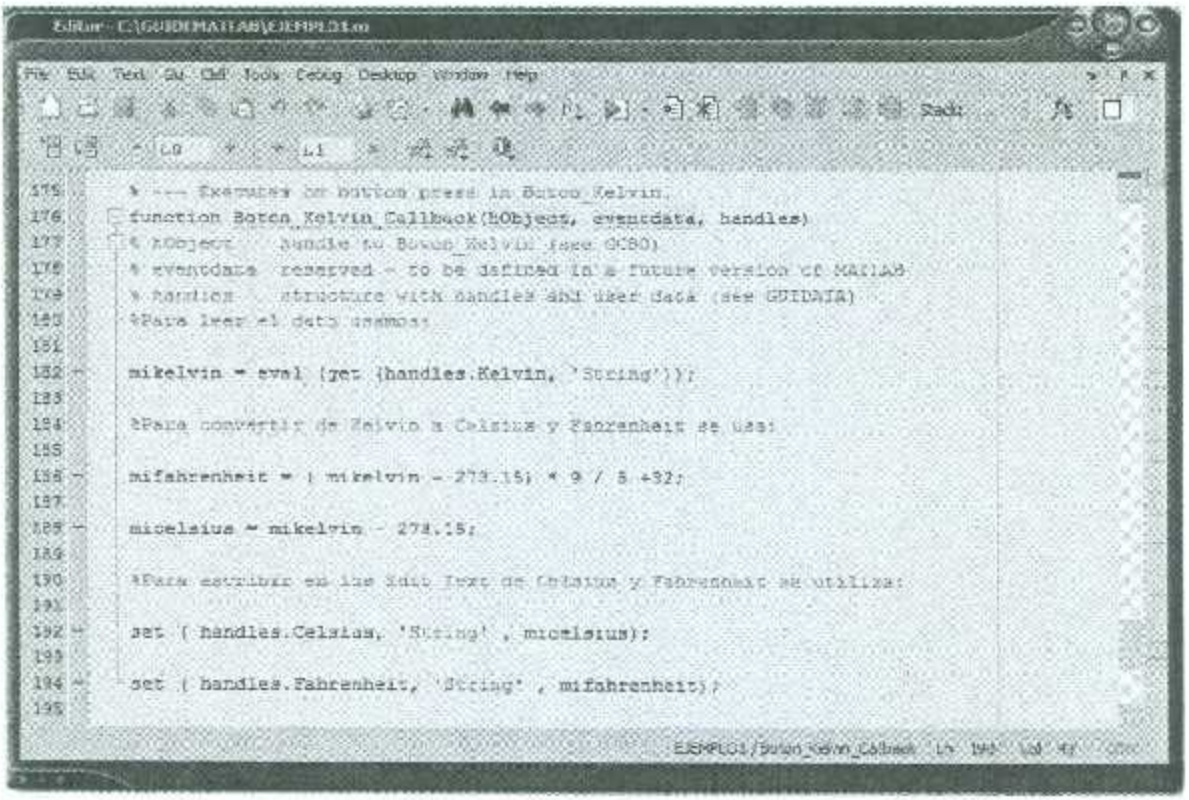


Figura 109. Código de botón "Calcula Kelvin".

Finalmente pondremos el código para convertir de Fahrenheit a Celsius y Kelvin, el cual se observa en la Figura 110. El algoritmo es muy similar al anterior y las formulas usadas son:

Fahrenheit a Celsius.

$$C = (F - 32) * 0.555$$

Fahrenheit a Kelvin.

$$K = \frac{F - 32}{9} * 5 + 273.15$$

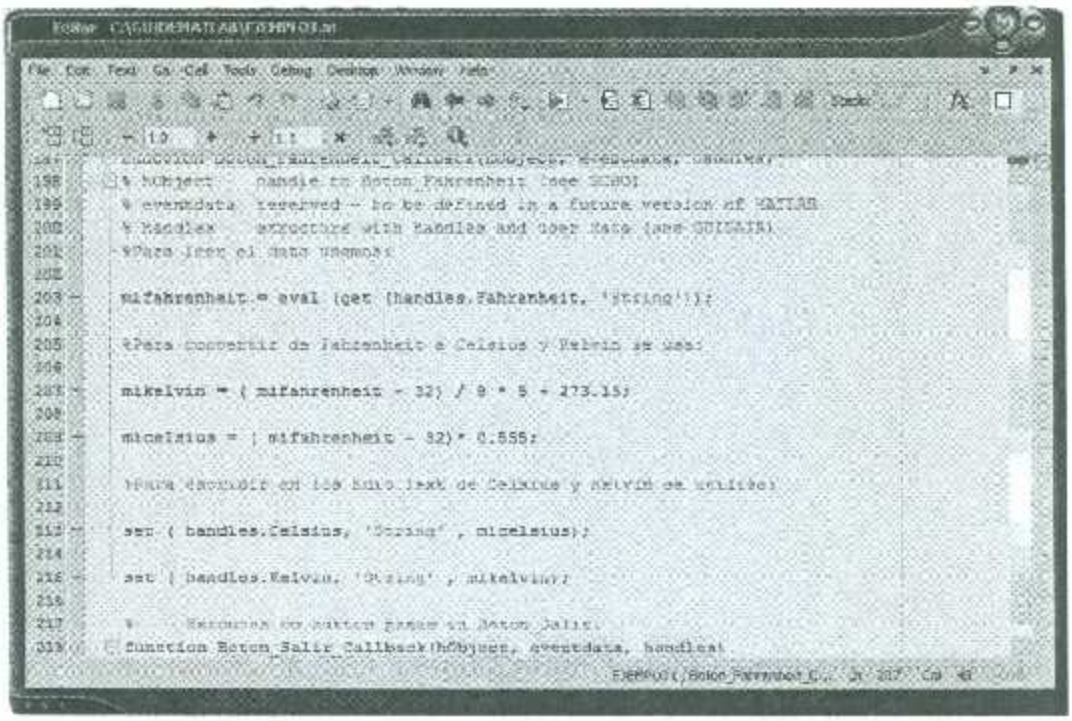


Figura 110. Código de botón "Calcula Fahrenheit".

Ahora se puede ejecutar escribiendo en la ventana de comandos "EJEMPLO1" y se obtendrá la salida de la Figura 111, en donde se muestra el programa ejecutándose.

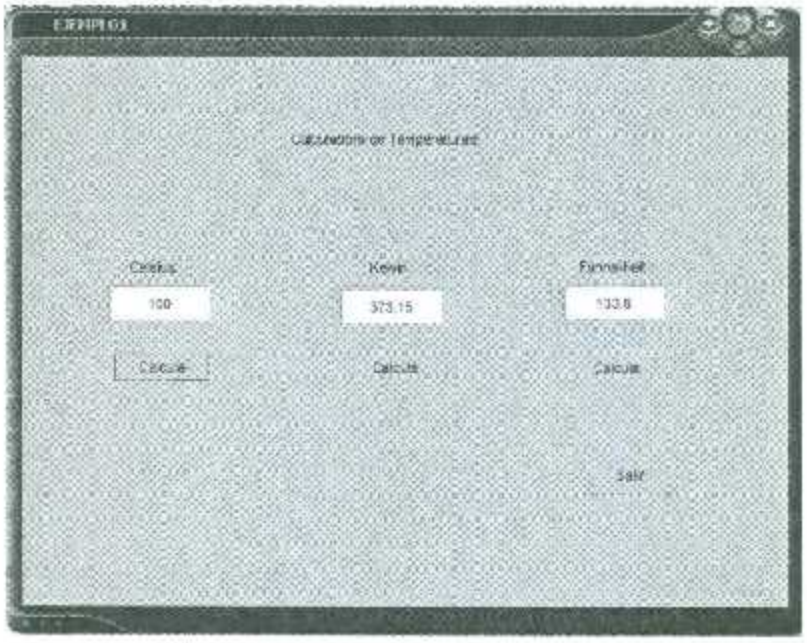


Figura 111. Interfaz gráfica creada exitosamente.

Creando una GUI en OCTAVE

Después de haber trabajado con Octave durante un tiempo, se puede observar que carece de una herramienta oficial para crear GUIs, por otra parte, todas las aplicaciones con este fin aún se encuentran en fase "beta" como lo es Qt-Designer, estas poseen muchos errores y no pueden utilizarse en todas las versiones de Octave. Hasta el momento, la única alternativa para crear una "pseudo interfaz gráfica" en Octave es con el paquete Zenity.

Instalación de Zenity para OCTAVE en Windows.

1. Como Zenity es una herramienta de Linux, se necesita descargar con compatibilidad para el sistema Windows, la página donde se puede encontrar es la siguiente: http://www.placella.com/software/zenity/ .Ya que se ha descargado, se instala como cualquier otro programa de Windows y se coloca en el directorio donde está instalado Cygwin y Octave [15].

2. Abrir Octave y escribir el código de instalación pkg *install forge zenity-0.5.7.tar.gz*, el proceso de instalación de paquete se puede observar en el Anexo B

3. Para ver si el paquete se instaló correctamente, y ver que otro paquetes están instalados, introducir el comando *pkg list.* En la lista aparecerá todos los paquetes instalados, y con un asterisco a un lado los paquetes que están cargados (véase Anexo B).

4. Con el fin de utilizar Zenity para Octave, cargar el paquete con el comando *pkg load zenity* (véase Anexo B).

Ahora se está listo para utilizar Zenity para Octave en Windows.

Uso general y funciones en Zenity (Véase Tabla 4) [15].

Tabla 4. Uso general y funciones en Zenity.

Función	Definición
zenity_calendar(titulo, día, mes, año)	Despliega un calendario, los argumentos opcionales día, mes y año devuelven la fecha seleccionada
zenity_entry(texto, entrada_texto, contraseña)	Despliega un cuadro de dialogo de introducción de texto. La variable texto

	coloca el titulo del cuadro de dialogo, la variable "entrada de texto" coloca el texto por default del campo de entrada de texto. Si la variable contraseña no está vacía, el valor del campo de entrada de texto no aparecerá en la pantalla. Todos los argumentos son opcionales. Devuelve el valor escrito.
zenity_message(texto, tipo)	Despliega un mensaje gráfico de dialogo. La variable texto coloca el mensaje del cuadro de dialogo, y la variable opcional tipo coloca el tipo de mensaje. El "tipo" puede ser uno de los siguientes: error, información, pregunta y advertencia. El valor por default es información. El cuadro de dialogo devuelve un estado "0" para Ok y un "1" para el botón cancelar, un valor de ".1" indica un error en el cuadro de dialogo.
zenity_notification(texto, icono)	Despliega un icono con un texto y un área de notificación. La variable texto, coloca el texto en el área de notificación y la variable opcional icono determina que icono será mostrado. El icono puede ser ya sea información, pregunta, advertencia y error.
zenity_list(titulo, columnas, dato, opción1, ...)	Despliega una lista gralica de datos. La variable titulo coloca el titulo de la lista. La variable columna debe ser un arreglo de caracteres NxM conteniendo los datos de la lista, es posible cambiar el funcionamiento de la lista cambiando los argumentos de la función. Estos pueden ser: • "checklist" la primera fila en la lista será una casilla de verificación. El primer valor de cada dato debe ser "VERDADERO" ó "FALSO" • "radiolist" la primera fila en la lista será una casilla de verificación "radiolist". El

	primer valor de cada dato debe ser"VERDADERO" ó "FALSO" • "editable" los valores de la lista pueden ser editables por el usuario. • "a numeric value" el valor retornado por la función será el valor de esa columna de la fila seleccionada por el usuario. • "all" el valor retornado por la función será la fila entera seleccionada por el usuario.
zenity_text_info(titulo, texto, editable)	Despliega una gran cantidad de texto en un cuadro de texto grafico. El titulo del cuadro de texto será colocado por la variable título, el texto desplegado se introduce con la variable texto. Si el argumento opcional "editable" es activado el texto podrá ser editado. En este caso el texto alterado es retornado por la función.

Tutorial

A continuación se presenta 3 ejemplos para poner en práctica las funciones estudiadas anteriormente.

Implementación del comando *zenity_message*

Como se mostro en la Tabla anterior, este comando despliega un mensaje gráfico de dialogo. La variable "texto" coloca el mensaje del cuadro de dialogo, y la variable opcional "tipo" coloca el tipo de mensaje. Para este ejemplo se desea crear un cuadro de dialogo de tipo "pregunta" que nos informe si "deseamos cerrar el programa", el procedimiento es sumamente sencillo, iniciamos Octave, cargamos el paquete de zenity mediante el comando *"pkg load zenity"* y se escribe lo siguiente [15]:

zenity_message('Estas seguro de cerrar el programa', 'question')

Figura 112. Comando "zenity_message".

Como se pudo observar en la Figura 112, el código da como resultado un cuadro de texto gráfico de información, donde se pregunta si "deseamos cerrar el programa".

Implementación del comando *zenity_entry*

Para este ejemplo con el comando *zenity_entry* se desplegará un cuadro de dialogo para introducir una "contraseña", el comando es el siguiente:

zenity_entry('Introduzca la contraseña', 'clave', 'password')

El comando solo requiere colocar el titulo en este caso *'Introduzca la contraseña'*, el texto a desplegar por default *'clave'* y activar la propiedad "password" para que no se muestre el texto a introducir [15]. Todo este se puede observar en la Figura 113.

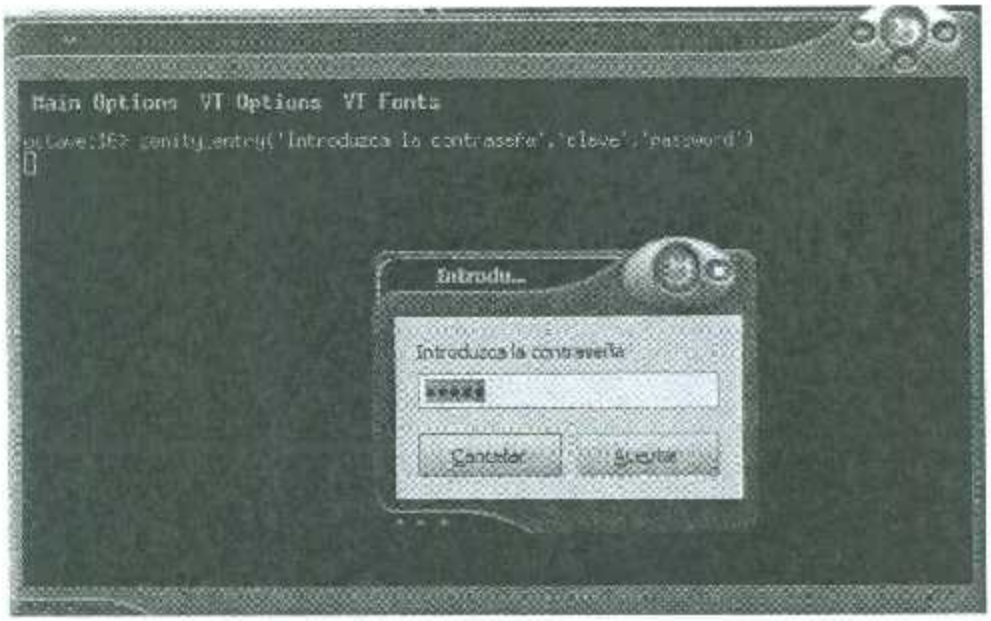


Figura 113. Ejemplo del comando "zenity_entry"

Implementación del comando *zenity_progress*

zenity_progress crea una barra de progreso que puede ser actualizada manualmente con el fin de mostrar al usuario el progreso de cierta acción. La barra de progreso deber ser inicializada como una variable. El primer parámetro es el título, los siguientes son dos parámetros opcionales estos pueden ser "auto-close" para cerrar la barra de progreso cuando se alcance el 100 por ciento ó "pulsate" para que la barra "avance" por medio de la barra espaciadora. Para actualizar la barra la función zenity_progress debe ser llamada nuevamente, es decir, el primer parámetro es la variable de la barra de progreso que fue inicializada, el segundo parámetro es el nuevo progreso, y el tercer parámetro opcional es el texto del progreso actual [15]. Lo expuesto anteriormente se muestra en las Figuras 114 y 115.

Figura 114. Ejemplo 1 del comando zenity_progress.

Figura 115. Ejemplo 2 del comando zenity_progress.

ANEXO D

Conceptos básicos y creación de Interfaces Gráficas en Python.

Para comenzar a programar en Python se deben tener en cuenta algunos conceptos básicos

Llamar al intérprete

En Linux, el intérprete de Python se suele instalar como '/usr/local/bin/python' en aquellas máquinas donde esté disponible. En Windows, se instala en el directorio 'Archivos de programa' ó en este caso instalándolo por medio de Cygwin (véase Anexo A). Poner este directorio en el path hace posible arrancarlo tecleando en el intérprete de órdenes el comando: *python* (Véase Figura 116) [16].

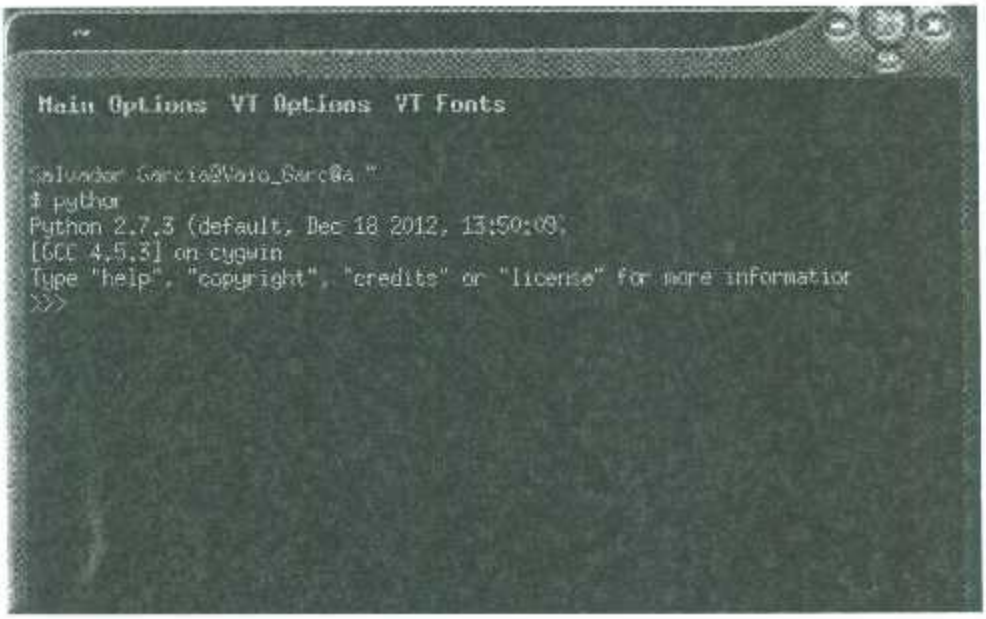


Figura 116. Llamar al intérprete en Python.

En Python todo está modularizado

Esta sí es una diferencia esencial entre Matlab/Octave y Python. En estos lenguajes cualquier función de la biblioteca está accesible al intérprete. Esto hace que, a medida que el número de funciones crece, crezca también la probabilidad de conflictos.

En Python todas las bibliotecas, incluso la biblioteca estándar, están modularizadas. Por ejemplo, si queremos calcular el seno de *pi* tendremos que importar antes el módulo que contiene tanto la función seno como el valor de *pi* [16].

A continuación se presentan algunas nociones básicas de Python, para gente que ya ha programado en algún otro lenguaje.

Entrada/salida básica

La orden para escribir es "**print**". Se puede pedir datos al usuario mediante consola con "**input**". Si los datos son numéricos, podemos usar "**int(input())**" para convertirlos. Las líneas no necesitan punto y coma: cada orden termina al final de la línea correspondiente. No es necesario declarar "**variables**": se inicializan automáticamente la primera vez que se usan, y el tipo de datos se deduce automáticamente a partir de su declaración [16]:

```
print ("Dime un numero")
primerNumero = int( input() )
print ("Dime otro numero")
segundoNumero = int( input() )
print ("Su suma es ")
print (primerNumero + segundoNumero)
```

Condiciones

Existe una orden "**if**" para comprobar condiciones. La condición que se verifica no necesita estar indicada entre paréntesis, al contrario que en los lenguajes que derivan de C (como C, C++, C#, Java, PHP y algún otro).
La línea del "if" debe terminar con un símbolo de "**dos puntos**", y todo el bloque de órdenes que se va a ejecutar en caso de que se cumpla la condición, estará tabulado algo más a la derecha (típicamente 4 espacios; el mínimo es dos... o un carácter de tabulación). Para indicar qué hacer si no se cumple la condición, tenemos una cláusula "**else**" opcional, que también debe ir seguida por un símbolo de "dos puntos" [16]:

```
num1 = int(input( "Dime un numero " ))
num2 = int(input( "Dime otro numero " ))
if num1 > num2:
   print( "El primero es mayor" )
else:
   print ("El segundo es mayor" )
```

Condiciones múltiples

La orden para escribir es "**print**"

```
nota = int(input("Introduzca la nota "))
```

```
if nota == 10:
   print ("Sobresaliente alto")
elif nota == 9:
   print ("Sobresaliente bajo")
elif nota == 8:
(...)
```

Repeticiones

Podemos repetir bloques de órden con "**while**", que tiene una sintaxis muy similar a "if". No existe un repeat-until, ni un do-while, que se pueden conseguir modificando ligeramente la estructura del "while" para garantizar que siempre se dé un primer paso [16]:

```
print ("Dime un numero")
x = int( input ( ) )
suma = 0
while x != 0:
  suma = suma + x
  print ("La suma es {}".format(suma) )
  print ("Dime otro numero")
  x = int( input ( ) )
print ("Terminado")
```

Contadores

Si queremos contar del uno al 10, lo podemos hacer con un "while", pero también existe una orden "for", que recorre los valores de un conjunto; si se trata de un contador, lo habitual es crear un "rango de valores" usando "range", al que se le indica el valor inicial (incluido) y el valor final (no incluido) [16]:

```
for x in range(1,11):
  print (x)
```

Y si queremos que no vaya de uno en uno, podemos añadir un tercer parámetro, el incremento:

```
for x in range(10,21,2):
  print (x)
```

Funciones

Las funciones se definen con la palabra "**def**". A continuación, se indica el nombre de la función, los parámetros y el símbolo de dos puntos. El contenido de la función deberá estar tabulado más a la derecha, como siempre [16]:

```
def saludar(nombre):
   print ("hola")
   print (nombre)
```

Si la función devuelve valores, el único cambio será incluir la correspondiente sentencia "**return**":

```
def sumar(x,y):
   return x+y
```

Ejemplo de programación

Con el fin de poner en práctica lo aprendido anteriormente, se propone realizar un ejercicio simple. Dados los vectores $X = [0,1,2,3,4,5,6,7,8,9,10]$ y $Y = [0,3,4,8,6,4,3,5,7,4,0]$ realizar un programa en Python que grafique estos dos arreglos.

Para ello, se abre un archivo *.txt y se escribe el siguiente código.

```
#importando librerias

from matplotlib import pyplot as plt

#definiendo vectores

x = [0,1,2,3,4,5,6,7,8,9,10]

y = [0,3,4,8,6,4,3,5,7,4,0]

#agregando propiedades al objeto figura

fig = plt.figure()

plt.xlim((0, 10))

plt.ylim((0, 10))

plt.grid(True)

#
```

#Graficando

plt.plot(x,y,'r', linestyle='-',)

plt.show ()

El código anterior esta divido en 4 etapas. La primera consiste en importar las librerías en este caso de la librería "*matplotlib*" solo se importa el modulo "*pyplot*" que es utilizado para graficación y se define como "*plt*" para realizar una programación más rápida y dinámica, lo siguiente es definir los vectores antes mencionados, la tercera etapa corresponde a dar formato al objeto figura para la gráfica, es decir, se definen límites y se coloca cuadrícula a la figura, por último se procede a graficar por medio de la instrucción plot y proporcionarle características al objeto línea como el color y el estilo , es importante resaltar que antes de utilizar este comando se debe importar el modulo "*plt*" como se había mencionado anteriormente en este anexo.

Ya que se ha escrito el código anterior, el archivo *.txt es guardado y se cambia su extensión por *.py, se guarda en el directorio donde se encuentra el programa Cygwin, se ejecuta el emulador de Linux y escribimos el siguiente comando: *python nombredelarchivo.py*, en la Figura 117 se muestran los resultados.

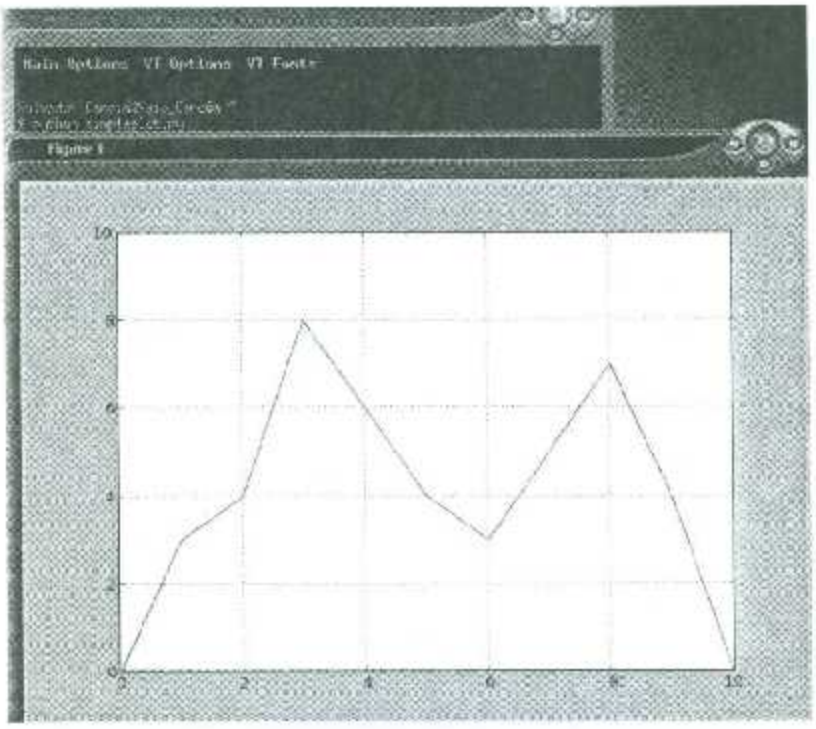


Figura 117. Ejemplo de programación en Python.

Como se puede observar, se ha graficado exitosamente los vectores X y Y, cumpliéndose así el objetivo.

Creación de interfaces graficas con Python

Python provee varias opciones para desarrollar interfaces graficas de usuario (GUIs). Las más importantes están listadas abajo:

- **Tkinter:** Tkinter es la interfaz de Python para la herramienta Tk GUI que viene por default con Python.
- **wxPython:** Se trata de una interfaz de Python de código abierto para wxWindows http://wxpython.org
- **JPython:** JPython es un puerto de Python para Java que proporciona a los scripts de Python acceso a las librerías de Java http://www.jython.org.

Hay muchas otras interfaces disponibles que no se han nombrado aquí y se pueden encontrar en la red.

Programación en Tkinter

Para este proyecto se decidió utilizar Tkinter debido a que cuando Python se combina con Tkinter proporciona una potente interfaz orientada a objetos y una manera rápida y fácil de crear aplicaciones GUIs.

Crear una aplicación con interfaz gráfica utilizando Tkinter es una tarea fácil. Todo lo que se necesita hacer es realizar los siguientes pasos:

- Importar el módulo Tkinter.
- Crear la ventana principal de la aplicación GUI.
- Agregue uno o más de los "widgets" a la aplicación GUI.
- Introduzca el bucle de eventos principal para tomar medidas contra cada evento activado por el usuario.

Ejemplo

```
import Tkinter
top = Tkinter.Tk()
#El código para agregar los widgets deberá ir aquí....
top.mainloop()
```

Esto debería crear la siguiente ventana como se muestra en la Figura 118.

Figura 118. Ventana de Interfaz Gráfica hecha en Tkinter con Python.

Tkinter ofrece varios controles, como botones, etiquetas y cuadros de texto utilizados en una aplicación GUI. Estos controles son comúnmente llamados widgets. En este momento hay 15 tipos de widgets en Tkinter. A continuación en la Tabla 5 se presentan estos widgets, así como una breve descripción de ellos.

Tabla 5. Descripción de los widgets en Tkinter.

Operador	Descripción
Button	Se utiliza para mostrar los botones de la aplicación.
Canvas	Se utiliza para dibujar formas, tales como líneas, óvalos, polígonos y rectángulos, en su aplicación.
CheckButton	Se utiliza para mostrar una serie de opciones como casillas de selección. El usuario puede seleccionar varias opciones a la vez.
Entry	Se utiliza para mostrar un campo de texto de una sola línea para aceptar los valores de un usuario.
Frame	Se utiliza como un widget contenedor para organizar otros widgets.
Label	Se utiliza para proporcionar un título de una sola línea para otros widgets. También puede contener imágenes.
Listbox	Se utiliza para proporcionar una lista de opciones para el usuario.
Menubutton	Se utiliza para mostrar los menús de la aplicación.
Menu	Se utiliza para proporcionar varios comandos de un usuario. Estos comandos

	están contenidas dentro botón MENU.
Radiobutton	Se utiliza para mostrar una serie de opciones como botones de radio. El usuario Puede seleccionar sólo una opción a la vez.
Scale	Se utiliza para proporcionar un widget deslizador.
Scrollbar	Se utiliza para agregar la capacidad de desplazamiento a varios widgets, tales como cuadros de lista.
Text	Se utiliza para mostrar el texto en varias líneas.
Toplevel	Se utiliza para proporcionar una ventana separada.
Spinbox	Es una variante del control Entry Tkinter estándar, que se puede utilizar para seleccionar a partir de un número fijo de valores.
PanedWindow	Es un widget que puede contener cualquier número de paneles, dispuestos en sentido horizontal o vertical.
Label Frame	Su propósito principal es actuar como un espaciador o contenedor para diseños de ventana complejos.
tkMessageBox	Este módulo se utiliza para mostrar cuadros de mensaje en sus aplicaciones.

Calculadora con Tkinter y Python

Para poner en práctica lo estudiado anteriormente, Se creará una pequeña GUI que asemeje una calculadora simple que pida dos números cualquiera y realice las operaciones básicas de suma, resta, multiplicación y división entre ellos dos utilizando Python y la herramienta Tkinter.

El código es el siguiente:

```
from Tkinter import *

import Tkinter as Tk

import math

#

def Add():
```

```
    x1 = float(Entry.get(x))
    y1 = float(Entry.get(y))
    xy = x1 + y1
    textbox.insert(Tk.INSERT,"%d\n" % (xy))
    textbox.delete(2.0,Tk.END)#borro los datos desde el segundo valor hasta el final
    textbox.mark_set(Tk.INSERT, 1.0)#coloco el cursor desde el inicio
    print xy
#
def Sub():
    x1 = float(Entry.get(x))
    y1 = float(Entry.get(y))
    xy = x1 - y1
    textbox.insert(Tk.INSERT,"%d\n" % (xy))
    textbox.delete(2.0,Tk.END)#borro los datos desde el segundo valor hasta el final
    textbox.mark_set(Tk.INSERT, 1.0)#coloco el cursor desde el inicio
    print xy
#
def Mul():
    x1 = float(Entry.get(x))
    y1 = float(Entry.get(y))
    xy = x1 * y1
    textbox.insert(Tk.INSERT,"%d\n" % (xy))
    textbox.delete(2.0,Tk.END)#borro los datos desde el segundo valor hasta el final
    textbox.mark_set(Tk.INSERT, 1.0)#coloco el cursor desde el inicio
    print xy
```

```
#
def Div():
    x1 – float(Entry.get(x))
    y1 = float(Entry.get(y))
    xy = x1 / y1
    textbox.insert(Tk.INSERT,"%d\n" % (xy))
    textbox.delete(2.0,Tk.END)#borro los datos desde el segundo valor hasta el final
    textbox.mark_set(Tk.INSERT, 1.0)#coloco el cursor desde el inicio
    print xy
#
#
root = Tk.Tk()
root.title("Calculadora hecha por Ing.Salvador Garcia")
root.minsize(5,10)
root.geometry("270x250")
#
x = Entry(root,width=4)
x.grid(row=0, column=0, columnspan=1)
#
y = Entry(root,width=4)
y.grid(row=0, column=1, columnspan=1)
#
w = Label(root, text="Operaciones")
w.grid(row=1, column=0, columnspan=4)
#
```

```
w1 = Label(root, text="Resultado")
w1.grid(row=4, column=0, columnspan=4)
#
w2 = Label(root, text="Calculadora\nhecha por:\nIng.Salvador Garcia")
w2.grid(row=2, column=10, columnspan=4)
#
textbox=Tk.Text(root,height=2,width=16)
textbox.grid(row=5, column=0, columnspan=4)
#
Button(root, text='Sum', command=Add).grid(row=2, column=0, columnspan=1)
Button(root, text='Rest', command=Sub).grid(row=2, column=1, columnspan=1)
Button(root, text='Mult', command=Mul).grid(row=3, column=0, columnspan=1)
Button(root, text='Div', command=Div).grid(row=3, column=1, columnspan=1)
Button(root, text='Salir',activebackground='blue', command=root.quit).grid(row=6, column=0, columnspan=4)
#
root.mainloop()
```

El código anterior es relativamente sencillo, solo se debe seguir lo expuesto en este anexo. Primero que nada hay que definir las funciones para cada evento de la interfaz y enseguida definir la parte del bucle de eventos principal de la GUI (root = Tk.Tk()) para tomar medidas contra cada evento activado por el usuario y añadir diversas características a cada "widget" agregado.

Dicho lo anterior, el primer paso es agregar las librerías Tkinter (creación d GUIs) y math (operaciones matemáticas) que son los módulos requeridos para este ejemplo. A continuación se define cada función para cada evento de la interfaz, por ejemplo para la operación suma, la función tendrá el nombre de *"Add"* y se define de la siguiente manera *def Add():* con esto se ha declarado la función. Dentro del cuerpo de la función se agrega dos variables *x1* y *y1* junto con los comando *Entry.get(x)* y *Entry.get(y)* que serán cajas de texto editables para obtener el valor de los dos números que se desean calcular en la GUI. Después, se realiza la operación (en este caso la suma) por medio de la instrucción *xy = x1*

+ *y1* *y* por último, se mostrará el resultado utilizando el comando *textbox.insert(Tk.INSERT,"%d\n" % (xy))* que será una caja de texto para mostrar la respuesta dentro de la GUI. El mismo procedimiento se realizará para cada función (resta, multiplicación y división) modificando solo la parte de la operación.

Ya que se ha creado la parte de las funciones, lo que resta es dar formato y características a los widgets dentro del bucle de eventos principal de la GUI (root = Tk.Tk()). Por ejemplo, para el caso de los cuadros de texto editables se escribe las siguientes lineas de código:

```
x = Entry(root, width=4)

x.grid(row=0, column=0)
```

Aquí se está especificando el ancho del cuadro de texto editable (4) y la posición de este mismo, que sería reglón 0, columna 0, cabe resaltar que para el cuadro de texto que mostrará los resultados se sigue exactamente el mismo procedimiento solo que en vez del comando *Entry* se utiliza el comando *Text*. Si se quisiera agregar algunas etiquetas para indicar los nombres de cada parte de la GUI, se puede utilizar el comando *Label*. Como ejemplo, en el programa se pone el título del trabajo de la siguiente manera:

```
w2 = Label(root, text="Calculadora\nhecha por:\nIng.Salvador Garcia")

w2.grid(row=2, column=10)
```

Como se puede apreciar, se está estableciendo lo que se desea que se muestre en la etiqueta que es "Calculadora hecha por: Ing. Salvador Garcia" y la posición en donde se requiere que aparezca (renglón 2, columna 10).

Para los botones se utiliza la siguiente estructura:

```
Button(root, text='Sum', command=Add).grid(row=2, column=0)
```

Nuevamente tomamos el ejemplo de la suma. Observando las lineas de código anterior, se necesita declarar el texto que tendrá el botón (Sum) y el "comando", que para este botón es la función "*Add*" que ya se había explicado anteriormente, enseguida se establece la posición del botón que será renglón 2 y columna 0. Así sucesivamente se realizara el mismo procedimiento para cada botón de cada operación, cambiando solamente la función ó comando y la posición deseada.

Finalmente, se termina el bucle principal de la GUI por medio de la instrucción:

```
root.mainloop()
```

Si todo lo anterior se ha hecho correctamente, solo resta abrir el emulador Cygwin y escribir el comando *python nombredelprograma.py* para iniciar la aplicación. En la Figura 119 se muestran los resultados.

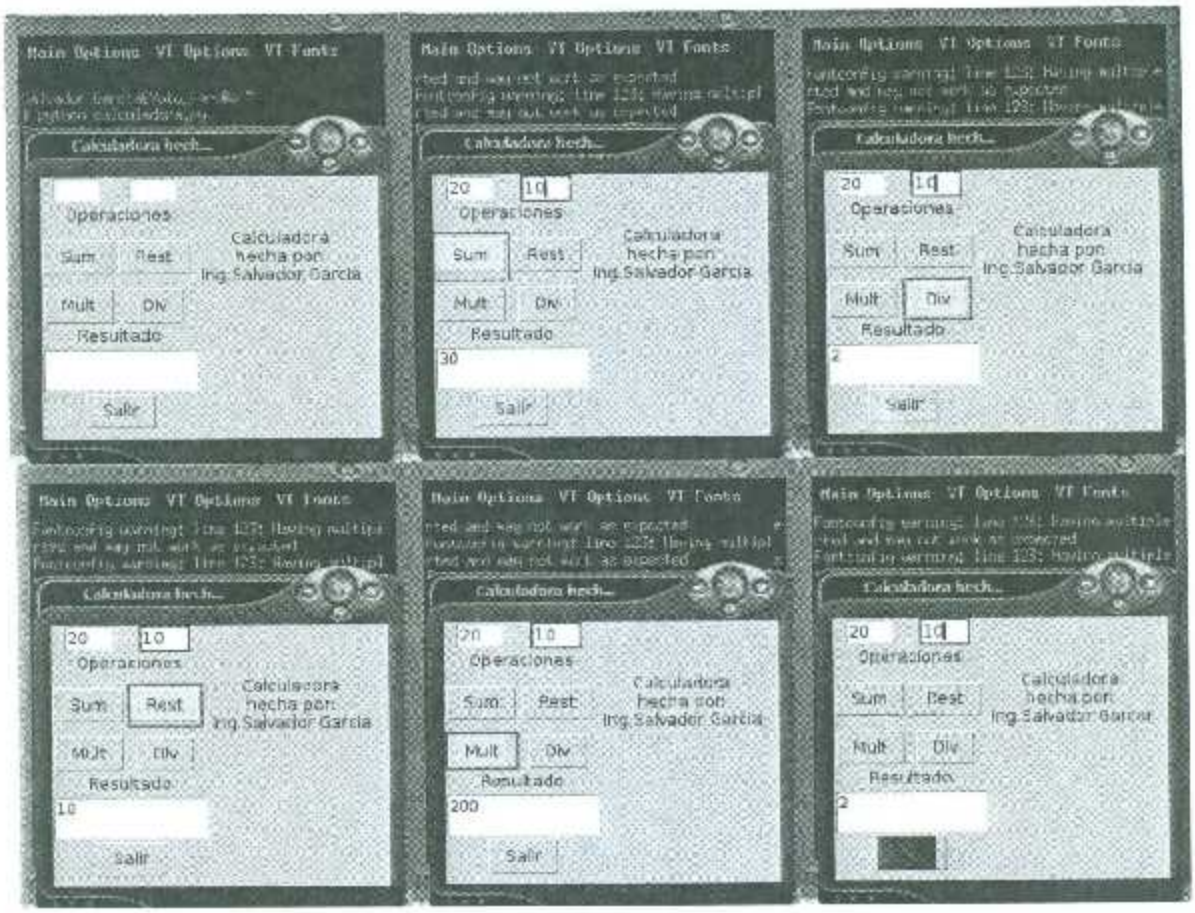


Figura 119. Calculadora hecha en Tkinter con Python.

Como se puede observar, se ha creado una pequeña GUI de una calculadora simple que puede realizar las operaciones básicas de suma, resta, multiplicación y división entre dos números cualquiera, cumpliéndose así el objetivo.

Con esto se da por terminado, la guía básica para programación en Python.

ANEXO E

Configuración de los módulos de radiofrecuencia X-Bee

Para la configuración de los módulos X-Bee será necesario utilizar los programadores USB de X-Bee (ver Figura 120), así como llevar a cabo la instalación del programa X-CTU el cual, se puede descargar de la página http://ftp1.digi.com/support/utilities/40002637_c.exe, además de descargar los controladores necesarios para crear puertos COM (seriales) virtuales los cuales son necesarios para identificar en el sistema operativo de la computadora los dispositivos convertidores de USB-RS232, como es el caso de los programadores USB de X-Bee. La descarga de los controladores de estos mismos se puede hacer de la página http://www.ftdichip.com/FTDrivers.htm.

Descargado lo anterior, es necesario realizar la configuración de los dispositivos por medio del programa X-CTU. Cabe mencionar que los parámetros para el protocolo de comunicación serial a usar en los módulos X-Bee, serán: Baud Rate: 9600, Flow Control: None, Data Bits: 8, Parity: None, Stop Bits: 1.

1. Conectar los dos módulos X-Bee a la computadora a través de los programadores USB que aparecen en la Figura 120 y después se abren dos programas X-CTU para cada dispositivo X-Bee correspondiente. Es importante introducir primero los módulos X-Bee y después abrir el programa X-CTU, de lo contrario aparecerá un mensaje de error en el programa X-CTU. Una vez hecho lo anterior, aparecerá una pantalla como la que se muestra en la Figura 121 donde se podrá visualizar los programadores USB conectados a los diferentes puertos COM virtuales.

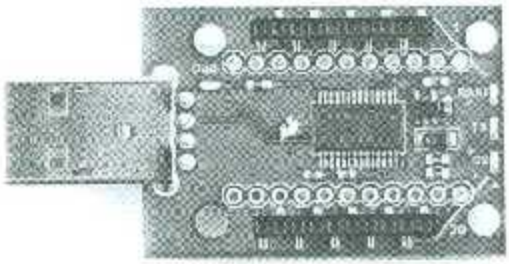


Figura 120. Programador USB de X-Bee.

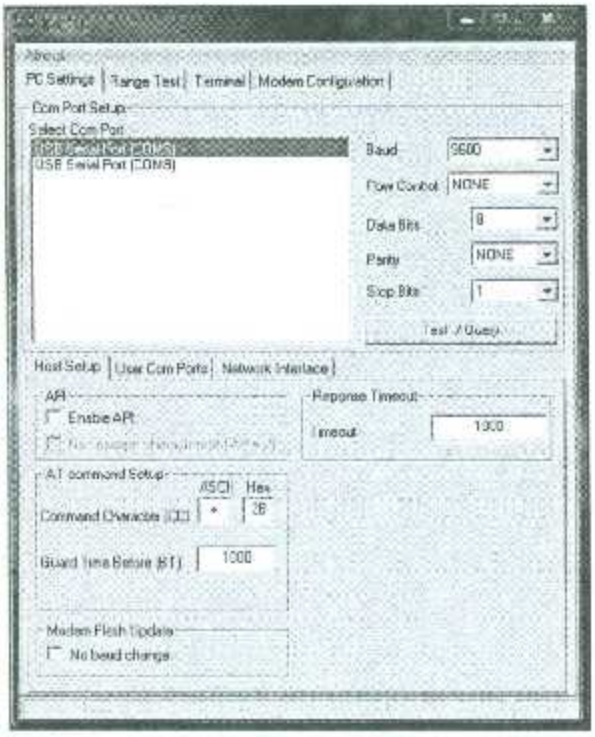


Figura 121. X-CTU (pestaña PC Settings).

2. Seleccione uno de los puertos COM para cada dispositivo X-Bee correspondiente como se muestra en la Figura 121 y configure los cinco parámetros para el protocolo de comunicación serial (Baud, Flow Control, Data Bits, Parity, Stop Bits) con los valores que aparecen en la pantalla de la Figura 121. Coloque el cursor sobre el botón Test/Query y de un click sobre él. Después de unos segundos aparecerá un mensaje que confirmará si el dispositivo (módulo X-Bee) ha sido reconocido por el programa X-CTU y que se puede comunicar con el dispositivo, como lo muestra la Figura 122. En caso contrario de que la comunicación resulte fallida, pasar al Paso 3 y Paso 4 consecutivamente. Repetir el mismo procedimiento con el puerto COM restante del otro dispositivo USB.

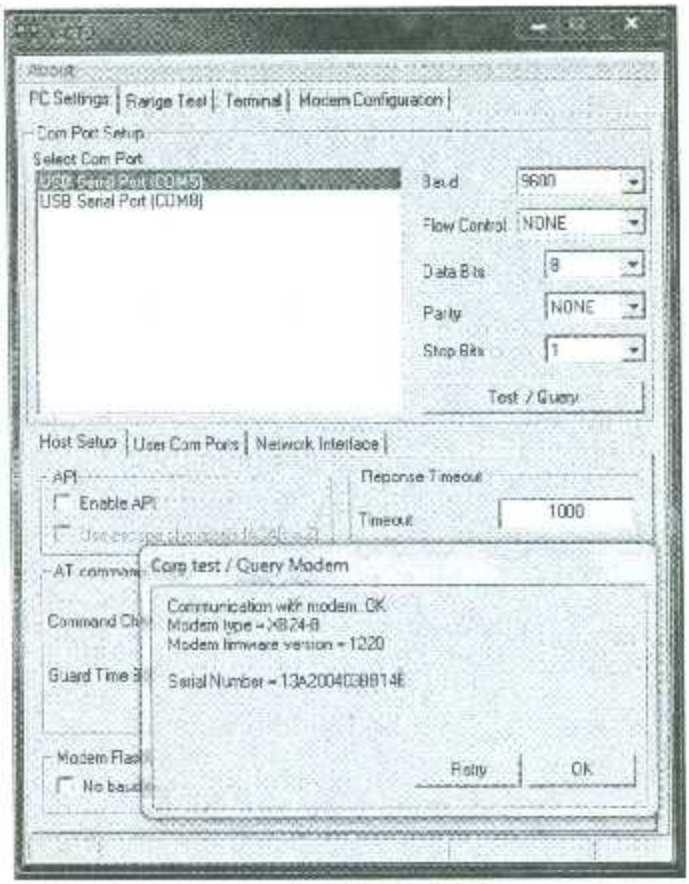


Figura 122. X-CTU (función de botón Test/Query).

3. Seleccione la pestaña de Modem Configuration y de un click sobre el botón Read que se encuentra en el área de Modem Parameter and Firmware. Espere unos segundos hasta que se termine de leer los parámetros de comunicación y se termine de identificar el modelo, el número serial, la función programada y la versión del firmware grabado en el módulo X-Bee, como lo muestra la Figura 123. En caso contrario de que no sea posible leer los parámetros antes dichos, será necesario conectarse a internet y descargar nuevas versiones de módulos X-Bee que nos ayuden a reconocer dicho dispositivo. Lo anterior se hace conectándose a una red de internet y dando click sobre el botón Download new versions. Repetir el mismo procedimiento con el puerto COM restante del otro dispositivo USB.

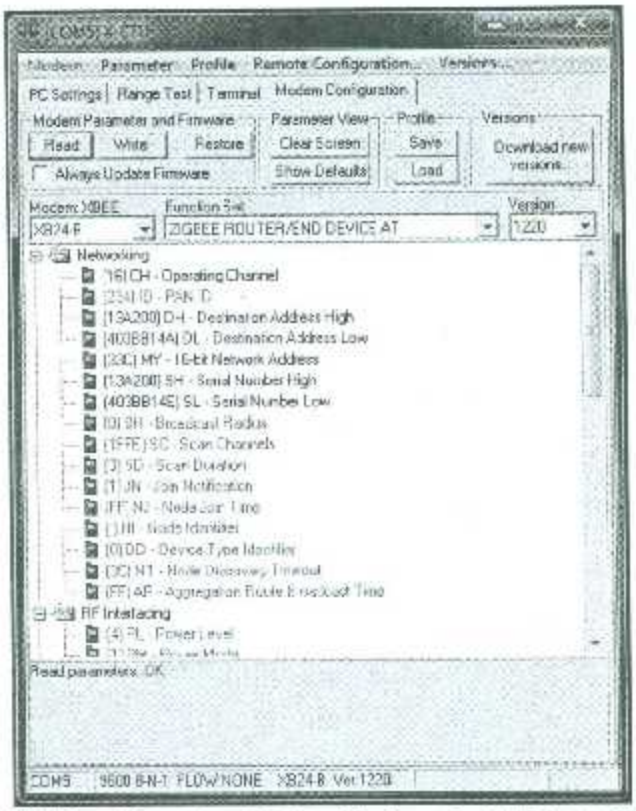


Figura 123. X-CTU Read Parameters (Modem Configuration) del Puerto COM5.

4. Asegúrese que los parámetros descritos en la pestaña de Modem Configuration, en la sección de Serial Interfacing mostrados en la Figura 124, sean idénticos a los cinco parámetros para el protocolo de comunicación serial (Baud, Flow Control, Data Bits, Parity, Stop Bits) mostrados en la Figura 122. Si los cinco parámetros anteriores no coinciden, ajústelos para que sean idénticos dando click con el puntero en cada parámetro y seleccionando el valor deseado. Después de ajustar los parámetros de la sección de Serial Interfacing, se deberá dar click sobre el botón Write que se encuentra a un lado del botón Read. Esto último para que se queden grabados los parámetros deseados para el protocolo de comunicación serial que usarán los módulos X-Bee. Repetir el mismo procedimiento con el puerto COM restante del otro dispositivo USB (solo si es necesario).

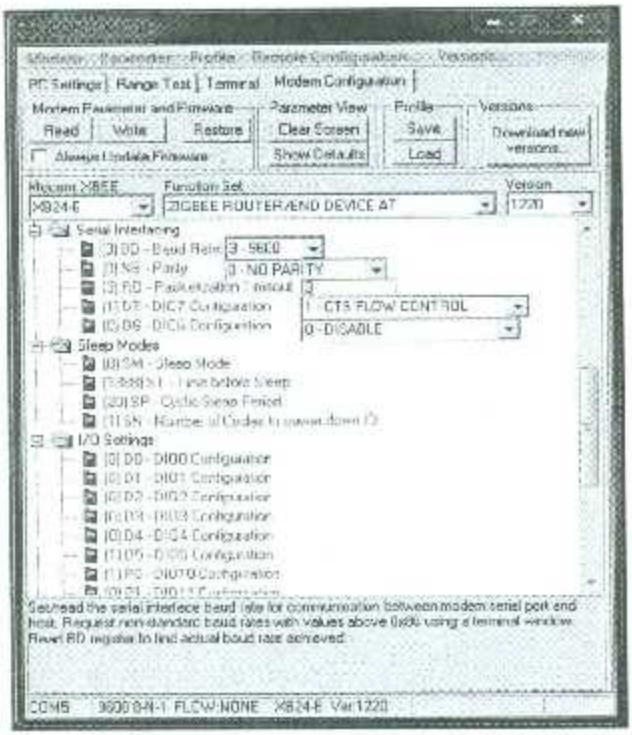


Figura 124. X-CTU Serial Interfacing (Modem Configuration).

5. Después de haber realizado los primeros cuatro pasos anteriores, se debe de confirmar si el dispositivo (módulo X-Bee) en cuestión ha sido reconocido por el programa X-CTU y que es posible comunicarse con él, como se realizó en el paso 2. Este paso solo es necesario en el caso de que la prueba realizada (Test/Query) en el paso 2 haya resultado fallida. Repetir el mismo procedimiento con el puerto COM restante del otro dispositivo USB (solo si es necesario).

6. En este momento, habiéndose realizado los cinco pasos anteriores, ya es posible realizar la configuración de los parámetros de comunicación entre dos módulos X-Bee para que operen como transceptores entre ellos mismos. Para que se lleve a cabo la comunicación inalámbrica entre los 2 módulos X-Bee, será necesario configurar a uno de los dispositivos (un módulo X-Bee) conectado a un puerto COM especifico, con la función de ZNET 2.5 COORDINATOR AT, y al otro dispositivo (módulo X-Bee) conectado a un puerto COM diferente, con la función de ZIGBEE ROUTER/END DEVICE AT (como se muestra en las Figuras 124 y 125). La configuración de la función de cada dispositivo o módulo X-Bee, se hace desde la pestaña de Modem Configuration y dando click donde dice Function Set, esto para cada dispositivo X-Bee en su correspondiente programa X-CTU.

7. Verificar y/o ajustar que los parámetros de Modem Configuration que se encuentran en la Figura 125 en la sección de Networking del dispositivo con función de

Coordinador, para que sean iguales a los parámetros correspondientes de la Figura 123 del dispositivo con función de Router. Asegúrese también de verificar y/o ajustar que los parámetros de Modem Configuration que se encuentran en la Figura 125 en la sección de Addressing del dispositivo con función de Coordinador, para que sean también iguales a los parámetros correspondientes de la Figura 123 del dispositivo con función de Router. Cabe mencionar, que existen 4 parámetros que son distintos para cada dispositivo o módulo X-Bee, y que son el número de serie alto y bajo (SH y SL), y la dirección de destino alta y baja (DH y DL), que se pueden ver en las Figuras 123 y 125. Es importante que la dirección de destino (DH y DL) del primer dispositivo X-Bee sea el número serial del segundo dispositivo X-Bee, y que la dirección de destino del segundo dispositivo X-Bee sea el número serial del primer dispositivo X-Bee. Después de haber configurado los parámetros y las funciones de cada dispositivo X-Bee descritos en este paso, será necesario dar click sobre el botón llamado "Escribir" (Write) para cada dispositivo X-Bee en su correspondiente programa X-CTU. Esto último se hace para que los parámetros ajustados sean grabados sobre cada dispositivo X-Bee correspondiente, y así quede configurada una comunicación inalámbrica entre los dos dispositivos X-Bee transceptores.

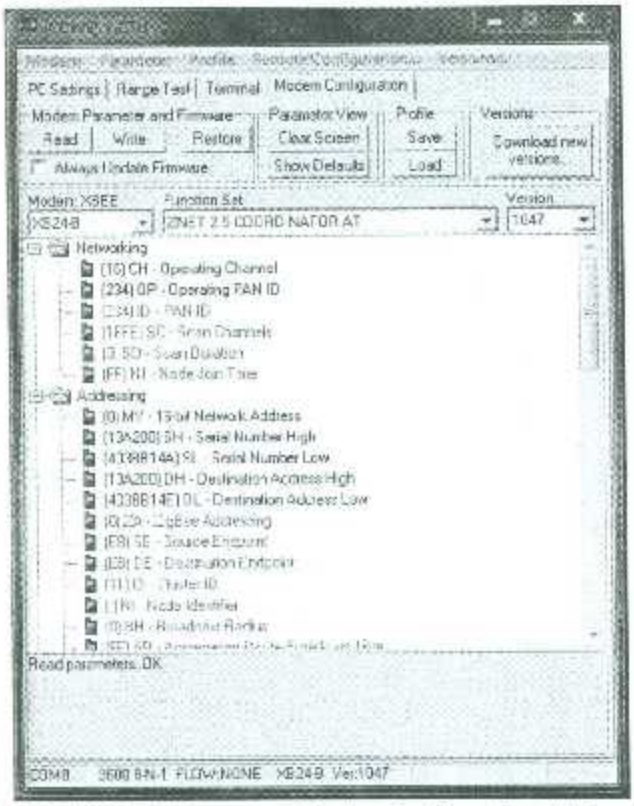


Figura 125. X-CTU Read Parameters (Modem Configuration) del Puerto COM8.

8. Por último, para verificar que exista un enlace de comunicación entre los dos dispositivos X-Bee, será necesario abrir el cuadro de diálogo de la pestaña de "Terminal" de cada programa X-CTU correspondiente a cada dispositivo X-Bee, y teclear sobre el área en blanco de cada pantalla dos comandos para establecer el enlace de comunicación. En una pantalla se teclea el primer comando "+++" y se espera unos segundos hasta que salga un mensaje de OK seguido del primer comando. Después se teclea el segundo comando AT y se oprime la tecla Enter para obtener otro mensaje de "OK" debajo de la línea donde se escribió el segundo comando. Después de teclear el segundo comando será necesario esperar aproximadamente diez segundos para permitir al dispositivo X-Bee establecer la comunicación. Se repiten los dos comandos anteriores en el siguiente programa X-CTU en la pestaña de Terminal para el otro dispositivo X-Bee. Una vez hecho lo anterior, se puede escribir en el área en blanco de la pestaña de Terminal de cualquiera de los dos programas X-CTU correspondientes a cada dispositivo X-Bee, y se podrá visualizar el mensaje tecleado en el área en blanco de la pestaña de Terminal del otro programa X-CTU, lo cual significará que se puede enviar y recibir datos de un dispositivo X-Bee a otro. Cabe resaltar, que los mensajes o comandos que se envían de un dispositivo a otro aparecen en color azul, y los mensajes o repuestas que se reciben de un dispositivo a otro aparecen en color rojo. Para mayor aclaración de este paso, ver Figuras 126 y 127.

Figura 126. X-CTU (Terminal) del Puerto COM5.

Figura 127. X-CTU (Terminal) del Puerto COM8.

ANEXO F

Diseño de filtros activos.

Dado que para el diseño del electrocardiógrafo se requiere de una etapa de filtraje muy precisa, se estudiará el diseño de filtros activos de segundo y *"n"* orden para su posterior uso.

Diseño de filtros activos pasa bajas de segundo orden tipo Sallen-Key.

Para diseñar un filtro activo pasa bajas de segundo orden se requiere solamente que el analista especifique la frecuencia de resonancia, ω_N, y el factor de calidad, Q. Así, Una vez definido el valor requerido de estos parámetros, se hace uso de las ecuaciones que los determinan, en términos de sus componentes [11]:

$$Q = \frac{R_1 R_2 C_1 C_2 \omega_N}{C_2 (R_1 + R_2)} = \sqrt{\frac{C_1}{C_2}} \left(\frac{\sqrt{R_1 R_2}}{R_1 + R_2} \right)$$

$$\omega_N^2 = \frac{1}{R_1 R_2 C_1 C_2}$$

La selección de los componentes R_1, R_2, C_1 y C_2 no es única; existen infinidad de valores de dichos componentes que cumplirán con los requerimientos de ω_N, y del factor de calidad, Q. por ello, el estudiante puede elegir arbitrariamente los valores de los componentes, según su criterio. Sin embargo, una estrategia que en la práctica se utiliza frecuentemente consiste en elegir las resistencias R_1 y R_2 de valor idéntico; así, si $\mathbf{R_1=R_2=R}$, las ecuaciones anteriores se convierten en:

$$Q = \frac{1}{2} \sqrt{\frac{C_1}{C_2}} \quad \text{y} \quad \omega_N = \frac{1}{R \sqrt{C_1 C_2}}$$

Se invita a demostrar, con base en las dos ecuaciones anteriores, que una vez definidos los valores de Q, ω_N y $R_1=R_2=R$, los valores de las capacitancias C_1 y C_2 se calcularán con las fórmulas siguientes:

$$C_1 = \frac{2Q}{\omega_N R} = \frac{2Q}{2\pi f_N R} \quad \text{y} \quad C_2 = \frac{1}{2Q\omega_N R} = \frac{1}{4\pi f_N Q R}$$

Ejemplo:

Diseñar un filtro activo pasa bajas de segundo orden, tipo Sallen-Key, de modo que la frecuencia de resonancia, ω_N, sea de 2KHz y el factor de calidad, Q, sea de .707.

Solución:

Si R1=R2=10KΩ, las capacitancias serán:

$$C_1 = \frac{2Q}{2\pi f_N R} = \frac{2\times 0.707}{2\pi \times 2{,}000 \times 10{,}000} \approx 11.25nF$$

$$C_2 = \frac{1}{4\pi f_N Q R} = \frac{1}{4\pi \times 2{,}000 \times 0.707 \times 10{,}000} \approx 5.63nF$$

Diseño del circuito

En la Figura 128 se muestra el circuito esquemático del filtro.

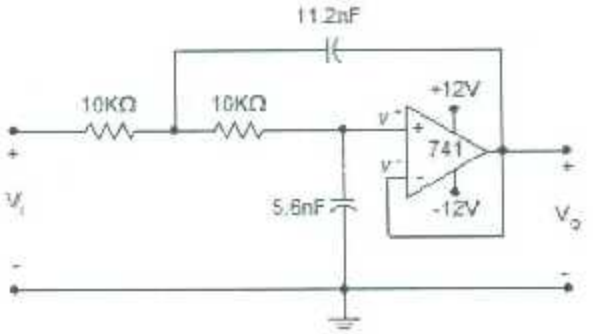


Figura 128. Circuito esquemático de filtro activo pasa bajas de segundo orden tipo Sallen-Key [11].

Respuesta a la frecuencia.

En la Figura 129 aparece la respuesta de frecuencia teórica, calculada con MATLAB, del filtro pasa bajas de segundo orden tipo Sallen Key, diseñado para f_N=2KHz y Q=0.7.

La Figura 130 muestra la respuesta en físico del filtro. Esta se obtuvo aplicando a la entrada del filtro una señal senoidal de 2Vpp en modo de "barrido" (Sweep Mode), con frecuencia de inicio, Fi, de 20 Hz y frecuencia final, Ff, de 200 KHz; el tiempo de barrido, Tb, es de 800mS y el barrido es logarítmico. Dado que el barrido de frecuencia es de 4 décadas, desde 20 Hz hasta 200KHz, en 800 ms, entonces en la grafica cada década será representada cada 2 divisiones, si la base de tiempo del osciloscopio es de 100ms. El estudiante puede comprobar que, efectivamente, los 2KHz (frecuencia de corte) se manifiestan en la cuarta marca. En esa marca la amplitud correspondiente es de 700mV, es decir, aproximadamente el valor de Q [11].

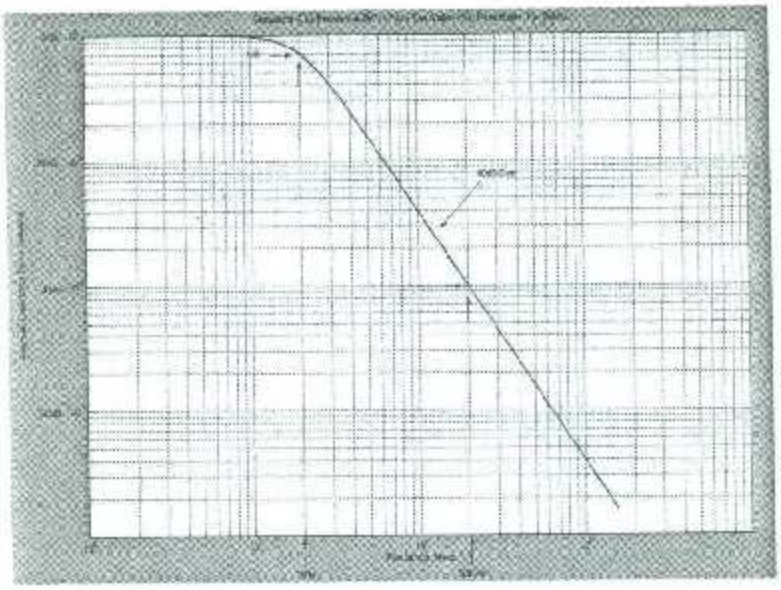


Figura 129. Respuesta de frecuencia teórica de filtro pasa bajas de segundo orden tipo Sallen Key [11].

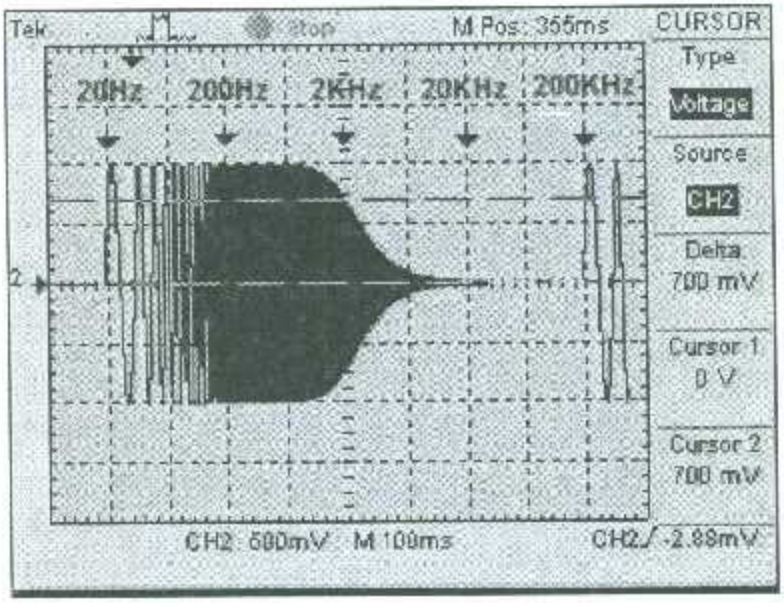


Figura 130. Respuesta en físico del filtro [11].

Diseño de filtros activos pasa altas de segundo orden tipo Sallen-Key

Para diseñar un filtro activo pasa altas de segundo orden se requiere solamente que el analista especifique la frecuencia de resonancia, ω_N, y el factor de calidad, Q. Así, Una vez definido el valor requerido de estos parámetros, se hace uso de las ecuaciones que los determinan, en términos de sus componentes [11]:

$$Q = \frac{R_1 R_2 C_1 C_2 \omega_N}{R_1 (C_1 + C_2)} = \sqrt{\frac{R_2}{R_1}} \left(\frac{\sqrt{C_1 C_2}}{C_1 + C_2} \right)$$

$$\omega_N^2 = \frac{1}{R_1 R_2 C_1 C_2}$$

Como ocurrió en el caso de los filtros pasa bajas, también en este caso la selección de los componentes R_1, R_2, C_1 y C_2 no es única; existen infinidad de valores de dichos componentes que cumplirán con los requerimientos de ω_N, y del factor de calidad, Q. por ello, se puede elegir arbitrariamente los valores de los componentes, según su criterio. En este caso, sin embargo, la estrategia práctica suele ser la elección de las capacitancias C_1 y C_2 de valor idéntico; así, si $\mathbf{C_1=C_2=C}$, las ecuaciones anteriores se convierten en:

$$Q = \frac{1}{2} \sqrt{\frac{R_2}{R_1}} \text{ y } \omega_N = \frac{1}{C\sqrt{R_1 R_2}}$$

Nuevamente, se invita a demostrar, con base en las dos ecuaciones anteriores, que una vez definidos los valores de Q, ω_N y $C_1=C_2=C$, los valores de las resistencias R_1 y R_2 se calcularán con las fórmulas siguientes:

$$R_1 = \frac{1}{2Q\omega_N C} = \frac{1}{4\pi f_N Q C} \text{ y } R_2 = \frac{2Q}{\omega_N C} = \frac{2Q}{2\pi f_N C}$$

Ejemplo:

Diseñar un filtro activo pasa altas de segundo orden, tipo Sallen-Key, de modo que la frecuencia de resonancia, ω_N, sea de 5KHz y el factor de calidad, Q, sea de 2.

Solución:

Si C1=C2=C=.01μF, las resistencias serán:

$$R = \frac{1}{4\pi f_N Q C} = \frac{1}{4\pi \times 5{,}000 \times 2 \times .01 \times 10^{-6}} \approx 795.77\Omega$$

$$R_2 = \frac{2Q}{2\pi f_N C} = \frac{2 \times 2}{2\pi \times 5{,}000 \times .01 \times 10^{-6}} \approx 12.732K\Omega$$

Diseño del circuito

En la Figura 131 se muestra el circuito esquemático del filtro.

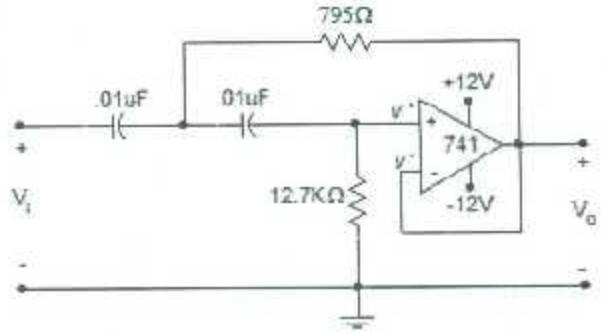


Figura 131. Circuito esquemático del filtro activo pasa altas de segundo orden tipo Sallen-Key [11].

Respuesta a la frecuencia

En la Figura 132, aparece la respuesta de frecuencia teórica, calculada con MATLAB, del filtro pasa altas de segundo orden tipo Sallen-Key, diseñado para f_N=5KHz y Q=2.

La Figura 133 muestra la respuesta en físico del filtro. Esta se obtuvo aplicando a la entrada del filtro una señal senoidal de 2Vpp en modo de "barrido" (Sweep Mode), con frecuencia de inicio, Fi, de 50 Hz y frecuencia final, Ff, de 500 KHz; el tiempo de barrido, Tb, es de 800mS y el barrido es logarítmico. Dado que el barrido de frecuencia es de 4 décadas, desde 50 Hz hasta 500KHz, en 800 ms, entonces en la grafica cada década será representada cada 2 divisiones, si la base de tiempo del osciloscopio es de 100ms. El estudiante puede comprobar que, efectivamente, los 5KHz (frecuencia de resonancia) se manifiestan en la cuarta marca. En esa marca la amplitud correspondiente es de 2V, es decir, aproximadamente el valor de Q [11].

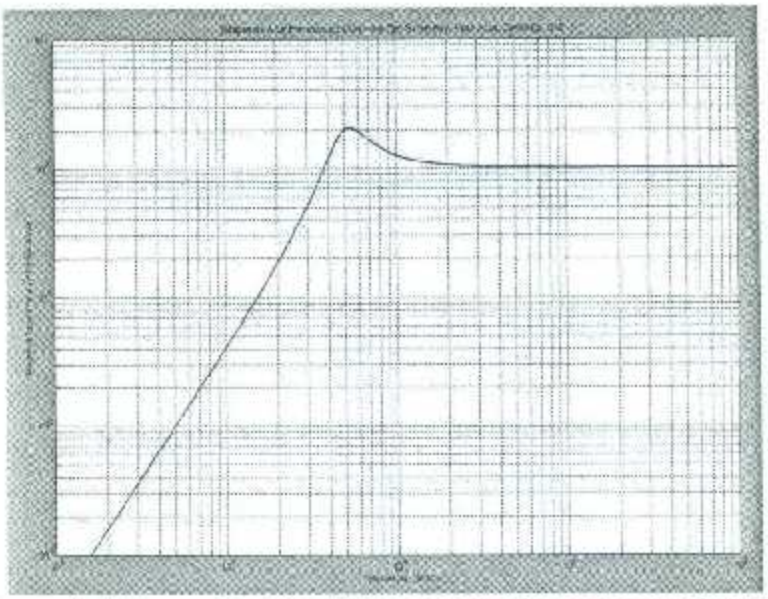


Figura 132. Respuesta de frecuencia teórica de filtro pasa altas de segundo orden tipo Sallen Key [11].

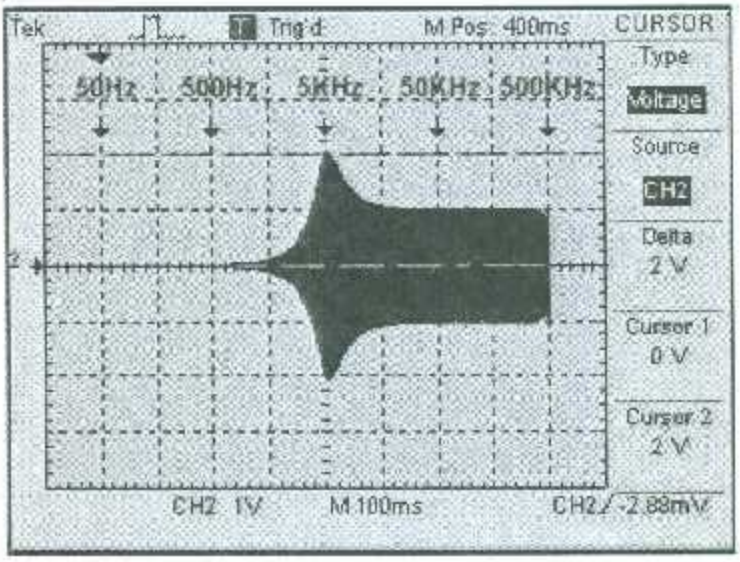


Figura 133. Respuesta en físico del filtro [11].

Cascada de filtros activos.

En general, lo que todo diseñador de filtros desearía es obtener circuitos que muestren gráficas de respuesta a la frecuencia ideales, como las comentadas al principio de este tema. Es decir, que las **pendientes de transición** sean infinitas (90°), entre las regiones de paso de banda y supresión de banda ó como se dice coloquialmente, en forma de "pared". Esta pretensión resulta imposible de lograr en la realidad, debido a que todo sistema físico consume energía y manifiesta retardos de tiempo, según sea su dinámica. Sin embargo, es

posible aproximarse a una respuesta en forma de "pared", si el diseñador coloca en "cascada" una serie de etapas de filtro [11].

La colocación de varias etapas o células de filtro individuales en cascada, eleva el orden del filtro total resultante (véase Figura 134). En general, mientras más alto sea el orden del filtro, más se aproximará su respuesta a la ideal o a la de forma de "pared" [11].

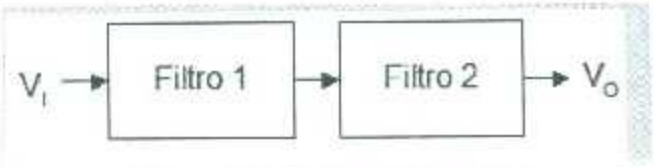


Figura 134. Filtro de dos etapas idénticas.

Existen varias técnicas para diseñar filtros de orden superior (mayor que dos); algunas de ellas son muy sencillas, pero burdas, y se aplican cuando no se requiere de una exactitud y precisión calculada. Por ejemplo, suponga que diseña un filtro simple pasa bajas individual de primer orden, para una frecuencia de corte de 1KHz. En esta circunstancia seguramente el filtro manifestaría -3 decibeles en 1KHz, con pendiente de transición de -20 decibeles por década. No obstante, si el diseñador desea mejorar la respuesta del filtro puede colocar dos etapas idénticas de primer orden y con ello elevar la pendiente de transición hasta -40 dB por década. El costo que el diseñador pagará por mejorar la pendiente de transición, será que ahora, a la frecuencia de 1KHz, la atenuación total será de -6dB y la frecuencia de corte (-3dB) del filtro total se habrá recorrido hacia la izquierda. La gráfica de respuesta a la frecuencia, que se muestra en la Figura 135, pone de manifiesto este efecto de corrimiento, que en el caso de un filtro pasa bajas resulta hacia la izquierda pero en el caso de un filtro pasa altas, resultaría hacia la derecha. También es de suponerse que entre más etapas idénticas se coloquen, mayor será el efecto de corrimiento [11].

El ingeniero británico S. Butterworth investigó sobre el problema de diseño de filtros de orden superior en 1930 y escribió un artículo, (On The Theory Of Filter Amplifiers), donde ofrece una propuesta sumamente ingeniosa y precisa, que se convirtió en estándar universal, en el diseño de filtros [11].

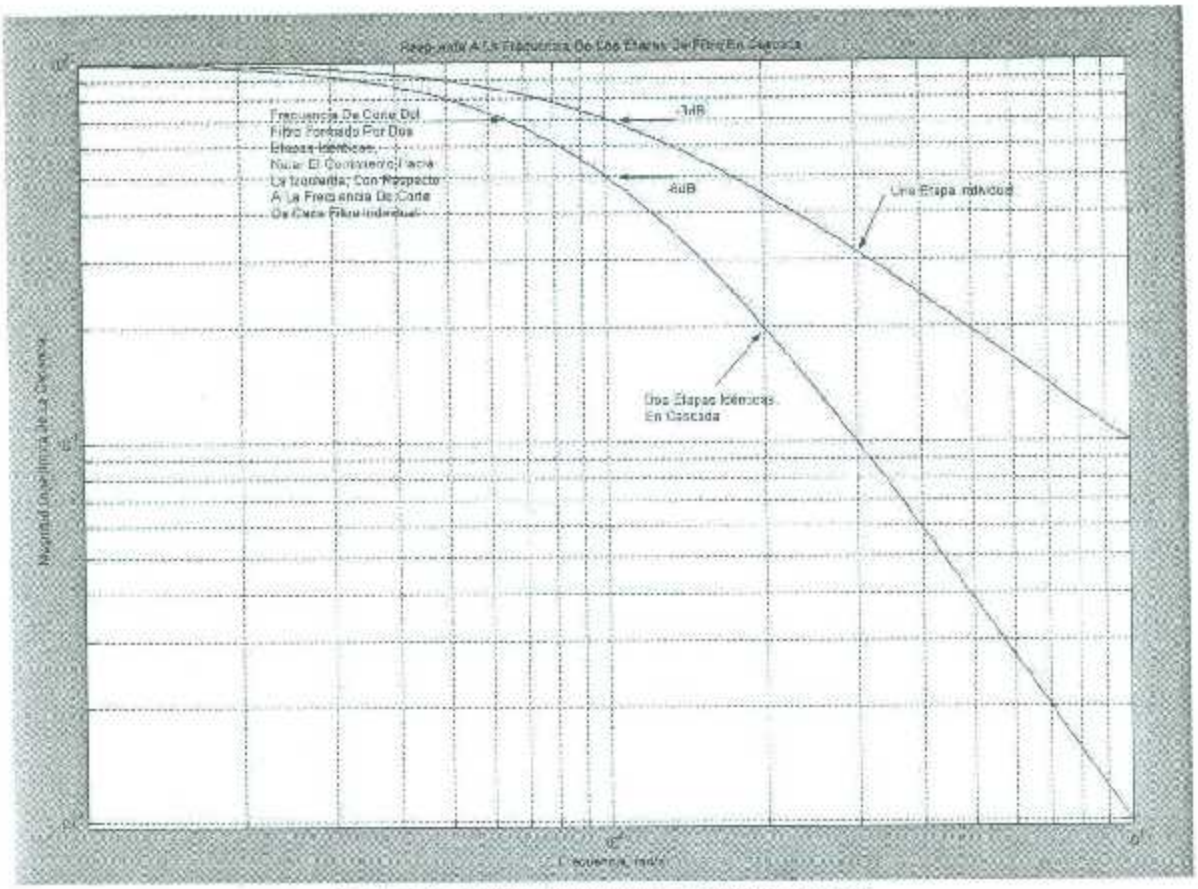


Figura 135. Gráfica de respuesta a la frecuencia [11].

Diseño de filtros Butterworth de n orden

El punto de partida del ingeniero Butterworth fue la pretensión de implementar filtros de respuesta **máximamente plana** con pendientes de transición que dependieran del orden, n, del filtro y que siempre mostraran una frecuencia de corte precisa, de -3dB, independientemente del orden del filtro total.

La Tabla 6 ofrece los valores de las raíces, para diferentes casos del orden n, de filtros Butterworth [11].

Tabla 6. Valores de las raices, para diferentes casos del orden n, de filtros Butterworth [11].

Orden n	Raices (ω_0=1 rad/s)	Factor De Calidad,Q	Función De Transferencia De Cada Etapa De Filtro
2	s_1=-0.7071+j0.7071 s_2=-0.7071-j0.7071	1.4142	$\frac{N_1(s)}{(s^2+1.4142s+1)}$
3	s_1=-0.5000+j0.8660 s_2=-0.5000-j0.8660 s_3=-1.0000	1.0000	$\frac{1}{(s+1)}\times\frac{N_1(s)}{(s^2+1.000s+1)}$
4	s_1=-0.9238+j0.3827 s_2=-0.9238-j0.3827 s_3=-0.3826+j0.9238 s_4=-0.3826-j0.9238	1.0823 2.6131	$\frac{N_1(s)}{(s^2+1.8477s+1)}\times\frac{N_2(s)}{(s^2+0.7653s+1)}$
5	s_1=-0.8090+j0.5878 s_2=-0.8090-j0.5878 s_3=-0.3090+j0.9510 s_4=-0.3090-j0.9510 s_5=-1.0000	1.2360 3.2360	$\frac{1}{(s+1)}\times\frac{N_1(s)}{(s^2+1.6180s+1)}\times\frac{N_2(s)}{(s^2+0.6180s+1)}$
6	s_1=-0.9659+j0.2588 s_2=-0.9659-j0.2588 s_3=-0.7071+j0.7071 s_4=-0.7071-j0.7071 s_5=-0.2588+j0.9659 s_6=-0.2588-j0.9659	1.0352 1.4142 3.8637	$\frac{N_1(s)}{(s^2+1.9318s+1)}\times\frac{N_2(s)}{(s^2+1.4142s+1)}\times\frac{N_3(s)}{(s^2+0.5176s+1)}$
7	s_1=-0.9009+j0.4339 s_2=-0.9009-j0.4339 s_3=-0.6234+j0.7818 s_4=-0.6234-j0.7818 s_5=-0.2225+j0.9749 s_6=-0.2225-j0.9749 s_7=-1.0000	1.1013 1.8020 0.5550	$\frac{1}{(s+1)}\times\frac{N_1(s)}{(s^2+1.8019s+1)}\times\frac{N_2(s)}{(s^2+1.2469s+1)}\times\frac{N_3(s)}{(s^2+0.4450s+1)}$
8	s_1=-0.9808+j0.1950 s_2=-0.9808-j0.1950 s_3=-0.8314+j0.5556 s_4=-0.8314-j0.5556 s_5=-0.5556+j0.8314 s_6=-0.5556-j0.8314 s_7=-0.1950+j0.9808 s_8=-0.1950-j0.9808	1.0196 1.3028 1.7989 5.1258	$\frac{N_1(s)}{(s^2+1.9615s+1)}\times\frac{N_2(s)}{(s^2+1.6629s+1)}\times\frac{N_3(s)}{(s^2+1.1111s+1)}\times\frac{N_4(s)}{(s^2+0.3901s+1)}$
9	s_1=-0.9397+j0.3421 s_2=-0.9397-j0.3421 s_3=-0.7660+j0.6428 s_4=-0.7660-j0.6428 s_5=-0.5000+j0.8660 s_6=-0.5000-j0.8660 s_7=-0.1736+j0.9848 s_8=-0.1736-j0.9848 s_9=-1.0000	1.0641 1.3054 1.9319 5.7587	$\frac{1}{(s+1)}\times\frac{N_1(s)}{(s^2+1.8794s+1)}\times\frac{N_2(s)}{(s^2+1.5320s+1)}\times\frac{N_3(s)}{(s^2+1.0000s+1)}\times\frac{N_4(s)}{(s^2+0.3472s+1)}$

El diseño de filtros pasa bajas y pasa altas Butterworth de n orden es muy sencillo. Todo lo que se tiene que hacer es definir el tipo de filtro, el orden n del mismo y la frecuencia de corte (-3dB), ω_0, requerida.

Enseguida, según sea el orden n, se determina en la Tabla 6, el número de etapas Sallen-Key necesarias para implementar el filtro y las raíces que determinan la ubicación de los polos de cada etapa de segundo orden. Con esa información se calcula el factor de calidad Q, de cada etapa Sallen-Key de segundo orden; recuerde que la parte real de cada par de polos está dada por [11]:

$$s_R = -\frac{\omega_N}{2Q}$$

De tal manera que siempre calcularemos el factor de calidad de cada etapa con la fórmula:

$$Q = \frac{\omega_0}{-2s_R}$$

Conocidos ω_0 y Q de cada etapa, se procederá a diseñar cada una de ellas según los procedimientos ya discutidos en apartados anteriores.

Ejemplo:

Diseñar un filtro Butterworth, pasa bajas de quinto orden (n=5), cuya frecuencia de corte ω_0, de -3dB, sea de 5KHz.

Solución:

De la Tabla 6, para n=5, se tiene que se requieren tres etapas: dos de segundo orden y una etapa de primer orden. La parte real de cada par de polos, de las etapas de segundo orden es:

$$s_{R1}=-0.8090\,\omega_0$$
$$s_{R2}=-0.3090\,\omega_0$$

Por lo tanto, el factor de calidad de cada etapa de segundo orden es:

$$Q_1=\frac{\omega_0}{-2(-0.8090)\omega_0}=\frac{1}{1.618}=0.6180$$

$$Q_2=\frac{\omega_0}{-2(-0.3090)\omega_0}=\frac{1}{0.618}=1.618$$

Ahora se procede a determinar los componentes para cada etapa del filtro de quinto orden.

Consideraremos filtros pasa bajas, Sallen-Key, con resistencias idénticas $R_{11}=R_{21}=R=10K\Omega$. Para la primera etapa:

$$C_{11}=\frac{2Q_1}{2\pi f_0 R}=\frac{2\times 0.618}{2\pi\times 5{,}000\times 10{,}000}\approx 3.93nF$$

$$C_{21}=\frac{1}{4\pi f_N Q_1 R}=\frac{1}{4\pi\times 5{,}000\times 0.618\times 10{,}000}\approx 2.5753nF$$

Para la segunda etapa, también con resistencias idénticas, $R_{12}=R_{22}=R=10K\Omega$ se tiene:

$$C_{12}=\frac{2Q_2}{2\pi f_0 R}=\frac{2\times 1.618}{2\pi\times 5{,}000\times 10{,}000}\approx 10.3nF$$

$$C_{22} = \frac{1}{4\pi f_N Q_2 R} = \frac{1}{4\pi \times 5{,}000 \times 1.618 \times 10{,}000} \approx 0.983nF$$

Para la tercera etapa de primer orden, se tiene que:

$$RC = \frac{1}{2\pi f_3} = \frac{1}{2\pi \times 5000} = 3.18309 \times 10^{-5} s$$

Si se propone un capacitor C=10nF, la resistencia será:

$$R = \frac{3.18309 \times 10^{-5}}{C} \frac{3.18309 \times 10^{-5}}{10 \times 10^{-9}} = 3.183K\Omega$$

Diseño del circuito

En la Figura 136 se muestra el circuito esquemático del filtro Butterworth de 5 orden.

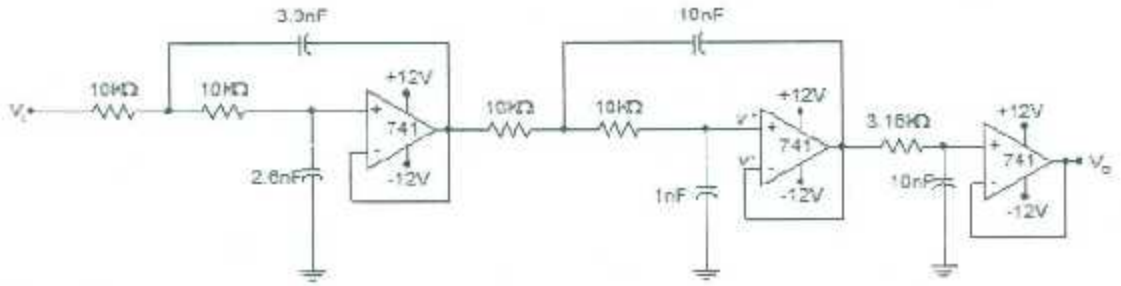


Figura 136. Circuito esquemático del filtro Butterworth de 5 orden [11].

Respuesta a la frecuencia

En la Figura 137 se muestra la respuesta obtenida en físico, del filtro de quinto orden. El oscilograma se obtuvo aplicando una señal de entrada de 2Vpp. El generador se programó en modo de barrido (sweep mode) con frecuencia de inicio de 500Hz y frecuencia final de 50KHz; el tiempo del barrido es de 800ms y es de tipo logarítmico [11].

La Figura 138 muestra una gráfica de respuesta a la frecuencia donde aparecen las contribuciones de cada una de las tres etapas, a la respuesta total del filtro.

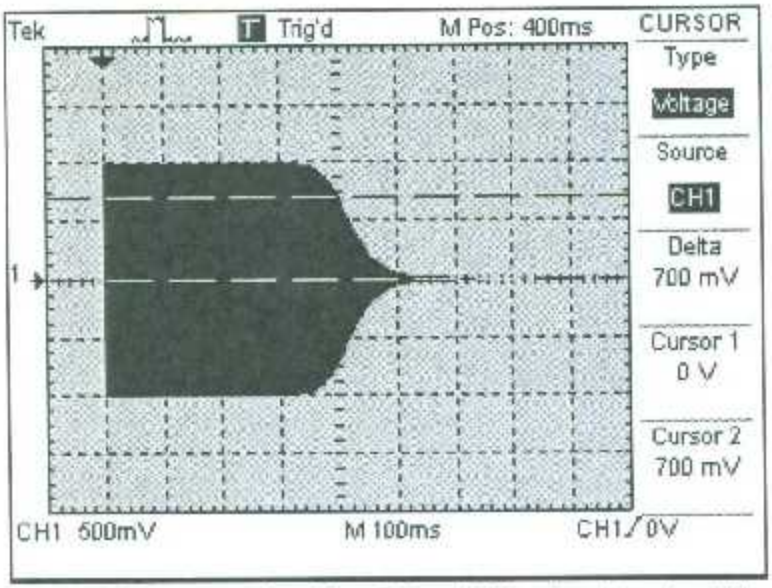


Figura 137. Respuesta obtenida en físico, del filtro de quinto orden [11].

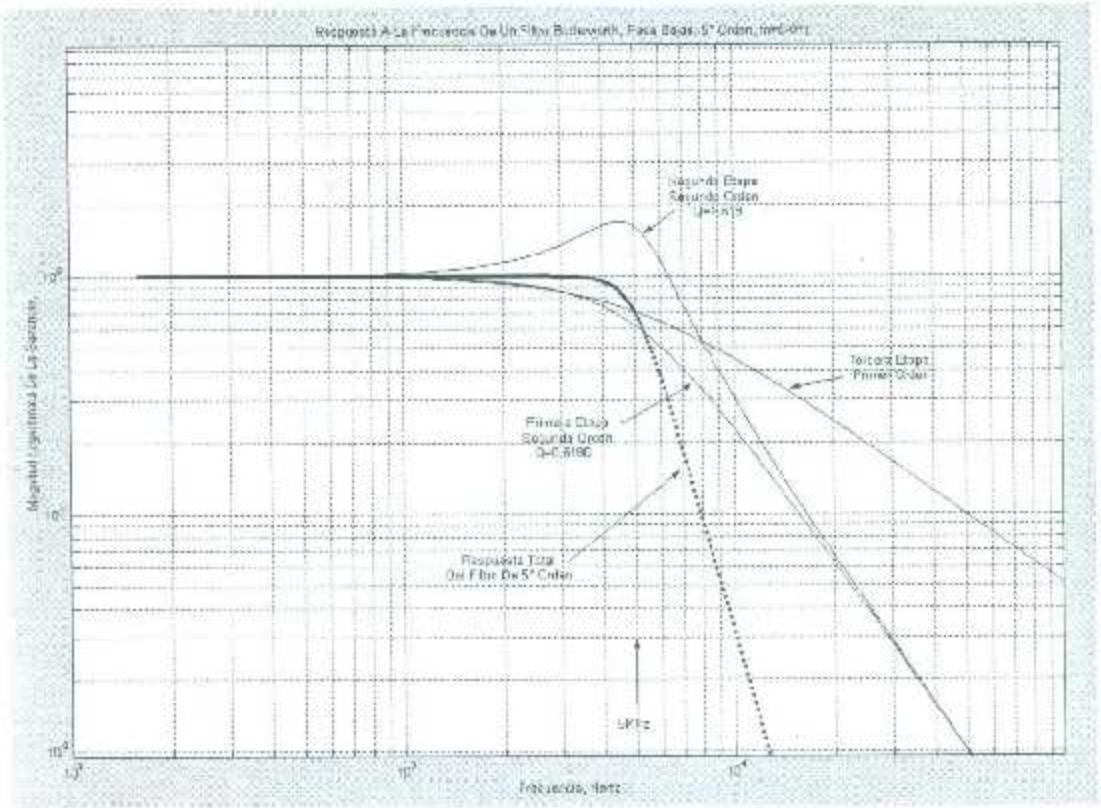


Figura 138. Respuesta a la frecuencia de las contribuciones de cada una de las tres etapas, a la respuesta total del filtro [11].

Diseño de filtros rechazo de banda o filtros notch

Una de las principales aplicaciones de los filtros tipo "notch" es en el diseño de filtros supresores de la frecuencia de 60 Hz que son útiles en varias aplicaciones, en equipos médicos como los electrocardiógrafos donde "limpian" la señal de esta interferencia. En los equipos de audio se utilizan para eliminar el "hum" o ruido causado por la fuente de energía.

Para diseñar un filtro notch se requiere solamente que el analista especifique la frecuencia característica, ω_o. Así, Una vez definido el valor requerido de este parámetro, se hace uso de las ecuaciones que la determinan, en términos de sus componentes:

$$Cf = 2C$$

$$Rf = R/2$$

$$\omega_o = \frac{1}{R_1 R_2 C_1 C_2}$$

La selección de los componentes R_1, R_2, C_1 y C_2 no es única; existen infinidad de valores de dichos componentes que cumplirán con los requerimientos de ω_o. por ello, se puede elegir arbitrariamente los valores de los componentes, según su criterio. Sin embargo, una estrategia que en la práctica se utiliza frecuentemente consiste en elegir las resistencias C_1, C_2, R_1 y R_2 de valor idéntico; así, si $\mathbf{C_1=C_2=C}$ y $\mathbf{R_1=R_2=R}$, la ecuación anterior se convierten en:

$$R = \frac{1}{2\pi C f_N}$$

Ejemplo:

Diseñar un filtro activo de rechazo de banda, tipo notch, de modo que la frecuencia de corte, ω_o, sea de 90Hz.

Solución:

Si C1=C2=C=100nF, las resistencias serán:

$$R = \frac{1}{2\pi C f_N} = \frac{1}{2\pi \times 100 \times 10^{-9} \times 90} \approx 17.6K\Omega$$

$$Rf = \frac{R}{2} = \frac{17.6K\Omega}{2} \approx 8.8K\Omega$$

Y por último el capacitor Cf será:

$$Cf = 2 \times 100nF \approx 200nF$$

Diseño del circuito

En la Figura 139 se muestra el circuito esquemático del filtro activo de rechazo de banda.

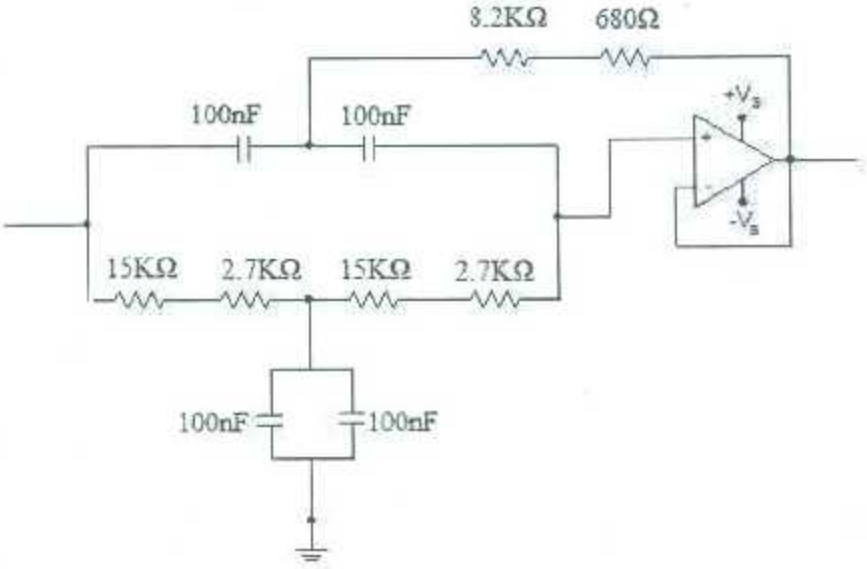


Figura 139. Circuito esquemático del filtro activo de rechazo de banda de 90 Hz.

Respuesta a la frecuencia

En la Figura 140 se muestra la respuesta obtenida en físico, del filtro notch. El oscilograma se obtuvo aplicando una señal de entrada de 1Vpp. El generador se programó en modo de barrido (sweep mode) con frecuencia de inicio de 9Hz y frecuencia final de 900Hz; el tiempo del barrido es de 800ms y es de tipo logarítmico, entonces en la grafica cada década será representada cada 3.3 divisiones, si la base de tiempo del osciloscopio es de 100ms.

La Figura 141 muestra una gráfica de respuesta a la frecuencia donde se puede apreciar que la frecuencia de corte es de 90Hz.

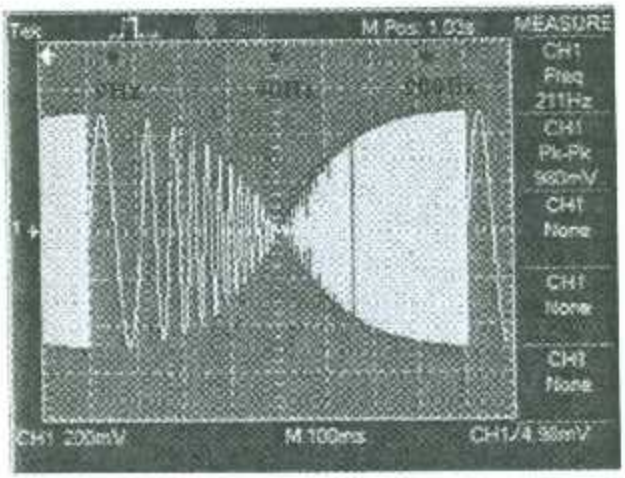


Figura 140. Respuesta obtenida en físico.

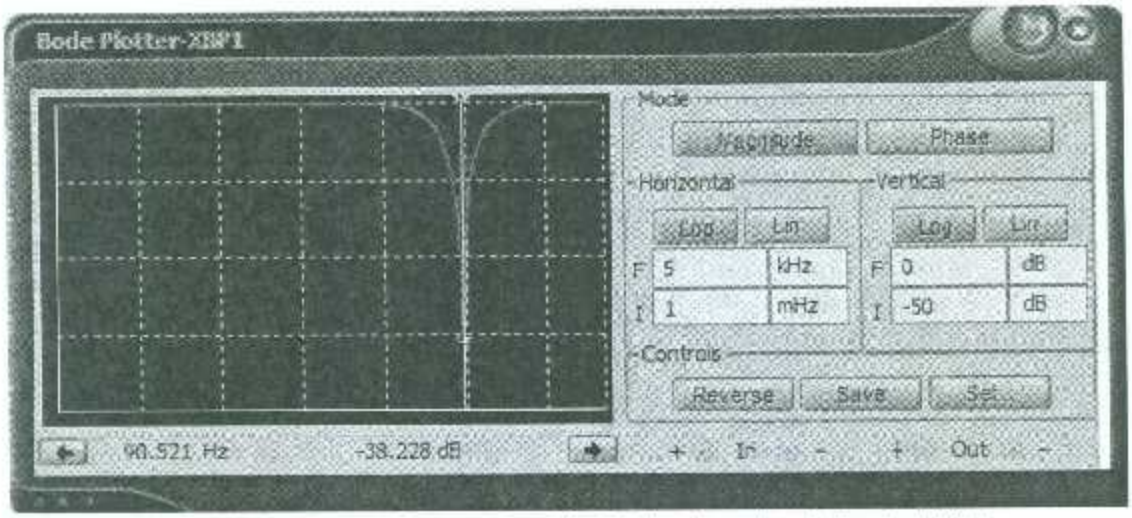


Figura 141. Respuesta a la frecuencia teórica del filtro de rechazo de banda de 90 Hz.

REFERENCIAS.

[1] R. Gupta, J. B. (2010). Development of an embedded system and MATLAB-based GUI for online acquisition and analysis of ECG signal. Measurement , 1119–1126.

[2] M. en C. David García Mora, (2010). Curso Tratamiento de Bioseñales, Unidad 7 Electrocardiografía.

[3] Microchip PIC16F87A Data Sheet 2003

[4] Matlab. (2012). Recuperado el Febrero de 2012, de matlab www.mathworks.com

[5] Interfaces Gráficas MATLAB Salvador García Bernal, Miguel Sandino Parra y David Báez López

[6] Octave, G. (s.f.). GNU Octave. Recuperado el Septiembre de 2013, de http://www.gnu.org/software/octave/

[7] Cygwin. (s.f.). Cygwin. Recuperado el Agosto de 2013, de http://www.cygwin.com/

[8] Python. (s.f.). Python. Recuperado el Octubre de 2013, de http://www.python.org/

[9] Instruments, N. (s.f.). National Instruments. Recuperado el Marzo de 2013, de http://www.ni.com/data-acquisition/what-is/esa/

[10] Oyarce, A. (2008). Guía de usuario XBee Series 1. En A. Oyarce, Guía de usuario XBee Series 1 (pág. 69). México.

[11] M. en C. David García Mora, (2010). Curso Tratamiento de Bioseñales, Unidad 6 Amplificadores de intrumentación y filtros activos.

[12] Mercador, P. S. (s.f.). Instalación de cygwin-X. Recuperado el Agosto de 2013, de http://www.uam.es/departamentos/ciencias/quimica/estruct/psm/documentacion/instalacion-cygwin-X.pdf

[13] Cifuentes, J. M. (s.f.). Manual de iniciación a GNU OCTAVE. Recuperado el Octubre de 2013, de http://softlibre.unizar.es/manuales/aplicaciones/octave/manual_octave.pdf

[14] Salvador García Bernal, M. S. (2013). Capítulo 7, Interfases Gráficas.

[15] Zenity. (s.f.). Zenity. Recuperado el Noviembre de 2013, de https://help.gnome.org/users/zenity/stable/index.html.es

[16] Duque, R. G. (2013). Python para todos.

[17] Tkinter, P. (s.f.). Python Tkinter. Recuperado el Noviembre de 2013, de http://www.tutorialspoint.com/python/python_gui_programming.htm

[18] MedlinePlus. (s.f.). MedlinePlus. Recuperado el Abril de 2014, de http://www.nlm.nih.gov/medlineplus/spanish/ency/article/003399.htm

[19] Biolaster. (s.f.). Biolaster. Recuperado el Abril de 2014, de http://www.biolaster.com/productos/pulsometros_polar/variabilidad_frecuencia_cardiaca

[20] Gil Rodas, J. R. (2008). Variabilidad de la Frecuencia Cardiaca: Concepto, Medidas y Relación con Aspectos Clínicos. Archivos de Medicina del Deporte , 41-47.