UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMA VIRTUAL PARA ENTRENAMIENTO MIOELÉCTRICO DE PRÓTESIS DE MIEMBRO SUPERIOR

TESIS

QUE PARA OBTENER EL TÍTULO DE

INGENIERO MECÁTRÓNICO

PRESENTA:

JUAN MIGUEL RAMÍREZ ROCAMORA

DIRECTOR DE TESIS:

DR. JESÚS MANUEL DORADOR GONZÁLEZ

MÉXICO D.F. **AGOSTO DE 2011**

Reconocimientos

Se agradece el apoyo de mi asesor el Dr. Jesús Manuel Dorador González. Sin su motivación y guía este trabajo no se hubiera concluido.

Se agradece el apoyo de mis sinodales el Ing. Mariano García del Gallego, el Ing. Yair Bautista Blanco, la Mtra. Rosa Itzel Flores Luna yla Ing. María del Socorro Armenta Servín por su asesoría y apoyo en la revisión de este trabajo.

Se agradece el apoyo de los proyectos PAPIIT IN110809-3 “Diseño y caracterización de prótesis de miembro inferior y superior” y del proyecto IXTLI IX100510 “Inmersión en realidad virtual para la obtención de parámetros biológicos en el diseño e implementación de prótesis de miembro superior”.

ÍNDICE

Introducción

Los avances en la ingeniería biomédica han logrado el desarrollo de las prótesis de miembro superior con características asombrosas. Muchos veteranos de la armada de Estados Unidos gozan ya de prótesis mioeléctricas complejas que permiten movimientos relativamente naturales de la mano. Podemos apreciar la complejidad de las prótesis en los recientes prototipos del "DEKA Arm" el dispositivo "i-Limb".

La existencia de dispositivos comerciales de alta tecnología como los mencionados nos lleva a pensar, como en muchas otras áreas, que la tecnología de las prótesis mioeléctricas simples está ya al alcance de muchos investigadores. En efecto, el desarrollo de una tarjeta de adquisición para señales mioeléctricas, como se explicará en este trabajo, puede ser desarrollado a muy bajo costo.

Ya hay en nuestro país algunos proyectos dedicados al diseño de prótesis y al análisis de señales mioeléctricas. Por lo tanto, podemos considerar que muchas más personas en un futuro cercano podrán gozar de los beneficios de prótesis de bajo costo en México. No obstante, uno de los principales problemas de las prótesis consiste en la necesidad de ser desarrollados para adecuarse a las características de usuarios específicos, lo cual aumenta su tiempo de entrega. Pese a que el usuario ya necesite la prótesis suele tener que esperar un tiempo considerable y posteriormente aprender a usarla.

Con el presente trabajo buscamos desarrollar una aplicación simple, con la cual el usuario aproveche el tiempo de entrega de la prótesis entrenándose a usarla de manera interactiva.

Objetivos

Diseñar e implementar un sistema virtual de entrenamiento mioeléctrico para prótesis de miembro superior. Diseñar una tarjeta de captura simple para emular el comportamiento de las prótesis comerciales.

Alcances

El presente proyecto logró implementar una aplicación para el sistema operativo Windows que presenta una simulación de prótesis tridimensional. Se diseñó una tarjeta de adquisición para el funcionamiento del presente proyecto.

Se utilizó el lenguaje de programación C/C++ usando la librería de gráficos de OpenGL para la aplicación. Se utilizó lenguaje ensamblador para PIC para el firmware de la tarjeta de adquisición. Se utilizó el programa de software libre Blender para adecuar los modelos de la prótesis.

El sistema se utiliza de la siguiente manera. Se deben colocar los electrodos y cables al usuario. Hecho esto se debe realizar las conexiones adecuadas a la tarjeta de adquisición y de esta a su alimentación y al puerto COM adecuado de la computadora que contenga la aplicación. Hecho esto se ejecuta la aplicación, en la cual se verá la prótesis virtual funcionando con base en los estímulos mioeléctricos generados por el usuario. Entrar a modo de entrenamiento puede ser necesario para una mejor operación.

Capítulo I: Estado del Arte de Prótesis y Entrenadores Mioeléctricos

En los últimos años se han desarrollado prótesis de miembro superior sumamente complejas. Tenemos como ejemplo el “DEKA Arm”, comúnmente llamado “Luke’sArm” en referencia a la prótesis de la película de la guerra de las galaxias. Se puede apreciar en la Figura 1.1 a un veterano de la armada de Estados Unidos usando la prótesis mencionada.

Figura1.1

Específicamente, para prótesis de mano se ha desarrollado recientemente la i-Limb, que permite tomar objetos con la mano, siendo está controlada con impulsos mioeléctricos. Una captura de esta prótesis siendo usada se presenta en la Figura 1.2.

Figura 1.2

En cuanto al entrenamiento mioeléctrico existen algunas compañías que ofrecen entrenadores simples. Lamentablemente, la mayoría de ellos tienen precios bastante elevados y el entrenamiento es muy simple sin posibilidades a expandir sus opciones. Esto es similar a lo que ocurre con el entrenador mioeléctrico para NI ELVIS.

El NI ELVIS es la Suite de Instrumentación Virtual de Laboratorio Educacional de National Instruments. Es una plataforma educativa para diseño y generación de prototipos basada en NI LabVIEW. Ofrece como sus complementos un entrenador mioeléctrico que permite al usuario controlar un servomotor por medio de señales mioeléctricas. Se puede apreciar en la Figura 1.3 una captura de dicho sistema de entrenamiento mioeléctrico

Figura 1.3

Basados en estos dispositivos buscamos ahora diseñar un sistema que permita emular prótesis mioeléctricas simples, pero implementando cada paso para permitir su flexibilidad y posterior desarrollo.

En los capítulos siguientes se describirá la parte de captura y procesamiento mioeléctrico en primer lugar y la simulación y unión de estos temas en el último.

Capítulo II: Adquisición, Procesamiento y Comunicación de Información Mioeléctrica

Una parte esencial del presente proyecto es el diseño, o bien, el desarrollo teórico de una tarjeta de adquisición para señales mioeléctricas. Es decir, un dispositivo electrónico que permita ser conectado a los músculos a entrenar y comunique la señal generada por éstos a la aplicación desarrollada también en este trabajo.

Esta parte del proyecto consta de una serie de etapas principales que se describen en la Figura 2.1.

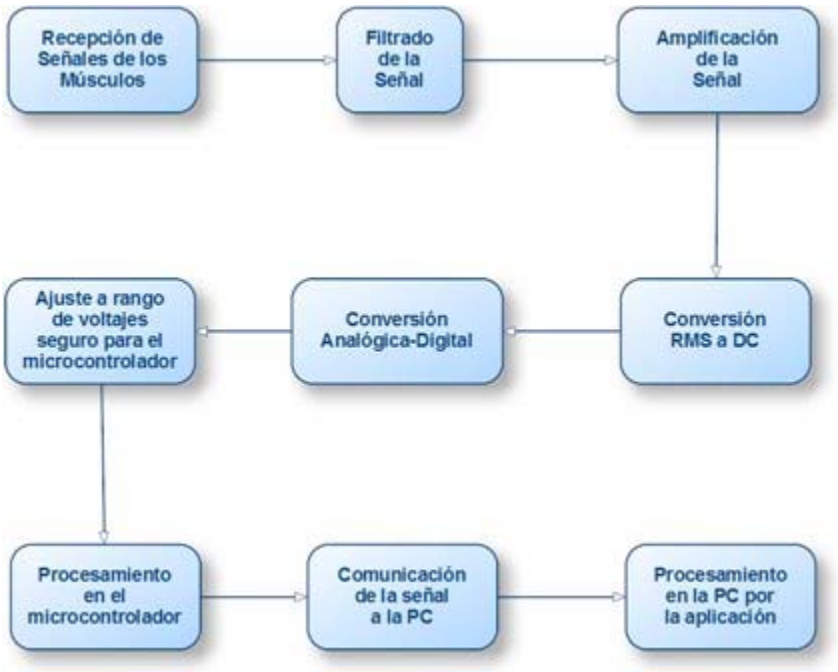

Figura 2.1

En los siguientes apartados se discutirán las etapas mencionadas. No obstante, antes de adentrarnos en el desarrollo de estos pasos es preciso revisar qué constituye una señal mioeléctrica y tener sus características principales presentes.

Señales Mioeléctricas

Las fuentes de prácticamente toda señal bioeléctrica endógena son cambios transitorios en el potencial transmembranal observado en toda célula viva. En particular, las señales bioeléctricas surge de los potenciales transmembranales variantes en el tiempo que se observan en células nerviosas (potenciales de acción y generación neuronales) y en células musculares, incluyendo al corazón. El fundamento electroquímico para los potenciales transmembranales en células vivas se conforma de dos fenómenos: (a) las membranas celulares son semipermeables, es decir, tienen diferentes conductancias y permeabilidades transmembranales para diferentes iones y moléculas, y (b) las membranas celulares contienen bombas de iones controladas por energía metabólica como ATP. Las bombas de iones transportan activamente iones y moléculas a través de membranas celulares contra barreras de energía dispuestas por el potencial transmembranal y/o gradientes de concentración entre el interior y exterior de la célula. En el estado estable, existen fugas continuas de iones hacia el interior de la célula (Na+) o hacia su exterior (K+) y el bombeo constante restablece las concentraciones del estado estable.

El conocimiento de las concentraciones es importante para realizar un estudio avanzado del sistema neuromuscular. Existe información de concentración iónica para las neuronas y músculos de una gran variedad de especies vertebradas e invertebradas.

Usando las concentraciones mencionadas es posible modelar el potencial transmembranal en estado estable utilizando la ecuación de Goldman-Hodgkin-Katz, que se muestra en la Figura 2.2.

$$V_{mo} = -\frac{RT}{\mathrm{F}} \ln\left\{ \frac{\left[\mathrm{Na}^+\right]_\mathrm{i} P_{\mathrm{Na}^+} + \left[\mathrm{K}^+\right]_\mathrm{i} P_{\mathrm{K}^+} + \left[\mathrm{Cl}^-\right]_\mathrm{i} P_{\mathrm{Cl}^-}}{\left[\mathrm{Na}^+\right]_\mathrm{e} P_{\mathrm{Na}^+} + \left[\mathrm{K}^+\right]_\mathrm{e} P_{\mathrm{K}^+} + \left[\mathrm{Cl}^-\right]_\mathrm{e} \mathrm{P}_{\mathrm{Cl}^-}} \right\}$$

Figura 2.2

En la Figura 2.2, T es la temperatura en Kelvin; R es la constante de gas en MKS; F es el número de Faraday y Px es la permeabilidad para especies iónicas X. El potencial transmembranal de neuronas resultante,Vmo, varía según la especie, tipo de neurona medio iónico y temperatura; puede encontrarse entre 60 y 90 mV. Las fibras musculares, tienen un potencial transmembranal de aproximadamente 80 <Vmo< 95 mV.

Expuesto esto podemos ahora analizar los potenciales de acción nerviosa y muscular. No obstante, Northrop ofrece una explicación concisa para los potenciales de acción nerviosa, por lo que nos concentraremos en su complemento muscular.

Una señal bioeléctrica que tiene gran importancia diagnóstica para muchas enfermedades neuromusculares es el electromiograma, que puede ser grabado de la superficie epitelial con electrodos idénticos a aquellos utilizados por la electrocardiografía, aunque en algunos casos, los electrodos tienen áreas menores que aquellos usados para ECGs. Para realizar registros de unidades motoras individuales (SMU) o incluso fibras musculares individuales, que suelen conformar SMUs, es posible el uso de electrodos de aguja que penetran la piel hasta el cuerpo un músculo superficial. Los registros EMG se utilizan para diagnosticar algunas causas de debilidad o parálisis muscular, problemas musculares o motores como tremores, daños de nervios motores, entre otros.

Hay varios tipos de músculos en el cuerpo, por ejemplo, estriado, cardiaco y liso. Los músculos estriados en los mamíferos se subdividen a su vez en músculos rápidos y lentos. Los músculos rápidos se usan precisamente para movimientos rápidos, incluyen los músculos extraoculares, músculos de la laringe, etc. Los músculos lentos se utilizan para control de postura contra gravedad e incluyen a los músculos abdominales, los de la espalda, los del cuello, etc. Los registros EMG se suelen aplicar a ambos tipos de músculos esqueléticos. Puede también realizarse en músculos menos superficiales como los músculos extraoculares que controlan el movimiento ocular, los músculos de los párpados y los que trabajan la laringe. Podemos apreciar en la Figura 2.3 un potencial de acción de un músculo de fibra única típico, registrado intracelularmente al final de la placa motora y 2 mm a lo largo de la fibra.

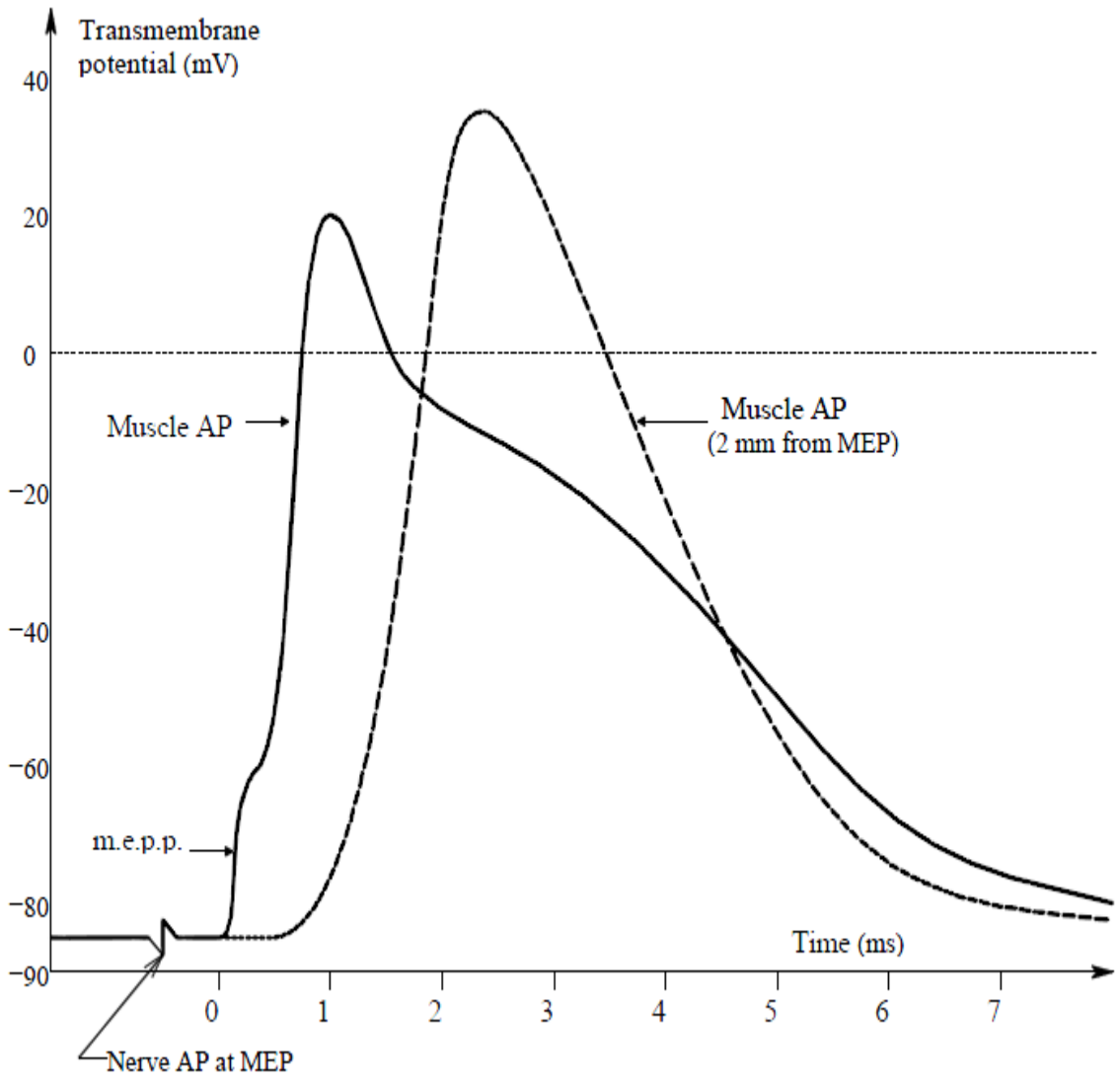

Figura 2.3

Recepción de las señales generadas por los músculos

Para la parte de recepción de la señales generadas por los músculos se utilizan electrodos pediátricos para monitoreo *micropore* de la marca 3M, utilizando cables utilizando cables broche con terminales R, B, N hembras de la marca Fiab. Esta elección se debe a la facilidad de uso que presenta la conexión entre electrodos y cables, la buena captura que se tiene con los electrodos por contener gel conductor, la

sencilla colocación de los electrodos por usar tecnología micropore para su adhesión al miembro deseado, el blindaje de los cables para reducir la interferencia eléctrica del medio con la señal adquirida, entre otras razones.

El electrodo es la interfaz existente entre el paciente y la instrumentación. La aplicación y uso correctos de los electrodos son los principales requerimientos para obtener buenas señales, aun así es común que esto esté mal implementado. Existen dos tipos principales de electrodos: los superficiales y los de aguja.

Figura 2.4

Los electrodos superficiales se utilizan para estudios de conducción nerviosa (NCS). Los electrodos son típicamente de anillo o de disco como se aprecia en la Figura 2.4. Los electrodos de este tipo no desechables están hechos de acero inoxidable, plata o en raras ocasiones de oro que se encuentra soldado a cables multifilares de conducción. Estos electrodos se suelen adherir a la piel usando cinta adhesiva y pueden ser reutilizados. Es preciso que sean limpiados al cambiar de paciente. Es también necesario utilizar gel conductor con electrodos no desechables para reducir impedancia y prevenir errores debido a irregularidades en la piel y la presencia de folículos. También se pueden usar electrodos de anillo como los mostrados en la Figura 2.5.

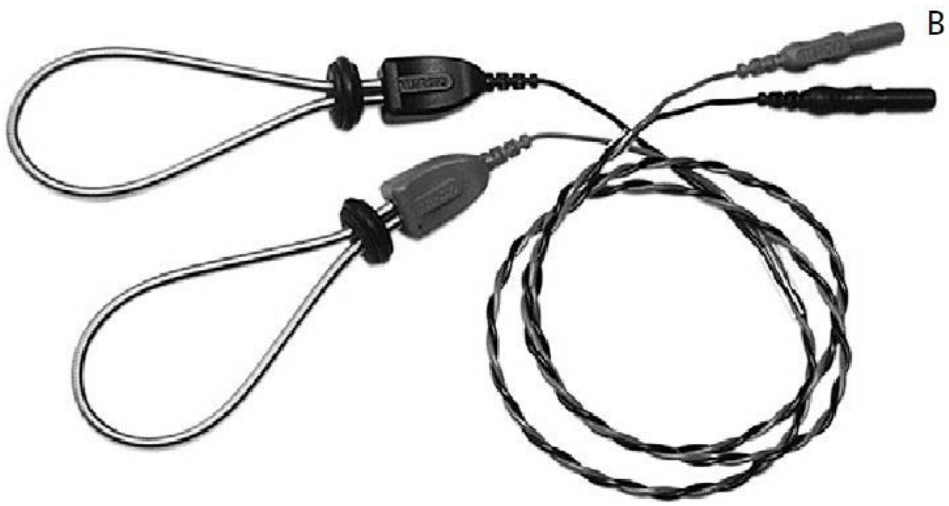

Figura 2.5

Los electrodos desechables suelen tener algún adhesivo en su parte inferior que los permiten aplicarse a la piel sin necesidad de gel o cinta. En la Figura 2.6 imagen se pueden apreciar el modelo de electrodo desechable usado para el presente proyecto.

Figura 2.6

Para NCS, como en nuestro caso, se utilizan tres electrodos superficiales electrodos de grabación activo y de referencia, además de un electrodo de tierra. En estudios electromiográficos, los electrodos superficiales se utilizan para la tierra y en ocasiones como el electrodo de grabación referencial. En la Figura 2.7 se puede apreciar un electrodo de aterrización.

Los electrodos de aguja suelen estar reservados para electromiografías aunque se utilizan ocasionalmente en NCS. Hoy en día prácticamente todos los electrodos de

aguja son desechables y se usan en un solo paciente. Una clasificación de éstos los separa en monopolares, bipolares y concéntricos.

Figura 2.7

Amplificación de las señales mioeléctricas

Los amplificadores son una parte muy complicada del equipo para electrodiagnóstico, aun así el concepto es bastante sencillo. Los amplificadores magnifican la señal de manera que pueda ser mostrada. La mayor parte de la amplificación utiliza circuitos integrados. Los preamplificadores atenúan la señal biológica antes de que esta llegue al amplificador principal para: a) asegurar que los filtros tengan una señal de suficiente voltaje para trabajar, y b) asegurar que el nivel de voltaje de la señal sea mucho mayor que el ruido del sistema. La señal viaja en primer lugar al preamplificador, posteriormente a los filtros y finalmente al amplificador. El amplificador diferencial es utilizado extensivamente en estudios para electrodiagnóstico debido a tener la ventaja

de rechazo en modo común. Esto significa que las señales no deseadas, en vez de ser amplificadas al mismo nivel que las señales biológicas que se busca estudiar, son rechazadas. La señal más común que se desea eliminar clínicamente es la actividad a 60-Hz, que se debe al paso de la línea de voltaje a través de circuitos eléctricos. Para nuestro objetivo otra señal que se elimina de este modo es la señal del pulso cardiaco. En la Figura 2.8 se puede ver un preamplificador comercial.

Figura 2.8

El amplificador diferencial toma los impulsos eléctricos del electrodo activo y los amplifica. Toma entonces los impulsos del electrodo de referencia, los invierte y amplifica. Hecho esto combina este par de potenciales. De esta manera, cualquier ruido común a los dos electrodos se elimina, esto incluye actividad eléctrica externa, ruido miogénico distante y ruido electrocardiaco. Aun así, las diferencias entre los dos electrodos se verán amplificadas. Esta es la señal deseada. Cualquier factor común, tal como ruido externo se rechaza llevando al término *rechazo en modo común*. La razón de rechazo a modo común (CMRR) es una medida de qué tan bien el amplificador elimina este tipo de ruido común. Esto se ilustra en la Figura 2.9.

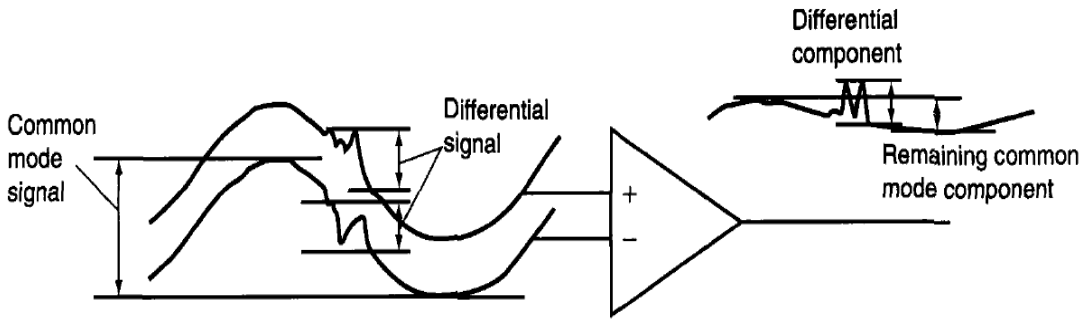

Figura 2.9

Cabe mencionar que una manera adicional de mejorar la señal mioeléctrica es aterrizando al paciente. Al unir al paciente a una tierra real, la interferencia de 50- ó 60-Hz puede reducirse en 10 o hasta en 100 veces. La tierra real no es una referencia para cada uno de los amplificadores diferenciales; mejor dicho, es usada para drenar el exceso de voltaje a modo común. El voltaje inducido en el cuerpo se acopla capacitivamente. Se induce al acoplar entre el área expuesta del cuerpo, el dieléctrico del espacio libre y el área del cable, lámpara o aquel equipo que tenga altos voltajes de ruido. La tierra forma una desviación de corriente de baja impedancia para estas señales. Aun así, conectar el cuerpo a una tierra real es un peligro potencial para el paciente y *no* se recomienda como una manera de reducir interferencia a modo común. En efecto, los equipos de instrumentación modernos conectan al paciente a una tierra aislada, en vez de a la tierra real, eliminando así el riesgo de descarga eléctrica. En la Figura 2.10 se puede apreciar un sistema de cómo se debe aterrizar y aislar dispositivos biomédicos.

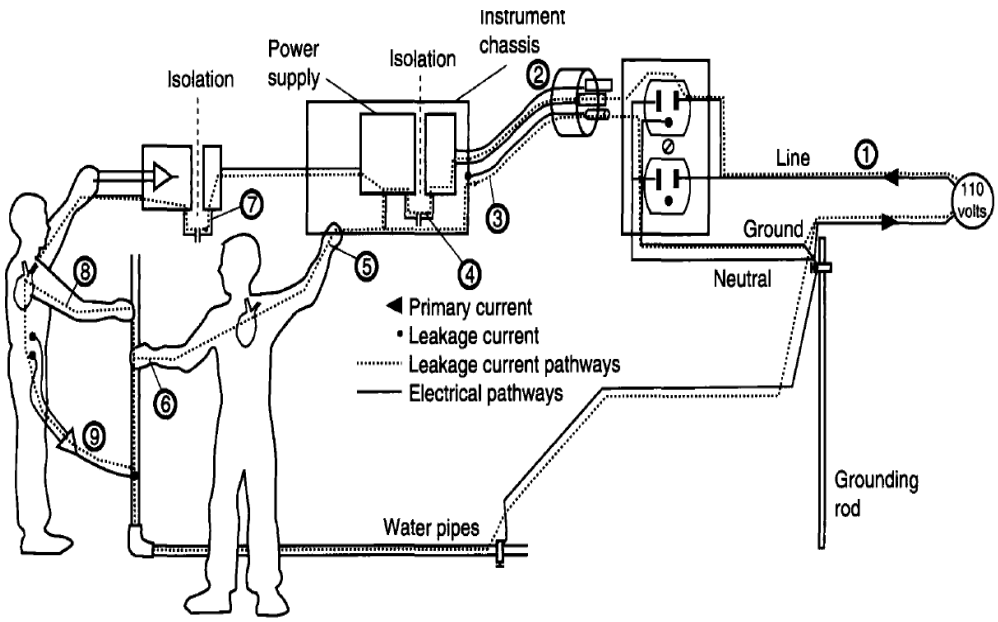

Figura 2.10

Un aspecto muy importante a considerar es aterrizar los instrumentos. El chasis del instrumento electrofisiológico se conecta a una tierra real por medio de un cable de tierra en el cable de alimentación. Si el enchufe no está debidamente aterrizado, o si la línea de tierra está rota a lo largo de su camino del instrumento a la tierra, se crea una conexión de alta impedancia. El ruido del instrumento será excesivo y el ruido generado por otros dispositivos regresará al instrumento y se acoplará a los electrodos.

Una manera más segura de reducir el efecto del voltaje a modo común, y así reducir la necesidad de un CMRR alto e impedancias de electrodos balanceadas, es "flotar" el amplificador de la tierra real y conectarlo únicamente al paciente. El circuito de amplificación conducirá eléctricamente el voltaje a modo común en el cuerpo. La medición usada para describir el rechazo de ruido a modo común es la razón de rechazo a modo de aislamiento y típicamente se encuentra sobre los 100 dB. El aislamiento requiere acoplar la alimentación, las señales y las líneas de control a través de una barrera de muy baja capacitancia. Se suelen utilizar transformadores, optoacopladores y acopladores capacitivos usando modulación de frecuencia, modulación de ancho de pulso o técnicas de modulación lineal. Este aislamiento aumenta la seguridad del paciente porque las corrientes anómalas no pueden fluir a través del amplificador y el paciente.

En efecto, como se puede apreciar en la Figura 2.11,un acoplamiento capacitivo obliga a pequeñas corrientes inducidas a fluir a través del cuerpo. La corriente crea una caída de voltaje entre los electrodos, que es una señal diferente. Una mayor CMRR no tiene efecto en esta interferencia de 60-Hz.

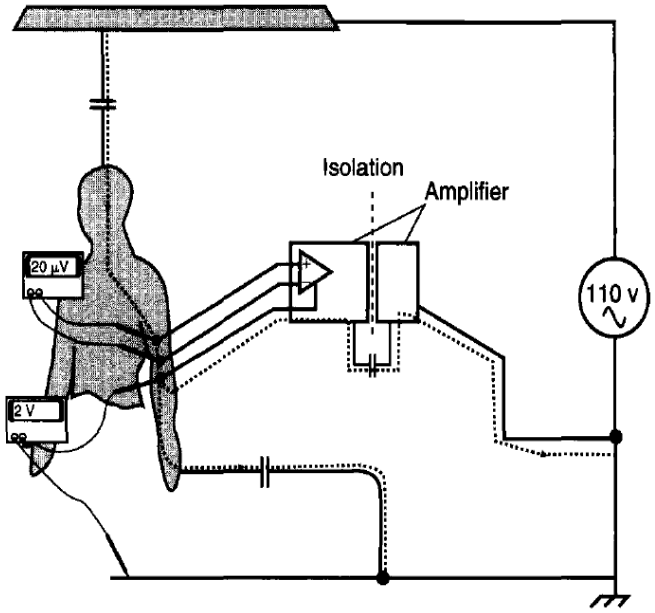

Figura 2.11

Un amplificador operacional sin retroalimentación por sí ya es un amplificador diferencial y amplifica la diferencia de voltaje entre las dos entradas. No obstante, su ganancia no puede controlarse y suele ser demasiado alta para ser de alguna utilidad práctica. Cuando uno comienza a estudiar el comportamiento de los circuitos con amplificadores operacionales, se suele sacrificar el potencial útil de una de sus entradas, usándose sólo una de las entradas. Con un poco de imaginación es posible construir un circuito con amplificadores operacionales que conserve ambas entradas de voltaje. Para realizar esto se debe usar un arreglo externos de resistores que controlen la ganancia. En la Figura 2.12 se puede apreciar un diagrama del circuito sugerido para este efecto.

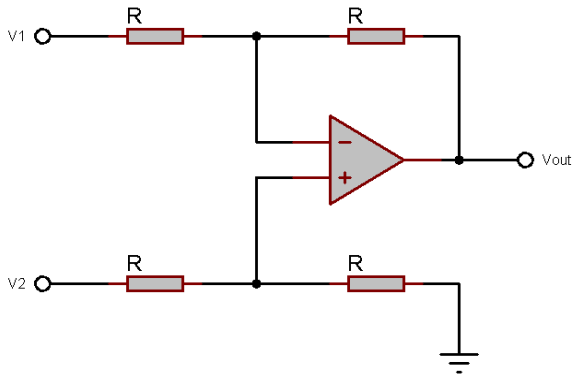

Figura 2.12

Si todos los valores de los resistores son iguales, este amplificador tendrá una ganancia de voltaje diferencial unitaria. Esto se puede demostrar si se analiza el circuito utilizando los principios de idealización de los amplificadores operacionales, las leyes de Kirchhoff y el principio de superposición. En efecto, el análisis de este circuito es esencialmente el mismo que el de un amplificador inversor, salvo que la entrada no-inversora del amplificador operacional se encuentra a un voltaje igual a una fracción de V2, en vez de estar conectado directamente a tierra. Como se puede deducir, V2 funciona como la entrada no-inversora y V1 actúa como la entrada inversora del circuito final.

Si deseáramos tener una ganancia diferencial diferente a la unitaria tendríamos que ajustar las resistencias en tanto los divisores de voltaje superiores e inferiores, requiriendo cambios de resistores múltiples y balancearlos entre los dos divisores para operación simétrica. Esto no suele ser muy práctico por claras razones.

Otra limitante de este diseño de amplificador es el hecho de que sus impedancias de entrada son relativamente bajas comparadas con las de otras configuraciones de amplificadores operacionales, especialmente el amplificador no inversor. Cada fuente de voltaje de entrada debe conducir corriente a través de una resistencia, que constituye menos impedancia que la de una entrada simple del amplificador operacional solo. La solución a este problema es suficientemente simple. Lo único que se necesita es aplicar un búfer a cada señal de voltaje de entrada a través de un seguidor de voltaje como se muestra en la Figura 2.13.

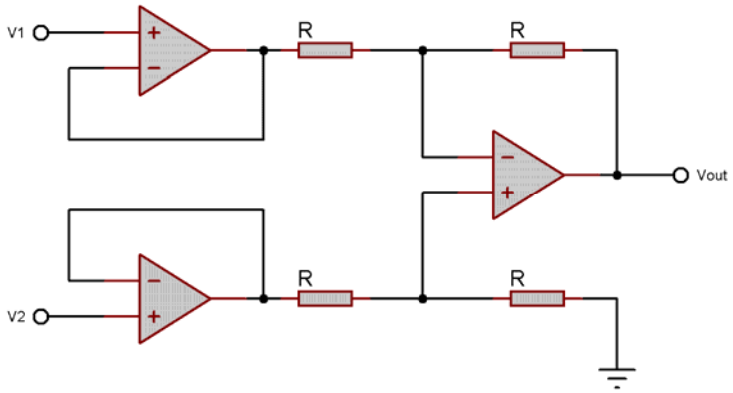

Figura 2.13

Ahora las líneas de entrada V1 y V2 se conectan directamente a las entradas de dos amplificadores operacionales en configuración de seguidor de voltaje, dándonos una impedancia muy alta. Los dos amplificadores operacionales de la izquierda se encargan de conducir la corriente a través de los resistores en vez de permitir que las fuentes de voltaje de entrada lo hagan. La mayor complejidad para nuestro circuito es mínima para un beneficio sustancial. Además, por medio de estas modificaciones se mejora sustancialmente el CMRR del amplificador original, subiendo de 90 dB hasta 100 dB, para el caso en que se utiliza el circuito integrado 741.

En efecto, si consultamos la hoja de especificaciones para el amplificador AD620, que es el dispositivo que utilizamos para realizar pruebas de este proyecto observaremos la familiar imagen en la Figura 2.14.

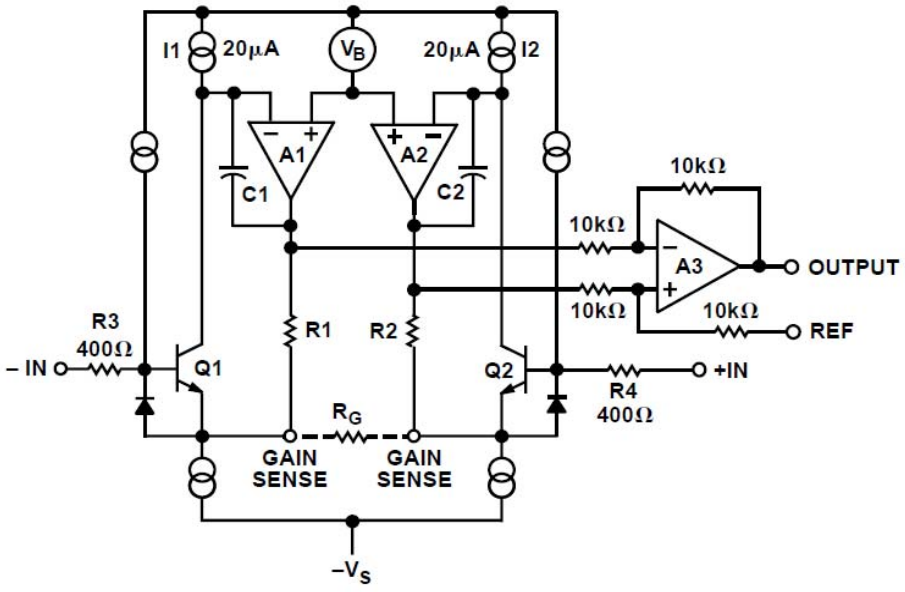

Figura 2.14

Como se puede apreciar, el diagrama simplificado para este circuito coincide con el amplificador diferencial discutido. Una de las primeras preguntas que nos podemos hacer al descubrir esto es en cuanto a la razón de comprar el dispositivo en vez de construirlo. Esto es porque el AD620 ofrece operación mejorada respecto a diseños de amplificadores de instrumentación con tres amplificadores operacionales, tiene menores dimensiones, menos componentes y una corriente de alimentación de la décima parte del diseño con 3 op-amps.

Se puede usar también el circuito integrado AD624, pero su precio es hasta 3 veces mayor que el del integrado AD620. Pese a que la calidad del AD624 es ligeramente superior a nuestra alternativa, ya hemos considerado que podríamos bien armar el circuito amplificador y aún tener una calidad de amplificación suficientemente buena para nuestros objetivos. Se puede apreciar en la Figura 2.15 que la funcionalidad interna del circuito AD624 es muy similar a la del AD620 pero cuenta con opciones adicionales.

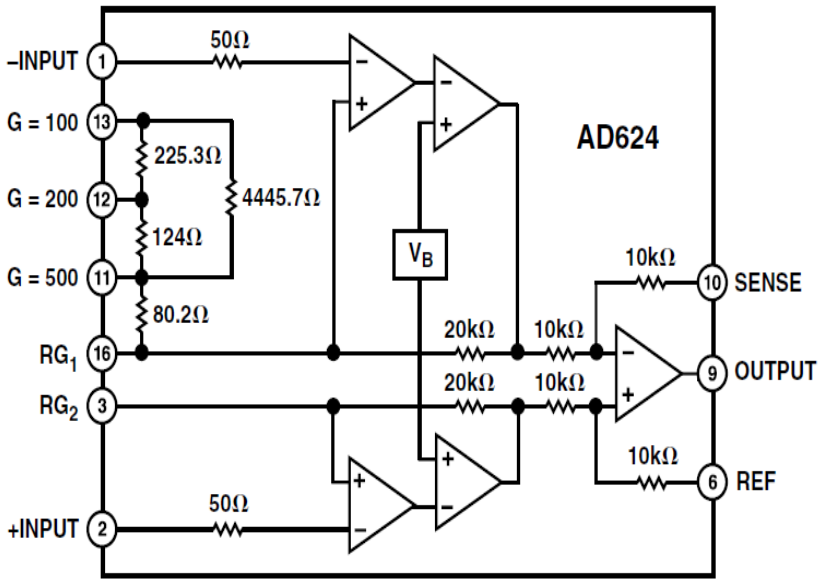

Figura 2.15

Filtrado de la señal amplificada

Los filtros se utilizan para reproducir fielmente la señal que se desea mientras se trata de excluir tanto ruido eléctrico de alta como de baja frecuencia. Toda señal para estudio mioeléctrico pasa a través de un filtro pasobajas y uno pasoaltas antes de mostrarse. Los filtros de baja frecuencia se llaman pasoaltas por dejar pasar solo estas señales y los filtros de alta frecuencia se llaman pasobajas, por razones similares. El rango en el que hay un corte de señales de baja frecuencia depende de cómo se configure el filtro. Es importante comprender que siempre al utilizar filtros la señal que se desea será alterada en cierto grado. Por ejemplo, a manera de que el filtro de baja frecuencia es reducido más señales de frecuencia baja pasarán a través de él y la duración del potencial grabado será un poco mayor. En la Figura 2.16 se muestra un diseño simple de un filtro pasoaltas que puede ser fácilmente analizado.

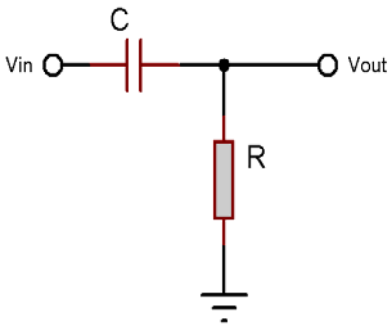

Figura 2.16

Para hacer del circuito descrito suficientemente funcional para nuestros propósitos es preciso agregar un seguidor de voltaje con un amplificador operacional para desacoplar impedancias. En el diagrama mostrado en la Figura 2.17 se puede apreciar un filtro activo paso altas de segundo orden. Este filtro tiene una mejor funcionalidad y estabilidad que el filtro de la Figura 2.16.

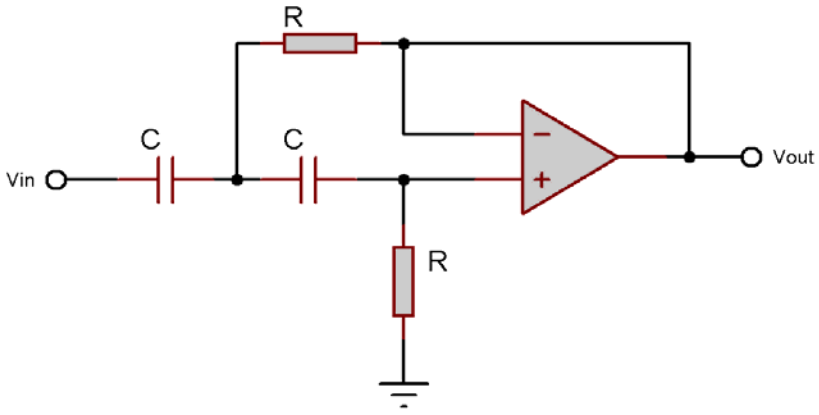

Figura 2.17

Conversión RMS-DC de la señal

Un convertidor analógico de RMS verdadera es un sistema que proporciona una salida de corriente directa proporcional a la raíz-cuadrada-media de la señal de entrada. Una operación de RMS analógica primero toma la segunda potencia del voltaje de entrada, luego estima el valormedio del voltaje consecuente, generalmente promediándolo en el tiempo por medio de filtros pasobajas. Finalmente, toma la raíz cuadrada del valor medio cuadrado. En otras palabras, es una medida estadística de la magnitud de una cantidad variante. Es especialmente útil cuando las variaciones son positivas y negativas. En la Figura 2.18 podemos apreciar una gráfica que describe el resultado de aplicar un cálculo de RMS a una señal senoidal.

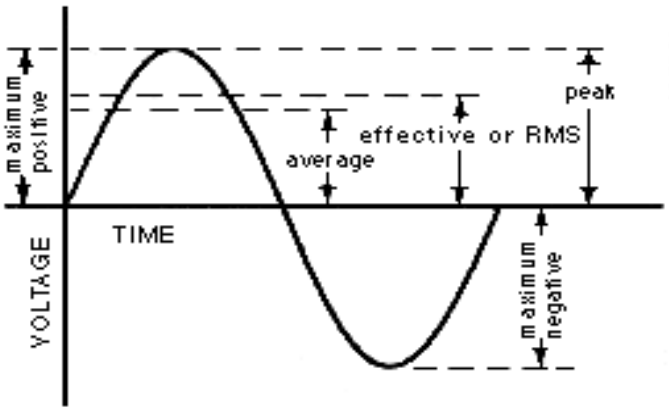

Figura 2.18

Podemos utilizar un circuito que calcule el valor RMS de la señal capturada para obtener valores de corriente directa que nos indiquen la magnitud de la señal. Esto lo podemos usar directamente para clasificar la señal entrante en el estado de relajación muscular o en el estado de activación con solo ubicar la señal en uno de los dos niveles posibles de voltaje. En la figura 2.19 se puede observar una propuesta de un circuito que calcule el valor RMS de una señal. No obstante, existen circuitos comerciales si se busca tener una tarjeta compacta y se cuenta con el presupuesto.

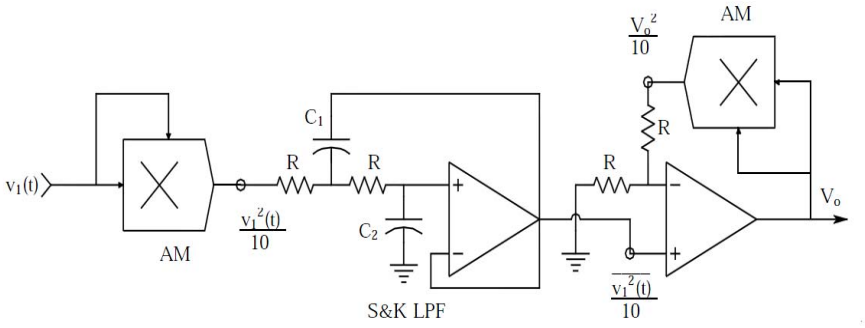

Figura 2.19

El circuito integrado que se sugiere para dicha conversión es el AD536A de AnalogDevices, ya que su desempeño es comparable o superior a unidades híbridas o modulares de mucho mayor costo. El AD535A calcula directamente el valor RMS verdadero de cualquier onda de entrada compleja constituida por componentes tanto de AC como de DC. Cabe mencionar que una característica importante del AD536A no contenida en convertidores anteriores es tener una salida dB auxiliar. El logaritmo de la señal RMS de salida se conduce a un pin separado para permitir la conversión de dB, con un límite dinámico de hasta 60 dB. Otra de las ventajas de este dispositivo es la necesidad de un solo componente externo para su operación, que es el capacitor que configura el periodo a promediar. En la Figura2.20 se muestra el diagrama de funcionamiento del integrado tomado de su hoja de especificaciones.

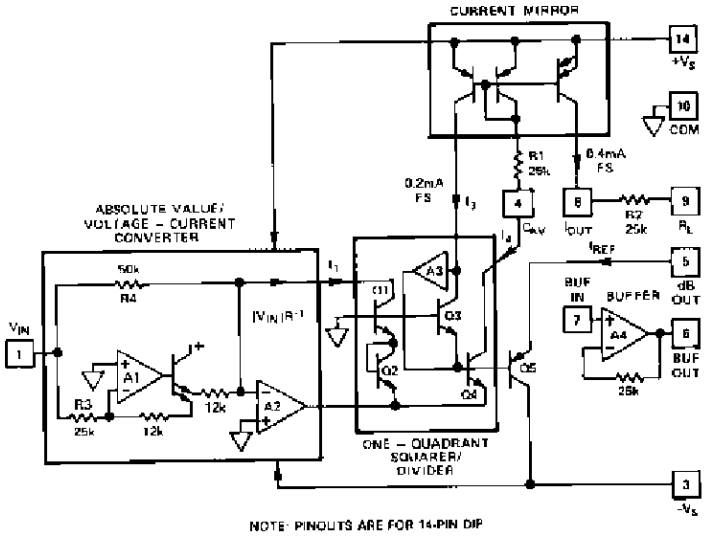

Figura 2.20

Aunque es el dato resultante que utilizaremos, el valor RMS a DC es sólo una opción de muchas para visualizar la señal mioeléctrica. La información mioeléctrica puede también analizarse en el dominio del tiempo, en el dominio de la frecuencia, o en el dominio tiempo-frecuencia.

El registro de señales mioeléctricas en el dominio del tiempo es particularmente útil al registrar fibras individuales o SMUs y una manera de obtener estas señales siguiendo lo más posible los pasos efectuados en este proyecto sería omitir la conversión de RMS a DC y en su lugar utilizar un circuito sujetador para ajustar la posición de la señal, o bien, considerar un voltaje de referencia negativo para el convertidor analógico digital, centrando la señal a capturar en el rango permisible del convertidor.

En el caso del registro de las señales en el dominio de la frecuencia la Transformada Rápida de Fourier (FFT) se toma de una electromiografía superficial completa bajo condiciones estándar. El implementar un sistema que nos permita visualizar la señal de este modo es relativamente complejo y requiere de un dispositivo para el procesamiento digital de señales (DSP). Podríamos bien utilizar un dsPIC para implementar la operación de FFT por hardware, no obstante esta opción se sugiere como trabajo futuro de este proyecto.

Ajuste de la señal a un rango de voltajes seguro para el microcontrolador

Hasta este punto se ha considerado la captura de la señal mioeléctrica, su prefiltrado, amplificación y su conversión a valores de corriente directa sea de RMS o por rectificación. Cabe mencionar que para la conversión de la señal analógica a digital utilizaremos el convertidor disponible en el PIC16F887 por las razones descritas en la siguiente sección .

Pese a las ventajas que tiene el uso de un microcontrolador para dicha conversión, la diferencia de voltajes máxima de la señal analógica que éste soporta es de 5 V, lo que nos obliga a realizar una calibración cuidadosa del circuito amplificador para nunca

rebasar estos voltajes. No obstante, con el fin de proteger al circuito es necesario el uso de un circuito recortador.

Un circuito recortador es aquel que impide a la salida de un circuito exceder un nivel de voltaje predeterminado sin distorsionar la parte restante de la onda de salida. Es una tarea simple implementar este tipo de circuitos y nos ofrece muchas ventajas de seguridad. Basta con colocar el diodo zener de manera que actúe en corto circuito hacia tierra en caso de sobrepasarse el valor máximo deseado, limitando así la salida al valor de corte del diodo zener. Esto se ilustra en la Figura2.21.

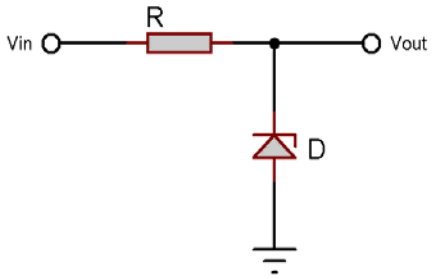

Figura 2.21

Pese a que sería deseable cortar la señal al máximo de 5 V, los valores comerciales típicos cercanos a éste son 4.7 V y 5.1 V. Por cuestiones de seguridad es preferible utilizar un diodo zener con voltaje de corte de 4.7 V. Una posible elección es el diodo 1N5230, identificador de diodos zener de 4.7 V utilizado por diversos proveedores.

Considerando el caso de utilizar un convertidor analógico-digital externo que soporte una diferencia de voltajes mayor podríamos consultar un catálogo de diodos zener para localizar el identificador de aquel que tenga el voltaje de corte inferior más cercano al rango permisible por el convertidor.

Conversión de información analógica a digital

La conversión analógica-a-digital requiere un circuito que pueda "congelar" la señal durante algunos microsegundos (S/H, Sample and Hold) y un circuito que convierta la amplitud de la señal "congelada" a un valor digital (ADC, Convertidor Analógico Digital).

Ocasionalmente estos circuitos se implementan en un mismo dispositivo, por lo que se les referirá colectivamente como el ADC. El ADC se especifica por su velocidad de conversión y su resolución (número de bits).

Si deseáramos tener una captura muy rápida podríamos optar por utilizar un convertidor analógico-digital especializado. No obstante, como no buscamos capturar un gran número de señales nos basta utilizar un convertidor que nos permita tener suficiente información para poder actualizar la simulación que correrá enlacomputadora en tiempo real. El convertidor analógico-digital contenido en el microcontrolador PIC16F887 es de hasta 10 bits y sus tiempos de adquisición no suelen superar las decenas de microsegundos, lo cual es más que suficiente para nuestros objetivos.

Otra de las ventajas de utilizar el convertidor contenido en el microcontrolador es la reducción de las dimensiones de la tarjeta de adquisición debido a ser el microcontrolador ya un elemento indispensable. Cabe mencionar también la reducción de costo por razones semejantes a las expuestas.

Aunque un convertidor de 10 bits puede ser de suma importancia para adquirir una señal, debemos considerar que nuestro propósito es capturar una señal con suficiente resolución para distinguir entre un estado de relajación del músculo a entrenar y un estado de tensión en tal músculo. Por lo tanto, hemos optado por utilizar el convertidor del microcontrolador en modo de 8 bits. De esta manera aumentamos la eficiencia del código del microcontrolador, disminuimos el tiempo de adquisición en el convertidor y obtenemos una señal con menor ruido. En efecto, si la señal de entrada no se ha filtrado correctamente, el hecho de disminuir la resolución de captura funciona como un filtro simple para "promediar" la señal adquirida.

Ahora bien, considerando los resultados de las pruebas realizadas, dependiendo de la amplificación configurada, los valores digitales obtenidos en un estado de relajación fueron de 12 y en un estado de tensión de 36, en promedio. Considerando que nuestra referencia de voltaje para estas pruebas fue de +5V con rangos de conversión con valores de 0 a 255, podemos por interpolación lineal aproximar un valor de 235 mV para el estado de relajación del músculo y de 706 mV para el estado de tensión del músculo.

Podemos apreciar que pese a haber tenido una amplificación aparentemente poco adecuada para el procesamiento, considerando sus bajos valores una vez realizada la digitalización y el notable error de *offset*, aún nos es posible utilizar estos valores para discriminar entre estados de manera satisfactoria. Esta es ventaja que debemos a utilizar procesamiento por software, ya que implementar un sistema flexible, que entrene para cada estado deseado y logre clasificar cada señal usando esta información, es una tarea trivial para un programador.

Procesamiento de la información digital en un microcontrolador

La sección digital suele consistir de tres partes principales: procesador, memoria y promediador. El procesador podría considerarse el "cerebro" del instrumento: coordina el flujo de datos y las funciones de interfaz. Los procesadores se clasifican por el número de bits procesados en paralelo y por la velocidad de procesamiento. La memoria se usa para almacenar las instrucciones para el procesador y las señales digitalizadas. La cantidad de memoria se expresa en bytes. El promediador suma y escala respuestas sincronizadas de señales para mejorar la razón señal-contra-ruido y puede ser implementada por el procesador.

Existen muchas ventajas al utilizar circuitos digitales. Las operaciones sobre señales digitales son muy precisas. Sumar dos señales analógicas arroja un resultado con un porcentaje de error, y los errores son acumulativos. Dos señales digitales sumadas siempre darán de manera precisa el mismo resultado. El chip o los componentes sumen dos señales analógicas pueden tener alteraciones en el resultado debido al tiempo, temperatura, humedad, fuente de alimentación y otros factores. Es posible que requieran de calibración o compensación, y en ocasiones no pueden siquiera funcionar. El chip que sume dos señales digitales siempre dará exactamente la misma respuesta y es insensible a su medio. Los sistemas digitales eliminan las alteraciones analógicas con tiempo y temperatura, y eliminan la necesidad de recalibrar. Los valores analógicos componentes pueden variar, pero los coeficientes digitales son absolutos. Un capacitor con una tolerancia absoluta del 0.1% en un filtro es un componente muy raramente

utilizado, pero un sistema digital de 16-bits tiene una precisión absoluta del 0.001%, es sencilla de implementar, es de bajo costo, y nunca cambiará con el tiempo o la temperatura. Toda unidad digital es exactamente como cualquiera de su tipo, por lo que su fabricación y caracterización se simplifica. Los sistemas digitales pueden realizar operaciones que no serían prácticas, y en ocasiones posibles, con sistemas analógicos. Cabe mencionar la ley de Moore, que se ha cumplido desde los inicios de los procesadores. Esta ley describe una tendencia a largo plazo en la historia del hardware. Que indica que el número de transistores que puede colocarse a bajo costo en un circuito integrando se duplica aproximadamente cada dos años. Esta tendencia se ha cumplido durante más de medio siglo y se ilustra en la gráfica de la Figura 2.22.

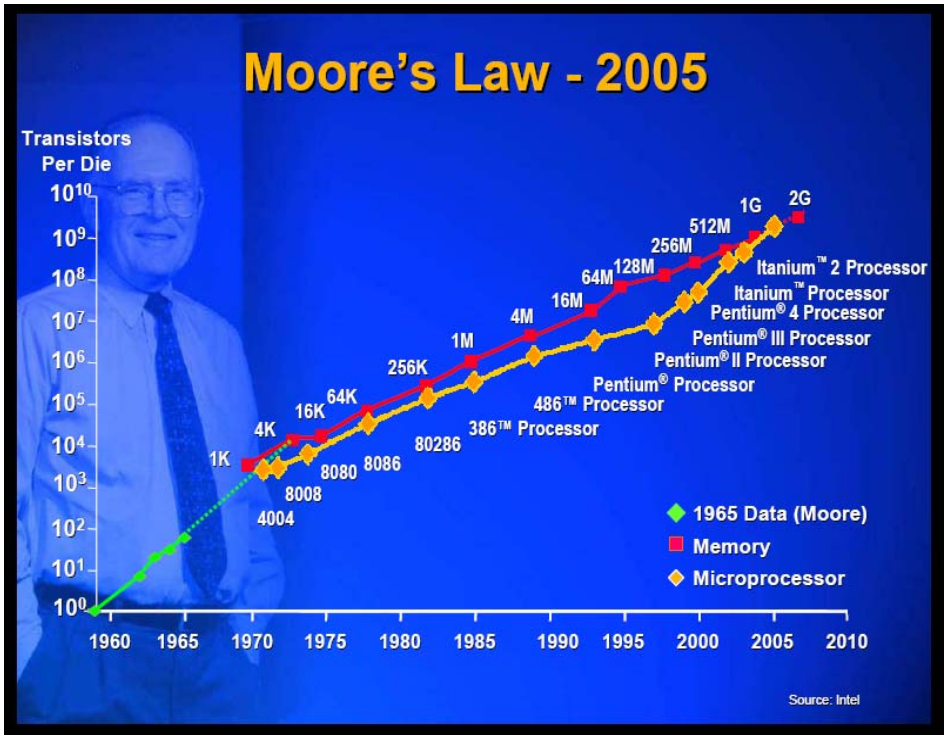

Figura 2.22

Las capacidades de muchos dispositivos electrónicos digitales se encuentran estrechamente relacionadas con esta ley. Entre ellas encontramos la velocidad de procesamiento, capacidad de memoria, sensores, entre otras.

Por todas estas razones, hoy en día son pocos los problemas o soluciones que no son más simples, baratos y confiables para resolver o implementar con sistemas digitales.

Es por esto que la mayoría de los instrumentos para electrodiagnóstico hoy en día rara vez utilizan filtros analógicos de y han sido reemplazados por filtros digitales. Los filtros digitales pueden duplicar a los analógicos, pero además pueden crear clases de filtros que no son fáciles de implementar usando componentes analógicos. Si se implementan correctamente, los filtros digitales multipolos pueden ser superiores a su equivalente analógico.

Comunicación con la PC y con el programa correspondiente

Una vez capturada la señal es indispensable comunicar los datos adquiridos a la computadora para que se pueda realizar la simulación interactiva que nos disponemos implementar en este proyecto. Ya hemos realizado la conversión analógica a digital a 8 bits de la señal, que nos ofrece una representación del valor obtenido de 0 a 255. Estos valores son más que suficientes ya que para distinguir de los dos estados posibles necesitamos únicamente un bit, es decir, 0 ó 1. No obstante, se utilizan datos de 8 bits para simplificar el diseño de la tarjeta por medio del uso de software. En efecto, el permitir tanta información nos permite utilizar una tarjeta de adquisición de muy bajo costo, lo cual resuelve uno de los principales problemas de sistemas de entrenamiento, el costo del hardware. Podemos recibir una señal con cierto offset para el estado de relajación y con la resolución aquí planteada nos es fácil separar la señal de aquellas que pertenezcan al estado de tensión muscular.

Usaremos el mismo microcontrolador utilizado para la conversión analógica-digital para la comunicación, en este caso un PIC16F887 de Microchip. La comunicación utilizará el protocolo serial RS-232 y tendrá un baudaje de 115200, la velocidad más alta para este tipo de comunicación soportada en la mayor parte de sistemas.

El microcontrolador fue programado en ensamblador, dada la baja complejidad del programa y la eficiencia del código escrito directamente en ensamblador. El código del microcontrolador se encuentra en el CD que acompaña a este trabajo y cada vez que se manda un carácter al microcontrolador, éste activa una interrupción por recepción serial

y responde a la computadora con dos valores, correspondientes al valor digital de la señal mioeléctrica.

Con este protocolo es fácil ver estos valores desde la mayoría de los monitores seriales para PC, como son el proporcionado por la interfaz para el *tinybootloader* o la interfaz para Arduino. También es relativamente sencillo implementar algún programa que permita capturar esta información dada la sencillez del protocolo de solicitud de datos. En efecto, en el siguiente capítulo se expone la sección de nuestro programa para PC que implementa comunicación serial con el microcontrolador.

Capítulo III: Sistema Virtual

Se desarrolló una aplicación que permite la interacción del usuario con una prótesis virtual por medio de impulsos mioeléctricos capturados por la tarjeta de adquisición descrita en el Capítulo II.

El programa podrá analizar las señales mioeléctricas capturadas por la tarjeta y comunicadas a la PC de entrenamiento por comunicación RS-232. El código puede modificarse posteriormente para soportar comunicación USB. Recibida la información se analizan las entradas y, con base en un entrenamiento inicial, se determina el estado de los músculos, se encuentren éstos en tensión o relajación. Considerando el estado se anima una prótesis tridimensional para mostrar la acción de agarre o de apertura de la mano. Para trabajo futuro se desea hacer el sistema flexible para poder utilizar diferentes modelos de prótesis virtuales.

La aplicación fue desarrollada en el lenguaje de programación C/C++ para ambiente Windows. Se utilizó la librería de gráficos OpenGL para permitir una visualización de gráficos rápida y atractiva. Para la compilación y desarrollo del código se utilizó el ambiente de desarrollo integrado Visual Studio 2008.

El desarrollo de aplicaciones con C/C++ para el sistema operativo Windows puede ser de dos tipos principales, considerando la clasificación respecto a su apariencia y a la manera en que operan con el sistema operativo. Los tipos más comunes son las aplicaciones de consola y las aplicaciones nativas de Windows.

Las aplicaciones de consola pueden considerarse aplicaciones nativas de MS-DOS. Estas aplicaciones suelen estar caracterizadas por un conocido estilo de caracteres blancos sobre un fondo negro, como la mostrada en la Figura 3.1.

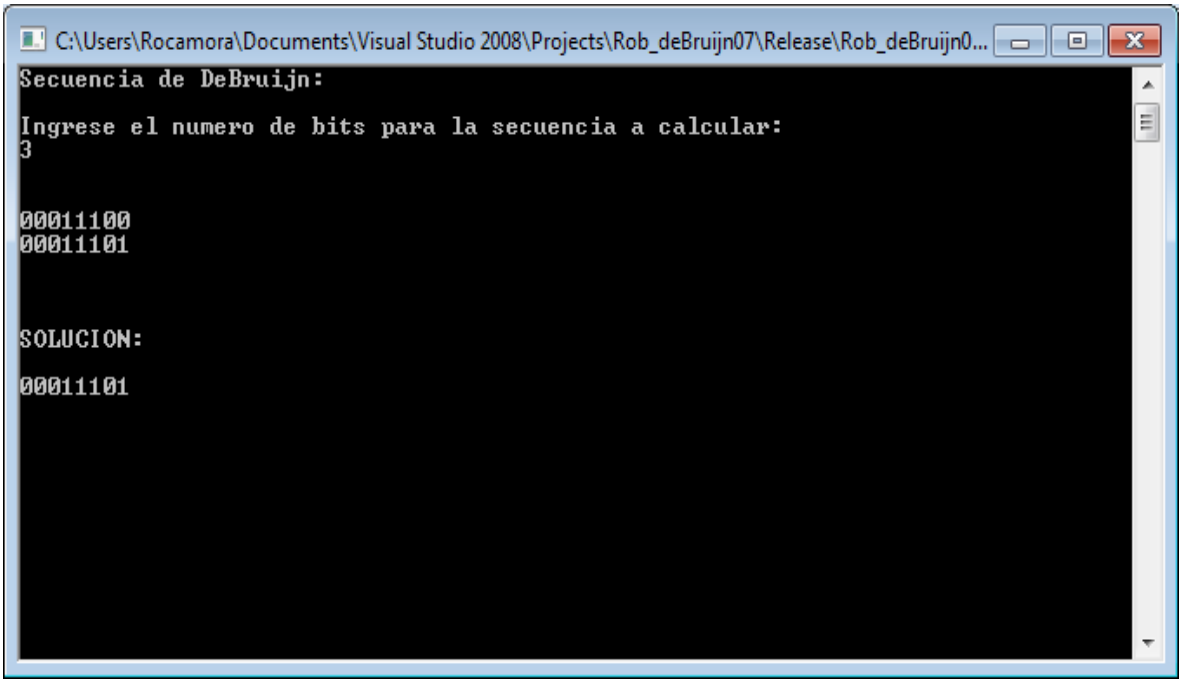

Figura 3.1

Una de las ventajas de este tipo de aplicaciones es la sencillez que tienen en su interacción con el usuario debido a que el único dispositivo de entrada suele ser el teclado. Además, cabe mencionar que la salida suele constar de texto. Existen muchas aplicaciones que muestran gráficos pero el desarrollo de éstas ha prácticamente desaparecido por ser mucho más atractivos y rápidos los gráficos que corren nativamente sobre Windows. La principal ventaja de este tipo de aplicaciones es la rapidez con la que se puede implementar algoritmos. Es decir, debido a la sencillez de la aplicación que se busca desarrollar, la cantidad de código necesario para obtener el resultado suele ser mínima a comparación de las aplicaciones nativas de Windows.

Por estas razones, pese al éxito que han tenido los sistemas operativos visuales como Windows, este tipo de aplicaciones siguen desarrollándose. Aún para proyectos nativos a Windows es común encontrar en el desarrollo una etapa de diseño en consola. En esta etapa se evalúa el comportamiento del algoritmo principal a usar antes de incorporar los elementos de visualización, interacción y comunicación en Windows.

Por el otro lado tenemos las aplicaciones nativas de Windows, que nos permiten aprovechar los diversos dispositivos de entrada que ofrece esta serie de sistemas operativos como el uso del ratón, *joysticks*, sensores como el Kinect, entre otros,

además de un excelente soporte para multitareas y gráficos más intuitivos como el uso de ventanas en nuestras aplicaciones como se muestra en la Figura 3.2.

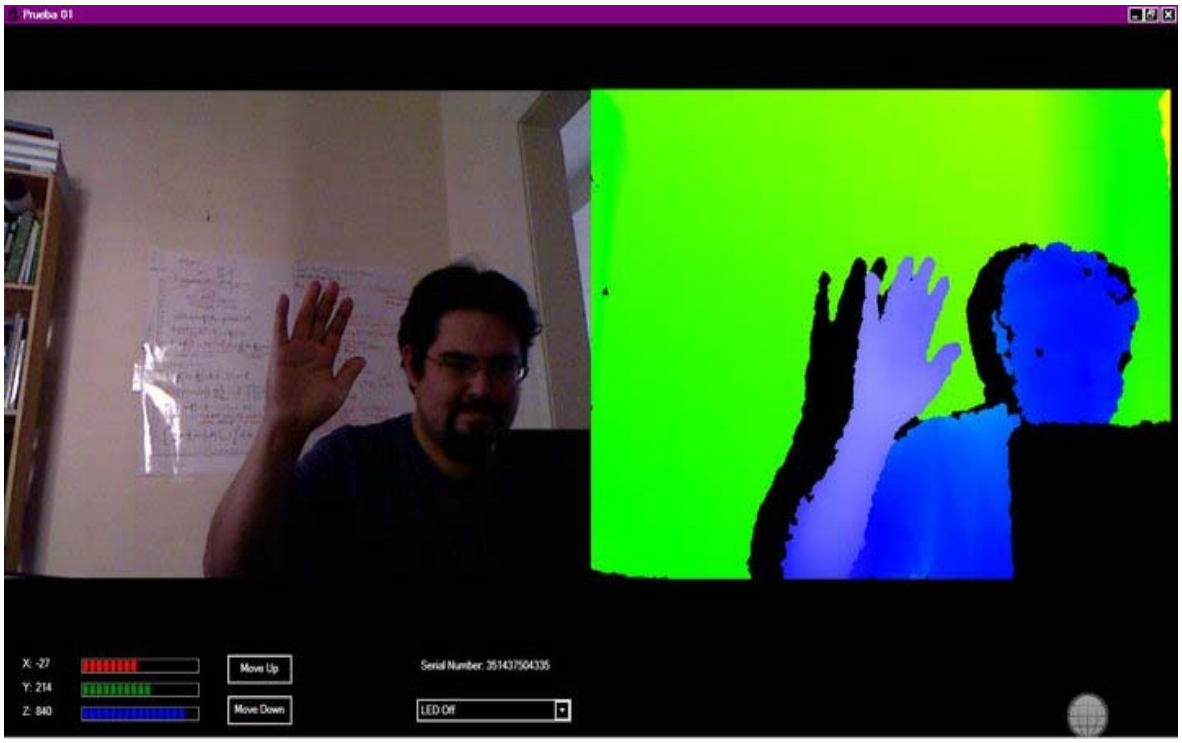

Figura 3.2

Como se busca hacer de la aplicación resultante la más atractiva e interactiva posible se ha optado por implementar los algoritmos en un programa nativo de Windows. Esto aumenta la probabilidad de que la aplicación funcione en versiones futuras del sistema operativo Windows sin necesidad de alterar el programa. Esto hará que la aplicación no sea obsoleta en sólo un par de años.

Desarrollo de aplicaciones en C/C++

Hace ya muchos años la única manera de desarrollar software era utilizando lenguaje ensamblador. El lenguaje ensamblador tenía una relación directa con las instrucciones básicas de la computadora pero auxiliaba al programador de tal forma que no tuviera que escribir dichas instrucciones en números hexadecimales, sino que utilizaba mnemónicos. Por ejemplo, para "mover" un número al acumulador se podía utilizar el

mnemónico MOV, El lenguaje ensamblador sigue usándose a la fecha, especialmente en la programación de microcontroladores y para optimizar rutinas de programas que requieran ser muy eficientes. Esto se hace principalmente porque al estar diseñado el programa considerando la estructura y arquitectura específica de la computadora se obtiene código óptimo, lo que hace de la velocidad de la implementación del algoritmo insuperable. Lamentablemente, hasta la programación de software de complejidad moderada es una tarea muy complicada, principalmente por la gran cantidad de código que se debe generar. Es por esta razón que comenzaron a surgir lenguajes de alto nivel.

Los lenguajes de alto nivel permiten programar aplicaciones de manera muy intuitiva para un humano. Una vez indicadas las instrucciones el compilador procede a convertirlas a lenguaje máquina para poder ser utilizadas por la computadora. La principal desventaja de este proceso es la calidad de la conversión. La conversión realizada es muy difícil que alcance la eficiencia de un programa diseñado para una arquitectura de computadora en particular por medio de lenguaje ensamblador. Como podemos ver, podemos aumentar la velocidad de desarrollo sacrificando la velocidad de operación de nuestra aplicación. Como buscamos desarrollar un sistema de simulación virtual que reciba impulsos mioeléctricos y los procese para efectuar diferentes acciones es indispensable manejar la mayor velocidad de operación posible. Afortunadamente, el lenguaje de programación C es considerado por muchos como un lenguaje de medio nivel, por ser el más eficiente para desarrollar después del lenguaje ensamblador.

La programación en C puede parecer una tarea obsoleta para muchas personas, debido principalmente a que, en muchos casos, el único acercamiento a este lenguaje que tienen los programadores es para desarrollar aplicaciones básicas. El desarrollo de estos programas suele usarse como un punto de partida para comenzar a programar, abandonando este lenguaje al poco tiempo para buscar desarrollar con lenguajes de alto nivel, como Java y C#. Es interesante notar que la mayoría de las aplicaciones de simulación, tanto en investigación como en videojuegos, suelen estar desarrolladas utilizando C o su versión orientada a objetos C++. En efecto, la mayor parte de los videojuegos para la consola Xbox 360 están desarrollados en el lenguaje C/C++. A continuación se muestra una imagen en la Figura 3.3 del programa XNARacer que fue desarrollado utilizando el lenguaje C# con la plataforma XNA.

Figura 3.3

En C/C++ se puede tener un funcionamiento mucho mejor, lo cual nos permite considerar posibilidades prácticamente ilimitadas para el nivel de inmersión que deseamos para nuestra aplicación.

El desarrollo de aplicaciones en C/C++ tiene cierta complejidad, que se verá aumentada por nuestra decisión de hacer nuestra aplicación nativa al sistema Windows. Es fácil comparar la complejidad que tienen las aplicaciones nativas de Windows y las aplicaciones de consola. A continuación se muestra el código de una aplicación de consola que no muestra ni recibe información alguna, se limita a iniciar el programa y a terminarlo en cuanto se presiona la tecla Escape.

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

void menu()
{
    int test=getch();
    if(test!=27)
        menu();
}
```

```
int main()
{
    menu();
    return 0;
}
```

El código mostrado nos genera la aplicación mostrada en la Figura 3.4.

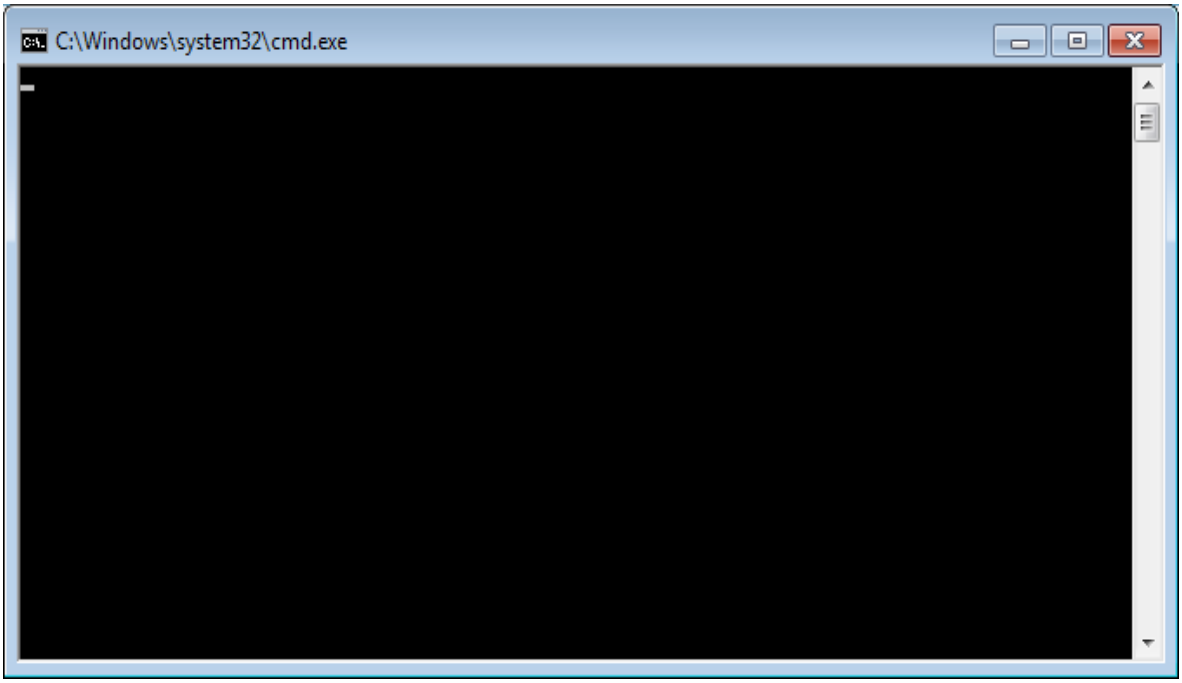

Figura 3.4

Como podemos ver, como sólo tenemos acceso al teclado como entrada y a texto en pantalla como salida la implementación es bastante sencilla.

Por otro lado, si deseamos implementar una aplicación basada en Windows que genere una ventana con funcionalidad mínima y nos permita salir de la aplicación al pulsar la tecla Escape, obtendremos entonces código muy similar al siguiente:

```
#include <windows.h>

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
```

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, PWSTR nCmdLine,
intnCmdShow)
{
    constwchar_t CLASS_NAME[] = L"WindowClass";
    WNDCLASS wc = { };
    wc.lpfnWndProc = WindowProc;
    wc.lpszClassName = CLASS_NAME;
    wc.hInstance = hInstance;
    RegisterClass(&wc);

    HWND hwnd = CreateWindowEx(
        0,
        CLASS_NAME,
        L"MyFirstWind",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL);

    if(hwnd == 0)
        return 0;

    ShowWindow(hwnd,nCmdShow);
    nCmdShow = 1;

    MSG msg = { };
    while(GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return 0;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
    switch(uMsg)
    {
    caseWM_DESTROY:PostQuitMessage(0); return 0;
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd,&ps);
            FillRect(hdc,&ps.rcPaint,(HBRUSH)(COLOR_WINDOW+5));
            EndPaint(hwnd,&ps);
        }return 0;
```

```
    case WM_CHAR:
        {
            if(wParam==27)
                PostQuitMessage(0);
        }return 0;
    case WM_CLOSE:
        {
            DestroyWindow(hwnd);
        }return 0;
    }
    returnDefWindowProc(hwnd,uMsg,wParam,lParam);
}
```

El ejecutable generado por este código se muestra en la imagen de la Figura 3.5.

Figura 3.5

Podemos fácilmente apreciar la diferencia que existe al buscar desarrollar código para aplicaciones nativas de Windows y para consola. Las ventajas que ofrece utilizar el sistema Windows parecen ser despreciables si se considera la diferencia de

complejidad de desarrollo. No obstante, si deseamos emplear gráficos tridimensionales para la simulación de prótesis en tiempo real es indispensable trabajar de este modo. En la sección correspondiente a gráficos se podrá apreciar los límites que nos imponen las aplicaciones de consola para manejar visualización y la ventaja de usar sistemas nativos de Windows.

Desarrollo de aplicaciones para Win32

La programación de aplicaciones de ventanas utilizando el lenguaje C es un tema extenso documentado en una gran cantidad de libros. Nuestro objetivo no es describir detalladamente el proceso de programación para desarrolla aplicaciones de este tipo, sino presentar su funcionamiento y diseño general.

En una aplicación de consola la implementación de un algoritmo requiere únicamente ordenar las instrucciones que la conforman e implementar las decisiones basadas en entradas, tiempo u otros estímulos. No obstante, como deseamos implementar nuestra aplicación en un sistema para Windows, debemos preparar la aplicación para interactuar con otros procesos en este sistema nativamente multitareas, en otras palabras, tener soporte para recibir y reaccionar a los mensajes que le envíe el sistema operativo.

Una de las principales características del sistema operativo Windows es su habilidad de permitir correr programas simultáneamente. Cuando se corren aplicaciones de consola el sistema operativo se encarga de asignar velocidades de operación a cada instancia. No obstante, al desarrollar aplicaciones para Windows podemos mejorar la eficiencia de la misma indicando a ésta la manera en que se comunica con el sistema operativo.

La manera en que Windows se comunica con sus aplicaciones nativas es por medio de mensajes. Al desarrollar aplicaciones que soporten dicha interacción será común encontrar el prefijo "WM_" que indica el uso de un mensaje de Windows (Windows Message). Existe una enorme cantidad de mensajes de Windows que nos sirven para saber si la aplicación quiere finalizar, si existe una nueva entrada del teclado o del

mouse, si el sistema operativo está por entrar a modo protector de pantalla. En las referencias se adjunta un enlace que lista arriba de 200 tipos de mensajes, éstos se pueden ver en el archivo WinUser.h incluido por todo compilador que soporte el desarrollo de aplicaciones de ventanas.

Ahora bien, en las aplicaciones de consola nos es posible declarar una serie de acciones que se ejecuten hasta llegar a un punto que se requiera una entrada por teclado para seguir su operación. En este caso basta con programar un ciclo que únicamente pida la entrada requerida para poder seguir y en caso negativo que vuelva a pedirla hasta cumplirse. Si hiciéramos esto en el caso de aplicaciones para Windows se nos imposibilitaría el poder presionar cualquier botón de la ventana y en caso de colocar la ventana de otra aplicación sobre ésta sus contenidos visibles reemplazarían la vista de nuestra aplicación. Esto sucedería porque una aplicación tiene distintas operaciones importantes que ocurren secuencialmente, entre ellas dibujar a la pantalla los contenidos de la ventana, recibir entradas de teclado y mouse, entre otras respuestas a mensajes.

Para solucionar los problemas descritos existen tres partes principales en la programación de una aplicación de ventanas. En primer lugar se tiene la inicialización donde se dan los valores iniciales a variables globales y se inician procesos que se requerirán posteriormente, como en el caso de un generador de números pseudoaleatorios. Posteriormente se pasa a un ciclo que no cesará hasta que se reciba el mensaje que pida la finalización de la aplicación. Por último se tiene una parte de finalización, donde se liberan los recursos de memoria asignados en la aplicación y se cierran los dispositivos abiertos y procesos adicionales iniciados.

El ciclo mencionado es la parte más importante y es donde se encuentra la lógica principal del programa. En aplicaciones de ventanas este ciclo se dedica únicamente a procesar y reaccionar a los mensajes de Windows para que la aplicación realice lo deseado. No obstante, si se utilizarán gráficos de alta eficiencia es preciso separar el ciclo en una parte que analice mensajes y una parte que se dedique totalmente a la aplicación a implementar. Es en esta parte del ciclo donde se implementa la lógica, renderizado y comunicación con puerto serial en la aplicación que se describe en este trabajo.

Para poder desarrollar aplicaciones para ventanas se requiere de capacitación extensa y ya se ha mencionado que no es nuestro objetivo actual instruir al lector para que sea capaz de desarrollar aplicaciones de este tipo, sino despertar su interés al ver la utilidad que tiene. Uno de los principales recursos para programar este tipo de aplicaciones es el que nos ofrece directamente Microsoft, llamado MSDN.

Comunicación con tarjeta de adquisición

Para poder recibir información de dispositivos externos, en nuestro caso de una tarjeta de adquisición que capture señales mioeléctricas generadas por el usuario, es indispensable tener cierta comunicación entre los dispositivos. Hace ya algunos años, cuando el desarrollo de aplicaciones de consola se encontraba en su auge la comunicación paralela predominaba, principalmente por su simplicidad de programación y el uso de librerías diseñadas para esto en la mayoría de los compiladores de C. Pese a que este tipo de comunicación podría ser implementado con facilidad no podemos utilizarlo hoy en día ya que es obsoleto. En efecto, este sistema de comunicación fue reemplazado con el tiempo por sistemas de comunicación serial. A diferencial de un sistema de comunicación paralelo, la comunicación serial transmite un bit a la vez, lo que podría parecer ineficiente respecto a la comunicación paralela aunque tiene ventajas, incluyendo la habilidad de usar cables de bajo costo y conectores pequeños.

Aunque se puede argüir que muchas computadoras ya no tienen el puerto serial esto puede refutarse fácilmente ya que los puertos USB (Universal Serial Bus) son precisamente puertos seriales y si se desea utilizarlos para interfaces RS-232 se puede con gran facilidad conseguir un convertidor USB/serial. Por lo tanto, no nos afecta el utilizar un puerto serial para la comunicación con la tarjeta de adquisición, ya que basta la existencia de un puerto USB para utilizar el protocolo. La principal desventaja respecto a utilizar directamente los puertos USB es tener un límite de velocidad de comunicación de 115200 baudios, que para la aplicación que buscamos desarrollar es más que suficiente. Esto aunado a la complejidad de programar comunicación USB en un microcontroladorhace el utilizar un puerto serial lo más viable.

Para el desarrollo de aplicaciones de consola es relativamente sencillo habilitar comunicación serial, gracias a las librerías que permiten esto. No obstante, como ya hemos comentado, buscamos desarrollar nuestra aplicación sobre la plataforma Windows, lo que nos obligará a diseñar el código para la comunicación con base en lo expuesto de mensajes en Windows.

Existe mucha información respecto a la captura de entradas por teclado, entradas del ratón, dibujar bidimensionalmente sobre una ventana, implementar un menú de opciones, entre muchas otras. No obstante, cuando uno busca comunicar una aplicación de Windows con el puerto serial se notará pronto la carencia de recursos que expliquen claramente la implementación de este sistema de comunicación. Tal fue la dificultad de implementar dicha comunicación, que aprovecharemos este apartado para dar una breve explicación del código desarrollado para este objetivo.

En primer lugar debemos declarar una variable del tipo HANDLE que sea el objeto de puerto serial:

```
//Serial Port Handle
HANDLE hSerial;
```

Ya tenemos con esto una manera de controlar lo que suceda al puerto serial. No obstante, para que éste nos sea de utilidad es preciso inicializar dicho puerto. Para esto se ha diseñado la siguiente función, que muestra la inicialización básica para el puerto. Se proporciona el código funcional pero una explicación detallada va más allá de los objetivos del presente trabajo, para esto se puede consultar alguno de los recursos mencionados como MSDN.

```
voidinitSerialPort()
{
    //ABRIR EL PUERTO
hSerial = CreateFile("COM4",
GENERIC_READ|GENERIC_WRITE,
                0,
                0,
                OPEN_EXISTING,
```

```
                    FILE_ATTRIBUTE_NORMAL,
                    0);
if(hSerial==INVALID_HANDLE_VALUE)
        {
if(GetLastError()==ERROR_FILE_NOT_FOUND)
{
mensajeError("No existe el puerto serial");
              }
mensajeError("Error inesperado");
        }

        //PREPARAR LOS PARAMETROS
DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
if (!GetCommState(hSerial, &dcbSerialParams)) {
mensajeError("Error al obtener el estado al puerto");
        }
     dcbSerialParams.BaudRate=CBR_115200;
dcbSerialParams.ByteSize=8;
dcbSerialParams.StopBits=ONESTOPBIT;
dcbSerialParams.Parity=NOPARITY;
if(!SetCommState(hSerial, &dcbSerialParams))
{
mensajeError("Error al asignar el estado al puerto");
}

        //PREPARAR LOS TIMEOUTS
        COMMTIMEOUTS timeouts={0};
timeouts.ReadIntervalTimeout=50;
timeouts.ReadTotalTimeoutConstant=50;
timeouts.ReadTotalTimeoutMultiplier=10;
```

```
timeouts.WriteTotalTimeoutConstant=50;
timeouts.WriteTotalTimeoutMultiplier=10;
if(!SetCommTimeouts(hSerial, &timeouts))
{
mensajeError("Algo paso con los timeouts");
        }
}
```

En el código mostrado se detallan las características que debe tener el puerto, entre ellas cual puerto se está usando (COM4), la velocidad de comunicación (115200 baudios), paridad, entre otros.

Para poder utilizar el puerto habilitado necesitamos primero declarar funciones que nos permitan escribir un número del tipo byte (0-255) o leerlo. Se tiene como ejemplo de implementación lo siguiente:

```
voidComunicacionSerialMandaByte(unsigned char *dat1,unsigned char *dat2,char cManda)
{
     charszBuff[1] = {cManda};
        DWORD dwBytesEscritos = 0;
if(!WriteFile(hSerial, szBuff, 1, &dwBytesEscritos, NULL))
{
mensajeError("Algun error en la escritura de datos");
}

     unsigned char szBuff1 ;
      DWORD dwBytesRead1 = 0;
     if(!ReadFile(hSerial, &szBuff1, 1, &dwBytesRead1, 0))
{
mensajeError("Algun error en la lectura de datos 1");
}
     *dat1=szBuff1;
```

```
unsigned char szBuff2 ;
    DWORD dwBytesRead2 = 0;
    if(!ReadFile(hSerial, &szBuff2, 1, &dwBytesRead2, 0))
{
mensajeError("Algun error en la lectura de datos 1");
    }
    *dat2=szBuff2;
}
```

Si se analiza con cuidado esta función se puede apreciar que en primer lugar se envía el carácter usado como segundo parámetro en la función al puerto serial e inmediatamente se leen dos valores de 128 bits del puerto serial, que corresponden al valor digital de las dos señales mioeléctricas capturadas. Esto se hace porque el microcontrolador tiene como función principal capturar y convertir las señales mioeléctricas que reciba hasta ser detenida momentáneamente por una bandera de interrupción por escritura serial. En ese momento, envía por comunicación serial el último valor convertido de las señales mioeléctricas, resetea la bandera de interrupción y continua capturando y convirtiendo los datos de las señales entrantes. Esto se discutió con mayor detalle en el capítulo anterior.

Podemos entonces apreciar el protocolo de comunicación que existe entre la PC y el microcontrolador. Con cada paso por el ciclo principal el microcontrolador actualiza los valores digitales de las señales mioeléctricas. Si deseamos recibir un valor en la PC debemos enviar desde ésta al microcontrolador un carácter que solicite el envío de los valores. Al detectar la llegada de un carácter por comunicación serial, el microcontrolador envía a la PC por comunicación los últimos valores actualizados de las señales que debe capturar. La PC debe estar lista para recibir los valores inmediatamente y utilizarlos para la simulación.

Gráficos

En este trabajo buscamos simular de manera realista el funcionamiento de una prótesis dada una señal de control. Por lo tanto, fue preciso utilizar un sistema de desarrollo que nos permita utilizar entornos tridimensionales en tiempo real, lo cual puede ser logrando corriendo aplicaciones sobre sistema Windows que aprovechen la velocidad de procesamiento que nos brinda la Unidad de Procesamiento Gráfico (GPU) de la computadora. Las GPUs pueden ser accedidas utilizando librerías que aprovechen los resultados que éstas arrojan y los dibujen sobre una ventana. Las librerías más importantes en la actualidad que realizan esto son las nombradas DirectX y OpenGL.

DirectX es la librería de gráficos ofrecida por Microsoft exclusivamente para el sistema operativo Windows. Ha sido criticada por muchos programadores porque hasta su versión 6 fue considerada una librería de muy baja calidad, cuyo uso era sumamente complicado debido a la confusa y escasa documentación existente. No obstante, en la actualidad es una de las mejores librerías gráficas en existencia y en la opinión del autor de este trabajo la mejor para el sistema Windows desde la versión DirectX 10. Una gran cantidad de sistemas de simulación utilizan esta librería, entre ellas cabe mencionar la línea de videojuegos de Halo, una captura de éstas se muestra en la Figura 3.6.

Figura 3.6

Por otro lado, existe la librería de gráficos OpenGL, que es la más popular actualmente. La principal razón de esto es que no se limita al uso exclusivo del sistema operativo Windows, sino que nos permite que nuestro código gráfico sea utilizado en otros sistemas que la soporten, como Linux, Mac OS, iOS, entre otros. En la actualidad, la calidad de los gráficos generados por aplicaciones es muy buena, aunque ha perdido la batalla contra DirectX en ámbito de los videojuegos. No obstante, sigue siendo la principal herramienta de simulación gráfica para investigaciones científicas y de ingeniería debido a que las áreas de ingeniería y diseño suelen utilizar los sistemas operativos de Apple y una gran cantidad de investigadores utilizan alguna distribución del sistema operativo Linux. En la Figura 3.7 se muestra una captura de una nube de puntos obtenida del sensor Kinect utilizada por el autor en un proyecto reciente dirigido por el Ing. Yair Bautista Blanco. Esta imagen fue renderizada utilizando la librería de OpenGL para Windows.

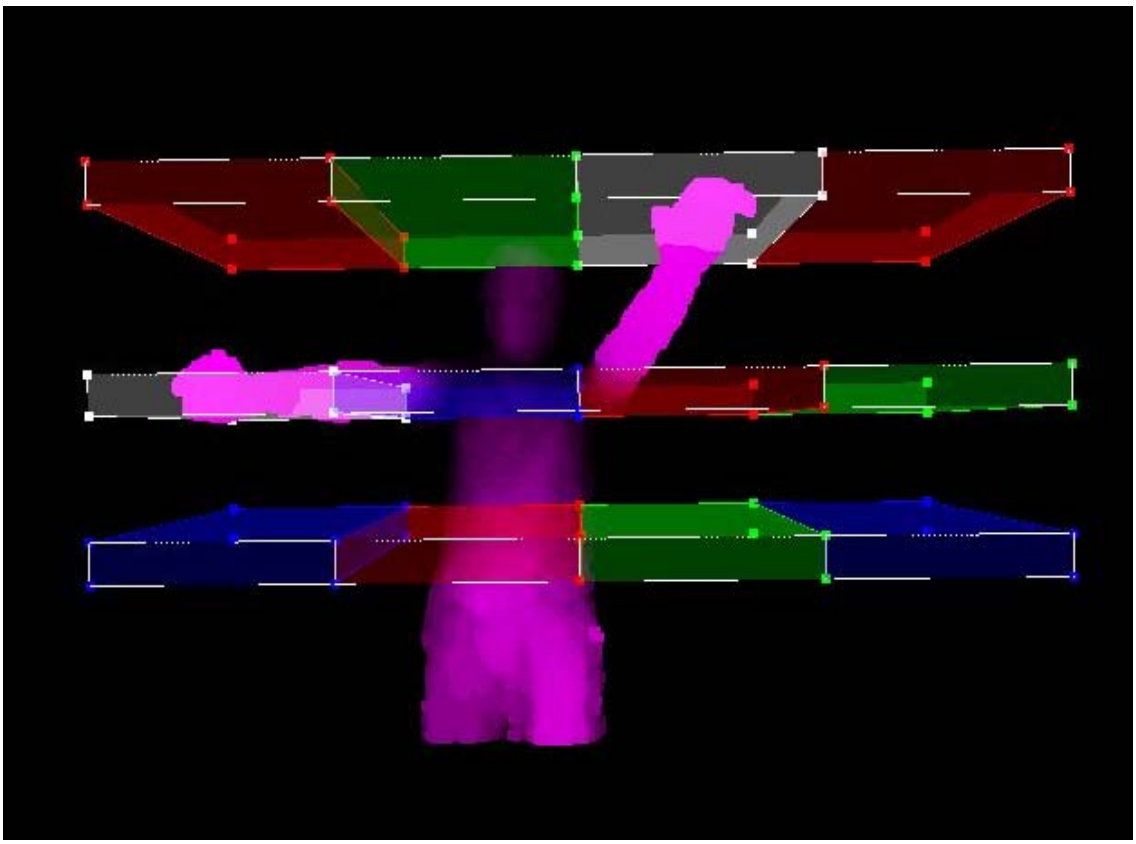

Figura 3.7

Por las razones discutidas se eligió la librería de gráficos OpenGL para implementar nuestro sistema de visualización virtual corriendo sobre una aplicación nativa de windows para optimizar la comunicación con el sistema operativo.

La palabra OpenGL significa Librería Abierta de Gráficos (Open Graphics Library) y fue desarrollada en un principio por SiliconGraphicsInc en 1992. Como se ha discutido es usada por muchos investigadores debidoprincipalmente a su soporte multiplataforma y sencilla portabilidad. Desde el 31 de julio de 2006 el control de la especificación OpenGLpaso al grupo Khronos, compuesto de muchas empresasy asociaciones, entre ellas Apple, Google, HP, Oracle, Dell, Blizzard e IBM. La más reciente revisión hecha a la especificación fue OpenGL 4.1, publicada el 26 de julio de 2010.

El desarrollo de aplicaciones profesionales en OpenGL requiere de un extenso estudio de temas de Computación Gráfica, Álgebra Lineal, Robótica, Diseño Gráfico, Algoritmos, entre otros. No obstante podemos rápidamente ejemplificar su uso con un ejemplo simple.

Consideremos dibujar en pantalla un rectángulo de color rojo. Ya hemos comentado que las aplicaciones para Windows tienen un ciclo de operación que tras pasar la etapa de inicialización es el que realizas las acciones programadas para las diversas entradas del programa hasta que se recibe una señal que ordene cerrar a la aplicación. Es en este ciclo que ordenaremos al programa dibujar nuestra figura cada vez que se entre a él.

En primer lugar debemos configurar las opciones de OpenGL en la inicialización del programa, es decir, el modo en que se dibujará la escena, el tipo de proyección, el uso de sombras u otras propiedades, entre otras. Hecho esto,

Consideremos el siguiente código:

```
glBegin(GL_LINE_LOOP);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(10.0f, 0.0f, 0.0f);
    glVertex3f(10.0f, 10.0f, 0.0f);
    glVertex3f(0.0f, 10.0f, 0.0f);
glEnd();
```

En este código la función glBegin() habilita las opciones de dibujo de primitivas de OpenGL. Las primitivas en OpenGL son un conjunto de formas base que se pueden tener en una escena, por ejemplo: puntos, líneas, triángulos, *quads* y polígonos. Al establecer la opción GL_LINE_LOOP estamos declarando que deseamos dibujar únicamente líneas y que formarán una figura cerrada en el orden que se indique al escribir los puntos.

La instrucción glColor3f() nos permite elegir el color de la primitiva seleccionada. Basta indicar en cada argumento de la función las componentes respectivas de color Rojo, Verde y Azul, donde 1.0 constituye el valor máximo y el sufijo "f" que es un valor de punto flotante.

Las instrucciones glVertex3f() declaran el uso de un punto virtual y reciben tres parámetros correspondientes a las componentes X, Y y Z , respectivamente. El orden en que se traza estos puntos corresponde a la secuencia en que son declarados. En este caso, Se traza una línea roja del punto (0,0,0) al (10,0,0), luego del (10,0,0) al (10,10,0), posteriormente del (10,10,0) al (0,10,0), y finalmente del (0,10,0) al (0,0,0), cerrando así la figura.

La instrucción glEnd() termina la rutina de dibujo de este tipo de primitiva y nos da la opción de dibujar otra figura usando otro tipo de primitivas o la misma de así requerirlo.

Como podemos apreciar, el código escrito usando la librería de OpenGL es relativamente intuitivo en su comportamiento. Ya hemos comentado que nuestro objetivo no es instruir al lector a desarrollar un sistema semejante al de este trabajo, sino a comprender de manera general su diseño y funcionamiento. Por lo que se pueden consultar excelentes referencias como las escritas por Wright y Shreiner.

La librería auxiliar GLUT

Ahora bien, vale la pena mencionar que al ser OpenGL un estándar de gráficos, existen diversas implementaciones del mismo y especialmente un gran número de librerías que ofrecen diferentes funciones de programación.

Las librerías gl.h y glu.h son de gran importancia para programar cualquier aplicación gráfica. Estas librerías nos dan acceso a funciones que nos permiten dibujar primitivas en pantalla, ajustar el modo de proyección de nuestra escena, aplicar transformadas a las cámaras con las que veremos la escena, entre muchas otras funciones. Cuando uno comienza a programar usando OpenGL, la mayor parte de tutoriales o libros sugieren el uso de una librería adicional llamada glut.h. Esta librería nos libera de las complicaciones que resultan de crear nuestra ventana usando código específico para la plataforma que programemos, tiene ya funciones que nos permiten dibujar cuerpos sólidos básicos en nuestra escena, tiene ya un ciclo para poder dibujar nuestra escena,entre muchas otras funciones de utilidad. Por ejemplo, en la Figura3.8 podemos observar una atractiva figura que se puede obtener con escribir el comando:

voidglutSolidTeapot(GLdouble*size*);

Figura 3.8

Pese a todas las ventajas que nos ofrece el uso de esta librería, el principal costo que tiene es en la eficiencia del código resultante. Es decir, cómo ya existen funciones que nos simplifican procesos indispensables para hacer una aplicación gráfica, no nos

damos a la tarea de desarrollar el código para optimizar dicho funcionamiento. Una aplicación implementada con la ayuda de glut.h será mucho menos eficiente que una diseñada específicamente para la plataforma y en que se trabaja a bajo nivel con el dibujo de primitivas. Por lo tanto, es bueno utilizar esta librería para aprender a utilizar gráficos independientemente de las complicaciones que ofrezca cada sistema operativo para su implementación y para realizar prototipos y demos. No obstante, si se busca tener una aplicación eficiente es preciso evitar su uso. Para el presente proyecto sólo se utilizaron las librerías gl.h y glu.h para la parte gráfica.

Configuración de Cámaras Virtuales

Todo diseñador de piezas mecánicas aprecia la importancia de poder tener diferentes vistas de un mismo objeto tridimensional en un pantalla plana. En la paquetería de CAD es sumamente importante conocer el sistema de navegación, o bien, cómo controlar la cámara virtual de la escena que estemos analizando.

La pieza contiene todos sus elementos en un mundo tridimensional virtual, pero el usuario sólo cuenta con una pantalla plana para interpretarla. Los movimientos de rotación de la escena, alejamiento y acercamiento de la misma y la traslación son de suma importancia para poder comprender la escena en cuestión.

Podemos fácilmente hacer una comparación del funcionamiento de cámaras virtuales con las reales. Si deseamos tomar una fotografía de un objeto, existe una cantidad mínima de parámetros para poder describir las posibles fotografías diferentes resultantes. En primer lugar debemos ubicar la cámara en algún punto del espacio, para esto tenemos tres coordenadas de posición, que definen al parámetro vectorial "eye". Posteriormente debemos apuntar la cámara hacia algún punto en el espacio, no necesariamente centrando al objetivo, ya que este puede encontrarse en otra región de la fotografía; esto describe al parámetro vectorial "at". Por último, podemos girar la cámara tomando como eje el vector se obtiene al conectar los parámetros vectoriales "eye" y "at", permitiéndonos manipular la orientación con la que se observa la escena. Si bien podemos asignar un ángulo para describirlo, nos es más sencillo asignar un vector

que nos indique la dirección hacia la que apunta la parte superior de la cámara, esto se describe con el parámetro vectorial "up". En la Figura3.9, se puede apreciar un diagrama de lo descrito.

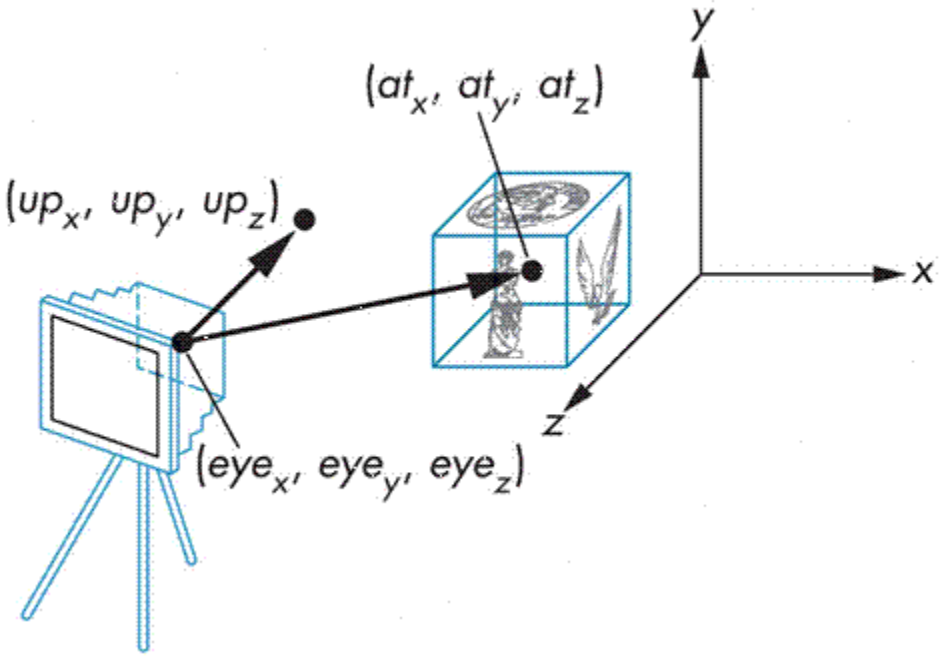

Figura3.9

Aprovecharemos este tema para presentar la aplicación Blender, que fue un recurso de suma importancia para este proyecto. Blender es una paquetería de herramientas de código abierto para la creación de objetos tridimensionales. En este programa podemos modelar piezas tridimensionales y generar animaciones tanto para juegos como películas. En nuestro caso lo hemos utilizado para importar piezas, proceso que discutiremos más adelante, adecuarlas para su procesamiento en OpenGL y exportarlas a un formato que soporte nuestra aplicación para poder visualizarlas.

En las figuras 3.10, 3.11 y 3.12, podemos apreciar las diferentes vistas de una escena. En la escena se encuentra una de las piezas de la prótesis que utilizaremos para este proyecto, una fuente de luz y la cámara. Es fácil apreciar un triángulo amarillo en la cámara, éste corresponde al parámetro vectorial "up" discutido anteriormente. Si deseamos visualizar de algún modo la escena podemos mover la cámara usando operaciones de traslación, lo que modifica al parámetro vectorial "eye". También

podemos modificar la dirección hacia la que apunta la cámara con aplicarle la operación de rotación, esto modifica al parámetro vectorial "at".

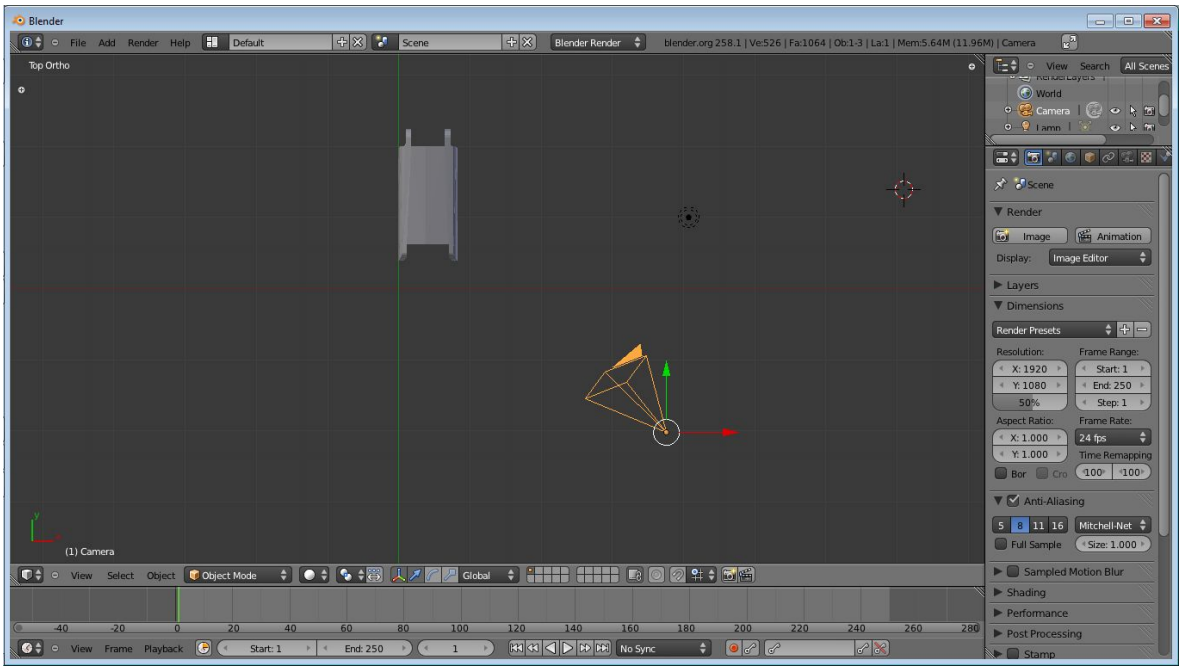

Figura 3.10

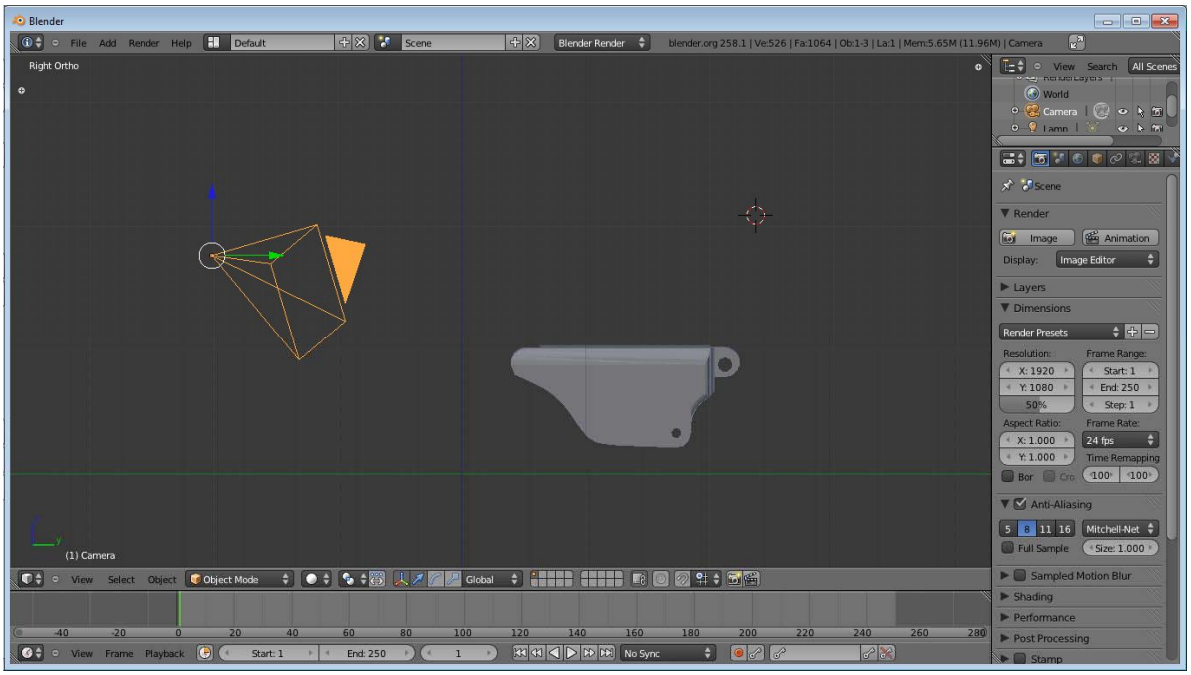

Figura 3.11

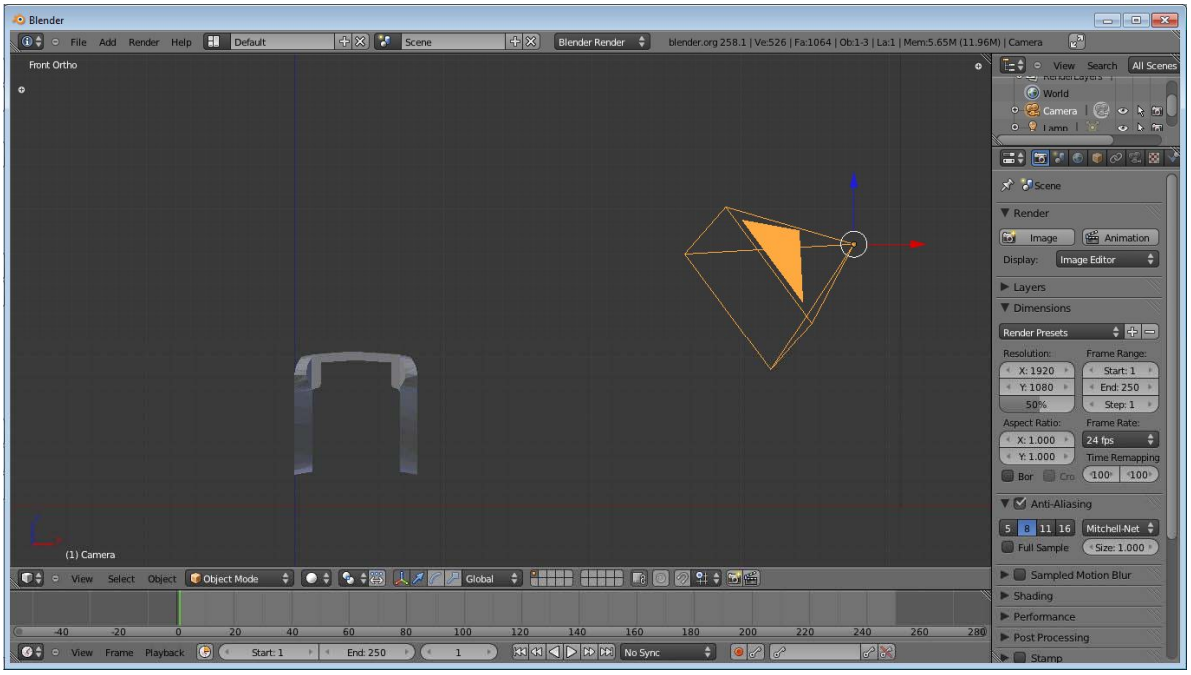

Figura 3.12

Podemos apreciar en la Figura 3.13cómo se renderizará finalmente la escena, gracias a los parámetros que definen completamente a la cámara. Si nos alejamos un poco de la cámara podemos apreciar que en efecto la imagen resultante corresponde con los parámetros ajustados de la cámara, de acuerdo con lo mostrado en la Figura 3.14.

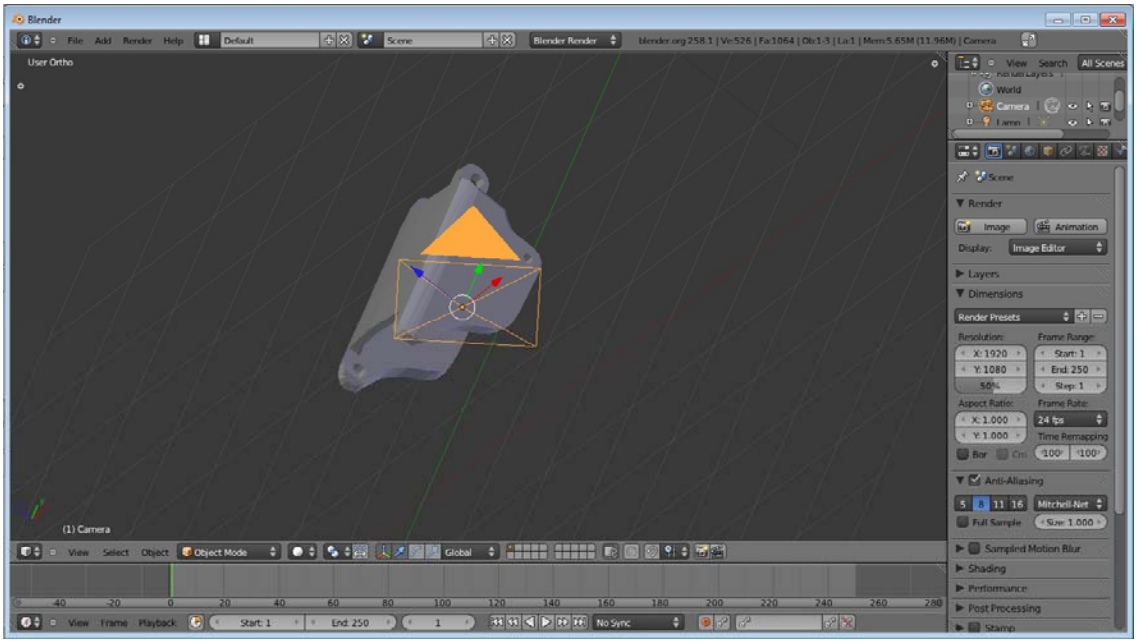

Figura3.13

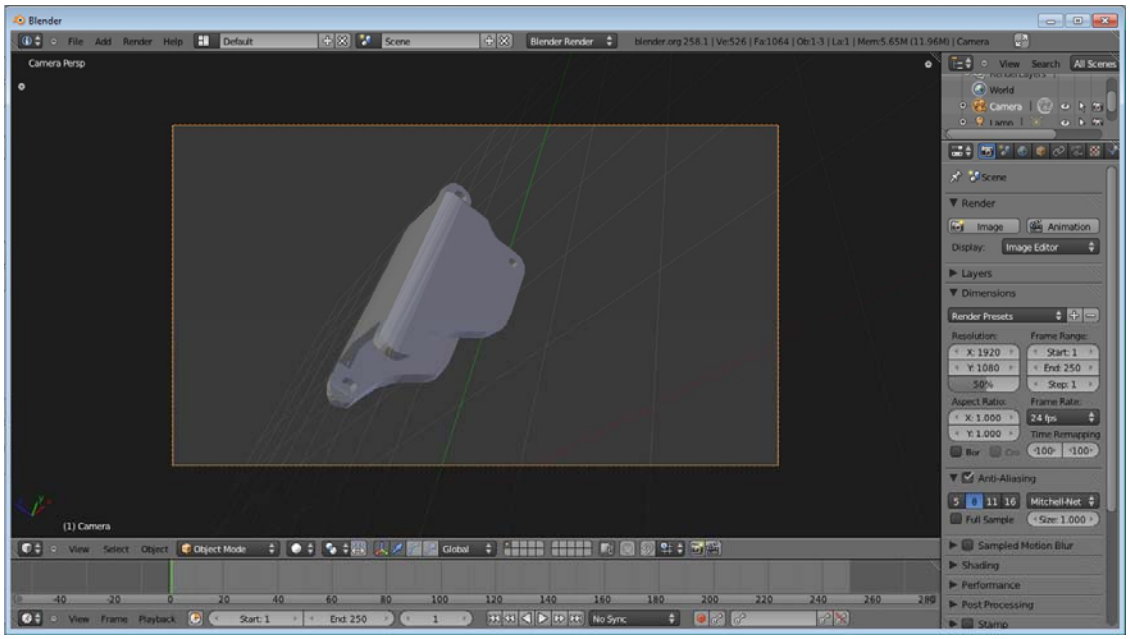

Figura 3.14

Nos hemos detenido en explicar cómo se utilizan las cámaras en ambientes tridimensionales porque al utilizar OpenGL es preciso definir los parámetros de la cámara para poder visualizar la escena en cuestión. En efecto, hemos implementado una característica sumamente útil en paquetería de diseño 3D, que es el poder cambiar la vista actual con solo presionar una tecla específica. En este caso, podemos cambiar a una vista de planta con la tecla Enter, y a la vista en perspectiva con la Barra Espaciadora. Esto se implementa con cierta facilidad utilizando la función:

gluLookAt(GLdoubleeyeX , GLdoubleeyeY, GLdoubleeyeZ,
GLdoublecenterX, GLdoublecenterY, GLdoublecenterZ,
GLdoubleupX, GLdoubleupY, GLdoubleupZ);

Se puede apreciar la correlación directa con los parámetros que discutimos anteriormente, la principal diferencia se encuentra en el cambio de notación de “at” a “center”. Solo tenemos que colocar las coordenadas de los parámetros discutidos y esta función realiza la operación de actualizar la cámara.

Formatos para el uso de los modelos de la prótesis

En la sección anterior pudimos ver un componente de la prótesis que utilizaremos para realizar la simulación buscada en este trabajo.

Estas piezas fueron diseñadas por la Ing. Rosa Itzel Flores Luna y la Ing. Ana Marissa Juárez Mendoza, su trabaja se encuentra descrita en la tesis "Diseño de Prótesis Mecatrónica de Mano" de abril de 2007. Se puede consultar este trabajo para ver las especificaciones y la justificación del diseño usado.

Buscamos desarrollar un sistema gráfico interactivo de prótesis de miembro superior para entrenamiento, como ya se ha comentado anteriormente. Para cumplir con este objetivo deseamos poder controlar una prótesis virtual utilizando impulsos reales para que el entrenamiento sea lo mejor posible. Para esto necesitamos adecuar las piezas de la prótesis para su uso en la aplicación que desarrollaremos.

Las piezas fueron modeladas en SolidEdge, de acuerdo con lo reportado en la tesis, y fueron exportadas desde este programa a formato .stl, ya que se precisaba imprimir a prototipo rápido los modelos para su evaluación y mejor visualización utilizando la paquetería de software Catalyst.

El formato STL es un tipo de archivo usado con gran frecuencia para prototipos rápidos, manufactura asistida por computadora y visualización inmersiva. Una de sus principales ventajas es que puede ser exportado desde e importado a la mayoría del software de diseño 3D, lo cual nos permite cerrar la brecha existente entre los formatos de diseño ingenieril y los formatos que podemos utilizar para animaciones en programas como el que buscamos desarrollar.

La librería de gráficos OpenGL no tiene funcionalidad incluida para cargar modelos tridimensionales en una escena, a diferencia de la plataforma DirectX que puede cargar modelos y animaciones en formato .x. Por lo tanto, es preciso implementar o encontrar una librería que permita cargar modelos tridimensionales usando OpenGL. Esto no es tan complicado como podría parecer, en primer lugar se debe analizar el formato.

El formato de STL binario consiste de una línea de cabecera de 80 bytes que podría interpretarse como una cadena de comentarios. Los siguientes 4 bytes se interpretan como tipo *longinteger* y da el número total de caras del modelo. Lo que sigue es una

normal y 3 vértices para cada cara, donde cada coordenada es representada como un número flotante de 4 bytes (con un total de 12 bytes). Existe un espaciador de 2 bytes entre cada cara. El resultado es que cada cara se representa por 50 bytes, 12 para la normal, 36 para los 3 vértices y 2 para el espaciador.

Como podemos ver, el formato es suficientemente simple de interpretar, sería una tarea trivial implementar un cargador de OpenGL para este formato. No obstante, como queremos utilizar un programa como Blender para adecuar el modelo antes de importarlo a nuestra aplicación no es completamente necesario implementarlo. Esto es por tener ya a nuestra disposición una librería para importar modelos en formato 3DS a OpenGL. El formato 3DS essoportado para importación y exportación en Blender, por lo que podemos usar los modelos de los componentes de la prótesis, que se encuentran en formato STL, importarlos a Blender, adecuarlos y exportarlos a formato 3DS para poder finalmente importarlos a nuestra aplicación que usa OpenGL.

Ya con las piezas de la prótesis debidamente preparadas en formato 3DS podríamos importarlas a nuestra aplicación usando una librería de importación ya sea implementada o libre. En nuestro caso optamos por la librería libre, específicamente una versión modificada del importador usado en el proyecto Anoid NG, que implementa una versión 3D del juego Arkanoid, sucesor temático a Breakout.

Es posible utilizar transformadas de traslación, rotación y escala de manera optimizada a los objetos a dibujar en una escena, por lo que es sencillo redireccionar diferentes piezas para ordenarlas cómo deben estar en nuestro espacio 3D. Lamentablemente, una vez que se encuentren en escena nuestras piezas no existe información que una a las distintas piezas, es decir, que nos permita extender y contraer las falanges un dedo con solo activar la función arbitraria de abrirdedo() y cerrardedo(), respectivamente; o bien, realizar una gestura de agarre con la mano. Todo esto debe realizarse por código, lo cual corresponde a una parte de la computación gráfica y robótica llamada modelado jerárquico.

No nos corresponde explicar el tema de modelado jerárquico, baste decir que podemos utilizar las funciones glPushMatrix() y glPopMatrix() para generar regiones en nuestro código en las que la aplicación de transformadas solo afecten a piezas o conjuntos de piezas específicas. Esto nos permite implementar enlaces padre-hijo con

los que podemos definir funcionalidad que ligue a diferentes piezas.En efecto, uno de los primeros ejercicios que se realizan para familiarizarse con este modelado es la implementación de los movimientos de un brazo robot o de un sistema solar. En la sección anterior pudimos ver algunas capturas de Blender, el programa que usamos para adecuar la pieza para su uso en nuestro programa en C/C++. Cabe mencionar que aprovechar el uso de Blender para reposicionar y reorientar las piezas nos ofrece ventajas al ahorrarnos trabajo en la aplicación y al reducir la cantidad de operaciones que debemos realizar sobre la pieza.

Podemos ahora diseñar animaciones con nuestras piezas debidamente jerarquizadas. En primer lugar se deben asentar rangos para los límites físicos humanos entre cada pieza, posteriormente se puede considerar el ángulo de conexión de la falange proximal y su base que varíe en un 100% de su rango, el ángulo entre la falange media y la proximal que varíe linealmente en un 50% de su rango, y el ángulo entre la falange distal y la media que varíe linealmente en 25% de su rango. Esto se ha comprobado que ofrece una solución visualmente atractiva y de bajo consumo de recursos en el programa. No obstante, esto sería una emulación y no una simulación, que es lo que se busca implementar en este trabajo. Si buscamos que la prótesis de la aplicación funcione lo más parecido posible a la real podemos buscar en las especificaciones, en este caso la tesis mencionada anteriormente, las tablas de posiciones ligadas al ángulo de la primera o bien al mecanismo que se utilice. Podemos entonces generar una función que interpole de esta tabla el valor correspondiente a la posición de cada falange en un caso dado y tener una animación muy cercana a la realidad.

Detección de Tensión y Relajación del Músculo

Ya hemos discutido como obtener la información mioeléctrica en tiempo real en nuestra aplicación y cómo dibujar la prótesis con sus propiedades de jerarquización. Ahora para realizar el entrenamiento, que es objetivo de este proyecto, debemos detectar el estado de tensión y animar la prótesis correspondientemente.

La animación ya se ha discutido en la sección anterior y podemos controlarla con el uso de banderas que indiquen la dirección de la animación dependiendo del estado. Ahora bien, para poder habilitar estas banderas requerimos detectar el estado.

Si calculamos bien la amplificación de las señales y confiamos en la estabilidad de todo el proceso podríamos asignar valores límite que separen a las señales entrantes de un estado detensión a uno de relajación. No obstante, por cualquier condición o diferencia de señal por usuarios, es mejor hacer un breve entrenamiento del sistema.

Para realizar el entrenamiento podemos realizar una captura pura de los datos usando el programa, para determinar el valor medio que presentan las señales correspondientes a relajación y considerar esto como offset, y podemos determinar el valor medio que presentan las señales correspondientes a la tensión del músculo y considerar esto como nuestro valor típico. Sólo nos resta calcular el punto medio entre los promedios calculados y tendremos nuestro límite para clasificar las señales.El proceso mencionado, pese a su sencillez, es sumamente efectivo como se comprobó en las pruebas realizadas.En la Figura3.16 podemos apreciar una captura de la animación del dedo índice en tensión, correspondiente al estado de tensión del músculo.

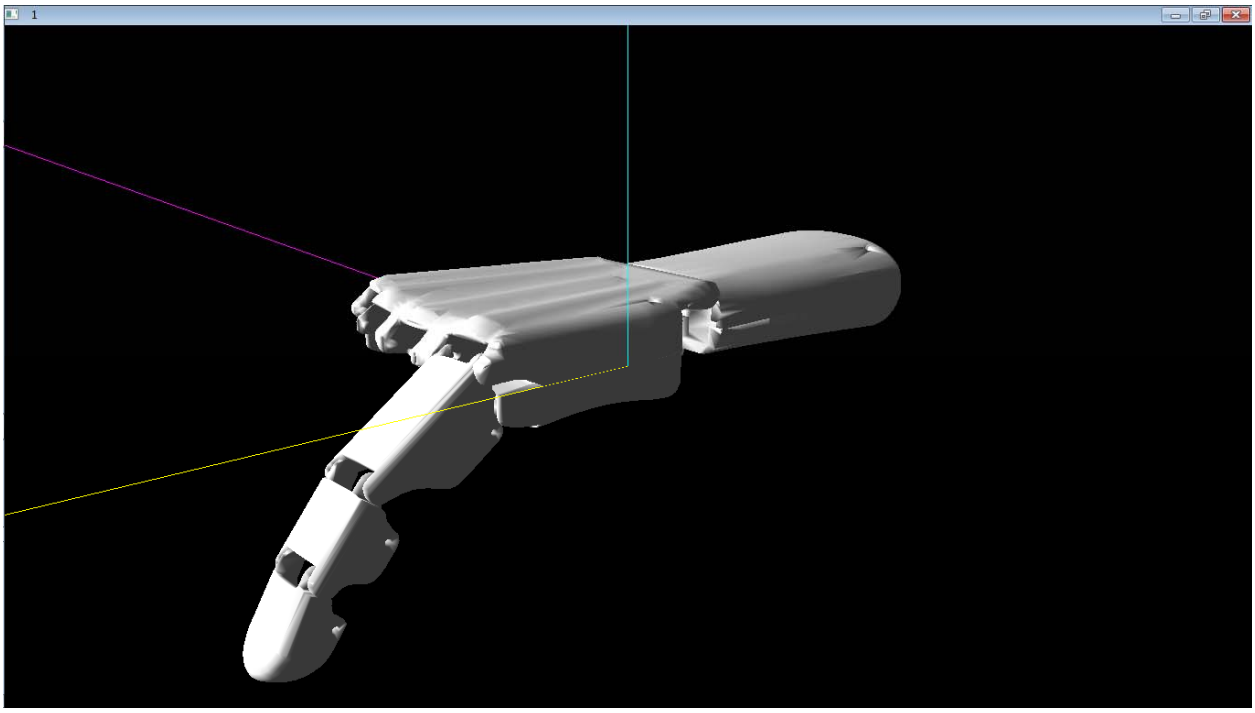

Figura3.16

Conclusiones y Trabajo Futuro

La implementación de programas optimizados para la plataforma Windows nos ofrece ventajas sustanciales, entre ellas la posibilidad de desarrollar aplicaciones gráficas interactivas como la presentada en este trabajo.

El uso de la librería de gráficos OpenGL nos abre las puertas para poder usar el programa desarrollado en otros sistemas operativos con tan solo adecuar el código para las funciones dependientes de plataforma. El uso de DirectX nos hubiera limitado a la plataforma Windows sin la opción de reutilizar aun parcialmente nuestro código.

El uso de comunicación RS-232 es una alternativa suficientemente rápida para el tipo de datos que se manejan y su implementación es relativamente sencilla. Además, si se utiliza un conversor a USB se puede utilizar en la gran mayoría de las computadoras actuales.

Considerando la continuación que se dará a este proyecto por quienes continuarán trabajando en el segundo año de este proyecto IXTLI, el presente trabajo servirá como un peldaño esencial para el desarrollo de aplicaciones más complejas.

Cabe mencionar que cuando se recomienda el uso de una prótesis mioeléctrica, muchos médicos no están seguros si la persona es sujeta de usarla. El sistema desarrollado puede evaluar dicha capacidad del usuario y ofrecer datos que permitan adecuar la prótesis en caso de ser posible.

En un futuro se busca implementar mayor funcionalidad de análisis opcional por hardware en el microcontrolador para simular el funcionamiento de los microprocesadores de prótesis mioeléctricas de bajo costo. Se desea implementar la comunicación USB para hacer el sistema más simple para el usuario. Además, sería recomendable implementar la aplicación para otros sistemas operativos, como Mac OS y sistemas operativos libres.

En cuanto a su funcionamiento, sería deseable poder importar diferentes tipos de prótesis sin precisar de la ayuda de un programador. Esto se podrá lograr usando lenguajes de programación orientados a *scripting*.

Anexos

El disco complementario al presente trabajo se puede encontrar en el Departamento de Mecatrónica de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.

En el disco se encuentra el código desarrollado tanto para la aplicación en PC como el código en lenguaje ensamblador para el microcontrolador.

Referencias

Bibliografía

- Axelson, Jan. Serial Port Complete: COM Ports, USB Virtual COM Ports, and Embedded Systems. Lakeview Research, 2a ed, USA, 2007, Pp 380.
- Northrop, Robert B. Analysis and Application of Analog Electronica Circuits to Biomedical Instrumentation. CRC Press, USA, 2004, Pp. 542.
- Leis, Arturo; Trapani, Vicente. Atlas of Electromyography. Oxford University Press, 2000, UK.
- Flores Luna, Rosa Itzel; Juárez Mendoza, Ana Marissa. Diseño de Prótesis Mecatrónica de Mano. Tesis UNAM, México, 2007, Pp. 153.
- Weiss, Lyn; *et al*. Easy EMG. Elsevier, UK, 2004, Pp. 271.
- Aminoff, Michael. Electrodiagnosis in Clinical Neurology. Elsevier, 5a ed., USA, 2005, Pp. 859.
- Morris, Alan. Measurement and Instrumentation Principles.ButterworthHeinemann, 3a ed., UK, 2001, Pp. 475.
- Astle, Dave; Hawkins, Kevin. Beginning OpenGL Game Programming. Thomson, USA, 2004, Pp. 315.
- Hearn, Donald; Baker, Pauling. Gráficos por Computadora en OpenGL. Pearson Alhambra, España, 2003, Pp. 918.
- Wright, Richard; *et al*. OpenGL SuperBible. Addison-Wesley, USA, 2007. Pp. 1248.
- Hoja de especificaciones de AD536A. AnalogDevices, 1999.
- Hoja de especificaciones de AD620. AnalogDevices, 1999.
- Hoja de especificaciones de AD624. AnalogDevices, 1999.
- Hoja de especificaciones de INA106. Burr-Brown Products, 1987.
- Hoja de especificaciones de PIC16F882/883/884/886/887. Microchip, 2007.
- Hoja de especificaciones de TL072. Texas Instruments, 2005.

- Hoja de especificaciones de TL081. National Semiconductor, 1995.
- Hoja de especificaciones de 1N5225-1N5267. Vishay, 2002.
- Truax, Barry. Handbook For Acoustic Ecology, ARC Publications, 1999.http://www.sfu.ca/sonic-studio/handbook/

Mesografía

- http://www.360oandp.com/OP360/WatchVideo.aspx?id=47
- http://www.greatmedicalsupplies.com/supply~3M+Healthcare+(88)~pediatric-electrode-44-cm-25-per-bag-2248.htm
- www.ieee.org
- www.ign.com
- http://anoid.dk/output/index.html
- http://www.cs.washington.edu/education/courses/cse457/CurrentQtr/lectures/hierarchical.pdf
- http://www.dekaresearch.com/deka_arm.shtml
- http://www.touchbionics.com
- http://zone.ni.com/wv/app/doc/p/id/wv-1330
- http://www.lancetahg.com/lanceta/listaProd/e_prod/e005.html
- http://www.lancetahg.com/lanceta/listaProd/c_prod/c002.html
- http://www.autohotkey.com/docs/misc/SendMessageList.htm
- http://jcsites.juniata.edu/faculty/rhodes/graphics/opengltrans.htm
- msdn.microsoft.com