DISEÑO E IMPLEMENTACIÓN DEL CONTROL DE ESTABILIZACIÓN PARA UNA PLATAFORMA ALTAZIMUT PARA USO EN FOTOGRAFÍA

ING. MAURICIO JORGE MACHADO BURITICÁ

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRIA DE INGENIERIA ELECTRÓNICA
BOGOTÀ
2013

DISEÑO E IMPLEMENTACIÓN DEL CONTROL DE ESTABILIZACIÓN PARA UNA PLATAFORMA ALTAZIMUT PARA USO EN FOTOGRAFÍA

MAURICIO JORGE MACHADO BURITICÁ

Trabajo de grado para optar al título de Magister en Electrónica

Director:
CAMILO ALBERTO OTALORA SÁNCHEZ
Magister en Electrónica

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRIA DE INGENIERIA ELECTRÓNICA
BOGOTÀ
2013

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCION

El siguiente trabajo de grado plantea el control de posición de la plataforma Altazimut utilizando encoders comandados desde una GUI desarrollada en MATLAB® y estabilización de posición a perturbaciones externas en el ángulo de elevación(θ) de la plataforma generadas por un mecanismo manual.

Se incluye la caracterización de la plataforma Altazimut también llamada plataforma pan-tilt, planteando un modelo en caja blanca desacoplado con la función de transferencia de los motores compuestos por esta, la posterior simulación y validación del modelo. La implementación del control de posición utilizando un algoritmo PID tipo paralelo, en un tarjeta embebida capaz de ejecutar los algoritmos de control.

Los recursos de hardware y software se describen enfocándose en la implementación especifica así como los procedimientos utilizados para calibrar los sensores utilizados para el control de posición con Encoders y para el control de estabilización con el sensor LSM303DLHC(ver [1]), además de los cálculos realizados para implementar en el sistema embebido basado en tecnología de microcontroladores Microchip y ATmega.

Finalmente se describen las pruebas realizadas a cada uno de los procedimientos utilizados en cada una de las plataformas hardware Seeeduino y Cerebot MC7^{TM}, además de los resultados obtenidos de los controles de posición y estabilización implementados en el trabajo de grado.

OBJETIVOS Y PLANTEAMIENTO DEL PROYECTO

Objetivo General

Estabilizar una plataforma altazimut para toma de fotografías usando acelerómetros y compas magnético.

Objetivos Específicos

- Obtener modelo dinámico de plataforma Altazimut disponible.
- Diseñar el algoritmo de control de estabilización en dos ejes.
- Implementar el control sobre un hardware dedicado. DSP, amplificadores, sensores, etc.
- Desarrollar interface usuario para el control y evaluación de la plataforma.
- Construcción e Implementación de un mecanismo para la generación de perturbación externa en un eje de movimiento a la plataforma altazimut.
- Comparación y validación del modelo analítico desarrollado versus planta real implementada.

Planteamiento Del Proyecto

En este trabajo de grado se presenta la implementación de una plataforma prototipo altazimut de la figura 2 para la toma de fotografías. En este contexto y en particular al orientar la cámara apuntando fijamente hacia una panorámica; es normal que en ocasiones las imágenes que esta nos ofrezca se vean afectadas por la posición del soporte rígido que puede estar montado en algún vehículo en movimiento

terrestre, marítimo o aéreo en un momento determinado. Se implementa una plataforma altazimut por medio de la tarjeta de desarrollo Cerebot basada en dsPIC que permitirá hacer correcciones automáticas de tal forma que se mantenga en todo momento estable. Para lograr este cometido se implementara una IMU(del inglés inertial measurement unit) de 6 grados de libertad que se apoya en un acelerómetro y un compas magnético de tres ejes cada uno, para sensar la posición del lente de la cámara fotográfica en un momento determinado.

Para el control de la plataforma Altazimut fue necesario el estudio de sensores y tarjetas de control de motores disponibles en tiendas electrónicas online, enfocándose en las herramientas de desarrollo conocidas basadas en microcontroladores de Microchip ®, y open hardware como Arduino que facilitaran el desarrollo del trabajo de grado. La selección de la herramienta de implementación para la creación de interfaz de usuario fue usada MATLAB® GUIDE por el conocimiento previamente adquirido durante el curso del programa académico de maestría en ingeniería electrónica.
Desde la interfaz grafica se activaran dos tipos de control: uno de posición para cada movimiento de la plataforma giro y elevación, y el segundo un control de estabilización que mantendrá la línea de vista con la finalidad de tomar fotografías en aplicaciones que no necesiten una respuesta rapiada a corrección de perturbaciones externas.

En este caso de estudio se generarán perturbaciones con amplitudes con variaciones de ±10° del ángulo de elevación(θ) de la plataforma altazimut. Y para generar las perturbaciones externas en un solo eje de movimiento de baja frecuencia se construyó un mecanismo se mostrará en el capítulo de pruebas y resultados.

Se obtendrá un modelo de la dinámica desacoplada para la plataforma altazimut con la cámara fotográfica montada sobre esta (ver figura 2), con la finalidad de simular la posición en 2 ejes de movimiento de giro (heading) y elevación (pitch), estas coordenadas describen la orientación y movimiento rotacional como es modelado en [1] y donde el sistema de medida empleado para los ensayos se realiza con una unidad de medida inercial (IMU). La sensorica inercial (IMU) será atada al dispositivo para obtener una medición de la posición.

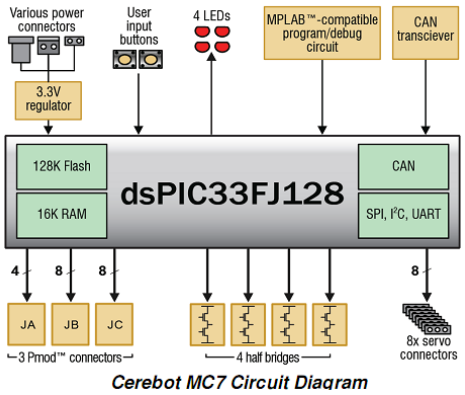


FIGURA 1. Diagrama de circuito Cerebot MC7™ basado en dsPIC. Tomado de **[2]**.

Para el procesamiento digital de señales y control de la plataforma altazimut se utilizara la tarjeta Cerebot MC7^{TM} en la figura 1 esta electrónica incorporada recursos de hardware con capacidades de procesamiento para algoritmos de hasta 80MHz basado en dsPIC33FJ128MC706A, periféricos como: entradas/salidas digitales, Conversores ADC(sigla en inglés de analogue to digital converter), módulos PWM (sigla en inglés de *Pulse-width modulation),* 2 puentes H para control de motores con MOSFET(sigla en inglés de *Metal-oxide-semiconductor Field-effect transistor*) *de potencia hasta 30V* a 8A para cada uno, interfaz de comunicaciones CAN(sigla en inglés de *Controller Area Network*), *SPI(sigla en inglés de Serial Peripheral Interface), I^{2}C(sigla en inglés de Inter-Integrated Circuit)* entre otras, además de tener la posibilidad de utilizar lenguajes de alto nivel como C dentro de su entorno para el desarrollo del proyecto.

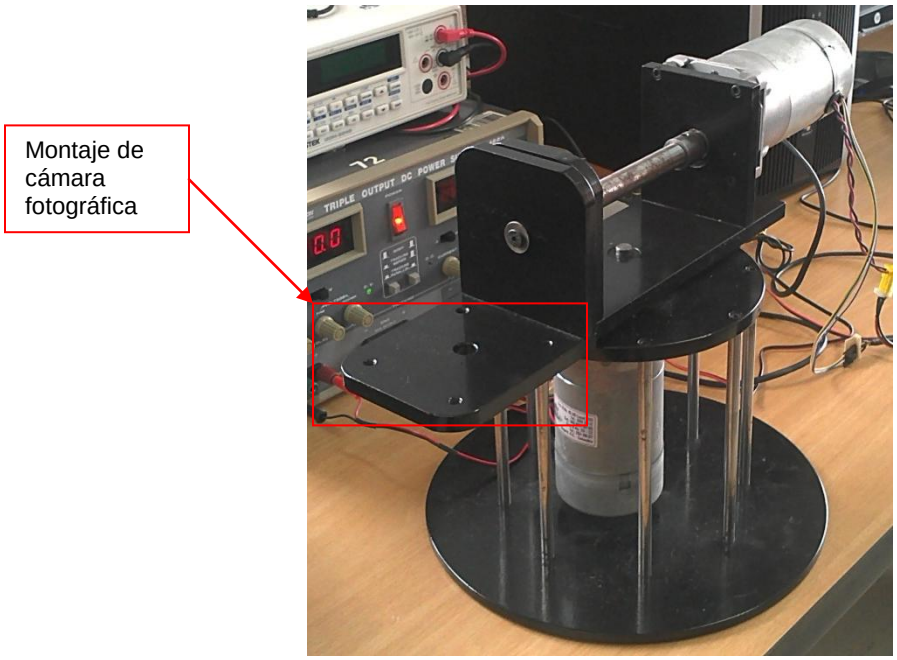


FIGURA 2. Plataforma altazimut, fuente el autor.

En la implementación del sistema se realizó una interfaz gráfica de usuario GUI que contendrá las principales variables de la IMU, cambio de setpoints y parámetros principales de controlador PID para facilitar la evaluación tanto del modelo simulado obtenido como de la técnica de control aplicada.

La simulación se realizó en entorno MATLAB® y/o SIMULINK® con el modelo dinámico obtenido, y la tarjeta Cerebot MC7^{TM} contendrá de manera embebida los algoritmos encargados de realizar la estabilización de la plataforma altazimut. Con la GUI se guardaran los datos de cada experimento para la validación del modelo dinámico analítico. Los datos obtenidos serán analizados para la posterior muestra de los resultados.

CAPITULO 1.

FUNDAMENTOS TEÓRICOS Y CONCEPTUALES DEL CONTROL

1.1. SENSORES INERCIALES

Los sensores inerciales miden aceleraciones, velocidades angulares basadas en las leyes físicas que estudian el movimiento de los cuerpos sobre su propio eje, y normalmente está compuesto por acelerómetros, giróscopos y magnetómetros los cuales conforman una unidad de medición inercial también conocida como IMU. Los acelerómetros miden la aceleración lineal con que se mueve el sensor, los giróscopos la velocidad angular y los magnetómetros dan información acerca del norte magnético, las unidades IMU pueden estar compuestas por estos tres tipos de sensores o la combinación de 2 de estos, además con estos tres sensores es posible estudiar el movimiento de la IMU en el plano o el espacio.

1.1.1. FUNDAMENTOS BASICOS Y APLICACIÓN

Para el proyecto de tesis solamente se hablará de la tecnología usada ya que para mediciones de aceleraciones existen varias tecnologías y/o principios físicos para su fabricación, entre las que se puede mencionar: los piezoeléctricos, piezoresistivos, capacitivos y mecánicos; además de los sensores magnéticos basados en efectos físicos magneto-resistivos, de inductancia y efecto hall. De esas tecnologías solamente se hablará de las que fueron usadas en el presente proyecto nos referimos a los MEMS (Micro-Electromechanical Systems), por sus siglas en ingles.

1.1.1. Sensores MEMS

Los avances en las técnicas de micro fabricación de sistemas micro electromecánicos (MEMS), ha conducido al desarrollo de dispositivos integrados, capaces de medir

distintas variables de interés en un solo chip, estas ventajas logran que estos pequeños dispositivos sean adecuados para ser implementados en distintas aplicaciones. La miniaturización lograda por estas nuevas técnicas de fabricación hace que estos sean dispositivos atractivos para ser implementados en un sistema de seguimiento y medición de las dinámicas del movimiento.

1.1.2. Acelerómetros MEMS

Los acelerómetros inerciales son sensores que miden la segunda derivada de la posición. Un acelerómetro mide la fuerza de inercia generada cuando una masa es afectada por un cambio de velocidad. Una característica especial que presenta el acelerómetro es que puede ser utilizado también como sensor de inclinación y/o medición de ángulos, para lo cual se usa la aceleración de la gravedad como un vector para determinar la orientación del objeto en el espacio (ver [1]), lo que más adelante se explicará con detalle.

En general un acelerómetro consiste en una masa de prueba suspendida por vigas compatibles ancladas a un marco fijo (ver figura 1-1, parte derecha). La masa de prueba tiene una masa M, las vigas en suspensión tiene una constante de resorte efectiva *K*, y un factor de amortiguación (*D*) que afecta el movimiento dinámico de la masa. El acelerómetro puede ser modelado como un sistema de masa-resorte-amortiguador de segundo orden. La aceleración externa desplaza el bastidor de soporte con relación a la masa de prueba, que a su vez cambia la tensión interna en el resorte de suspensión. Tanto este desplazamiento relativo y el estrés de suspensión de viga se pueden utilizar como una medida de la aceleración. Mediante el uso de la segunda ley de Newton y el modelo de acelerómetro, la función de transferencia mecánica se puede conseguir [3].

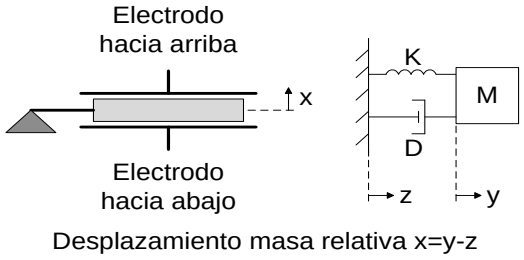


FIGURA 1-1. Estructura general del acelerómetro y modelo mecánico agrupado, tomado de [3].

$$H(s) = \frac{x(s)}{a(s)} = \frac{1}{s^2 + \frac{D}{M^2} + \frac{K}{M}} = \frac{1}{s^2 + \frac{\omega_r}{Q}s + \omega_r^2} \qquad (1 \cdot 1)$$

Donde a es la aceleración externa, x es la masa de prueba desplazada, $\omega_r = \sqrt{K/M}$ es la frecuencia de resonancia natural, y $Q = \sqrt{KM}/D$ es el factor de calidad. La sensibilidad estática del acelerómetro se muestra en $(1 \cdot 2)$.

$$\frac{x_{estatica}}{a} = \frac{M}{K} = \frac{1}{\omega_r^2} \qquad (1 \cdot 2)$$

La frecuencia de resonancia de la estructura se puede aumentar mediante el aumento de la constante del resorte y la disminución de la masa de prueba, mientras que el factor de calidad del dispositivo se puede incrementar mediante la reducción de amortiguación y mediante el aumento de masa de prueba y la constante del resorte. Por último, la respuesta estática del dispositivo se puede mejorar mediante la reducción de su frecuencia de resonancia.

Parámetro	Automovilismo	Navegación
Rango	±50g(airbag), ±2g(sistema estabilizacion de vehiculos)	±1g
Rango de Frecuencia	DC-400Hz	DC-100Hz
Resolucion	<100mg(airbag) , <10mg(sistema estabilizacion de vehiculos)	<40µg
Sensibilidad fuera de eje	<5%	<0.1%
Nolinealidad	<2%	<0.1%
Max choque en 1mseg	>2000g	>10g
Rango Temperatura	-40 ºC a 85 ºC	-40 ºC a 80 ºC

Tabla 1-1. Especificaciones típicas de Acelerómetros para aplicaciones de automovilismo y navegación inercial, Tomado de [3].

En general los acelerómetros se especifican según por su sensibilidad, alcance máximo funcionamiento, respuesta de frecuencia, resolución, la no linealidad a gran escala, offset, sensibilidad fuera de eje, y la supervivencia de choque. Acelerómetros MEMS se utilizan en una amplia gama de aplicaciones, sus especificaciones requeridas son también depende de la aplicación y cubren un espectro bastante

amplio. La tabla 1-1 resume los parámetros de rendimiento típicas de acelerómetros con una resolución media para aplicaciones de automovilismo y de alto rendimiento para aplicaciones de navegación inercial.

1.1.3. Magnetómetros MEMS

Los magnetómetros son sensores de medición de campo magnético y en este caso específico la del campo magnético terrestre, el principio de funcionamiento es la medición a través de magneto-resistencias que cambian su valor en función del campo que las atraviesa en su dirección usando sensores AMR (Anisotropic Magneto-Resistive por sus siglas en ingles), este fenómeno se produce en materiales ferrosos y se trata de un cambio en la resistencia cuando un campo magnético se aplica en una tira delgada de material ferroso llamada permalloy (ver figura 1-2) y forma cuatro elementos resistivos para convertirse en un sensor de puente de Wheatstone(ver figura 1-3) el principio de funcionamiento es tratado en [4], donde dice que trabaja en la región lineal de 45° del ángulo θ formado entre el momento magnético de (**M**) y vector de flujo de corriente (**I**) al ser aplicado el campo magnético incluyendo el terrestre, el material permaloy cambia su resistencia.

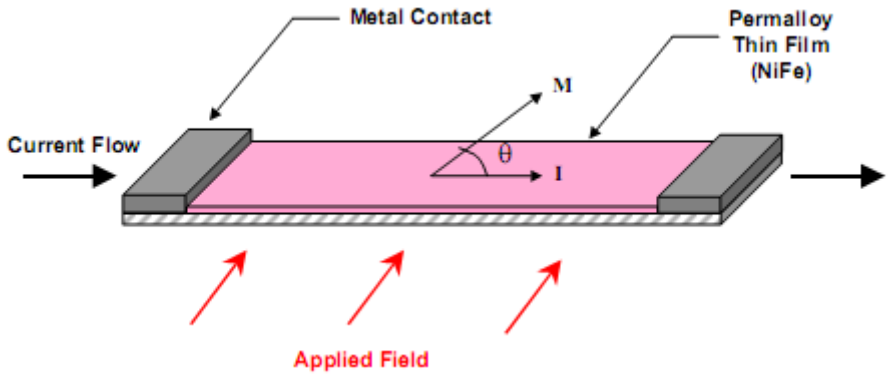


FIGURA 1-2. Elemento ARM, tomado de **[4]**.

El fabricante para crear el sensor con los elementos AMR usualmente los orienta en forma de diamante con los extremos conectados entre sí por metalización para formar el puente de Wheatstone (ver figura 1-3), y la diagonal de los elementos (R1, R2, R3 y R4) son iguales. Pero R1 y R2 tienen polaridad opuesta a R3 y R4. Cuando un campo magnético externo es aplicado, si R1-R2 incrementa la resistencia ΔR, y R3-R4 decrece la resistencia ΔR.

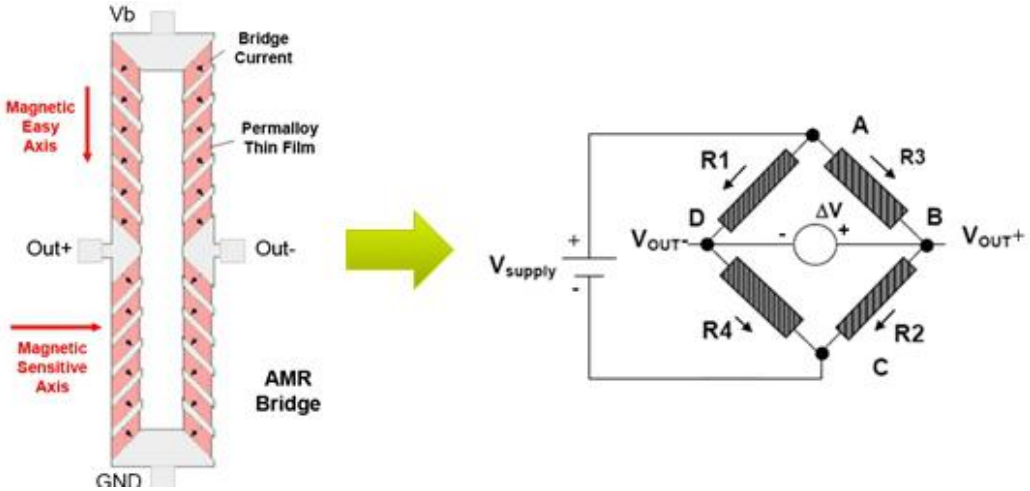


FIGURA 1-3. Puente de Wheatstone con tecnología ARM, tomado de **[5]**.

Explicando un poco más el principio de medición con las AMR, con el puente de Weatstone se realiza una conexión superior e inferior de los cuatro elementos idénticos y se les da un estímulo de corriente directa (DC) en la forma de una tensión de alimentación (Vsupply), con las conexiones de los lados restantes a ser medidos (Vout- , Vout+). Sin un campo magnético suministrado (0 gauss), los contactos laterales deben estar a la misma tensión, a excepción de una pequeña tensión de offset debido a las tolerancias de fabricación de los elementos de AMR. Estos contactos secundarios producen una tensión diferencial (ΔV) como una función de la tensión de alimentación, la relación de AMR, y el ángulo theta (θ); que es el ángulo entre el flujo de corriente y el elemento de magnetización (**M**). En otras palabras el cambio en el ángulo θ resulta expresado en el elemento AMR resistivo con el cambio de ΔR y lo que puede detectarse con el cambio de la salida de voltaje ΔV en el monitoreo del puente de Weatstone. Las siguientes ecuaciones explican el principio de trabajo del magnetómetro.

$$R1_0 = R2_0 = R3_0 = R4_0 = R \qquad (1 \cdot 3)$$

Cuando no se aplica campo externo:

$$\Delta V = V_{out+} - V_{out-} = V_{supply} \bullet \left[\left(\frac{R_2}{R_2 + R_3}\right) - \left(\frac{R_4}{R_4 + R_1}\right)\right]$$

$$= V_{supply} \bullet \left(\frac{R}{2R} - \frac{R}{2R}\right) = 0 \qquad (1 \cdot 4)$$

Cuando es aplicado un campo externo:

$$\Delta V = V_{supply} \bullet \left[\frac{R - \Delta R}{(R - \Delta R) + (R + \Delta R)} - \frac{R + \Delta R}{(R + \Delta R) + (R - \Delta R)}\right]$$

$$= V_{supply} \bullet \frac{\Delta R}{R} \qquad (1 \cdot 5)$$

1.2. IDENTIFICACION DE SISTEMAS DINAMICOS

Una gran cantidad de aplicaciones de las diferentes técnicas de control modernas en los diferentes clases de procesos, ven evidenciada su efectividad al partir de su implementación sobre un modelo matemático bastante aproximado al comportamiento real de la planta o proceso en estudio, situación que demuestra la gran importancia de poder contar con buen modelo de partida y/o el dispendioso estudio de la planta o proceso físico para la mejor obtención posible.

1.2.1. Definición

La identificación de los sistemas dinámicos consiste básicamente en la obtención de un modelo matemático aproximado para describir un sistema dinámico dado, a partir de un conjunto de datos experimentales de ENTRADA-SALIDA obtenidos del sistema y/o en la información basada en las leyes de la física que gobiernan el sistema en estudio.

La identificación de sistemas en gran parte tiene un enfoque experimental para el cual se deben tener en cuenta en su desarrollo los siguientes aspectos:

- **Planeación Experimental:** Es la realización de los experimentos que pueden ser con métodos clásicos basados en la respuesta al impulso o a funciones senoidales las cuales deben siempre excitar todos los modos de la planta o proceso.

- **Selección de la Estructura Modelo:** La estructura del modelo a considerar se deriva del conocimiento a priori que se tiene del sistema.
- **Estimación de Parámetros:** Los parámetros desconocidos en el proceso de identificación son generalmente los coeficientes del polinomio como también el orden del mismo.

  Estos parámetros pueden ser estimados a partir de dos clases de métodos:
- **Métodos fuera de línea:** Genera estimaciones a partir de un bloque de datos tomado previamente del proceso.
- **Métodos en línea:** Genera estimaciones de manera recursiva conforme se van obteniendo nuevos datos en tiempo real del proceso. Son la única opción cuando la identificación es usada como parte de un control adaptativo o en caso de parámetros variantes en el tiempo.

1.2.2. Clasificación

Los métodos de identificación pueden clasificarse también en función de los modelos obtenidos, de esta manera se pueden diferenciar entre: técnicas de identificación **no paramétricas** que se basan en modelos no paramétricos y técnicas de identificación **paramétricas**, que se basan en modelos paramétricos.

Dentro de las denominadas técnicas de identificación no paramétricas se pueden citar como las más importantes:

- **Análisis de la Respuesta Transitoria:** Se basa en la obtención de la respuesta del sistema a un impulso o a un escalón y la salida registrada genera el modelo correspondiente.
- **Análisis de Correlación:** Es un método en el dominio del tiempo, el cual obtiene la función de correlación entre las variables de interés y como caso especial una función de ponderación.
- **Técnicas Frecuenciales:** Son utilizadas para estimar la respuesta en frecuencia del sistema, siendo las más utilizadas el Análisis de Fourier y el Análisis Espectral.

Todas las anteriores técnicas son aplicables a procesos lineales o linealizables. Para su utilización no se debe suponer ningún tipo de estructura para el modelo y los resultados obtenidos son de tipo grafico.

En el caso de los métodos de identificación paramétricos, se debe tener en cuenta una cierta estructura para el modelo. Los parámetros del modelo se calculan minimizando ciertos criterios de error dependiendo del dominio en el cual se calculan los datos, por lo cual se pueden distinguir dos tipos de técnicas las Frecuenciales y las Temporales.

1.2.3. Pasos para la Identificación del Modelo

En general las etapas a seguir para la identificación del modelo de un proceso o sistema son:

- Diseño del experimento de identificación.
- Observación y mejora de la calidad de los datos capturados.
- Determinación de la estructura del modelo.
- Estimación de los parámetros (Si es el caso).
- Validación del Modelo.

1.3. MODELAMIENTO

1.3.1. Introducción

Un sistema representa una unidad donde se hacen tratamientos físicos o químicos de materiales que puede ser contrastada con un modelo que representa una descripción matemática del sistema real. La disposición de varios sistemas unidos entre sí por flujos comunes de materiales y/o información constituye un proceso. La salida del proceso es una función no solamente de las características de sus sistemas (o subsistemas) sino también de sus interacciones o interrelaciones. Una propiedad del sistema o de su entorno a la que se le puede asignar valores numéricos arbitrarios se denomina como un parámetro; también puede ser una constante o el coeficiente de una ecuación.

El estudio de un proceso, mediante la manipulación de su representación matemática o de su modelo físico, constituye una simulación. Los estudios clásicos de un proceso en estado estacionario se complementan con un análisis dinámico, lo que exige un conocimiento de los criterios de estabilidad y de los métodos de operación para evaluar exitosamente el funcionamiento del proceso.

El análisis de sistemas se refiere al reconocimiento y definición de problemas, su planteamiento o modelamiento mediante la aplicación de principios científicos y el desarrollo de procedimientos de solución con cuyos resultados se adquiera una total comprensión de la situación.

1.3.2. Tipos de modelos de sistemas

Debido a su utilización en diversos campos de la ciencia, es imposible incluir dentro de una sola definición las diferentes acepciones de la palabra modelo. Un sistema se puede modelar mediante, ya sea, una construcción física o analógica, una representación gráfica o un mapa, un enunciado teórico o un planteamiento matemático. Es decir, se pueden describir los siguientes tipos de modelos:

- **Modelos Físicos:** Son construcciones materiales que representen sistemas como barcos, plantas pilotos, maquetas de edificios y otros.

- **Modelos Analógicos:** Son construcciones materiales que representen circuitos eléctricos, electrónicos o mecánicos.

- **Teorías Provisionales:** Son postulaciones que explican comportamientos fenomenológicos en sistemas como la de los gases ideales o la de la gota de líquido para la nucleación.

- **Gráficos o Mapas:** Son representaciones mediante símbolos convencionales de estructuras de sistemas que explican en algunos casos su organización o su distribución o su logística, etc. Por ejemplo, la representación de un proceso químico mediante su diagrama de flujo.

- **Enunciados matemáticos y modelos en forma de símbolos:** Son sistemas de ecuaciones que expresan simbólicamente el fenómeno que se desarrolla en el sistema. Por ejemplo, el modelamiento matemático que exprese el flujo y los cambios de materia y energía a través de un reactor ideal de mezcla completa.

1.3.3. Ventajas del modelado

El análisis y modelado de procesos presentan las siguientes ventajas:

- **Experimentación Continua:** Es posible estudiar procesos existentes en una forma más rápida, económica y completa que en la planta real. La simulación puede aumentar o reducir el tiempo real de una forma análoga a como una cinematográfica acelera o retarda las imágenes; de esta forma se puede observar más fácilmente la operación del sistema.

- **Extrapolación:** Con un modelo matemático adecuado se pueden ensayar intervalos extremos de las condiciones de operación, que pueden ser impracticables o imposibles de realizar en una planta real. También es posible establecer características de funcionamiento.

- **Estudio de conmutabilidad y evaluación de otros planes de actuación:** Se pueden introducir nuevos factores o elementos de un sistema y suprimir otros antiguos al examinar el sistema con el fin de ver si estas modificaciones son compatibles. La simulación permite comparar distintos diseños y procesos que todavía no están en operación y ensayar hipótesis sobre sistemas o procesos antes de llevarlos a la práctica.

- **Repetición de experimentos:** La simulación permite estudiar el efecto de la modificación de las variables y parámetros con los resultados producibles. En el modelo matemático se puede introducir o retirar a voluntad un error, lo cual no es posible en la planta real.

- **Control de cálculo:** La simulación constituye una importante ayuda material para el estudio de los sistemas de control con lazos abiertos y cerrados.

- **Ensayo de sensibilidad:** Se puede ensayar la sensibilidad de los parámetros de costos y básicos del sistema; por ejemplo, un incremento de un 10 % en la velocidad de alimentación podrá tener, según los casos, un efecto mínimo o muy importante sobre el funcionamiento del sistema.
- **Estudio de la estabilidad del sistema:** Se puede examinar la estabilidad de sistemas y subsistemas frente a diferentes perturbaciones.

El modelado de un sistema dinámico es la obtención de un conjunto de ecuaciones matemáticas que describen el comportamiento de un sistema físico.

El modelado de un sistema dinámico consta de tres fases:

La identificación de las señales de entrada y señales de salida las cuales dependerán de la aplicación que se tenga para el modelo; además de las señales internas y los parámetros (constantes) a tener en cuenta.

Escribir las relaciones matemáticas que relacionan las señales de entrada y salida de cada elemento del sistema.

Adicionar las ecuaciones que acoplan unos elementos con otros. Se obtiene un modelo en espacios de estado o mediante funciones de transferencia del conjunto.

Este proceso se sigue muchas veces de forma inconsciente, además las ecuaciones resultantes del modelado de distintos sistemas tienen a menudo la misma forma, lo que hace posible el establecimiento de analogías.

1.3.4. Modelado de Sistemas Eléctricos

El modelado de un sistema eléctrico consiste en la descripción de este mediante una serie de ecuaciones generalmente diferenciales que relacionan las señales de interés (entradas y salidas) y que en muchos casos son voltajes y corrientes.

Una vez que se dispone de las ecuaciones que describen cada uno de los elementos del sistema o circuito, seguidamente con la aplicación de las Leyes de Kirchhoff se puede describir las ecuaciones de todo el sistema o circuito.

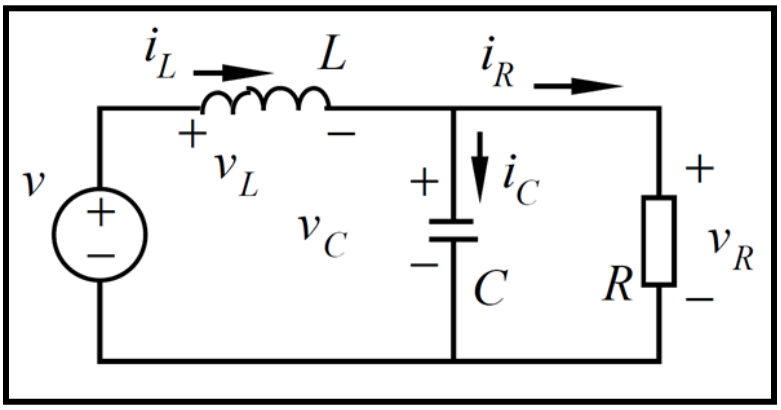


FIGURA 1-4. Circuito Eléctrico, fuente el autor.

Considérese el circuito eléctrico de la figura 1-4, aplicando un voltaje $V(t)$ (entrada), se desea conocer el voltaje $V_R(t)$ (salida).

Las ecuaciones que describen cada componente del circuito son:

$$V_L = L \times \frac{di_L}{dt} \quad (Voltaje\ en\ la\ bobina) \qquad (1 \cdot 6)$$

$$i_C = C \times \frac{dv_C}{dt} \quad (Corriente\ en\ el\ condensador) \qquad (1 \cdot 7)$$

$$V_R = R \times i_R \quad (Voltaje\ en\ la\ resistencia) \qquad (1 \cdot 8)$$

Ahora aplicando las Leyes de Kirchhoff tenemos que:

$$V = V_L + V_C \qquad i_L = i_C + i_R \qquad V_R = V_C \qquad (1 \cdot 9)$$

$$V = V_C + L \times \frac{di_L}{dt} \qquad i = \frac{V_R}{R + C} \times \frac{dv_C}{dt} \qquad (1 \cdot 10)$$

Por tanto realizando las reducciones respectivas la ecuación que describe el circuito es:

$$V = V_C + \left(\frac{L}{R}\right) \times \frac{dv_C}{dt} + (L \times C) \times \frac{dv_C}{dt} \qquad (1 \cdot 11)$$

1.3.5. Modelado de sistemas mecánicos

En los sistemas mecánicos las fuerzas, desplazamientos, velocidades y aceleraciones son las variables comúnmente encontradas. La ecuación fundamental es la 2da Ley de Newton.

$$\vec{f}(t) = M\vec{a}(t) = M\frac{d\vec{x}(t)}{dt} \qquad (1 \cdot 12)$$

Que indica que las suma de fuerzas que actúan sobre un cuerpo es igual al producto de su masa por la aceleración que este alcanza.

Para sistemas con desplazamiento angular en una dimensión, la ley de Newton se modifica; el par neto es igual al momento de inercia multiplicado por la aceleración angular.

$$m(t) = J\alpha(t) = J\frac{d\omega}{dt} = J\frac{d^2\theta}{dt^2} \qquad (1 \cdot 13)$$

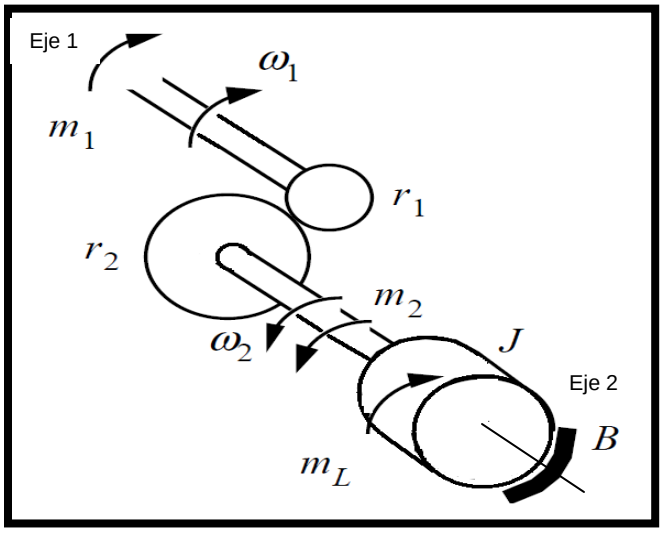


FIGURA 1-5. Transmisión mecánica, fuente el autor.

Donde $m(t)$ es el par aplicado, *J* es el momento de inercia, *y* {θ, ω, α} la posición, velocidad y aceleración angular respectivamente. En estos sistemas puede haber pares resistentes debidos a la elasticidad de los ejes y al rozamiento.

Para el sistema de transmisión de la figura 1-5, sobre el eje 1 se aplica un par $m_1(t)$, haciéndolo girar a una velocidad $\omega_1(t)$, de igual manera está acoplado a otro eje 2, atravez de dos ruedas de radios r_1 y r_2 que ruedan sin deslizar.

En el eje 2 hay conectada una carga que tiene un momento de inercia *J* y sobre el que actúa un par resistente de carga m_L. Además sobre el tambor hay un par de rozamiento viscoso proporcional a la velocidad, con constante de proporcionalidad *B*.

En las ruedas se conserva toda la potencia (par x velocidad angular), de manera que se obtiene la siguiente relación:

$$\frac{r_2}{r_1} = \frac{m_2(t)}{m_1(t)} = \frac{\omega_1(t)}{\omega_2(t)} = \frac{\theta_1(t)}{\theta_2(t)} \qquad (1 \cdot 14)$$

Donde $m_2(t)$ es el par aplicado en el eje 1 visto desde el eje 2 y θ_1 y θ_2 los ángulos girados por cada uno de ellos respectivamente.

Por tanto:

$$m_2(t) = J\frac{d\omega_2}{dt} + B\omega_2 + m_L \qquad (1 \cdot 15)$$

De las dos ecuaciones anteriores se puede elaborar el diagrama de bloques del sistema obteniéndose el siguiente esquema: figura 1-6.

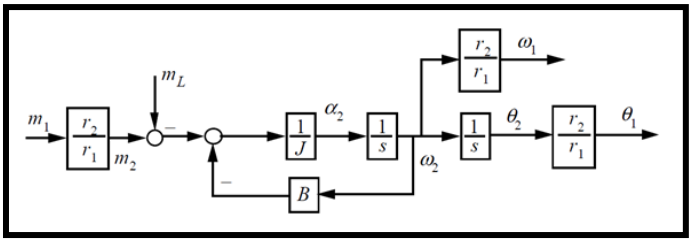


FIGURA 1-6. Diagrama de bloques del sistema.

1.4. IMPLEMENTACIÓN

La implementación física de cualquier sistema o planta está sujeto a diferentes aspectos ya que depende de su aplicación las prioridades cambian. Estos aspectos son entre otros los siguientes:

- Robustez.
- Precisión.
- Eficiencia.
- Tamaño.
- Presupuesto.
- Tiempo de Construcción.
- Medio Ambiente de aplicación.

Independiente de los aspectos anteriormente mencionados existen diferentes topologías para las diferentes clases de sistemas (mecánicos, eléctricos, térmicos, biológicos, eléctricos, humanos, astronómicos, neumáticos, tecnológicos, para medir el tiempo, para medir la velocidad, para medir el peso, para medir la cantidad de agua caída, entre otros.) utilizados muy frecuentemente por los diferentes fabricantes de equipos, dispositivos, industrias, etc.

1.4.1. Sistemas Mecánicos

1.4.1.1. Definición

Un sistema mecánico es un conjunto de elementos dinámicamente relacionados, que permiten producir, transmitir, regular o modificar movimiento.

Cuando aparecieron las primeras máquinas todas se basaban en sistemas, que utilizaba la energía de los músculos de los seres humanos o animales para moverlas. Luego fueron apareciendo otras formas de energía que ayudaron a generar movimiento para que operaran estos mecanismos como la energía térmica proveniente del carbón y la energía eléctrica.

1.4.1.2. Elementos Básicos

<table>
<tr><td colspan="4">Son todos aquellos elementos que relacionados forman un sistema mecánico y que veremos en detalle a continuación.</td></tr>
<tr><td>  </td><td>MANIVELA:
Componente mecánico que cambia la velocidad de movimiento de otros operadores.</td><td> 
1</td><td>LEVA:
Componente mecánico que al girar cambia el movimiento de otro operador en ascenso y descenso.</td></tr>
<tr><td> 
2</td><td>POLEA: Es una rueda con un eje apoyado por el cual puede pasarse una cadena o correa. Una manera fácil de fabricar las ruedas en tu s proyectos puede ser con cartón o con tapas de frascos. Los ejes con palitos de maqueta de madera y las correas con elásticos.</td><td>  </td><td>BIELA: Componente mecánico que aplicado debidamente a otro que da vueltas convierte su movimiento de giro a otro de vaivén. Una manera fácil de fabricarlo para tus proyectos es con un palito de helado o un trozo de madera delgado con uno o dos orificios en las puntas.</td></tr>
<tr><td>PALANCA:</td><td>DE PRIMER GÉNERO</td><td>DE SEGUNDO GÉNERO</td><td>DE TERCER GËNERO</td></tr>
<tr><td>Se compone de una barra rígida y de un punto de apoyo o fulcro (A). Sobre la barra rígida se aplican una fuerza llamada potencia (P) y otra llamada resistencia (R).
Según la colocación de estos tres elementos (punto de apoyo, potencia y resistencia), las palancas pueden ser de tres tipos.</td><td>El punto de apoyo se encuentra entre la potencia y la resistencia. Ejemplo: un balancín.
 

 
Imagina que hay niños jugando en él.</td><td>La resistencia se encuentra entre la potencia y el punto de apoyo. Mientras más cerca este la carga en la carretilla del punto de apoyo, (la rueda), más sencillo es desplazarla.
 

 
3</td><td>La potencia se encuentra entre el punto de apoyo y la resistencia. Como la carga está más alejada del punto de apoyo la fuerza aplicada debe ser mayor. Ejemplo: pinza.
 

  </td></tr>
</table>

Tabla 1-2. Elementos mecánicos básicos

[1] Imagen extraída desde el libro: Gómez Olalla, L.A.– Silva Rodríguez, F.– Jiménez Álvarez, J. –Almaraz Martín, Á. "Tecnología 1" Editorial Mac Graw Hill. Madrid – España - 1993.
[2] Imagen extraída desde el libro: Gómez Olalla, L.A.– Silva Rodríguez, F.– Jiménez Álvarez, J. –Almaraz Martín, Á. "Tecnología 1" Editorial Mac Graw Hill. Madrid – España - 1993.
[3] Imagen extraída de www.melon.cl/consejos/ calidad/medida.htm

1.4.2. Sistemas Eléctricos y Electrónicos

1.4.2.1. Definición

Un sistema eléctrico y/o es un conjunto de elementos dinámicamente relacionados, que permiten generar, conducir y recibir corriente eléctrica. Dependiendo de cómo estén dispuestos los elementos dentro del o los circuitos, las fallas o daños causados serán variables. Un problema en un componente puede producir una falla general, dañando un área extensa o una falla local, sin interrumpir todo el sistema.

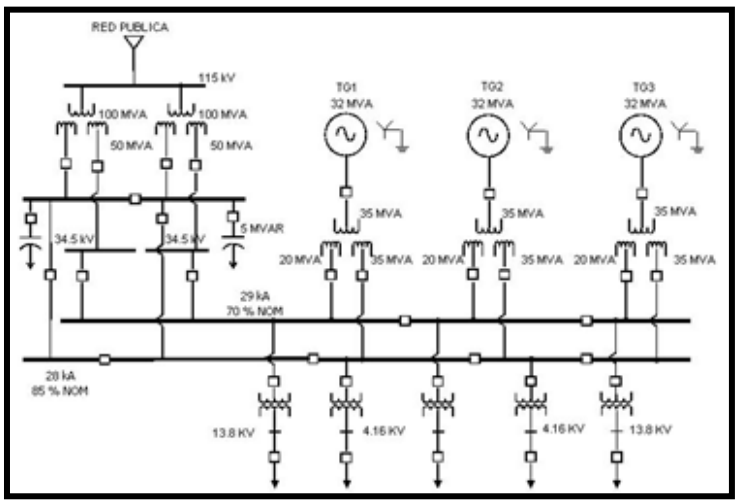


FIGURA 1-7. Ejemplo de sistema eléctrico.

FIGURA 1-8., Ejemplo de dispositivo electrónico.

1.4.3. Sistemas Mecatrónicos.

1.4.3.1. Definición

Un sistema mecatrónico es aquel sistema que recoge señales, las procesa y emite una respuesta por medio de actuadores, generando movimientos o acciones sobre el sistema en el que se va a actuar: Los sistemas mecánicos están integrados con sensores, microprocesadores y controladores. Los robots, las máquinas controladas digitalmente, los vehículos guiados automáticamente, etc. se deben considerar como sistemas mecatrónicos.

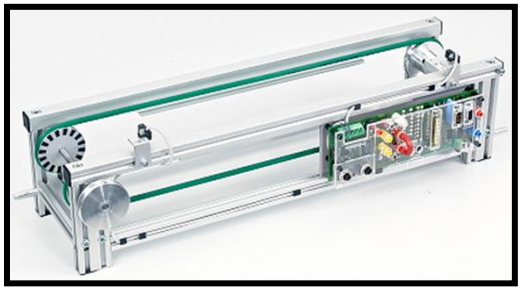


FIGURA 1-9. Ejemplo de dispositivo mecatrónico.

1.5. CONTROL

El control a nivel general trata de la medición del valor de cómo mínimo una variable controlada de un sistema en general, con el fin de variar intencionalmente el valor de otra variable manipulada para así poder corregir o limitar la desviación del valor medido, con respecto al valor deseado de la variable controlada.

Este control contempla dos aspectos a tener en cuenta para su implementación como los son la técnica de control (lógica de control) y la implementación del hardware (lógica operacional o programación sobre CPU).

De acuerdo a la técnica escogida a incorporar se selecciona la clase de hardware a utilizar, ya que técnicas con alto grado de complejidad son implementadas sobre programación de CPU.

CAPITULO 2.

DESCRIPCION DE HERRAMIENTAS Y HARDWARE USADO EN EL PROYECTO

En el presente capítulo se describirán los principales elementos usados para el desarrollo del trabajo de grado sus principales características así como su principio de funcionamiento según el caso; para el dsPIC se mencionaran los recursos hardware utilizados además del entorno de desarrollo sobre el cual fueron implementados los algoritmos de adquisición, comunicación, control de motores y PID.

Para la implementación de lo anteriormente mencionado en este proyecto se utilizaron las siguientes herramientas de hardware y software:

Cerebot MC7: es una tarjeta de desarrollo basado en un microcontrolador Microchip dsPIC® 16-bits controlador digital de señales fabricada por la empresa DIGILENT®. El Cerebot MC7 basado en el dsPIC33FJ128MC706A está destinado principalmente para ser utilizado como un controlador para dispositivos electro-mecánicos tales como motores de corriente continua. Esta proporciona cuatro circuitos de medio puente que son para 24V de hasta 5A. Estos medios puentes se pueden usar para controlar dos motores de corriente continua con escobillas, dos motores paso a paso bipolares, un motor de corriente continua sin escobillas, un motor paso a paso unipolar.

Seeeduino V3.0 (Atmega 328P): basada en la Arduino© Duemilanove fabricado por el facilitador de open hardware para electrónica Seeed Studio, para la integración del sensor inercial y compas por intermedio de protocolo de comunicaciones I^2C.

Sensor LSM303DLHC: del fabricante Pololu tienda de electrónica y robótica, esta pequeña placa combina un compas magnético de 3 ejes y un acelerómetro también de 3 ejes independientes que pueden ser configurados vía protocolo de comunicaciones serial I^2C.

Encoder Incremental: la plataforma altazimut utiliza 2 motoreductores Matsushita de 22VDC con motores de corriente directa con escobillas que tienen acoplados un encoder incremental de 2 canales A-B que proporcionan 100 pulsos por vuelta cada uno.

MPLAB-X y Compilador C-30: IDE MPLAB ® X es un programa de software que se ejecuta en un PC (Windows ®, Mac OS ®, Linux ®) para desarrollar aplicaciones para

microcontroladores Microchip y controladores de señales digitales. Se llama un entorno de desarrollo integrado (IDE), el compilador C-30 se integra en el MPLAB-X y es una completa herramienta compatible con el lenguaje ANSI C para los dispositivos Microchip de 16 bits: PIC24, dsPIC30F y dsPIC33F, más adelante se explicará el funcionamiento del compilador.

Arduino: El microcontrolador en la placa seeeduino se programa mediante el lenguaje de programación Arduino que se basa en C/C++.

Matlab: es una herramienta de desarrollo matemático con un entorno integrado (IDE) y lenguaje propio denominado M, dentro de este entorno fue desarrollada la GUI (creación de interface de usuario) para el trabajo de grado.

2.1. MICROCONTROLADOR dsPIC33FJ128MC706A

Para entender más adelante la implementación de los algoritmos de control debemos entender cómo funciona el hardware que tendrá de manera embebida el código que será ejecutado para cumplir las funciones con la finalidad de controlar la plataforma altazimut. Entonces podemos decir que los microcontroladores son computadores digitales integrados en un chip que cuentan con un microprocesador o unidad de procesamiento central (CPU), una memoria para almacenar el programa, una memoria para almacenar datos y puertos de entrada/salida. A diferencia de los microprocesadores de propósito general, como los que se usan en los computadores PC, los microcontroladores son unidades autosuficientes y más económicas. El funcionamiento de los microcontroladores está determinado por el programa almacenado en su memoria. Este puede escribirse en distintos leguajes de programación. Además, la mayoría de los microcontroladores actuales pueden reprogramarse repetidas veces. Por las características mencionadas y su alta flexibilidad, los microcontroladores son ampliamente utilizados como el cerebro de una gran variedad de sistemas embebidos que controlan máquinas, componentes de sistemas complejos, como aplicaciones industriales de automatización y robótica, domótica, equipos médicos, sistemas aeroespaciales, e incluso dispositivos de la vida diaria como automóviles, hornos de microondas, teléfonos y televisores.

Frecuentemente se emplea la notación µC o las siglas MCU (por sus siglas en ingles microcontroller unit) para referirse a los microcontroladores.

2.1.1. Características generales de los Microcontroladores

Las principales características de los microcontroladores son:

- **Unidad de Procesamiento Central (CPU):** Los dsPIC típicamente de 16 bits, pero también las hay de 8, 32 y hasta 64 bits con arquitectura Harvard, con memoria/bus de datos separada de la memoria/bus de instrucciones de programa, o arquitectura de von Neumann, también llamada arquitectura Princeton, con memoria/bus de datos y memoria/bus de programa compartidas. También con capacidades de procesamiento matemático extensivo
- **Memoria de Programa:** Es una memoria ROM (Read-Only Memory), EPROM (Electrically Programable ROM), EEPROM (Electrically Erasable/Programable ROM) o Flash que almacena el código del programa que típicamente puede ser de 1 kiloByte a varios MegaBytes.
- **Memoria de Datos:** Es una memoria RAM (Random Access Memory) que típicamente puede ser de 1, 2 4, 8, 16, 32 kiloBytes.
- **Generador del Reloj:** Usualmente un cristal de cuarzo de frecuencias que genera una señal oscilatoria de entre 1 a 80MHz, o también resonadores o circuitos RC.
- **Interfaz de Entrada/Salida:** Puertos paralelos, seriales (UARTs, Universal Asynchronous Receiver/Transmitter), I2C(Inter-Integrated Circuit), Interfaces de Periféricos Seriales (SPIs, Serial Peripheral Interfaces), Red de Area de Controladores (CAN,Controller Area Network), USB (Universal Serial Bus).
- **Otras opciones:**

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

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

2.1.2. Arquitectura dsPIC33F

Consultando al fabricante Microchip Technology Inc. describe los dsPIC® como controladores de señales digital (DSC) dsPIC33F que ofrecen el rendimiento de un DSP con la simplicidad de una MCU. El núcleo dsPIC33F está diseñado para ejecutar algoritmos de filtros digitales, circuitos de control digital de precisión de alta velocidad,

de audio digital y el procesamiento de voz, DSC dsPIC33F con periféricos para el control de motores que permiten el diseño de sistemas de control para motores de precisión y alto rendimiento, periféricos en chip diseñados específicamente para fuentes de alimentación de altas prestaciones, los periféricos son de alta velocidad y alta resolución como el PWM, ADC rápido y comparadores analógicos. Estos microcontroladores de la familia dsPIC33F son procesadores de arquitectura Harvard modificada de 16 bits con un conjunto de instrucciones mejorada. La figura 2-1 donde se muestra la arquitectura del motor DSP, doble acumulador de 40-bits, soporte de hardware para las operaciones de división, una CPU (Central Processing Unit por sus siglas en ingles) con extensiva capacidad de procesamiento matemático provista de una ALU(Arithmetic and Logic Unit por sus siglas en ingles) de 16bits. Dentro de las características está el manejo de interrupciones flexible y determinista, junto con una potente gama de periféricos. Además, acceso directo a memoria (DMA) permite libre transferencia de datos entre los varios periféricos y una RAM DMA dedicada.

Las principales características de recursos y periféricos que tiene el dsPIC33FJ128MC706A se resume en la tabla 2-1.

Memoria Flash de programa	Memoria RAM	Timers de 16bits	Input Capture	Output Cature	MCPWM	QEI	ADC	USART	SPI	I²C	CAN	E/S pines
128 Kbytes	16 Kbytes	9	8	8	8 canales	1	2ADC, 16canales	2	2	2	1	53

Tabla 2-1. Características dsPIC33FJ128MC706A, tomado de [6].

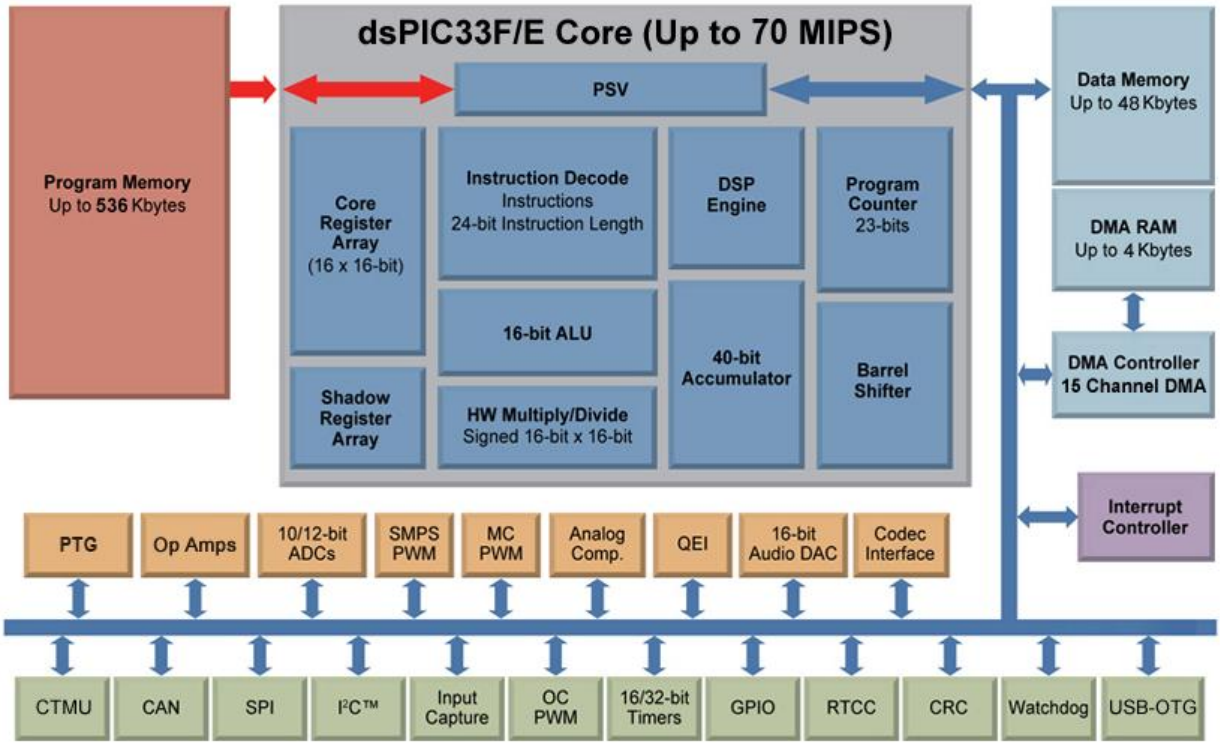


FIGURA 2-1. Arquitectura dsPIC33F, tomado de [7].

2.1.3. Módulo MCPWM del dsPIC33F

El MCPWM (Motor Control PWM por sus siglas en ingles), es uno de los recursos del dsPIC33FJ128MC706A para realizar la modulación por anchura de pulso de las 8 salidas que posee. El fabricante Microchip menciona en su datasheet [6] (hoja de datos) las aplicaciones soportadas y los modos de funcionamiento como “complementario”, “independiente”, “push-pull”, “multifase” y otros más, En este caso, el dsPIC33F dispone de 4 generadores PWM con 8 salidas, 4 “H” y 4 “L”. Cada uno de estos generadores tiene su propia base de tiempos y los pines de salida son nombrados como PWM1H/PWM1L hasta PWM4H/PWM4L (ver figura 2-2) de los cuales serán utilizados de pares en modo complementario.

Este módulo MCPWM genera una señal periódica con un el ciclo de trabajo (0-100%) los cuales son programables con los registros reservados para esta configuración y operación del mismo. La manera en que el dsPIC genera una base de tiempos en forma de rampa de subida o contador, es a partir del registro PTMR que recibe

directamente el reloj interno del dsPIC®. Los registros de configuración del MCPWM que para el trabajo de grado fueron utilizados son:

PTCON (PWM Time Base Control Register): este registro es usado para seleccionar el modo para la base de tiempo, reloj, preescala y postescala para base de tiempo, finalmente se habilita la base de tiempo para el conteo del timer.

PTMR (PWM Time Base Register): contienen el valor del conteo de la base de tiempo configurada en el PTCON y la dirección de conteo del mismo hacia arriba o hacia abajo.

PTPER (PWM Time Base Period Register): el valor base de tiempo de PWM está escrito en este registro, que determina la frecuencia de funcionamiento PWM.

PWMCON1 (PWM Control Register 1): la selección de cualquiera de los modos independientes y complementarias para cada par de PWM de I/O se lleva a cabo en este registro.

POVDCON (Override Control Register): es usado para habilitar los pines de salida que serán controlados por el generador PWM**.**

PDC1 (PWM Duty Cycle Register 1): escribe el valor en 16 bits, del ciclo de trabajo de la salida PWM para el par de pines 1.

PDC2 (PWM Duty Cycle Register 2): escribe el valor en 16 bits, del ciclo de trabajo de la salida PWM para el par de pines 2.

PDC3 (PWM Duty Cycle Register 3): escribe el valor en 16 bits, del ciclo de trabajo de la salida PWM para el par de pines 3.

PDC4 (PWM Duty Cycle Register 4): escribe el valor en 16 bits, del ciclo de trabajo de la salida PWM para el par de pines 4.

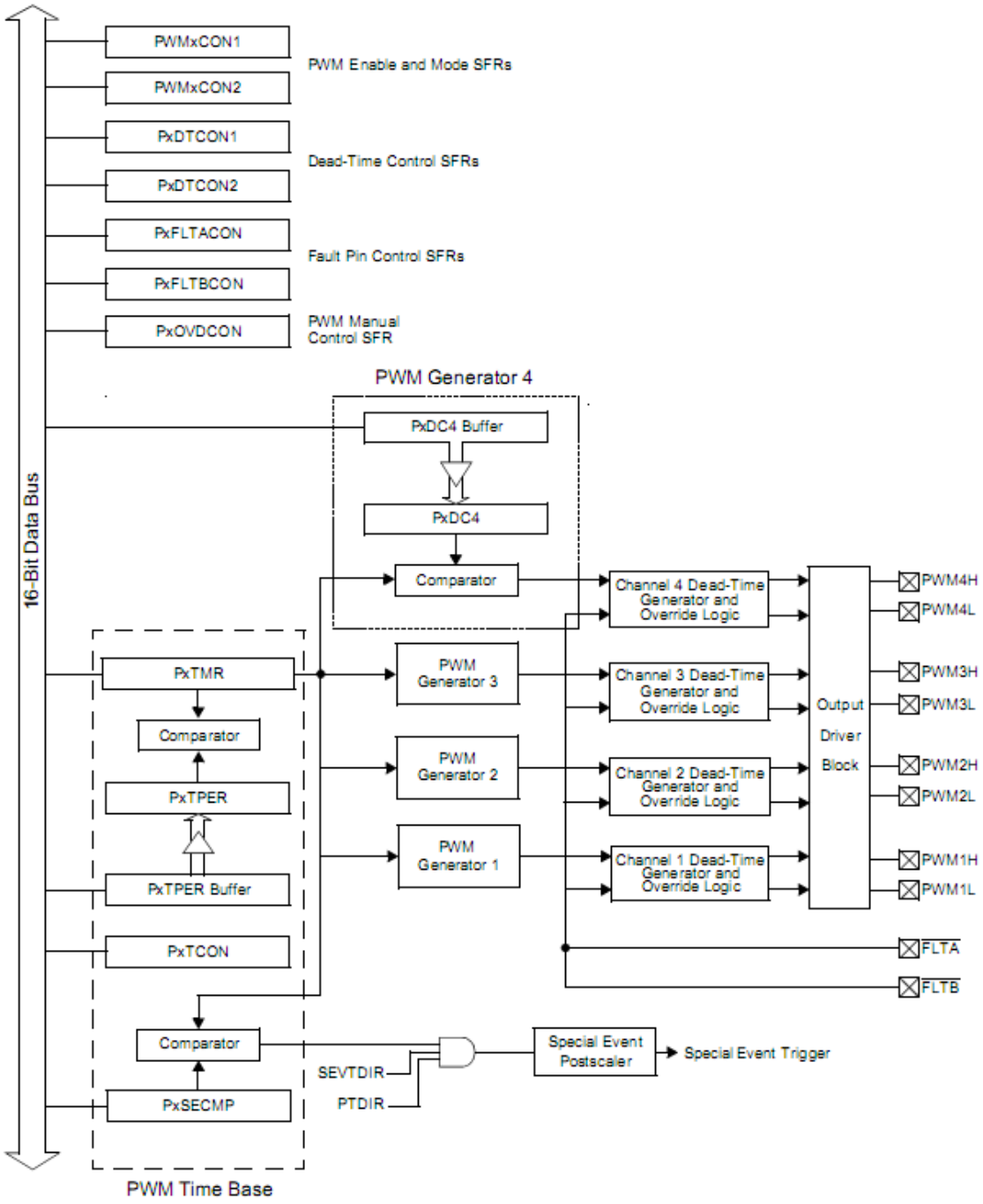


FIGURA 2-2. Módulo MCPWM dsPIC33F, tomado de [6].

La determinación de periodo de tiempo al que opera el módulo PWM es según como se configure el registro **PTPER** en modo de conteo free running o modo Up/Down (ver figura 2-3) que funciona como buffer para durante la operación, cuando este coincide con el valor del registro **PTMR,** base de tiempo del PWM cuando está en modo

Up/Down invierte el sentido de conteo con el siguiente flanco de reloj es así como lo describé el fabricante microchip en su datasheet [6].

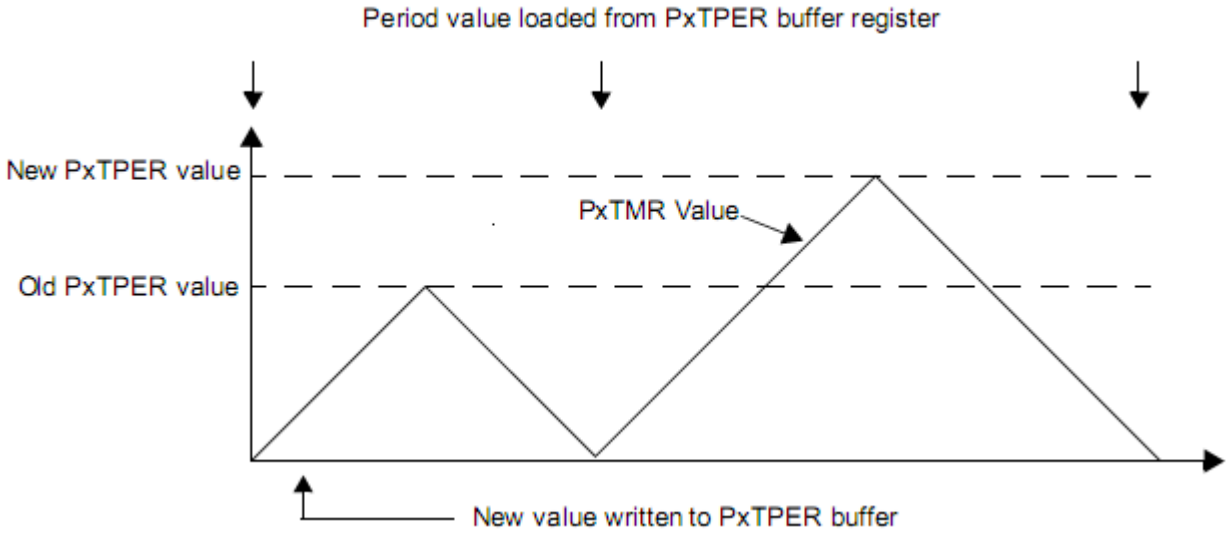


FIGURA 2-3. Actualización PWM en modo de conteo Up/Down, tomado de [6].

Ya que la actualización del buffer del periodo está directamente relacionada con la configuración de la frecuencia de operación en modo conteo Up/Down se debe utilizar la siguiente ecuación $(2 \cdot 1)$, que definirá el valor a cargar en el registro PTPER en función de la frecuencia de operación.

$$PTPER = \frac{F_{CY}}{F_{PWM} \times (PTMR\ Prescaler) \times 2} - 1 \qquad (2 \cdot 1)$$

2.1.4. Módulo QEI del dsPIC33F

El módulo QEI (quadrature encoder interface por sus siglas en ingles) está compuesto por 3 pines de entrada, las dos fases: Fase A (QEAx), Fase B (QEBx) y pulso Indice

(INDXx) que proporciona una interfaz para encoders incrementales que permite obtener datos de posición mecánicos. Los encoders incrementales o de cuadratura detectan la posición y la velocidad de sistemas de movimiento de rotación. Más adelante se explicará el principio de funcionamiento del Encoder.

Los canales Fase A y Fase B tienen una relación única. Si la Fase A va por delante de la Fase B, la dirección del motor se considera positiva o hacia adelante. Si la Fase B va por delante de la Fase A, se considera negativa o hacia atrás. El pulso que incrementa el Indice ocurre una vez por revolución y es usado como referencia para indicar la posición absoluta. Se observa un diagrama de tiempos relativo a estas tres señales en la figura 2-4.

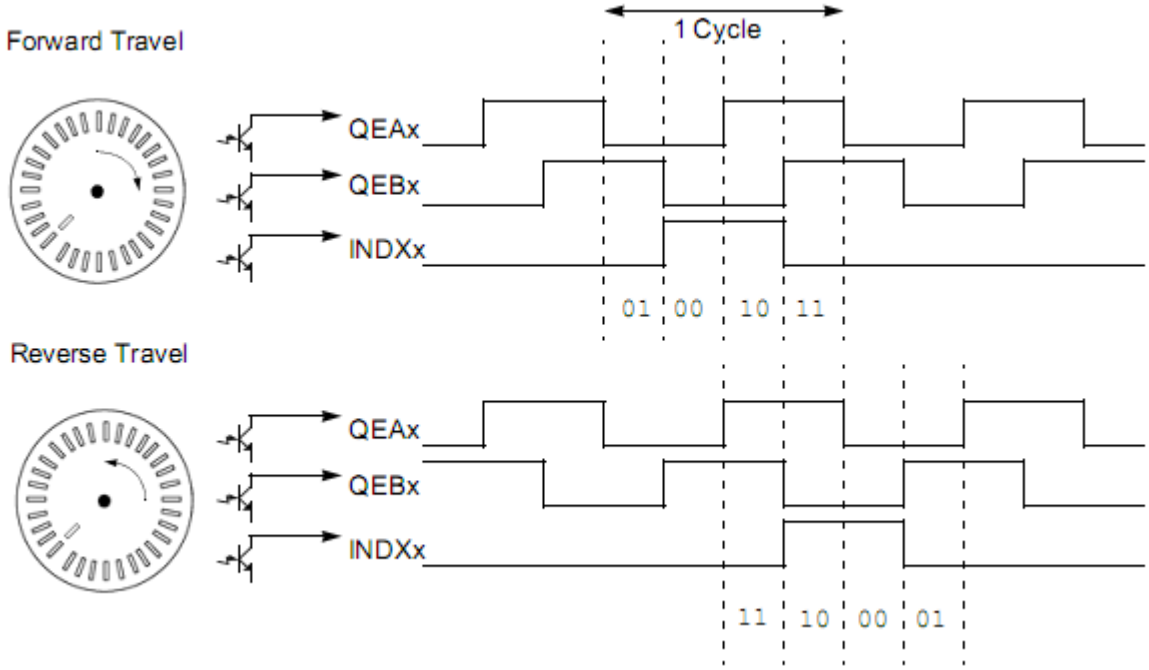


FIGURA 2-4. Interface de señales Encoder en Cuadratura, tomado de [6].

Los registros de configuración del módulo QEI son los siguientes.

QEICON (QEI Control Register): este registro controla la operación y proporciona flags de estado del módulo.

POSCNT (Position Count Register): este registro permite la lectura y escritura del contador de posición de 16 bits. Cuando este registro llega a su valor máximo, vuelve a cero y sigue incrementándose.

MAXCNT (Maximum Count Register): Mantiene un valor que es comparado con

el contador POSCNT en algunas operaciones como para setarlo a cero cuando son iguales.

2.1.5. Módulo USART del dsPIC33F

El módulo USART (Universal Asynchronous Receiver Transmitter por sus siglas en ingles) es canal de comunicación asíncrona serial full-dúplex que se comunica con los dispositivos periféricos y computadoras personales que utilizan protocolos como RS-232, RS-485, LIN y IrDA ®. El módulo también soporta la opción de control de flujo por hardware con $\overline{UxCTS}$ y $\overline{UxRTS}$ e incluye el codificador y decodificador IrDA®.

Las características principales del módulo UART son los siguientes:

- Full-Dúplex, 8 bits o 9 bits de transmisión y recepción de datos a través de los pines UxTX y UxRX (ver figura 2-5).
- Paridad Par, Impar o Ninguna (para datos de 8 bits).
- Uno o dos bits de parada.
- La función Auto-Baud Hardware.
- Opción de control de flujo de hardware con $\overline{UxCTS}$ y $\overline{UxCTS}$.
- Generador de velocidad de transmisión integrado (BRG) con Prescaler de 16 bits.
- Velocidades de transmisión que van desde 10 Mbps a 38 bps a 40 MIPS.
- Buffer de transmisión de datos, primero en entrar, primero en salir (FIFO).
- Buffer de recepción de datos FIFO.
- Paridad, encuadre y detección de errores por buffer de desbordamiento.
- Interrupciones de transmisión y recepción.
- El modo de bucle invertido para el apoyo diagnóstico.
- IrDA codificador y decodificador Lógico.

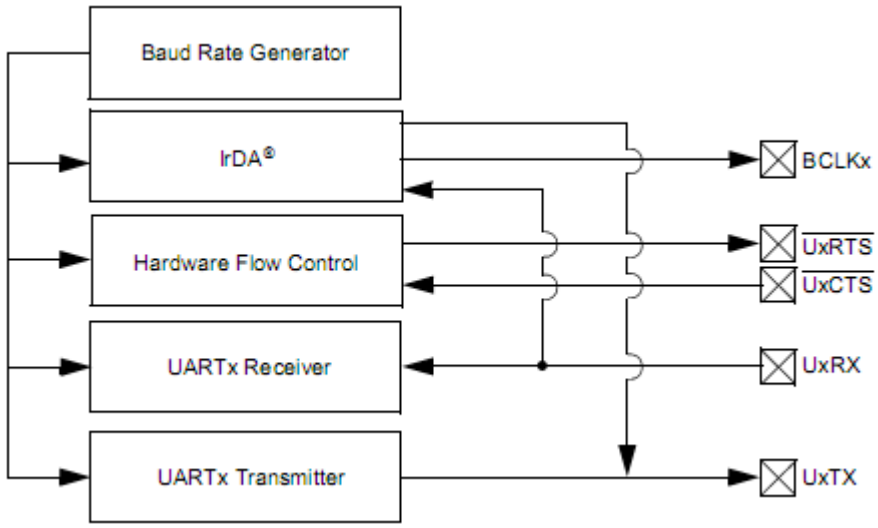


FIGURA 2-5. Diagrama de bloques simplificado de la USART, tomado de [6].

Los principales registros de configuración del módulo USART para controlar su modo de operación son los siguientes:

UMODE (UARTx Mode Register): en este registro se configura el modo de operación del módulo que incluye habilitación del mismo, habilitación del IrDA®, modo de control de flujo de datos, habilitar módulo cuando se encuentra en sleep, auto generador baudios, modo de alta o baja velocidad, configuración de paridad: par, impar, sin paridad, selección de 1 o 2 bits de parada de transmisión.

URXREG (UARTx Receive Register): en este registro se guardan los datos recibidos por comunicación serial.

UTXREG (UARTx Transmit Register): registro de solo escritura para enviar los datos de transmisión serial.

UBRG (UARTx Baud Rate Register): este registro es el encardado de guardar el valor de la configuración de la velocidad de baudios para la transmisión o recepción de datos asíncronos. Para calcular este valor es necesario utilizar la ecuación $(2 \cdot 2)$, si se configura el módulo en modo alta velocidad en el registro **UMODE** el bit **BRGH=1** donde F_{CY} es la frecuencia del ciclo de reloj del dsPIC33F.

$$UBRG = \frac{F_{CY}}{4 \times Baud\ rate} - 1 \qquad (2 \cdot 2)$$

2.1.6. Interrupciones de Controlador dsPIC33F

Las interrupciones son eventos en una secuencia de ejecución de instrucciones que provocan una interrupción del flujo normal del programa para acceder a una subrutina, ejecutarse y después volver a la misma posición en la que se estaba. Esto se usa para que el programa pueda atender o actuar antes posibles entradas externas o internas. Se debe configurar los registros propios de las interrupciones para habilitar aquellas que fuente de interrupción de interés, ya que el dsPIC33F dispone de hasta 118 fuentes de interrupción distintas puede ser por periféricos externos, conversión ADC, comunicación transmisión o recepción de USART, módulo de captura, módulo QEI, desbordamiento de timers entre otros más. Cuando se produce un evento que dispara una de estas interrupciones, el Program Counter (PC), salta hacía una posición de un vector de memoria donde cada interrupción tiene su posición. Este vector se conoce como IVT (interrupt vector table) y se muestra a continuación en la figura 2-6.

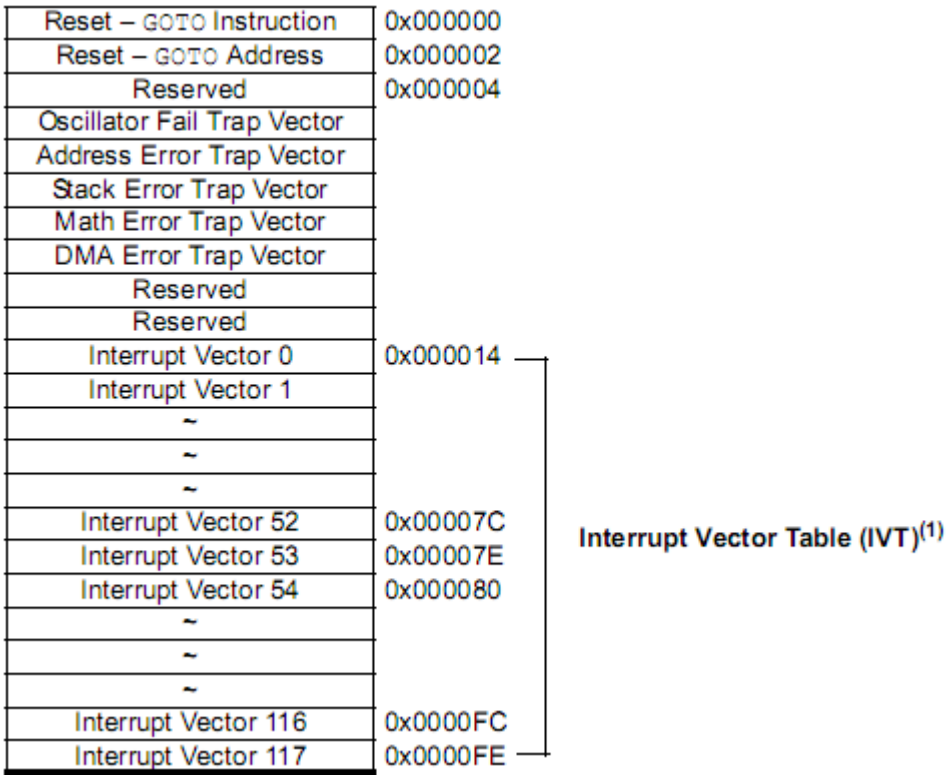


FIGURA 2-6. Vector de interrupciones, tomado de [6].

Para la configuración por parte del usuario de las interrupciones se manipulan principalmente los registros **IFSx**, **IECx** e **IPCx**.

Los registros **IFSx** contienen todas las banderas de estado de cada una de las solicitudes de interrupciones, "x" indica el número de registro. Cada fuente de interrupción tiene un bit de estado, que es fijado por los respectivos periféricos o por señales externas y se limpia por el software es decir el programa en ejecución.

Los registros **IECx** es donde se habilitan las fuentes de interrupción, "x" indica el número de registro. Estos bits de control se utilizan para activar de forma individual las interrupciones de los periféricos o por señales externas.

Los registros **IPCx** son los que controlan la prioridad de cada una de las interrupciones que fueron habilitadas asignándose niveles entre 1 a 8 siendo esta ultima la de más alta prioridad, esto con la finalidad que si existe el caso que se activen al mismo tiempo dos o más interrupciones será atendida la que tenga asignada la mayor prioridad.

2.1.7. Interrupción por CN del dsPIC33F

Dado que el dsPIC33FJ128MC706A no posee dos módulos QEI para la lectura del encoder incremental, se utiliza esta fuente de interrupción para la implementación de lectura de uno de los encoders. Su nombre es por **CN** (Change Notification por sus siglas en ingles), básicamente la familia dsPIC33F tienen la capacidad de generar peticiones de interrupción al procesador en respuesta a un cambio de estado en los pines de entrada seleccionados hasta 24 pines de entrada se puede seleccionar para generar interrupciones **CN**. En la figura 2-7 está el diagrama de bloques que ilustra el funcionamiento de este tipo de pines del dsPIC33F, para utilizar este recurso se debe seleccionar el pin o pines que se quieren para generar una interrupción por cambio de estado del pin ya sea un flanco de subida o bajada, cualquiera de los dos eventos realizará una petición de interrupción, los registros que contienen los bits de control son: **CNEN1** y **CNEN2** y estos a su vez tienen los bits **CNxIE** donde ‘x’ denota el número del pin de entrada como **CN**. Además se deben habilitar la respectiva interrupción en el registro **IECx,** y el evento activara el respectivo bit de la bandera de estado contenido en el registro **IFSx** que deberá ser “seteada” a cero dentro de la subrutina de interrupción por pin **CN**.

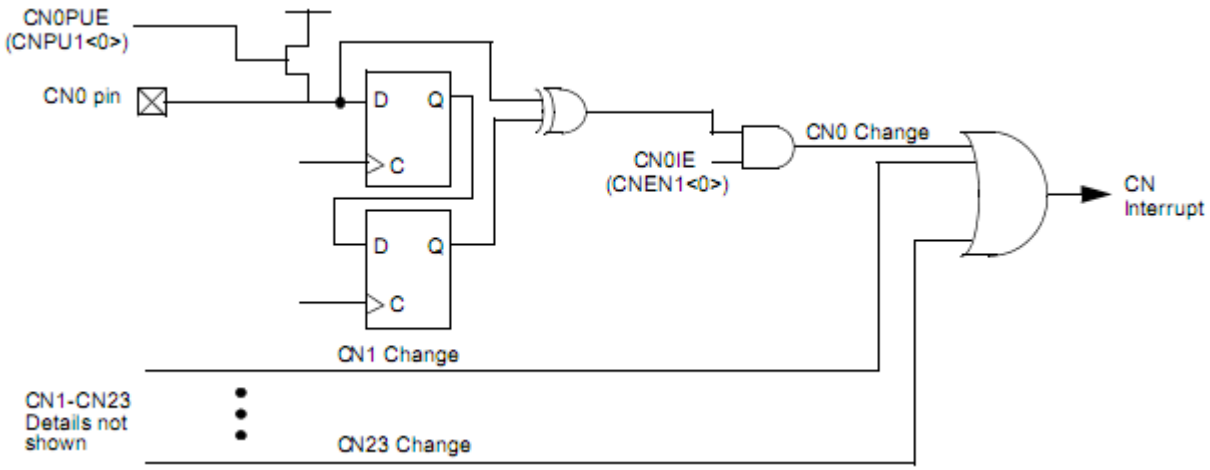


FIGURA 2-7. Diagrama de bloques para las entradas con CN, tomado de [6].

2.1.8. Módulo TIMER1 del dsPIC33F

Para la implementación del algoritmo PID se debe tener claro que se realiza una ejecución en tiempo discreto donde se incluye un tiempo de muestreo, que para este caso se desarrollo utilizando el **TIMER1,** el cual está disponible como un módulo dentro de las prestaciones del dsPIC33FJ128MC706A. Entonces el fabricante Microchip en [6] lo define como un módulo temporizador de 16 bits, que puede servir como el contador de tiempo para el reloj de tiempo real RTC (real time clock por sus siglas en ingles) o funcionar como un intervalo de temporizador / contador de funcionamiento libre. Básicamente el Timer1 puede funcionar en tres modos: temporizador de 16-bits, contador síncrono 16-bits o contador asíncrono 16-bits.

Para garantizar la exactitud en la temporización se realizó con implementación de interrupciones ya que este módulo nos proporcionara el tiempo periódico para ejecutar el algoritmo de control PID, seguidamente se configura el registro de control que es:

T1CON: en este registro de control se configura el TIMER1, y los bits manipulados nos permiten Iniciar el **TIMER1**, asignar la prescala entre los valores: 1, 8, 64 o 256, seleccionar el reloj como interno o externo. Esto se puede ver en la figura 2-7 donde adicional se debe setear el **TMR1** (registro temporizador de conteo) con el valor correspondiente al tiempo de conteo deseado.

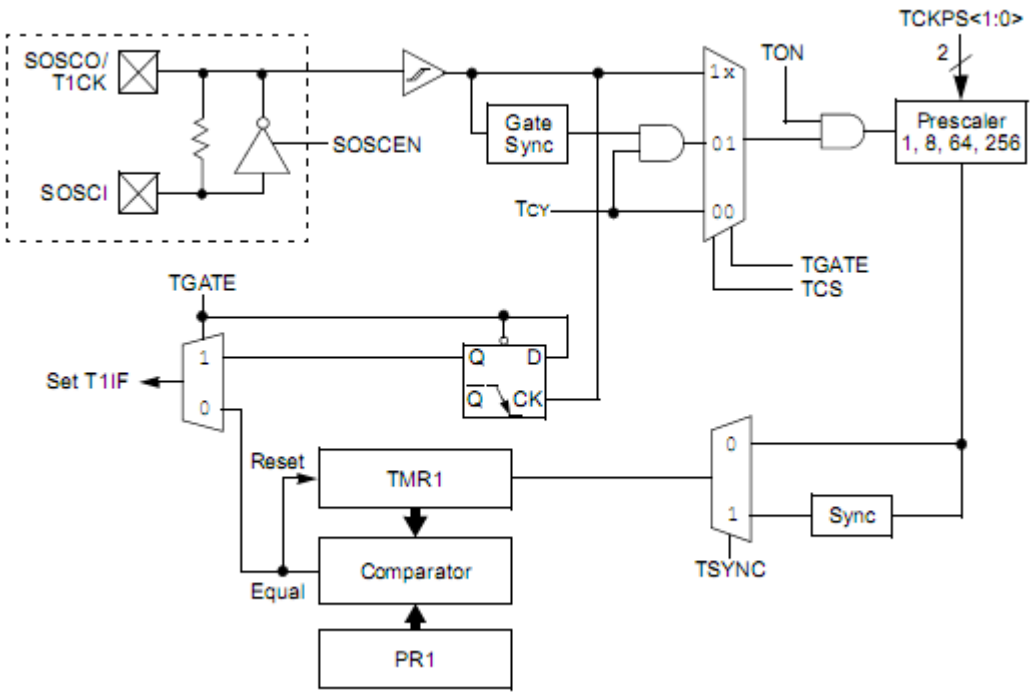


FIGURA 2-8. Diagrama de bloques del módulo TIMER1, tomado de [6].

El funcionamiento del módulo **TIMER1** genera la bandera de estado **T1IF** indicando que se cumplió el tiempo al cual fue configurado y en la figura 2-8 se observa que se comparan los valores del **TMR1** y **PR1** al ser iguales se genera este evento, en consecuencia para calcular el valor del tiempo de duración del timer1 se debe hallar el tiempo de un ciclo que utiliza el dsPIC33F para ejecutar una instrucción con la ecuación $(2 \cdot 3)$, donde F_{CY} es la frecuencia del ciclo de reloj. De la ecuación $(2 \cdot 4)$ se obtiene el tiempo cumplido de la temporización del **TIMER1** y que genera el evento en la bandera del registro **T1IF=1**.

$$tiempo\_ciclo = \frac{1}{F_{CY}/4} \qquad (2 \cdot 3)$$

$$T\_TIMER1 = tiempo\_ciclo \times Preescala \times TMR1 \qquad (2 \cdot 4)$$

2.2. ENTORNO MPLAB®X IDE Y COMPILADOR C30

El uso de herramientas de software conlleva al estudio de los entornos de desarrollo que en este caso el fabricante Microchip proporciona con el nombre de MPLAB® X IDE, por lo que se debe conocer en el detalle necesario el entorno grafico así como las capacidades funcionales que nos brinda el uso de esta herramienta para el desarrollo del trabajo de grado. Es entonces cuando se entra a ver la guía de usuario [8] donde se describe el MPLAB® X IDE (ver figura 2-9) como un programa de software que se ejecuta en un PC (Windows ®, Mac OS ®, Linux ®) para desarrollar aplicaciones para microcontroladores Microchip y controladores de señales digitales (dsPIC®).

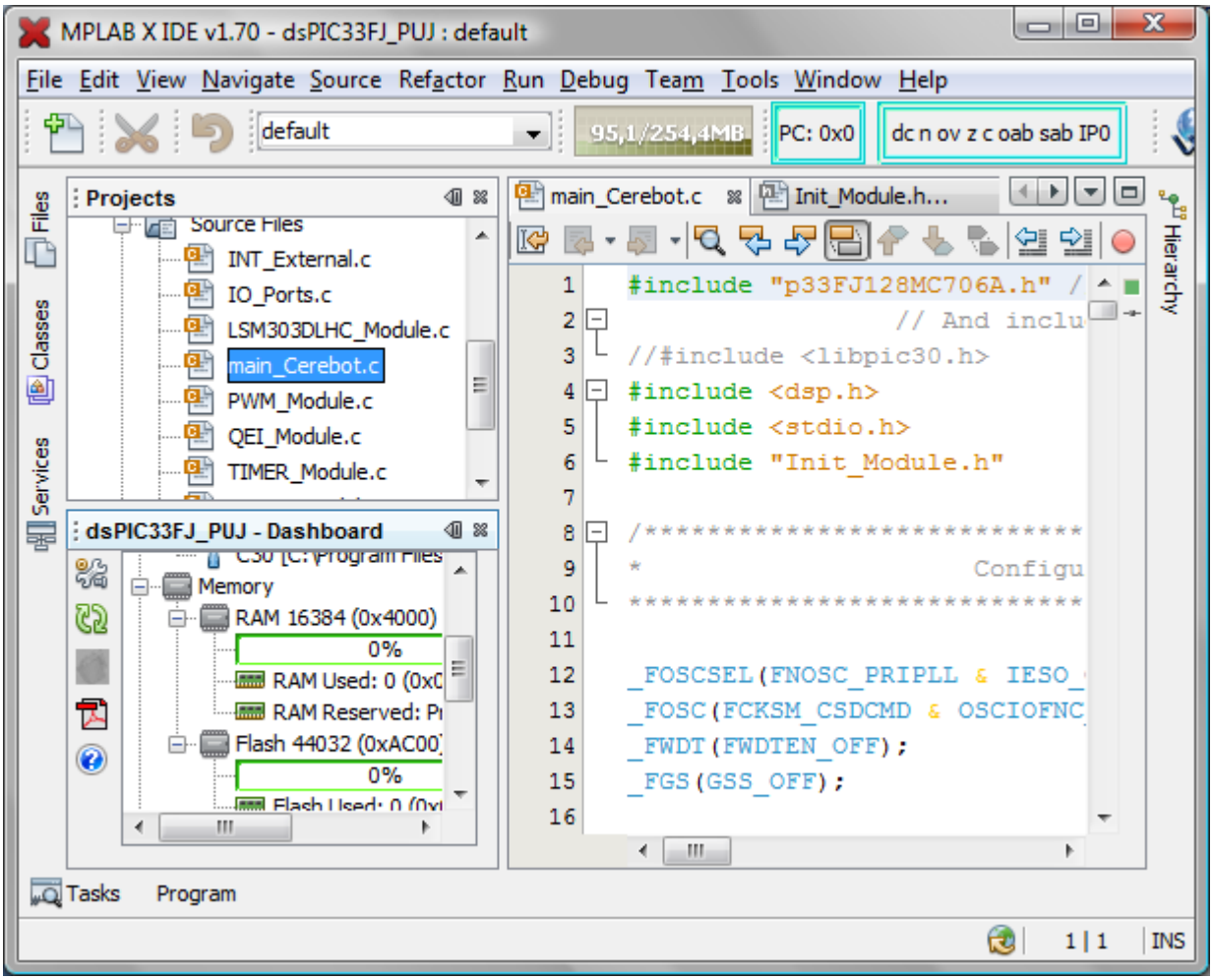


FIGURA 2-9. Diagrama de bloques del módulo TIMER1, fuente el autor.

Se llama un entorno de desarrollo integrado o IDE, ya que proporciona un único "ambiente" integrado para desarrollar código para microcontroladores embebidos. Es acá donde volvemos a consultar las definiciones de [8] donde se hace referencia a las diferencias de un controlador embebido y computador personal, se menciona que la principal diferencia entre ambos es que el controlador embebido está dedicado a una tarea o conjunto de tareas. Un computador personal está diseñado para funcionar con muchos tipos diferentes de programas y conectarse a diferentes dispositivos externos. Un controlador embebido tiene un solo programa y como resultado, se puede hacer a bajo costo al incluir sólo la suficiente potencia de computación y el hardware para realizar esa tarea dedicada. Un ordenador personal tiene una unidad central de proceso generalizado relativamente costosa (CPU) con muchos otros dispositivos externos (memoria, unidades de disco, controladores de video, circuitos de interfaz de red, etc). Un sistema embebido tiene una unidad de microcontrolador de bajo costo (MCU), con muchos circuitos periféricos en el mismo chip y con relativamente pocos dispositivos externos.
La implementación de estos sistemas embebidos es posible gracias a las herramientas de desarrollo proporcionadas por los fabricantes de estas tecnologías como lo es MPLAB® X IDE, que nos da la capacidad de programar los microcontroladores de las familias de 8bits, 16bits y 32bits que fabrica Microchip.

Dentro de las ventajas de este software encontramos las diferentes herramientas de hardware integrables con está para programar los microcontroladores de todas las familias disponibles de Microchip y en las que mencionamos los programadores ICD3, PICkit3, PICkit2, Real ICE, PM3 para depuración en circuito y emuladores, además se puede conectar con tarjetas de desarrollo de Microchip denominadas starter kits (ver figura 2-10) son compatibles para programación directamente desde el software MPLAB®X IDE. Para el desarrollo del proyecto fue utilizada la tarjeta Cerebot MC7 que cuenta con una licencia embebida para la conexión directa al MPLAB®X IDE como se observa en la figura 2-10. Otra de las ventajas de esta herramienta es la posibilidad de utilizar diferentes compiladores ASM30 (compilador de lenguaje ensamblador), C30 (compilador de lenguaje ANSI C) también la integración con otras marcas como el compilador de CCS C de la empresa Custom Computer Services Inc.

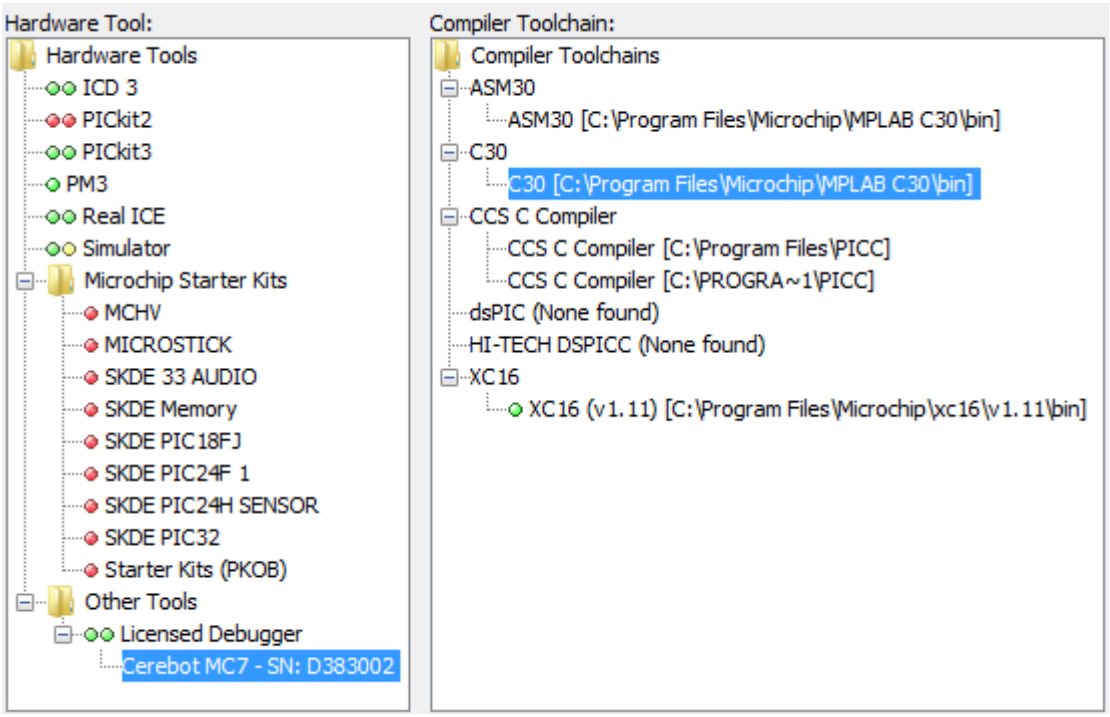


FIGURA 2-10. Diagrama de bloques del módulo TIMER1, fuente el autor.

Ahora bien, ya conociendo el entorno de desarrollo este no funciona sin un compilador que será el encargado de generar el archivo ejecutable el cual será descargado al dsPIC33FJ128MC706A a partir de los códigos escritos en lenguaje C, en el desarrollo de este trabajo de grado se utilizo el MPLAB® C30, el cual se instala bajo plataforma Windows® para ser utilizado como compilador principal, el principio básico de funcionamiento de este es explicado en la guía de usuario [9] donde dice que el MPLAB® C30 compila archivos de código fuente C, produciendo archivos en lenguaje ensamblador. Estos archivos generados por compilador se ensamblan y se vinculan con otros archivos objeto y bibliotecas para producir el programa de aplicación final en COFF ejecutable o formato de archivo ELF. El archivo COFF o ELF se puede cargar en el MPLAB IDE, donde se puede probar y depurar, con la utilidad de conversión se puede utilizar para convertir el archivo COFF o ELF para Intel ® formato hexadecimal, apto para la carga en el simulador de línea de comandos o un programador de dispositivos. Vea la Figura 2-11 para una visión general del flujo de datos de desarrollo de software.

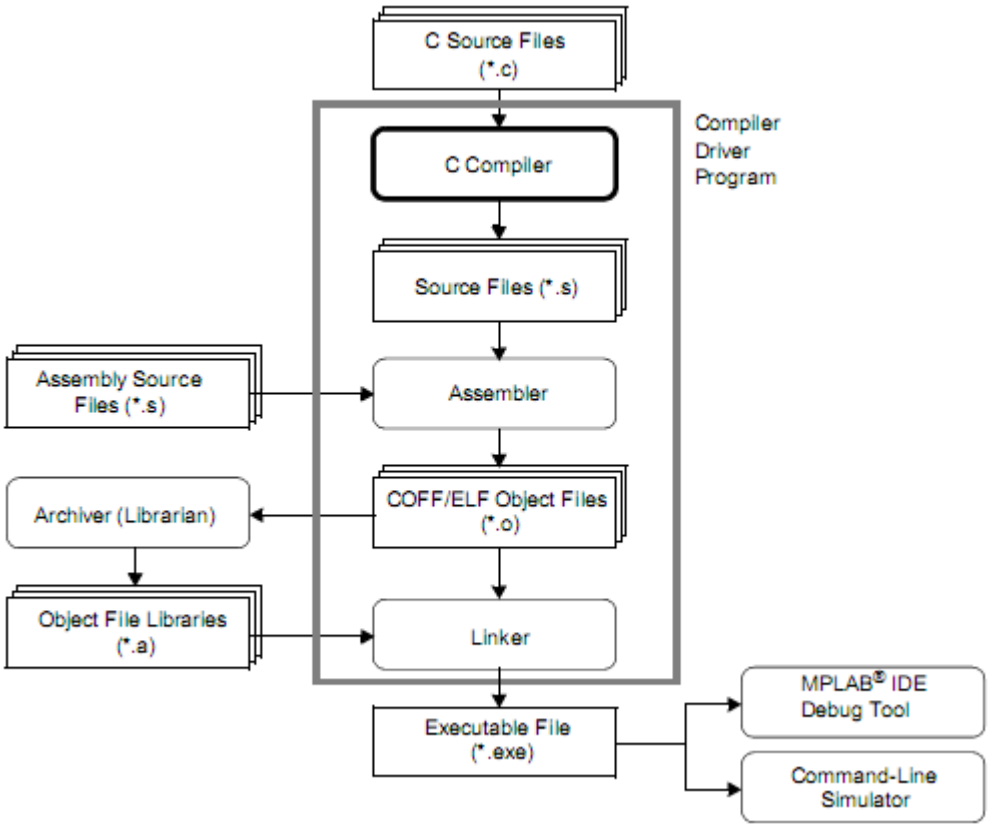


FIGURA 2-11. Flujo de datos de software en herramientas de desarrollo, tomado de [9].

2.3. TARJETA DE CONTROL CEREBOT MC7™

Para la implementación de los algoritmos de control se utilizó la tarjeta de desarrollo Cerebot MC7™ como ya fue mencionado antes, y cuyo cerebro es un microcontrolador de 16 bits de Microchip dsPIC33FJ128MC706A y del que ya se describieron sus principales características anteriormente. Esta tarjeta fue seleccionada por ser especializada para el control avanzado de motores de corriente directa ya que incluye 4 medios puentes H con MOSFET de potencia hasta 30V a 8A para cada uno a la salida de los pines PWM del dsPIC33F y facilidad de conexión de periféricos del motor y sus encoders. Dentro de las características más importantes esta la licencia embebida para su programación directamente desde el MPLAB®X IDE. En este apartado se mostrará en detalle los diagramas esquemáticos de medio puente H para explicar la principio de funcionamiento y también se explicará la interconexión con el módulo Pmod™ RS232X para la comunicación serial vía

protocolo RS232 entre tarjeta y computadora personal del módulo USART1 del dsPIC33F, también la comunicación del módulo USART2 con la tarjeta Seeeduino.

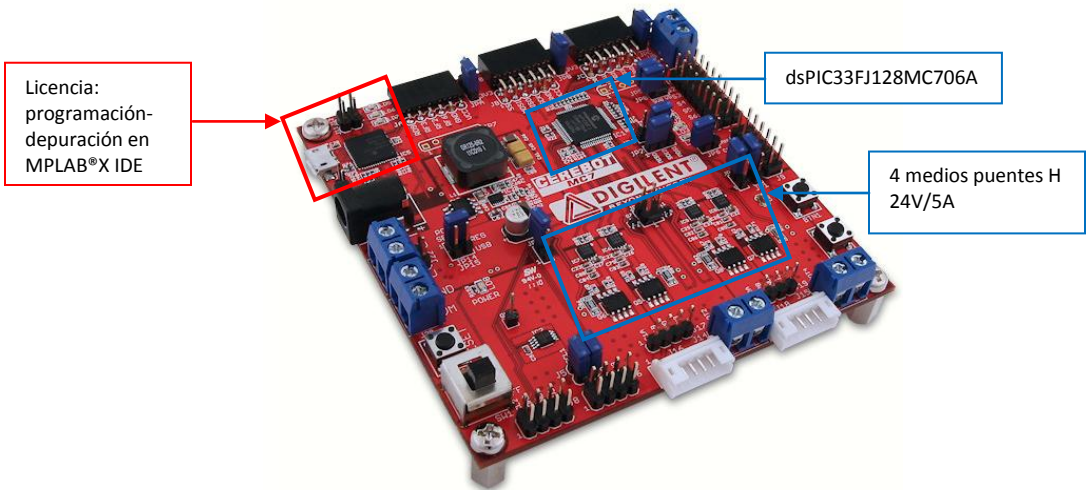


FIGURA 2-12. Tarjeta de control de motores Cerebot MC7™, tomado de [2].

También mencionamos la capacidad de procesamiento de hasta 40MIPS (millones de instrucciones por segundo), que la hace ideal para aplicaciones de control de motores en tiempo real.

2.3.1. Control de motores por MOSFET

La Cerebot MC7™ para el control de motores utiliza MOSFETs de potencia de canal N referencia NTMD4820N fabricados por On Semiconductor en package de montaje superficial que contiene 2 transistores y estos son controlados por el driver MCP14700MF de Microchip conformando medio puente. El MCP14700MF es un controlador MOSFET síncrono de alta velocidad diseñado para conducir de manera óptima un MOSFET de canal N de lado alto y del lado bajo, tiene dos entradas PWM para permitir el control independiente de los MOSFET de canal N externos, en la figura 2-13 se observa el diagrama esquemático del circuito implementado en la Cerebot MC7™, los pines PWM1H-PWM1L están conectados directamente al dsPIC33FJ128MC706A que son salidas del módulo MCPWM configuradas en modo complementario para controlar medio puente.

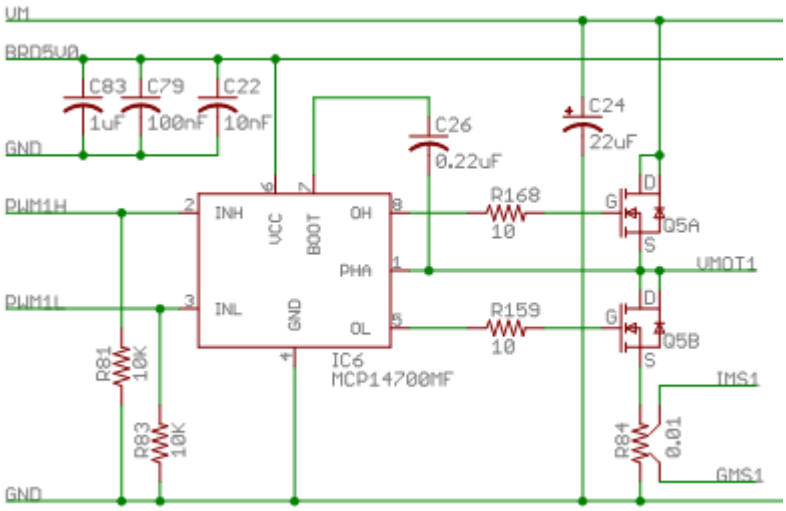


FIGURA 2-13. Circuito esquemático medio puente H, tomado de ANEXO A.

Las conexiones de los medios puentes entre las señales de salida y el módulo de control de MCPWM del dsPIC33FJ128MC706A están configuradas como se muestra en la Tabla 2-2, entonces realizando las conexiones al motor 1 las salidas VMOT1-VMOT2 del puente H #1 y el motor 2 las salidas VMOT3-VMOT4 al puente H #2 se tiene la posibilidad de realizar control independiente de cada motor con los MOSFET.

Salida MOSFETs	Driver MCP14700	Puente H
VMOT1	PWM1H-PWM1L	Puente H, # 1
VMOT2	PWM2H-PWM2L	
VMOT3	PWM3H-PWM3L	Puente H, # 2
VMOT4	PWM4H-PWM4L	

Tabla 2-2. Conexión puentes H, fuente el autor.

El principio de funcionamiento del puente H #1 se muestra en la figura 2-14, para un sentido de giro se activan los MOSFET encerrados en el círculo rojo, para el caso de la imagen izquierda los drivers1-2 controlados por el módulo MCPWM del dsPIC33F genera la señal PWM en el pin PWMH1 y configurado en modo complementario con PWML1 la señal de fase del driver en VMOT1(ver figura 2-13) es la del PWM, coloca a 1 lógico la señal PWML2 y las demás en 0 con esto se logra que la corriente circule del positivo VM a GND representada por la flecha azul. Entonces para invertir el sentido de giro que es el caso de la imagen parte derecha igualmente se activan los

MOSFET del círculo rojo para que circule la corriente desde la alimentación de voltaje VM a GND invirtiendo así el giro.

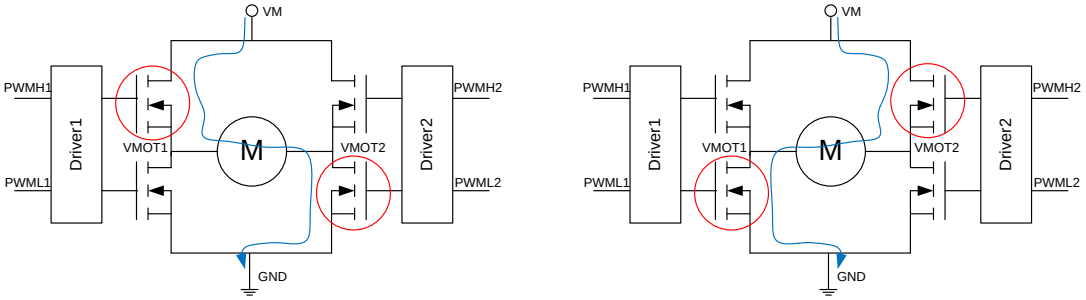


FIGURA 2-14. Puente H con MOSFET, fuente autor.

2.3.2. Comunicación serial RS232

La Cerebot MC7™ posee 2 conexiones para los módulos USART1-USART2 del dsPIC33FJ128MC706A uno es utilizado para conectar la tarjeta con la interfaz gráfica GUI y la otra es para comunicarse con la Seeediuno.

La interconexión del módulo USART1 que se hace entre la tarjeta y el computador, se utiliza el protocolo RS232, por lo que para adaptar los niveles apropiados entre el dsPIC33F de 3.3vdc y el computador por puerto USB de 5vdc se utiliza del lado de la tarjeta el Pmod® RS232X fabricado por Digilent Inc. mostrado en el figura 2-15, y de la parte del computador un conversor USB-RS232 (ver figura 2-16).

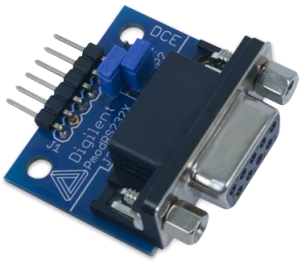


FIGURA 2-15. PmodRS232X™, tomado de [10].

FIGURA 2-16. Conversor USB-RS232, fuente el autor.

La interconexión entre la Cerebot MC7™ y la Seeeduino se hace directamente por cable en la configuración conocida como cable cruzado que comunica el módulo USART del dsPIC33FJ128MC706A y el puerto serial del microcontrolador de la placa Seeeduino con los pines Tx-Rx como se muestra en la figura 2-17.

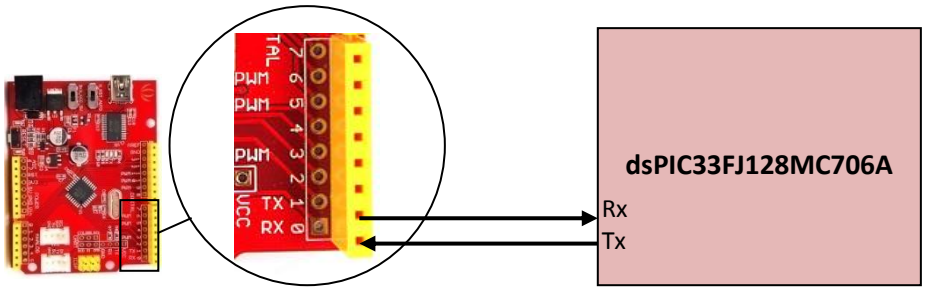


FIGURA 2-17. Conexión dsPIC33F y Seeeduino, fuente el autor.

2.4. SEEEDUINO Y ENTORNO ARDUINO

La tarjeta Seeeduino es basada en la Duemilanove de Arduino y en el trabajo de grado es utilizada para la lectura del sensor LSM303DLHC por medio del protocolo I^2C los valores con un filtro pasa-bajo simple y se transmite el dato sin ningún otro procesamiento al dsPIC33F por el protocolo serial RS232 como fue explicado e ilustrado en la figura 2-17. Las especificaciones más relevantes del procesador integrado en la Seeeduino y de la placa son resumidas en la tabla 2-3:

Microcontrolador	ATmega328p
Voltaje de Operación	5V ó 3.3v
Voltaje de entrada	7-12V
E/S Digitales (pines)	14(6 salidas PWM)
Entradas Analogas(pines)	6
Corriente DC por E/S(pin)	40mA
Corriente DC para 3.3v(pin)	50mA
Memoria Flash	32 KB
SRAM	2 KB
EEPROM	1 KB
Velocidad de reloj	16 MHz

Tabla 2-3. Especificaciones Seeeduino, fuente el autor.

Refiriéndonos a lo que es Arduino y tomando la definición de su página oficial [11] se describe como una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware libres y fáciles de usar. Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y controlar luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing).

El entorno de desarrollo para la programación de la plataforma Arduino se muestra en la figura 2-18, este lenguaje de programación se basa en C/C++ que son de alto nivel por lo que se facilita realizar la operación asignada de lectura del sensor LSM303DLHC. Básicamente desde esta interfaz de desarrollo se descarga el programa escrito en lenguaje Arduino al microcontrolador ATmega 328p y esto gracias a que dentro de este reside un bootloader especialmente diseñado para funcionar con esta herramienta software libre. Un bootloader es un programa desarrollado en lenguaje ensamblador para la optimización máxima del código, permanece en la parte alta del procesador y permite aprovechar la característica de borrar y grabar la memoria flash del procesador y solo ocupa 2 Bytes que estarán ocupados por el bootloader mientras no se realice un borrado con un programador externo.

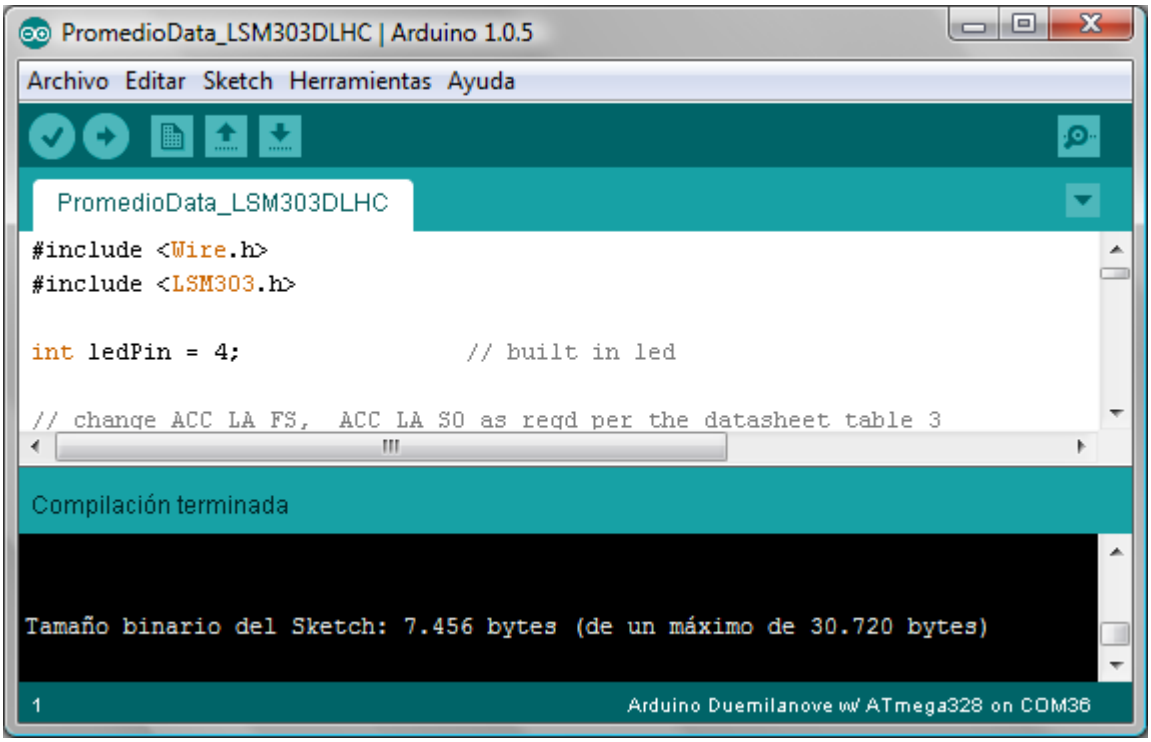


FIGURA 2-18. Entorno desarrollo Arduino, fuente el autor.

2.5. SENSORES

Un sensor, es un dispositivo diseñado para recibir información de una magnitud del exterior y transformarla en otra magnitud, normalmente eléctrica, que seamos capaces de cuantificar y manipular. Esto basado en diferentes principios de medición físicos que normalmente depende del tipo de magnitud a sensar dentro de las que se menciona posición lineal o angular, aceleración, temperatura, campo magnético, presión entre otros más. En este trabajo se mostraran los sensores utilizados para la medición específica de posición angular (encoder incremental), aceleración y campo magnético terrestre (LSM303DLHC).

2.5.1. Acelerómetro y compas magnético LSM303DLHC

El sensor utilizado es el LSM303DLHC fabricado por STMicroelectronics e integrado en una tarjeta compacta para su fácil uso por Pololu Robotics and Electronics tienda especializada en venta de componentes electrónicos para robótica móvil.

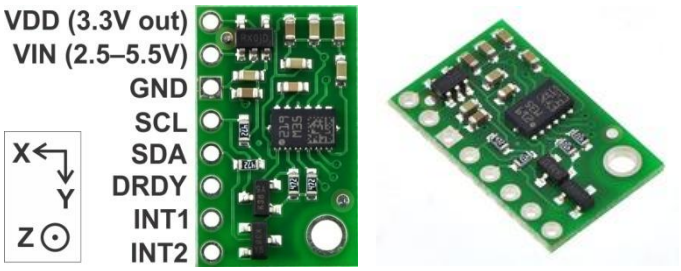


FIGURA 2-19. Sensor Pololu LSM303DLHC, tomado de [12].

De las características principales de este sensor se mencionan las seis lecturas independientes, cuya sensibilidad se puede ajustar en el rango de ±2 a ±16 **g** y ± 1.3 a ± 8.1 **gauss**, que están disponibles a través de una interfaz I²C. Esta placa portadora del sensor LSM303DLHC incluye un regulador de 3.3V de tensión y MOSFET de protección de nivel integrado que permite el funcionamiento de 2.5 a 5.5V de los pines SCL-SDA, que da la capacidad de conectar dos diferentes voltajes en el bus I^2C.

El diagrama de bloques del sensor LSM303DLHC se muestra en la figura 2-20, está compuesto por un bloque encargado del sensado de aceleración y campo magnético terrestre con un acelerómetro y un magnetómetro de 3 ejes cada uno con los principios de funcionamiento estudiados en el Capitulo 1 en la sección de sensores inerciales, contiene otro bloque encargado de darle tratamiento a las señales electrónicas con amplificadores para luego entregárselas al bloque del conversor analógico-digital, este a su vez se lo entrega al bloque de control lógico que es un pequeño procesador encargado de administrar los recursos que tiene el sensor como lo son manejo de interrupciones, filtros pasa-altos, sensor de temperatura, y configuraciones adicionales para el acceso de a los registros que contienen la información de cada uno de los ejes XYZ para el acelerómetro y magnetómetro, todo esto a través de una interfaz digital de comunicaciones I^2C y un conjunto de comandos explicados en la hoja de datos [13] del sensor.

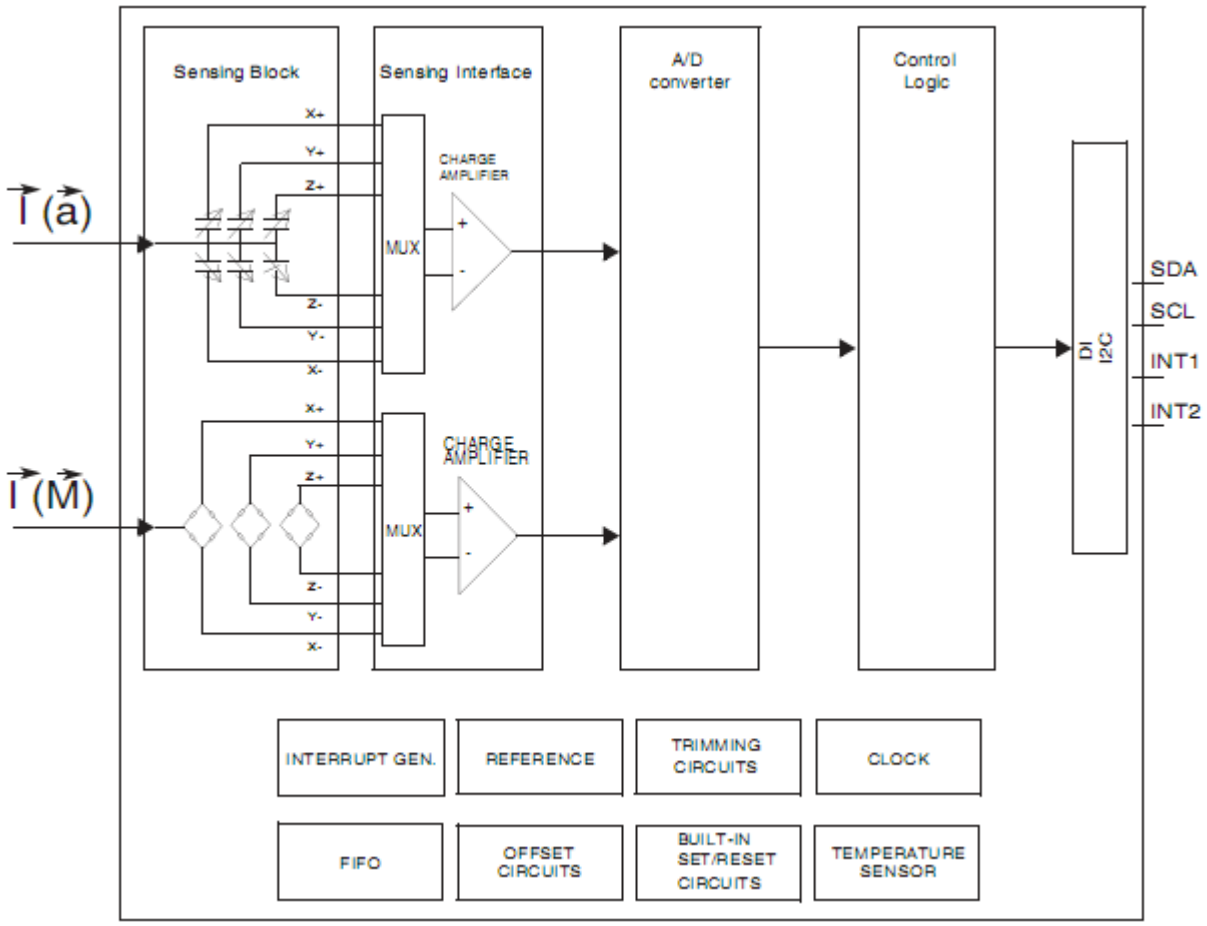


FIGURA 2-20. Diagrama de bloques LSM303DLHC, tomado de [13].

2.6. ENCODER INCREMENTAL Y MOTOREDUCTOR

La plataforma Altazimut utilizada en el presente trabajo de grado está construida con 2 motoreductores Matsushita que tienen acoplados mecánicamente su encoder incremental respectivamente. Por esta razón se describirán el funcionamiento de cada uno por aparte.

2.6.1. Motoreductor

Una definición de un motoreductor es simplemente un motor acoplado directamente a un reductor que es una caja de engranajes que reduce la velocidad pero aumenta el torque aplicado sobre el eje de salida del reductor. En la figura 2-21 se muestra el motoreductor utilizado en el trabajo de grado identificando las partes de interés que son eje reductor, caja reductora de engranajes, motor DC (corriente directa), y encoder incremental.

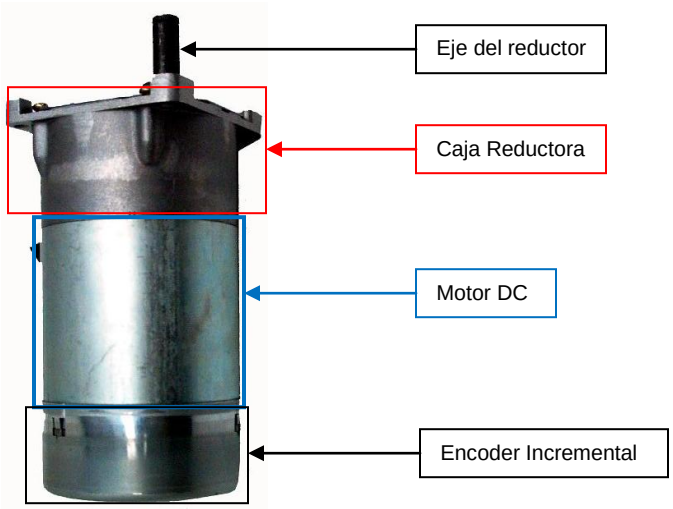


FIGURA 2-21. Motoreductor Matsushita, fuente autor.

El principio de funcionamiento de los motores DC descrito en [14], dice que un campo magnético es creado en el estator por un imán permanente, o por bobinas electromagnéticas, opuesto al campo del estator esta el campo de la armadura, el cual es generado por el flujo electromagnético cambiante que viene de las bobinas localizadas en el rotor. Los polos magnéticos del campo de la armadura intentaran alinearse con los polos magnéticos opuestos. La sección del rotor donde la electricidad entra a los embobinados del rotor se llama conmutador. La electricidad es conducida entre el rotor y el estator por las escobillas de cobre o los conductos de grafito, las cuales están en el contacto con los anillos que se encuentran sobre el estator. En la figura 2-22 se observa la configuración general de un motor DC. Es así como el motor gira hacia el punto de alineación de los polos, por ende, las escobillas saltan a través de un especio entre los anillos del estator.

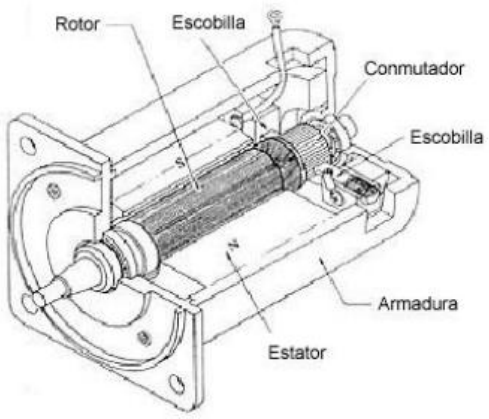


FIGURA 2-22. Configuración de motor DC, tomado de [14].

Las características más relevantes de este motor son: 230RPM, 8-22Vdc, consumo de corriente de 2A aproximadamente, la relación de reducción es 18.5:1 aproximadamente.

2.6.2. Encoder Incremental

El encoder(ver figura 2-21) acoplado a el motoreductor Matsushita es un encoder incremental óptico y su principio de funcionamiento es la generación de una señal cuadrada incremental de dos señales de los canales A-B desfasadas 90°(ver figura 2-13), y la construcción esta hecha por un disco dividido por ranuras que tienen ubicadas en un lado un led emisor que genera un rayo de luz que al otro lado del disco ranurado posee un fotoreceptor como se muestra en la figura 2-24, al girar este disco este fotoreceptor encenderá y apagara alternamente produciendo un tren de pulsos.

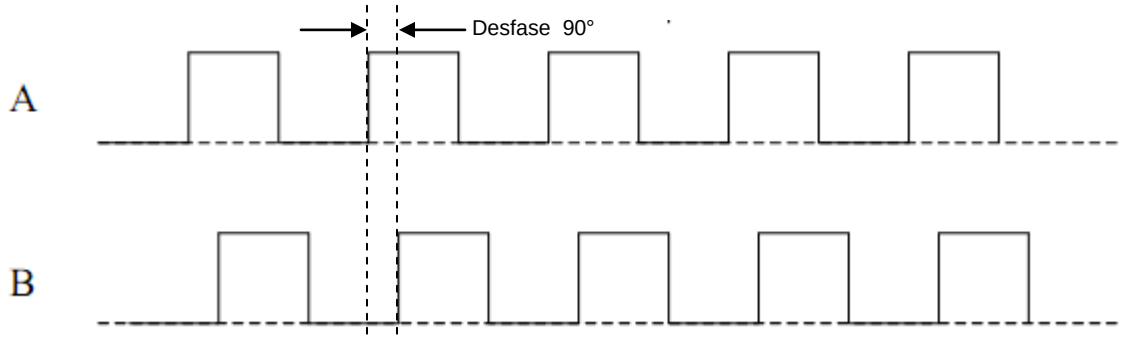


FIGURA 2-23. Señales en cuadratura incremental, fuente el autor.

Dado que tiene dos fotoreceptores desfasados genera un tren de pulsos en el canal A y B mostrado en la figura 2-23. Con esta señal se puede tener la posición angular, por canal el encoder óptico incremental del Matsushita genera 100 p.p.r. (pulsos por revolución) esto quiere decir que por 1 vuelta que gire el disco del encoder tienen 100 pulsos a la salida de los fotoreceptores.

Esto quiere decir que podríamos hallar la resolución que nos da este encoder con la siguiente ecuación:

$$Resolución(°) = \frac{360°}{p.p.r} \qquad (2 \cdot 5)$$

Aplicando la ecuación $(2 \cdot 5)$ se podría tener una resolución cada de 3.6°, en otras palabras a la salida de un canal del fotoreceptor un pulso positivo.

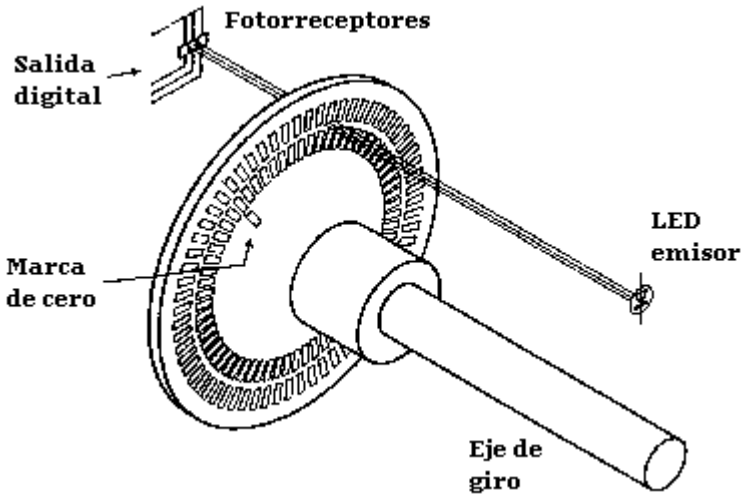


FIGURA 2-24. Disco encoder incremental, tomado de [15].

2.7. MATLAB® GUIDE

MATLAB® es una herramienta software para procesamiento matemático computacional desarrollado por MathWorks® y que incluye el entorno de desarrollo de interfaces graficas con la herramienta GUIDE que es un entorno de programación visual disponible en MATLAB® para realizar y ejecutar programas que necesiten ingreso continuo de datos. Tiene las características básicas de todos los programas visuales como Visual Basic o Visual C++. En esta oportunidad se utiliza el MATLAB® R2009a. Esta GUIDE funciona generando 2 archivos uno que contendrá el contenido visual extensión ***.fig*** y que contiene el código se estará ejecutándose para generar acciones o eventos dentro de la GUI desarrollada por el usuario con extensión ***.m*** propia del lenguaje de programación de MATLAB® ambos archivos con el mismo nombre. En la figura 2-25 se observa la GUI desarrollada para el trabajo de grado que incluye uso de componentes push botton, edit text, axes para visualizar las graficas, pop-up menú y panel para agrupar diferentes componentes.

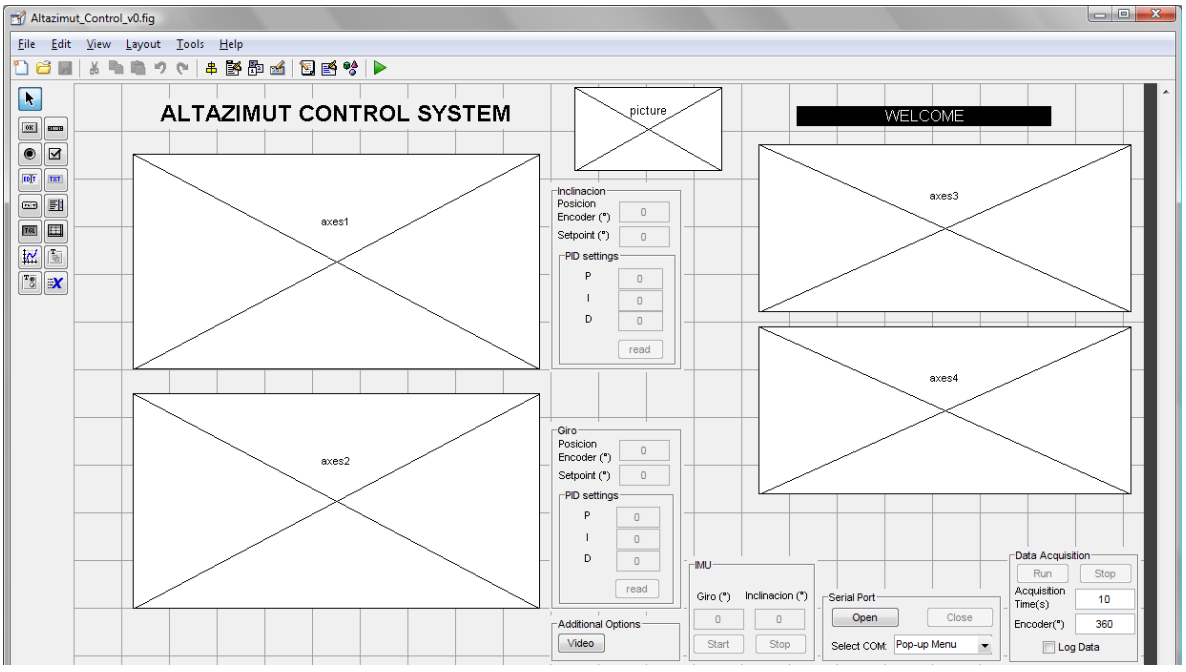


FIGURA 2-25. GUI en MATLAB con GUIDE, fuente el autor.

CAPITULO 3.

IDENTIFICACION Y MODELAMIENTO DE PLATAFORMA

Para el presente trabajo se utilizo básicamente la técnica de identificación newton-laplace basado en los parámetros del motor DC que compone la plataforma Altazimut, este es un modelo conocido caja blanca ya que se conocen los elementos internos que generan la salida a una entrada especifica que representara lo más real posible la planta modelada.

3.1. IDENTIFICACION Y OBTENCION DEL MODELO POR NEWTON-LAPLACE

El sistema a identificar está conformado por una plataforma con dos grados de libertad denominada Altazimut, los cuales ofrecen movimiento en un rango de 360° tanto en giro como en elevación, como se muestra en la figura 3-1 donde los motores mostrados generan estos dos movimientos mencionados.

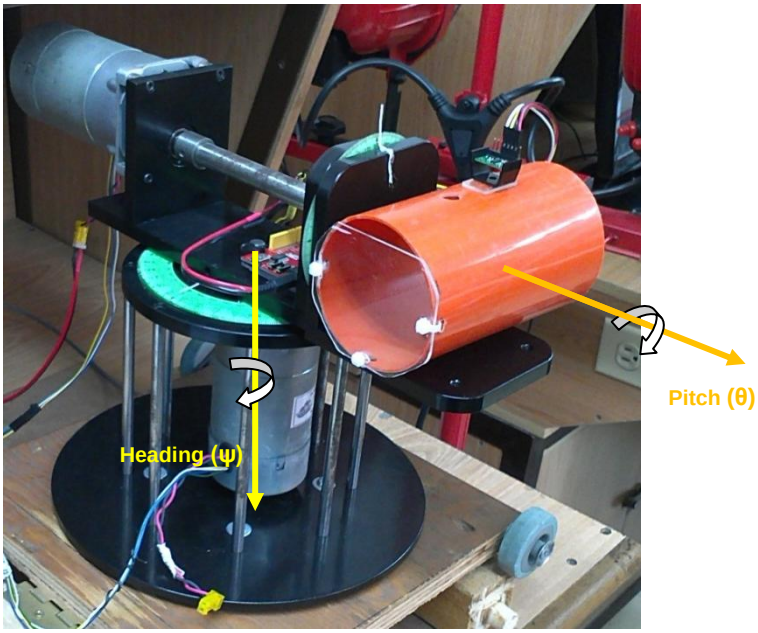


FIGURA 3-1. Plataforma Altazimut, fuente el autor.

3.1.1. Función de Transferencia Motor DC

Cada grado de libertad está gobernado por un motor-reductor eléctrico DC idéntico, controlado por armadura, el cual es caracterizado como el circuito mostrado en la figura 3-2 donde es incluida la masa de la plataforma en cada uno de los ejes de control para giro (ψ) y elevación (θ), que al realizar las mediciones correspondientes en cada uno de los motores esta masa es representada por la carga en el eje del motor-reductor como se ilustra en la figura 3-3.

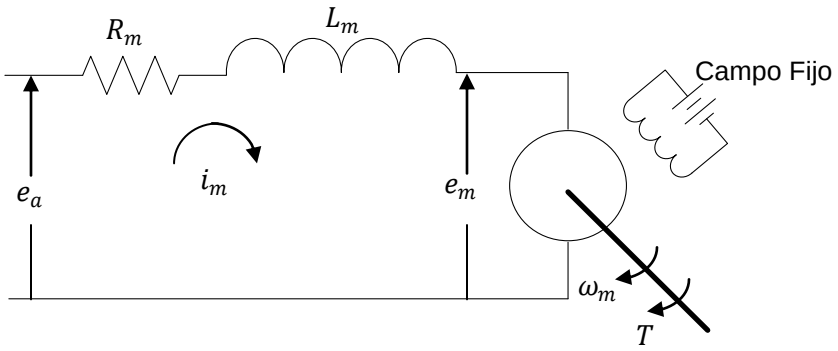


FIGURA 3-2. Diagrama del circuito de un motor DC, fuente el autor.

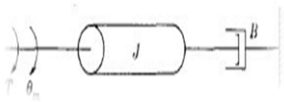


FIGURA 3-3. Inercia y fricción como carga de un motor, fuente el autor.

La siguiente tabla resume los parámetros del esquema del motor DC para entender las ecuaciones que relacionan la función de transferencia característica y los datos experimentales necesarios para obtener el modelo del motoreductor Matsushita.

J_m	Momento de inercia del motor ($Kg.m^2$)
B_m	Fricción viscosa del motor N-m/rad/s
R_a	Resistencia de armadura
L_a	Inductancia de Armadura
I_a	Corriente de armadura
V_a,e_a	Voltaje de armadura
V_b	Voltaje inducido(fem)
T_m	Torque desarrollado por motor (N-m)
θ	Desplazamiento angular (rad)
ω	Velocidad angular (rad/s)
K_b	Constante fem (V/rad/s)
$K_t=k_m$	Constante de torque(N-m/A)

Tabla 3-1. Parámetros de motor, fuente el autor.

La función de transferencia del motor DC controlador por inducido se describe en [16] y describe la corriente de armadura I_a como la variable de control. El campo en el estator se puede establecer mediante una bobina de excitación y corriente o un imán permanente.

$$I_a \longrightarrow \text{Variable de control}$$

Por definición en [16] utilizando imán permanente.

$$T_m(s) = K_T I_a(s) \qquad (3 \cdot 1)$$

$$v_a = R_a I_a + L_a \frac{dI_a}{dt} + v_b \qquad (3 \cdot 2)$$

La corriente del inducido se relaciona con el voltaje de entrada aplicado al inducido por:

$$v_a(s) = (R_a + L_a s) I_a(s) + v_b(s) \longrightarrow I_a(s) = \frac{v_a(s) - v_b(s)}{(R_a + L_a s)} \quad (3 \cdot 3)$$

Donde $v_b(s)$ es el voltaje de la fuerza contra electromotriz proporcional a la velocidad del motor. Por tanto se tiene:

$$v_b(s) = k_b \omega(s) \qquad (3 \cdot 4)$$

Reemplazando $(3 \cdot 4)$ en $(3 \cdot 3)$ despejamos y obtenemos la variable de control:

$$I_a(s) = \frac{v_a(s) - k_b \omega(s)}{(R_a + L_a s)} \qquad (3 \cdot 5)$$

Finalmente tenemos la **FUNCION DE TRANSFERENCIA DEL MOTOR**:

$$G(s) = \frac{\theta(s)}{v_a(s)} = \frac{k_m}{s[(R_a + L_a s)(Js + B) + k_b k_m]} \quad (3 \cdot 6)$$

Se debe hallar R_a , L_a , k_b , k_m , J y B de los datos experimentales para completar las constantes del modelo.

3.1.2. Constantes experimentales Modelo Motor DC

Para hallar $k_b = k_m$ se utiliza los datos en estado estacionario del motor $\frac{dI_a}{dt} = 0$
Se reemplaza $(3 \cdot 4)$ en $(3 \cdot 2)$ utilizando los datos en estado estacionario del motor y se obtiene:

$$v_a = R_a I_a + k_b \omega \qquad (3 \cdot 7)$$

Despejamos k_b $$k_b = \frac{v_a - R_a I_a}{\omega} \quad (3 \cdot 8)$$

Entonces aplicando los valores de voltaje (tabla 3-2) al motor y utilizando el montaje en simulink mostrado en la figura 3-4 compuesto por la derivada de la posición para obtener la velocidad, agregando un filtro digital pasabajo para suvizar la lectura final entregada por el encoder, lo que hace posible tomar la lectura en velocidad angular en las unidades rad/s.

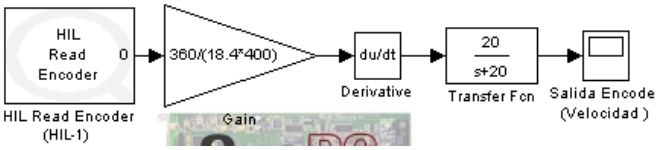


FIGURA 3-4. Montaje en Simulink lectura velocidad (rad/s), fuente el autor.

V_a(v)	I_a(A)	R_a(Ω)	ω(rad/s)	k_b(V/rad/s)
1	0,36	2,439	8,51	0,014
1,5	0,32	2,326	17,52	0,04313
2	0,32	2,247	28,17	0,04547
3	0,36	2,344	49,35	0,04369
			Promedio k_b =	0,03657

Tabla 3-2. Valor de k_b para velocidad angular del motor sin reductor, fuente el autor.

Se Calcula k_b para Motoreductor:

V_a(v)	I_a (A)	R_a(Ω)	ω(rad/s)	k_b(V/rad/s)
1	0,36	2,439	0,61	0,19993
1,5	0,32	2,326	1,25	0,75568
2	0,32	2,247	2,01	0,63729
3	0,36	2,344	3,53	0,610810
			Promedio k_b	0,55092

Tabla 3-3. Valor de k_b velocidad angular del motoreductor, fuente el autor.

Considerando la operación en estado estacionario del motor y el balance de potencia cuando se desprecia la resistencia del rotor. La potencia de entrada del rotor es $(k_b\omega)I_a$, y la potencia de entrada al eje es $T_m\omega$. En la condición de estado estacionario, la potencia de entrada es igual a la potencia entregada al eje, de modo que $(k_b\omega)I_a = T_m\omega$; como $T_m = K_m I_a$ reemplazando:

$$(k_b\omega)I_a = K_m I_a\omega \qquad (3\cdot 9),$$

Se encuentra que $k_b = k_m$

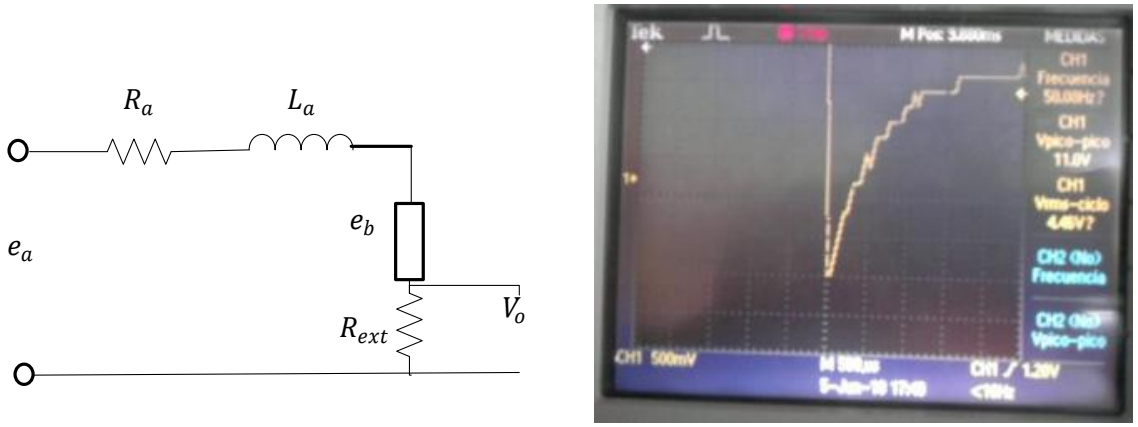
Se hallo R_a con el motor frenado (stall) y aplicando ley de ohm: $R_a = \frac{v_a}{I_a}$

Voltaje (V)	Corriente (A)	R_a (Ω)
1	0,41	2,4390244
1,5	0,645	2,3255814
2	0,89	2,247191
2,5	1,03	2,4271845
3	1,28	2,34375
	Promedio R_a=	2,360

Tabla 3-4. Valor resistencia de armadura R_a , fuente el autor.

Se continua realizando la caracterización del motor DC, se mide la inductancia L en el puente a 1kHz obteniendo R=3,078 Ω, L=1,8598mH.

Utilizando un osciloscopio se realiza medición del circuito mostrado en la figura 3-4(a), de $V_{Rext}\,(\text{voltaje en resistencia}) - \text{alimentando}\;\; e_a = V_a\,\text{de}\,8\text{Vdc} - 22\text{Vdc}$ con $f_{pwm} = 50Hz$ en e_a, $R_{ext} = 1{,}52ohm$. Con esta configuración, se aplica una onda cuadrada al circuito de armadura y de tal frecuencia que el rotor no gire; de esta forma el voltaje inducido es cero, debido a que no gira el motor ($e_b = 0$); esta característica genera entonces un comportamiento equivalente a un circuito RL, de tal forma que observando la forma de onda en la resistencia externa añadida es posible obtener la constante de tiempo que relación con la resitencia externa, la inductancia y resistencia de armadura.



a. Motor con resistencia b. Medida de R_{ext} en el osciloscopio

FIGURA 3-5. Motor con resistencia externa, fuente el autor.

La constante de tiempo de la gráfica 3-4(b), está definida como:

$$\tau_e = \frac{L_a}{R_a + R_{ext}} \qquad (3 \cdot 10)$$

		63%	
Voltaje(V_a)	V_{Rext} (V)	V_{Rext} (V)	τ_e (µs)
8	2,8	1,764	600
9	2,8	1,764	600
10	3,3	2,07	600
12	3,48	2,19	600
18	3,48	2,192	680
22	8,4	5,3	720
		Promedio	633,3333333

Tabla 3-5. Constante de tiempo τ_e , fuente el autor.

Finalmente se halla la inductancia de Armadura de la ecuación $(3 \cdot 10)$

$L_a = 2{,}45604\text{mH}$

Para hallar B_m se puede obtener despejando la siguiente ecuación Para V_a =2Vdc:

$$k_m = \frac{B\omega}{I_a} \quad \rightarrow \quad B = \frac{k_m I_a}{\omega} \qquad (3 \cdot 10)$$

$$B_m = \frac{0.03657(Nm/A) \times 0.32(A)}{28{,}17(rad/s)} = 415{,}42 \times 10^{-6}(Nm/rad/s)$$

Finalmente se halla el momento de inercia del motor J_m. Para V_a =2Vdc:

$$J_m = B_m \tau_m \qquad (3 \cdot 11)$$

V_a(v)	Rotor-reductor ω(°/seg)	63% ω(°/seg)	τ_m(seg)	Rotor-reductor ω(rad/seg)	63% ω(rad/seg)
2	70	25,9	0,081	1,225	0,45325
4	180	66,6	0,171	3,15	1,1655
6	300	111	0,265	5,25	1,9425
8	405	149,85	0,305	7,0875	2,622375
10	525	194,25	0,356	9,1875	3,399375
12	626	231,62	0,3908	10,955	4,05335
	promedio		0,261466667		

Tabla 3-6. Constante de tiempo τ_m , fuente el autor.

Entonces reemplazando en la ecuación $(3 \cdot 11)$ obtenemos:

$$J_m = 415{,}42 \times 10^{-6} (Nm/rad/s) \times 81ms = 33{,}65 \times 10^{-6} (kg \cdot m^2)$$

3.1.3. Modelo del Motor DC

Por tanto después de hallar las constantes de manera experimental la función de transferencia final para el motor, y reemplazando los valores $R_a = 2.36$, $L_a = 2.45604mH$, $k_b = 0.03657$, $k_m = 0.03657$, $J = 33.65 * 10^{-6}$, $B = 415.42 * 10^{-6}$ en la ecuación $(3 \cdot 6)$, la función de transferencia G(s) es equivalente al modelo de la planta a controlar en este caso con un controlador PID mostrado en el diagrama de bloques del sistema de control de la figura 3-5.

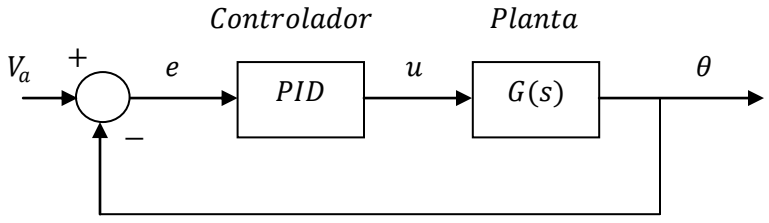


FIGURA 3-6. Estructura del sistema de control, fuente el autor.

$$G_{MOTOR}(s) = \frac{\theta(s)}{V_a(s)} = \frac{0.03657}{8.264 * 10^{-8} \cdot s^2 + 8.38 * 10^{-5} \cdot s + 0.002359}$$

De igual manera su respuesta paso en lazo abierto es:

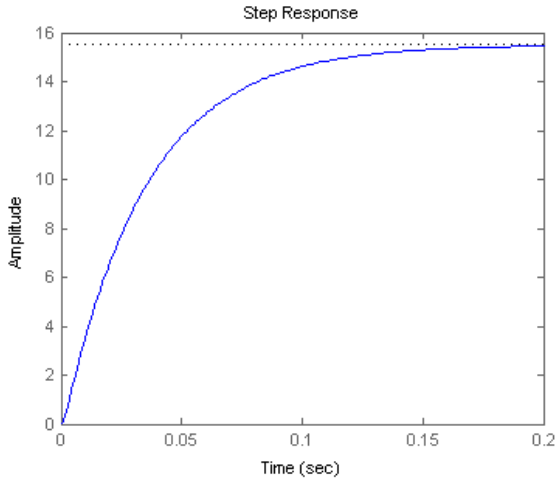


FIGURA 3-7. Respuesta paso en lazo abierto, fuente el autor.

Ahora para el motoreductor acoplado a la plataforma, la función de transferencia y respuesta paso en lazo abierto es $G_{MOTOREDUCTOR}\,(s)$ utilizando los parámetros para el motoreductor: $R_a = 2.36\,, L_a = 2.45604mH\,, k_b = 0.55092\,, k_m = 0.65092\,, J = 0.0229\,, B = 0.0877$:

$$G_{MOTOREDUCTOR}\,(s) = \frac{\theta(s)}{V_a(s)} = \frac{0.6509}{5.632 * 10^{-5} \cdot s^2 + 0.05663 \cdot s + 0.5744} \qquad (3 \cdot 12)$$

La anterior expresión es la función de transferencia de la plataforma Altazimut para el movimiento de elevación (θ).

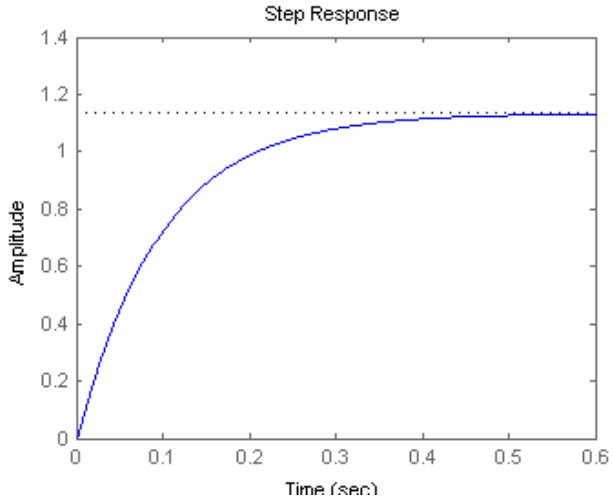


FIGURA 3-8. Respuesta paso lazo abierto Motor acoplado al reductor, fuente el autor.

Este procedimiento se repite para el subsistema de giro, ya que en esencia se trata de un modelo de posición independiente para cada grado de libertad, y de igual manera el motor-reductor utilizado es de la misma marca y referencia.

Por lo que recalculando los valores se tiene la función de transferencia $(3 \cdot 13)$ para el motor-reductor para el angulo giro(ψ)

$$G_{MOTOREDUCTOR}(s) = \frac{\theta(s)}{V_a(s)} = \frac{0.8982}{11.89 * 10^{-5} \cdot s^2 + 0.1194 \cdot s + 1.069} \qquad (3 \cdot 13)$$

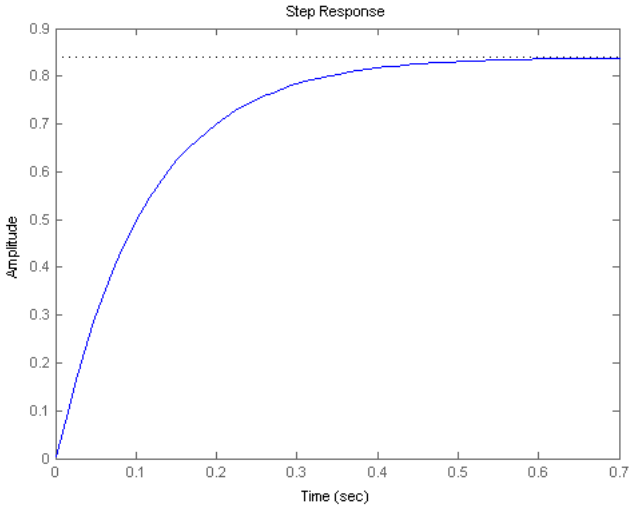


FIGURA 3-9. Respuesta paso lazo abierto Motor acoplado al reductor (giro), fuente el autor.

Más adelante se realiza validación del modelo $G_{MOTOREDUCTOR}(s)$ a una respuesta paso en lazo cerrado con un control PID que es programado en el dsPIC33FJ128MC706A de la Cerebot MC7^{TM}. Este modelo será utilizado para ambos motoreductores desacoplados no se planteo un modelo en espacio de estados acoplado para simplificar la aplicación embebida.

CAPITULO 4.

IMPLEMENTACION DE ALGORITMOS EMBEBIDOS

Este capítulo tratará de mostrar lo más claro posible los algoritmos que fueron implementados en las tarjetas Cerebot MC7™ y Seeeduino, explicando según sea el caso, la teoría de los cálculos matemáticos que debió realizar cada uno de los microcontroladores. Para el sensor LSM303DLHC se realizó el procedimiento de calibración para el compas magnético, y sus debidas transformaciones para calcular los ángulos utilizados en la realimentación de control de estabilización, utilizando el acelerómetro.

4.1. DIAGRAMA DE BLOQUES GENERAL

El funcionamiento general del proyecto está conformado por dos partes desarrolladas en dos diferentes microcontroladores como se observa en la figura 4-1. El Atmega328p, basado en Arduino, de la placa Seeeduino se encarga de la configuración y lectura de datos del sensor LSM303DLHC que tiene el acelerómetro y compas magnético, esto por la interfaz I^2C, adicional a esto, es encargado de realizar los cálculos básico de un filtro pasabajos muy simple que será explicado en detalle más adelante.

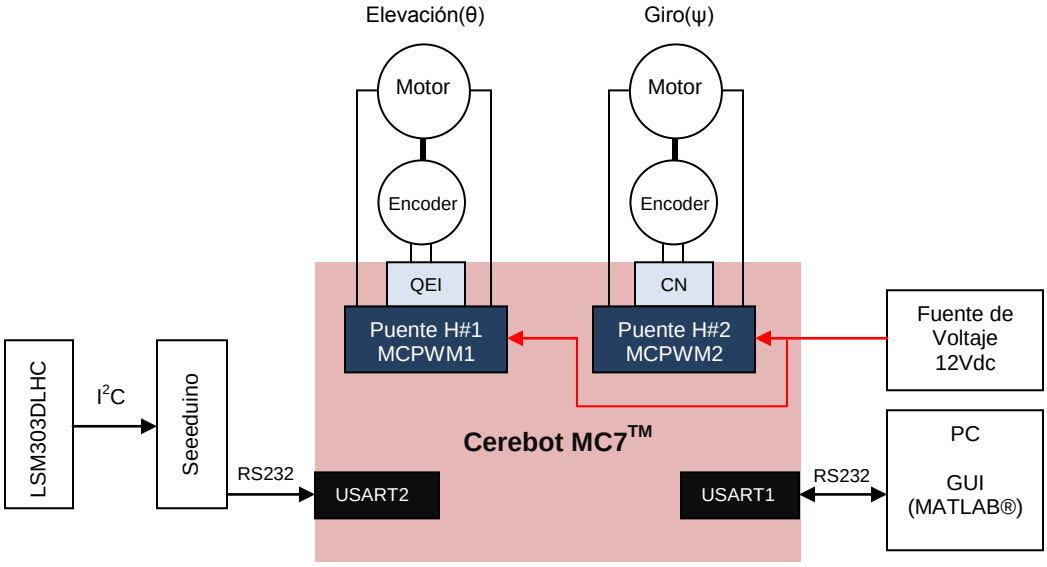


FIGURA 4-1. Diagrama de bloques general Plataforma Altazimut, fuente el autor.

La razón principal de utilizar la tarjeta Seeeduino básicamente es la distancia mayor a 50cm, a la que se ubica la Cerebot MC7^{TM} del sensor LSM303DLHC, y como se utiliza una interfaz de comunicación I^{2}C las distancias soportadas son bastante pequeñas, otra razón por la que se utilizo esta tarjeta intermedia es por la disponibilidad de la librería Arduino proporcionada por el fabricante Pololu de la tarjeta compacta, para facilitar el proceso de configuración y calibración del compas magnético.

Por otro lado la Cerebot MC7^{TM} recibe por intermedio de comunicación serial RS232 del módulo USART2 del dsPIC33FJ128MC706A los datos en bruto o raw del sensor LSM303DLHC para luego calcular los ángulos referenciados a la plataforma Altazimut Giro (θ) y Elevación (ψ). También se desarrolla la configuración para el control de los motores que controlarán la posición de los ángulos de la plataforma y la ejecución del control de estabilización en el ángulo de elevación (ψ) basado en los datos del sensor LSM303DLHC, todo esto por intermedio de la GUI desarrollada en MATLAB®.

4.2. IMPLEMENTACION EN SEEEDUINO

La programación del microcontrolador fue hecha con la librería **LSM303.cpp** proporcionada por Pololu Robotics and Electronics en [12], que sirve para las configuraciones propias del sensor por la interfaz I^{2}C.

4.2.1. Configuración Acelerómetro del LSM303DLHC

Para la configuración se utiliza la interfaz I^{2}C y la hoja de datos [13] del LSM303DLHC donde se referencian los registros de control compuestos por 8 bits cada uno y de los cuales solamente se manipularan los **CTRL_REG1_A** y **CTRL_REG4_A.**

El registro **CTRL_REG1_A,** es usado para activar los 3 ejes de medición X-Y-Z del acelerómetro y donde es seleccionada la velocidad de datos de adquisición del acelerómetro desde 1 Hz a 5.376 kHz. En la tabla 4-1 se muestran los bits del registro.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
ODR3	ODR2	ODR1	ODR0	Lpen	Zen	Yen	Xen

Tabla 4-1. Registro de Control CTRL_REG1_A, tomado de [13].

A continuación se describe la función de cada bit.

ODR3-0: Selección de velocidad de muestreo de datos del acelerómetro según la tabla 4-2.

LPen: Habilita el modo consumo de alimentación. 0: modo normal, 1: modo bajo-consumo.

Zen: Habilita el eje Z. 0: deshabilita eje Z, 1: habilita el eje Z.

Yen: Habilita el eje Y. 0: deshabilita eje Y, 1: habilita el eje Y.

Xen: Habilita el eje X. 0: deshabilita eje X, 1: habilita el eje X.

ODR3	ODR2	ODR1	ODR0	Modo Selección de Consumo de Energía
0	0	0	0	Modo de alimentación apagado
0	0	0	1	Modo Normal / Bajo (1 Hz)
0	0	1	0	Modo Normal / Bajo (10 Hz)
0	0	1	1	Modo Normal / Bajo (25 Hz)
0	1	0	0	Modo Normal / Bajo (50 Hz)
0	1	0	1	Modo Normal / Bajo (100 Hz)
0	1	1	0	Modo Normal / Bajo (200 Hz)
0	1	1	1	Modo Normal / Bajo (400 Hz)
1	0	0	0	Modo Bajo-Consumo (1.620 kHz)
1	0	0	1	Modo Normal (1.344 kHz) / Bajo (5.376 kHz)

Tabla 4-2. Bits ODR3-0 Modo consumo de energía, tomado de [13].

La configuración inicial en Arduino del registro CTRL_REG1_A se carga con el valor hexadecimal **0x57**, que habilita los 3 ejes X-Y-Z, el modo de consumo de energía como normal y el ancho de banda en BW=100Hz.

El registro **CTRL_REG4_A,** es usado para configurar la actualización de datos continua o bloquearla hasta que es realizada lectura de datos de los registros LSB y MSB, también la selección de registro más significativo LSB o MSB para lectura de datos del acelerómetro, la selección de escala desde +/- 2g hasta +/- 16g, se habilita además el modo de alta resolución hasta de 12bits y finalmente se puede configurar la interface de comunicación SPI en modo 4 o 3 cables. En la tabla 4-3 se muestran los bits del registro.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
BDU	BLE	FS1	FS0	HR	0	0	SIM

Tabla 4-3. Registro de Control CTRL_REG4_A, tomado de [13].

A continuación se describe la función de cada bit.

BDU: Bloqueo actualización de datos. 0: actualización continua, 1: no registra actualización hasta la lectura de MSB y LSB.

BLE: selección de registro más significativo LSB o MSB para lectura de datos del acelerómetro, 0: LSB datos menos significativos, 1: MSB datos menos significativos.

FS1-FS0: Selección de escala total. 00: +/- 2g, 01: +/- 4g, 10: +/- 8g, 11: +/- 16g.

HR: Modo de salida de alta resolución, 0: deshabilita alta resolución, 1: habilita alta resolución.

SIM: Modo selección para interface serial SPI, 0: interface de 4-hilos, 1: interface de 3-hilos.

La configuración inicial en Arduino del registro CTRL_REG1_A se carga con el valor hexadecimal **0x08**, para habilitar alta resolución a 12bits y escala total a +/- 2g.

Los detalles de la configuración están en el código fuente utilizado (ver el ANEXO B). Además la descripción de los demás registros se pueden consultar en la hoja de datos [13].

4.2.2. Configuración Compas Magnético del LSM303DLHC

Al igual que el acelerómetro se utiliza la misma interfaz I^2C y la hoja de datos [13] del LSM303DLHC donde se referencian los registros de control compuestos por 8 bits cada uno y de los cuales solamente se manipularán los **CRA_REG_M, CRB_REG_M** y **MR_REG_M.**

El registro **CRA_REG_M,** es usado para activar el sensor de temperatura integrado y donde es seleccionada la velocidad de datos de adquisición del magnetómetro desde 0.75 Hz a 220 Hz. En la tabla 4-4 se muestran los bits del registro.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
TEMP_EN	0	0	DO2	DO1	DO0	0	0

Tabla 4-4. Registro de Control CRA_REG_M, tomado de [13].

Los bits que no utilizados deben mantenerse en 0 para el correcto funcionamiento del dispositivo.

A continuación se describe la función de cada bit.

TEMP_EN: Habilita el sensor de temperatura, 0: sensor deshabilitado, 1: sensor habilitado.

DO2-0: Selección de velocidad de muestreo de datos del magnetómetro según la tabla 4-5.

DO2	DO1	DO0	Minima velocidad de salida de datos (Hz)
0	0	0	0.75 Hz
0	0	1	1.5 Hz
0	1	0	3.0 Hz
0	1	1	7.5 Hz
1	0	0	15 Hz
1	0	1	30 Hz
1	1	0	75 Hz
1	1	1	220 Hz

Tabla 4-5. Bits DO2-0, tomado de [13].

La configuración inicial en Arduino del registro CRA_REG_M se carga con el valor hexadecimal **0x30**, que selecciona un ancho de banda en BW=75Hz.

El registro **CRB_REG_M,** es donde se configura las ganancias para los 3 ejes de medición X-Y-Z según el rango de medición del sensor para este caso el rango utilizado es +/- 1.3 Gauss ya que la intensidad del campo magnético terrestre está entre 0.5 a 0.6 Gauss según [13] que finalmente es el rango necesario para utilizar el sensor como brújula electrónica. La salida de los datos se encuentra en el rango digital de -2048 a 2047, que es una resolución de 12 bits.

El registro **MR_REG_M,** seleccionan el modo de operación del magnetómetro entre modo de conversión continua, modo de conversión simple y modo sleep. La configuración es hecha en modo de conversión continua cargando el valor hexadecimal **0x00**.

Los detalles de la configuración están en el código fuente utilizado (ver el ANEXO B). La descripción de los demás registros se pueden consultar en la hoja de datos [13].

4.2.3. Filtro pasabajo, datos del LSM303DLHC

Se puede entender intuitivamente el efecto paso bajo con la operación de la ecuación $(4 \cdot 1)$, ya que al promediar los datos se atenúan las variaciones bruscas de la señal, lo que origina un suavizado de la señal de entrada.

$$Y(n) = \frac{1}{n}\sum_{k=1}^{n} X(k) \qquad (4 \cdot 1)$$

Donde n es el número de muestras, esto se realizó con un buffer de lectura con la función **compass.read()** de la librería Arduino para el sensor LSM303DLHC, por la interfaz I^2C se consultan los datos para cada eje X-Y-Z del acelerómetro y magnetómetro. Cuando n=1 el valor actual se promedia con los datos anteriormente guardados y muestreados según el valor de n. Los detalles en el código fuente implementado (ver el ANEXO B).

4.2.4. Comunicación Serial con dsPIC33F

Los datos ya promediados de los sensores del LSM303DLHC son enviados sin ningún otro procesamiento al dsPIC33FJ128MC706A por intermedio de comunicación serial a una velocidad de 115200 Baudios a un periodo de 10ms que coincide con la lectura por la interfaz I^2C del LSM303DLHC con el comando **compass.read()** de la librería Arduino.

El envío de los datos se realiza utilizando el formato que le da la función **printf()**; nativa en el lenguaje C, y en Arduino como función **Serial.print()**; se formatean los datos como enteros se envía una trama de la siguiente manera:

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8	Byte9	Byte10	Byte11	Byte12	Byte13	Byte14
"A"	*Ax*	**","**	*Ay*	**","**	*Az*	**","**	**"M"**	*Mx*	**","**	*My*	**","**	*Mz*	**"}"**

Tabla 4-6. Trama de datos seriales a dsPIC33F, fuente el autor.

La cabecera de la trama es el carácter 'A' que indica al dsPIC33F que se recibirá primero los datos del Acelerómetro en el orden Ax-Ay-Az separados por el carácter '**,**' para separar cada dato. Luego el Byte 8 con el carácter 'M' indica que el siguiente

conjunto de datos que se recibe será el del Magnetómetro en el orden Mx-My-Mz separados también por el carácter '**,**' y como finalización de trama de datos se utiliza el carácter '**}**' el cual dejara a la espera del siguiente dato al dsPIC33F.

4.2.5. Diagrama de Flujo del Programa

El programa simplificado implementado en Seeeduino es mostrado en el diagrama de flujo mostrado en la figura 4-2, donde se inicia con los headers de las librerías utilizadas: la <Wire.h> para el uso de I^2C y la del sensor LSM303DLHC <LSM303.h>. La siguiente ejecución del polling es la definición de variables locales. El polling continua en la función void setup() donde son cargadas las configuraciones explicadas en los ítems 4.2.1 y 4.2.2; la función principal del programa es llamada en Arduino como void loop(), y este programa se incluye la lectura del sensor, la ejecución del filtro pasabajo de la ecuación 4-1 y posterior envió de los datos por el puerto serial del microcontrolador en una trama de datos (tabla 4-6) esto cada 10ms.

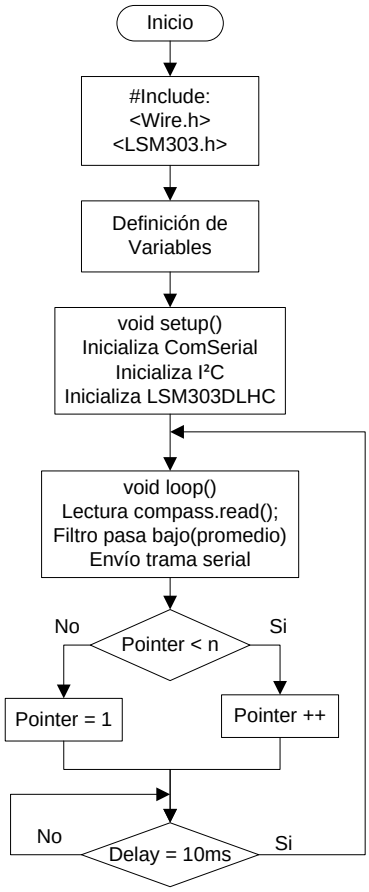


FIGURA 4-2. Diagrama de flujo programa Arduino, fuente el autor.

4.3. IMPLEMENTACION EN CEREBOT MC7™

El desarrollo de los algoritmos como se mencionó en el capítulo 2 en lenguaje C utilizando el concepto de programación modular [17] se generaron 8 subrutinas para facilitar la escritura del código implementado. La estructura del proyecto en MPLAB®X IDE se muestra en la figura 4-3 conformado por 2 archivos Header el <Init_Module.h> que contiene todas las definiciones, estructuras de datos y declaración de funciones creadas para utilizarse en las subrutinas y el <p33FJ128MC706A.h> que contiene todos las definiciones propias del dsPIC33FJ128MC706A proporcionada por el compilador C30.

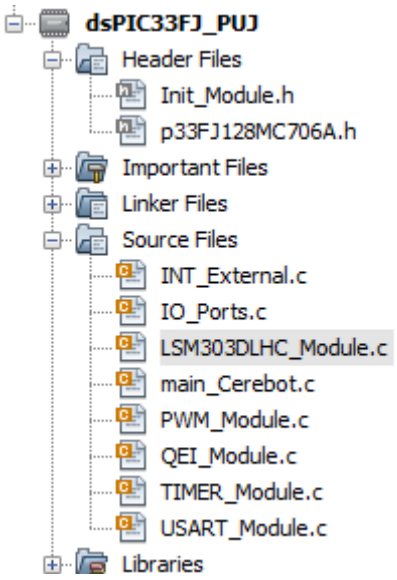


FIGURA 4-3. Estructura proyecto dsPIC33FJ, fuente el autor.

Se describirán las configuraciones realizadas para cada recurso hardware del dsPIC33FJ128MC706A utilizado y los cálculos que realiza cada subrutina según sea el caso para realizar una tarea específica. Para cada subrutina se mostrara un diagrama de flujo que simplifique la secuencia de ejecución según sea el caso.

Al final del capítulo se tendrá un diagrama de flujo general para sintetizar la explicación del programa desarrollado en lenguaje C. Las subrutinas de programación pueden ser consultadas en el ANEXO C.

4.3.1. Configuración e Implementación Entradas/Salidas

Este módulo de código **IOPorts.c** básicamente se utiliza para inicializar todos los periféricos de entrada / salida y configuración de la frecuencia del oscilador. Contiene 4 funciones 1 de estas activada cuando se genera interrupción por **CN.** A continuación se describe cada función.

Init_IO(): Esta función es de inicialización llamada desde el main_Cerebot del programa, que configuran como salida los 8 pines de los módulos MCPWM1-MCPWM2(pines del dsPIC33FJ128MC706A RE0-RE7), los 4 pines de lectura para los dos Encoder pines RB4-RB5 motor#1(Elevación) y RC13-RC14 motor#2(Giro). También los 4 leds son pines de salida y los 2 push-botton como entradas disponibles en la Cerebot MC7^{TM}.

La configuración del oscilador de la CPU del microcontrolador es hecho en esta función con la ecuación $4 \cdot 2$ y en esta se realiza cálculo de la frecuencia del reloj de operación del dsPIC33F con la ecuación $4 \cdot 2$.

$$F_{OSC} = F_{IN}\left(\frac{M}{N1 \times N2}\right) \qquad (4 \cdot 2)$$

$$F_{CY} = \frac{F_{OSC}}{2} \qquad (4 \cdot 3)$$

Ya que el cristal XT que tiene la tarjeta Cerebot MC7TM es $F_{IN} = 8MHz$ para hallar la frecuencia de oscilación pensando en tener operando el procesador del dsPIC33F a la mayor frecuencia, se utiliza N1=2, N2=2, M=40, entonces reemplazando en $4 \cdot 2$.

$$F_{OSC} = 8MHz\left(\frac{40}{2 \times 2}\right) = 80MHz$$

Reemplazando para halla frecuencia de operación

$$F_{CY} = \left(\frac{80M}{2}\right) = 40MIPS$$

Finalmente la velocidad de ejecución del código en el dsPIC es 40MIPS (40 millones de instrucciones por segundo).

Init_CN(): Es una función también de inicialización llamada desde el main_Cerebot del programa, configura el pin CN1/RC13 como fuente de interrupción que se inicia el contador de posición en Cont_ENC2 = 3714, más adelante se explica porque se carga este valor.

_**CNInterrupt()**: Es la función de interrupción generada por el pin RC13 conectado al Encoder. Ya que el Encoder tiene 2 canales de entrada se utiliza el pin RC14 para aumentar la resolución a 2x. Una vuelta del motoreductor cuenta 3714 pulsos en un contador que se incrementa al detectar giro hacia un sentido. La prioridad de esta interrupción es configurada a 2 ya que no es crítica para el funcionamiento del control.

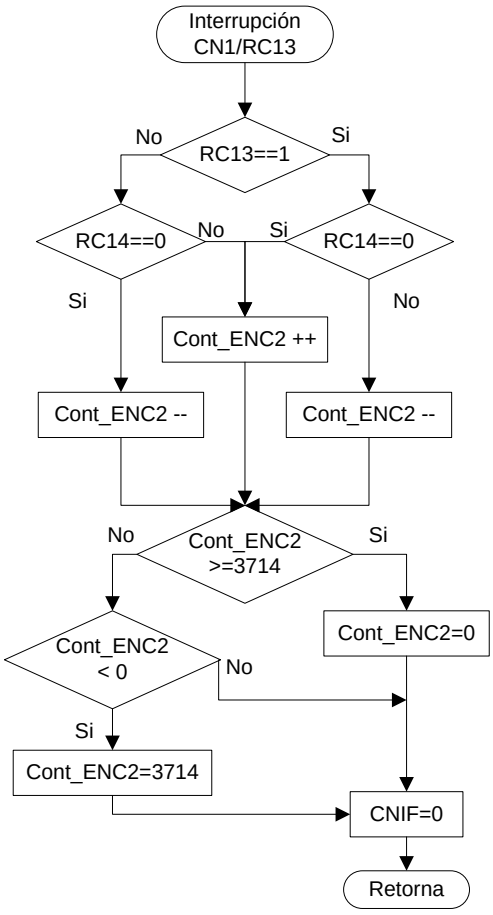


FIGURA 4-4. Diagrama de flujo Interrupción CN1/RC13, fuente el autor.

Posicion_Encoder2(): Esta función calcula de acuerdo al número del contador Cont_ENC2 y retorna la posición en grados, en un rango de 0-360°.

Para este cálculo, experimentalmente se halló la relación de reducción del motoreductor, encontrando que es 18.57:1 a partir del número de pulsos del encoder por revolución del eje del reductor. Entonces para completar la ecuación $(4 \cdot 4)$ la constante $Factor\_Pos\_Enc2$ es definida en el archivo <Init_Module.h> y se calcula de la siguiente forma:

$$Factor\_Pos\_Enc2 = \frac{360}{(18.57 \times 200)} = 0.0969305331$$

$$Pos\_Motorx = 360 - Cont\_ENCx * Factor\_Pos\_Encx \qquad (4 \cdot 4)$$

Donde x corresponde al "número de motor" y encoder que en este caso es encoder2 correspondiente al motor#2 de Giro(ángulo ψ) de la plataforma Altazimut.

4.3.2. Configuración e Implementación TIMER1

Este módulo de código **TIMER_Module.c** funciona en combinación con la interrupción para poder establecer la ejecución del control a un intervalo de tiempo regular. Y contiene 2 funciones más, una de inicialización y otra de interrupción.

Init_timer1(): Función de inicialización del TIMER1, se habilitan interrupción con el registro **IEC0bits.T1IE=1** , selección de reloj interno F_{CY} , preescala 1:8, prioridad de interrupción a 6 por ser la que genera la bandera que da el tiempo periódico de ejecución del control. También se carga el registro **TMR1**=65285 para el cálculo del tiempo en que se genera la interrupción. Se crea la variable global de **anidador_10ms** con la finalidad de contabilizar 10mseg. Y la variable **flag_10ms** que es bandera de indicación del tiempo de conteo de 10ms.
El tiempo base con un reloj de $F_{CY} = 80MHz$ se tiene una frecuencia de ciclos máquina $F_{CICLO} = \frac{80MHz}{4} = 20MHz$. Por lo que un ciclo máquina corresponde a de ecuación $(2 \cdot 3)$ $tiempo\_ciclo = 50ns$. En principio, el contador del TIMER1 se incrementará cada 50 nanosegundos.

Ya que se desea una duración de tiempo del TIMER1 a 100 microsegundos se utiliza la ecuación $(2 \cdot 4)$ para despejar TMR1.

$$TMR1 = \frac{T\_TIMER1}{tiempo\_ciclo \times Preescala} = \frac{100\mu s}{50ns \times 8} = 250 \qquad (4 \cdot 5)$$

Como el registro TMR1 es de 16 bits de conteo incremental entonces el valor debe cargarse es $TMR1 = 65535 - 250 = 65285$.

Para tener un tiempo de 10milisegundos se utiliza una variable como contador que active una bandera indicando que se cumplió el tiempo. Utilizando la expresión $(4 \cdot 5)$ calculamos el nuevo contador:

$$T\_TIMER\_10ms = T_{Ciclo} \times Preescala \times TMR1 \times anidador\_10ms \qquad (4 \cdot 7)$$

Despejando

$$anidador\_10ms = \frac{T\_TIMER\_10ms}{T_{Ciclo} \times Preescala \times TMR1} = \frac{10ms}{50ns \times 8 \times 250} = 100 \qquad (4 \cdot 8)$$

Para confirmar se reemplazan los valores hallados en $(4 \cdot 7)$:

$$T\_TIMER\_10ms = 50ns \times 8 \times 250 \times 100 = 10ms$$

_**T1Interrupt()**: Es la función de interrupción generada por el TIMER1 que fue explicado en el capítulo 2, básicamente al cumplirse los 10ms se activa la bandera **flag_100ms** que se utiliza para ejecutar el algoritmo control.

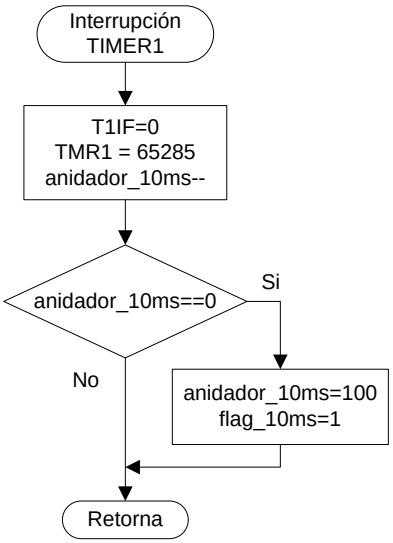


FIGURA 4-5. Diagrama de flujo Interrupción TIMER1, fuente el autor.

4.3.3. Configuración e Implementación QEI

Este módulo de código **QEI_Module.c** contiene 2 funciones, una es la función de inicialización del recurso interface de encoder en cuadratura del dsPIC33FJ128MC706A. Y otra función es la que calcula la posición del motor#1(elevación).

Init_QEI1(): Es una función también de inicialización llamada desde el main_Cerebot del programa, configura el módulo hardware QEI del microcontrolador con las siguientes características: modo 4x para aumentar resolución, conteo en dirección positiva, carga el registro POSCNT=5569 para que la posición inicial sea 90° con inclinación en 0°, además el registro MAXCNT=7424 explicados en el capitulo 2 y finalmente se activa el contador del QEI de 16bits con QEIM1=1.

Posicion_Encoder1(): Esta función calcula de acuerdo al número del contador POSCNT del módulo QEI retorna la posición en grados en un rango de 0-360°.
Al igual que la función utilizada para el cálculo de posición del motor#2(giro) experimentalmente se halló la relación de reducción del motoreductor, encontrando que es 18.56:1 a partir del número de pulsos del encoder por revolución del eje del reductor. Para hallar la constante $Factor\_Pos\_Enc1$ de la ecuación $(4 \cdot 4)$ que es definida en el archivo <Init_Module.h> y se calcula de la siguiente forma:

$$Factor\_Pos\_Enc1 = \frac{360}{(18.56 \times 400)} = 0.04849137931$$

Reemplazando en la ecuación $(4 \cdot 4)$ donde x corresponde al número de motor y encoder que en este caso es encoder1 correspondiente al motor#1 de Elevación(ángulo θ) de la plataforma Altazimut, tenemos:

$$Pos\_Motor1 = 360 - Cont\_ENC1 * Factor\_Pos\_Enc1 \qquad (4 \cdot 9)$$

Por lo tanto la expresión final $(4 \cdot 9)$ será utilizada para retornar el dato de posición angular del eje del motoreductor.

4.3.4. Configuración e Implementación PWM

Este módulo de código **PWM_Module.c** contiene 5 funciones, 2 de inicialización correspondientes para el MCPWM1 y MCPWM2 recursos del dsPIC33FJ128MC706A explicados en el capítulo 2. Y otras 3 funciones más de implementación de los MCPWM para controlar los puente H de los motores de giro y elevación de la plataforma altazimut y una función de parada de ambos motores. A continuación se describe las funciones

Init_PWM1(): Esta función inicializadora del módulo MCPWM1 del dsPIC33FJ128MC706A, carga 0 a los pines de salida PWM1L- PWM1H, PWM2L- PWM2H para evitar que se activen los motores, con el registro P1TCON es configurado el Prescaler a 1:4, para la base de tiempo cuando está en modo Idle, postescala en 1:1, modo de operación del tiempo base del PWM en conteo Up/Down. Las demás configuraciones hechas por código son explicadas en el capítulo 2 y se puede consultar el código fuente en el ANEXO C.

Para calcular el valor que se cargará en el registro PTPER se utiliza la ecuación $(2 \cdot 1)$ ya que se desea que el PWM funcione a una frecuencia de $F_{PWM} = 20kHz$ y reemplazando las constantes configuradas tenemos:

$$PTPER = \frac{40M}{20k \times (4) \times 2} - 1 = 249$$

Se carga entonces el registro PTPER=249, y se habilita el tiempo base del módulo PWM con el bit PTEN=1.

Init_PWM2(): Esta función inicializadora del MCPWM2 es exactamente configurada igual a la **Init_PWM1()** pero con los pines de salida PWM3L- PWM3H y PWM4L- PWM4H. Ver código fuente en el ANEXO C.

MCPWM1(int x): Es una función con un argumento pasado por el valor entero **x**, la que se encarga del control del motor#1 a través del módulo MCPWM1 donde el signo de **x** da el sentido de giro. El ciclo útil es cargado en los registros P1DC1- P1DC2 según sea el sentido de giro del motor explicado en el capítulo 2 apartado Control de

motores por MOSFET. Este valor esta en un rango -510 a 510, si este valor es positivo la dirección (+) es en el sentido horario y la polarización del motor#1 será VMOTOR1(+) VMOTOR2(-) pero si el valor es negativo la dirección(-) es en el sentido anti horario y la polarización del motor#1 será VMOTOR1(-) VMOTOR2(+) según el circuito esquemático del ANEXO C. Y es llamada como subrutina en la ejecución del control de posición o estabilización.

La figura 4-6 contiene el diagrama de flujo de la subrutina donde se simplifica la explicación.

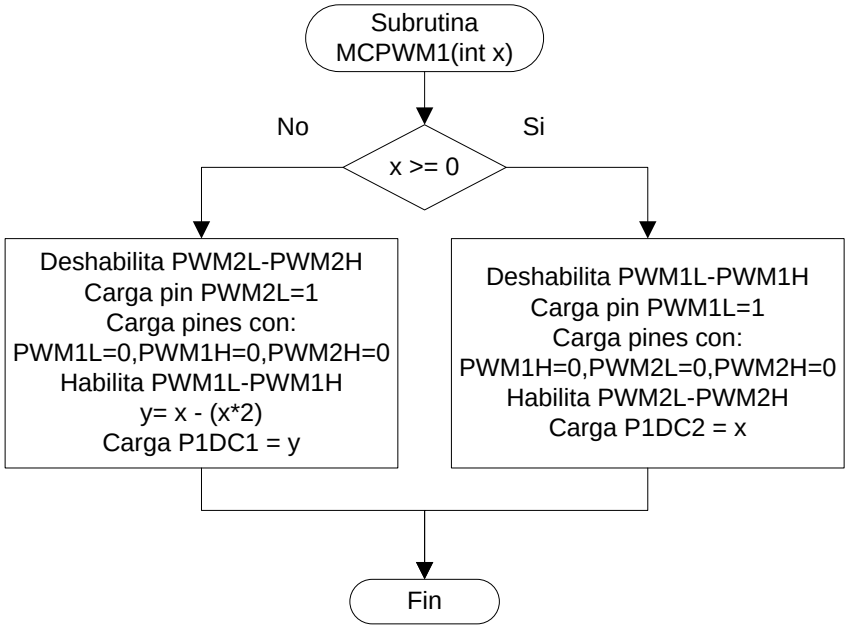


FIGURA 4-6. Diagrama de flujo Subrutina MCPWM1(int x), fuente el autor.

MCPWM2(int x): Es una función es idéntica en operación pero para el control del motor#1 a través del módulo MCPWM2. El ciclo útil es cargado en los registros P1DC3- P1DC4 según sea el sentido de giro del motor. Si el valor **x** es positivo la dirección (+) es en el sentido horario y la polarización del motor#2 será VMOTOR3(+) VMOTOR4(-) pero si el valor es negativo la dirección(-) es en el sentido anti horario y la polarización del motor#2 será VMOTOR3(-) VMOTOR4(+) según el circuito esquemático del ANEXO C. Y también es llamada como subrutina en la ejecución del control de posición o estabilización.

La figura 4-7 contiene el diagrama de flujo de la subrutina donde se simplifica la explicación.

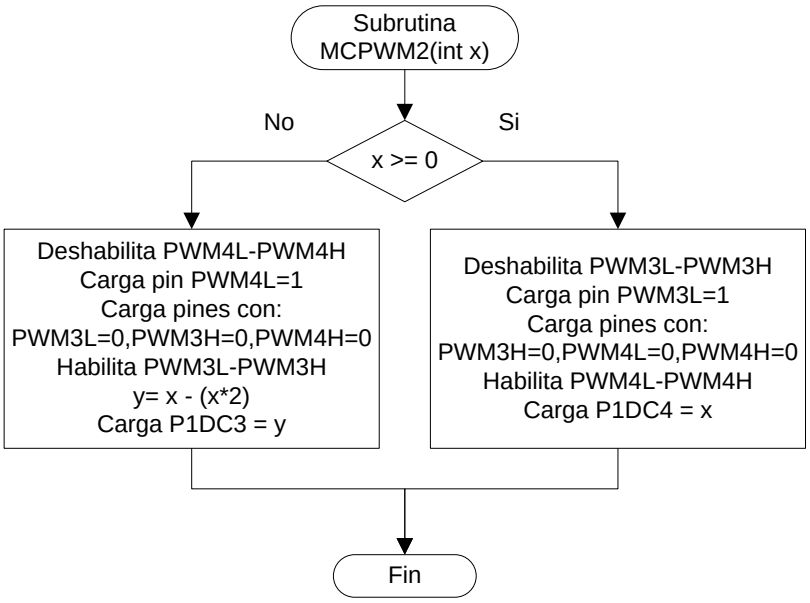


FIGURA 4-7. Diagrama de flujo Subrutina MCPWM2(int x), fuente el autor.

Stop_PWM(): Esta subrutina es utilizada para deshabilitar los dos módulos MCPWM1-MCPWM2 del dsPIC33FJ128MC706A y además coloca todos los pines de salida que controlan los puente H a 0. Esta subrutina es llamada desde los módulos software **INT_External.c** para protección por sobre corriente y **USART_Module.c** por comando desde la GUI. La figura 4-8 muestra el diagrama de flujo de ejecución de la subrutina.

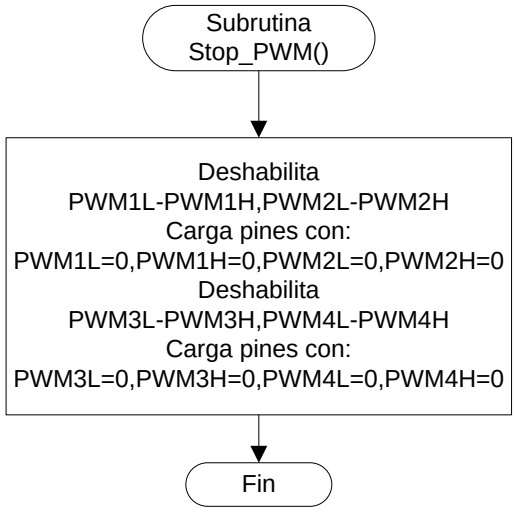


FIGURA 4-8. Diagrama de flujo Subrutina Stop_PWM(), fuente el autor.

4.3.5. Configuración e Implementación USART

Este módulo de código **USART_Module.c** contiene 5 funciones, 2 de inicialización correspondientes al uso de la USART1 y USART2 que son recursos del dsPIC33FJ128MC706A explicados en el capítulo 2. Contiene 2 funciones de interrupción para la recepción de datos por el módulo USART1 y la USART2, también una función para la subrutina de envío de datos a través de la USART1.

A continuación se describe las funciones

Init_Usart(): Es función inicializadora del módulo USART1 del dsPIC y básicamente es donde se configura la recepción y transmisión que incluye características propias de la comunicación serial RS232, fueron configuradas de la siguiente manera: la velocidad a 115200 baudios, paridad ninguna, comunicación datos de transmisión y recepción full-dúplex a 8bits, 1 bit de parada, modo alta velocidad y se activa la interrupción por recepción(Rx). Para el cálculo del valor a cargar en el registro UBRG es utilizada la ecuación $(2 \cdot 2)$ que es definida en el archivo <Init_Module.h> y se calcula de la siguiente forma:

$$UBRG = \frac{40M}{4 \times 115200} - 1 = 85.8$$

En el ANEXO C contiene el código fuente de esta función, que es llamada desde el main_Cerebot.c para inicializar este módulo para la comunicación serial con la GUI desarrollada en MATLAB®.

Init_Usart2(): Es función inicializadora del módulo USART2 del dsPIC configurada igual a la Init_Usart(). Ver ANEXO C, la función es llamada desde el main_Cerebot.c para inicializar este módulo para la comunicación serial con la Seeeduino.

putUART1(char c): Es una función con un argumento pasado por valor del char **c**, es bastante simple y es encargada de enviar un carácter por la interfaz serial USART1 cargado **c** en el registro U1TXREG(Ver ANEXO C).

_**U1RXInterrupt()**: Es la función de interrupción generada por recepción de datos de la USART1 guarda el dato en el registro de U1RXREG. En esta función se implementaron comandos (ver tabla 4-7) para interpretar la información recibida de la GUI y los cuales ejecutan acciones correspondientes a los botones y textos de edición de la interfaz grafica de usuario.

Descripción	Comando
Comandos Generales	
Comando de Start(Run Control PID)	**"S"**
Comando de Stop Control PID	**"s"**
Comando de enganche Control Estabilizacion ALTAZIMUT	**"E"**
Comando de desenganche Control Estabilizacion ALTAZIMUT	**"e"**
Comandos de Lectura dsPIC33FJ128MC706A	
Comando de Read(dsPIC a MATLAB) Parametros PID1-Motor1	**"R"**
Comando de Read(dsPIC a MATLAB) Parametros PID2-Motor2	**"r"**
COMANDOS ESCRITURA PID-MOTOR 1	
Comando de CAMBIO de Setpoint Control PID-Motor1	**"C"**
Comando de cambio de Kp Control PID-Motor1	**"P"**
Comando de cambio de Ki Control PID-Motor1	**"I"**
Comando de cambio de Kd Control PID-Motor1	**"D"**
COMANDOS ESCRITURA PID-MOTOR 2	
Comando de CAMBIO de Setpoint Control PID-Motor2	**"c"**
Comando de cambio de Kp Control PID-Motor1	**"p"**
Comando de cambio de Ki Control PID-Motor1	**"i"**
Comando de cambio de Kd Control PID-Motor1	**"d"**

Tabla 4-7. Comandos implementados, fuente el autor.

La recepción de datos en el arreglo **ArrayRx[PointerRx]** de 20 datos como el buffer el cual será llenado hasta encontrar un final de trama especificado con el carácter '**}**', luego esta trama es descompuesta según el comando recibido desde la GUI, en el caso de comando de escritura se utilizo la función **atof()**; nativa en el lenguaje C en la librería stdlib, y que convierte un dato string a float entonces la trama de datos recibida sería de la siguiente manera:

Byte1	Byte2	Byte3	Byte n	Byte(n+1)
Carácter Comando **" "**	dato	dato	….	**"}"**

Tabla 4-8. Trama de recepción de datos USART1, fuente el autor.

La trama recibida se reorganiza para tener los bytes que contienen los datos en un arreglo temporal **ReadString** para luego ser convertido en un dato tipo float con la función **atof().** La figura 4-9 se muestra el diagrama de flujo de la interrupción.

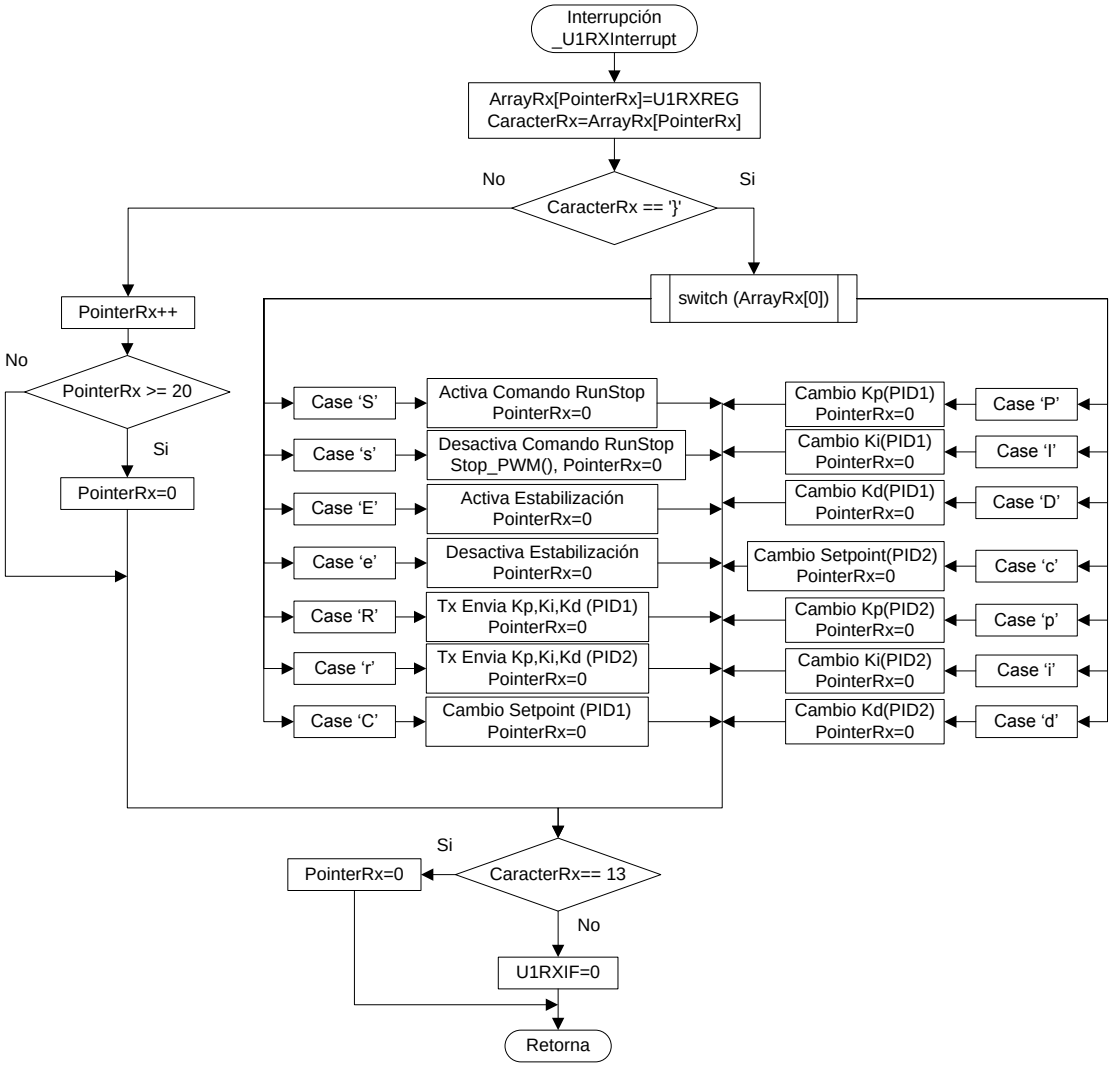


FIGURA 4-9. Diagrama de flujo Interrupción USART1, fuente el autor.

_**U2RXInterrupt()**: Es la función de interrupción generada por recepción de datos de la USART2 guarda el dato en el registro de U2RXREG. La recepción de datos en el arreglo **ArrayRx2[PointerRx2]** de 50 datos como el buffer el cual será llenado hasta encontrar un final de trama especificado con el carácter '**}**', luego esta trama es descompuesta de la misma forma que se hace en la interrupción de la función de la USART1, esto para interpretar la información recibida de la Seeeduino una trama mostrada en la Tabla 4-6.

La trama recibida se reorganiza para tener los bytes que contienen los datos en un arreglo temporal **ReadString_2** para luego ser convertido en un dato tipo float con la función **atof().** La figura 4-10 se muestra el diagrama de flujo de la interrupción.

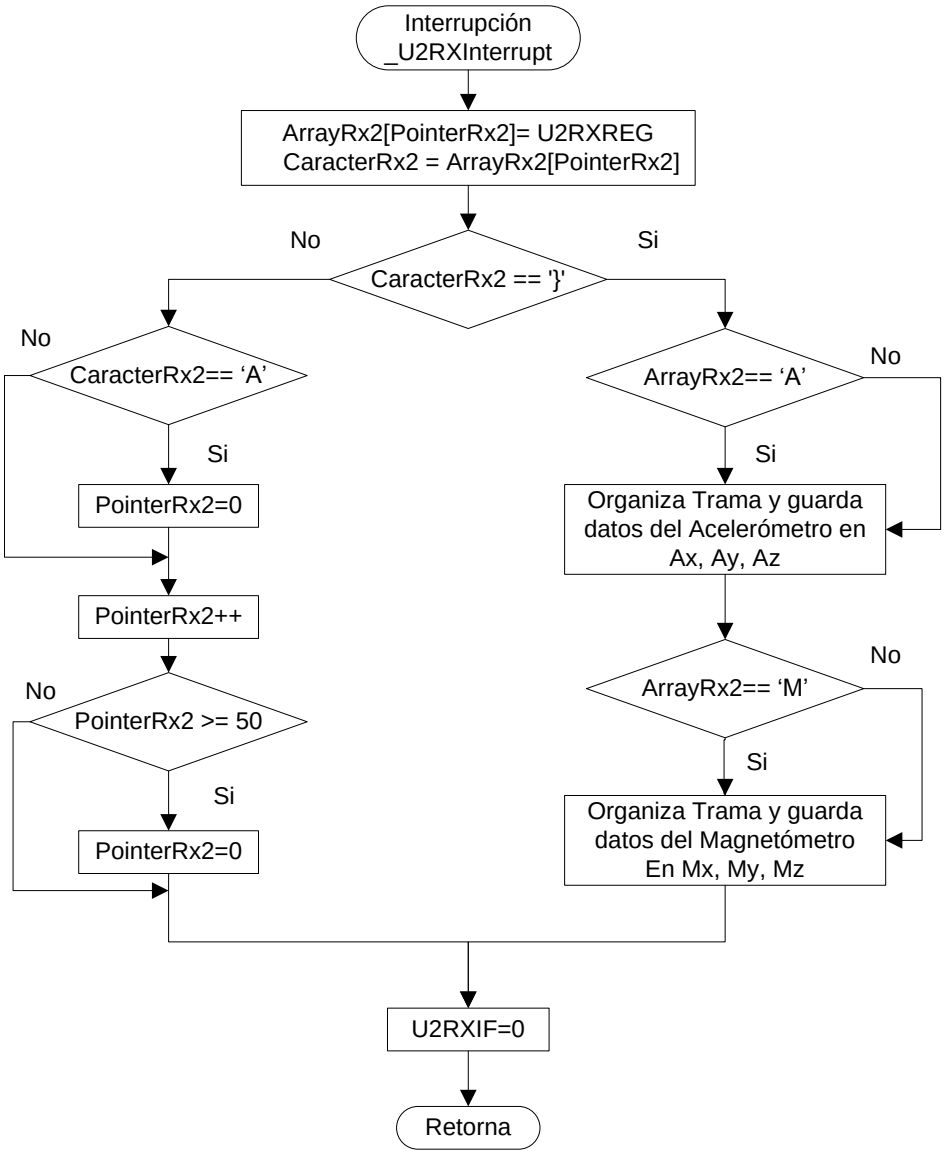


FIGURA 4-10. Diagrama de flujo Interrupción USART2, fuente el autor.

4.3.6. Protección Contra sobre Corriente

Para la protección de sobre corriente fue implementada la interrupción externa por periféricos en el pin RD8. Este pin tiene conectado un comparador (ver ANEXO A) que determina condición de sobre corriente. La señal es activada cuando se detecta 6A en cualquier medio Puente H. En el módulo de código **INT_External.c** están contenidas las dos funciones, una de inicialización y otras correspondientes a la función de interrupción.

Init_INTx_OI(): Esta función se inicializa como fuente de interrupción externa el pin RD8 con detección de flanco de subida, y la prioridad es configurada como 7 la más alta para protección de hardware de la Cerebot MC7^{TM}.

_**INT1Interrupt()**: Esta función básicamente lo que hace es desactivar todos los pines del módulo MCPWM y colocarlos a 0, es muy parecida a la subrutina de Stop_PWM() con la diferencia que esta coloca un retardo de 10 segundos antes de volver a iniciar la operación normal. La figura 4-11 se muestra el diagrama de flujo de la interrupción.

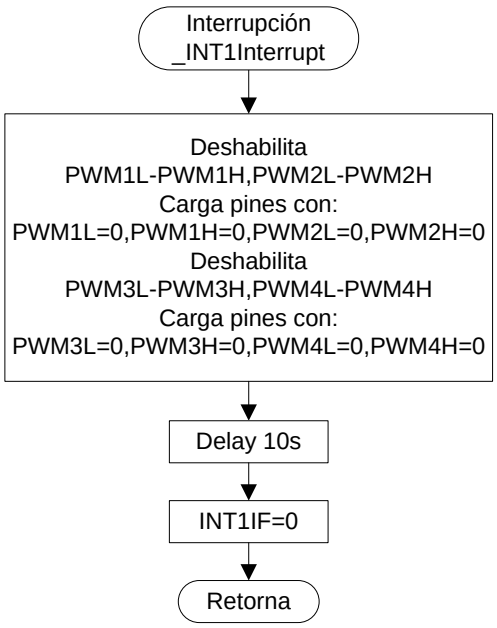


FIGURA 4-11. Diagrama de flujo Interrupción INT1 externa, fuente el autor.

4.3.7. Configuración e Implementación LSM303DLHC

Este módulo de código **LSM303DLHC_Module.c** contiene 5 funciones, una de inicialización con los valores de calibración del compas magnético, una función principal que, con base a los datos brutos o raw enviados por la Seeeduino, se encarga de calcular los ángulos denominados Elevación o Pitch(ψ) /Roll(Φ)/Giro o Heading(θ), las otras cuatro son subrutinas para realizar los cálculos mencionados. Entonces siguiendo la nota de aplicación [18] del fabricante STMicroelectronics el ángulo de Giro(ψ) se hallará utilizando los datos del compás magnético compensado

por inclinación utilizando los datos del acelerómetro y para el cálculo de Elevación(θ) solamente los datos del acelerómetro.

4.3.7.1. Calibración Magnetómetro del LSM303DLHC

Debido a que el compas magnético se comporta como una brújula es susceptible a interferencias de campo magnético a cuerpos ferrosos este tipo de distorsión es conocido con hard-iron producido por materiales que presentan una adición en campo constante magnético de la tierra, generando de ese modo una adición al valor constante a la salida de cada uno de los ejes del magnetómetro. La plataforma Altazimut está construida en aluminio y hierro por lo que este tipo de distorsión esta presentes y es visiblemente identificada por un desplazamiento del origen del círculo ideal desde (0, 0) a H_x-H_y, como se muestra en la figura 4-12.

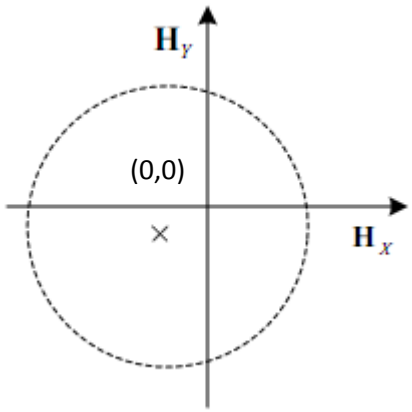


FIGURA 4-12. Distorsión hard-iron, tomado de [19].

La correcciones de hard-iron se determinan normalmente mediante la rotación del sensor a través de un mínimo de 360° realizando movimientos dentro de una esfera imaginaria, entonces la determinación de la distancia a partir de (0,0) en el centro del círculo mediante la identificación de la media de los valores máximos y mínimos para cada uno de los ejes. Estos valores se hallan ejecutando una rutina de calibración utilizando la librería Arduino(ver ANEXO B).

4.3.7.2. Cálculo de Ángulos Pitch(θ) y Roll(Φ) con datos del LSM303DLHC

Teniendo en cuenta que la plataforma Altazimut es un sistema Strapdown lo que quiere decir que los ejes de la unidad sensorial LSM303DLHC están referenciados y atados de igual manera que los de la plataforma, esto se pude observar en la imagen 4-12, donde los ejes del sensor son equivalentes a los de la plataforma y se mueve con esta. En este caso solamente se tiene movimiento en 2 ejes Giro o Heading(ψ) y Elevación o Pitch(θ) de la plataforma Altazimut.

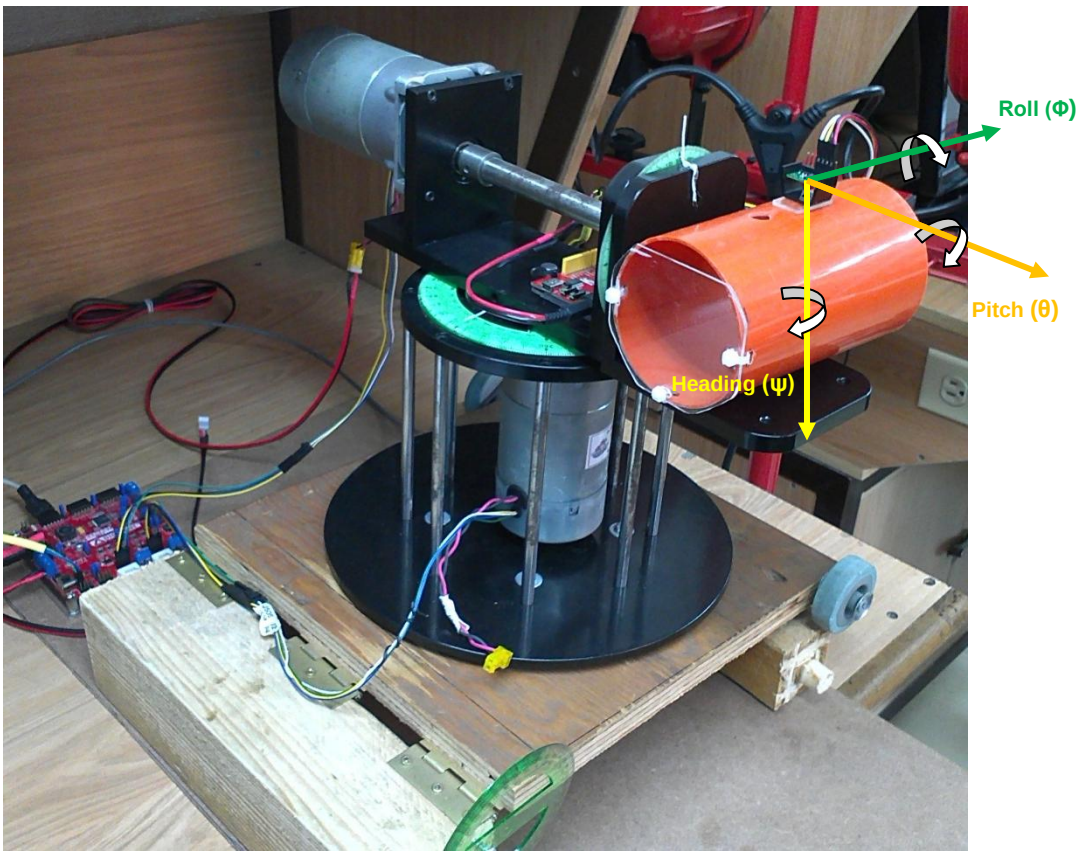


FIGURA 4-13. Ángulos lectura de LSM303DLHC, fuente el autor.

Entonces estos ángulos podemos definirlos según la ubicación del sensor como lo mencionan en [18] y que es mostrada en la figura 4-14 donde X_b , Y_b y Z_b son los ejes de un dispositivo, $X_{A,M}$, $Y_{A,M}$, y $Z_{A,M}$ son ejes de detección del acelerómetro y el sensor magnético, respectivamente . Estos ángulos son descritos a continuación:

El ángulo de giro, azimut o heading (ψ): se define como el ángulo con respecto al polo norte magnético. Siempre es positivo desde 0 º a 359 º al girar alrededor del eje Z_b visto desde arriba con la regla de la mano derecha.

El ángulo elevación, alta o pitch (θ): se define como el ángulo entre el eje X_b y el plano horizontal. Va desde 0 º a 90 º al girar alrededor del eje Y_b con el eje X_b en movimiento hacia arriba. Cuando el eje X_b se mueve hacia abajo, el ángulo va de 0 º a -90 º.

El ángulo alabeo o roll (Φ): se define como el ángulo entre el eje de Y_b y el plano horizontal. Va desde 0 º a 90 º al girar alrededor del eje X ter en los Y_b eje en movimiento hacia abajo. Cuando el eje Y_b se mueve hacia arriba, el ángulo de balanceo va de 0 º a -90 º.

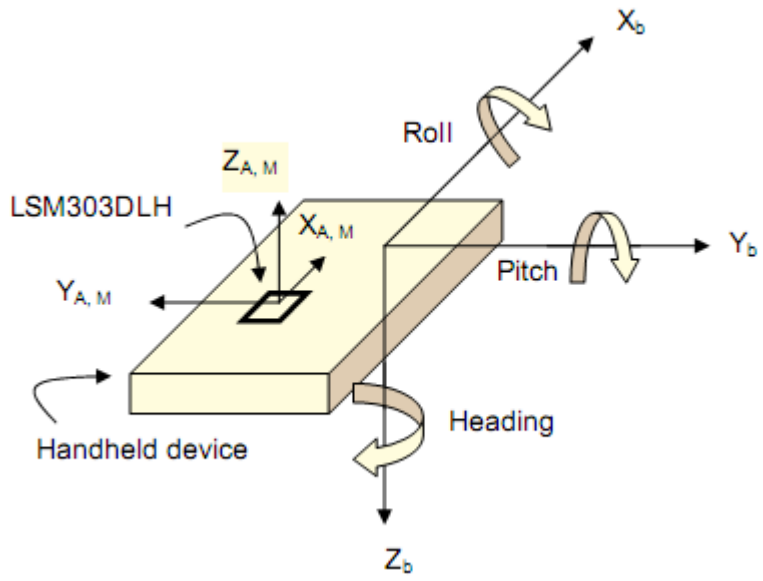


FIGURA 4-14. Sistema Coordenado compas magnético del LSM303DLHC, tomado de [18].

Entonces siguiendo la nota de aplicación [18] del fabricante STMicroelectronics el ángulo de Giro(ψ) se hallará utilizando los datos del compas magnético compensado por inclinación utilizando los datos del acelerómetro y para el cálculo de Elevación(θ) solamente los datos del acelerómetro.

Continuando con el procedimiento para calcular los ángulos de **elevación o pitch (θ)** y **alabeo o roll (Φ)** serán utilizados únicamente los datos proporcionados por el acelerómetro que al llevar a una posición arbitraria 3D con los ejes X'_b, Y'_b y Z'_b, hay

algunos procedimientos de rotación para girar el marco de referencia desde el nivel local X_b, Y_b y Z_b, de la figura 4-14, a la posición 3D. Diferentes procedimientos de rotación resultan en matriz de rotación diferente.

En primer lugar, girar el eje Z_b en sentido de las agujas del reloj un ángulo(ψ) con la vista desde el origen a la baja. Luego se gira el eje alrededor de Y_b en un ángulo(θ) con X_b hacia arriba. Finalmente se gira el eje alrededor X_b en un ángulo (Φ) con Y_b hacia abajo. Entonces los nuevos ejes de referencia se convierten en X'_b, Y'_b, y Z'_b, como se muestra en la Figura 9.

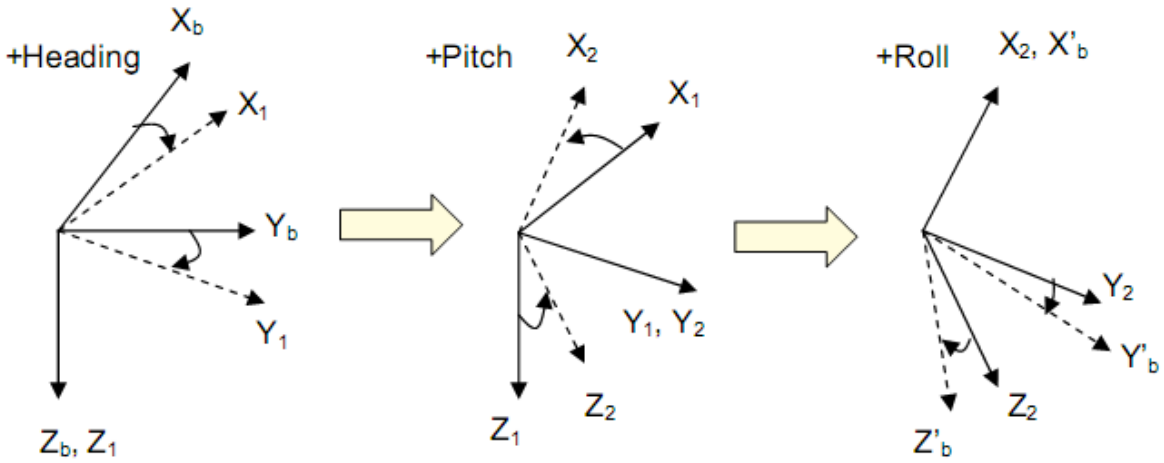


FIGURA 4-15. Procedimientos de rotación, tomado de [18].

Cada matriz de rotación puede definirse según el ángulo como:

$$R_{\psi} = \begin{bmatrix} \cos\psi & \text{sen}\psi & 0 \\ -\text{sen}\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4 \cdot 10)$$

$$R_{\theta} = \begin{bmatrix} \cos\theta & 0 & -\text{sen}\theta \\ 0 & 1 & 0 \\ \text{sen}\theta & 0 & \cos\theta \end{bmatrix} \quad (4 \cdot 11)$$

$$R_{\Phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \text{sen}\Phi \\ 0 & -\text{sen}\Phi & \cos\Phi \end{bmatrix} \quad (4 \cdot 12)$$

Y la relación entre X'_b / Y'_b / Z'_b y X_b / Y_b / Z_b es:

$$\begin{bmatrix} X_b' \\ Y_b' \\ Z_b' \end{bmatrix} = R_\Phi R_\theta R_\psi \cdot \begin{bmatrix} X_b \\ Y_b \\ Z_b \end{bmatrix} \qquad (4 \cdot 13)$$

En el plano horizontal local mostrado en la figura 4-14, $X_b = Y_b = 0$, $Z_b = 1g$. En X'_b / Y'_b / Z'_b, las mediciones en bruto o raw del acelerómetro LSM303DLH son A_x, A_y y A_z. Se utilizan los valores normalizados del acelerómetro tipo float y representados por A_{x1}, A_{y1} y A_{z1} se convierten en menos de 1 en términos de g(gravedad de la tierra). Reemplazando en la ecuación $(4 \cdot 13)$ se convierte en:

$$\begin{bmatrix} A_{x1} \\ A_{y1} \\ A_{z1} \end{bmatrix} = R_\Phi R_\theta R_\psi \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad (4 \cdot 14)$$

Y reemplazando en $(4 \cdot 14)$ las matrices de rotación para cada ángulo tenemos:

$$\begin{bmatrix} A_{X1} \\ A_{Y1} \\ A_{Z1} \end{bmatrix} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\,\mathrm{sen}\psi & -\mathrm{sen}\theta \\ \cos\psi\,\mathrm{sen}\theta\,\mathrm{sen}\Phi - \cos\Phi\,\mathrm{sen}\psi & \cos\Phi\cos\Phi + \mathrm{sen}\theta\,\mathrm{sen}\Phi\,\mathrm{sen}\psi & \cos\theta\,\mathrm{sen}\Phi \\ \cos\psi\,\mathrm{sen}\theta\cos\Phi + \mathrm{sen}\Phi\,\mathrm{sen}\psi & -\mathrm{sen}\Phi\cos\psi + \mathrm{sen}\theta\cos\Phi\,\mathrm{sen}\psi & \cos\theta\cos\Phi \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Por lo tanto, el ángulo de **elevación o pitch (θ)** y **alabeo o roll (Φ)** se puede calcular como:

$$\text{Pitch} = \theta = \mathrm{sen}^{-1}(-A_{X1}) \qquad (4 \cdot 15)$$

$$\text{Roll} = \Phi = \mathrm{sen}^{-1}\left(\frac{A_{Y1}}{\cos\theta}\right) \qquad (4 \cdot 16)$$

4.3.7.3. Cálculo de Ángulo Heading(ψ) con datos del LSM303DLHC

Para el cálculo del ángulo Heading(ψ), con las mediciones del sensor magnético de 3 ejes se normalizan sus componentes X-Y-Z luego de realizar la calibración del mismo, este vector de campo se proyecta en el plano horizontal y se realiza compensación de inclinación con las mediciones del acelerómetro, como se muestra en la figura 4-16.

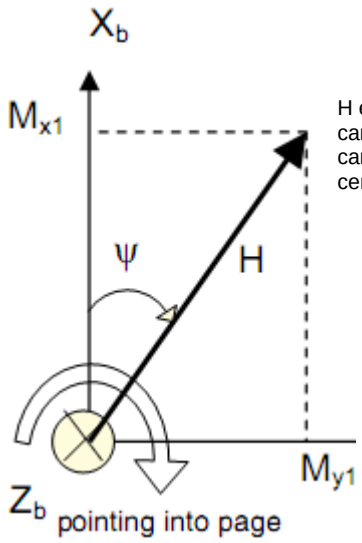


FIGURA 4-16. Cálculo de ángulo de Giro(ψ), tomado de [18].

Siguiendo la nota de aplicación con la figura 4-14, y si giran los ejes $X_b/Y_b/Z_b$ a $X''_b/Y''_b/Z''_b$ por la rotación del ángulo roll(Φ) de seguido por la rotación del ángulo de elevación o pitch(θ),

$$\begin{bmatrix} X_b \\ Y_b \\ Z_b \end{bmatrix} = R_\Phi^{-1} R_\theta^{-1} \begin{bmatrix} X''_b \\ Y''_b \\ Z''_b \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \text{sen}\theta \\ \text{sen}\Phi\text{sen}\theta & \cos\Phi & -\text{sen}\Phi\cos\theta \\ -\cos\Phi\text{sen}\theta & \text{sen}\Phi & \cos\Phi\cos\theta \end{bmatrix} \cdot \begin{bmatrix} X''_b \\ Y''_b \\ Z''_b \end{bmatrix} \qquad (4 \cdot 17)$$

Entonces siendo M_{x1}, M_{y1} y M_{z1} las mediciones de los sensores magnéticos normalizadas y después de aplicar la corrección por distorsión hard-iron con los parámetros de calibración en el sensor magnético a las mediciones brutas o raw M_x, M_y y M_z en nuevas posiciones $X''_b/Y''_b/Z''_b$. A partir de la ecuación $(4 \cdot 17)$, las componentes M_{x2}, M_{Y2}, y M_{Z2} con la compensación por inclinación se puede obtener como:

$$M_{X2} = M_{X1}\cos\theta + M_{Z1}\text{sen}\theta \qquad (4 \cdot 18)$$

$$M_{Y2} = M_{X1}\text{sen}\Phi\text{sen}\theta + M_{Y1}\cos\Phi - M_{Z1}\text{sen}\Phi\cos\theta \qquad (4 \cdot 19)$$

$$M_{Z2} = -M_{X1}\cos\Phi\text{sen}\theta + M_{Y1}\text{sen}\Phi + M_{Z1}\cos\Phi\cos\theta \qquad (4 \cdot 20)$$

Donde los ángulos de giro(θ) y elevación(Φ) son hallados con el acelerómetro con las ecuaciones $(4 \cdot 15)$ y $(4 \cdot 16)$.

Finalmente el ángulo

$$\psi = \text{tang}^{-1}\left(\frac{M_{Y2}}{M_{X2}}\right) \qquad (4 \cdot 21)$$

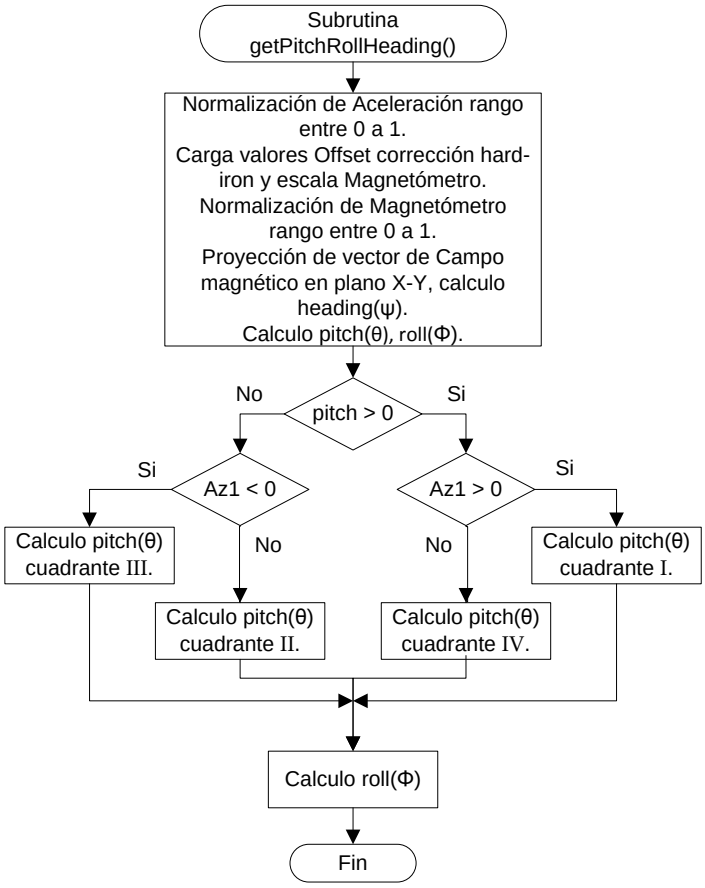


FIGURA 4-16. Diagrama de flujo subrutina cálculo de ángulos, fuente el autor.

El diagrama de flujo 4-16 muestra el diagrama de flujo de la subrutina getPitchRollHeading() que para calcular los ángulos de giro, elevación y alabeo que básicamente utiliza las 3 subrutinas para realizar operaciones de normalización, producto punto y producto cruz de vectores para así hallar la proyección del vector de campo magnético sobre el plano X-Y.

A partir del ángulo pitch(θ) calculado y la componente A_{z1} de aceleración normalizada se definen los cuadrantes en el movimiento de elevación de la plataforma altazimut con la finalidad de tener los grados 0-90° en la parte superior en los cuadrantes 1-4 del plano cartesiano conformado por X-Z como se muestra en la figura 4-17.

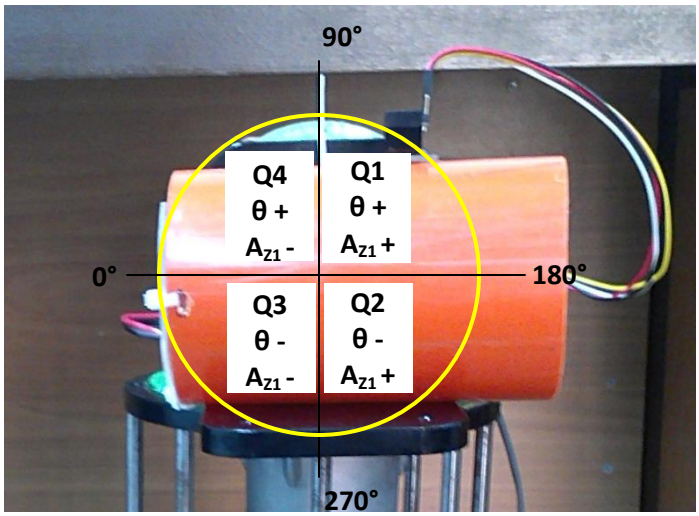


FIGURA 4-17. Cuadrantes Elevación (θ), fuente el autor.

4.3.8. Programa Principal

Finalmente el módulo **main_Cerebot.c** es el programa principal que se ejecuta en el dsPIC33FJ128MC706A sobre la tarjeta Cerebot MC7^{TM} que se encarga de hacer los llamados a las subrutinas necesarias para realizar el control de posición de la plataforma Altazimut basado en los comandos configurados desde la GUI. En la figura 4-19 se simplifica con un diagrama de flujo la ejecución del main.

En este módulo software se ejecuta un regulador o controlador PID (proporcional, integral, derivativo) que básicamente como se define en [20] es un mecanismo de control retroalimentado ampliamente usado en sistemas de control industrial. El PID calcula un error como la diferencia entre el valor actual del sistema y el valor al que se desea llegar. En el cálculo del regulador PID intervienen tres parámetros distintos. Estos valores se pueden interpretar en función del tiempo. El proporcional P depende del error actual, el integral I depende de la suma de todos los errores pasados, y el derivativo D es la predicción de errores futuros, basándose en a tasa de cambio actual. El tipo de control que se implementa es el algoritmo PID tipo paralelo (ver figura 4-18).

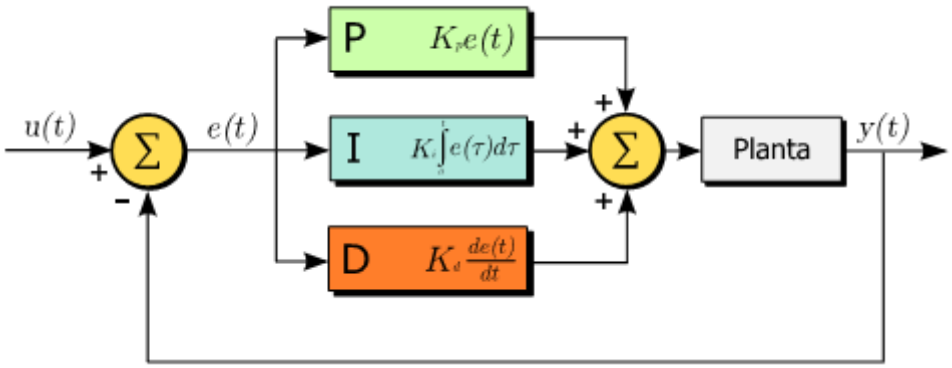


FIGURA 4-18. Diagrama de bloques de un PID tipo paralelo, tomado de [20].

Los términos proporcional, integral y derivativo son sumandos para hallar la salida del regulador PID, sabiendo que $\boldsymbol{u(t)}$ es tal salida, la ecuación que define al algoritmo es la siguiente:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t) \qquad (4 \cdot 22)$$

Dado que el dsPIC33F es el que ejecutará este algoritmo de la ecuación $(4 \cdot 22)$ y que solamente es capaz de procesar valores discretos y finitos, así que para poder implementar el algoritmo de control de un regulador PID, es necesario discretizar la ecuación del regulador PID como se menciona en [20], una vez discretizada se calcula para cada uno de los instantes de tiempo muestreados el valor de la acción de control. Para cada instante de tiempo se vuelve a calcular una nueva acción de control utilizando la nueva salida del sistema, esto permitirá al sistema avanzar hacia el estado deseado marcado por la referencia con cada nuevo cálculo. La ecuación discreta del regulador para poder ser implementada en un computador se puede expresar mediante la siguiente ecuación.

$$u(t) = K_p e_{(t)}(t) + K_i T_s \sum_{k=0}^{t} e_k + K_d \frac{e_{(t)} - e_{(t-1)}}{T_s} \qquad (4 \cdot 23)$$

Donde $\boldsymbol{e_{(t)}}$ es el error de la respuesta del sistema en el instante $\boldsymbol{t}$, $\boldsymbol{T_s}$ es periodo de muestreo de la señal y $\boldsymbol{K_p, K_i}$ y $\boldsymbol{K_d}$ son la ganancia proporcional, integral y derivativa del regulador, respectivamente.

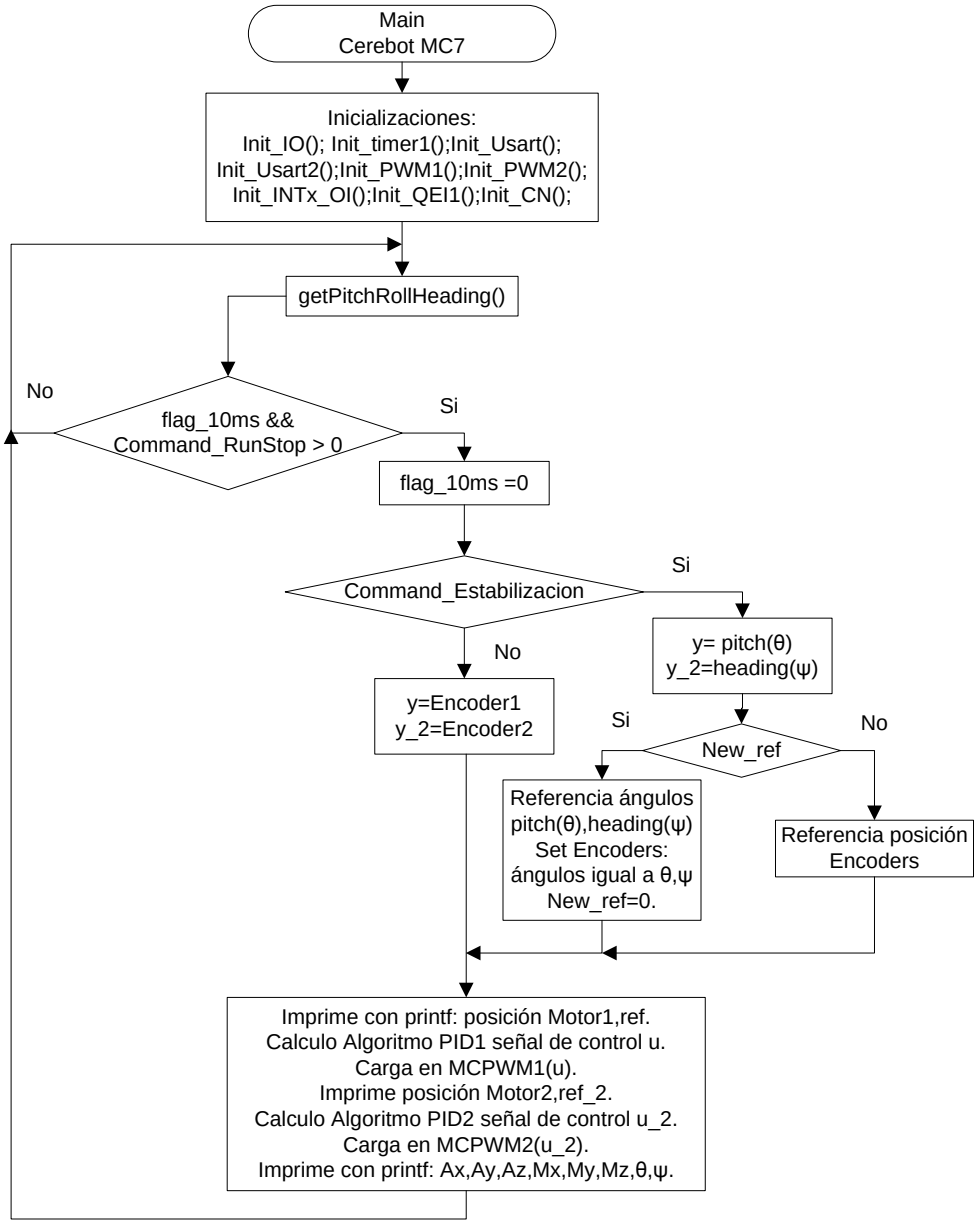


FIGURA 4-19. Diagrama de flujo programa principal, fuente el autor.

CAPITULO 5.

PRUEBAS Y RESULTADOS EXPERIMENTALES

Este capítulo muestra las pruebas realizadas para recolectar los datos experimentales a los procedimientos descritos en el capítulo 4. Se inicia mostrando los resultados de calibración del magnetómetro del sensor LSM303DLH seguidamente las gráficas de respuesta el filtro promediador implementado todo esto con la plataforma Seeeduino.
Luego se mostrarán los resultados de la validación del modelo del motor DC, esto con la herramienta MATLAB®. Los demás resultados obtenidos se visualizan con la GUI desarrollada para el proyecto que facilita la adquisición de datos del control PID implementado con realimentación de los encoders y el control de estabilización de posición del ángulo de elevación(θ) a una perturbación conocida, con la ayuda del mecanismo construido para este fin.

5.1. RESULTADOS DE IMPLEMENTACION EN SEEEDUINO

Ya ubicado el sensor en la plataforma Altazimut que se muestra en la figura 4-13 se verifica la generación de distorsión tipo hard-iron, también por intermedio de una prueba de seguimiento de rampa se mostrará el desempeño del promediador de datos como filtro pasabajo.

5.1.1. Resultados Calibración Magnetómetro del LSM303DLHC

Como dice el procedimiento de calibración descrito en [18] y explicado en el capítulo 4. Se grafican los datos en bruto o raw del magnetómetro por intermedio de MATLAB®, al trazar una esfera imaginaria se tienen los datos graficados de la figura 5-1 donde la línea azul son los datos de los tres ejes M_x,M_y y M_z sin procesamiento, las líneas rojas, verdes y amarillas corresponden a una circunferencia imaginaria en los planos xy, xz y yz respectivamente.

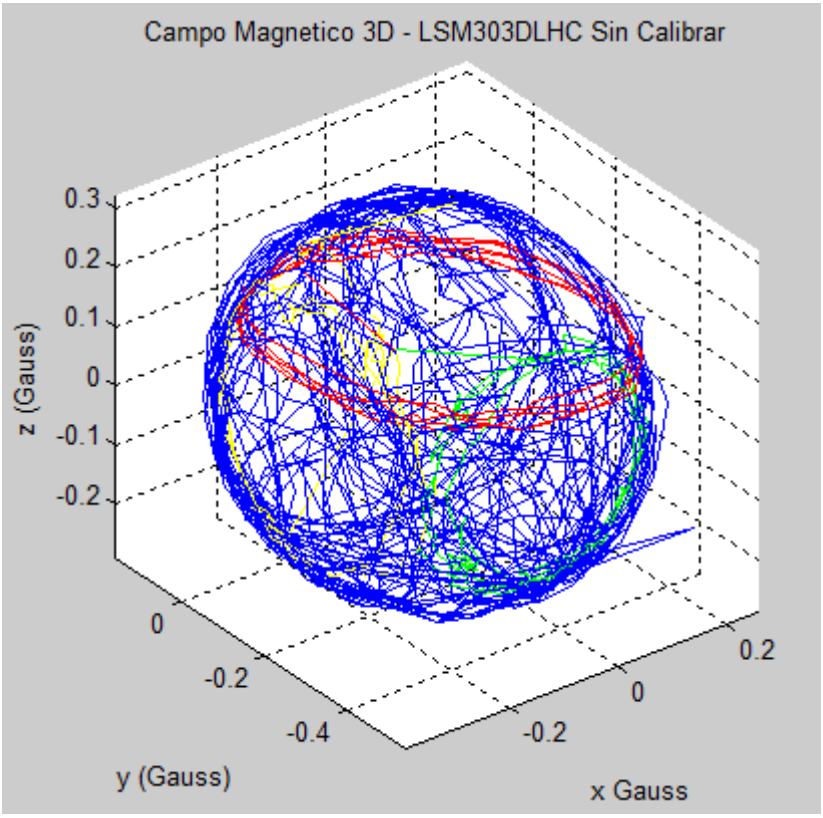
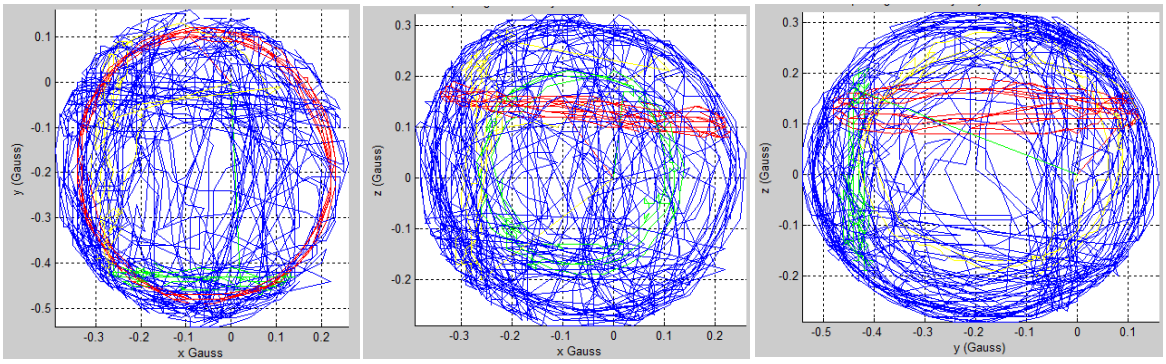


FIGURA 5-1. Lectura 3D magnetómetro sin calibración, fuente el autor.

En la figura 5-2 se evidencia el error de offset que hay en cada circulo desalineado con centro diferente en (0,0) por lo que debe realizarse calibración del sensor dado que se identifica que la distorsión es hard-iron por lo explicado en el capítulo 4.



(a). Ejes xy (b). Ejes xz (c). Ejes zy

FIGURA 5-2. Lectura 2D magnetómetro sin calibración, fuente el autor.

Utilizando el programa de ejemplo incluido en la librería Arduino (ver ANEXO B.1.) para la calibración del magnetómetro del LSM303DLHC para corrección de distorsión hard-iron se obtienen los parámetros offset mostrados en la figura 5-3.

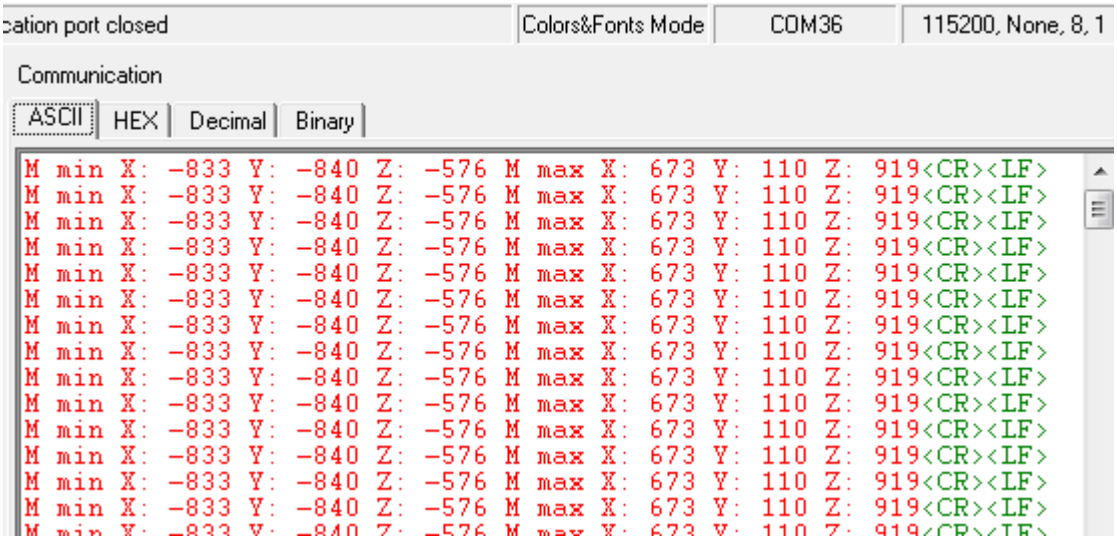


FIGURA 5-3. Parámetros de calibración magnetómetro, fuente el autor.

Después de utilizar los valores mostrados en la figura anterior se vuelven a trazar los valores brutos o raw (ver figura 5-4).

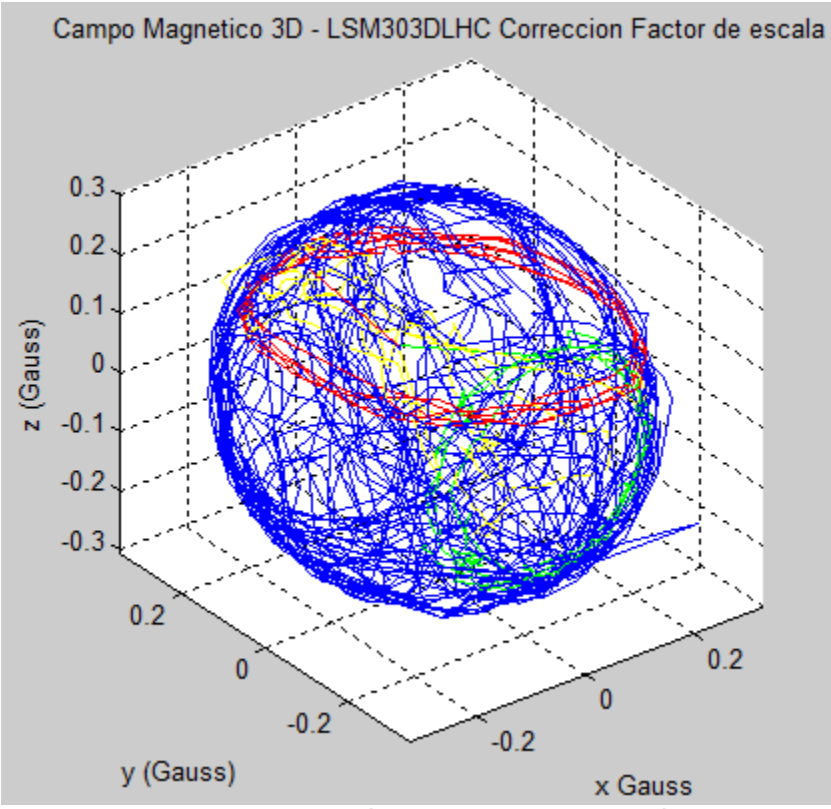


FIGURA 5-4. Lectura 3D magnetómetro con la calibración, fuente el autor.

En las figuras 5-5 a 5-7 se muestra la corrección realizada por distorsión hard-iron en cada plano 2D y con los parámetros hallados al ejecutar la prueba de la esfera imaginaria. Así se obtienen los datos para el magnetómetro Mx1, My1 Mz1 mencionados en el capítulo 4 el ítem 4.3.7.3 con el que se puede calcular el ángulo de giro(θ).

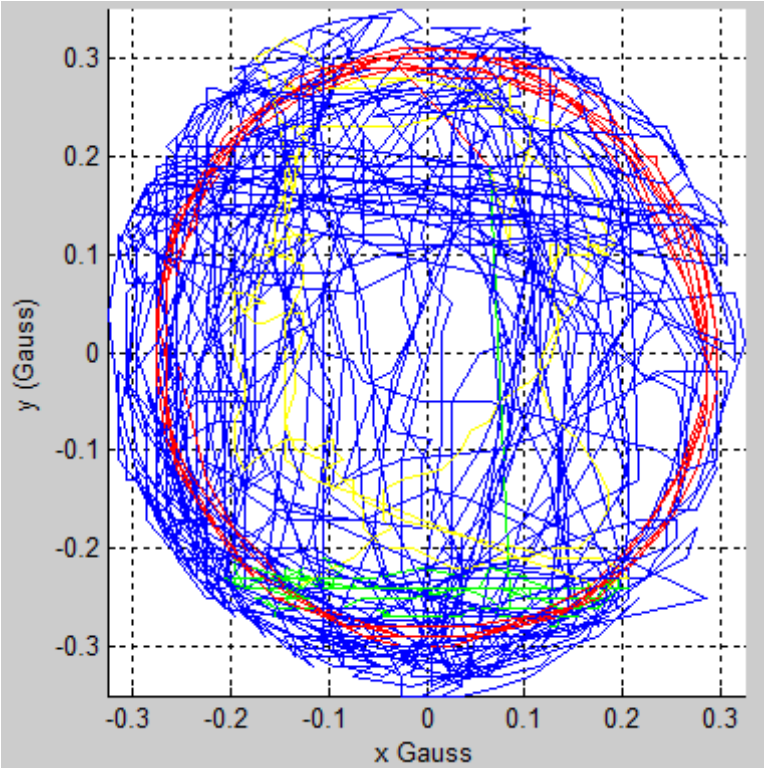


FIGURA 5-5. Lectura 2D magnetómetro con la calibración ejes xy, fuente el autor.

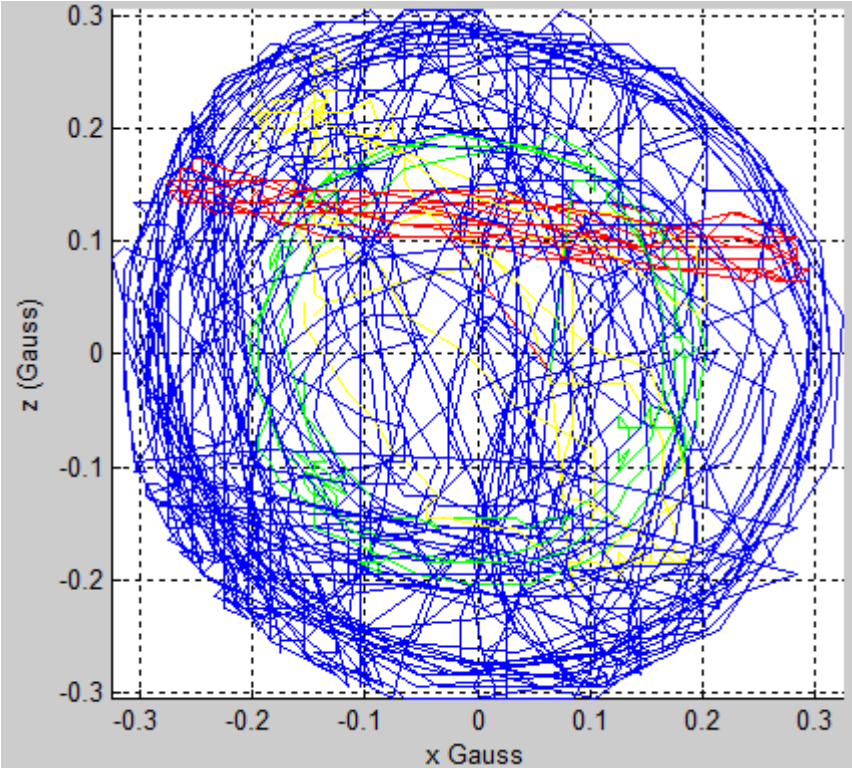


FIGURA 5-6. Lectura 2D magnetómetro con la calibración ejes xz, fuente el autor.

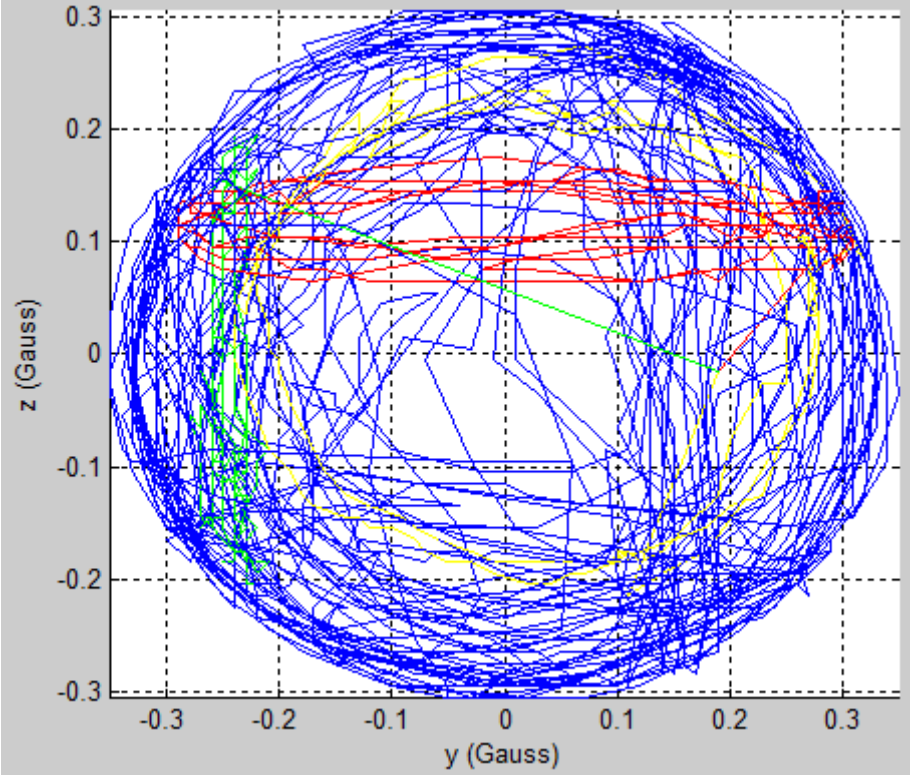


FIGURA 5-7. Lectura 2D magnetómetro con la calibración ejes zy, fuente el autor.

5.1.2. Resultados promediado de datos del LSM303DLHC

Para mostrar el suavizado de los datos del acelerómetro y magnetómetro se diseño una prueba de seguimiento de entrada rampa con realimentación de los encoders para cada eje de la plataforma Altazimut realizando combinaciones de movimientos que son ángulos resumidos en la tabla 5-1.

Prueba #	Elevación(θ)	Giro(ψ)
1	0°	0°
2	0°	90°
3	90°	0°
4	90°	90°

Tabla 5-1. Pruebas datos Raw LSM303DLHC, fuente el autor.

Para todas las pruebas se toman datos del magnetómetro ya calibrado. Y según la tabla anterior se enumeran las pruebas.

Prueba #1:

Se realiza la adquisición de datos del sensor sin mover ninguno de los ejes obteniendo en ambos sensores mediciones con ruido (ver figura 5-8).

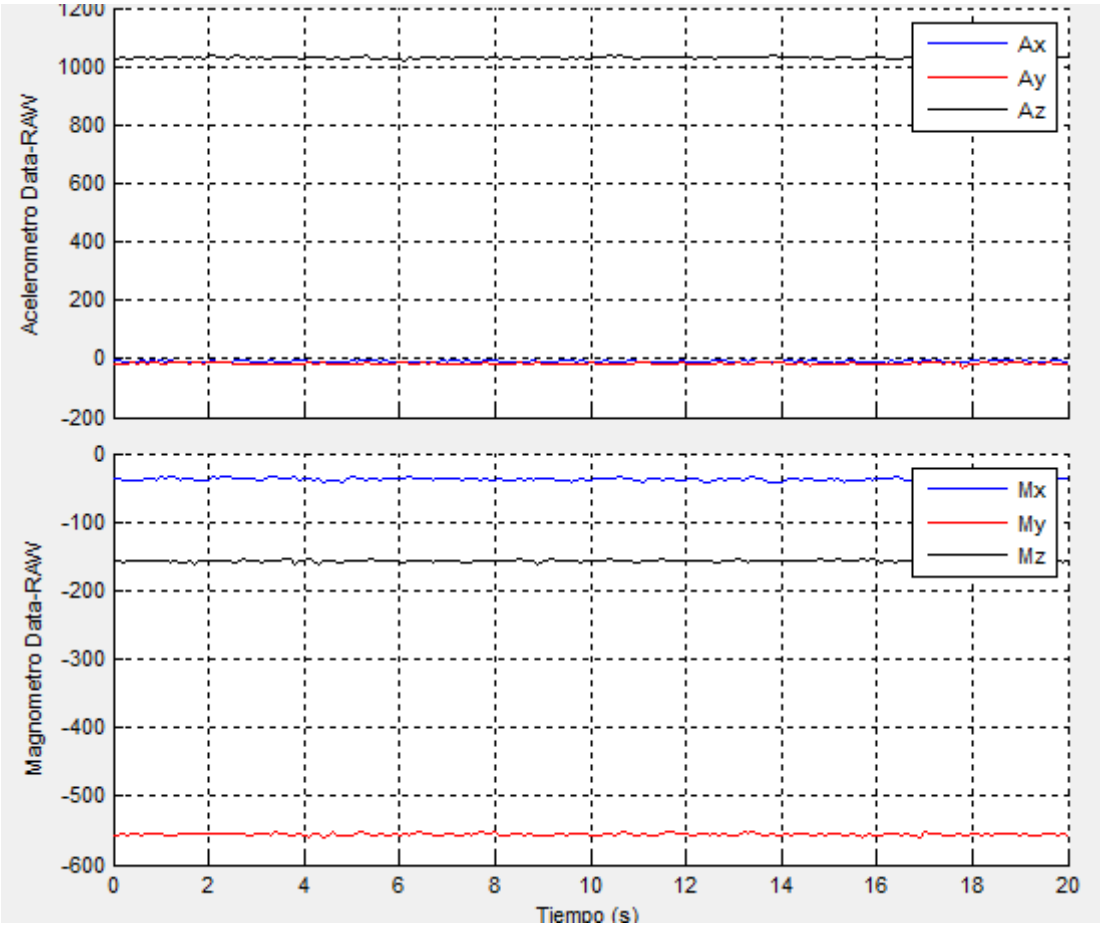


FIGURA 5-8. Datos raw sensor LSM303DLHC prueba#1, fuente el autor.

Prueba #2:

Se realiza la adquisición de datos del sensor moviendo 90° el ángulo de giro(ψ) y sin mover el eje de elevación(θ) obteniendo en ambos sensores mediciones con ruido (ver figura 5-9).

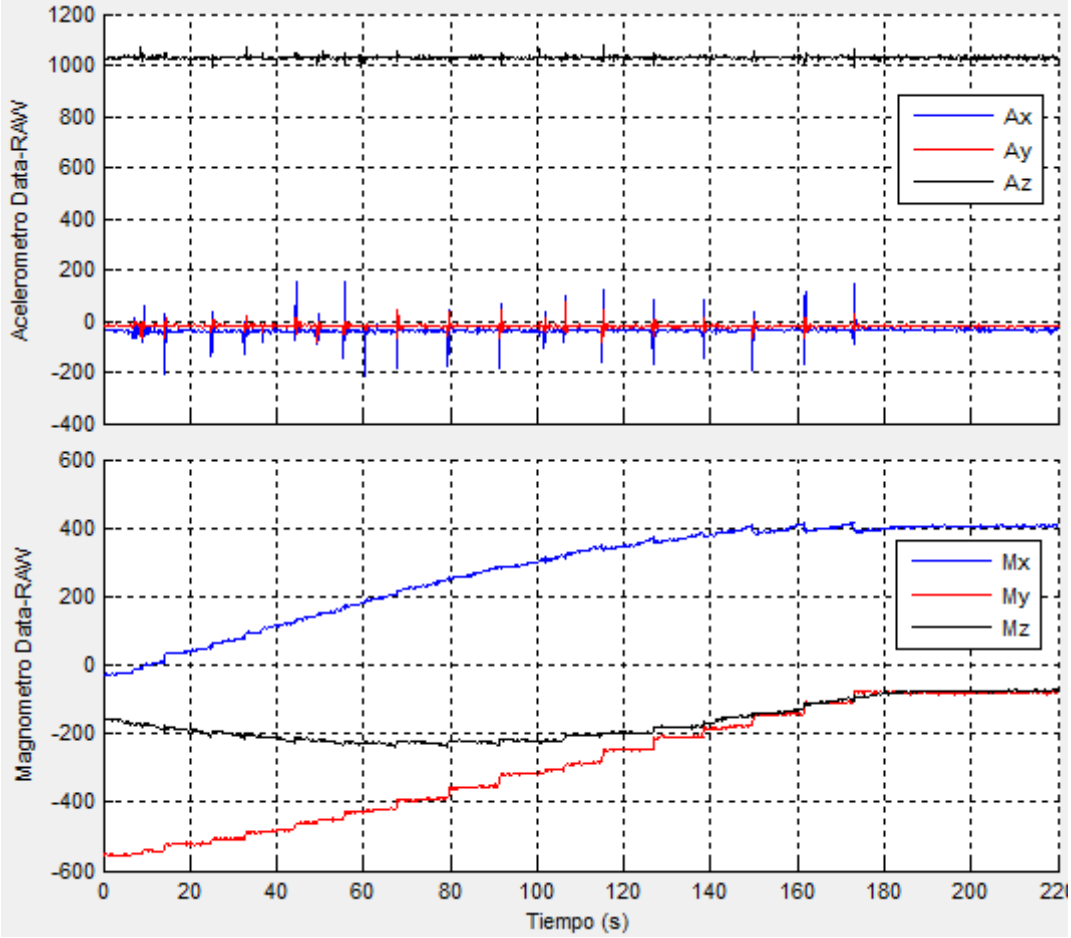


FIGURA 5-9. Datos raw sensor LSM303DLHC prueba#2, fuente el autor.

La rampa de seguimiento para la prueba es la mostrada en la figura 5-10 del eje de giro(ψ), donde la línea roja es la rampa de seguimiento, azul es la posición realimentada por el encoder y la línea negra muestra el ángulo calculado con los datos del sensor LSM303DLHC que se validaran más adelante.

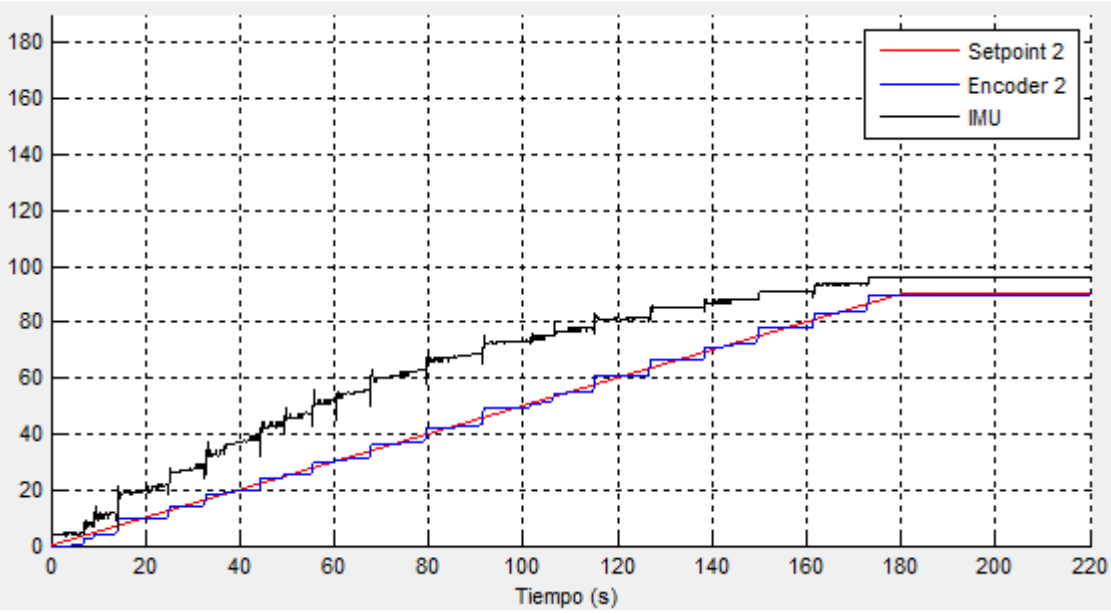


FIGURA 5-10. Rampa de seguimiento prueba#2, fuente el autor.

Prueba #3:

Se realiza la adquisición de datos del sensor moviendo 90° el ángulo de elevación(θ) y sin mover el eje de giro(ψ) obteniendo en ambos sensores mediciones con ruido (ver figura 5-12). La rampa de seguimiento para la prueba es la mostrada en la figura 5-11 del eje de elevación(θ).

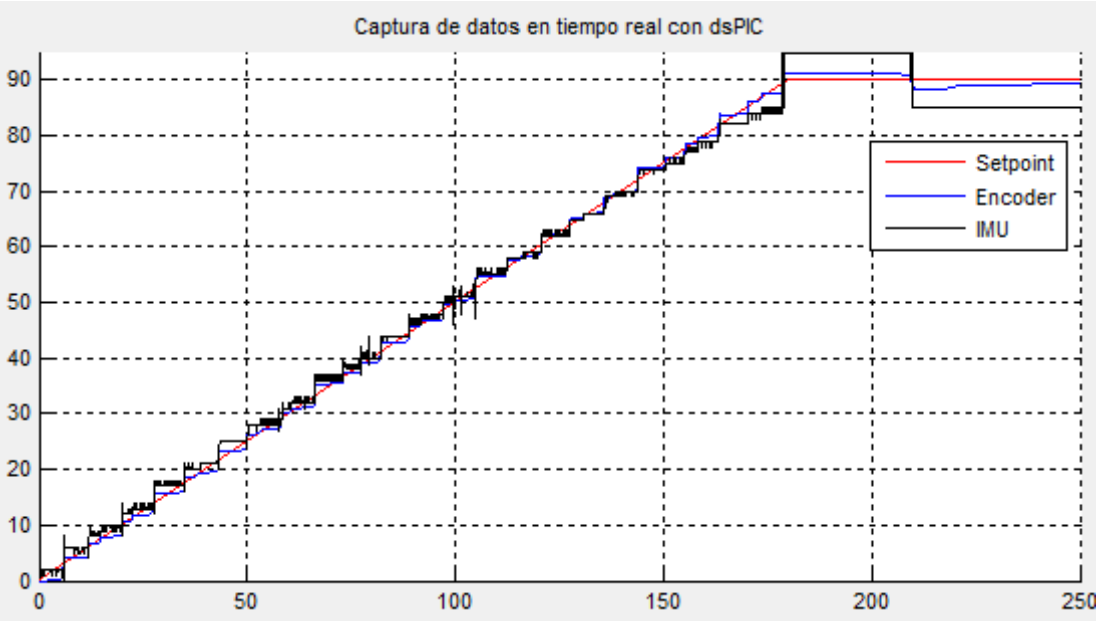


FIGURA 5-11. Rampa de seguimiento prueba#3, fuente el autor.

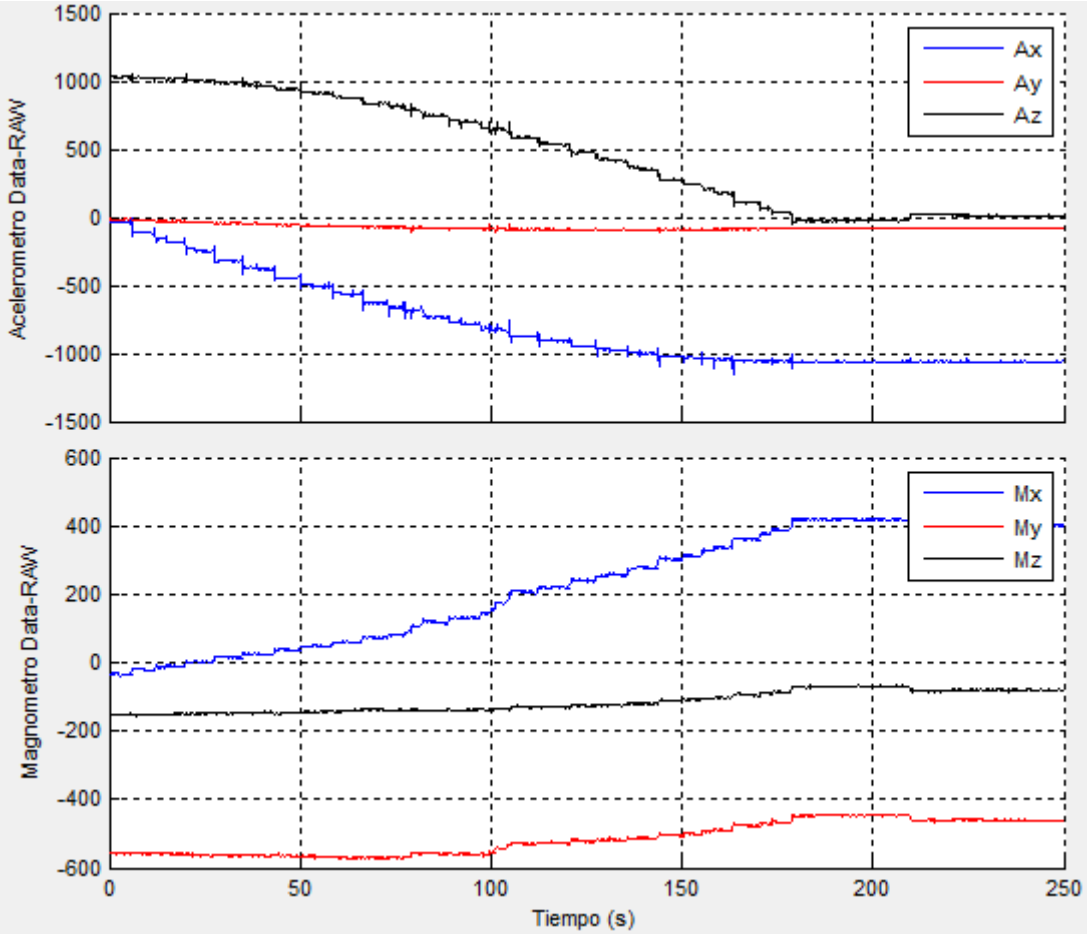


FIGURA 5-12. Datos raw sensor LSM303DLHC prueba#3, fuente el autor.

Prueba #4:

Se realiza la adquisición de datos del sensor moviendo 90° el ángulo de giro(ψ) y el eje de elevación(θ) obteniendo en ambos sensores mediciones con ruido (ver figura 5-13).

Las rampas de seguimiento para la prueba son las mismas utilizadas en las pruebas #2 y #3 mostradas en la figura 5-10 y 5-11 de los ejes de giro(ψ) y elevación(θ).

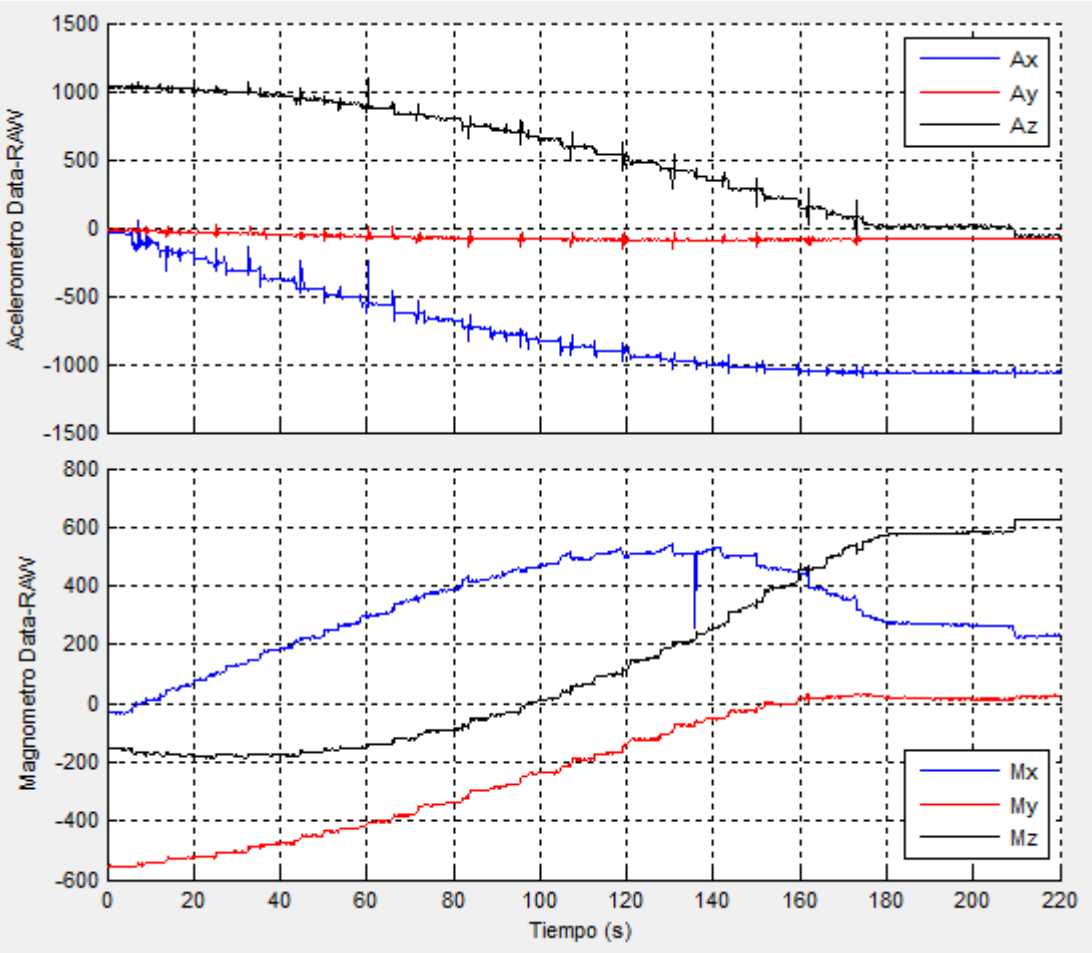


FIGURA 5-13. Datos raw sensor LSM303DLHC prueba#4, fuente el autor.

Entonces utilizando la ecuación $(4 \cdot 1)$ se realiza el promedio de los datos para conformar el filtro pasabajo se realizan las mismas pruebas planteadas en la tabla 5-1. Excluyendo la prueba número uno, tenemos las graficas 5-14 a 5-16 registran los datos de los experimentos denominados prueba#2, prueba#3 y prueba#4. Entonces comparando los datos del sensor sin filtrar la señal se observa que es suavizada al promediar los últimos 10 datos medidos el ruido es disminuido en gran parte, cabe anotar que la respuesta del acelerómetro en estado estacionario es mucho más precisa que cuando se encuentra en movimiento.

Prueba #2 con datos filtrados:

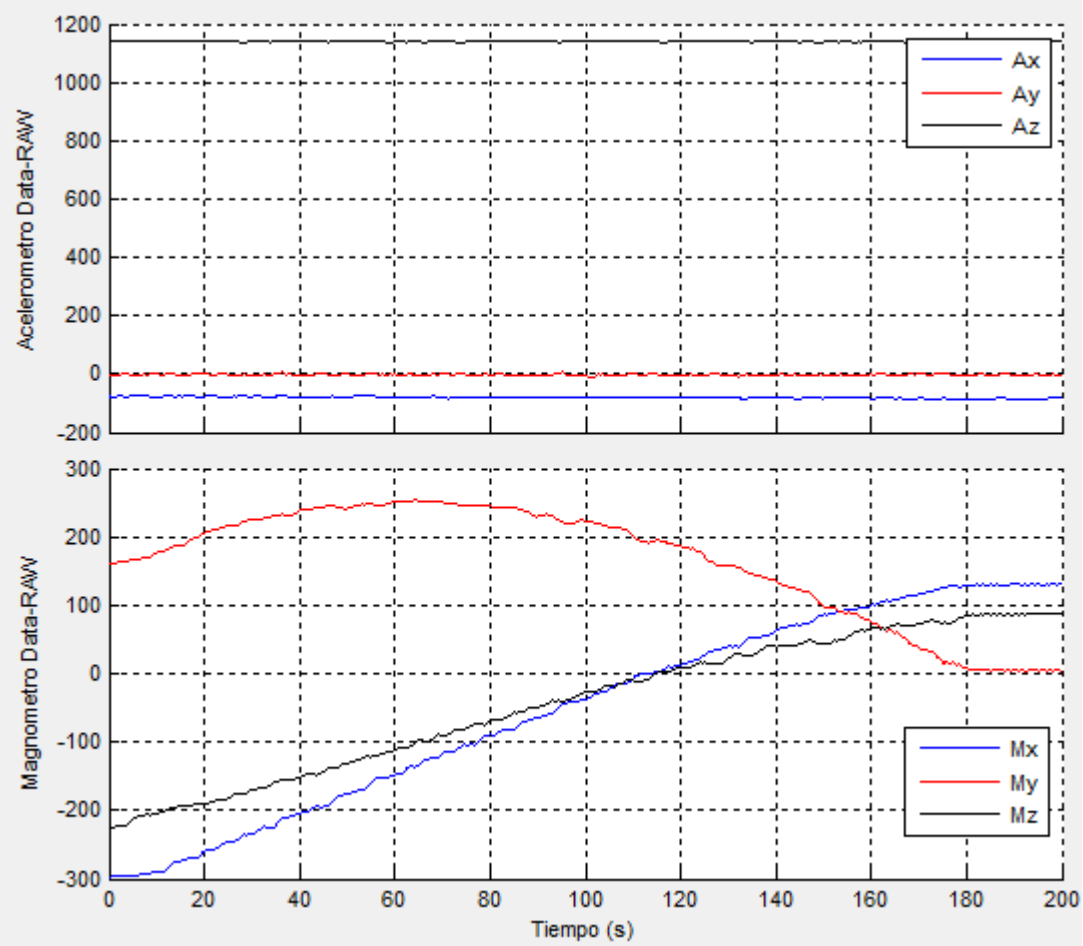


FIGURA 5-14. Datos filtrados sensor LSM303DLHC prueba#2, fuente el autor.

Prueba #3 con datos filtrados:

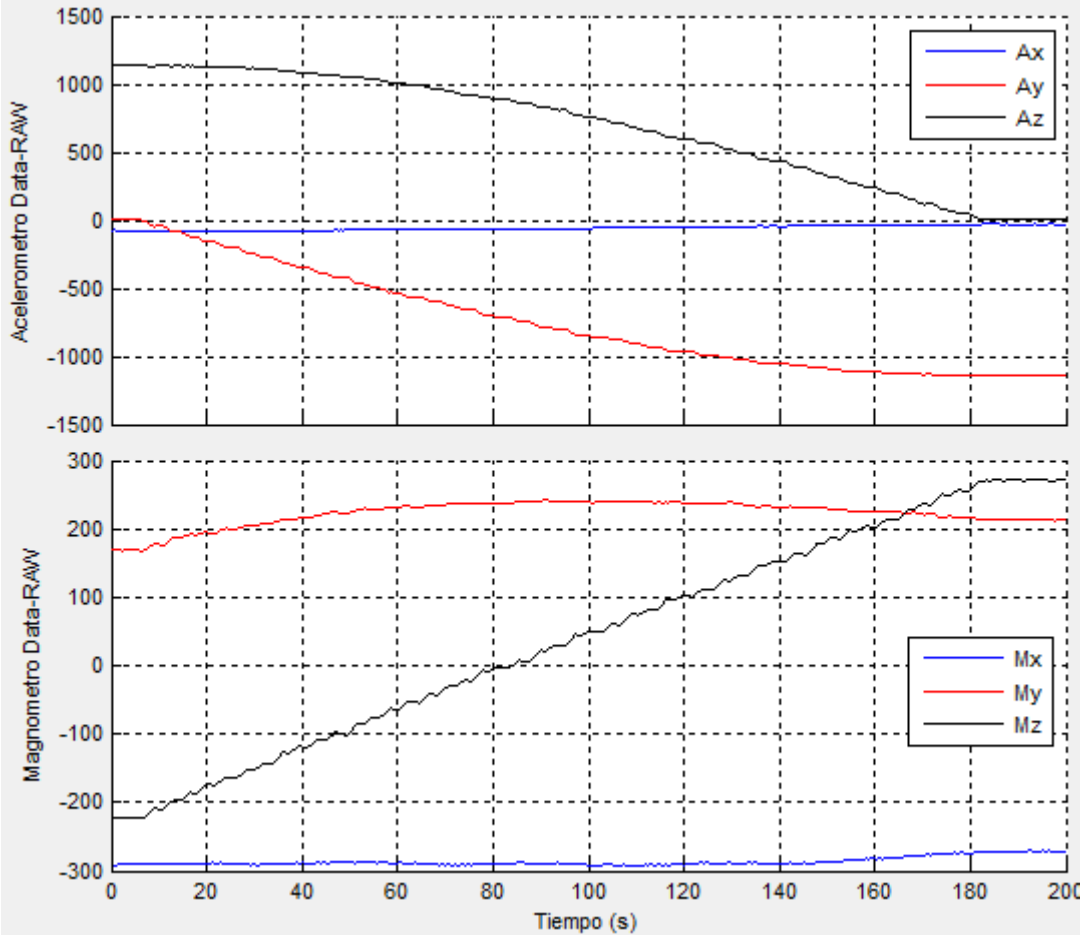


FIGURA 5-15. Datos filtrados sensor LSM303DLHC prueba#3, fuente el autor.

Prueba #4 con datos filtrados:

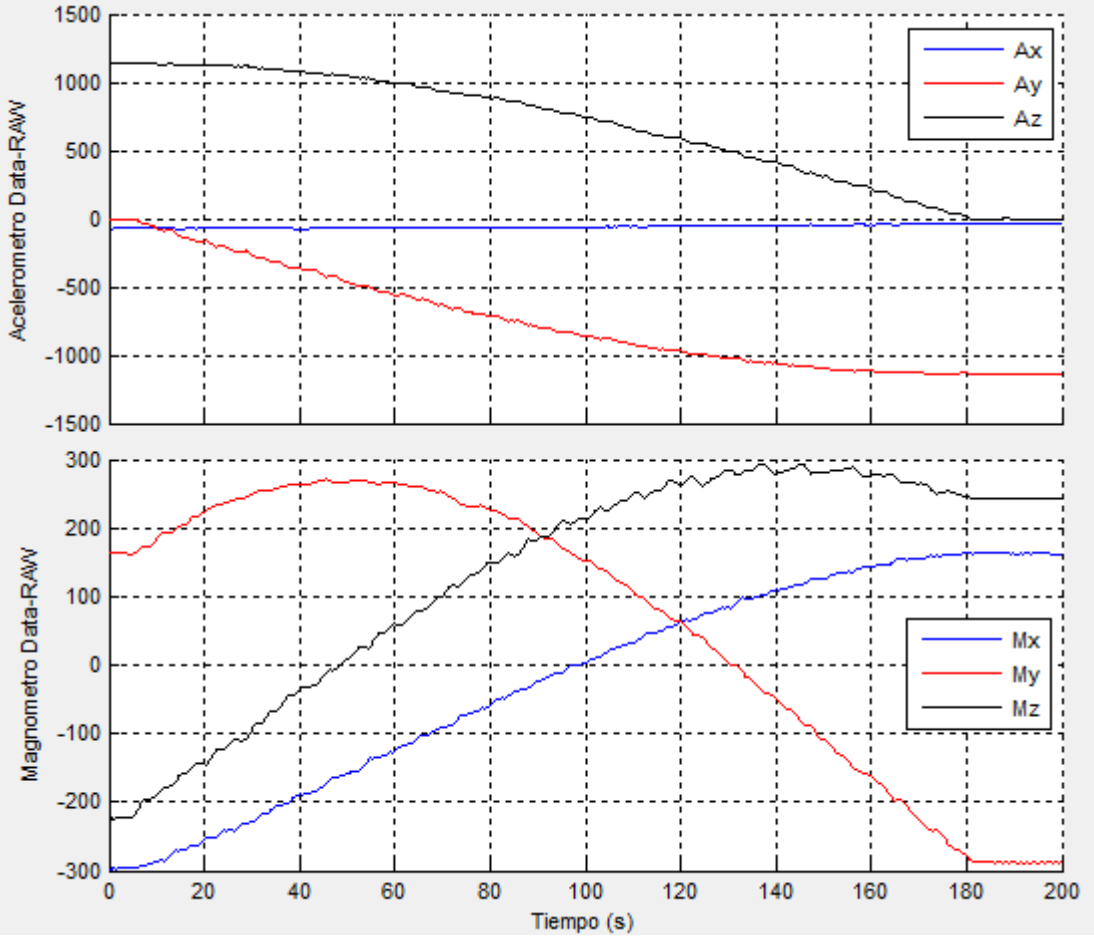


FIGURA 5-16. Datos filtrados sensor LSM303DLHC prueba#4, fuente el autor.

5.2. RESULTADOS DE IMPLEMENTACION EN CEREBOT MC7™

Utilizando la misma terminal serial para lectura de los datos enviados por el dsPIC33FJ128MC706A se calibra la posición entregada por cada uno de los encoders. Con la herramienta MATLAB® se validara el modelo del motor DC obtenido en el capítulo 4, su posterior sintonización con la misma herramienta y las pruebas de los controles de posición y estabilización.

5.2.1. Calibración datos de Encoders

Para el encoder del motor#2 del ángulo de giro(ψ), al rotarlo una vuelta completa de 360° los pulsos recibidos en resolución de 2x fueron 3714, la verificación visual se realizo con un transportador y una aguja que marca el ángulo como se muestra en la figura 5-17. Entonces experimentalmente se halla la relación del reductor de la siguiente manera:

$$Relacion\_Reductor2 = \frac{Reductor\ p.p.r.}{Encoder\ p.p.r.} = \frac{3714}{200} = 18.57 \qquad (5 \cdot 1)$$

FIGURA 5-17. Numero de pulsos por vuelta Encoder 2, fuente el autor.

Luego para el encoder del motor#1 del ángulo de elevación(θ) al girarlo una vuelta completa de 360° los pulsos recibidos en resolución de 4x fueron 7424 y la verificación también visual se realizo con un transportador y una aguja que marca el ángulo como se muestra en la figura 5-18. De la ecuación se halla la relación de reducción:

$$Relacion\_Reductor1 = \frac{7424}{400} = 18.56$$

Entonces de esta manera se ajustaron los pulsos máximos por vuelta del eje del reductor para cada motor (ver ANEXO C).

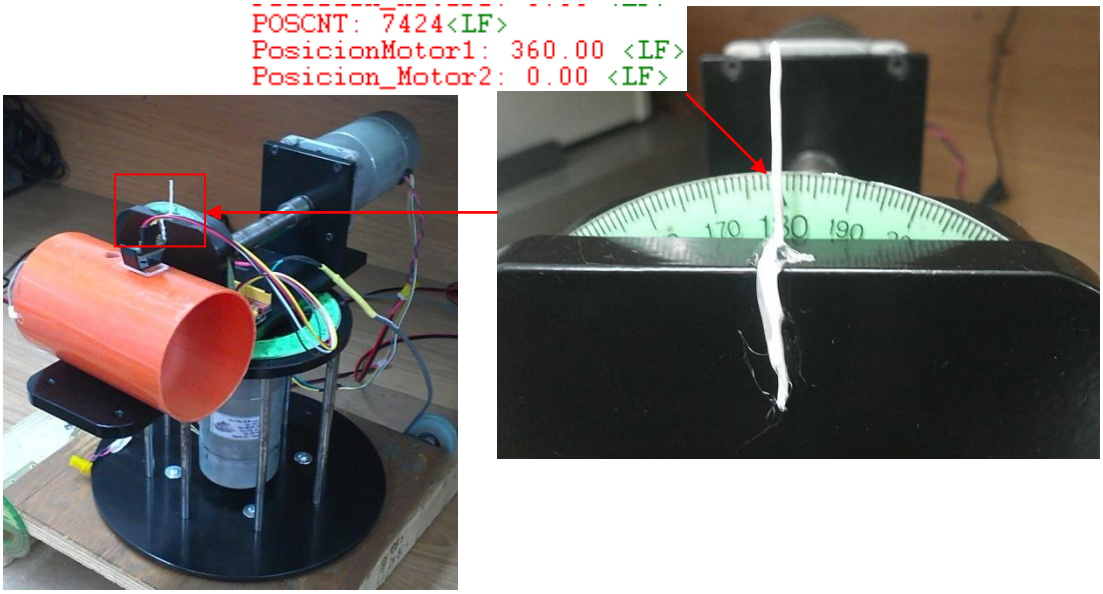


FIGURA 5-18. Numero de pulsos por vuelta Encoder 1, fuente el autor.

5.2.2. Validación modelo Motor DC

Para dicha validación del modelo analítico planteado en $(3 \cdot 12)$ con la GUI implementada en entorno MATLAB®, como se menciona en [21] para verificar la validez del modelo, la predicción acerca del funcionamiento obtenida al resolver las ecuaciones del modelo, la predicción acerca del funcionamiento obtenida al resolver las ecuaciones del modelo, se compara con los resultados experimentales. Si los resultados experimentales se alejan de la predicción en forma considerable, debe modificarse el modelo. Utilizando la misma estructura del sistema de control mostrado en la figura 3-5 donde el controlador es un PID se realiza pruebas en lazo cerrado para comparar el modelo y la planta real. Aplicando transformada de laplace en la ecuación $(4 \cdot 22)$ la representación en modo de función de transferencia del controlador PID la cual se muestra a continuación:

$$\mathrm{PID} = \frac{\mathrm{K_d s^2 + K_p s + K_i}}{\mathrm{s}} \qquad (5 \cdot 2)$$

Donde:

K_p: Constante Proporcional (Ajusta la ganancia)

K_i: Constante Integral (Ajusta el tiempo de establecimiento)

K_d: Constante Derivativa (Ajusta el sobrepaso)

Entonces utilizando ese sistema de control en lazo cerrado para el ángulo de elevación(θ) se obtiene la respuesta paso a un escalo de 45° y 90° para verificar los parámetros característicos como tiempo de subida, % de sobrepaso, tiempo de asentamiento y valor final en estado estable se hallan los errores de la simulación respecto a los valores de la planta real.

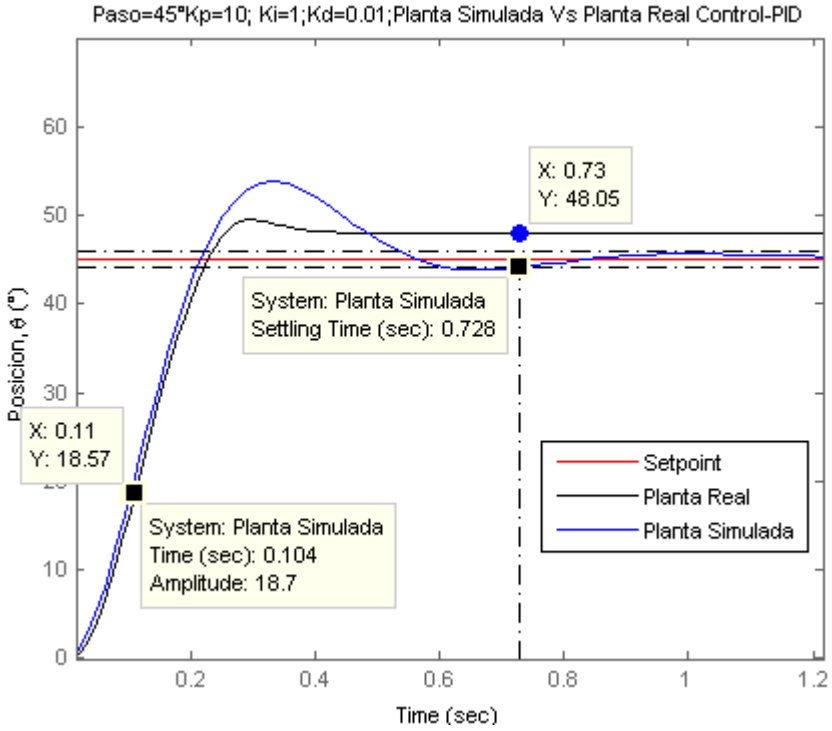


FIGURA 5-19. Planta real Vs Simulada con paso=45°, fuente el autor.

La tabla 5-2 resumen los valores de los principales parámetros del modelo con entrada paso de 45° de la figura 5-19.

Planta	Tiempo de Subida (s)	% Sobrepaso	Tiempo de Asentamiento(s)	Valor Final(°)
Real	0,19	7,14	5,03	46,21
Simulada	0,188	19,6	0,728	45

Tabla 5-2. Planta Simulada paso=45°, fuente el autor.

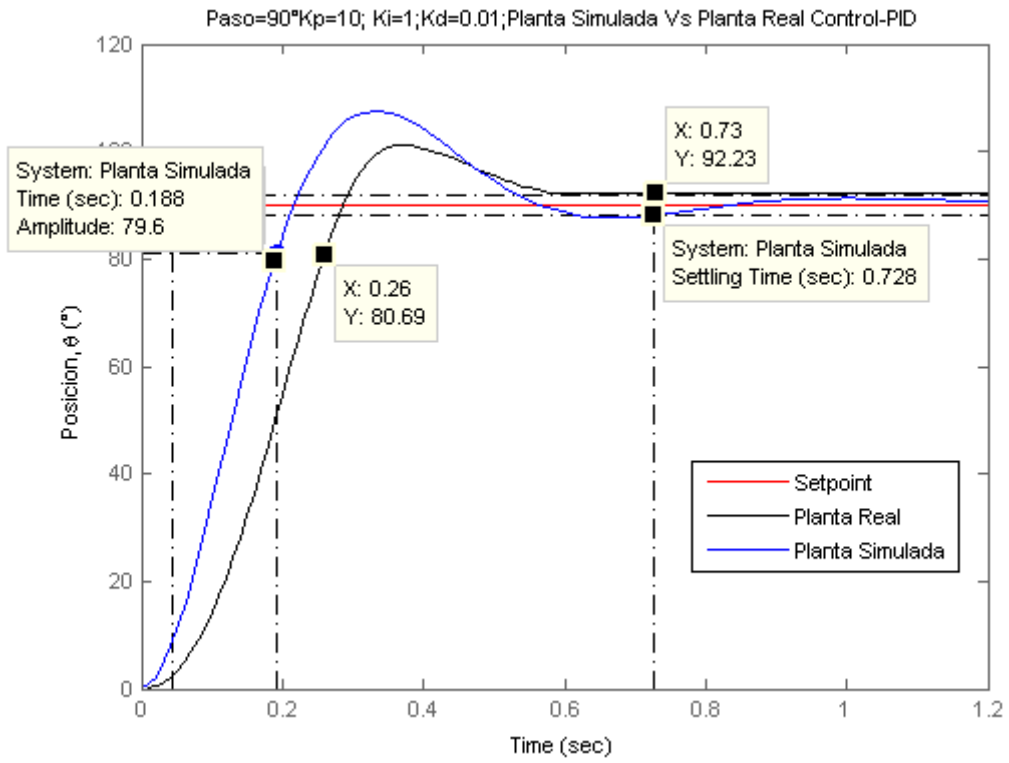


FIGURA 5-20. Planta real Vs Simulada con paso=90°, fuente el autor.

La tabla 5-3 resume el error porcentual de los principales parámetros del modelo con entrada paso de 90° de la figura 5-20.

Planta	Tiempo de Subida (s)	% Sobrepaso	Tiempo de Asentamiento(s)	Valor Final(°)
Real	0,2	9,9	0,61	92,23
Simulada	0,188	19,6	0,728	90

Tabla 5-3. Planta Simulada paso=90°, fuente el autor.

Después de realizar varias iteraciones del modelo simulado Vs la planta se ajustan los parámetros utilizados para caracterizar el motoreductor, entonces la función de transferencia de $G_{MOTOREDUCTOR}(s)$ utilizando los siguientes parámetros para el motoreductor: $\mathrm{R_a} = 2.36\,, \mathrm{L_a} = 2.45604\mathrm{mH}\,, \mathrm{k_b} = 0.8082\,, \mathrm{k_m} = 0.998201\,, \mathrm{J} = 0.0363\,,$ $\mathrm{B} = 0.1048$ reemplazando en la ecuación $(3 \cdot 6)$ obtenemos:

$$G_{MOTOREDUCTOR}(s) = \frac{\theta(s)}{V_a(s)} = \frac{0.9982}{8.923 * 10^{-5} \cdot s^2 + 0.08964 \cdot s + 1.065} \qquad (5 \cdot 1)$$

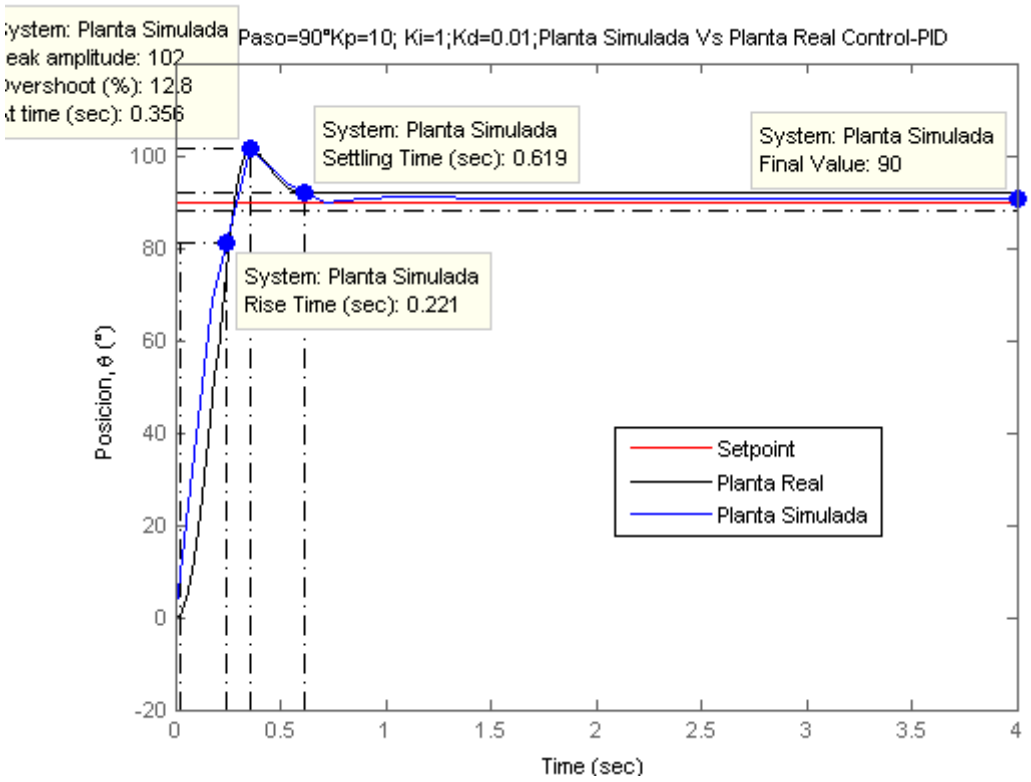


FIGURA 5-21. Planta Real Vs Simulada (ajustada) con paso=90°, fuente el autor.

La tabla 5-4 resume el error porcentual de los principales parámetros del modelo ajustado con entrada paso de 90° de la figura 5-21.

Planta	Tiempo de Subida (s)	% Sobrepaso	Tiempo de Asentamiento(s)	Valor Final(°)
Real	0,18	10,19	0,61	91,75
Simulada	0,221	12,8	0,619	90

Tabla 5-4. Planta Simulada(ajustada) paso=90°, fuente el autor.

5.2.3. Pruebas de Posición Sensor LSM303DLHC

Con la GUI se abstraen los datos de posición para los ángulos Giro(ψ) y Elevación(θ), comparándolos con los valores del Encoder.

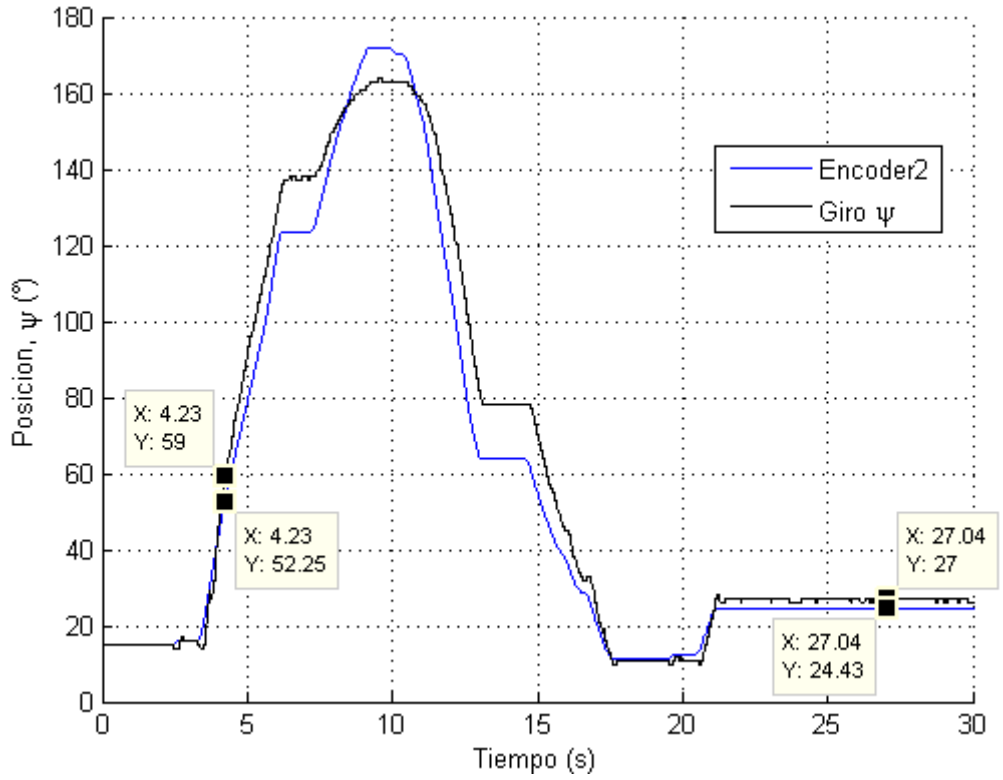


FIGURA 5-22. Medición Angulo de Giro(ψ), fuente el autor.

La tabla 5-5 resume el error porcentual de las mediciones del ángulo calculado con el sensor LSM303DLHC.

Sensor	Angulo de Giro(ψ)	Angulo de Giro(ψ)
Encoder	52,25	24,43
LSM303	59	27

Tabla 5-5. Medición Angulo de Giro(ψ), fuente el autor.

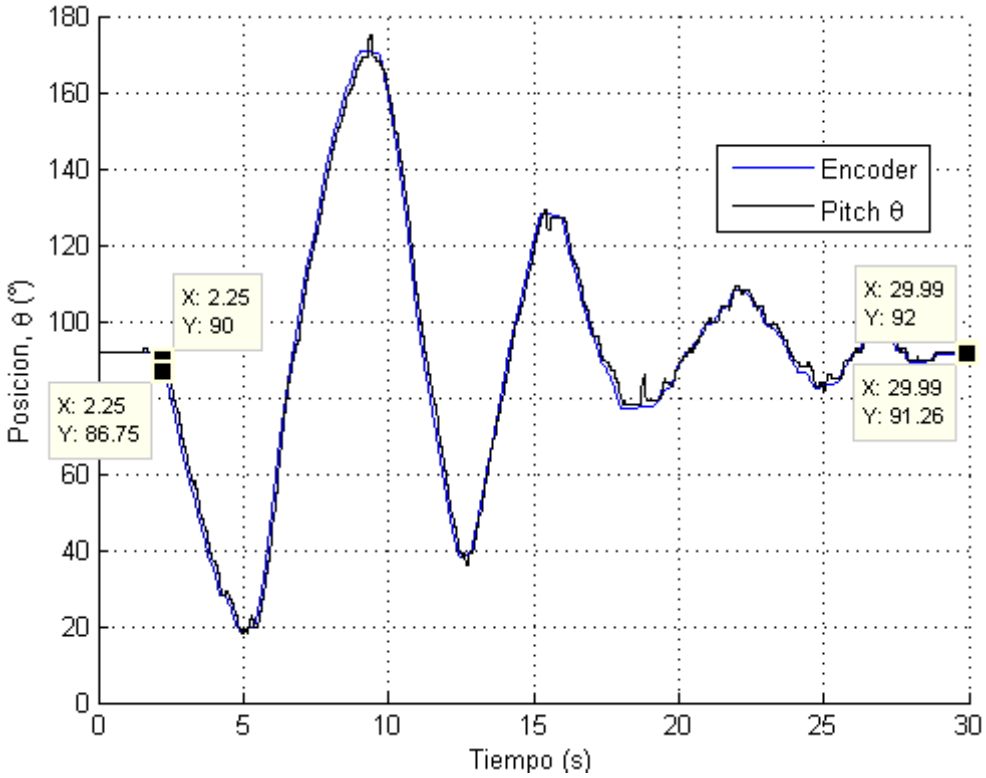


FIGURA 5-23. Medición Angulo de Elevación(θ), fuente el autor.

La tabla 5-6 resume el error porcentual de las mediciones del ángulo calculado con el sensor LSM303DLHC.

Sensor	Angulo de Elevación(θ)	Angulo de Elevación(θ
Encoder	86,75	91,26
LSM303	90	92

Tabla 5-6. Medición Angulo de Elevación(θ), fuente el autor.

Esta medición es mucho más exacta debido que no se tienen distorsiones relacionadas al campo magnético, se tienen distorsiones por ruido en la medición.

5.2.4. Pruebas Control de estabilización PID

Para la prueba del control de estabilización se utilizo el mecanismo mostrado en la figura 5-24 para la generación de un disturbio conocido de amplitud aproximadamente de +/-10° en el ángulo de Elevación(θ).

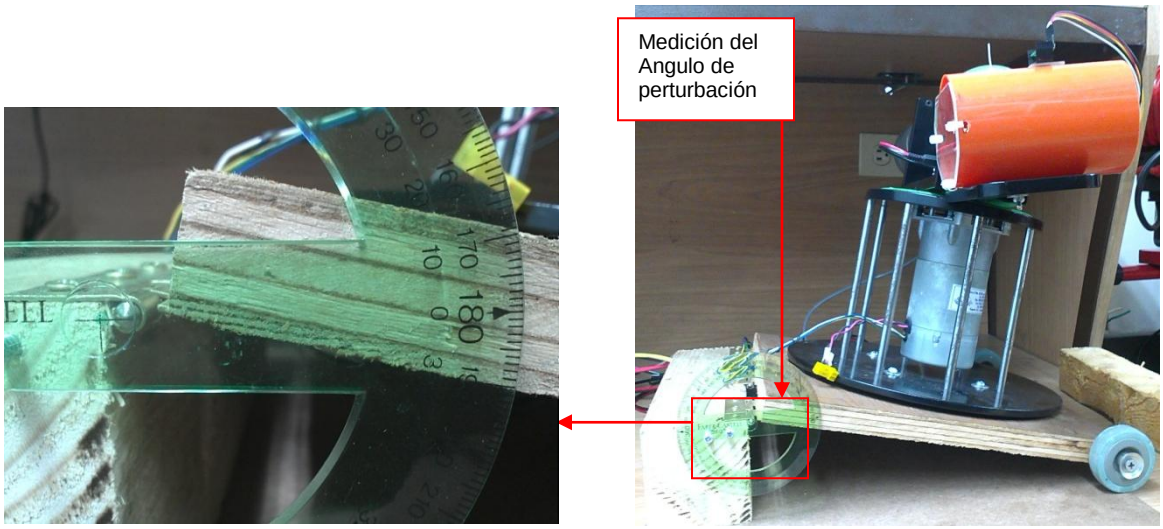


FIGURA 5-24. Plataforma de Disturbio en ángulo de Elevación(θ), fuente el autor.

Debido a que la respuesta de los sensores en los transitorios de movimiento no son tan exactos por las distorsiones y ruidos en las señales del sensor LSM303DLHC se realizo un ajuste en los parámetros del control PID para hacer que la respuesta sea más lenta y evitar desestabilizar la planta controlada. Entonces cuando se baja la plataforma que genera el disturbio se tendría un ángulo de inclinación -10° mostrado en la figura 5-24, cuando la plataforma está alineada el ángulo observado en la escala del transportador es 0° o 180° (ver figura 5-25).

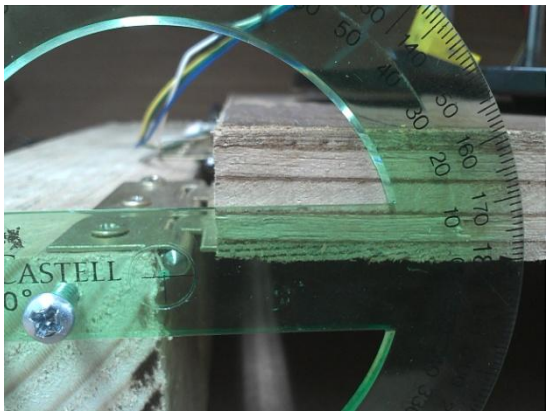


FIGURA 5-25. Angulo de Disturbio Alineado a 0°, fuente el autor.

Prueba #1:

La prueba consiste en mover el mecanismo manualmente, donde para dejarlo a nivel se mantiene fija la altura del extremo de la bisagra y la otra es puesta sobre un soporte de la misma altura alineando la tabla de madera como se muestra en la figura 5-25 manteniendola en la misma posición hasta tomar los datos generados desde la GUI, esto genera una pertubación de 10° aproximadamente sobre el plano horizontal. Utilizando los parámetros PID: K_p=4.53456, K_i=1.52001, K_d=0.03045

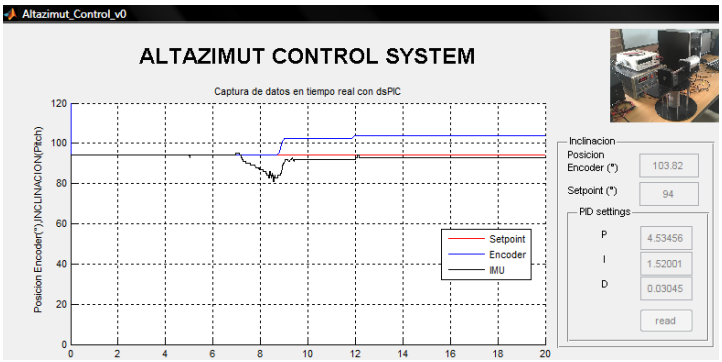


FIGURA 5-26. Prueba Disturbio -10°, fuente el autor.

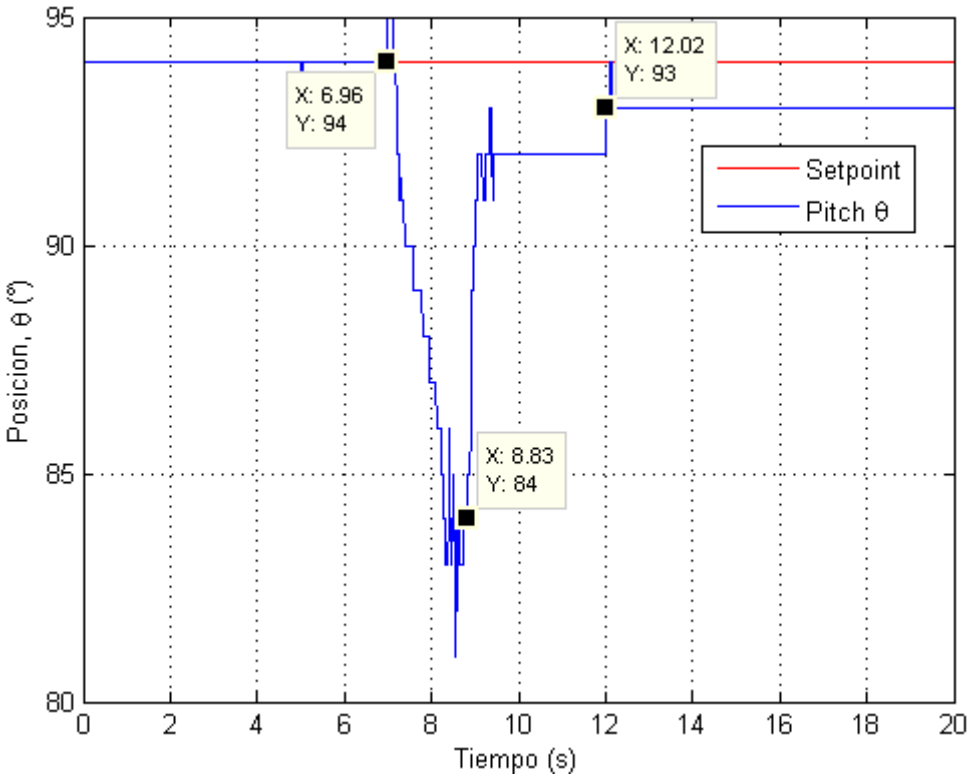


FIGURA 5-27. Zoom Prueba Disturbio -10°, fuente el autor.

La prueba #1, después de generar el disturbio la respuesta del control de estabilización es de 5.06 segundos. Este tiempo es bastante grande para realizar correcciones a disturbios externos que sean generados con frecuencias mayores a 0.2Hz.

Se realiza un ajuste iterativo para que la respuesta del control realice correcciones más rápidas sin generar inestabilidad en la planta, y se tienen los siguientes parámetros PID: K_p=4.53456, K_i=4.8201, K_d=0.3045.

Prueba #2:

La prueba es similar a la prueba #1 pero en esta la tabla de madera es movida manualmente hacia abajo quitando el soporte que la mantiene alineada sobre la horizontal como se muestra en la figura 5-24, manteniendola inclinada hasta tomar los datos generados desde la GUI, esto genera una pertubación de -10° aproximadamente sobre el plano horizontal.

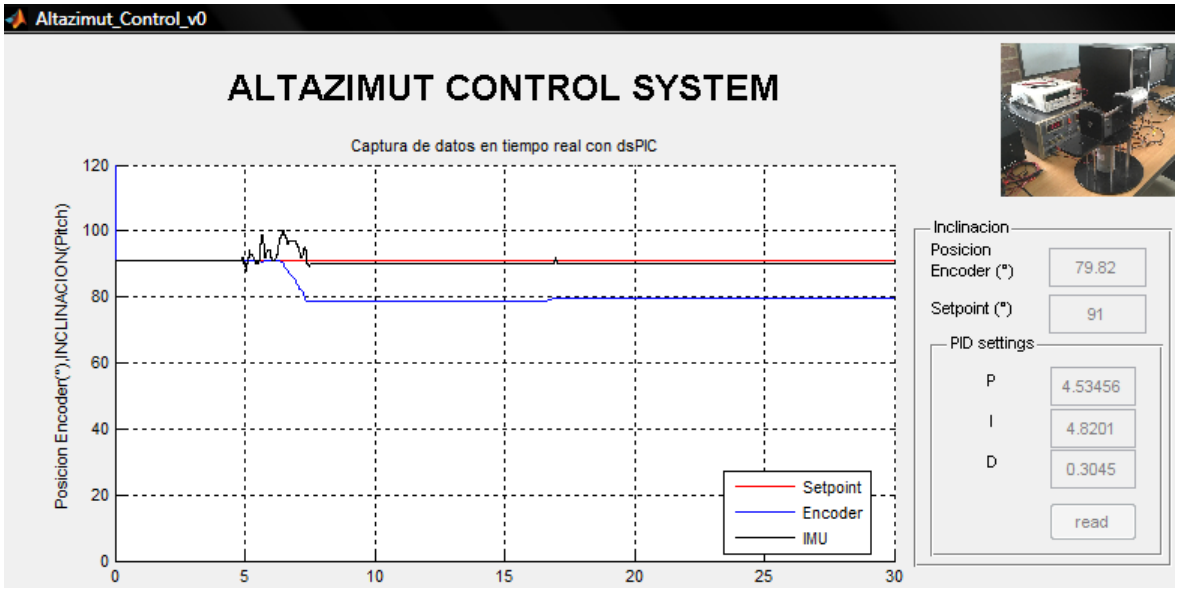


FIGURA 5-28. Prueba Disturbio 10°, fuente el autor.

Los datos de la prueba #2, después de generar el disturbio la respuesta del control de estabilización es de 2.26 segundos.

Este tiempo es valido para realizar correcciones a disturbios externos que sean generados con frecuencias menores a 0.45Hz, se realizaron más pruebas sin lograr mejores resultados en el tiempo de respuesta y sin afectar la estabilidad de la planta.

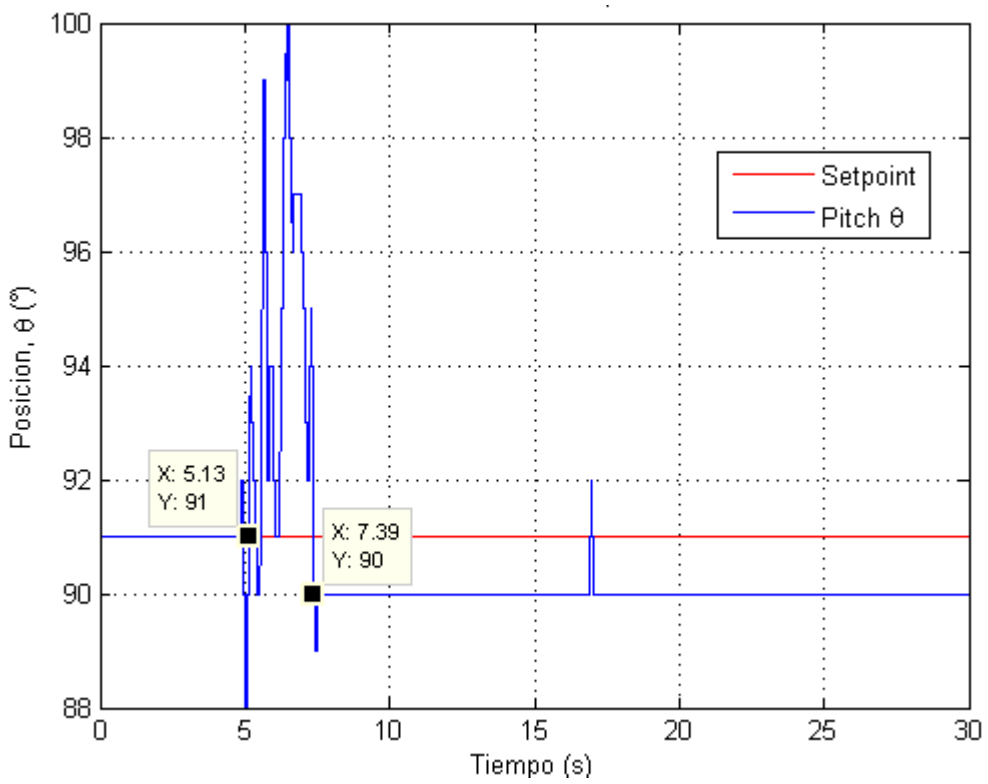


FIGURA 5-29. Zoom Prueba Disturbio 10°, fuente el autor.

CAPITULO 6.

CONCLUSIONES

6.1. CONCLUSIONES

Se realizo un control para la estabilización de la plataforma Altazimut usando acelerómetros y compas magnético con el sensor LSM303DLHC para disturbios menores a 0.45Hz.

Se obtuvo un modelo dinámico de la plataforma Altazimut de caja blanca con las funciones de transferencia de los motores DC.

Se diseño un control de estabilización a partir del controlador PID tipo paralelo para los dos ejes(giro y elevación) por separado.

Se implemento el control sobre hardware dedicado en la tarjeta Seeeduino para lectura del sensor LSM303DLHC y la tarjeta Cerebot MC7^{TM} que incluye etapa de potencia y procesamiento de algoritmos de control.

Se desarrollo una interfaz grafica de usuario en MATLAB® GUIDE para configurar los principales parámetros PID de cada eje, además de activar el control de posición o estabilización.

Fue construido un mecanismo que genera perturbaciones externas en el eje de elevación(θ) de la plataforma Altazimut de 10° sobre el plano horizontal medibles con un transportador.

Se valido el modelo analítico de la plataforma Altazimut confrontándolo con la planta real y sus parámetros característicos obteniendo un error promedio del 10.59%.

La utilización de sistemas computacionales como MATLAB® fue de gran ayuda, principalmente en los procesos de evaluación, sintonización y calibración del control.

El sistema de control de estabilización no responde muy rápido a perturbaciones de baja amplitud debido a que se debe implementar estimador de posición incluyendo giróscopos inerciales, ya que el acelerómetro por sí solo no es suficiente para proporcionar los ángulos de realimentación de posición debido a que solo es preciso en estado estacionario.

Para lograr mejores resultados en el control de estabilización se debe probar una técnica de control robusta que reduzca el tiempo de corrección a perturbaciones.

La medición del ángulo de Giro(ψ) es bastante compleja por la distorsión presente en el sensor magnético del LSM303DLHC que desvían la señal y además el ruido del acelerómetro.

Un sistema embebido como el utilizado es suficiente para el desarrollo de aplicaciones de alta complejidad.

6.2. TRABAJOS FUTUROS

Explorar con una técnica de control que mejore la respuesta a perturbaciones externas, que pueda reducir el tiempo de estabilización.

Adicionar una unidad IMU completa que contenga giroscopio MEMS para aplicar técnicas de estimación de posición basadas en filtro de Kalman o Kalman extendido.

Es posible las múltiples aplicaciones de esta plataforma ya que cuenta con un campo móvil de 360° para cada grado de libertad, descartándose limitaciones de sectores ciegos generados por su propia estructura.

Dicha plataforma podrá ser la base para diferente clase de sistemas de vigilancia por video, sistemas de seguimiento por radar (control de tráfico), además quedando la opción de visualizarse aplicaciones en el campo militar.

Bibliografía

[1] Aranda J. Díaz Martínez J. M. Muñoz Mansilla R., "Un sistema de medida para ensayos en entornos abiertos con modelos a escala de buques de alta velocidad," in *XXIV Jornadas de Automática, Universidad de León*, Madrid, Septiembre, 2003, p. 8.

[2] Digilent Inc., "Cerebot MC7™ Board," Digilent, Pullman, WA, Manual de referencia Doc: 502-185, Septiembre 28,2011.

[3] FARROKH AYAZI, AND KHALIL NAJAFI NAVID YAZDI, "Micromachined Inertial Sensors," *PROCEEDINGS OF THE IEEE*, vol. 86, no. 8, pp. 1640-1642, Aug. 1998.

[4] HONEYWELL. HONEYWELL Magnetic Sensors. [Online]. http://www.magneticsensors.com/literature.php

[5] ST Microelectronics. DigiKey Corporation. [Online]. http://dkc1.digikey.com/il/en/tod/STMicroelectronics/MEMS-Magnetometer_NoAudio/MEMS-MagnetometerMagnetometer_NoAudio.html

[6] Microchip Technology Inc., "dsPIC33FJXXXMCX06A/X08A/X10A," Microchip, Data Sheet DS70594C, 2011.

[7] Microchip Technology Inc. (2013) Microchip. [Online]. http://www.microchip.com/pagehandler/en-us/family/16bit/architecture/dspic33f.html

[8] Microchip Technology Inc., "MPLAB® X IDE User's Guide," Microchip, Manual DS52027B, 2012.

[9] Microchip Technology Inc., "MPLAB® C30 C COMPILER USER'S GUIDE," Microchip , Manual DS51284F, 2007.

[10] Digilent Inc., "PmodRS232X™ ," Digilent, Pullman, WA, Manual de referencia Doc: 502-240, Julio 30, 2012.

[11] Arduino. (2013, Octubre) Arduino. [Online]. http://www.arduino.cc/es/

[12] Pololu Robotics and Electronics. (2013, Octubre) Pololu. [Online]. http://www.pololu.com/catalog/product/2124

[13] STMicroelectronics, "Ultra compact high performance e-compass 3D accelerometer and 3D magnetometer module," STMicroelectronics, Manual Doc ID 018771 Rev 1, Abril, 2011.

[14] RAFAEL HUMBERTO CORDOBA Y ALVARO MARTINEZ LUIS GUILLERMO AYAMA, *SISTEMA DE MEDICION ANGULAR MICROCONTROLADO, DE ALTA RESOLUCION, UTILIZANDO UN SYNCHRO*, 0417th ed., Pontificia Universidad Javeriana, Ed. Bogota, Colombia: Tesis, 2004.

[15] Andres Cuenca. (2005, Marzo) Foros Electronica. [Online]. http://www.forosdeelectronica.com/f16/encoders-informacion-tecnica-25/

[16] Richard C. Dorf and Robert H. Bishop, "Sistemas de Control Moderno," in *Sistemas de Control Moderno*, Miguel Martin-Romo, Ed. Madrid, España: PEARSON PRENTICE HALL, 2005, ch. 2, pp. 58-61.

[17] ALEJANDRA MARGARITA MENDEZ GIRON, , Marcelo Angeles Lomelí Díaz, Ed. México D.F., México: Alfaomega Grupo Editor S.A., 2013, ch. 4, pp. 185-236.

[18] STMicroelectronics, "Using LSM303DLH for a tilt compensated electronic compass," STMicroelectronics, Nota de Aplicación AN3192 Doc ID 17353 Rev 1, 2010.

[19] Ho Cheol Jeong, Jeong Won Kim, Dong-Hwan Hwang, and Sang Jeong Lee Sul Gee Park, "Magnetic Compass Fault Detection Method for GPS/INS/Magnetic Compass Integrated

Navigation Systems ," *International Journal of Control, Automation, and Systems*, vol. 9, no. 2, pp. 276-284, Abril 2011.

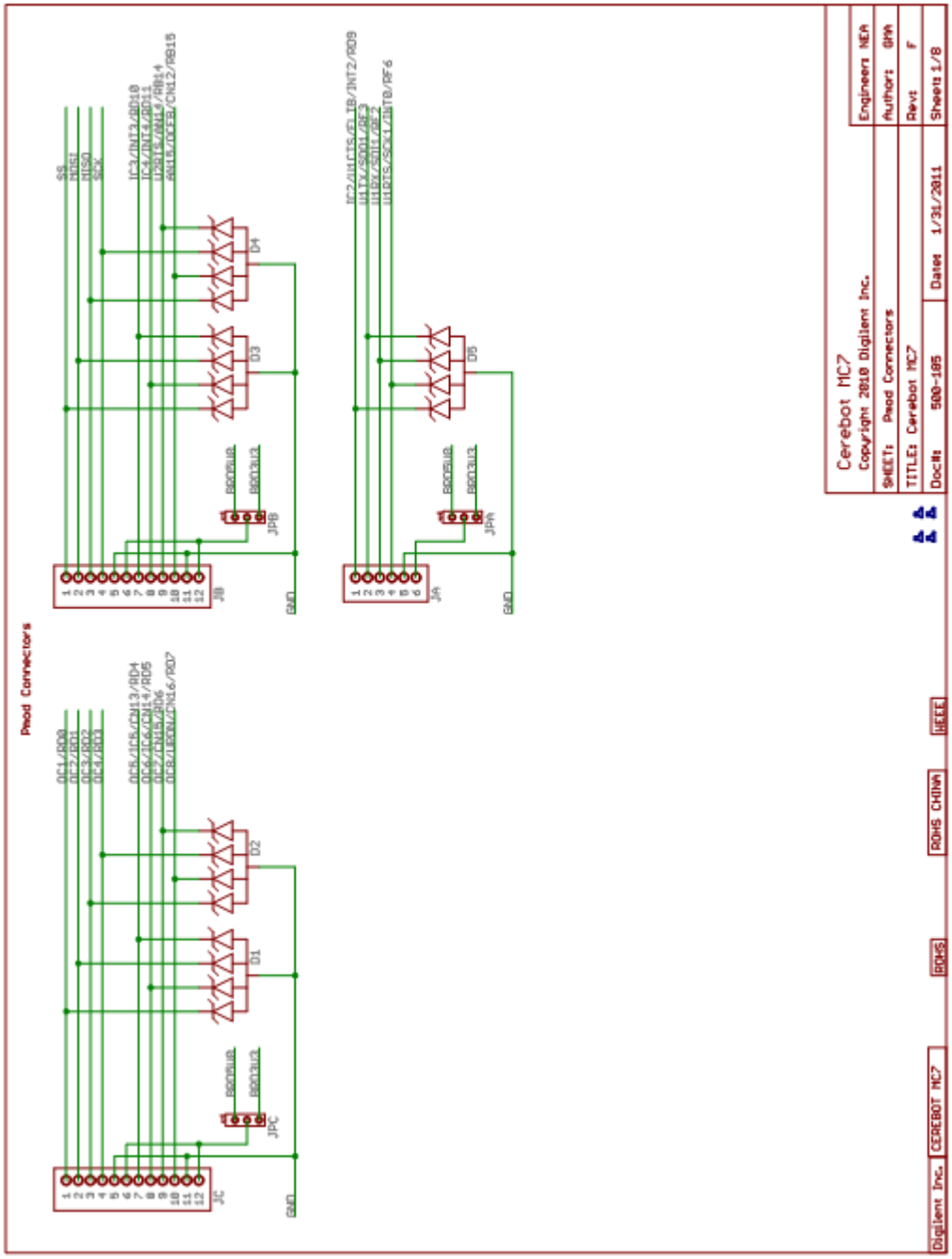
[20] FRANCISCO SALAVERT TORRES, "Implementación de un Regulador PID en un dsPIC33F," Universidad Politécnica de Valéncia. Escuela Técnica Sup. de Ingeniería Informática, Valencia, Tesis 15 de Diciembre, 2010.

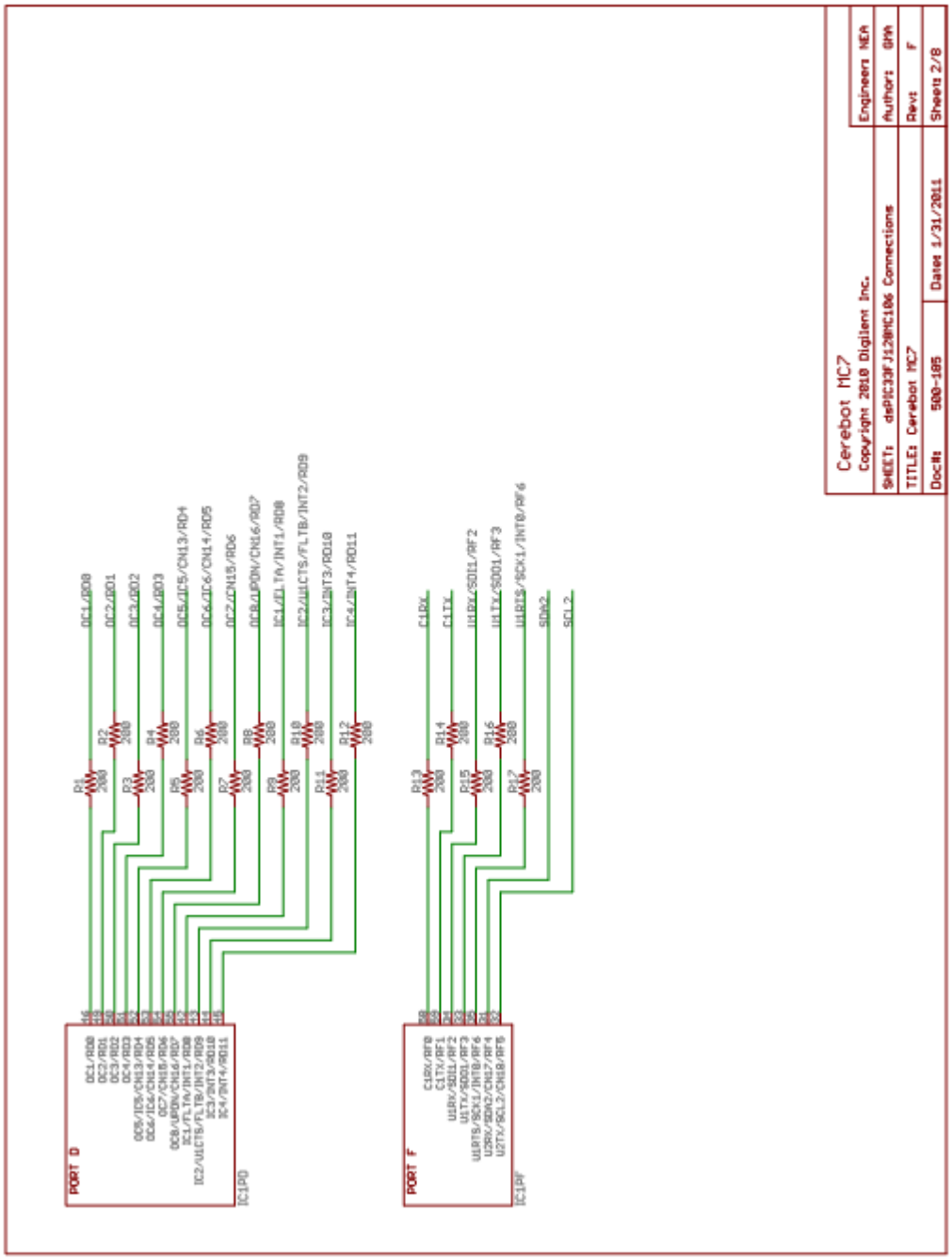
[21] OGATA KATSUHIKO, "Dinamica de sistemas," in *Dinamica de sistemas*. México, México: Prentice-Hall Hispanoamericana, 1987, ch. 1, pp. 4-5.

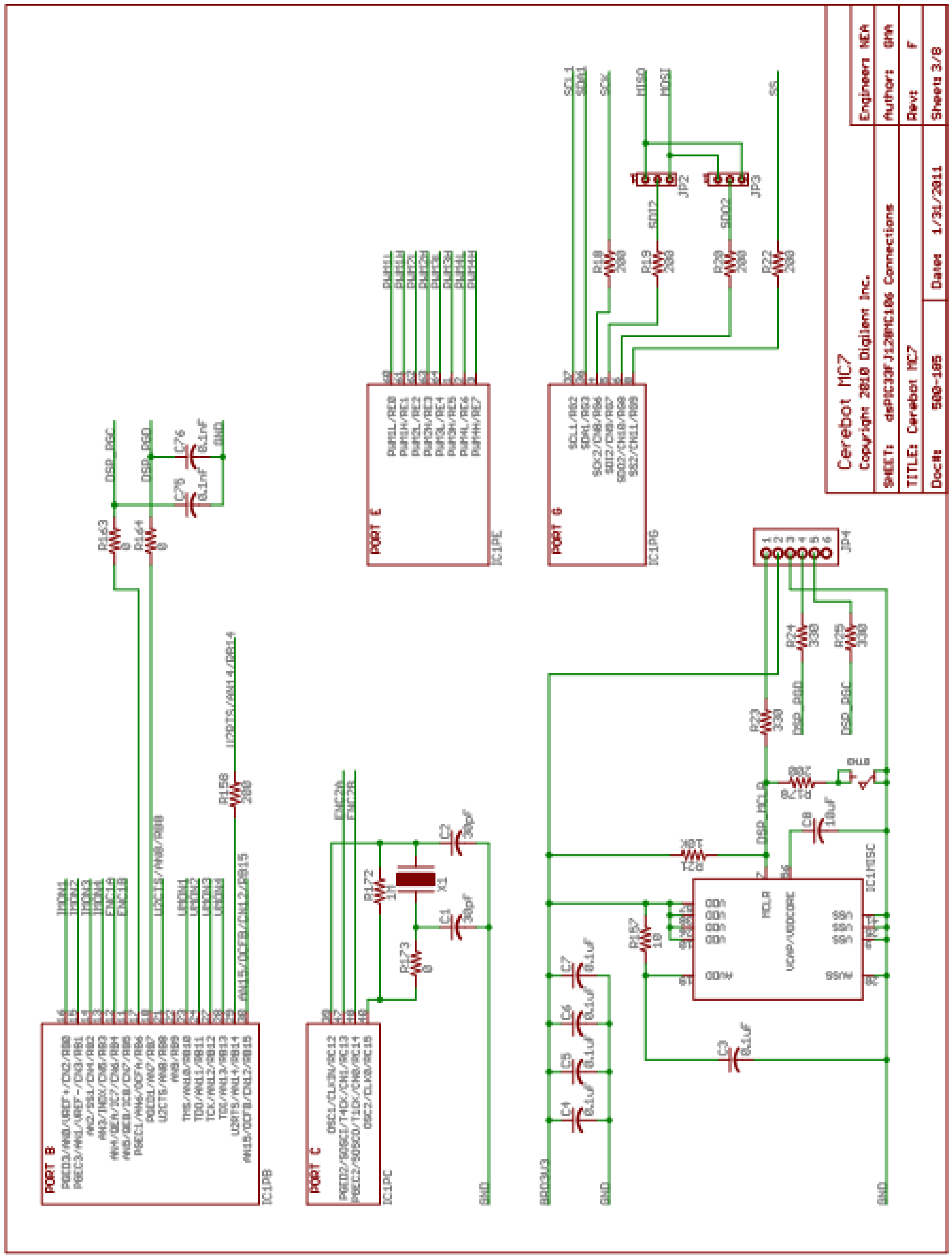
[22] CHRISTHOPHER KONVALIN. (2009, Diciembre) Sensors. [Online]. http://www.sensorsmag.com/sensors/motion-velocity-displacement/compensating-tilt-hard-iron-and-soft-iron-effects-6475

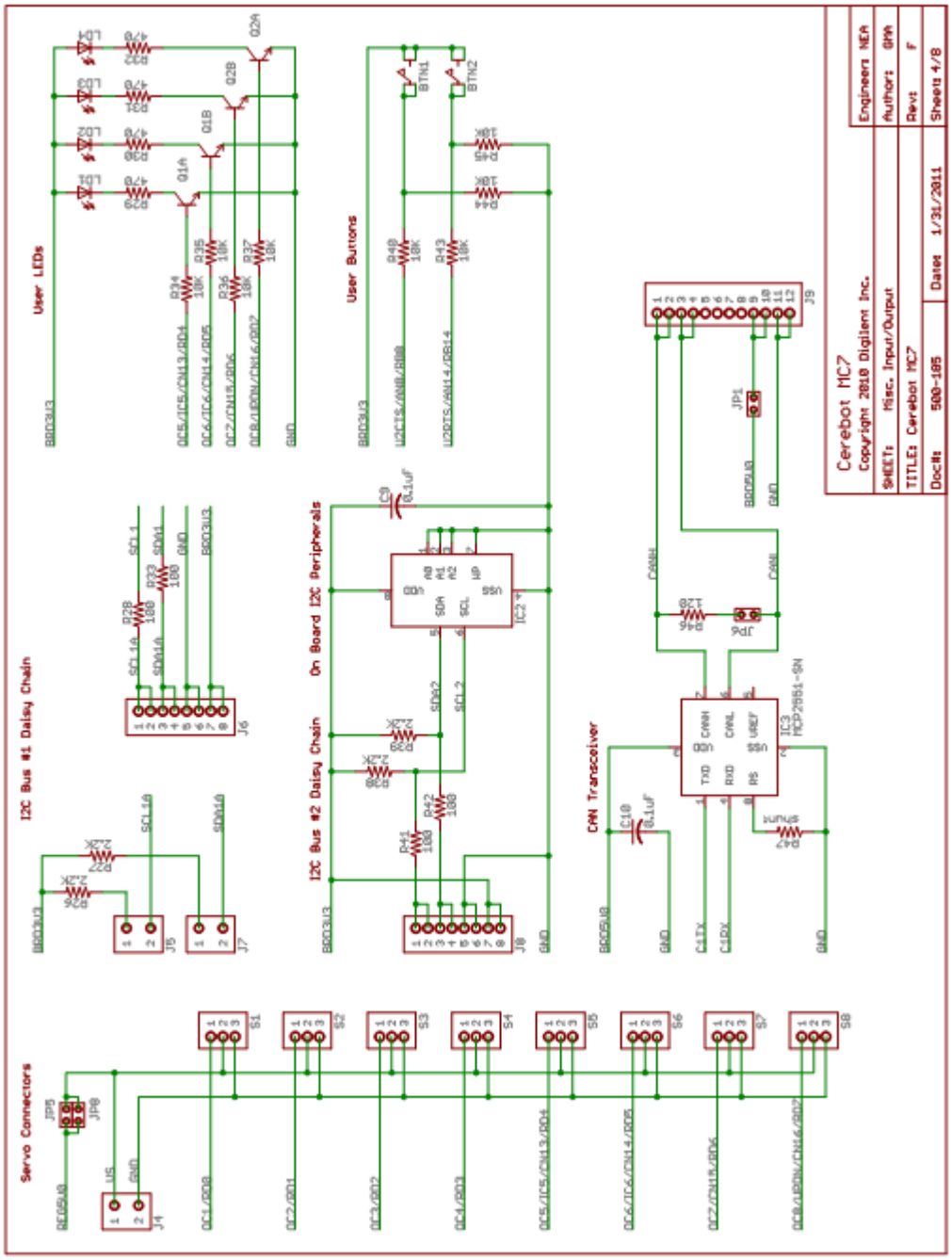
ANEXOS

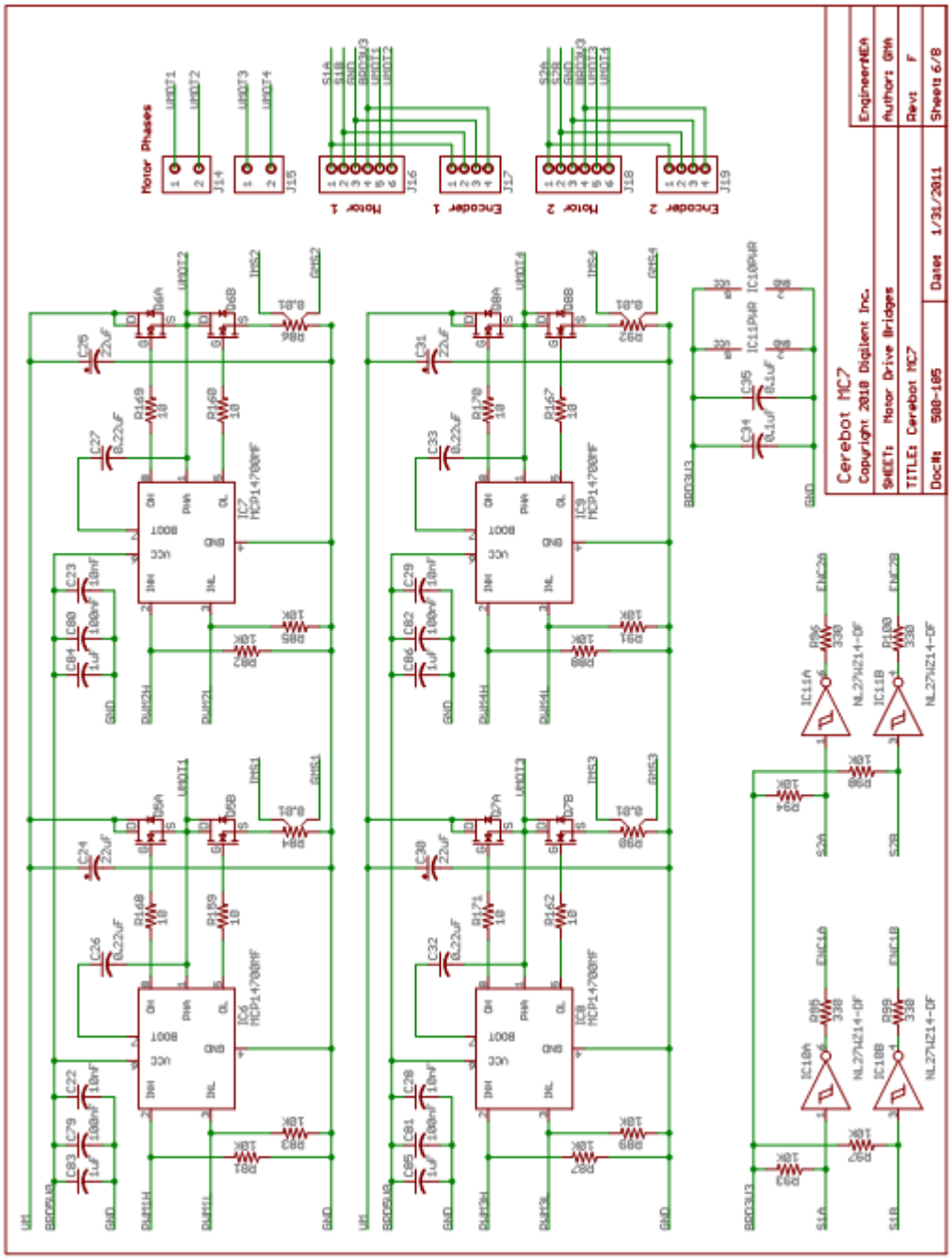
ANEXO A. DIAGRAMAS ESQUEMATICOS CEREBOT MC7™

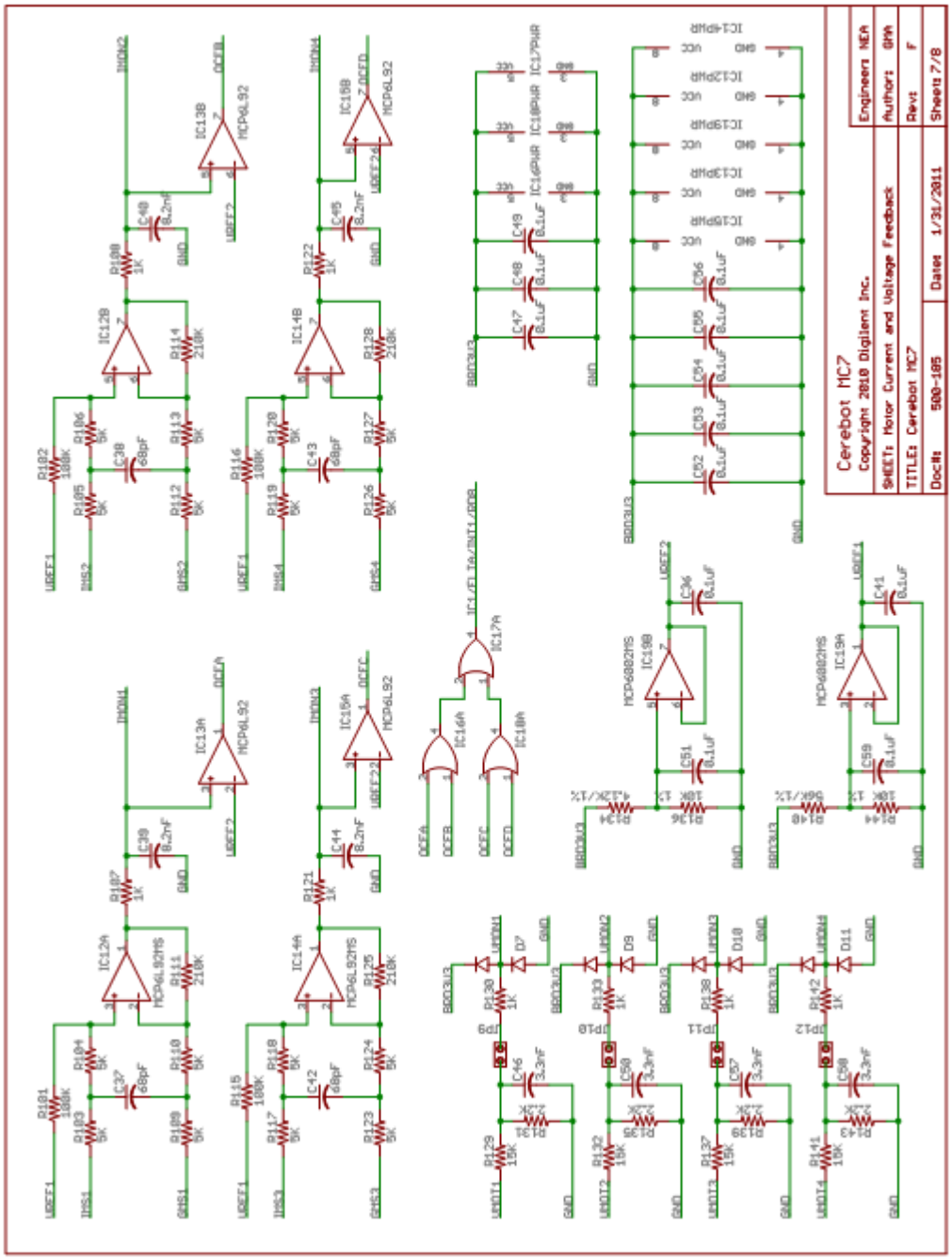


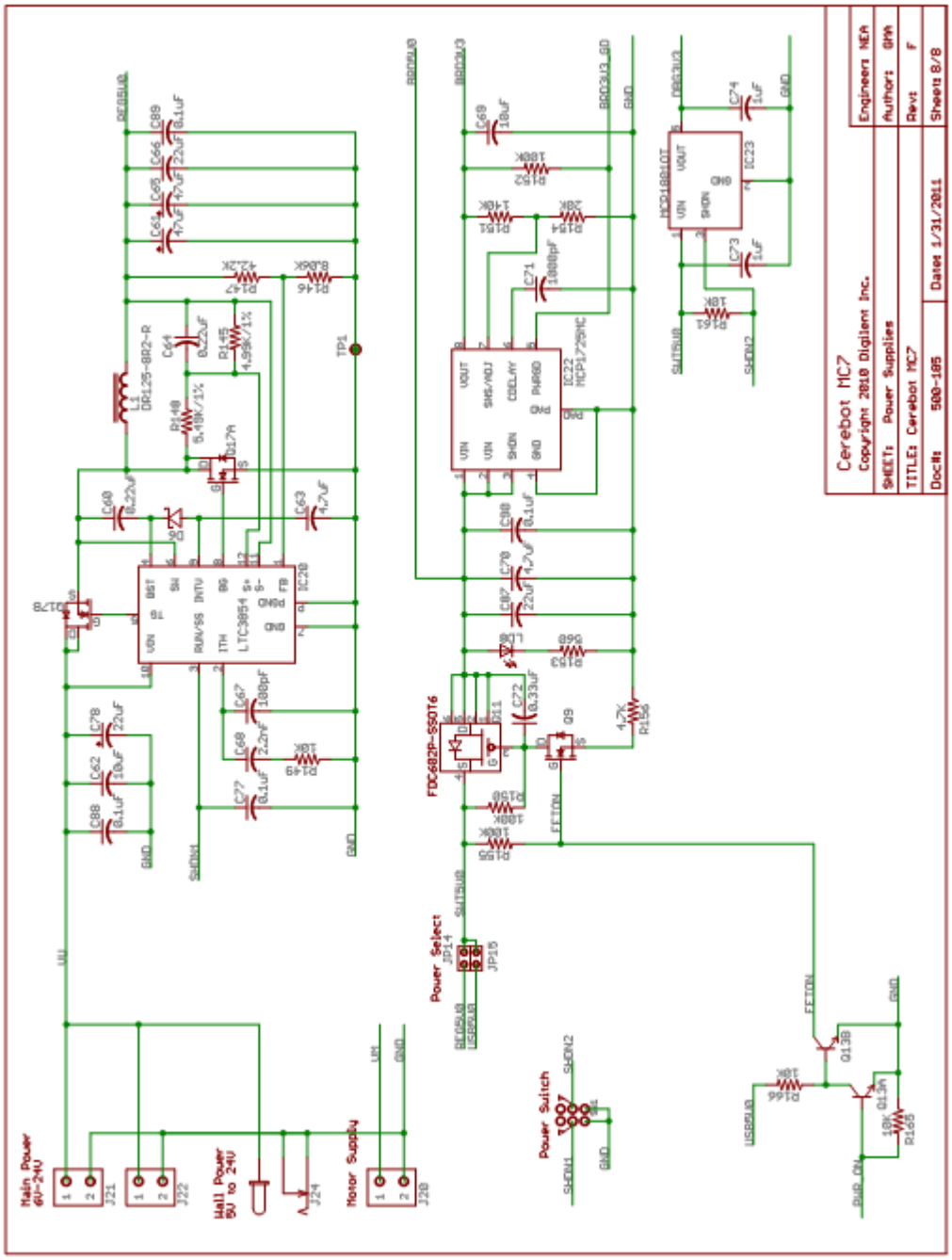

Cerebot MC7
Copyright 2010 Digilent Inc.
Engineer: NEA
SHEET: dsPIC33FJ128MC106 Connections
Author: GPA
TITLE: Cerebot MC7
Rev: F
Doc#: 500-185
Date: 1/31/2011
Sheet 2/8
PORT D
PORT F
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
200
SCL2
SDA2


Cerebot MC7
Copyright 2010 Digilent Inc.
SHEET: dsPIC33FJ128MC106 Connections
TITLE: Cerebot MC7
Doc#: 500-185
Date: 1/31/2011
Engineer: NEA
Author: GMA
Rev: F
Sheet: 3/8


User LEDs
User Buttons
I2C Bus #1 Daisy Chain
On Board I2C Peripherals
I2C Bus #2 Daisy Chain
CAN Transceiver
Servo Connectors
Cerebot MC7
Copyright 2010 Digilent Inc.
Engineer: NEA
Author: GPA
Misc. Input/Output
Rev: F
TITLE: Cerebot MC7
Date: 1/31/2011
Sheet 4/8
Doc#: 500-185


Motor Phases
Motor 1
Encoder 1
Motor 2
Encoder 2
Cerebot MC7
Copyright 2010 Digilent Inc.
SHEET: Motor Drive Bridges
TITLE: Cerebot MC7
Doc#: 500-185
Date: 1/31/2011
Rev: F
Sheets 6/8
Author: GHW
Engineer:NEA


Cerebot MC7
Copyright 2010 Digilent Inc.
SHEET: Motor Current and Voltage Feedback
TITLE: Cerebot MC7
Doc#: 500-185
Date: 1/31/2011
Sheet 7/8
Engineer: NEA
Author: GPW
Rev: F


Cerebot MC7
Copyright 2010 Digilent Inc.
SHEET: Power Supplies
TITLE: Cerebot MC7
Doc#: 500-185
Date: 1/31/2011
Engineer: NEA
Author: GPW
Rev: F
Sheet 8/8
Main Power
6V-24V
Hall Power
5V to 24V
Motor Supply
Power Select
Power Switch
LTC3854
DCR125-802-R
FDC6823P-SSOT6
MCP1725HC
MCP1801OT

ANEXO B. CODIGO FUENTE ARDUINO

ANEXO B.1. CALIBRACION MAGNETOMETRO DE LSM303DLHC

```
#include <Wire.h>   //Libreria I2C
#include <LSM303.h> //Libreria para uso LSM303DLHC

LSM303 compass;
LSM303::vector running min = {2047, 2047, 2047}, running max = {-2048, -
2048, -2048};

void setup() {
  Serial.begin(9600);
  Wire.begin();
  compass.init();
  compass.enableDefault();
}

void loop() {
  compass.read();

  running min.x = min(running min.x, compass.m.x);
  running min.y = min(running min.y, compass.m.y);
  running min.z = min(running min.z, compass.m.z);

  running max.x = max(running max.x, compass.m.x);
  running max.y = max(running max.y, compass.m.y);
  running max.z = max(running max.z, compass.m.z);

  Serial.print("M min ");
  Serial.print("X: ");
  Serial.print((int)running min.x);
  Serial.print(" Y: ");
  Serial.print((int)running min.y);
  Serial.print(" Z: ");
  Serial.print((int)running min.z);

  Serial.print(" M max ");
  Serial.print("X: ");
  Serial.print((int)running max.x);
  Serial.print(" Y: ");
  Serial.print((int)running max.y);
  Serial.print(" Z: ");
  Serial.println((int)running max.z);

  delay(100);
}
```

ANEXO B.2. DATOS FILTRADOS DE LSM303DLHC

```
#include <Wire.h>   //Libreria I2C
#include <LSM303.h> //Libreria para uso LSM303DLHC
```

```
// change ACC LA FS,  ACC LA SO as reqd per the datasheet table 3
#define ACC LA FS   0b00
#define ACC LA SO   1
#define ACC SCALE (1000 / ACC LA SO)


/*
magnetic field sensor ranges and scale
- Earth's magnetic field varies from ~0.25-0.65 Gauss;
http://en.wikipedia.org/wiki/Earth's magnetic field
- same for x,y axes, z is slightly less sensitive
- ref. datasheet for other ranges
GN    Range       Divisor(x,y)  Divisor(z)
001   +/- 1.3G    1100          980
*/

#define MAG GN           0b001
#define MAG GN SCALE XY   1100
#define MAG GN SCALE Z     980

LSM303 compass;
int ax[11],ay[11],az[11];
int mx[11],my[11],mz[11];
int Ax,Ay,Az;
int Mx,My,Mz;
char Pointer;
char i;

void setup()
{
  //Inicializa comunicacion serial a 115200baudios
  //con la USART2 del dsPIC33FJ128MC706A
  Serial.begin(115200);
  Wire.begin();
  compass.init();
  // Accelerometer Full scale +/- 2G High Resolution=enable,
  //Continuos update
  compass.enableDefault();
  // magnetic scale = +/-1.3Gauss
  compass.setMagGain(compass.magGain 13);
  // 0x57 = normal power mode, all accel axes on, BW=100Hz
  compass.writeAccReg(LSM303 CTRL REG1 A, 0x57);
  // 0x30 = mag 75Hz output rate
  compass.writeMagReg(LSM303 CRA REG M, 0x30);
when the program starts

  //Inicializa Pointer
  Pointer=1;
}

void loop()
{
  //Lectura de Raw Data Acelerometro y Compass del LSM303LDLHC
  compass.read();
  //Movimiento al arreglo circular de Raw data del Accelerometro y
  //Compass
  ax[Pointer]=compass.a.x;
  ay[Pointer]=compass.a.y;
  az[Pointer]=compass.a.z;
  mx[Pointer]=compass.m.x;
  my[Pointer]=compass.m.y;
  mz[Pointer]=compass.m.z;

  //Promedio de señales cada 10 muestras
  for (i=1; i < 11; i++)
```

```
  {
    Ax += ax[i];
    Ay += ay[i];
    Az += az[i];
    Mx += mx[i];
    My += my[i];
    Mz += mz[i];
  }// final for(i=1; i < 11; i++)
  Ax /= 10;
  Ay /= 10;
  Az /= 10;
  Mx /= 10;
  My /= 10;
  Mz /= 10;

  // Envia al dsPIC33FJ128MC706A Data-Promedio del Acelerometro en x,y,z
  Serial.print("A");
  Serial.print((int)Ax);
  Serial.print(",");
  Serial.print((int)Ay);
  Serial.print(",");
  Serial.print((int)Az);
  Serial.print(",");
  // Envia al dsPIC33FJ128MC706A Data-Promedio del Compass en x,y,z
  Serial.print("M");
  Serial.print((int)Mx);
  Serial.print(",");
  Serial.print((int)My);
  Serial.print(",");
  Serial.print((int)Mz);
  Serial.print("}");
  //Cada 10ms envia informacion
  delay(10);
  if(Pointer < 10)Pointer++;  //Incrementa Pointer
  else Pointer=1;
}
```

ANEXO C. CODIGO FUENTE dsPIC33FJ128MC706A

ANEXO C.1. HEADER FILE: Init_Module.h

```
/*
 * File:   Init Module.h
 * Author: Mauricio Machado
 *
 * Created on 30 de julio de 2013, 07:48 PM
 */

#ifndef INIT MODULE H
#define     INIT MODULE H

//#include "p33FJ128MC706A.h" // Include this in your MPLAB project under
"Header Files" from C:\Program Files (x86)\Microchip\MPLAB
C30\support\dsPIC30F\h

#define FCY             40000000    //Instruction Cycle Frequency 40MIPS
#define Fs              8000 // 8000 = 8kHz
#define Fs ADC2         5000 // 8000 = 8kHz
#define BAUDRATE        115200        //Baudios
#define BRGVAL          ((FCY/BAUDRATE)/4)-1
#define REL MOTO REDUCTOR1  18.56 // 18.56:1= relacion Moto-Reductor 1
Experimental 7424ppr
#define REL MOTO REDUCTOR2  18.57               // 18.57:1= relacion
Moto-Reductor 2 Experimental 3714 ppr
#define FACTOR POS ENC1 360/(REL MOTO REDUCTOR1 * 400)    //Factor de
conversion de Posicion Eje de moto-reductor
#define FACTOR POS ENC2 360/(REL MOTO REDUCTOR2 * 200)    //Factor de
conversion de Posicion Eje de moto-reductor
#define TRUE            1
#define FALSE           0
/*
Rangos y escalas de sensor de Campo Magnetico(Compass)
- Campo Magnetico de la Tierra varía entre ~0.25-0.65 Gauss;
- http://en.wikipedia.org/wiki/Earth's magnetic field
- igual para ejes x,y, z es ligeramente menos sensible
- ref. datasheet para otros rangos
GN    Range       Divisor(x,y)  Divisor(z)
001   +/- 1.3G    1100          980
*/
#define MAG GN           0b001
#define MAG GN SCALE XY   1100
#define MAG GN SCALE Z     980
// Cambio de rangos en ACC LA FS,  ACC LA SO escala para +/-2G ver
datasheet tabla 3
#define ACC LA FS   0b00
#define ACC LA SO   1
#define ACC SCALE (1000 / ACC LA SO)
#define X 0
#define Y 1
#define Z 2

typedef struct vector
{
    float x, y, z;
} vector;
// vector functions
static void LSM303 vector cross(const vector *a, const vector *b, vector
*out);
static float LSM303 vector dot(const vector *a,const vector *b);
static void LSM303 vector normalize(vector *a);

// Funciones de inicialización
void Init IO(void);
```

```
void Init Timer1(void);
void Init Usart(void);
void Init Usart2(void);
void putUART1( char c);
void Init PWM1(void);
void Init INTx OI(void);     // Interrupccion externa en pin INT1/RD8
void Init QEI1(void);        // QEI para motor1
void Init CN(void);          // Interrupcion por Cambio en entrada RC13-
RC14
void Init QEI2(void);        // QEI para motor2 por interrupcion externa
RC13
float Posicion Encoder1(void); // Retorna la posicion de 0-360° del
Encoder1-Motor1
float Posicion Encoder2(void); // Retorna la posicion de 0-360° del
Encoder2-Motor2
void MCPWM1(int);            // Control puente H(VMOTOR1-VMOTOR2) de
Motor1-PWM1 con sentido de giro
void MCPWM2(int);            // Control puente H(VMOTOR3-VMOTOR4) de
Motor2-PWM con sentido de giro
void Stop PWM(void);         // Apaga los módulos MCPWM del motor1 y
motor2
void getPitchRollHeading(void); // Calcula Angulo de Giro(Heading-
Yaw),Pitch, Roll
void LSM303 Data Init(void);

#endif      /* INIT MODULE H */
```

ANEXO C.2. IO_Ports.c

```
#include "p33FJ128MC706A.h"
#include "Init Module.h"

extern int Cont ENC2;   // Contador de Posicion Encoder2-Motor2
extern float Posicion Motor2;

void Init IO(void)  //Inicializacion de IO Ports
{
   TRISBbits.TRISB4=1;      //configura Pin RB4 como entrada ENC1A
Cerebot
   TRISBbits.TRISB5=1;      //configura Pin RB5 como entrada ENC1B
Cerebot
   TRISBbits.TRISB8=1;      //configura Puerto RB8 como entrada BTN1
Cerebot
   TRISBbits.TRISB14=1;     //configura Puerto RB14 como entrada BTN2
Cerebot
   TRISCbits.TRISC13=1;     //configura Pin RC13 como entrada ENC2A
Cerebot
   TRISCbits.TRISC14=1;     //configura Pin RC14 como entrada ENC2B
Cerebot
   TRISDbits.TRISD4=0;      //configura puerto RD4 como salida LED1
Cerebot
   TRISDbits.TRISD5=0;      //configura puerto RD5 como salida LED2
Cerebot
   TRISDbits.TRISD6=0;      //configura puerto RD6 como salida LED3
Cerebot
   TRISDbits.TRISD7=0;      //configura puerto RD7 como salida LED4
Cerebot
   TRISDbits.TRISD8=1;      //configura puerto RD8 como entrada Falla PWM
Cerebot proteccion hardware
```

```
   AD1PCFGLbits.PCFG4=1;    //1=Port pin in Digital mode, PCFG4 = AN4
   AD2PCFGLbits.PCFG4=1;    //1=Port pin in Digital mode, PCFG4 = AN4
   AD1PCFGLbits.PCFG5=1;    //1=Port pin in Digital mode, PCFG5 = AN5
   AD2PCFGLbits.PCFG5=1;    //1=Port pin in Digital mode, PCFG5 = AN5
   AD1PCFGLbits.PCFG8=1;    //1=Port pin in Digital mode, PCFG8 = AN8
   AD2PCFGLbits.PCFG8=1;    //1=Port pin in Digital mode, PCFG8 = AN8
   AD1PCFGLbits.PCFG14=1;   //1=Port pin in Digital mode, PCFG14 = AN14
   AD2PCFGLbits.PCFG14=1;   //1=Port pin in Digital mode, PCFG14 = AN14

   //AD1PCFGH/AD1PCFGL: Port Configuration Register
   AD1PCFGL=0xFFFF;       //Inicia todos los pines analogos como
digitales

   AD1PCFGLbits.PCFG15=0;//0=Port pin in Analog mode, PCFGx = AN15/RB15
   AD2PCFGLbits.PCFG15=0;//0=Port pin in Analog mode, PCFGx = AN15/RB15
   AD1PCFGLbits.PCFG13=0;//0=Port pin in Analog mode, PCFG13 = VMON4 /
AN13 / RB13
   AD2PCFGLbits.PCFG13=0;//0=Port pin in Analog mode, PCFG13 = VMON4 /
AN13 / RB13
   AD1PCFGLbits.PCFG12=0;//0=Port pin in Analog mode, PCFG12 = VMON3 /
AN12 / RB12
   AD2PCFGLbits.PCFG12=0;//0=Port pin in Analog mode, PCFG12 = VMON3 /
AN12 / RB12
   AD1PCFGLbits.PCFG11=0;//0=Port pin in Analog mode, PCFG11 = VMON2 /
AN11 / RB11
   AD2PCFGLbits.PCFG11=0;//0=Port pin in Analog mode, PCFG11 = VMON2 /
AN11 / RB11
   AD1PCFGLbits.PCFG10=0;//0=Port pin in Analog mode, PCFG10 = VMON1 /
AN10 / RB10
   AD2PCFGLbits.PCFG10=0;//0=Port pin in Analog mode, PCFG10 = VMON1 /
AN10 / RB10
   AD1PCFGLbits.PCFG3=0;//0=Port pin in Analog mode, PCFG3 = IMON4 / AN3
/ RB3
   AD2PCFGLbits.PCFG3=0;//0=Port pin in Analog mode, PCFG3 = IMON4 / AN3
/ RB3
   AD1PCFGLbits.PCFG2=0;//0=Port pin in Analog mode, PCFG2 = IMON3 / AN2
/ RB2
   AD2PCFGLbits.PCFG2=0;//0=Port pin in Analog mode, PCFG2 = IMON3 / AN2
/ RB2
   AD1PCFGLbits.PCFG1=0;//0=Port pin in Analog mode, PCFG1 = IMON2 / AN1
/ RB1
   AD2PCFGLbits.PCFG1=0;//0=Port pin in Analog mode, PCFG1 = IMON2 / AN1
/ RB1
   AD1PCFGLbits.PCFG0=0;//0=Port pin in Analog mode, PCFG0 = IMON1 / AN0
/ RB0
   AD2PCFGLbits.PCFG0=0;//0=Port pin in Analog mode, PCFG0 = IMON1 / AN0
/ RB0
   //Configuracion Pines Módulo PWM1
   TRISEbits.TRISE0=0; //configura todo puerto RE0/PWM1L como salida
   TRISEbits.TRISE1=0; //configura todo puerto RE1/PWM1H como salida
   TRISEbits.TRISE2=0; //configura todo puerto RE2/PWM2L como salida
   TRISEbits.TRISE3=0; //configura todo puerto RE3/PWM2H como salida
   //Configuracion Pines Módulo PWM2
   TRISEbits.TRISE4=0; //configura todo puerto RE4/PWM3L como salida
   TRISEbits.TRISE5=0; //configura todo puerto RE5/PWM3H como salida
   TRISEbits.TRISE6=0; //configura todo puerto RE6/PWM4L como salida
   TRISEbits.TRISE7=0; //configura todo puerto RE7/PWM4H como salida
   /*******************************************************
   *     Oscillator Calculations                          *
   * Fin= 8Mhz,  N1=2, N2=2, M=40 **** Fosc= Fin x(M/N1xN2)  *
   * Fosc= 8Mhz x (40/2x2)= 80Mhz                        *
   *******************************************************/
   CLKDIVbits.PLLPRE=0;     //N1=2
   CLKDIVbits.PLLPOST=0;    //N2=2
   PLLFBD= 38;              //M=40
```

```
   OSCTUN=0;
}

// Inicializacion de Notificacion por cambio en los Pines RC13-RC14 para
el Encoder Motor2
void Init CN(void)
{
    IPC4bits.CNIP   = 2;   // Prioridad: 2 de la interrupccion por Cambio
en entrada RC13-RC14
    CNEN1bits.CN1IE = 1;   // Enable CN1/RC13 pin for interrupt detection
ENC2A
    IEC1bits.CNIE   = 1;   // Enable CN interrupts
    IFS1bits.CNIF   = 0;   // Reset de bandera CN interrupccion
    Cont ENC2 = 3714;      // Inicia Contador de Posicion Encoder2-Motor2
    Posicion Motor2 = 0;   // Inicializa valor de posicion de encoder
}

void   attribute  ((interrupt, no auto psv)) CNInterrupt(void)
{
    if (PORTCbits.RC13)     // Si RC13/ENC2A se ha puesto a 1 (flanco de
subida),
    {
       if (PORTCbits.RC14 == 0)Cont ENC2++;  // Si RC14/ENC2B se ha
puesto a 0 (flanco de bajada)
                                             // entonces incrementar una
unidad el valor de Cont ENC2.
       else
           Cont ENC2--;         // entonces decrementar una unidad el
valor de Cont ENC2.
    }
    else                        // Si RC13/ENC2A se ha puesto a 0 (flanco
de bajada),
    {
        if(PORTCbits.RC14 == 0)Cont ENC2--;  // Si RC14/ENC2B se ha
puesto a 0 (flanco de bajada)
                                             // entonces decrementar una
unidad el valor de X.
        else
            Cont ENC2++;         // entonces incrementar una unidad el
valor de Cont ENC2.
    }
    if(Cont ENC2 >= 3714) Cont ENC2 = 0; //Reiniciamos contador
    else if(Cont ENC2 < 0)Cont ENC2= 3714; //Reiniciamos contador para
que sea decrementado

    IFS1bits.CNIF = 0;                   // Clear CN interrupt
}

float Posicion Encoder2(void)
{
    Posicion Motor2 = 360 - (float)Cont ENC2 * FACTOR POS ENC2;
    return Posicion Motor2;
}
```

ANEXO C.3. TIMER_Module.c

```
#include "p33FJ128MC706A.h"
#include "Init Module.h"

extern long anidador 10ms;
```

```
extern char flag 10ms;

//================================= GRADUACIONES TIMER1
======================
//t ciclo= 1/(frec xtal/4) = 1/(80 MHz/4) = 50 ns
//Cálculo timer1:
//Durac timer= t ciclo * PREESCALA * TMR1 => tiempo base de 100us
//Durac timer 10ms= 50ns * 8 * 250 * 100(n anidaciones) = 10ms
//================================================================================
=======
void Init timer1() //funcion de inicialización del Timer1
{
    // IEC0 INTERRUPT ENABLE CONTROL REGISTER 0
    IEC0bits.T1IE   =   1;  //Interrupt request enabled del Timer1
    IPC0bits.T1IP   =   6; // Prioridades : Timer1=6;
    //   configuración bits T1CON del Timer1
    T1CONbits.TON   =   1;  //habilita timer1
    T1CONbits.TCS   =   0;  //reloj interno(FCY) Fosc/4
    T1CONbits.TSYNC =   0;  //bit ignorado es para sincronización externa
    T1CONbits.TCKPS1=   0;  //bits de Preescala 1:8
    T1CONbits.TCKPS0=   1;
    T1CONbits.TSIDL =   0;  //Continue module operation in Idle mode
    TMR1            =   65285; //se carga en el TMR1 65285,65535 - 65285
= 250
    anidador 10ms  =   100;    //100= inicializa contador de anidaciones
para PID periodo de 10ms
    flag 10ms      =   0;    //se inicializa bandera 100ms para PID
}

/*********************Interrupcion TIMER1*****************************/
void   attribute  ((interrupt, no auto psv)) T1Interrupt(void)
{// Timer1 address in the interrupt vector table
  IFS0bits.T1IF = 0;   // Interrupt flag reset
  TMR1            =   65285; //se carga en el TMR1 65285
  anidador 10ms--;
  if(anidador 10ms==0)
  {
      anidador 10ms = 100; //se reinicia contador de anidaciones
      flag 10ms=1;         //se sube bandera para indicar que pasaron
10ms
  }

}
```

ANEXO C.4. USART_Module.c

```
#include "p33FJ128MC706A.h"
#include "Init Module.h"
#include <stdio.h>
#include <stdlib.h>

extern char CaracterRx;
extern char CaracterRx2;
extern char ArrayRx[20];
extern char ArrayRx2[50];
```

```
extern int Ascci2Decimal[10];  //Conversion a decimal
extern int int reg[9];  // memoria para registros de entrada.
char PointerRx;
char PointerRx2;
char ReadString[20];
char ReadString 2[20];
float Conver2Float;
float Conver2Float 2;
extern char Command RunStop;
extern char Command Estabilizacion;
extern char New ref;
//PID1- Motor1(PWM1)
extern float ref;  // referencia
extern float Kp;   // ganancia proporcional
extern float Ki;    // ganancia integral
extern float Kd;    // ganancia derivativa
// PID2- Motor2-PWM2
extern float ref 2;  // referencia
extern float Kp 2;   // ganancia proporcional
extern float Ki 2;   // ganancia integral
extern float Kd 2;   // ganancia derivativa

//Variables del Sensor LSM303DLHC
extern float Ax, Ay, Az;    // Datos Raw del Acelerometro
extern float Mx, My, Mz;    // Datos Raw del Compass
char i,j,iReadString,jReadString;       // para el buble for


void Init Usart()
{
    // configure U1MODE
    U1MODEbits.UARTEN = 0;    // Bit15 TX, RX DISABLED, ENABLE at end of
func
    //U1MODEbits.notimplemented;    // Bit14
    U1MODEbits.USIDL = 0;     // Bit13 Continue in Idle
    U1MODEbits.IREN = 0;      // Bit12 No IR translation
    U1MODEbits.RTSMD = 0;     // Bit11 Simplex Mode
    //U1MODEbits.notimplemented;    // Bit10
    U1MODEbits.UEN = 0;       // Bits8,9 TX,RX enabled, CTS,RTS not
    U1MODEbits.WAKE = 0;      // Bit7 No Wake up (since we don't sleep
here)
    U1MODEbits.LPBACK = 0;    // Bit6 No Loop Back
    U1MODEbits.ABAUD = 0;     // Bit5 No Autobaud (would require sending
'55')
    U1MODEbits.URXINV = 0;    // Bit4 IdleState = 1  (for dsPIC)
    U1MODEbits.BRGH = 1;      // Bit3 1= High-speed mode
    U1MODEbits.PDSEL = 0;     // Bits1,2 8bit, No Parity
    U1MODEbits.STSEL = 0;     // Bit0 1-stop bit

    //U1BRG = 85.8;
    U1BRG = BRGVAL;             // 40Mhz osc, 115200 Baud, BAUD Rate
Setting
    // Load all values in for U1STA SFR
    U1STAbits.UTXISEL1 = 0;   //Bit15 Int when Char is transferred (1/2
config!)
    U1STAbits.UTXINV = 0;     //Bit14 N/A, IRDA config
    U1STAbits.UTXISEL0 = 0;   //Bit13 Other half of Bit15
    //U1STAbits.notimplemented = 0; //Bit12
    U1STAbits.UTXBRK = 0;     //Bit11 Disabled
    U1STAbits.UTXEN = 0;      //Bit10 TX pins controlled by periph
    U1STAbits.UTXBF = 0;      //Bit9 *Read Only Bit*
    U1STAbits.TRMT = 0;       //Bit8 *Read Only bit* Transmit Shift
Register is not empty
    U1STAbits.URXISEL = 0;    //Bits6,7 Int. on character recieved
    U1STAbits.ADDEN = 0;      //Bit5 Address Detect Disabled
```

```
    U1STAbits.RIDLE = 0;      //Bit4 *Read Only Bit*Receiver is active
    U1STAbits.PERR = 0;       //Bit3 *Read Only Bit*Parity error has not
been detected
    U1STAbits.FERR = 0;       //Bit2 *Read Only Bit*Framing error has not
been detected
    U1STAbits.OERR = 0;       //Bit1 *Read Only Bit*Receive buffer has
not overflowed.
    U1STAbits.URXDA = 0;      //Bit0 *Read Only Bit*Receive buffer is
empty

    IPC2bits.U1RXIP    = 4;     // UART1 Receiver Interrupt Priority bits
    IFS0bits.U1TXIF = 0;      // Clear the Transmit Interrupt Flag
    IEC0bits.U1TXIE = 0;      // Disable Transmit Interrupts
    IFS0bits.U1RXIF = 0;      // Clear the Recieve Interrupt Flag
    IEC0bits.U1RXIE = 1;      // Enable Recieve Interrupts

    U1MODEbits.UARTEN = 1;    // And turn the peripheral on
    U1STAbits.UTXEN = 1;
    PointerRx=0;
    CaracterRx=0;
    Command RunStop = FALSE;    // Inicia comando en Stop Control Motores
    Command Estabilizacion = FALSE; // Inicia comando en Estabilizacion
desactivado
    New ref = FALSE;            // Inicia en 0 bandera de cambio de
referencia por LSM303DLHC
}

void Init Usart2()
{
    // configure U2MODE
    U2MODEbits.UARTEN = 0;    // Bit15 TX, RX DISABLED, ENABLE at end of
func
    //U1MODEbits.notimplemented;    // Bit14
    U2MODEbits.USIDL = 0;     // Bit13 Continue in Idle
    U2MODEbits.IREN = 0;      // Bit12 No IR translation
    U2MODEbits.RTSMD = 0;     // Bit11 Simplex Mode
    //U1MODEbits.notimplemented;    // Bit10
    U2MODEbits.UEN = 0;       // Bits8,9 TX,RX enabled, CTS,RTS not
    U2MODEbits.WAKE = 0;      // Bit7 No Wake up (since we don't sleep
here)
    U2MODEbits.LPBACK = 0;    // Bit6 No Loop Back
    U2MODEbits.ABAUD = 0;     // Bit5 No Autobaud (would require sending
'55')
    U2MODEbits.URXINV = 0;    // Bit4 IdleState = 1  (for dsPIC)
    U2MODEbits.BRGH = 1;      // Bit3 1= High-speed mode
    U2MODEbits.PDSEL = 0;     // Bits1,2 8bit, No Parity
    U2MODEbits.STSEL = 0;     // Bit0 1-stop bit

    //U2BRG = 259;
    U2BRG = BRGVAL;             // 40Mhz osc, 115200 Baud, BAUD Rate
Setting
    // Load all values in for U2STA SFR
    U2STAbits.UTXISEL1 = 0;  //Bit15 Int when Char is transferred (1/2
config!)
    U2STAbits.UTXINV = 0;     //Bit14 N/A, IRDA config
    U2STAbits.UTXISEL0 = 0;   //Bit13 Other half of Bit15
    //U1STAbits.notimplemented = 0; //Bit12
    U2STAbits.UTXBRK = 0;     //Bit11 Disabled
    U2STAbits.UTXEN = 0;      //Bit10 TX pins controlled by periph
    U2STAbits.UTXBF = 0;      //Bit9 *Read Only Bit*
    U2STAbits.TRMT = 0;       //Bit8 *Read Only bit* Transmit Shift
Register is not empty
    U2STAbits.URXISEL = 0;    //Bits6,7 Int. on character recieved
    U2STAbits.ADDEN = 0;      //Bit5 Address Detect Disabled
    U2STAbits.RIDLE = 0;      //Bit4 *Read Only Bit*Receiver is active
```

```
    U2STAbits.PERR = 0;       //Bit3 *Read Only Bit*Parity error has not
been detected
    U2STAbits.FERR = 0;       //Bit2 *Read Only Bit*Framing error has not
been detected
    U2STAbits.OERR = 0;       //Bit1 *Read Only Bit*Receive buffer has
not overflowed.
    U2STAbits.URXDA = 0;      //Bit0 *Read Only Bit*Receive buffer is
empty

    IPC7bits.U2RXIP    = 3;     // UART1 Receiver Interrupt Priority bits
    IFS1bits.U2TXIF = 0;      // Clear the Transmit Interrupt Flag
    IEC1bits.U2TXIE = 0;      // Disable Transmit Interrupts
    IFS1bits.U2RXIF = 0;      // Clear the Recieve Interrupt Flag
    IEC1bits.U2RXIE = 1;      // Enable Recieve Interrupts

    U2MODEbits.UARTEN = 1;    // And turn the peripheral on
    U2STAbits.UTXEN = 1;
    PointerRx2=0;
    CaracterRx2=0;
}


/**************************FuncionTx*****************************/
void putUART1( char c)
{
      while ( U1STAbits.UTXBF);          // wait while Tx buffer full
      U1TXREG = c;
}

/*****************Interrupcion Rx USART1**************************/
void   attribute   ((interrupt, no auto psv))  U1RXInterrupt(void)
{
    ArrayRx[PointerRx]= U1RXREG;
    CaracterRx = ArrayRx[PointerRx];

    if (CaracterRx == '}')  // Final de dato recibido
    {
        // Lectura de Dato recibido, valor Setpoint del puerto Serial
        switch (ArrayRx[0])
        {
            case 'S':       // Comando de Start(Run Control PID)
                Command RunStop= TRUE;
                PointerRx=0;   //reinicia apuntador del arreglo
                LATDbits.LATD4=1;
                LATDbits.LATD5=0;
                LATDbits.LATD6=0;
                LATDbits.LATD7=0;
            break;

            case 's':       // Comando de Stop Control PID
                Command RunStop= FALSE;
                Stop PWM();
                PointerRx=0;   //reinicia apuntador del arreglo
                LATDbits.LATD4=0;
                LATDbits.LATD5=1;
                LATDbits.LATD6=0;
                LATDbits.LATD7=0;
            break;
            // Enganchar y Activa Control de Estabilizacion
            case 'E':       // Comando de enganche Control Estabilizacion
ALTAZIMUT
                Command Estabilizacion = TRUE;
                New ref= TRUE; //bandera de cambio de referencia por
LSM303DLHC
                PointerRx=0;   //reinicia apuntador del arreglo
```

```
            break;
             // Desenganchar y Desactiva Control de Estabilizacion
            case 'e':       // Comando de desenganche Control
Estabilizacion ALTAZIMUT
                Command Estabilizacion = FALSE;
                New ref= FALSE; //bandera de cambio de referencia por
LSM303DLHC
                PointerRx=0;   //reinicia apuntador del arreglo
            break;

            // PID-MOTOR1
            case 'R':       // Comando de Read(dsPIC a MATLAB) Parámetros
PID1
                printf("%3.5f",Kp);
                printf(",");
                printf("%3.5f",Ki);
                printf(",");
                printf("%3.5f",Kd);
                PointerRx=0;   //reinicia apuntador del arreglo
                LATDbits.LATD4=0;
                LATDbits.LATD5=0;
                LATDbits.LATD6=1;
                LATDbits.LATD7=0;
            break;

            // PID2-MOTOR2
            case 'r':       // Comando de Read(dsPIC a MATLAB) Para
metros PID2
                printf("%3.5f",Kp 2);
                printf(",");
                printf("%3.5f",Ki 2);
                printf(",");
                printf("%3.5f",Kd 2);
                PointerRx=0;   //reinicia apuntador del arreglo
                LATDbits.LATD4=0;
                LATDbits.LATD5=0;
                LATDbits.LATD6=1;
                LATDbits.LATD7=0;
            break;


//*****************************************************************
            //*****************  COMANDOS PID-MOTOR 1
*************************
            // Comando de CAMBIO de Setpoint Control PID-Motor1
            case 'C':
                Ascci2Decimal[0]= (int)ArrayRx[1] - 48; // Centenas
                Ascci2Decimal[1]= (int)ArrayRx[2] - 48; // Decenas
                Ascci2Decimal[2]= (int)ArrayRx[3] - 48; // Unidades

                int reg[2]=(Ascci2Decimal[0]*100) + (Ascci2Decimal[1]*10)
+ Ascci2Decimal[2];
                ref=(float)int reg[2];
                PointerRx=0;   //reinicia apuntador del arreglo
            break;
            // Comando de cambio de Kp Control PID-Motor1
            case 'P':

                for (i=1;i <= PointerRx-1; i++ )
                {
                    ReadString[i-1] = ArrayRx[i] ;
                }
                Conver2Float = atof(&ReadString); //Conversion de String
a Float
                Kp = Conver2Float;  //Cambio de variable de local a
```

```
global
                Conver2Float=0; // Limpia variable local
                PointerRx=0;   //reinicia apuntador del arreglo
            break;
            // Comando de cambio de Ki Control PID-Motor1
            case 'I':

                for (i=1;i <= PointerRx-1; i++ )
                {
                    ReadString[i-1] = ArrayRx[i] ;
                }
                Conver2Float = atof(&ReadString); //Conversion de String
a Float
                Ki = Conver2Float;  //Cambio de variable de local a
global
                Conver2Float=0; // Limpia variable local
                PointerRx=0;   //reinicia apuntador del arreglo
            break;
            // Comando de cambio de Kd Control PID-Motor1
            case 'D':

                for (i=1;i <= PointerRx-1; i++ )
                {
                    ReadString[i-1] = ArrayRx[i] ;
                }
                Conver2Float = atof(&ReadString); //Conversion de String
a Float
                Kd = Conver2Float;  //Cambio de variable de local a
global
                Conver2Float=0; // Limpia variable local
                PointerRx=0;   //reinicia apuntador del arreglo
                break;

//*****************************************************************
            //****************  COMANDOS PID2-MOTOR 2
*************************
            // Comando de CAMBIO de Setpoint, Control PID2-Motor2
            case 'c':

                for (i=1;i <= PointerRx-1; i++ )
                {
                    ReadString[i-1] = ArrayRx[i] ;
                }
                Conver2Float = atof(&ReadString); //Conversion de String
a Float
                ref 2 = Conver2Float;  //Cambio de variable de local a
global
                Conver2Float=0; // Limpia variable local
                PointerRx=0;   //reinicia apuntador del arreglo
            break;
            // Comando de cambio de Kp Control PID2-Motor2
            case 'p':

                for (i=1;i <= PointerRx-1; i++ )
                {
                    ReadString[i-1] = ArrayRx[i] ;
                }
                Conver2Float = atof(&ReadString); //Conversion de String
a Float
                Kp 2 = Conver2Float;  //Cambio de variable de local a
global
                Conver2Float=0; // Limpia variable local
                PointerRx=0;   //reinicia apuntador del arreglo
            break;
            // Comando de cambio de Ki Control PID2-Motor2
```

```
            case 'i':

                for (i=1;i <= PointerRx-1; i++ )
                {
                    ReadString[i-1] = ArrayRx[i] ;
                }
                Conver2Float = atof(&ReadString); //Conversion de String
a Float
                Ki 2 = Conver2Float;  //Cambio de variable de local a
global
                Conver2Float=0; // Limpia variable local
                PointerRx=0;   //reinicia apuntador del arreglo
            break;
            // Comando de cambio de Kd Control PID2-Motor2
            case 'd':

                for (i=1;i <= PointerRx-1; i++ )
                {
                    ReadString[i-1] = ArrayRx[i] ;
                }
                Conver2Float = atof(&ReadString); //Conversion de String
a Float
                Kd 2 = Conver2Float;  //Cambio de variable de local a
global
                Conver2Float=0; // Limpia variable local
                PointerRx=0;   //reinicia apuntador del arreglo
            break;

            default:
                PointerRx=0;   //reinicia apuntador del arreglo
                break;

        } //switch
        // Borra Variable ReadString antes de cada lectura del buffer
USART-Rx
        for(iReadString=0; iReadString <= 10;
iReadString++)ReadString[iReadString]=0;
        iReadString=0;  //Reinicia indice de arreglo String para
siguiente lectura
    }// if(CaracterRx == '}')
    else
    {
        PointerRx++;
        if(PointerRx >= 20)PointerRx=0;   //reinicia apuntador del
arreglo
    }
    //Renicia pointer para limpiar CR(carrier return) que envia MATLAB
    if(CaracterRx == 13)PointerRx=0;   //reinicia apuntador del arreglo
    IFS0bits.U1RXIF = 0;    //-Clear the Recieve Interrupt Flag
}

/***************Interrupcion Rx  USART2*****************************/
void   attribute   ((interrupt, no auto psv))  U2RXInterrupt(void)
{
    ArrayRx2[PointerRx2]= U2RXREG;
    CaracterRx2 = ArrayRx2[PointerRx2];
//    putUART1(CaracterRx2); //Prueba de eco de datos recibidos por
Arduino
    // Decodificacion de valores Raw del Sensor LSM303DLHC enviados por
Arduino
    // A-> Recibido inicio de valores del Acelerometro
    // M-> Recibido inicio de valores del Compass
    if(CaracterRx2 == '}') //Indica Final de la trama
    {
        j=1; //Inicia indice para lectura del arreglo
```

```
        jReadString=0;
        if(ArrayRx2[j-1] == 'A')
        {
            //Lectura del Valor Ax de Acelerometro
            while(ArrayRx2[j] != ',')
            {
                ReadString 2[jReadString] = ArrayRx2[j] ;
                j++;
                jReadString++;
            }//fin while
            Conver2Float 2 = atof(&ReadString 2); //Conversion de String
a Float
            Ax = Conver2Float 2;  //Cambio de variable de local a global
            Conver2Float 2=0; // Limpia variable local
            j++; //Incrementa apuntador para lectura de siguiente dato
            // Borra Variable ReadString 2 antes de cada lectura del
buffer USART2-Rx
            for(jReadString=0; jReadString <= 6;
jReadString++)ReadString 2[jReadString]=0;
            jReadString=0;  //Reinicia indice de arreglo String para
siguiente lectura

            //Lectura del Valor Ay de Acelerometro
            while(ArrayRx2[j] != ',')
            {
                ReadString 2[jReadString] = ArrayRx2[j] ;
                j++;
                jReadString++;
            }//fin while
            Conver2Float 2 = atof(&ReadString 2); //Conversion de String
a Float
            Ay = Conver2Float 2;  //Cambio de variable de local a global
            Conver2Float 2=0; // Limpia variable local
            j++; //Incrementa apuntador para lectura de siguiente dato
            // Borra Variable ReadString 2 antes de cada lectura del
buffer USART2-Rx
            for(jReadString=0; jReadString <= 6;
jReadString++)ReadString 2[jReadString]=0;
            jReadString=0;  //Reinicia indice de arreglo String para
siguiente lectura

            //Lectura del Valor Az de Acelerometro
            while(ArrayRx2[j] != ',')
            {
                ReadString 2[jReadString] = ArrayRx2[j] ;
                j++;
                jReadString++;
            }//fin while
            Conver2Float 2 = atof(&ReadString 2); //Conversion de String
a Float
            Az = Conver2Float 2;  //Cambio de variable de local a global
            Conver2Float 2=0; // Limpia variable local
            j++; //Incrementa apuntador para lectura de siguiente dato
            // Borra Variable ReadString 2 antes de cada lectura del
buffer USART2-Rx
            for(jReadString=0; jReadString <= 6;
jReadString++)ReadString 2[jReadString]=0;
            jReadString=0;  //Reinicia indice de arreglo String para
siguiente lectura
        }// Fin if(ArrayRx2[j-1] == 'A')

        if(ArrayRx2[j] == 'M')
        {
            j++;
            //Lectura del Valor Mx de Compass
```

```
            while(ArrayRx2[j] != ',')
            {
                ReadString 2[jReadString] = ArrayRx2[j] ;
                j++;
                jReadString++;
            }//fin while
            Conver2Float 2 = atof(&ReadString 2); //Conversion de String
a Float
            Mx = Conver2Float 2;  //Cambio de variable de local a global
            Conver2Float 2=0; // Limpia variable local
            j++; //Incrementa apuntador para lectura de siguiente dato
            // Borra Variable ReadString 2 antes de cada lectura del
buffer USART2-Rx
            for(jReadString=0; jReadString <= 6;
jReadString++)ReadString 2[jReadString]=0;
            jReadString=0;  //Reinicia indice de arreglo String para
siguiente lectura

            //Lectura del Valor My de Compass
            while(ArrayRx2[j] != ',')
            {
                ReadString 2[jReadString] = ArrayRx2[j] ;
                j++;
                jReadString++;
            }//fin while
            Conver2Float 2 = atof(&ReadString 2); //Conversion de String
a Float
            My = Conver2Float 2;  //Cambio de variable de local a global
            Conver2Float 2=0; // Limpia variable local
            j++; //Incrementa apuntador para lectura de siguiente dato
            // Borra Variable ReadString 2 antes de cada lectura del
buffer USART2-Rx
            for(jReadString=0; jReadString <= 6;
jReadString++)ReadString 2[jReadString]=0;
            jReadString=0;  //Reinicia indice de arreglo String para
siguiente lectura

            //Lectura del Valor Mz de Compass
            while(ArrayRx2[j] != '}')
            {
                ReadString 2[jReadString] = ArrayRx2[j] ;
                j++;
                jReadString++;
            }//fin while
            Conver2Float 2 = atof(&ReadString 2); //Conversion de String
a Float
            Mz = Conver2Float 2;  //Cambio de variable de local a global
            Conver2Float 2=0; // Limpia variable local
            // Borra Variable ReadString 2 antes de cada lectura del
buffer USART2-Rx
            for(jReadString=0; jReadString <= 6;
jReadString++)ReadString 2[jReadString]=0;
            jReadString=0;  //Reinicia indice de arreglo String para
siguiente lectura
         } //Fin if(ArrayRx2[j] == 'M')
        PointerRx2=0;   //reinicia apuntador del arreglo

        // Borra Variable ReadString 2 antes de cada lectura del buffer
USART2-Rx
        for(jReadString=0; jReadString <= 10; jReadString++)
        {
            ReadString 2[jReadString]=0;
        }// Fin for(jReadString=0; jReadString <= 10; jReadString++)

        // Borra Variable ArrayRx2 antes de lectura siguiente buffer
```

```
USART2-Rx
        for(j=0; j <= 40; j++)
        {
            ArrayRx2[j]=0;
        }// Fin for(j=0; j <= 40; j++)
        j=0;  //Reinicia valor del indice

    }//Fin if verificacion de final de trama '}'

    else
    {
        if(CaracterRx2 == 'A')PointerRx2 = 0;
        PointerRx2++;
        if(PointerRx2 >= 50)PointerRx2=0;   //reinicia apuntador del
arreglo
    }
    IFS1bits.U2RXIF = 0;    // Limpia bandera interrupccion USART2
}
```

ANEXO C.5. PWM_Module.c

```
#include "p33FJ128MC706A.h"
#include "Init Module.h"

void Init PWM1(void)
{
   /*******************INICIALIZA DRIVE DE FET*************************/
   LATEbits.LATE0=0;        // Carga valor módulo PWM1L
   LATEbits.LATE1=0;        // Carga valor módulo PWM1H
   LATEbits.LATE2=0;        // Carga valor módulo PWM2L
   LATEbits.LATE3=0;        // Carga valor módulo PWM2H
   /*********************CONFIGURACION MÓDULO
PWM1***********************/
   /**** P1TCON: PWM Time Base Control Register ****/
   P1TCONbits.PTEN   = 0;   // Timer Enable bit.15:          DISABLE
MCPWM
   P1TCONbits.PTSIDL = 0;   // Stop in Idle Mode bit.13:           NO
   P1TCONbits.PTOPS  = 0;   // 0000 = 1:1 Output Clock Postscale
bits<7:4>
   P1TCONbits.PTCKPS = 1;   // 01 = PWM time base input clock period is 4
TCY (1:4 prescale)bits<3:2>
   P1TCONbits.PTMOD  = 2;   // 10 = PWM time base operates in a
Continuous Up/Down Counting mode

   /********CEREBOT PERIFERICO VMOT1**********************************/
   /********** PWM1 CONTROL REGISTER   ************/
   PWM1CON1bits.PMOD1 = 0;  // 0 = PWM I/O pin pair is in the
Complementary Output mode bits<11:8>
   PWM1CON1bits.PEN1H = 1;  // 1 = PWM1H pin is enabled for PWM output
   PWM1CON1bits.PEN1L = 1;  // 1 = PWM1L1 pin is enabled for PWM output

   P1OVDCONbits.POVD1L =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator
   P1OVDCONbits.POVD1H =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator

   //P1TCONbits.PTEN   = 1;   // Timer Enable bit:          ENABLE MCPWM
   /**** P1TPER: PWM Time Base Period Register ****/
   //P1TPER= FCY/ (Fpwm * P1TMR prescaler*2)  -1
   //FCY=40MIPS, Fpwm=20,000Hz, P1TMR prescaler=1:4
```

```
   //PTPER= 40M / (20k*8) -1 = 249
   P1TPERbits.PTPER = 249;  // Period Value bits bits<14:0>
   P1DC1 = PTPER;           // PDC1= PTPER ciclo util del 50%,efecto
parar motor

   /********CEREBOT PERIFERICO VMOT2**********************************/
   /********** PWM1 CONTROL REGISTER   ************/
   PWM1CON1bits.PMOD2 = 0;  // 0 = PWM I/O pin pair is in the
Complementary Output mode bits<11:8>
   PWM1CON1bits.PEN2H = 0;  // 0 = PWM1H2 pin disabled; I/O pin becomes
general purpose I/O
   PWM1CON1bits.PEN2L = 0;  // 0= PWM1L2 pin disabled; I/O pin becomes
general purpose I/O

   P1OVDCONbits.POVD2L =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator
   P1OVDCONbits.POVD2H =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator

   P1DC2 = PTPER;           // PDC2= PTPER ciclo util del 50%,efecto
parar motor

   P1TCONbits.PTEN   = 1;   // Timer Enable bit:          ENABLE MCPWM
}

//PWM2 Moludo
void Init PWM2(void)
{
   /******************INICIALIZA DRIVE DE FET*************************/
   LATEbits.LATE4=0;        // Carga valor módulo PWM3L
   LATEbits.LATE5=0;        // Carga valor módulo PWM3H
   LATEbits.LATE6=0;        // Carga valor módulo PWM4L
   LATEbits.LATE7=0;        // Carga valor módulo PWM4H
   /*********************CONFIGURACION MÓDULO
PWM1***********************/
   /**** P1TCON: PWM Time Base Control Register ****/
   PTCONbits.PTEN   = 0;   // Timer Enable bit.15:          DISABLE MCPWM
   PTCONbits.PTSIDL = 0;   // Stop in Idle Mode bit.13:           NO
   PTCONbits.PTOPS  = 0;   // 0000 = 1:1 Output Clock Postscale bits<7:4>
   PTCONbits.PTCKPS = 1;   // 01 = PWM time base input clock period is 4
TCY (1:4 prescale)bits<3:2>
   PTCONbits.PTMOD  = 2;   // 10 = PWM time base operates in a Continuous
Up/Down Counting mode

   /********CEREBOT PERIFERICO VMOT3**********************************/
   /********** PWM1 CONTROL REGISTER   ************/
   PWMCON1bits.PMOD3 = 0;  // 0 = PWM I/O pin pair is in the
Complementary Output mode bits<11:8>
   PWMCON1bits.PEN3H = 1;  // 1 = PWM3H pin is enabled for PWM output
   PWMCON1bits.PEN3L = 1;  // 1 = PWM3L pin is enabled for PWM output

   P1OVDCONbits.POVD3L =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator
   P1OVDCONbits.POVD3H =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator

   /**** P1TPER: PWM Time Base Period Register ****/
   //P1TPER= FCY/ (Fpwm * P1TMR prescaler*2)  -1
   //FCY=40MIPS, Fpwm=20,000Hz, P1TMR prescaler=1:4
   //PTPER= 40M / (20k*8) -1 = 249
   PTPERbits.PTPER = 249;  // Period Value bits bits<14:0>
   P1DC3 = PTPER;           // PDC3= PTPER ciclo util del 50%,efecto
parar motor

   /********CEREBOT PERIFERICO VMOT4**********************************/
```

```
   /********** PWM1 CONTROL REGISTER   ************/
   PWMCON1bits.PMOD4 = 0;  // 0 = PWM I/O pin pair is in the
Complementary Output mode bits<11:8>
   PWMCON1bits.PEN4H = 0;  // 0 = PWM4H2 pin disabled; I/O pin becomes
general purpose I/O
   PWMCON1bits.PEN4L = 0;  // 0 = PWM4L2 pin disabled; I/O pin becomes
general purpose I/O

   P1OVDCONbits.POVD4L =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator
   P1OVDCONbits.POVD4H =1;  //1 = Output on PWMxHy/PWMxLy I/O pin is
controlled by the PWM generator

   P1DC4 = PTPER;           // PDC4= PTPER ciclo util del 50%,efecto
parar motor

   PTCONbits.PTEN   = 1;   // Timer Enable bit:          ENABLE MCPWM
}


// Funcion Control de Motor1 PWM1 con sentido de giro
// MCPWM con el puente H: VMOTOR1-VMOTOR2
void MCPWM1(int x)
{
    unsigned int y;
    // Direccion (+)sentido horario del Motor1
    // Polarizacion VMOTOR1(+)  VMOTOR2(-)
    if(x >= 0)  // Valor Positivo
    {
        /************CONMUTACION AL DRIVE VMOT2 -> PWM2H,
PWM2L***********/
        PWM1CON1bits.PEN1H = 0;  // 0= PWM1H1 pin disabled; I/O pin
becomes general purpose I/O
        PWM1CON1bits.PEN1L = 0;  // 0= PWM1L1 pin disabled; I/O pin
becomes general purpose I/O
        LATEbits.LATE0=1;        // Carga valor módulo PWM1L
        LATEbits.LATE1=0;        // Carga valor módulo PWM1H
        LATEbits.LATE2=0;        // Carga valor módulo PWM2L
        LATEbits.LATE3=0;        // Carga valor módulo PWM2H
        PWM1CON1bits.PEN2H = 1;  // 1= PWM1H2 pin is enabled for PWM
output
        PWM1CON1bits.PEN2L = 1;  // 1= PWM1L2 pin is enabled for PWM
output
        P1DC2 = x;  // Carga valor del Algoritmo de control en ciclo util
módulo PWM2H
    }
    // Direccion (-)sentido antihorario del Motor1
    // Polarizacion VMOTOR1(-)  VMOTOR2(+)
    else
    {
        /***********CONMUTACION AL DRIVE VMOT1 -> PWM1H,
PWM1L************/
        PWM1CON1bits.PEN2H = 0;  // 0= PWM1L2 pin disabled; I/O pin
becomes general purpose I/O
        PWM1CON1bits.PEN2L = 0;  // 0= PWM1L2 pin disabled; I/O pin
becomes general purpose I/O
        LATEbits.LATE0=0;        // Carga valor módulo PWM1L
        LATEbits.LATE1=0;        // Carga valor módulo PWM1H
        LATEbits.LATE2=1;        // Carga valor módulo PWM2L
        LATEbits.LATE3=0;        // Carga valor módulo PWM2H
        PWM1CON1bits.PEN1H = 1;  // 1= PWM1H1 pin is enabled for PWM
output
        PWM1CON1bits.PEN1L = 1;  // 1= PWM1L1 pin is enabled for PWM
output
        y= x - (x*2);            // convierte valor en positivo para
```

```
cargar al módulo PWM2H
        P1DC1 = y;       // Carga valor del Algoritmo de control en ciclo
util módulo PWM1H
     }
}

// Funcion Control de Motor2 PWM con sentido de giro
// MCPWM con el puente H: VMOTOR3-VMOTOR4
void MCPWM2(int x)
{
    unsigned int y;
    // Direccion (-)sentido horaria del Motor2
    // Polarizacion VMOTOR3(+)  VMOTOR4(-)
    if(x >= 0)  // Valor Positivo
    {
        /************CONMUTACION AL DRIVE VMOT4 -> PWM4H,
PWM4L***********/
        PWMCON1bits.PEN3H = 0;  // 0= PWM3H1 pin disabled; I/O pin
becomes general purpose I/O
        PWMCON1bits.PEN3L = 0;  // 0= PWM3L1 pin disabled; I/O pin
becomes general purpose I/O
        LATEbits.LATE4=1;       // Carga valor módulo PWM3L
        LATEbits.LATE5=0;       // Carga valor módulo PWM3H
        LATEbits.LATE6=0;       // Carga valor módulo PWM4L
        LATEbits.LATE7=0;       // Carga valor módulo PWM4H
        PWMCON1bits.PEN4H = 1;  // 1= PWM4H2 pin is enabled for PWM
output
        PWMCON1bits.PEN4L = 1;  // 1= PWM4L2 pin is enabled for PWM
output
        P1DC4 = x;              // Carga valor del Algoritmo de control
en ciclo util módulo PWM4H
    }
    // Direccion (+)sentido antihoraria del Motor2
    // Polarizacion VMOTOR3(-)  VMOTOR4(+)
    else
    {
        /***********CONMUTACION AL DRIVE VMOT3 -> PWM3H,
PWM3L************/
        PWMCON1bits.PEN4H = 0;  // 0= PWM1L2 pin disabled; I/O pin
becomes general purpose I/O
        PWMCON1bits.PEN4L = 0;  // 0= PWM1L2 pin disabled; I/O pin
becomes general purpose I/O
        LATEbits.LATE4=0;        // Carga valor módulo PWM3L
        LATEbits.LATE5=0;        // Carga valor módulo PWM3H
        LATEbits.LATE6=1;        // Carga valor módulo PWM4L
        LATEbits.LATE7=0;        // Carga valor módulo PWM4H
        PWMCON1bits.PEN3H = 1;  // 1= PWM1H1 pin is enabled for PWM
output
        PWMCON1bits.PEN3L = 1;  // 1= PWM1L1 pin is enabled for PWM
output
        y= x - (x*2);           // convierte valor en positivo para
cargar al módulo PWM2H
        P1DC3 = y;       // Carga valor del Algoritmo de control en ciclo
util módulo PWM3H
     }
}

void Stop PWM(void)
{
    /***** Secuencia de apagado del Motor1,PUENTE H DE FET:VMOT1-VMOT2
*****/
    PWM1CON1bits.PEN1H = 0;  // 0= PWM1H1 pin disabled; I/O pin becomes
general purpose I/O
    PWM1CON1bits.PEN1L = 0;  // 0= PWM1L1 pin disabled; I/O pin becomes
general purpose I/O
```

```
    PWM1CON1bits.PEN2H = 0;  // 0= PWM1L2 pin disabled; I/O pin becomes
general purpose I/O
    PWM1CON1bits.PEN2L = 0;  // 0= PWM1L2 pin disabled; I/O pin becomes
general purpose I/O
    LATEbits.LATE0=0;        // Carga valor módulo PWM1L
    LATEbits.LATE1=0;        // Carga valor módulo PWM1H
    LATEbits.LATE2=0;        // Carga valor módulo PWM2L
    LATEbits.LATE3=0;        // Carga valor módulo PWM2H

    /***** Secuencia de apagado del Motor2,PUENTE H DE FET:VMOT3-VMOT4
*****/
    PWMCON1bits.PEN3H = 0;  // 0= PWM3H1 pin disabled; I/O pin becomes
general purpose I/O
    PWMCON1bits.PEN3L = 0;  // 0= PWM3L1 pin disabled; I/O pin becomes
general purpose I/O
    PWMCON1bits.PEN4H = 0;  // 0= PWM4L2 pin disabled; I/O pin becomes
general purpose I/O
    PWMCON1bits.PEN4L = 0;  // 0= PWM4L2 pin disabled; I/O pin becomes
general purpose I/O
    LATEbits.LATE4=0;        // Carga valor módulo PWM3L
    LATEbits.LATE5=0;        // Carga valor módulo PWM3H
    LATEbits.LATE6=0;        // Carga valor módulo PWM4L
    LATEbits.LATE7=0;        // Carga valor módulo PWM4H
}
```

ANEXO C.6. INT_External.c

```
#include "p33FJ128MC706A.h"
#include "Init Module.h"

// Declaracion de funcion de interrupccion
//void   attribute  ((  interrupt  ))  INT1Interrupt(void);

/*=====================================================================
======
Init INTx OI() es usada para configurar la interrupcion externa en el Pin
RD8.
Este pin tiene conectado un comparador que determina condicion de Sobre
Corriente.
La señal es activada cuando se detecta 6A en cualquie medio Puente H.
=====================================================================
====*/
void Init INTx OI(void)
{
    INTCON2bits.INT1EP = 0;     //0 = Interrupt on positive edge
                                /*Setup INT1/RD8 pins to interupt */
                                /*on rising edge */
    IPC5bits.INT1IP = 7;    //111 = priorpidad 7 La más alta para
proteccion de Hardware
    IFS1bits.INT1IF = 0;    /*Reset INT1 interrupt flag */
    IEC1bits.INT1IE = 1;    /*Enable INT1 Interrupt Service Routine */
}
/*
 INT1Interrupt() is the INT1 interrupt service routine (ISR).
The routine must have global scope in order to be an ISR.
The ISR name is chosen from the device linker script.
*/
void   attribute  ((interrupt, no auto psv))  INT1Interrupt(void)
{
    /***** Secuencia de apagado del Motor1,PUENTE H DE FET:VMOT1-VMOT2
```

```
*****/
    PWM1CON1bits.PEN1H = 0;  // 0= PWM1H1 pin disabled; I/O pin becomes
general purpose I/O
    PWM1CON1bits.PEN1L = 0;  // 0= PWM1L1 pin disabled; I/O pin becomes
general purpose I/O
    PWM1CON1bits.PEN2H = 0;  // 0= PWM1L2 pin disabled; I/O pin becomes
general purpose I/O
    PWM1CON1bits.PEN2L = 0;  // 0= PWM1L2 pin disabled; I/O pin becomes
general purpose I/O
    LATEbits.LATE0=0;        // Carga valor módulo PWM1L
    LATEbits.LATE1=0;        // Carga valor módulo PWM1H
    LATEbits.LATE2=0;        // Carga valor módulo PWM2L
    LATEbits.LATE3=0;        // Carga valor módulo PWM2H
    /***** Secuencia de apagado del Motor2,PUENTE H DE FET:VMOT3-VMOT4
*****/
    PWMCON1bits.PEN3H = 0;  // 0= PWM3H1 pin disabled; I/O pin becomes
general purpose I/O
    PWMCON1bits.PEN3L = 0;  // 0= PWM3L1 pin disabled; I/O pin becomes
general purpose I/O
    PWMCON1bits.PEN4H = 0;  // 0= PWM4L2 pin disabled; I/O pin becomes
general purpose I/O
    PWMCON1bits.PEN4L = 0;  // 0= PWM4L2 pin disabled; I/O pin becomes
general purpose I/O
    LATEbits.LATE4=0;        // Carga valor módulo PWM3L
    LATEbits.LATE5=0;        // Carga valor módulo PWM3H
    LATEbits.LATE6=0;        // Carga valor módulo PWM4L
    LATEbits.LATE7=0;        // Carga valor módulo PWM4H
      delay32(400000000);             // 10 segundos de retardo, 400M
ciclos de reloj
    IFS1bits.INT1IF = 0;    //Clear the INT1 interrupt flag or else
                            //the CPU will keep vectoring back to the ISR
}
```

ANEXO C.7. QEI_Module.c

```
#include "p33FJ128MC706A.h"
#include "Init Module.h"

extern float Posicion Motor1;

// Inicializacion de módulo QEI del dsPIC para lectura Encoder Motor1
void Init QEI1(void)
{
    QEICONbits.UPDN     = 1;    //1= Direccion(+) del Contador de
posicion,sentido antihorario
    QEICONbits.QEIM     = 7;    //7= QEI Habilitado(x4mode) Reset
contador de posicion por(MAXCNT)
    QEICONbits.POSRES   = 0;    //0= Index pulse no restablece contador
de posición
    QEICONbits.TQCS     = 0;    //0 =Reloj Interno(TCY)
    //Registro DFLTCON: Control de Filtro Digital para QEI
    DFLTCONbits.QEOUT   = 0;    //0 = Filtro Digital
deshabilitado(operacion normal de pin)
    DFLTCONbits.QECK    = 0;    //000 = 1:1 clock divide
    POSCNT=5569;                // 5569= posicion inicial 90° con
inclinacion en 0°
    MAXCNT=7424;                // Maximo conteo del POSCNT 7424= 1vuelta
eje reductor
    Posicion Motor1=0;          // se inicializa el valor de posicion del
Encoder1-Motor1
```

```
    QEICONbits.QEIM1 = 1;       //1 =Inicia con 16-bit Timer
}

float Posicion Encoder1(void)
{
    Posicion Motor1 = 360 -(float)POSCNT * FACTOR POS ENC1;
    return Posicion Motor1;
}
```

ANEXO C.8. LSM303DLHC_Module.c

```
#include "p33FJ128MC706A.h"
#include "Init Module.h"
#include<stdlib.h>
#include<dsp.h>

//Variables del Sensor LSM303DLHC
extern float Ax, Ay, Az;    // Datos Raw del Acelerometro
extern float Mx, My, Mz;    // Datos Raw del Compass
float Acc[3];        // Cálculo de acceleracion en unidades g
float Mag[3];        // Cálculo de magnometro en Gauss
float Magn[3];       // Cálculo de magnometro normalizado
extern int Pitch;  // Cálculo de Angulo(°) de Inclinacion(tilt-Pitch)
extern int Roll;   // Cálculo de Angulo(°) de Alabeo(Roll)
extern int Heading;// Cálculo de Angulo(°) de Giro(Heading-Yaw)
compensado por inclincacion
int M minx, M miny, M minz,M maxx,M maxy,M maxz;
float Am,Mm;        //Magnitud de Acceleracion y Magnometro
float temp heading, temp pitch, temp roll, temp pitch sum;
vector a; // accelerometer readings
vector m; // magnetometer readings
vector temp a, E, N, from;

extern char Command Estabilizacion;

//Calibration values. Use the Calibrate example program to get the values
for
// your compass.
void LSM303 Data Init(void)
{
  M minx = -833; M miny = -840; M minz = -576;
  M maxx = 673; M maxy = 110; M maxz = 919;

  //shift and scale
  Mag[X] = (Mx - M minx) / (M maxx - M minx) * 2 - 1;
  Mag[Y] = (My - M miny) / (M maxy - M miny) * 2 - 1;
  Mag[Z] = (Mz - M minz) / (M maxz - M minz) * 2 - 1;
}

void LSM303 vector normalize(vector *a)
{
  float mag = sqrt(LSM303 vector dot(a,a));
  a->x /= mag;
  a->y /= mag;
  a->z /= mag;
}

void LSM303 vector cross(const vector *a,const vector *b, vector *out)
{
  out->x = a->y*b->z - a->z*b->y;
```

```
  out->y = a->z*b->x - a->x*b->z;
  out->z = a->x*b->y - a->y*b->x;
}

float LSM303 vector dot(const vector *a,const vector *b)
{
  return a->x*b->x+a->y*b->y+a->z*b->z;
}

/***************** Cálculo Pitch / Roll / Heading
*************************/
void getPitchRollHeading(void)
{
    //Normalizacion de Acceleracion rango entre 0 a 1 :
    Am = sqrt( pow(Ax,2) + pow(Ay,2)+ pow(Az,2));
    Acc[X] = Ax / Am;
    Acc[Y] = Ay / Am;
    Acc[Z] = Az / Am;

     //Guarda en estructura vector datos de Acelerometro
     a.x = Acc[X]; a.y = Acc[Y]; a.z = Acc[Z];
     temp a = a;


    //Carga valores Offset correccion hard-iron y escala Magnetometro
    Mag[X] = (Mx - M minx) / (M maxx - M minx) * 2 - 1;
    Mag[Y] = (My - M miny) / (M maxy - M miny) * 2 - 1;
    Mag[Z] = (Mz - M minz) / (M maxz - M minz) * 2 - 1;

    //Normalizacion de Magnetometro rango entre 0 a 1 :
    Mm = sqrt( pow(Mx,2) + pow(My,2)+ pow(Mz,2));
    Magn[X] = Mx / Mm;
    Magn[Y] = My / Mm;
    Magn[Z] = Mz / Mm;

    //Guarda en estructura vector datos de Magnometro
    m.x = Mag[X]; m.y = Mag[Y]; m.z = Mag[Z];
    from.x = 0; from.y = -1; from.z = 0;

    // normalize
    LSM303 vector normalize(&temp a);
    // compute E and N
    LSM303 vector cross(&m, &temp a, &E);
    LSM303 vector normalize(&E);
    LSM303 vector cross(&temp a, &E, &N);

    // compute heading
    temp heading= atan2(LSM303 vector dot(&E, &from),
LSM303 vector dot(&N, &from)) * 180 / PI;
    Heading = (int)temp heading;
    if (Heading < 0) Heading += 360;

    // Ver apendice A en nota de aplicacion AN3192, Ecuacion 10
    temp pitch = asin(-Acc[X]);
    temp roll = asin(Acc[Y]/cos(temp pitch));

    // Conversion en grados de angulo(°) de inclinacion pitch;
    temp pitch = 180 * temp pitch / PI;

    if(temp pitch > 0)
    {
        if(a.z > 0 )Pitch = 90+(int)temp pitch;//Cudrante I
        else
        {
            //Cuadrante IV
```

```
            temp pitch = abs(temp pitch);
            Pitch = 270 - (int)temp pitch;
        }
    }
    else
    {
        if(a.z < 0 )
        {
            //Cuadrante III
            temp pitch = abs(temp pitch);
            Pitch = 270 + (int)temp pitch;
        }
        else
        {
            //Cuadrante II
            Pitch = abs(temp pitch);
            Pitch = 90 + (int)temp pitch;
        }
    }

    // Cálculo de angulo de inclinacion pitch en grados(°);
    temp roll = 180 * temp roll/PI;
    Roll = (int)temp roll;
}
```

ANEXO C.9. main_Cerebot.c

```
#include "p33FJ128MC706A.h" // Include this in your MPLAB project under
"Header Files" from C:\Program Files (x86)\Microchip\MPLAB
C30\support\dsPIC30F\h
                // And include the p30f4011.gld linker script under
"Linker Script" from C:\Program Files (x86)\Microchip\MPLAB
C30\support\dsPIC30F\gld
//#include <libpic30.h>
#include <dsp.h>
#include <stdio.h>
#include "Init Module.h"

/*****************************************************
*                    Configure Fuses                 *
*****************************************************/
 FOSCSEL(FNOSC PRIPLL & IESO OFF);             // Primary (XT, HS, EC)
oscillator with PLL, Two-Speed Oscillator Disabled
 FOSC(FCKSM CSDCMD & OSCIOFNC OFF & POSCMD XT); // Disable clock monitor,
OSC2 is clock O/P, XT oscillator 8MHz
 FWDT(FWDTEN OFF);                             // Disable Watch-Dog
Timer.
 FGS(GSS OFF);                                 // Disable Code
Protection

//Variables
long anidador 10ms;
char flag 10ms;
char CaracterRx;
char CaracterRx2;
char ArrayRx[20];   //Buffer de datos USART Rx
char ArrayRx2[50];  //Buffer de datos USART2 Rx
int Ascci2Decimal[10];  //Conversion a decimal
char Command RunStop;
char Command Estabilizacion;
```

```
char New ref;
int dato adc;
float resultado;
float resultado1;
float Posicion Motor1;
float Posicion Motor2;
float Posicion Motor1 actual;
float Posicion Motor2 actual;
int int reg[9];   // memoria para registros de entrada.
int imon1, imon2, imon3, imon4; // valores de corriente aplicados al A/D
int vmon1, vmon2, vmon3, vmon4; // valores de voltaje aplicados al A/D
unsigned int BufferA[4]   attribute  ((space(dma))); // buffer A for DMA
//unsigned int BufferB[8]   attribute  ((space(dma))); // Ping pong
buffer B
int flag;                                    // Flag
int  Cont ENC2;   // Contador de Posicion Encoder2-Motor2

// Datos Raw del Sensor LSM303D
float Ax, Ay, Az;  // Datos Raw del Acelerometro
float Mx, My, Mz;  // Datos Raw del Compass
int Pitch;         // Cálculo de Angulo(°) de Inclinacion(tilt-Pitch)
int Roll;          // Cálculo de Angulo(°) de Alabeo(Roll)
int Heading;       // Cálculo de Angulo(°) de Giro(Heading-Yaw)
compensado por inclincacion

// Variables PID1- Motor1(Inclinacion)-PWM1
float ref= 90 ; // referencia
float y;        // salida del sistema
float e;        // error
float ek =0;    // suma de todos los errores
float e 1 =0;   // error del instante de tiempo anterior
float Ts =0.01; // periodo de muestreo de la señal a 10ms
float u;        // salida del controlador
float Kp = 30.03;    // 36 ganancia proporcional
float Ki = 2.5674;   // 0.1 ganancia integral
float Kd = 0.03045;  // 1 ganancia derivativa
float max, min;   //Variables para anti-windup

// Variables PID2- Motor2(Giro)-PWM2
float ref 2= 0 ;  // referencia
float y 2;        // salida del sistema
float e 2;        // error
float ek 2 =0;    // suma de todos los errores
float e 1 2 =0;   // error del instante de tiempo anterior
float u 2;        // salida del controlador
float Kp 2 = 30.03;    // ganancia proporcional
float Ki 2 = 2.5674;   //  ganancia integral
float Kd 2 = 0.03045;  // ganancia derivativa

//Setpoint filtrado de magnometro
float ref e = 0;
float ref e 1 = 0;
float ref ek = 0;
float ref k = 0.1;

//MAIN PROGRAMA PRINCIPAL
int main(void)
{
   Init IO();        // Initializa módulo Configuracion de puertos
   Init timer1();    // Inicializa timer1
   Init Usart();     // Inicia UART1 del dsPIC
   Init Usart2();    // Inicia UART2 del dsPIC
   Init PWM1();      // Inicia Módulo PWM1 para el Puente H :VMOT1-VMOT2
   Init PWM2();      // Inicia Módulo PWM2 para el Puente H :VMOT3-VMOT4
   Init INTx OI();   // Inicia Int1 externa de RD8 para proteccion de
```

```
sobrecorriente(6Amp)
   Init QEI1();      // Inicia QEI para motor1 modo 4x
   Init CN();        // Interrupcion por Cambio en entrada RC13-RC14
   LSM303 Data Init(); //Carga valores de calibracion por distorsion
hard-iron del Magnetometro
   LATDbits.LATD4=0;
   LATDbits.LATD5=1;
   LATDbits.LATD6=0;
   LATDbits.LATD7=0;

   max = 510;   //Valor maximo para cargar al Módulo MCPWM
   min = -510;  //Valor minimo para cargar al Módulo MCPWM

   while(1) // Loop forever
   {
       getPitchRollHeading();   //Cálculo de angulos de Giro y elevacion
       // EJECUCION CONTROL PID
       // Con un periodo de cada 10ms
       if(flag 10ms && Command RunStop)
       {
           flag 10ms = 0;      // baja bandera de 100ms
      //-------------------------------------------------------------
      //---------Ejecucion PID1- Motor1,PWM1(Inclinacion)-----------
      // posicion actual seleccionada desde GUI-MATLAB, se hace cambio de
      // referencia para control PID entre Encoder-SensorLSM303DLHC
           if(Command Estabilizacion)
           {
               y = (float)Pitch;
               if(New ref)//Bandera de cambio de ref a sensor LSM303DLHC
               {
                   ref = (float)Pitch;
                   //Seteamos Encoder Motor1 en Posicion actual de sensor
                   Posicion Motor1 actual = (360-ref) / 0.048491379;
                   POSCNT= (int)Posicion Motor1 actual;
                   ek=0; //Reinicia errores acumulados de la integral
               }
               else Posicion Motor1 actual = Posicion Encoder1();
           }
           else
           {
                y = Posicion Encoder1();
                Posicion Motor1 actual = y;
                e = ref - y;     // error actual
           }
           /* Registro donde se encuentra la salida
           del sistema , es decir el avance total del motor */
           //Envia posicion actual Encoder
           printf("%3.2f",Posicion Motor1 actual);
           printf(",");
           printf("%3.2f",ref); //Envia referencia a la GUI
           // regulador PID
           e = ref - y;     // error actual
           ek += e;         // acumulo todos los errores para el integral
           u = Kp*e+Ki*Ts*ek+ Kd *(e- e 1 )/Ts;// accion de control
           // Anti-wind up
           if (u >= max) u = max;
           else
           {
               if (u<min) u=min;
           }

           MCPWM1(u);            //Transferencia de senal de control al
actuador

           e 1 =e;// error para el instante anterior e(t -1)
```

```
           ref e 1= ref e;
 //-------------------------------------------------------------------
 //-------------------------------------------------------------------
 //------Ejecucion PID2- Motor2,PWM2(Giro)---------------------------
 // posicion actual
 // posicion actual seleccionada desde GUI-MATLAB, se hace cambio de
 // referencia para control PID entre Encoder-SensorLSM303DLHC
           if(Command Estabilizacion)
           {
               y 2 = (float)Heading;
               if(New ref)//Bandera de cambio de ref a sensor LSM303DLHC
               {
                   ref 2 = (float)Heading;
                   //Seteamos Encoder Motor2 en Posicion actual de sensor
                   Posicion Motor2 actual = (360 - ref 2) / 0.09693;
                   Cont ENC2= (int)Posicion Motor2 actual;
                   Posicion Motor2 actual = Posicion Encoder2();
                //Cambio de referencia desactivado mantiene ultimo valor
                   New ref = FALSE;
                   ek 2=0; //Reinicia errores acumulados de la integral
               }
               else Posicion Motor2 actual = Posicion Encoder2();
           }
           else
           {
                y 2 = Posicion Encoder2();
                Posicion Motor2 actual = y 2;
           }
           /* Registro donde se encuentra la salida
           del sistema , es decir el avance total del motor */
           printf(",");
           //Envia posicion actual Encoder
           printf("%3.2f",Posicion Motor2 actual);
           printf(",");
           printf("%3.2f",ref 2);//Envia referencia a la GUI
           // regulador PID
           e 2 = ref 2 - y 2;   // error actual
           ek 2 += e 2;   // acumulo todos los errores para el integral
           // accion de control
           u 2 = Kp 2*e 2+Ki 2*Ts*ek 2+ Kd 2 *(e 2- e 1 2 )/Ts;
           // Anti-wind up
           if (u 2 >= max) u 2 = max;
           else
           {
               if (u 2<min) u 2=min;
           }

           MCPWM2(u 2);   //Transferencia de senal de control al actuador

           e 1 2 =e 2;// error para el instante anterior e(t -1)
//-------------------------------------------------------------------
//-------------------------------------------------------------------
//---------------Envia Raw-data de LSM303LHC -----------------------
           // Inicia con envio de valores del acelerometro Ax,Ay,Az
           printf(",");
           printf("%3.2f",Ax);
           printf(",");
           printf("%3.2f",Ay);
           printf(",");
           printf("%3.2f",Az);
           // Finaliza con envio de valores del Compass Mx,My,Mz
           printf(",");
           printf("%3.2f",Mx);
           printf(",");
           printf("%3.2f",My);
```

```
            printf(",");
            printf("%3.2f",Mz);
 //-------------------------------------------------------------------------
 //******Envia valores de Giro(Heading) y Elevacion(Pitch)************
            printf(",");
            printf("%3u",Heading);
            printf(",");
            printf("%3u\r",Pitch);

//*************************************************************************
       }
   }//while
}//main
```

ANEXO D. CODIGO FUENTE DE INTERFAZ GRAFICA (GUI)

```
function varargout = Altazimut_Control_v0(varargin)
% ALTAZIMUT_CONTROL_V0 M-file for Altazimut_Control_v0.fig
```

```
%      ALTAZIMUT_CONTROL_V0, by itself, creates a new
ALTAZIMUT_CONTROL_V0 or raises the existing
%      singleton*.
%
%      H = ALTAZIMUT_CONTROL_V0 returns the handle to a new
ALTAZIMUT_CONTROL_V0 or the handle to
%      the existing singleton*.
%
%      ALTAZIMUT_CONTROL_V0('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in ALTAZIMUT_CONTROL_V0.M with the given
input arguments.
%
%      ALTAZIMUT_CONTROL_V0('Property','Value',...) creates a new
ALTAZIMUT_CONTROL_V0 or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Altazimut_Control_v0_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Altazimut_Control_v0_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Altazimut_Control_v0

% Last Modified by GUIDE v2.5 26-Sep-2013 19:12:42

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Altazimut_Control_v0_OpeningFcn,
...
                   'gui_OutputFcn',  @Altazimut_Control_v0_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Altazimut_Control_v0 is made visible.
```

```
function Altazimut_Control_v0_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Altazimut_Control_v0 (see
VARARGIN)

% Choose default command line output for Altazimut_Control_v0
handles.output = hObject;

% borrar previos puertos COM
instrreset;
clc;

% Set defaults for the program
handles = set_defaults(handles);

% Carga lista desplegable con puertos seriales disponibles
serialPorts = instrhwinfo('serial');
nPorts = length(serialPorts.AvailableSerialPorts);
set(handles.port_list, 'String', ...
    [{'Select a port'} ; serialPorts.AvailableSerialPorts ]);
set(handles.port_list, 'Value', 2);


I = imread('IMG341.jpg'); %Read the image of the system
imshow(I,'Parent',handles.picture);                %Show the image

% PREPARA FIGURAS
% Figura para Inclinacion Plataforma ALTAZIMUT
axes(handles.axes1);
set(handles.axes1,'XLim',[0 handles.data.tmax],'YLim',[0
handles.data.ymax]);
ylabel('Posicion Encoder(°),INCLINACION(Pitch)')
title('Captura de datos en tiempo real con dsPIC')
grid on
hold on
% Figura para Giro Plataforma ALTAZIMUT
axes(handles.axes2);
set(handles.axes2,'XLim',[0 handles.data.tmax],'YLim',[0
handles.data.ymax]);
ylabel('Posicion Encoder(°),GIRO(Yaw)')
xlabel('Tiempo (s)')
grid on
hold on
% Figura para Acelerometro en Ax,Ay,Az
axes(handles.axes3);
set(handles.axes3,'XLim',[0 handles.data.tmax]);
set(handles.axes3,'XTickLabel',[]);
ylabel('Acelerometro Data-RAW')
grid on
hold on
% Figura para Acelerometro en Mx,My,Mz
axes(handles.axes4);
```

```
set(handles.axes4,'XLim',[0 handles.data.tmax]);
ylabel('Magnometro Data-RAW')
xlabel('Tiempo (s)')
grid on
hold on

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Altazimut_Control_v0 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Altazimut_Control_v0_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function handles = set_defaults(handles)
% Initialize structure for saving/manipulating data the plot
handles.data.tmax = 10;      % tiempo de captura en segundos
handles.data.rate = 10;     % resultado experimental (comprobar)
handles.data.datos_x = handles.data.tmax / 0.01; %Cantidad de datos en
eje X
handles.data.ymax = 360;     % posicion encoder en grados(°) eje Y de axes
% Inicializa Variables PID1
handles.pid.setpoint = 45;
handles.pid.kp       = 25;
handles.pid.ki       = 0.1;
handles.pid.kd       = 1.3;
% Inicializa Variables PID2
handles.pid.setpoint2= 45;
handles.pid.kp2      = 25;
handles.pid.ki2      = 0.1;
handles.pid.kd2      = 1.3;


% --- Executes on button press in open_serial.
function open_serial_Callback(hObject, eventdata, handles)
% hObject    handle to open_serial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(hObject,'Enable'),'on') % currently disconnected
    serPortn = get(handles.port_list, 'Value');    %lectura COMM
seleccionado de lista
    if serPortn == 1
        errordlg('Select valid COM port');         %debe ser diferente de
1
    else
        serList = get(handles.port_list,'String'); %lectura del puerto
seleccionado
```

```
        serPort = serList{serPortn};                %carga string a
variable serPort
        serCom = serial(serPort);                   %crea objeto serial
con el COM seleccionado
        set(serCom,'BaudRate',115200);              %configuración de
puerto
        set(serCom,'DataBits',8);
        set(serCom,'Parity','none');
        set(serCom,'StopBits',1);
        set(serCom,'FlowControl','none');
        set(serCom,'InputBufferSize',8096);
        set(serCom,'OutputBufferSize',8096);
        set(serCom,'TimerPeriod',0.1);
        set(serCom,'Terminator','CR');

        try
            h = waitbar(0,'Abriendo Puerto por favor espere...');
            fopen(serCom);                          %abrir puerto
            handles.serCom = serCom;               %copia objeto serial en
estructura
            set(handles.text_message,'String',sprintf('Puerto %s
Abierto',handles.serCom.Name));  %Muestra texto en banner

            disp('Puerto COM8 abierto');            % Barra de espera
hasta que habra puerto
            steps = 1000;
            for step = 1:steps
                % computations take place here
                waitbar(step / steps)
            end
            close(h)

            %Habilita Edits Texts PID1 para enviar y leer datos del
puerto serial
            set(handles.edit_encoder, 'Enable', 'On');
            set(handles.edit_setpoint, 'Enable', 'On');
            set(handles.edit_kp, 'Enable', 'On');
            set(handles.edit_ki, 'Enable', 'On');
            set(handles.edit_kd, 'Enable', 'On');
            %Habilita Edits Texts PID2 para enviar y leer datos del
puerto serial
            set(handles.edit_encoder2, 'Enable', 'On');
            set(handles.edit_setpoint2, 'Enable', 'On');
            set(handles.edit_kp2, 'Enable', 'On');
            set(handles.edit_ki2, 'Enable', 'On');
            set(handles.edit_kd2, 'Enable', 'On');

            %Habilita los Push Buttons
            set(handles.read_pid_button, 'Enable', 'On');
            set(handles.read_pid2_button, 'Enable', 'On');
            set(handles.run_control_motor1, 'Enable', 'On');
            set(handles.stop_control_motor1, 'Enable', 'On');
            set(handles.start_control_stabilizer, 'Enable', 'On');
            set(handles.stop_control_stabilizer, 'Enable', 'On');

            enable_button = get(handles.close_serial,'Enable');
```

```
            set(handles.open_serial, 'Enable', 'Off');

            if enable_button
                set(handles.close_serial, 'Enable', 'On');
            end

        catch e
            errordlg(e.message);
        end

    end
else
    h = waitbar(0,'Cerrando Puerto por favor espere...');
    fclose(handles.serCom);
    steps = 100;
    for step = 1:steps
        % computations take place here
       waitbar(step / steps)
    end
    close(h)
    set(handles.text_message,'String','Puerto Abierto');    %Muestra
texto en banner
    set(handles.text_message,'String',sprintf('Puerto %s
Cerrado',handles.serCom.Name));  %Muestra texto en banner
    enable_button = get(handles.open_serial,'Enable');
    set(handles.close_serial, 'Enable', 'Off');

    if enable_button
        set(handles.open_serial, 'Enable', 'On');
    end
    disp('Puerto COM8 cerrado');

end

guidata(hObject, handles);

% --- Executes on button press in close_serial.
function close_serial_Callback(hObject, eventdata, handles)
% hObject    handle to close_serial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
h = waitbar(0,'Cerrando Puerto por favor espere...');
fclose(handles.serCom);
steps = 100;
for step = 1:steps
    % computations take place here
   waitbar(step / steps)
end
close(h)

%Deshabilita Edits Texts para enviar y leer datos del puerto serial
set(handles.edit_encoder, 'Enable', 'Off');
set(handles.edit_setpoint, 'Enable', 'Off');
set(handles.edit_kp, 'Enable', 'Off');
set(handles.edit_ki, 'Enable', 'Off');
```

```
set(handles.edit_kd, 'Enable', 'Off');
set(handles.edit_encoder2, 'Enable', 'Off');
set(handles.edit_setpoint2, 'Enable', 'Off');
set(handles.edit_kp2, 'Enable', 'Off');
set(handles.edit_ki2, 'Enable', 'Off');
set(handles.edit_kd2, 'Enable', 'Off');

%Deshabilita los Push Buttons
set(handles.read_pid_button, 'Enable', 'Off');
set(handles.read_pid2_button, 'Enable', 'Off');
set(handles.run_control_motor1, 'Enable', 'Off');
set(handles.stop_control_motor1, 'Enable', 'Off');

set(handles.text_message,'String',sprintf('Puerto %s
Cerrado',handles.serCom.Name));  %Muestra texto en banner
enable_button = get(handles.open_serial,'Enable');
set(handles.close_serial, 'Enable', 'Off');

if enable_button
    set(handles.open_serial, 'Enable', 'On');
end

disp('Puerto COM8 cerrado');
guidata(hObject, handles);

% --- Executes on selection change in port_list.
function port_list_Callback(hObject, eventdata, handles)
% hObject    handle to port_list (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns port_list contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
port_list

% --- Executes during object creation, after setting all properties.
function port_list_CreateFcn(hObject, eventdata, handles)
% hObject    handle to port_list (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in video_open.
function video_open_Callback(hObject, eventdata, handles)
% hObject    handle to video_open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
vid = videoinput('winvideo', 1, 'YUY2_640x480'); %Define video input
object
preview(vid);  % Preview the video object

function edit_run_time_Callback(hObject, eventdata, handles)
handles.data.tmax = get(handles.edit_run_time, 'String');
handles.data.tmax = str2double(handles.data.tmax);
set(handles.axes1,'XLim',[0 handles.data.tmax]);  %redimenciona el
handles.data.tmax de lectura de datos
set(handles.axes2,'XLim',[0 handles.data.tmax]);  %redimenciona el
handles.data.tmax de lectura de datos
set(handles.axes3,'XLim',[0 handles.data.tmax]);  %redimenciona el
handles.data.tmax de lectura de datos
set(handles.axes4,'XLim',[0 handles.data.tmax]);  %redimenciona el
handles.data.tmax de lectura de datos
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_run_time_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_run_time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in run_control_motor1.
function run_control_motor1_Callback(hObject, eventdata, handles)
% Desactiva boton Run
enable_button = get(handles.run_control_motor1,'Enable');
set(handles.run_control_motor1, 'Enable', 'Off');
%Activa boton Stop si se encuentra desactivado
if enable_button
    set(handles.stop_control_motor1, 'Enable', 'On');
end

%Envia comando de Start Control Motores
fprintf(handles.serCom,'S}');
%fprintf(handles.serCom,'}');
set(handles.text_message,'String','Running Motor Control');  %Muestra
texto en banner
pause(0.1);

axes(handles.axes1);
cla;      %Borra figura anterior
axes(handles.axes2);
cla;      %Borra figura anterior
axes(handles.axes3);
cla;      %Borra figura anterior
axes(handles.axes4);
cla;      %Borra figura anterior
```

```
% parámetros de medidas
handles.data.tmax = get(handles.edit_run_time, 'String');
handles.data.tmax = str2double(handles.data.tmax);   % tiempo de captura
en s

% inicializar
x = 0:0.01:handles.data.tmax;
Encoder = zeros(1,(handles.data.tmax * handles.data.rate));
Encoder2 = Encoder;
Giro = Encoder;
pitch = Encoder;
Ax=Encoder; Ay=Encoder; Az=Encoder; Mx=Encoder; My=Encoder; Mz=Encoder;
Setpoint = zeros(1,(handles.data.tmax * handles.data.rate));
Setpoint2 = Setpoint;
size_x = size(x);
handles.data.datos_x = size_x(1,2);

% dibujar en la figura
% l1 = line(nan,nan,'Color','r','LineWidth',2);
% l2 = line(nan,nan,'Color','b','LineWidth',2);
handles.data.stop = 0;
%Inicializa mensaje de waitbar
h = waitbar(0,'Aquiriendo Datos por favor espere...');
try
    for i=1:handles.data.datos_x
        a = fscanf(handles.serCom,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%u,%u');
        Encoder(i) = a(1);
        Setpoint(i)= a(2);
        Encoder2(i)= a(3);
        Setpoint2(i)= a(4);
        %Lectura Sensor LSM303LHC
        %Lectura de valores Raw-Acelerometro
        Ax(i) = a(5);
        Ay(i) = a(6);
        Az(i) = a(7);
        %Lectura de valores Raw-Compass
        Mx(i) = a(8);
        My(i) = a(9);
        Mz(i) = a(10);
        %Lectura de valores Giro(Heading) e Inclincacion(Pitch)
        Giro(i) = a(11);
        pitch(i) = a(12);

        set(handles.edit_encoder,'String',Encoder(i));  %Escribe valor
actual de Encoder1 en GUI
        set(handles.edit_setpoint,'String',Setpoint(i));  %Escribe valor
actual de Setpoint1 en GUI
        set(handles.edit_encoder2,'String',Encoder2(i));  %Escribe valor
actual de Encoder2 en GUI
        set(handles.edit_setpoint2,'String',Setpoint2(i));  %Escribe
valor actual de Setpoint2 en GUI
        %Escribe valores leidos en edit texts de Giro e Inclinacion
        set(handles.edit_heading,'String',Giro(i));  %Escribe valor
actual de Giro en GUI
        set(handles.edit_pitch,'String',pitch(i));  %Escribe valor actual
de Inclinacion en GUI
```

```
        waitbar(i/handles.data.datos_x);
    end

catch me
    errordlg(me.message);
    set(handles.text_message,'String','Error Adquisicion');  %Muestra
texto en banner
    disp('Stop Adquisicion');
    pause(1);
    %Envia comando de Stop Control Motor1-2
    fprintf(handles.serCom,'s}');
    close(h);%Cierre waitbar si se encuentra error
end

close(h);%Cierre waitbar al finalizar adquisicion de datos
% Crea plot en axes 1-Eje Pitch(inclinacion)
axes(handles.axes1);
hold on
plot(x,Setpoint,'-r');
plot(x,Encoder,'-b');
plot(x,pitch,'-k');
grid on
h=legend('Setpoint','Encoder','IMU','location','best');
set(h,'Interpreter','none')
grid on

% Crea plot en axes 2 - Eje Heading(Giro)
axes(handles.axes2);
hold on
plot(x,Setpoint2,'-r');
plot(x,Encoder2,'-b');
plot(x,Giro,'-k');
grid on
h=legend('Setpoint 2','Encoder 2','IMU',1);
set(h,'Interpreter','none')
grid on

% Crea plot en axes 3
axes(handles.axes3);
plot(x,Ax);
hold on
plot(x,Ay,'-r');
plot(x,Az,'-k');
h=legend('Ax','Ay','Az',1);
set(h,'Interpreter','none')
grid on

% Crea plot en axes 4
axes(handles.axes4);
plot(x,Mx);
hold on
plot(x,My,'-r');
plot(x,Mz,'-k');
h=legend('Mx','My','Mz',1);
set(h,'Interpreter','none')
grid on
```

```
%Exporta al workspace los datos recolectados en la adquisicion
assignin('base', 'Encoder', Encoder);
assignin('base', 'Encoder2',Encoder2);
assignin('base', 'Setpoint',Setpoint);
assignin('base', 'Setpoint2',Setpoint2);
assignin('base', 'Ax',Ax);
assignin('base', 'Ay',Ay);
assignin('base', 'Az',Az);
assignin('base', 'Mx',Mx);
assignin('base', 'My',My);
assignin('base', 'Mz',Mz);
assignin('base', 'Giro',Giro);
assignin('base', 'Pitch',pitch);
assignin('base', 'x',x);

set(handles.text_message,'String','Finishing the Acquisition');  %Muestra
texto en banner
disp('Finaliza Adquisicion');
set(handles.run_control_motor1, 'Enable', 'On');    %Activa boton Run
pause(2);
%Guarda Datos en Excel
if(get(handles.checkbox_log,'Value')==1)
    c = clock;
    xlswrite(['Data Motor1-' date num2str(c(4)) num2str(c(5))],[Encoder'
Setpoint']);
    set(handles.text_message,'String','Data Logged!') %Log data in xls
file
end
% %Envia comando de Stop Control Motor1-2
% fprintf(handles.serCom,'s}');
% pause(2);
guidata(hObject, handles);

% --- Executes on button press in stop_control_motor1.
function stop_control_motor1_Callback(hObject, eventdata, handles)
%Envia comando de Stop Control Motor1
fprintf(handles.serCom,'s}');
%fprintf(handles.serCom,'}');
%Muestra texto en banner
set(handles.text_message,'String','Stop Motor Control');
pause(1);

%Activa boton Stop si se encuentra desactivado
enable_button = get(handles.run_control_motor1,'Enable');
TF= strcmp(enable_button, 'Off');
if (TF == 0)
    set(handles.run_control_motor1, 'Enable', 'On');
end

guidata(hObject, handles);


function edit_encoder_Callback(hObject, eventdata, handles)
% hObject    handle to edit_encoder (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_encoder as text
%        str2double(get(hObject,'String')) returns contents of
edit_encoder as a double

% --- Executes during object creation, after setting all properties.
function edit_encoder_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_encoder (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_setpoint_Callback(hObject, eventdata, handles)
handles.pid.setpoint = get(handles.edit_setpoint, 'String');
%Envia nuevo valor de setpoint para Control Motor1
fprintf(handles.serCom,'C%s}',handles.pid.setpoint); %Finaliza trama de
datos  con }
%Muestra texto en banner
set(handles.text_message,'String',sprintf('Cambio de setpoint a
%s',handles.pid.setpoint));  %Muestra texto en banner
%Actualiza estructura de datos
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_setpoint_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_setpoint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_kp_Callback(hObject, eventdata, handles)
% Lectura string del Edit Text del Kp
handles.pid.kp = get(handles.edit_kp, 'String');

%Envia nuevo valor de Kp para Control Motor1
fprintf(handles.serCom,'P%s}',handles.pid.kp);

%Muestra texto en banner
```

```
set(handles.text_message,'String',sprintf('Cambio de Kp a %s
',handles.pid.kp));  %Muestra texto en banner

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_kp_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_ki_Callback(hObject, eventdata, handles)
% Lectura string del Edit Text del Ki
handles.pid.ki = get(handles.edit_ki, 'String');

%Envia nuevo valor de Ki para Control Motor1
fprintf(handles.serCom,'I%s}',handles.pid.ki);

%Muestra texto en banner
set(handles.text_message,'String',sprintf('Cambio de Ki a %s
',handles.pid.ki));  %Muestra texto en banner

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_ki_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit_kd_Callback(hObject, eventdata, handles)
% Lectura string del Edit Text del Kp
handles.pid.kd = get(handles.edit_kd, 'String');

%Envia nuevo valor de Kp para Control Motor1
fprintf(handles.serCom,'D%s}',handles.pid.kd);

%Muestra texto en banner
set(handles.text_message,'String',sprintf('Cambio de Kd a %s
',handles.pid.kd));  %Muestra texto en banner

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
```

```
function edit_kd_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in read_pid_button.
function read_pid_button_Callback(hObject, eventdata, handles)
h = waitbar(0,'Lectura de Parámetros PID-1,MOTOR1 por favor espere...');
% Lectura string del Edit Text del Kp,Ki,Kd
handles.pid.kp = get(handles.edit_kp, 'String');
handles.pid.ki = get(handles.edit_ki, 'String');
handles.pid.kd = get(handles.edit_kd, 'String');

%Envia peticion de lectura parametros PID-MOTOR1 del dsPIC
fprintf(handles.serCom,'R}');

%Lectura del puerto serial
R = fscanf(handles.serCom,'%f,%f,%f')';
%Organizacion de parametros en su memoria en la estructura handles
handles.pid.kp = R(1);
handles.pid.ki = R(2);
handles.pid.kd = R(3);

%Ejecucion de waitbar
steps = 500;
for step = 1:steps
% computations take place here
waitbar(step / steps)
end
close(h)

%Escribe valores leidos de dsPIC en Edit Texts Kp,Ki,Kd
set(handles.edit_kp,'String',handles.pid.kp);
set(handles.edit_ki,'String',handles.pid.ki);
set(handles.edit_kd,'String',handles.pid.kd);

%Muestra texto en banner
set(handles.text_message,'String','Lectura PID1 Finalizada');  %Muestra
texto en banner

guidata(hObject, handles);

% --- Executes on button press in checkbox_log.
function checkbox_log_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox_log (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox_log
```

```
function edit_encoder2_Callback(hObject, eventdata, handles)
% hObject    handle to edit_encoder2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_encoder2 as text
%        str2double(get(hObject,'String')) returns contents of
edit_encoder2 as a double


% --- Executes during object creation, after setting all properties.
function edit_encoder2_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_setpoint2_Callback(hObject, eventdata, handles)
handles.pid.setpoint2 = get(handles.edit_setpoint2, 'String');
%Envia nuevo valor de setpoint para Control Motor2
fprintf(handles.serCom,'c%s}',handles.pid.setpoint2); %Finaliza trama de
datos  con }

%Muestra texto en banner
set(handles.text_message,'String',sprintf('Cambio de setpoint a
%s',handles.pid.setpoint2));  %Muestra texto en banner

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_setpoint2_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_ki2_Callback(hObject, eventdata, handles)
% Lectura string del Edit Text del Ki
handles.pid.ki2 = get(handles.edit_ki2, 'String');

%Envia nuevo valor de Ki para Control Motor2
fprintf(handles.serCom,'i%s}',handles.pid.ki2);

%Muestra texto en banner
set(handles.text_message,'String',sprintf('Cambio de Ki a %s
',handles.pid.ki2));  %Muestra texto en banner

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_ki2_CreateFcn(hObject, eventdata, handles)
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_kd2_Callback(hObject, eventdata, handles)
% Lectura string del Edit Text del Kd
handles.pid.kd2 = get(handles.edit_kd2, 'String');

%Envia nuevo valor de Kp para Control Motor2
fprintf(handles.serCom,'d%s}',handles.pid.kd2);

%Muestra texto en banner
set(handles.text_message,'String',sprintf('Cambio de Kd a %s
',handles.pid.kd2));  %Muestra texto en banner

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function edit_kd2_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in read_pid2_button.
function read_pid2_button_Callback(hObject, eventdata, handles)
h = waitbar(0,'Lectura de Parámetros PID-2,MOTOR2 por favor espere...');
% Lectura string del Edit Text del Kp2,Ki2,Kd2
handles.pid.kp2 = get(handles.edit_kp2, 'String');
handles.pid.ki2 = get(handles.edit_ki2, 'String');
handles.pid.kd2 = get(handles.edit_kd2, 'String');

%Envia peticion de lectura parametros PID2-MOTOR2 del dsPIC
fprintf(handles.serCom,'r}');

%Lectura del puerto serial
r = fscanf(handles.serCom,'%f,%f,%f')';
%Organizacion de parametros en su memoria en la estructura handles
handles.pid.kp2 = r(1);
handles.pid.ki2 = r(2);
handles.pid.kd2 = r(3);

%Ejecucion de waitbar
steps = 500;
for step = 1:steps
% computations take place here
waitbar(step / steps)
end
```

```
close(h)

%Escribe valores leidos de dsPIC en Edit Texts Kp2,Ki2,Kd2
set(handles.edit_kp2,'String',handles.pid.kp2);
set(handles.edit_ki2,'String',handles.pid.ki2);
set(handles.edit_kd2,'String',handles.pid.kd2);

%Muestra texto en banner
set(handles.text_message,'String','Lectura PID2 Finalizada');  %Muestra
texto en banner

guidata(hObject, handles);


function edit_kp2_Callback(hObject, eventdata, handles)
% Lectura string del Edit Text del Kp
handles.pid.kp2 = get(handles.edit_kp2, 'String');

%Envia nuevo valor de Kp para Control Motor2
fprintf(handles.serCom,'p%s}',handles.pid.kp2);

%Muestra texto en banner
set(handles.text_message,'String',sprintf('Cambio de Kp a %s
',handles.pid.kp2));  %Muestra texto en banner

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function edit_kp2_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%funcion Redimenciona el eje Y de los Axes1- Axes2
function edit_axesY_encoder_Callback(hObject, eventdata, handles)
handles.data.ymax = get(handles.edit_axesY_encoder, 'String');
handles.data.ymax = str2double(handles.data.ymax);
set(handles.axes1,'YLim',[0 handles.data.ymax]);  %redimenciona el
handles.data.ymax de lectura de datos
set(handles.axes2,'YLim',[0 handles.data.ymax]);  %redimenciona el
handles.data.ymax de lectura de datos
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_axesY_encoder_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function edit_heading_Callback(hObject, eventdata, handles)
% hObject    handle to edit_heading (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_heading as text
%        str2double(get(hObject,'String')) returns contents of
edit_heading as a double


% --- Executes during object creation, after setting all properties.
function edit_heading_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_heading (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit_pitch_Callback(hObject, eventdata, handles)
% hObject    handle to edit_pitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_pitch as text
%        str2double(get(hObject,'String')) returns contents of edit_pitch
as a double


% --- Executes during object creation, after setting all properties.
function edit_pitch_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_pitch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in start_control_stabilizer.
function start_control_stabilizer_Callback(hObject, eventdata, handles)
%Envia comando de Start Control de Estabilizacion con sensor LSM303DLHC
fprintf(handles.serCom,'E}');
%Muestra texto en banner
set(handles.text_message,'String','Activo Control de Estabilizacion');
pause(1);

% Desactiva boton Start
enable_button = get(handles.start_control_stabilizer,'Enable');
set(handles.start_control_stabilizer, 'Enable', 'Off');
%Activa boton Stop si se encuentra desactivado
if enable_button
    set(handles.stop_control_stabilizer, 'Enable', 'On');
end
guidata(hObject, handles);  %Actualiza estructura de datos

% --- Executes on button press in stop_control_stabilizer.
function stop_control_stabilizer_Callback(hObject, eventdata, handles)
%Envia comando de Stop Control de Estabilizacion con sensor LSM303DLHC
fprintf(handles.serCom,'e}');
%Muestra texto en banner
set(handles.text_message,'String','Desactivado Control de
Estabilizacion');
pause(1);

%Activa boton Start si se encuentra desactivado
enable_button = get(handles.start_control_stabilizer,'Enable');
TF= strcmp(enable_button, 'Off');
if (TF == 0)
    set(handles.start_control_stabilizer, 'Enable', 'On');
end
guidata(hObject, handles); %Actualiza estructura de datos
```