VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SUBSISTEMAS DE ESTABILIZACIÓN ACTIVA Y SENSORES PARA UN SIMULADOR SATELITAL

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO ELÉCTRICO - ELECTRÓNICO

P R E S E N T A :

JIMÉNEZ MADRIGAL EMILIO AUGUSTO

DIRECTOR DE TESIS:

DR. ESAÚ VICENTE VIVAS

México D.F. 2009

Agradecimientos

A Dios por permitirme llegar hasta este punto y levantarme de la mano en tantos días oscuros para impulsarme a seguir.

A la Universidad Nacional Autónoma de México y la Facultad de Ingeniería por haberme permitido formar parte de este recinto que es la máxima casa de estudios en México y todas las oportunidades que me brindaron para ser no solo un buen profesionista si no también un buen ser humano.

A mis padres por su enorme apoyo, comprensión y educarme a lo largo de toda mi vida ya que no solo es que a través de sus virtudes sino también a través de sus defectos es hoy puedo ser un mejor ser humano.

A mis hermanos por su compañía y apoyo.

A toda mi familia que está y los que ya se han ido por todos los alientos, el apoyo y el cariño a lo largo de mi carrera y mi vida.

A mis compañeros de proyecto Francisco Gómez, Zaira Carrizales, Rodrigo Alva, Rodrigo Córdova, Roy Rodríguez, Alejandro Sánchez G., Genaro Islas y Carlos Sánchez S. por su amistad, sus enseñanzas, su compañía y apoyo a lo largo de todo este tiempo, y deseándoles que tengan todo el éxito en sus vidas.

A todos los amigos y compañeros que tuve el honor de conocer a lo largo de mi vida y carrera como Félix Lezama, Rodrigo Torres, Arturo Fuentes, José Luis Hernández, Santa Centeno, Miguel Mejía, Angélica Piliado, Sergio López, Joseba Marcos, Carmen González, Alejandro Carbajal, Diana Delgadillo, Felipe Rincón, Lucero Massé, Anahí Navarro, Julio López, Manuel Herrera, Raúl Resendez, Alejandro Rosales, Ronay Grajales, Anabel Martínez, Alonso Basurto (pollo), Cristian De Jesús (borrego), Carlos R., Antonio Aguilar (Tony), Ricardo Moreno, Josué Sánchez (Johny), Gerardo Aguilar, Seiji Osada, Carlos Moreno, Erick Hernández, Tania Rosas, Andrés Sánchez, y a tantos, tantos otros amigos y compañeros con los que pude convivir y aprender tanto.

A mis profesores de la Facultad de Ingeniería por cuya labor desinteresada y amor a la UNAM pude aprender tanto e hicieron mas grande el amor que le tengo a la ingeniería.

A mi asesor el Dr. Esaú Vicente Vivas por el apoyo y la oportunidad de participar dentro de este proyecto.

" Las almas más grandes son capaces de los mayores vicios, como de las mayores virtudes;
y los que andan muy despacio pueden llegar mucho más lejos,
si van siempre por el camino recto, que los que corren, pero se apartan de él"
René Descartes

Índice

Resumen

Se presentan los desarrollos de los subsistemas de estabilización activa y sensores para un simulador satelital como una propuesta a ser utilizados como parte de un sistema que pueda ser utilizado como satélite educativo para la formación de recursos humanos en el área satelital.

Cada subsistema posee un microcontrolador de manera que los subsistemas sean capaces de realizar algunas tareas a través de la recepción de comandos que provienen de la computadora de vuelo del simulador satelital o de una PC.

El subsistema de estabilización es capaz de controlar a una rueda inercial y 6 bobinas de torque magnético, de manera que es capaz de poder orientarse sobre un eje a través de la rueda inercial y sobre tres ejes a través de las bobinas de torque magnético, y posee un sistema que lo protege de una falla en el motor para evitar el consumo de corriente excesivo.

El subsistema de sensores está constituido por una brújula electrónica, un acelerómetro triaxial y 3 giróscopos capaces de tomar lecturas de su orientación actual y tomar una serie de lecturas para describir su movimiento en tiempo real.

Se presenta el diseño y la metodología del montaje de los circuitos impresos para ambas tarjetas así como de una de una descripción de los componentes electrónicos que las componen.

Se describen las herramientas de programación utilizadas en los microcontroladores así como el formato y contenido de los comandos que fueron programados dentro de los microcontroladores de ambos subsistemas y el programa desarrollado para el envío de comandos a los subsistemas desde una PC para realizar pruebas individuales de cada subsistema sin necesidad de la computadora de vuelo.

Se presentan las pruebas que se realizaron en protoboard y en las tarjetas impresas para la depuración software y la corrección de errores en las tarjetas impresas.

Capítulo 1

Orígenes del simulador satelital (SIMSAT) para su uso como Satélite Educativo (SATEDU)

1.1 Introducción

Desde la década de los 80's México ha incursionado en las áreas de tecnologías satelitales, lanzando en aquella década los sistemas Morelos 1 y 2, siguiendo en los 90s con los sistemas Solidaridad 1 y 2, y Satmex 5 (Morelos 3), señalando que éste último fue el primer satélite comercial mexicano lanzado mediante financiamientos privados por la empresa estatal Telecomm (Telecomunicaciones de México). Todos estos satélites han sido construidos fuera de nuestro país con tecnología extranjera, y aunque han sido satélites de clase mundial, paradójicamente no son obras mexicanas y hasta el momento no se ha podido colocar un satélite de tecnología mexicana en el espacio que sirva como propulsor para promover el desarrollo de tecnología mexicana.

Hasta el momento se han tenido grandes experiencias dentro del área de desarrollo de satélites experimentales, una de estas fue el microsatélite experimental SATEX desarrollado por diversas instituciones, entre ellas, el Instituto de Ingeniería de la UNAM, proyecto iniciado en 1995. La finalización de las actividades encargadas al Instituto de Ingeniería fue en 2005 cuando se dieron por terminados los sistemas de la computadora de vuelo, el subsistema de sensores, protocolos de comunicaciones, el software de vuelo y el software de estación terrena encargado de monitorear el satélite. Desafortunadamente, el satélite nunca fue lanzado debido a que no fue completamente terminado.

Figura 1.1 Proyecto SATEX.

Entre otras experiencias que ha tenido México hasta el momento se encuentran: los Unamsats A y B, el proyecto Unamsat III, el nanosatélite Pumasat y otras iniciativas en proceso de gestación en otras dependencias educativas Mexicanas.

Todos los proyectos anteriores han sido muy importantes para México, ya que significan el inicio de una propuesta para iniciar de manera más intensa el desarrollo tecnológico del país con fin de reducir la brecha tecnológica que existe entre México y los países de primer mundo. Pero, para realizar tal línea de trabajo se necesita de capacitar una gran cantidad de recursos humanos en esta área tecnológica, ya que de las experiencias anteriores se ha notado que se necesitan de varios años de desarrollo, además de decenas de personas trabajando en diferentes partes del proyecto.

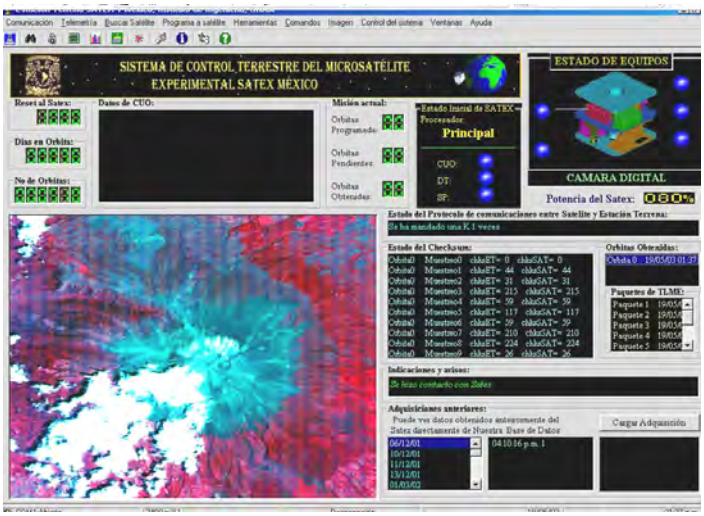


Figura 1.2
Proyecto SATEX
Software de estación terrena.

Analizando los problemas que representa la capacitación de personal en el área de satélites, es que se concibe la idea de crear un simulador satelital (SIMSAT) con el fin de crear un satélite educativo (SATEDU), para capacitar a los recursos humanos necesarios dentro del área de satélites de manera general o especifica dentro de las diferentes partes de logística que requiere la creación de un satélite.

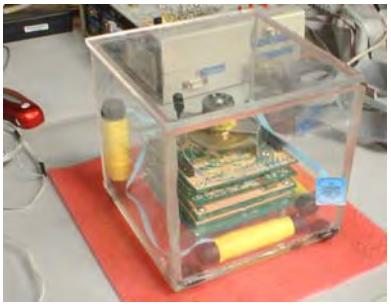


Figura 1.3
Maqueta SATEDU.

Se pretende entonces que el prototipo sea económico (USD$2000 o alrededor de $27,000 M.N.), flexible y versátil para utilizarlo en laboratorios o cursos de entrenamiento y aprendizaje (atractivos, modulares y rápidos) en el campo satelital. De este modo, se persigue desarrollar un sistema que permita acercar a las nuevas generaciones al apasionante mundo de los satélites, del espacio y de las tecnologías de la información, y en general para apoyar la difusión de la ciencia y la tecnología en México.

SATEDU estará constituido fundamentalmente por un pequeño prototipo satelital totalmente instrumentado (en un volumen de 1.5 a 2 dm^3) además de software interactivo que correrá en computadoras personales, para supervisar y controlar inalámbricamente al prototipo.

El propósito de este capítulo es ofrecer una visión general de todos los sistemas que lo componen, algunas de las características de cada uno y las posibilidades y ventajas de su uso para difundir el conocimiento del área satelital. Cabe aclarar que aunque posee con varios de los subsistemas con los que cuenta un satélite pequeño real este no cuenta con los componentes electrónicos adecuados para este propósito.

1.2 Las conveniencias de desarrollar SATEDU

Actualmente, varios países se han dado cuenta de la trascendencia que tiene el crear sistemas que permitan capacitar recursos humanos o incentivar el desarrollo de satélites en las nuevas generaciones. Un par de ejemplos de esto se dan en la fuerza área y marina de los Estados Unidos, donde se han creado pequeños sistemas que emulan a un satélite real de manera genérica, todo esto dentro de los últimos 5 años llegando incluso a comercializar su producto.

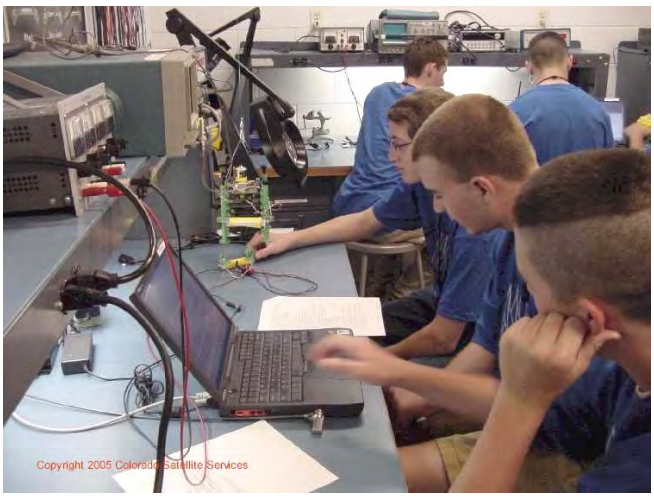


Figura 1.4 Laboratorio de la Fuerza Aérea de EE.UU.

La idea de generar esta clase de sistema ya se había concebido en el Instituto de Ingeniería a raíz del la creación de SATEX, solo que no había cristalizado debido a la falta de presupuesto para realizar el proyecto.

Además de la principal conveniencia que es el crear recursos humanos en el área satelital, también representa una oportunidad a la que pueden acceder los estudiantes para desarrollar investigación debido a lo versátil que llega a ser este sistema. Algunos de ellos son: la simulación de una constelación de satélites que se comunican entre sí, la depuración de programas de manera general emulando a un sistema real, pruebas de estabilización a partir de un modelo de cuerpo rígido en ambientes sin fricción, además de que en este proyecto en particular ha sido posible desarrollar en paralelo tanto un simulador como un sistema real usando la misma tarjeta electrónica en varios de los subsistemas de un satélite, solo faltando los componentes electrónicos adecuados.

1.3 Similitudes entre un sistema satelital real y SATEDU

Como se había mencionado en el punto anterior el proyecto comprende desarrollos en paralelo de varios subsistemas para ser idénticos tanto en SATEDU como en un sistema satelital real, de modo que dentro de estos subsistemas se encuentran: la computadora de vuelo (CV) y el subsistema de potencia (SP) la única diferencia la representa la electrónica que se usa, una es de clasificación militar mientras que la usada para SATEDU es comercial.

Así ambas versiones funcionan de la misma manera, solo que están hechas para trabajar dentro de ambientes muy distintos, además de que no solo consiste en diferencias de fabricación de componentes electrónicos sino también algunos circuitos que le fueron agregados para proteger componentes de alta escala de integración cuya fabricación no era de calificación espacial o militar. Como en el caso de la CV y el SP proteger a sus microcontroladores del efecto “latch-up” que consiste en la creación de corrientes parásitas a través del semiconductor haciendo que este consuma mucha corriente hasta destruirse. Este circuito en SATEDU puede o no utilizarse ya que en un laboratorio este efecto no se presenta y por ello puede omitirse.

1.4 Arquitectura propuesta para SATEDU

Lo primero que hubo que definir en el proceso para diseñar el SATEDU fue el tamaño que tendrían las placas de circuito impreso donde se colocaría toda la electrónica que le daría vida al satélite. En vista de los avances hasta hoy hechos en materia electrónica, y por cuestiones de movilidad fue adoptado el tamaño de un picosatélite que no debe de rebasar las dimensiones de 10x10x10 cm en cuanto a la estructura de un sistema real, pero los circuitos impresos deben ser de menor tamaño, y dado que el más importante es la computadora de vuelo esta placa fue la base para definir el tamaño de las demás, y este quedó definido en un tamaño de 8.9 x 8.9 cm.

La forma adoptada para colocar los circuitos impresos se basó en la estructura de SATEX donde los impresos se apilaban uno sobre otro, interconectados entre si a través de conectores de terminales largas.

En cuanto a estructura, como SATEDU es solo un simulador, no es necesario que esta sea un cubo de las proporciones de un picosatélite, y es por eso que se adoptó un contenedor plástico muy común, y una plataforma donde el SATEDU podrá girar en 3 ejes.

De esta forma el SATEDU queda conformado por estructura, computadora de vuelo (CV), subsistema de potencia (SP), subsistema de comunicaciones (SC), subsistema de estabilización (SE), subsistema de sensores de estabilización (SSE) y espacio para próximos desarrollos.

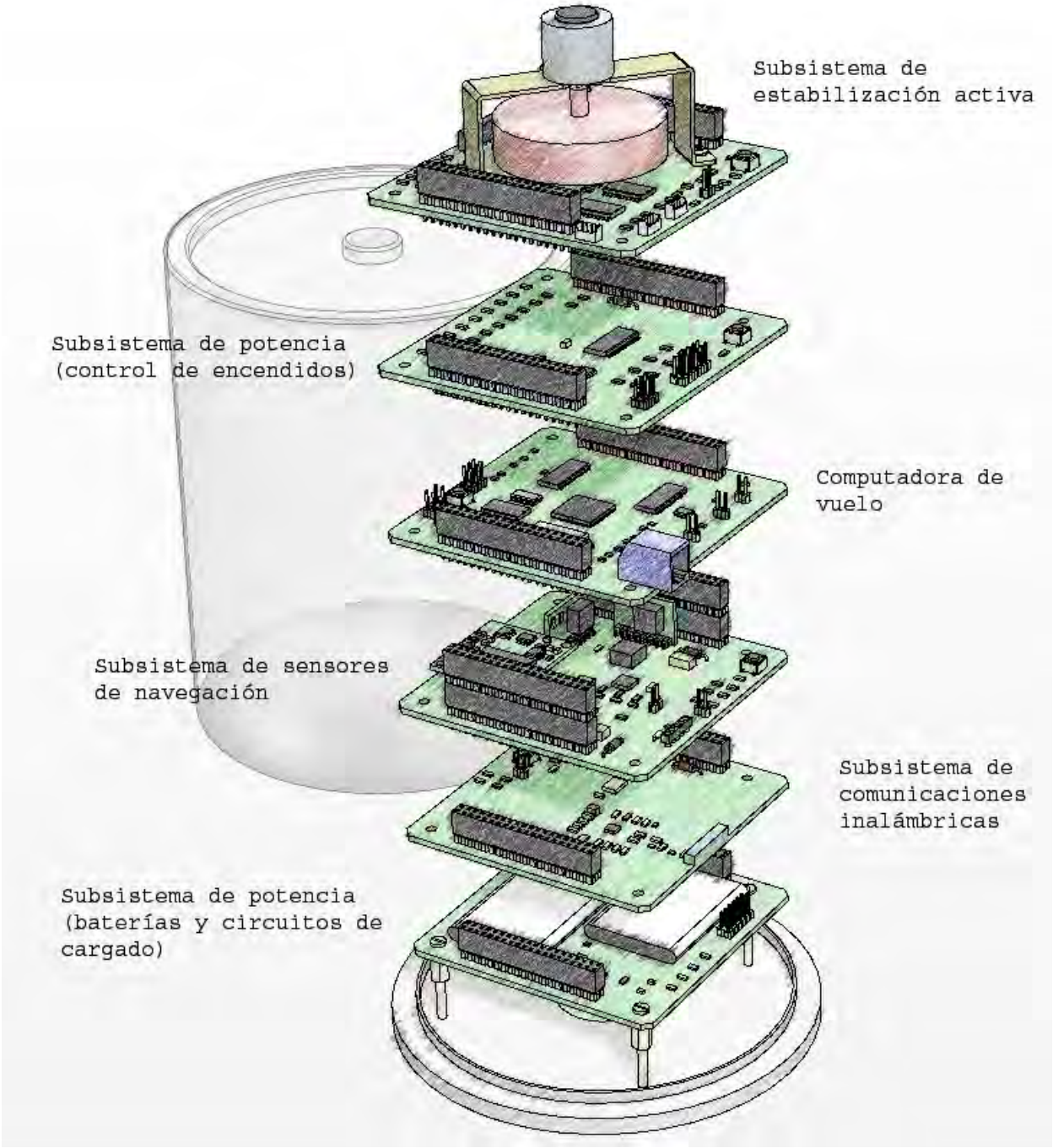


Figura 1.5 Prediseño de SATEDU hecho en Solid Edge V18

1.4.1 Subsistema estructural

El subsistema estructural está pensado en dos partes, la primera la parte que sostendrá a los circuitos impresos que componen los subsistemas del SATEDU y la otra que sostendrá a la primera estructura para maniobrarla, la primera estructura pensada fue un sistema de anillos tipo giroscopio o sistema de anillos basculantes.

La estructura que contendrá a los circuitos impresos en principio fue pensada en realizarse de acrílico, pero al ver que este tipo de material representa muchos cuidados al momento de su construcción, se optó por una estructura ya hecha y que es barata y fácil de encontrar, un envase para discos compactos, donde los circuitos impresos se fijarán a la estructura mediante tornillos.

En cuanto a la estructura que permitirá sostener al contenedor del SATEDU de tipo giroscopio fue desechada debido a lo difícil que resultaba su construcción y balanceo, por lo que la idea más simple hasta el momento ha sido optar por simplemente suspender al simulador por medio de hilo a partir de su estructura. Sin embargo, solo es posible hacer simulaciones en dos ejes por lo que si se deseara realizar experimentos más serios sobre estabilización satelital convendría usar otras plataformas como son equipos con balero de aire en donde es posible simular condiciones de aproximadamente cero fricción y es posible usar los tres ejes.

1.4.2 Computadora de vuelo

La CV está montada en un circuito impreso de 2 capas de 8.9 x 8.9 cm, y para poder comunicarse con las demás tarjetas se colocaron dos conectores en lados opuestos haciendo que cada tarjeta pueda montarse sobre la otra y conectarse al mismo bus. Adicionalmente para darle más firmeza a la estructura se le agregaron espacios para tornillos en sus 4 esquinas y también con el fin de que puedan fijarse a la estructura.

Particularmente la tarjeta de la CV está pensada para ser compatible con un sistema de un picosatélite real, es por eso que toda la electrónica fue pensada para soportar el ambiente espacial agregando arreglos de protección contra efecto “latch-up” que son inútiles dentro del simulador, por lo que tales espacios pueden quedar sin circuitos.

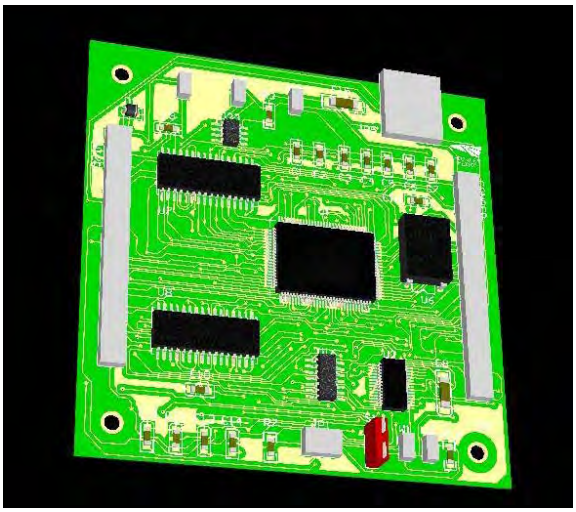


Figura 1.6 Modelo 3D del circuito impreso de la CV hecho en Protel DXP

La CV esencialmente está compuesta por un procesador RISC (del inglés, Reduced Instruction Set Computer, computadora con conjunto de instrucciones reducido) SAB80C166 de Siemens, con oscilador de 40 MHz, 256 kb de memoria RAM donde se cargan los programas de operación del SATEDU, un microcontrolador PIC16F876A de

Microchip, que es el procesador que sirve para cargar un nuevo programa al SAB, 32 MB de memoria flash, 3 sensores de temperatura y 3 arreglos de latch-up.

1.4.3 Subsistema de Potencia (SP)

El SP está constituido por 2 tarjetas, en una de ellas se monta toda la electrónica, esta tarjeta también se persigue usarla en un sistema real por lo que tiene una serie de circuitos adicionales para protegerla del efecto “latch-up” y asegurar su encendido al momento de liberar el satélite, además de los interruptores que requiere un sistema real, que es un interruptor para remover antes del vuelo y el llamado “kill switch” que mantiene totalmente apagado al sistema antes de su liberación. La segunda tarjeta básicamente contiene las baterías y circuitos complementarios.

El SP principalmente está constituido por un microcontrolador PIC18F2321 de Microchip, que se comunica con la CV para controlar los apagados y encendidos de las fuentes de energía de los demás subsistemas, una serie de reguladores y un convertidor de DC-DC que surtirá a los subsistemas con energía, un sistema de recargado para las baterías y las fuentes de energía primaria y secundaria del SATEDU que son 4 baterías Li-On recargables y celdas solares autoadheribles.

1.4.4 Subsistema de Comunicaciones (SC)

El SC consiste en 2 tarjetas, un pequeño módulo conectado a la computadora donde se encuentra el software para manipular al SATEDU y la tarjeta de comunicaciones que sigue el mismo diseño que las demás. El uso de este subsistema es de uso exclusivo para laboratorio porque para un sistema real se necesita de otra clase de sistema de comunicaciones.

Figura 1.7 Forma de operación del Subsistema de Comunicaciones

Ambas tarjetas poseen esencialmente la misma base de construcción, un transceptor de RF que trabaja en la banda de los 2.4 GHz y un microcontrolador PIC18F2321 de Microchip, la diferencia radica en que el módulo para la PC usa un transceptor adicional que es utilizado para comunicarse con la PC vía USB debido a que

como también se requiere movilidad, las PCs portátiles actualmente carecen en su mayoría de un puerto serial tradicional.

1.4.5 Subsistema de Estabilización (SE)

El SE consiste de una tarjeta que maneja dos métodos de estabilización activa, que son una rueda inercial (momentum y reacción) y bobinas de torque magnético (BTM). La rueda inercial es una pequeña masa circular sujeta a un pequeño motor mientras que las BTM son embobinados puestos uno en cada eje para que en conjunto con la rueda inercial, sea posible mover el SATEDU y poder llevar a cabo varios experimentos sobre estabilización.

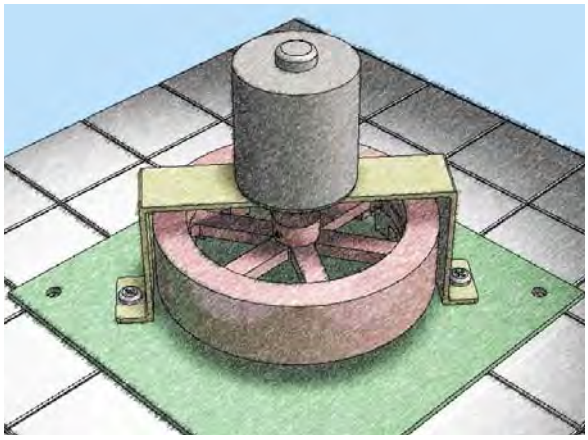


Figura 1.8 Bosquejo preliminar del sistema de estabilización

Los componentes y desarrollo de este subsistema serán discutidos más a fondo dentro de esta tesis.

1.4.6 Subsistema de Sensores de Estabilización (SSE)

El SSE es una tarjeta que aloja los sensores necesarios para monitorear el movimiento del SATEDU y algunos de ellos servirán para visualizar su posición en tiempo real. Los sensores que la componen son una brújula electrónica con una precisión de 0.1°, un acelerómetro, y 3 giróscopos electrónicos puestos de manera ortogonal para monitorear los 3 ejes del satélite.

Los componentes y desarrollo de este subsistema serán discutidos más a fondo dentro de esta tesis.

1.4.7 Subsistemas de nuevos desarrollos (carga útil)

El SATEDU ha sido pensado para poder desarrollar expansiones que permitan adaptarse al mismo sistema sin necesidad de hacer cambios al circuito impreso de la CV, debido a que se han dejado los espacios pertinentes dentro de los conectores laterales para que estos nuevos subsistemas puedan comunicarse con la CV.

Uno de los nuevos desarrollos pensados actualmente es la de implementar una pequeña cámara que sea capaz de almacenar fotografías en la memoria flash de la CV, y enviar estos archivos a una computadora que simula ser la estación terrena.

1.4.8 Segmento terrestre para supervisión y mando de SATEDU

Este segmento de supervisión comprende a una computadora donde se encuentra la interfaz de comunicación entre el SATEDU y la computadora, un sistema a través del cual es posible desplegar la telemetría del SATEDU, envío de programas de cargado en la CV y visualización de telemetría en tiempo real sobre su posición.

1.5 La necesidad de incluir medios de estabilización activa para SATEDU

En un proyecto anterior, el microsatélite experimental SATEX, los medios de estabilización eran bobinas de torque magnético y un “boom”, que son medios de estabilización muy lentos y limitados, se tenía calculado que la estabilización del mismo después de haber sido lanzado hubiera tomado días y hasta semanas antes de poder utilizar sus cargas útiles.

Por eso la necesidad de empezar a conocer y experimentar con medios de estabilización más rápidos y eficientes como es el caso de ruedas inerciales de momentum/reacción, que llegan a dar precisión de 0.1° de precisión en la posición deseada y llega a lograrse en solo cuestión de minutos y hasta horas si el movimiento requerido es muy extenso. Los medios de estabilización son de vital importancia, para surtir de energía al satélite con la mayor eficiencia orientando las celdas solares, así como permitir que los experimentos realicen sus tareas y establecer comunicaciones directivas con la Tierra.

Otro medio de estabilización más rápido es la llamada tobera de propulsión (thruster), que es un propulsor a base de gas, este sistema de estabilización por su costo y complejidad no ha sido incluido en el SATEDU.

1.6 La reducción de costos como base global de diseño

En SATEDU se busca emular de manera casi total un satélite real, pero con la diferencia de que está hecho con partes comerciales, a diferencia de un sistema real que está hecho con partes especializadas de uso militar o aeroespacial de un costo muy elevado.

Aunque actualmente la empresa Colorado Satellite Services, comercializa un sistema satelital educativo, este tiene un costo considerablemente elevado que es de USD$7995, este costo es debido en principal medida a su manufactura, ya que mientras este está hecho con circuitos impresos que son multicapa, tiene una estructura de acrílico, espaciadores de las tarjetas hechos a la medida y un subsistema de comunicaciones comercial, SATEDU está compuesto por circuitos impresos que solo son de dos capas, una estructura barata y fácil de reemplazar, un subsistema de comunicaciones de desarrollo propio y además cabe destacar que algunos de estos subsistemas pueden ser utilizados dentro de un sistema satelital real.

Esto hace que aunque SATEDU sea más barato y un sistema más valioso por la cantidad de desarrollos que tiene además de sus posibilidades de expansión.

Bibliografía

1. Colorado Satellite Service, EEUU. "*EyasSat The Classroom Satellite"*. [http://www.eyassat.com]
2. Ortíz, H. "*Implantación de técnicas de tolerancia a fallas, ensamble y validación operativa de la computadora de vuelo del microsatélite experimental SATEX1"*. Tesis de Licenciatura. Facultad de Ingeniería, UNAM. México, 2003.
3. Vivas, E. et al. "*Steps towards the development of a portable and cost-effective system for human resources training in small satellite technology"*. Revista: Research On Computer Science, Control, Virtual Instrumentation and Digital Systems, Vol. 24, pp.117-130, ISSN: 1870-4069, Centro de Investigación en Computación, IPN, México, DF, Noviembre de 2006.

Capítulo 2

Capítulo 2

Subsistema de estabilización activa por rueda inercial

2.1 Introducción

Los medios de estabilización son un área crítica en el desarrollo de un satélite. Desde que éste es liberado en el espacio se ve sometido a perturbaciones externas y debe de obtener y mantener una orientación adecuada en la tarea que le es asignada como puede ser: la comunicación con Tierra apuntando sus antenas directivas hacia la estación terrena; mantenerse en una posición estable cuando su carga útil es una cámara que debe apuntar de manera precisa a su objetivo en Tierra o algún otro punto; o bien, si es necesario que los paneles siempre estén apuntando al sol en un determinado momento.

En el presente capítulo se muestra parte del subsistema de estabilización del SIMSAT, en particular el que está compuesto por una rueda inercial. Este subsistema se emplea en muchos satélites para fijar su posición ya sea sobre uno o más ejes dependiendo de las necesidades del satélite y su tamaño. De igual forma, el subsistema permite que un satélite realice maniobras de posicionamiento operativo, dependiendo del eje sobre el que se encuentre la rueda inercial. Debido a que el SIMSAT es un satélite pequeño y didáctico, sus necesidades de estabilización no son exigentes, por ello resulta suficiente una rueda inercial para cubrir sus necesidades ante la mayoría de los experimentos que pudiera contemplar. El propósito de este capítulo es demostrar la influencia de la rueda inercial sobre el SIMSAT, así como describir el método de control para la rueda inercial.

2.2 Técnicas de estabilización de satélites

Dentro del área satelital hay 2 métodos de estabilización, los pasivos y los activos, estos se describen a continuación.

2.2.1 Estabilización pasiva

Los métodos pasivos no emplean energía eléctrica continua o de un medio de control por parte del sistema satelital. Aunque simples y baratos, los medios de estabilización pasiva tienen varias carencias entre ellas el alcance en su precisión, que es de unos cuantos grados. Al respecto se debe considerar que varias aplicaciones como los telescopios espaciales requieren de una precisión de menos de unos cuantos arco-segundos (1 arco-segundo = 1 / 3600 grados). En segundo lugar se tiene que los esquemas basados en

sistemas pasivos no pueden ser usados para desempeñar maniobras muy grandes de control de giro.

En estos casos la estabilización se alcanza naturalmente a través de las propiedades físicas de los cuerpos y su movimiento. Dentro de los principales métodos pasivos de estabilización se encuentran la estabilización por gradiente gravitacional, por giro o rotación, y otro bastante empleado en pequeños satélites es el uso de barras de torque magnético.

Estabilización por gradiente de gravedad: Está basado en el torque de balanceo natural debido al diferencial de gravedad en dos puntos de un cuerpo a distancias diferentes del centro de la Tierra, figura 2.1. Es una forma particularmente efectiva para estabilizar estructuras alargadas en una órbita terrestre baja donde la fuerza de atracción gravitatoria de la Tierra es más fuerte. El resultado de este método de estabilización es mantener la dimensión más larga de la estructura sobre el eje vertical que apunta en dirección al centro de la Tierra, manteniendo así la cara de un satélite siempre mirando hacia la Tierra.

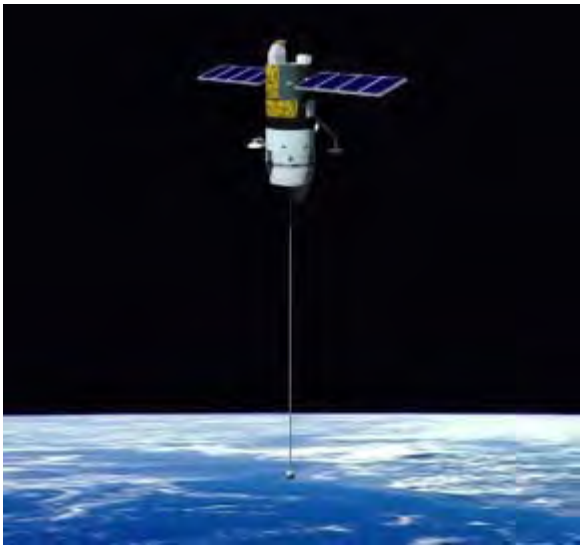


Figura 2.1 Estabilización por gradiente de gravedad

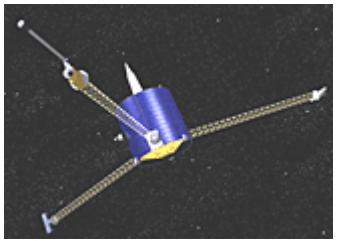


Figura 2.2 Estabilización por rotación

Estabilización por rotación (spin): Este método toma ventaja de la tendencia natural del vector de momento angular para permanecer inercialmente fijo en ausencia de fuerzas externas. El término de rigidez giroscópica es usado frecuentemente para describir esta propiedad del vector de momento angular. El trompo de un niño está basado en el mismo principio. La estabilización por rotación tiende a mantener al eje de rotación y al vector de momento angular paralelos. Esto asegura que el eje de rotación permanezca inercialmente fijo. Si el eje de rotación y el vector de momento angular no son paralelos, se dice que el cuerpo experimenta *nutación*, la cual se manifiesta como un movimiento de bamboleo.

Ambos métodos son excluyentes para controlar la posición de un cuerpo alrededor del vector de gravedad o el eje de rotación, esa es su principal desventaja, pero se pueden usar pares de ellos como son los rotadores dobles que estabilizan al satélite haciendo que no gire continuamente, pero ahora la desventaja radica en que el sistema

se hace grande y complejo, además de que no puede ofrecer una gran precisión de apuntamiento, además de que no pueden ser usados de manera efectiva para desempeñar maniobras largas de estabilización.

2.2.2 Estabilización activa

Los métodos de estabilización activa son aquellos que requieren del uso de actuadores para mover un satélite, y que por tanto requieren de un consumo de energía, sin depender de otros factores para su estabilización. Algunos de los actuadores más utilizados se describen enseguida.

Rueda inercial: Esta consiste de una masa circular usada para contener o transferir momentum angular, este término está referido exclusivamente a la rueda excluyendo sus aditamentos extra como es la electrónica. Hay varios sistemas que hacen uso de las ruedas inerciales como sistemas de posicionamiento como son:

a) *Rueda de reacción:* Este sistema está constituido por una rueda inercial y la electrónica de control. Está unido a la estructura del satélite, diseñado para operar con cero momentum angular y es capaz de girar en ambos sentidos, para hacer que el satélite gire en un sentido u otro.

b) *Rueda de momentum angular diferente de cero:* Este sistema es similar al de la rueda de reacción, tiene básicamente los mismos componentes, pero este sistema se mantiene siempre en movimiento para mantener un momentum angular en una dirección fija, haciendo al satélite un poco tolerante a perturbaciones.

c) *Giroscopio de control de momentos (CMG, del inglés Control Moment Gyro):* Este dispositivo está constituido por una rueda inercial, 2 motores, y la electrónica de control. Este sistema tiene la particularidad que se encuentra unido a un anillo, llamado cardan, en el que puede girar, este anillo es perpendicular al momento que produce el giro del motor. El eje del anillo se encuentra unido a otro motor que hace rotar al motor ubicado en el anillo, figura 2.3. Esto hace que el momento de inercia producido por el giro del motor en el anillo no sea fijo y sea posible cambiar el momento para crear un movimiento en el cuerpo. Este sistema se ha probado en un satélite turco en conjunto con el Centro Espacial de Surrey, en Inglaterra, figura 2.4.

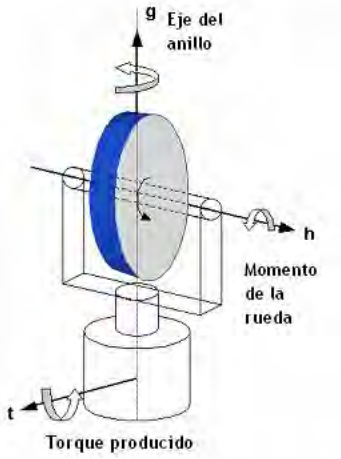


Figura 2.3 Giroscopio de control de momentos

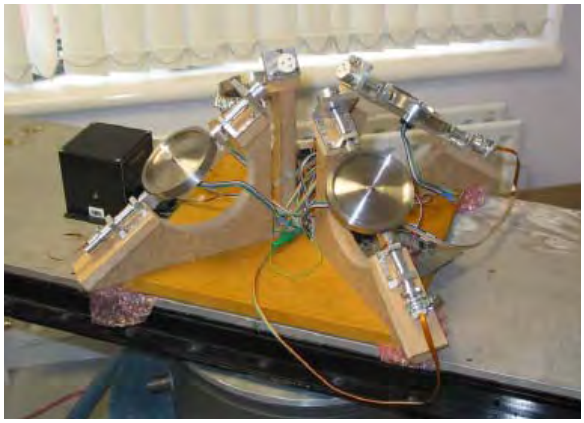


Figura 2.4 Prototipo de sistema de estabilización satelital por CMGs desarrollado en Surrey, Inglaterra

Bobinas de torque magnético: Este sistema en particular es bastante efectivo para un satélite en orbitas bajas del planeta debido a que el campo magnético es más fuerte a baja altura. Está constituido por bobinas por las que se hacen pasar pulsos de corriente en presencia del campo magnético del planeta, lo cual genera movimiento en el cuerpo sobre el que están sujetas las bobinas, este sistema será abordado con más detalle en el capítulo siguiente.

Toberas de propulsión (thrusters): Las toberas de propulsión, figura 2.5, son los medios más comunes para maniobras de un satélite en el espacio, principalmente satélites de gran tamaño que se encuentran en órbitas muy altas donde el campo magnético es casi nulo. Este sistema es capaz de mover al satélite con gran rapidez y fuerza. Como desventajas se tienen el consumo de combustible para ejercer la propulsión y el hecho de que se necesita colocar más de un propulsor en cada eje de control para prevenir fallas en puntos singulares, en cuyo caso el satélite se quedaría sin movimiento en ese eje.

Figura 2.5 Propulsor de gas frío CGMT-000-9

2.3 Estabilización por medio de ruedas inerciales

El subsistema de estabilización activa a través de una rueda inercial nace de la necesidad de un satélite de mantener una posición fija alrededor de un eje específico y tolerar perturbaciones externas que eviten cumplir con cierta operación. Estas operaciones como ya se ha comentado, pueden ser ya sea la comunicación directiva con una estación terrena, la maximización del aprovechamiento de los paneles solares para que capten luz solar de forma óptima, o bien el posicionamiento fijo al capturar imágenes con una cámara que se encuentre montada sobre el satélite.

Algunos satélites emplean tres o más ruedas inerciales para controlar su orientación y la de su carga útil, como en el caso de una cámara digital que debe obtener una imagen o serie de imágenes de un objetivo dado. Esta clase de configuración se usa en satélites de varios tamaños (figura 2.6) dependiendo de la capacidad de potencia que se tenga. Estos sistemas se utilizan en donde no es factible emplear otros sistemas de menor capacidad de momento para compensar perturbaciones externas, como es el caso de satélites que poseen telescopios y cámaras multiespectrales a bordo.

La rueda inercial empleada en el SATEDU puede ser usada como rueda de reacción o rueda con momentum angular diferente de cero.

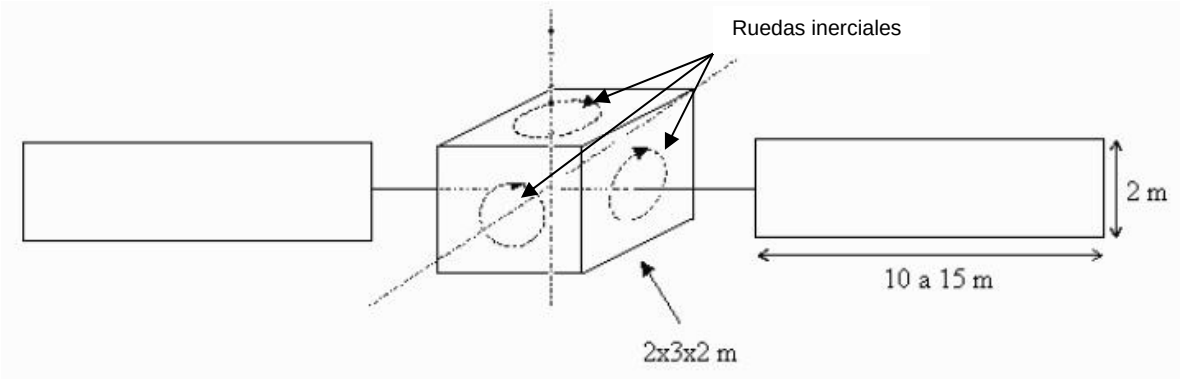


Figura 2.6 Satélite de comunicaciones estabilizado por tres ejes

En el caso del satélite educativo solo se tiene una rueda inercial y por tanto solo se puede controlar un eje, es decir, se podrá girar al satélite en el sentido deseado y sobre el eje en que se encuentra la rueda. Cabe señalar que el SIMSAT girará en sentido opuesto al que lo hace la rueda debido a la transferencia de momento angular.

2.3.1 Función de la rueda inercial como sistema de orientación

Como ya se había mencionado antes la rueda inercial se puede usar de varias formas para constituir un sistema de apuntamiento, para que esta funcione como sistema de posición bidireccional por lo que aquí se ejemplifica a la rueda para su uso como sistema de reacción, como se apuntaba antes este sistema consiste en accionar la rueda en sentido u otro partiendo del reposo de modo que al acelerar la rueda hasta una velocidad deseada esta generará una fuerza (figura 2.7), como nos lo indica la segunda ley de Newton que originalmente dice que *"el cambio de movimiento es proporcional a la fuerza motriz impresa y ocurre según la línea recta a lo largo de la cual aquella fuerza se imprime"* y se expresa como: $\vec{F} = m\vec{a}$, esta ley que aunque enuncia que es aplicada a trayectorias lineales también puede ser extrapolada movimiento circulares uniformes como es el caso de la rueda. Al generar una fuerza en la rueda también se generará un torque, este torque a su vez generará un torque de reacción sobre el satélite en el cual está anclado el subsistema de estabilización.

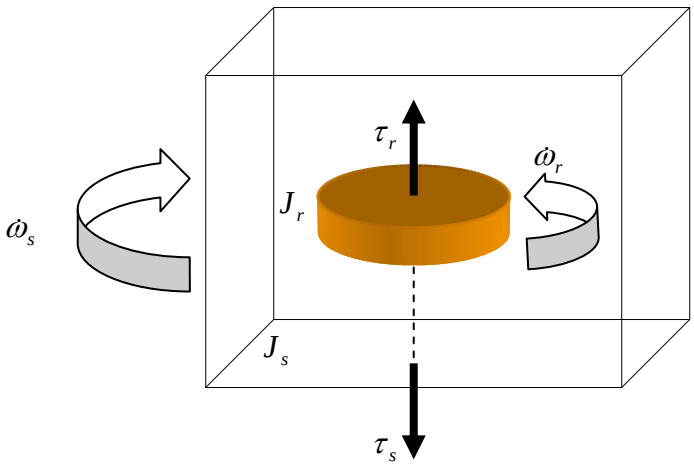


Figura 2.7 Rueda de reacción

Este torque de reacción puede interpretarse como otro par de fuerzas que actúa en sentido contrario como lo enuncia la tercera ley de Newton que dice *"a toda acción corresponde siempre una reacción de la misma magnitud y dirección pero en un sentido opuesto"*. El torque sobre la rueda también puede expresarse como:

$$\tau_r = J_r \dot{\omega}_r$$

Donde:

τ_r = torque de reacción en la rueda
J_r = inercia de la rueda
$\dot{\omega}_r$ = aceleración angular de la rueda

Y por la tercera ley de Newton es posible decir que:

$$\tau_r = -\tau_s \quad \text{ó} \quad \tau_s = -J_s \dot{\omega}_s$$

Donde:

τ_s = torque del satélite
J_S = inercia del satélite
$\dot{\omega}_s$ = aceleración angular del satélite

Y es así que es posible estimar la velocidad angular con la que se requiera mover todo el satélite o bien a través de los sensores de navegación, específicamente los giróscopos, es posible también calcular los pares de control necesarios para contrarrestar las posibles perturbaciones que pudieran llegar a actuar sobre el satélite, de manera que si por ejemplo, una fuerza actuara sobre un extremo del satélite y lo hiciera girar a la izquierda, el motor tendría que girar la rueda inercial en el mismo sentido para que el torque o par de control anule al par perturbador.

2.3.2 Efecto giroscópico

Para conocer los efectos de una rueda inercial funcionando como un sistema de rueda de reacción con momentum angular diferente de cero sobre el SIMSAT se requiere saber como trabaja un giroscopio. Los giroscopios pueden ser objetos muy peculiares por la forma en que trabajan, pareciendo que desafían a la gravedad. Esta propiedad es muy importante y se utiliza para que funcione desde una bicicleta hasta un avanzado sistema de navegación de una aeronave. Es así que un avión típico emplea alrededor de una docena de giróscopos en los instrumentos de navegación, que en estos casos se usan como sistemas de medición.

Uno de los efectos visuales más interesantes de los giroscopios se presenta cuando se pone a girar uno de ellos. Posteriormente se le puede balancear incluso en la punta de un dedo, observando que se resiste al movimiento alrededor del eje sobre el que giran. Sin embargo, su efecto más interesante es el llamado de precesión. Este efecto es la parte que parece desafiar la gravedad cuando se usa una bicicleta sin que ésta caiga de lado.

En general, la precesión trabaja de la siguiente manera, si se tiene un giroscopio rotando y se trata de mover su eje de rotación, el giroscopio tratará de girar en torno a un eje perpendicular al eje de la fuerza aplicada como se ve en la figura 2.8.

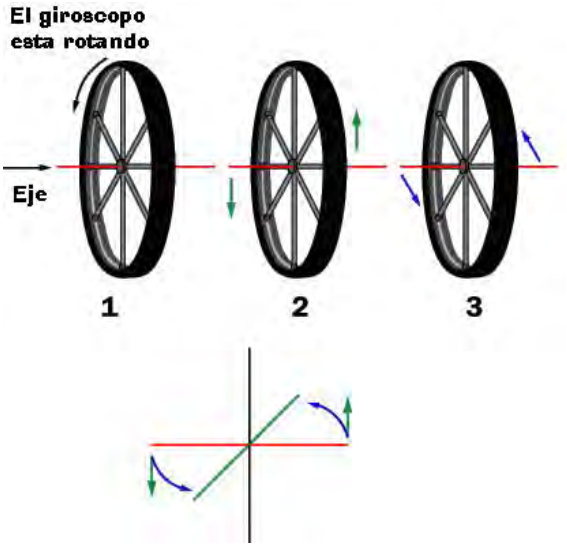


Figura 2.8

(1) El giroscopio gira en su eje
(2) Una fuerza es aplicada para tratar de mover el eje de rotación
(3) El giróscopo reacciona a la fuerza a lo largo de un eje perpendicular a la dirección de la fuerza aplicada

El por que de este comportamiento, es como sigue. Si se toma al giroscopio de 2 de sus extremos como se ve en la figura 2.9, se verá que cuando se aplica una fuerza al eje, se producirá un momento en sus extremos (flechas en los extremos superior e inferior de la rueda).

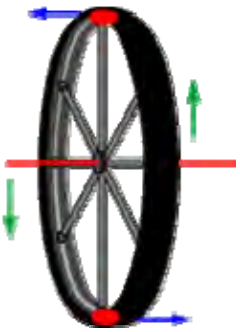


Figura 2.9 Las fuerzas son aplicadas al eje, los puntos rojos trataran de moverse como se indica

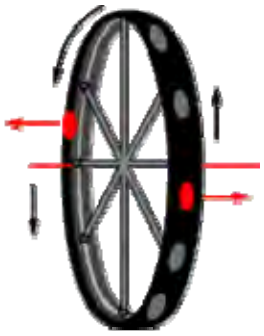


Figura 2.10 Los puntos continúan con su movimiento

Si el giroscopio no estuviera girando, la rueda cedería ante la fuerza aplicada a su eje. En la precesión, los puntos de la rueda reciben la misma fuerza en un punto, pero al estar girando la rueda estos puntos cambian de posición y cuando giran 90° a partir del punto donde la fuerza fue aplicada, continúan en su inercia de moverse hacia la izquierda y lo mismo pasa con el otro punto solo que este sigue en su inercia de moverse a la derecha como se ve en la figura 2.10. De modo que al girar los puntos hasta 180° su movimiento original se cancela de modo que el eje del giróscopo puede mantenerse en su nueva posición.

En un satélite real este efecto es el que hace que al hacer girar sus ruedas inerciales quede fijo y resistente a fuerzas perturbadoras.

2.4 Arquitectura del hardware de control de estabilización por rueda inercial

El hardware de control de estabilización está formado por un motor pequeño de DC, una rueda acoplada al motor, un codificador de posición y un microcontrolador. Las partes que se consideran en la planta son 2 básicamente: el motor y la rueda, en tanto que el codificador constituye el sistema de muestreo que monitorea la velocidad de la rueda inercial para realimentar esta velocidad al microcontrolador. Esto se logra midiendo el tiempo entre los pulsos que genera el codificador, pero al estar éste atado a la flecha del motor también será considerado como parte del modelo matemático de la planta.

La rueda inercial del subsistema de estabilización fue diseñada para montarse en el circuito impreso, figura 2.11, por lo que se dejaron un par de espacios en la placa para montar el sistema de soporte del motor. A su vez, el motor se atornilla al soporte indicado.

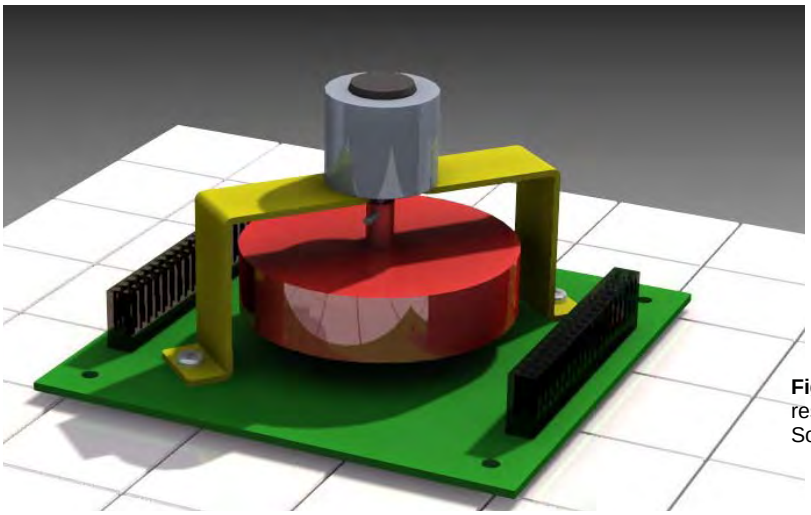


Figura 2.11 Rueda inercial y su respectivo motor, realizados en Solid Edge V18

2.5 La rueda inercial del subsistema de estabilización activa

La rueda inercial del subsistema de estabilización será fabricada con una de las aleaciones más comunes de aluminio [2], en este caso la 6061 T6, cuya composición química y propiedades físicas se presentan en las tabla 2.1 y 2.2.

Composición química		
	Mínimo	**Máximo**
Silicio:	0.4%	0.8%
Hierro:	--	0.7%
Cobre:	0.15%	0.40%
Manganeso:	--	0.15%
Magnesio:	0.80%	1.2%
Cromo:	0.04%	0.35%
Zinc:	--	0.25%
Titanio:	--	0.15%
Otros (total):	--	0.15%

Tabla 2.1 Composición química de la aleación 6061 T6

Propiedades físicas	
Densidad:	2.71 kg/dm^3
Módulo Elástico:	68.9 GPa
Punto de Fusión:	582-652 ºC
Calor Específico (0 a 100ºC):	896 J/kg. ºC
Conductividad Térmica:	167 W/m ºC
Resistencia Específica:	3.99 x 10^{-6} Ω cm
Coeficiente de Dilatación Lineal (20-250ºC):	25.2 x 10^{-6} m/mºC

Tabla 2.2 Propiedades físicas de la aleación 6061 T6

Se eligió el aluminio debido a que algunos sensores son sensibles a materiales con propiedades ferromagnéticas, el aluminio siendo un material diamagnético no ofrece problemas en este sentido. Además, el aluminio tiene buenas características para fabricar la rueda, es un material liviano, resistente a la corrosión y con buena apariencia. Esta aleación en particular es muy resistente, fácil de trabajar y para un mejor acabado y más resistencia a la corrosión es posible anodizarlo. Este proceso consiste en crear una capa de oxido de aluminio desde la superficie hacia el interior del aluminio en un medio sulfúrico. Al anodizarlo se le da más dureza al aluminio, le otorga baja conductividad eléctrica y un buen acabado, mejor y más duradero que el de la pintura.

Las medidas de la rueda inercial se muestran en la figura 2.12.

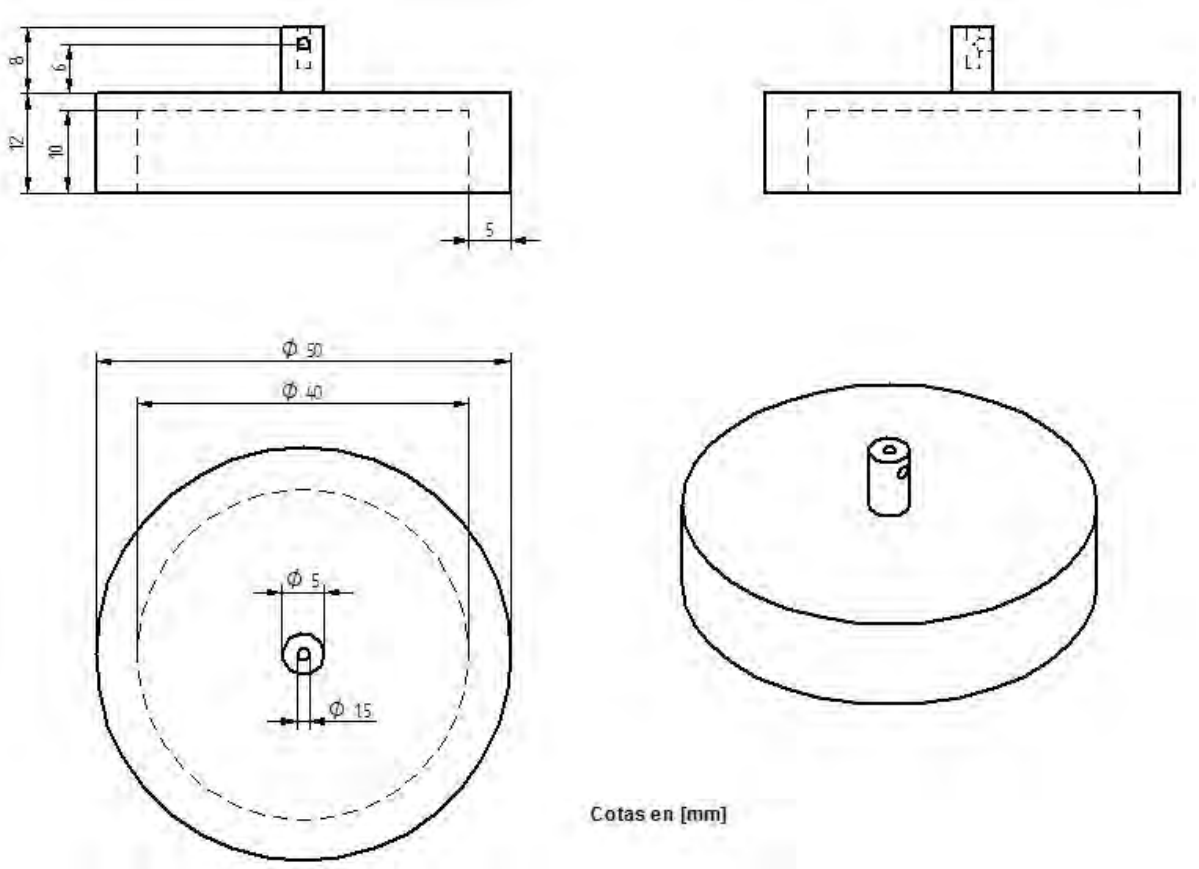


Figura 2.12 Diseño de la rueda inercial

2.6 Control de velocidad para la rueda inercial

La rueda inercial montada sobre la tarjeta se puede mover por medio de una diferencia de voltaje ya que es un motor de DC, y con ayuda de un puente H es posible hacer que esta gire en una u otra dirección, pero si se desea hacer que trabaje para un control de apuntamiento es necesario hacer que esta gire graduando su velocidad. Para graduar la velocidad del sistema compuesto por el motor y la rueda fue necesario usar una señal de PWM (Modulación por Ancho de Pulso, del inglés Pulse Width Modulation), la señal utilizada es básicamente una señal cuadrada en la cual podemos variar su ciclo de trabajo, que es la relación entre el período y el tiempo que pasa la señal en un valor positivo, esto

permite controlar la cantidad de energía que se le proporciona al motor permitiendo variar su velocidad.

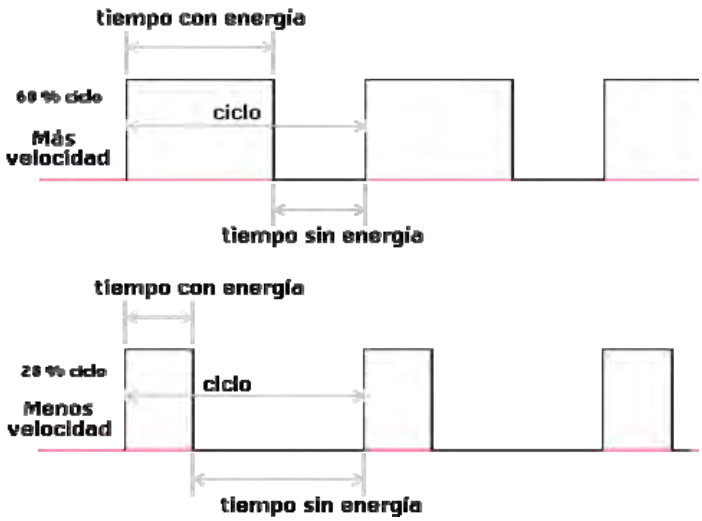


Figura 2.13 Señal de PWM

Para que el motor de DC pudiera fijar la velocidad de la rueda inercial a un cierto número de RPMs fue necesario obtener un modelo matemático del subsistema de estabilización.

Primero se estableció el conjunto de variables que describen las características dinámicas de dicho sistema, que en particular es un sistema de control realimentado o en lazo cerrado. En estos sistemas la entrada se compara con su salida para después enviar una señal actuadora proporcional a la diferencia entre la entrada y la salida para corregir el error.

Otra de las características del subsistema de estabilización es que se consideró lineal e invariable con el tiempo. Hablando de manera estricta los sistemas lineales no existen, pues todos los sistemas tienen un grado de no linealidad. No obstante, los sistemas lineales se crean por cuestiones de facilidad en su diseño y análisis. También se les considera invariables con el tiempo debido a que se supone que los parámetros del sistema no cambian con el tiempo. Debe subrayarse que estrictamente hablando esto tampoco es así, ya que al menos uno de los parámetros del sistema variará, en el caso del motor por ejemplo uno de los parámetros que puede variar es la resistencia de los embobinados con el calor.

Para empezar con el modelado del subsistema de estabilización, se realizó un diagrama esquemático (modelo) que representa a las variables del subsistema, tal como se ve en la figura 2.14.

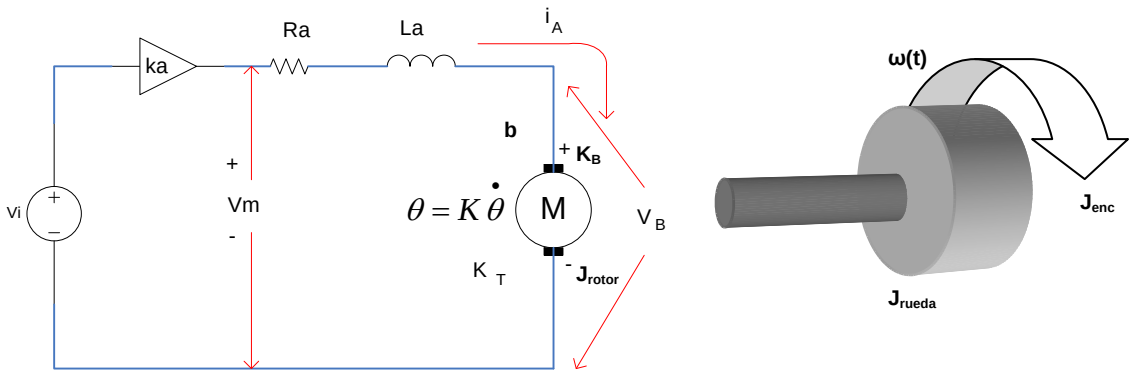


Figura 2.14 Modelo del sistema de control motor – rueda

Donde:

V_B = Fuerza contraelectromotríz [V]
V_i = Voltaje de entrada [V]
V_m = Voltaje del motor [V]
Ka = Ganancia del amplificador [V/V] = 2.4 [V/V][1]
Ra = Resistencia de armadura [Ω] = 31.4 [Ω][2]
La = Inductancia de la bobina [H] = 0.475 [mH][2]
K_T = Constante de torque del motor [Nm/A] = 6.14 [mNm/A][2]
K_B = Constante de fuerza contraelectromotriz [Nm/A] = 6.14 [mNm/A][2]
J_{rueda} = Inercia de la rueda [kg m^2] = 13.154 x 10^{-6} [kg m^2][3]
J_{rotor} = Inercia del rotor [kg m^2] = 7.35 x 10^{-8} [kg m^2][2]
J_{enc} = Inercia del disco magnético del encoder [kg m^2] = 7 x 10^{-9} [kg m^2][4]
ω(t) = Velocidad angular [rad/s]
b = Coeficiente de fricción dinámica [Nms] ≈ 0 [Nms]

Del modelo del sistema de control se obtienen las siguientes ecuaciones.

Ecuaciones de la parte eléctrica:

Usando las leyes de Kirchhoff, se tiene:

$$V_m(t) = k_a V_i(t)$$

$$V_m(t) = R_A i_A(t) + L_A \frac{di_A(t)}{dt} + V_B(t)$$

$$V_B(t) = K_B \omega(t)$$

Ecuaciones mecánicas:

$$T(t) = K_T i_A(t)$$

$$T(t) = J_{total} \frac{d\omega(t)}{dt} + b\omega(t)$$

[1] Valor obtenido de la configuración del amplificador en el circuito.
[2] Valor obtenido de la hoja de datos del motor.
[3] Valor obtenido del modelo hecho en Solid Edge V18.
[4] Valor obtenido de la hoja de datos del encoder.

En las ecuaciones se asume que no hay torsión en el eje que une a la rueda inercial con el motor.

Ya sea por transformaciones de Laplace o variables de estado, ambas representaciones son equivalentes, no obstante al emplear variables de estado resulta más fácil resolver el sistema resultante de ecuaciones de variables múltiples.

En primer lugar se debe resolver el sistema de manera continua, pero hay que considerar que la implantación se hará de manera digital y por tanto habrá que discretizar el sistema posteriormente.

En vista de que el subsistema de estabilización no es demasiado complicado, se optó resolverlo mediante Laplace. Para dar una solución de control al sistema se debe obtener una función de transferencia que represente las relaciones de entrada y salida entre variables. Por tanto, aplicando la transformada de Laplace a las ecuaciones obtenidas anteriormente y caracterizando el sistema por medio de la respuesta al impulso con todas las condiciones iniciales igual a cero, se obtiene:

$$V_m(s) = K_A V_i(s) \quad \ldots (2.1)$$
$$V_m(s) = R_A i_A(s) + sL_A i_A(s) + V_B(s) \quad \ldots (2.2)$$
$$V_B(s) = K_B \omega(s) \quad \ldots (2.3)$$
$$T(s) = K_T i_A(s) \quad \ldots (2.4)$$
$$T(s) = sJ_{total}\omega(s) + b\omega(s) \quad \ldots (2.5)$$

Sustituyendo (2.1) y (2.3) en (2.2),

$$K_A V_i(s) = R_A i_A(s) + sL_A i_A(s) + K_B \omega(s) \quad \ldots (2.6)$$

Sustituyendo (2.4) en (2.5),

$$K_T i_A(s) = sJ_{total}\omega(s) + b\omega(s) \quad \ldots (2.7)$$

Despejando $i_A(s)$ en (2.6),

$$i_A(s) = \frac{K_A V_i(s) - K_B \omega(s)}{sL_A + R_A} \quad \ldots (2.8)$$

Sustituyendo (2.8) en (2.7),

$$\left(\frac{K_A V_i(s) - K_B \omega(s)}{sL_A + R_A}\right)(K_T) = (sJ_{Total} + b)(\omega(s)) \quad \ldots (2.9)$$

Despejando, finalmente se obtiene,

$$\frac{\omega(s)}{V_i(s)} = \frac{K_T K_A}{(sL_A + R_A)(sJ_{total} + b) + K_T K_B} \quad \ldots (2.10)$$

Ahora bien, para simplificar un poco el modelo y de acuerdo con datos del fabricante del motor, se considera que *b = 0.* Aunque K_T y K_B son considerados

parámetros separados, su valor está estrechamente relacionado. Para aclarar esta relación, la potencia mecánica en la armadura se representa como:

$$P = V_B(t) i_A(t)$$

y también como,

$$P = T_m(t)\omega(t)$$

en donde, T_m está en [Nm] y ω(t) en [rad/s] y dado que,

$$T_m(t) = K_T i_A(t) \quad \text{y}$$
$$V_B(t) = K_B \omega(t),$$

al sustituir e igualar ambas ecuaciones se obtiene:

$$P = T_m(t)\omega(t) = K_B \omega(t) \frac{T_m(t)}{K_T}$$

y despejando se tiene,

$$K_B\left(\frac{V}{rad/s}\right) = K_T\left(\frac{Nm}{A}\right) = K$$

Finalmente la función de transferencia queda como:

$$\frac{\omega(s)}{V_i(s)} = \frac{K_A K}{s^2 \cdot L_A J_{total} + s \cdot (R_A J_{total} + bL_A) + bR_A + K^2}$$

sustituyendo K= 6.14 [mNm/A] , L_A = 0.475 [mH], R_A = 31.4 [Ω], b = 0 y a que

$$J_{total} = J_{motor} + J_{rueda} + J_{enc} = 13.235 \times 10^{-6} \ [\text{kg m}^2]$$

sustituyendo se tiene que,

$$\frac{\omega(s)}{V_i(s)} = \frac{2.4096 \times 10^6}{s^2 + 66105s + 5997}$$

Ya que Laplace es utilizado dentro de espacio continuo es necesario obtener una ecuación que pueda ser utilizada en espacios discretos. Los sistemas de control en tiempo discreto tienen características únicas en las que sus señales están formadas por trenes de pulsos, o bien, codificadas en forma digital. Estos procesos a menudo tienen componentes analógicos, como en el caso del motor de CD, que al ser un dispositivo analógico puede ser controlado ya sea mediante un controlador que produzca señales analógicas o mediante un controlador digital que produzca información digital. En particular, la entrada y la salida de un controlador digital se pueden representar mediante secuencias numeradas, con números separados por el tiempo de muestreo *T*.

Para operación lineal el convertidor D/A se puede representar como un dispositivo de muestreo y retención, S/H (Sample and Hold), el cual es un muestreador y un dispositivo de retención de datos. El S/H se utiliza frecuentemente para el análisis de sistemas en tiempo discreto y está formado por un muestreador ideal y un retén de orden cero, ZOH (Zero-Order Hold).

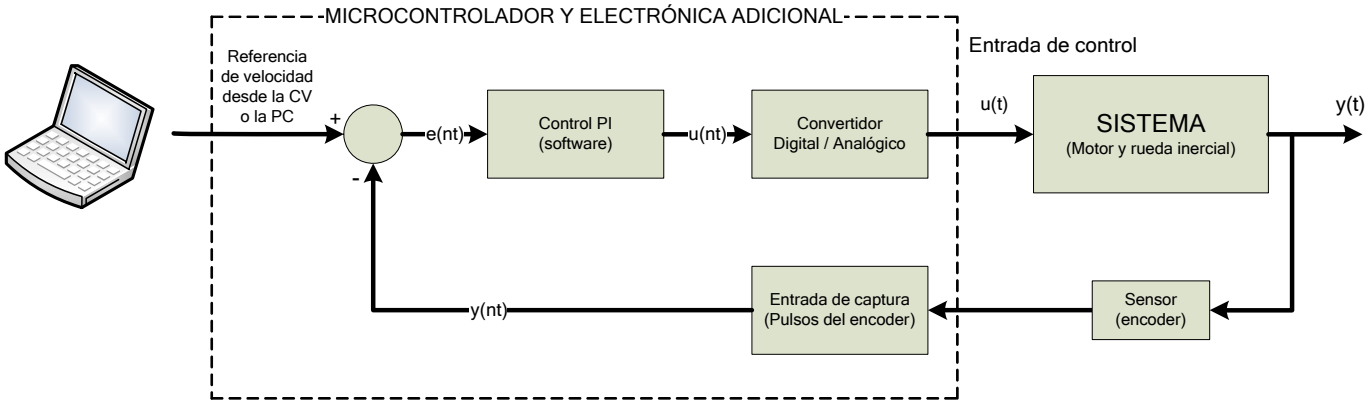


Figura 2.15 Diagrama a bloques del sistema de control

Ahora, usando un ZOH para obtener la función de transferencia, se utiliza la ecuación:

$$H(z) = \left(1 - z^{-1}\right) \cdot Z\left\{\frac{H(s)}{s}\right\}$$

Como se observa, en este caso se utiliza la transformada Z, esta operación es lo que Laplace representa al resolver ecuaciones diferenciales en tiempo continuo, es decir representa un método operacional para resolver ecuaciones en diferencias lineales y sistemas lineales con datos discretos o digitales. Para despejar la ecuación se usó MATLAB pasando de un modelo en modo continuo a un modelo en modo discreto, a través de la función *c2dm*, introduciendo los datos como sigue:

```
>>Ra=31.4;        % Resistencia de armadura
>>La=.475e-3;     % Inductancia de armadura
>>Kt=6.14e-3;     % Constante de torque del motor
>>Ka=2.4;         % Ganancia del amplificador
>>J=13.235e-6;    % Suma de inercias

>>num=Kt*Ka;
>>den=[J*La Ra*J Kt^2];

>>Ts=0.01         % tiempo de muestreo

>>[numz denz]=c2dm(num,den,Ts,'zoh')
```

después de ejecutar la instrucción con los datos:

```
>>numz =

         0    0.3638    0.0006
```

```
>>denz =

    1.0000   -0.9991    0
```

la ecuación en un espacio discreto queda como:

$$\frac{\omega(z)}{V_i(z)} = \frac{0.3638z + 0.0006}{z^2 - 0.9991z}$$

Ahora se verá la respuesta del sistema sin controlador, para esto es necesario obtener la función de transferencia en lazo cerrado para ello se usó el comando *cloop* en MATLAB. Para visualizar la respuesta al escalón se usó el comando *dstep*, en tanto que el comando *stairs* conecta los puntos en la gráfica. Se siguieron los siguientes pasos:

```
>> numz = [numz(2) numz(3)];
>> [numz_lc denz_lc]=cloop(numz,denz);

>> [y]=dstep(numz_lc,denz_lc,201);
>> t=0:0.010:2;
>> stairs(t,y)
```

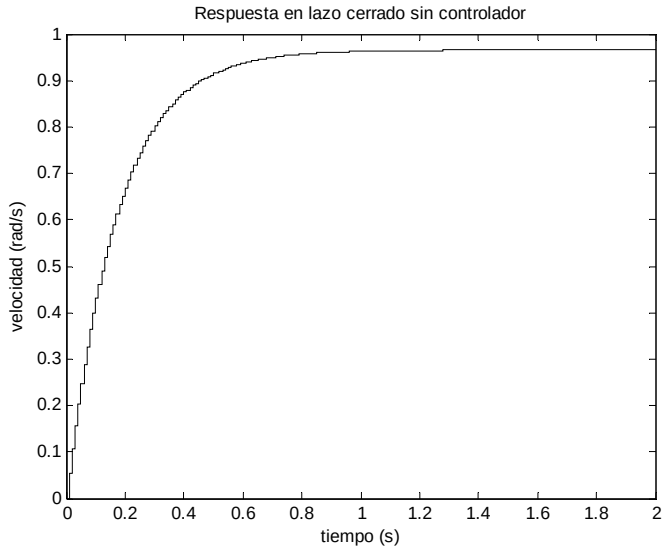


Figura 2.16 Respuesta a escalón del sistema

Aun sin controlador, la respuesta es buena, pero al pasar a un modelo experimental puede cambiar de manera significativa por lo que es conveniente agregar alguno y se eligió un controlador PI (proporcional-integral) que suele ser el más común. La representación del controlador usando la transformada de Laplace es:

$$C(s) = kp + \frac{ki}{s}$$

Ahora es necesario obtener su representación en el dominio de Z, para ello se utilizó la transformación bilineal o método de Tustin definido como:

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

La cual se sustituye en la ecuación del controlador en la forma de Laplace.

Para implementar un controlador es necesario sintonizarlo para que realice adecuadamente su función. La sintonización es un proceso que permite escoger los valores adecuados de kp y ki que van en el controlador para que actúe adecuadamente sobre el sistema. Las técnicas de control de velocidad de una rueda inercial para que actúen en un satélite son complicadas y es algo que va mas allá del propósito de esta tesis, por lo que basándose en la literatura con este problema y las capacidades del microcontrolador se eligieron constantes pequeñas *kp = 0.5* y *ki = 0.0625,* de modo que el sistema presente una respuesta críticamente amortiguada o sobreamortiguada. En la figura 2.17 se presenta una simulación con el controlador puesto en la forma de Laplace, los pasos realizados en MATLAB fueron:

```
>>numc=[0.5 0.0625];
>>denc=[1 0];
>>num_cont=conv(numc,num);
>>den_cont=conv(denc,den);
>>[numlc denlc]=cloop(num_cont,den_cont);  % obtiene ec. en lazo cerrado
>>step(numlc,denlc)          %se obtiene su respuesta al escalón
```

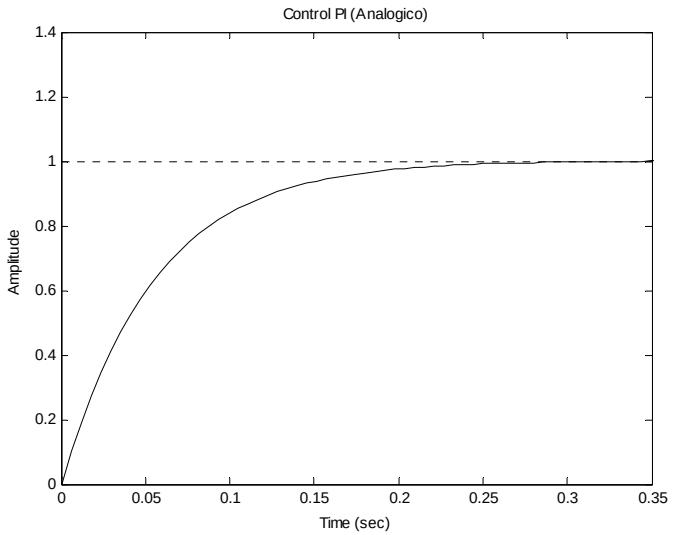


Figura 2.17 Respuesta del sistema en tiempo continuo

Para obtener la solución del sistema en tiempo discreto y recordando que un factor importante que determina en gran medida la respuesta del sistema es el tiempo de muestreo, basándose en la constante de tiempo mecánica del motor elegido y que otorga el fabricante que es de 61.2 [ms], es que se puede elegir uno igual o menor para registrar correctamente su respuesta, en este caso se eligió un tiempo de 10 [ms].

En la simulación al introducir los datos en MATLAB se obtuvo la respuesta mostrada en la figura 2.18.

```
>>Ts=0.01;  % tiempo de muestreo
>>kp=0.5;
>>ki=0.0625;
>>[dencz numcz]=c2dm([1 0],[kp ki],ts,'tustin');  % obteniendo la función
del controlador por el método de Tustin en tiempo discreto
>>numaz=conv(numz,numcz);
>>denaz=conv(denz,dencz);
>>[numaz_cl denaz_cl]=cloop(numaz,denaz);
>>[x2]=dstep(numaz_cl,denaz_cl,101);
>> stairs(t,x2)
```

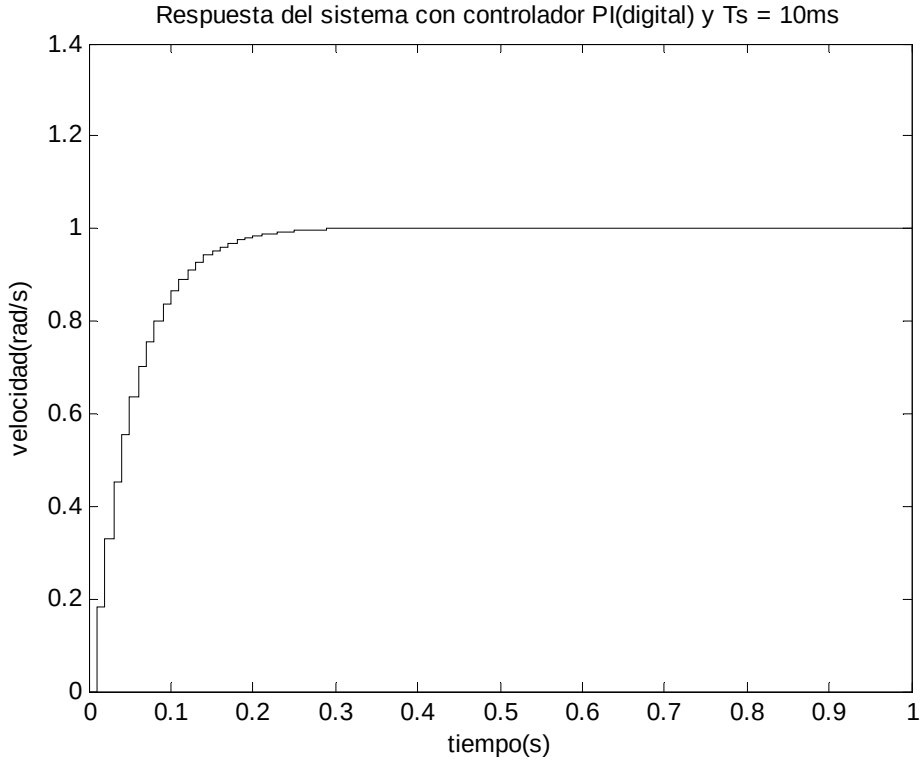


Figura 2.18 Respuesta del sistema en tiempo discreto con Ts= 10 ms

Como se observa, se obtiene una respuesta muy similar a la que se obtuvo en tiempo continuo. Si ahora se adopta un tiempo de muestreo mayor como pueden ser 100 [ms], se obtiene ahora la respuesta de la figura 2.18.

Al comparar las figuras 2.18 y 2.19 se observa que el sobrepaso con Ts = 100 [ms] es muy grande y además se presentan oscilaciones, las que en un sistema como el que se pretende diseñar no son convenientes, debido a que las oscilaciones alterarían la posición del satélite y sería problemática su corrección cada vez que se utilicen las ruedas inerciales.

Debido a que durante el desarrollo de las pruebas operativas aun no se había fabricado la rueda inercial presentada en esta tesis, no fue posible corroborar prácticamente estos resultados. Por esta razón, cuando el sistema sea implantado

completamente y posteriormente se lleve a cabo el control de la rueda inercial con estos parámetros y se llegaran a presentar irregularidades de control, otra forma de obtener buenos resultados es optar por un método empírico para sintonizar el controlador adecuado al sistema. En este caso se podría aplicar el método de sintonización de Ziegler-Nichols, que se utiliza cuando no se tiene disponible un modelo matemático completo del sistema.

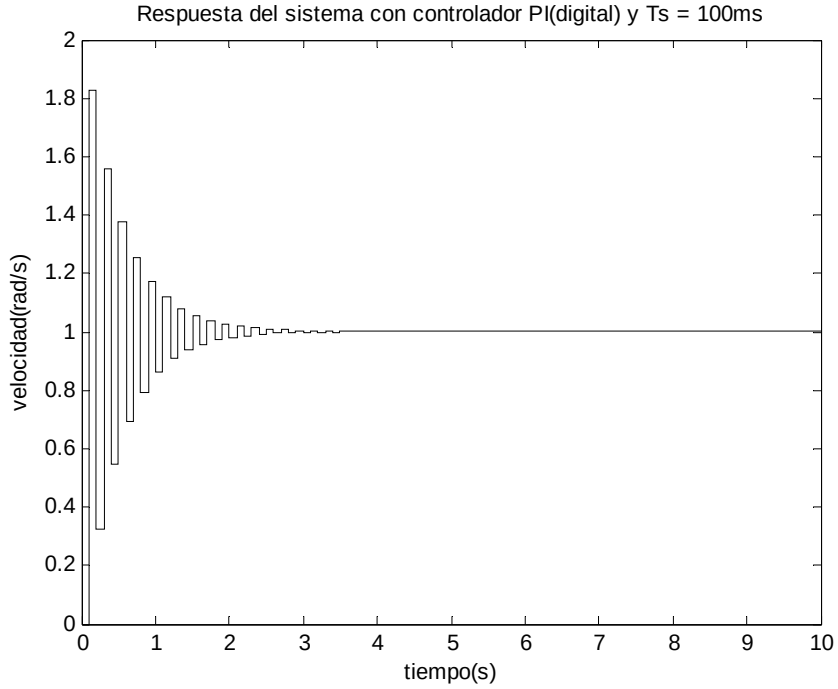


Figura 2.19 Respuesta del sistema en tiempo discreto con Ts= 100 ms

Aunque está demostrada la estabilidad del sistema resulta conveniente ver el diagrama de polos y ceros del sistema para determinar la máxima ganancia con la que el sistema se torna estable, este diagrama se presenta en la figura 2.20. Dado que se está trabajando con la representación en tiempo discreto el área dentro de un círculo unitario representa la zona dentro de la cual los polos se pueden mover sin que el sistema se desestabilice. Para obtener de manera aproximada el máximo valor de ganancia se utilizó el comando *rlocfind* sobre la función de transferencia del sistema, esto se realizó de la manera siguiente:

```
>>[K polos]=rlocfind(numaz,denaz,[-1])
```

Se indica el lugar donde se desea esté uno de los polos, en este caso -1 que según el diagrama de polos es donde uno de ellos se puede seguir moviendo. Y se obtiene:

```
>>K =
   11.3144
>>polos =
    0.9988
   -1.0000
   -0.0030
```

Por lo que en un valor de ganancia igual o superior a este el sistema empieza a desestabilizarse.

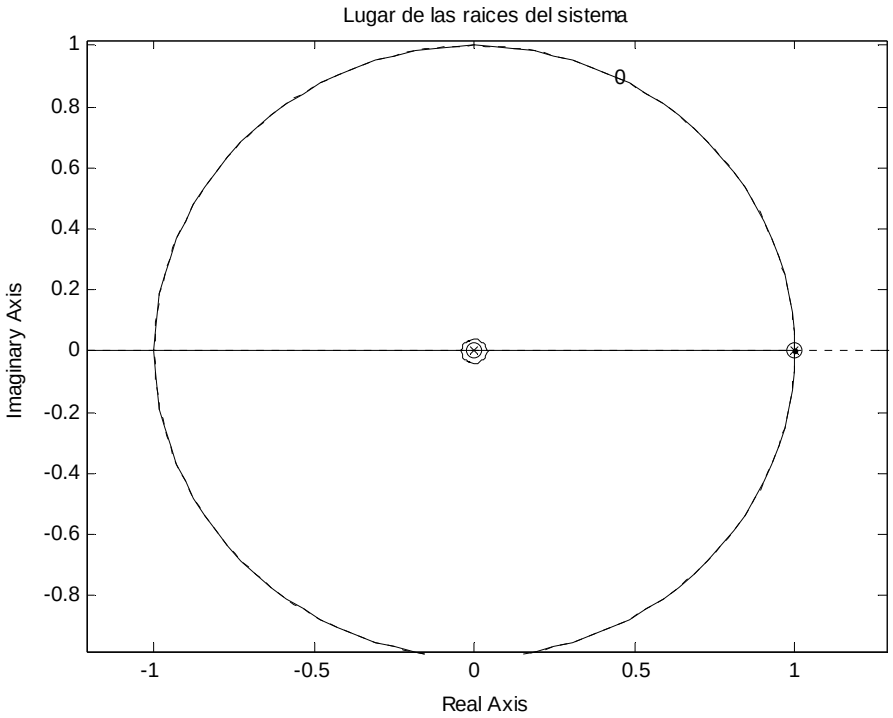


Figura 2.20 Diagrama de polos y ceros del sistema con el controlador

Bibliografía

1. Kuo, B. *"Control Automático"*. Ed. PrenticeHall Hispanoamericana. 7ª edición, 2005.

2. Datos del aluminio, ASM Aerospace Specification Metals, 2008.
   [http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA6061T6]

3. Sitio HowStuffWorks, 2008. [http://science.howstuffworks.com/gyroscope3.htm]

4. Universidad de Michigan, EEUU. "*Control Tutorials for MATLAB and Simulink",* 2008. [http://www.engin.umich.edu/class/ctms/examples/motor/digital.htm]

5. Howley, B. "*Overview of Spacecraft Attitude Determination and Control*". Artículo de Lockheed Martin Space Systems Company.

6. Lappas, V. "*Control Techniques for Aerospace Systems"*. Surrey Space Centre METU Aerospace Seminar. Artículo del seminario realizado el 8 de Mayo de 2003.

7. Tohami, S. "*Sensor modeling, attitude determination and control for micro-satellite"*. Tesis de Maestría. Universidad Noruega de Ciencia y Tecnología (NTNU), 2005.

Capítulo 3

Capítulo 3

Subsistema de estabilización activa por bobinas de torque magnético

3.1 Introducción

Existen varios métodos para estabilizar un satélite, unos de ellos son pasivos y otros métodos son activos como se explicó en el capítulo anterior. Uno de los métodos activos más populares, debido a su sencillez, es el de las Bobinas de Torque Magnético (BTMs), con el cual se llegan a obtener condiciones de apuntamiento similares a las obtenidas con ruedas inerciales en un satélite. Sin embargo, debe señalarse que éste método es de acción lenta, debido a que las fuerzas de corrección dinámica que genera son de menor magnitud que las generadas por las ruedas inerciales.

En el presente capítulo se muestra la parte complementaria del subsistema de estabilización para el satélite educativo, la cual está compuesta por tres BTMs, de igual forma se muestra su forma de funcionamiento, las consideraciones para fabricar las BTMs y su forma general de control por medio de un microcontrolador que se encuentra embebido en la tarjeta de estabilización.

3.2 Estabilización por medio de bobinas de torque magnético

3.2.1 Principio de operación

El principio básico de operación de una BTM, puede ejemplificarse de manera sencilla por medio de una brújula similar a las que usaban los antiguos navegantes como Colón, Magallanes o Marco Polo. Estas tenían una aguja magnetizada que siempre apuntaba hacia el norte Terrestre debido a que la aguja se alineaba con el campo magnético terrestre.

De la misma forma, cuando un imán se expone a un campo magnético externo y local, experimenta una fuerza que lo hace girar, esta fuerza también recibe el nombre de torque o momento magnético dipolar. Este torque actúa solo si el campo magnético que emite el imán no está alineado con el campo magnético local. Este principio se emplea en todas las brújulas, ya sea que se tome un imán de barra o se imante el extremo de una aguja y la pongamos a flotar en agua, estos girarán hasta que su campo magnético quede alineado, en este caso con el de la Tierra.

El mismo principio se usa en el SATEDU para tener la posibilidad de generar fuerzas que permitan maniobrar el sistema.

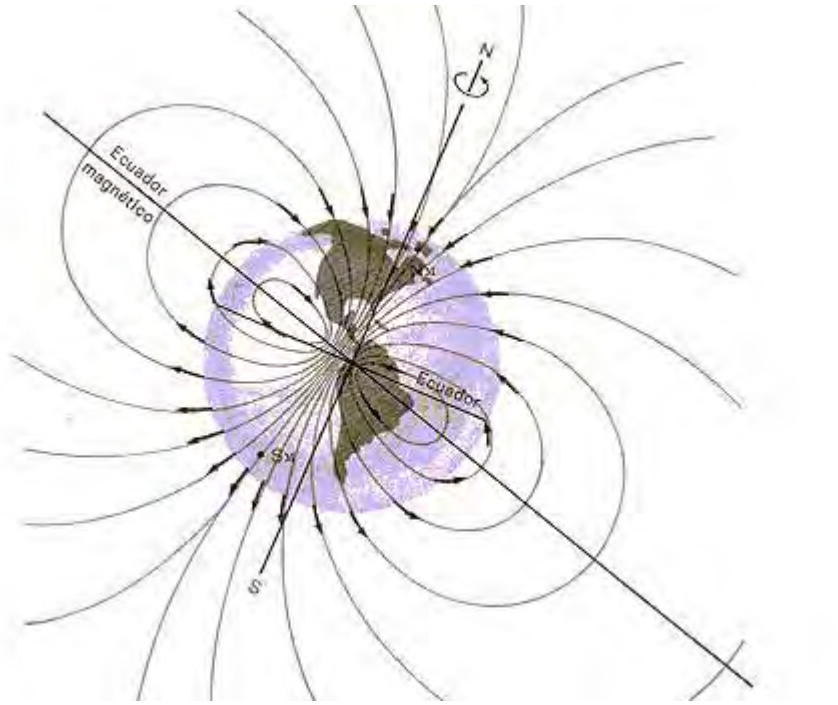


Figura 3.1 Representación del campo magnético terrestre

3.2.2 Diseño y funcionamiento de las bobinas de torque magnético

Algunos sistemas de orientación han consistido en imanes unidos a la estructura para orientar a un satélite pequeño pero el problema radica en que no se puede apagar un imán permanente para que deje de emitir un campo magnético.

Aun así muchos satélites, sobre todo los pequeños, ya han volado con esta clase de sistemas de control, haciendo que los satélites apunten de manera constante hacia la Tierra en la mayor parte de su órbita.

Para tener un sistema magnético que se pueda encender y apagar a voluntad se utilizan BTMs, es decir, solenoides, que contienen conductor enrollado de tal manera que forme un conjunto cilíndrico de N espiras sucesivas, en las cuales se hace pasar corriente para que en conjunto con el campo magnético que emula a un imán en forma de barra sea posible mover al SIMSAT con una menor cantidad de energía.

Si se dejara a un solenoide suspendido en el aire, por medio de un hilo atado en su parte media, éste trataría de alinearse con el campo magnético terrestre, es por eso que podemos también decir que un solenoide es un electroimán, figura 3.2.

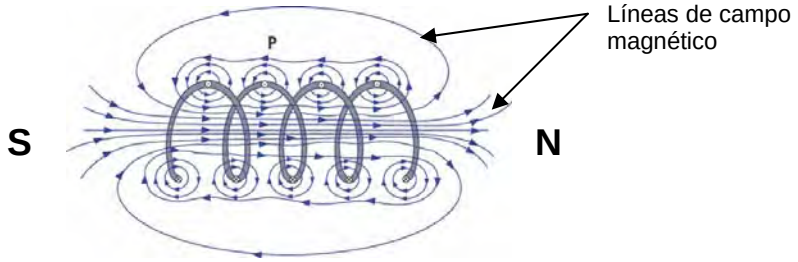


Figura 3.2 Campo magnético en un solenoide por el que fluye una corriente.

Si ahora suponemos que se acoplan este tipo de solenoides a la estructura y se accionan por medio de una secuencia de control, es posible controlar la orientación del sistema al que estén unidas siempre y cuando exista un campo magnético en el medio donde se encuentre y no estén alineadas con el mismo a este. Este sistema es el concebido como bobinas de torque magnético.

Para llevar a cabo el diseño de las BTMs, hay que conocer el dipolo que producen las mismas, el cual determinará la fuerza del par que podrán emitir en un momento dado. Entre mayor sea el dipolo mayor será el par producido por las BTMs.

El dipolo, *d*, producido por una BTM está dado por la siguiente expresión:

$$d = \left(\mu_r \cdot N \cdot i_B \cdot A\right) \cdot \hat{a}_n \quad \text{... (3.1)}$$

donde:

μ_r = Permeabilidad magnética relativa del núcleo [1]
N = Número de vueltas del enrrollamiento de la bobina [1]
i_B = Corriente a través de la bobina [A]
A = Área transversal de la bobina [m^2]
a_n = Vector unitario normal al plano del enrrollamiento [1]

Como se observa, el dipolo producido depende principalmente de 2 factores, su geometría y el núcleo, que pueden llevar a generar una bobina de características espaciales. En el caso de un satélite real, estos factores son determinantes para que llegue a tener una vida útil en el espacio, es decir que sea maniobrable. En este caso, si el satélite opta por un núcleo de aire la corriente que tendría que aplicar a las bobinas sería muy grande para igualar el dipolo de una BTM con núcleo. Si esto se compensara con el área transversal de la bobina aun así tendría que ser un área muy grande, por lo que en sistemas reales se puede optar por agregar un núcleo a las BTMs (sacrificando peso) o se pueden tener BTMs del tamaño de una de las caras del satélite. Para el caso del SIMSAT estos aspectos no son tan críticos, pues solo se requiere demostrar el principio de su funcionamiento y por tanto se opta por utilizar unos pequeños solenoides con núcleo de hierro que tienen una permeabilidad magnética relativa de 5000 aproximadamente.

Otro factor importante a considerar en un satélite real es el material y el calibre del alambre de embobinado, además de que se puede considerar el aluminio, la plata y el cobre; generalmente se selecciona el cobre por su costo, densidad y conductividad. En el SIMSAT se utilizará cobre por las razones mencionadas.

En el satélite educativo las BTMs a construir estarán constituidas por dos embobinados uno de menor número de vueltas que el otro. De esta forma el dipolo generado por el embobinado de menor número de vueltas es pequeño y capaz de producir ajustes finos de orientación, en tanto que el embobinado de mayor número de vueltas se usará para hacer ajustes de orientación más grandes. Aunque no se han construido aún las bobinas se tiene pensado que estén constituidas de 2 calibres de alambre diferentes. También se perseguirá que las resistencias generadas por los embobinados sean similares y lleven a cabo un consumo de corriente moderado para ahorrar energía, de igual forma se perseguirá que el factor determinante del dipolo sea el número de vueltas de las bobinas y no la corriente, el diseño de las BTMs se muestra en la figura 3.3.

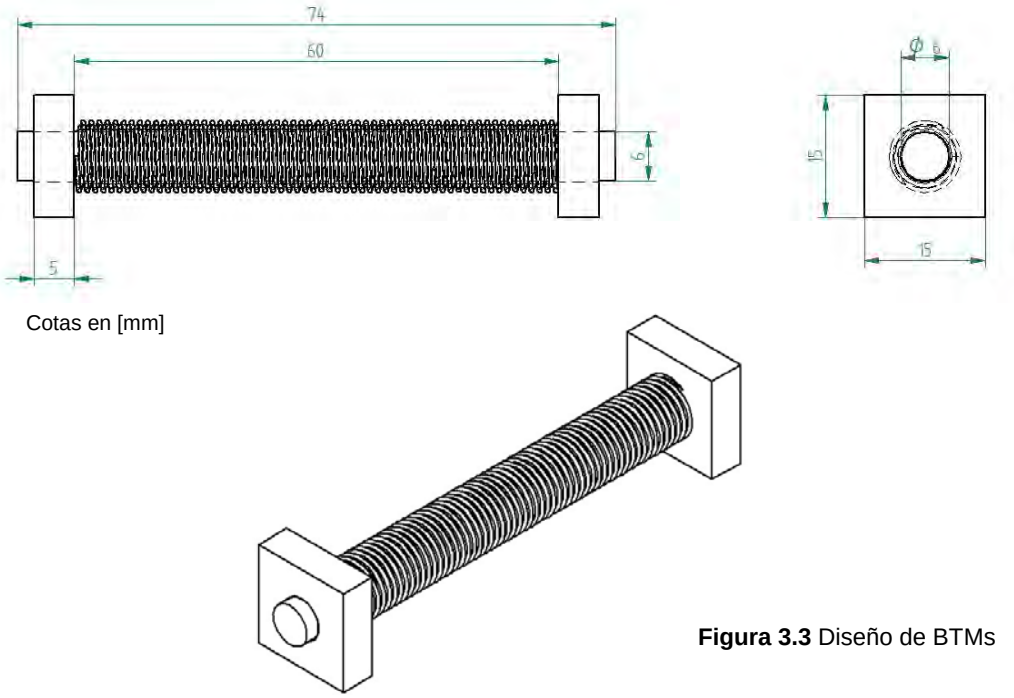


Figura 3.3 Diseño de BTMs

Para calcular la magnitud del torque, τ_B, generado por las BTMs se utiliza la siguiente expresión:

$$\tau_B = d \times B_M \quad \text{... (3.2)}$$

donde:

d = Dipolo magnético [Am^2]
B_M = Campo magnético del medio [T] (en este caso el de la Tierra)

Ambas variables se representan de manera vectorial, el campo magnético es del medio ya que es posible emular un campo magnético o bien utilizar solo el de la Tierra, considerando que la dirección de éste dependerá de la ubicación geográfica donde se trabaje.

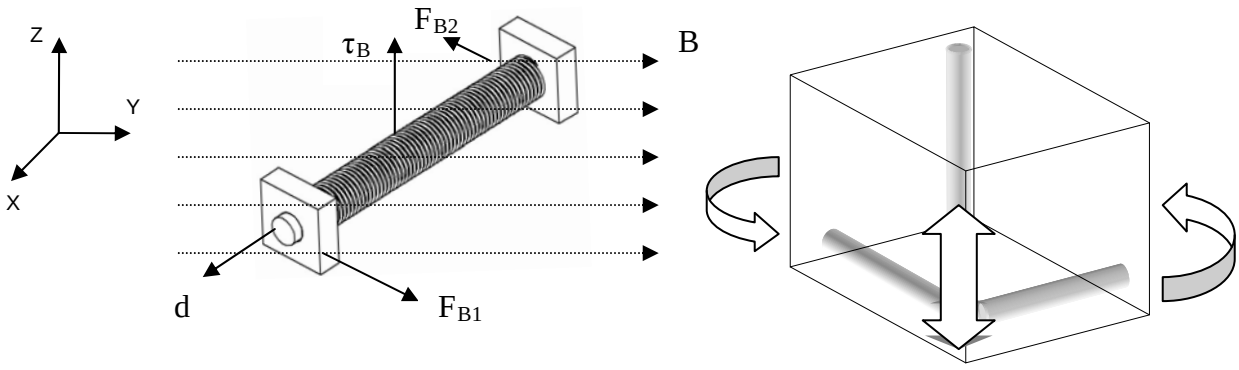


Figura 3.4 Torque generado por la BTM

En la figura 3.4 se puede apreciar como es que se generarían los pares de fuerza que moverían a las BTMs y al unir estas a la estructura es posible mover a todo el satélite. Como se apuntaba en un principio, la magnitud del torque será principalmente determinada por las características de las bobinas notadas a partir del dipolo máximo que se desee generar. Cabe apuntar por último que para el propósito de un satélite real es muy importante también el cálculo del campo geomagnético, porque este también tiene que ver en la magnitud del par generado, y es esencial que este considerado dentro del modelo de control de orientación que se genere.

3.3 Arquitectura y diseño del hardware de control de estabilización por medio de bobinas de torque magnético

Como se ha descrito, con BTMs y por medio del electromagnetismo es factible afectar la orientación o la dinámica de viaje de objetos, sin tener la necesidad de usar combustibles como el gas en un sistema de propulsión. Ahora es necesario conocer la forma en que se tienen que colocar las BTMs y cuantas se deben colocar. Si se tiene una sola BTM se puede alinear a un objeto dentro del campo de acción de un plano con el campo magnético de 2 maneras diferentes (suponiendo que podemos controlar la dirección de la corriente que circula por la bobina). Por esta razón, en un satélite real se usan tres tipos de BTMs colocadas ortogonalmente para tener un campo de control en tres planos, figura 3.5, o bien, en un sistema tridimensional, X, Y y Z. De esta manera se puede tener un buen sistema de apuntamiento.

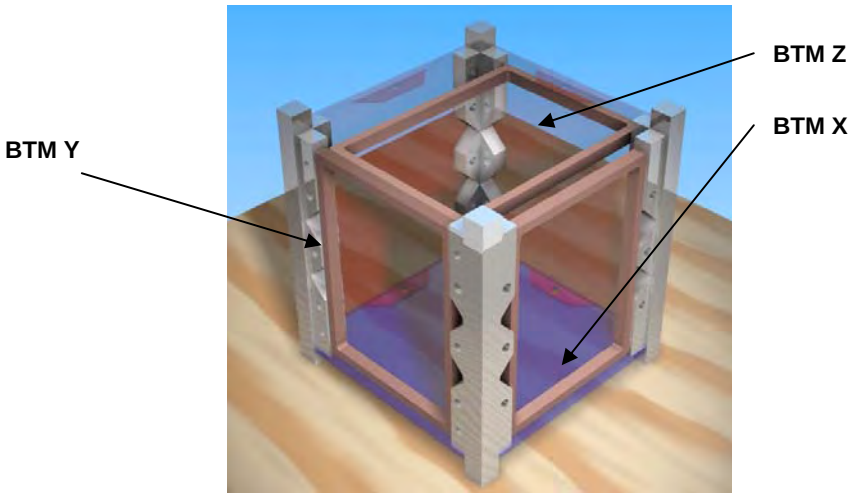


Figura 3.5 BTMs con núcleo de aire en un Picosatélite.

Para el SIMSAT se desarrolló un hardware capaz de manejar 6 BTMs (3 bobinas dobles), cuyos detalles se darán en el siguiente capítulo. Se tiene proyectado colocar 3 bobinas en su estructura, sin embargo, para aplicaciones de laboratorio convencional es difícil implantar la posibilidad real de visualizar la acción de la 3ª bobina. Por ello y para tener un sistema más económico se descartará el uso de las bobinas del eje Z, como se observa en la figura 3.6, para formar un sistema de orientación en 2 ejes. No obstante, el SIMSAT con bobinas en 3 ejes si se construirá y se empleará en una simulación recreada sobre una pequeña mesa suspendida en aire, en la cual será completamente factible y observable el control de estabilización en 3 ejes con efectos mínimos de fricción.

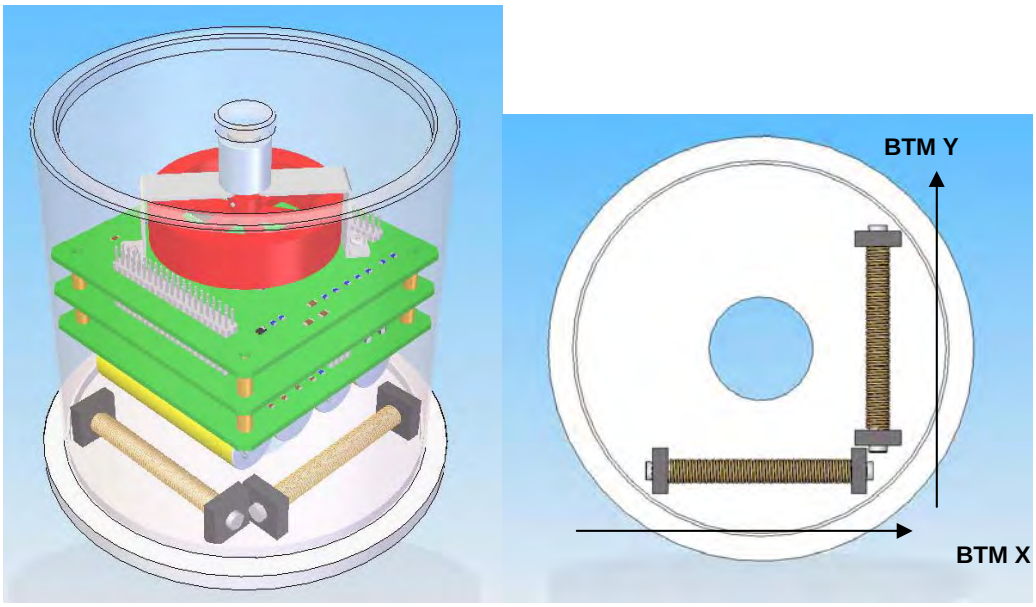


Figura 3.6 Posición de BTMs en SIMSAT

Para realizar un control de campo magnético en las BTMs que permita generar un dipolo variable, se decidió aplicar señales PWM a las BTMs, figura 3.7, de tal modo que al variar el voltaje aplicado a las bobinas se controla la corriente que conducen y por tanto, según las ecuaciones 3.1 y 3.2 mostradas anteriormente, también el dipolo.

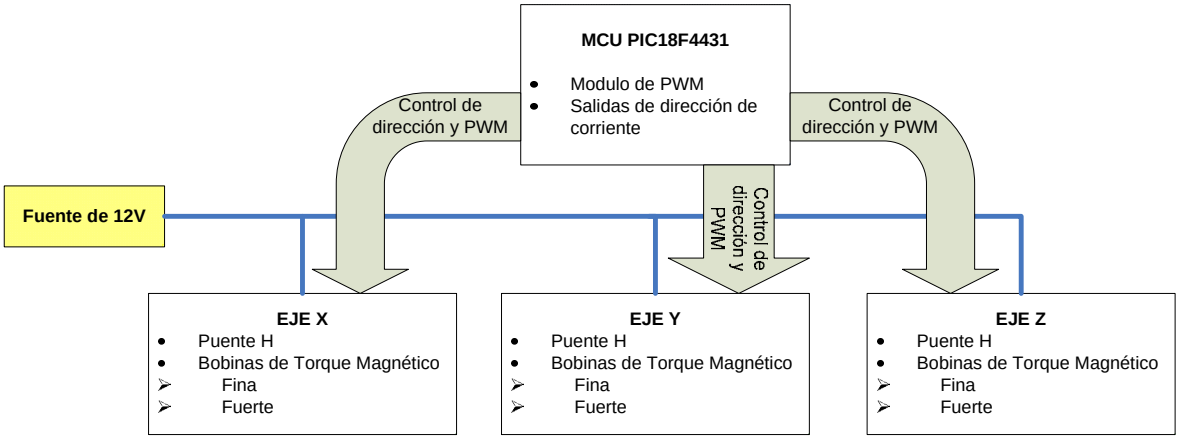


Figura 3.7 Esquema de control de las BTMs

Un problema adicional que se tiene en el uso de bobinas con núcleo, es que todo material ferromagnético presenta histéresis magnética, que es la tendencia del material a conservar el campo magnético que se le ha inducido. De este modo, al tener un campo magnético inducido en el núcleo este inducirá un campo magnético de una magnitud determinada. Sin embargo, si elevamos la corriente aplicada a la bobina y después la disminuimos quedando operando con la corriente original, observaremos que la magnitud del campo magnético inducido no será la misma, como se observa en la figura 3.8.

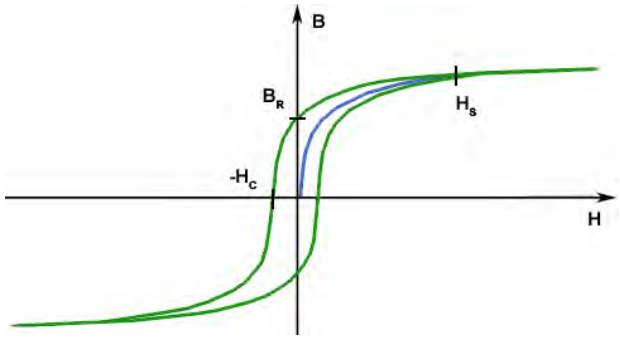


Figura 3.8 Histéresis de un núcleo ferromagnético

En la figura 3.8, B, representa la magnitud del campo magnético y H, la intensidad del campo magnético en el interior del núcleo. Esta última se relaciona con B debido a que está relacionada también con la corriente en el embobinado de la siguiente forma:

$$H = \frac{iN}{L}$$

Donde i es la corriente, N el número de vueltas del solenoide y L su longitud, por ello, la relación cambiará de acuerdo con la geometría de la bobina. En un satélite real y también en el SIMSAT esto es relevante dentro del control de apuntamiento del sistema. Debe subrayarse que el problema de histéresis es exclusivo de las bobinas con núcleo, en tanto que las BTMs con núcleo de aire no presentan este tipo de problemas, por lo cual presentan una gran ventaja sobre las primeras.

Bibliografía

1. Fleeter, R. "*The Logic of Microspace: Technology and Management of Minimum-cost space missions"*. Ed. Springer Space Technology Library. EEUU, 2000.

2. Prado, J. "*Dispositivo simulador para pruebas de detección de orientación y control de satélites pequeños".* Tesis de Doctorado. Facultad de Ingeniería, UNAM. México, 2008.

3. Serway, R y Jewett, J. "*Electricidad y Magnetismo"*. 6ª edición. Ed. Thomson Paraninfo. México, 2005.

Capítulo 4

Subsistema de sensores de estabilización para el SIMSAT

4.1 Introducción

Para que un satélite pueda realizar maniobras de orientación es necesario que este pueda conocer la posición en la que se encuentra y la posición a la que debe llegar para realizar alguna tarea como alinear los paneles solares hacia el sol, apuntar la antena hacia la Tierra para llevar a cabo comunicaciones, apuntar para tomar una fotografía, etc., es aquí donde los sensores de estabilización tienen que actuar.

En el presente capítulo se presentan los detalles generales acerca de los sensores utilizados en el SIMSAT para utilizarlos como una Unidad de Medición Inercial (UMI), que comprende el complemento para que el subsistema de estabilización pueda llevar a cabo la tarea de orientar al SIMSAT en los experimentos.

4.2 Sensores

La palabra sensor viene del latín *sentire,* que significa percibir. Podemos decir que la palabra sensor tiene una relación con los sentidos humanos como el tacto o la vista. Es así que los sensores pueden proveernos de información sobre el mundo físico y químico con señales, algunas de las cuales no podrían ser percibidas directamente por nuestros sentidos. Es así que podemos definir a un sensor como a un objeto capaz de responder ante estímulos físicos o químicos determinados.

Otra palabra que recurrentemente se confunde con el término sensor es la palabra transductor que viene del latín *transducere* que significa, guiar a través de. Un transductor es un dispositivo que convierte una señal de una forma física en una señal correspondiente pero de una forma física distinta. Es por tanto un dispositivo que convierte un tipo de energía en otro.

Sensor y transductor suelen emplearse como sinónimos, pero la palabra sensor sugiere un significado más extenso: la ampliación de los sentidos para alcanzar un conocimiento de cantidades físicas que por su naturaleza o tamaño no pueden ser percibidas de manera directa por los sentidos. Transductor en cambio, sugiere que la señal de entrada y de salida no pueden ser homogéneas. En la presente tesis, se usa la palabra sensor como la unión de ambos términos más los demás sistemas de procesamiento de señal y de conversión que los dispositivos referidos pudieran tener.

4.3 Sensores de estabilización

Los sensores de estabilización en el SIMSAT pretenden darnos la posición de nuestro satélite a partir de una posición establecida y que es conocida. Las categorías de sensores de posición en un satélite están divididos en 2 y una es el complemento de la otra, ellas son:

Sensores de referencia: Son los que otorgan un ajuste definido mediante la medición de la dirección respecto de un objeto, tales como el Sol, una estrella o Tierra. Algunos de ellos son:

- Sensores finos de Sol
- Sensores de horizonte de la Tierra
- Sensores de estrellas
- Magnetómetros
- Determinación de posición usando el GNSS (Global Navigaton Satellite Systems)

Sensores de inercia: Estos miden continuamente, pero solo miden cambios en su posición, como los giróscopos. Estos necesitan un ajuste y una calibración de los sensores de referencia. Los giróscopos representan el único sensor de inercia conocido hasta ahora; las únicas variantes que tiene este son en cuanto a su fabricación, el mejor conocido hasta ahora es uno que se basa en la generación de anillos láser.

En el simulador usamos un sensor de referencia, representado por la brújula electrónica y 4 sensores inerciales mencionados anteriormente que son 3 giróscopos de un eje dispuestos de manera ortogonal y un acelerómetro de 3 ejes.

Es así que se construyó un sistema que emula a un sistema de sensado de posición similar a uno satelital para simularlo y se construyó un sistema similar a una Unidad de Medición Inercial más modesta y menos precisa de las que pueden ser conseguidas de manera comercial debido a la calidad de sus sensores.

4.3.1 Unidad de Medición Inercial (UMI)

Una UMI o IMU (del inglés, Inertial Measurement Unit) es el componente principal de un sistema de navegación inercial usado en vehículos como aeroplanos y submarinos. Una UMI trabaja sensando su propia velocidad y dirección de movimiento usando una combinación de acelerómetros y giróscopos, la cual le permite a una computadora guía seguir sus movimientos usando un proceso llamado “cálculo muerto “ que es el proceso de estimar la posición actual de un objeto basado en una posición previa calculada mediante una velocidad y tiempo conocidos.

Un ejemplo de una UMI comercial es la comercializada por la compañía Xsens Motion Technologies que vende varias versiones de su instrumento para uso de captura de movimiento en tiempo real desde sencillos sistemas de navegación de vehículos no tripulados hasta un conjunto de ellos para la captura de movimiento humano.

Figura 4.1 UMI de la compañía Xsens Motion Technologies

En el simulador (SATEDU) se intentan emular los mismos resultados de una UMI comercial. Se integró una pequeña UMI con giróscopos y un acelerómetro triaxial de fabricación MEM (Micro Electro Mecánica) aunque no de tan buena calidad pero en el satélite educativo sirven para su propósito de demostración.

En el SIMSAT se dispuso de los sensores para crear una IMU como se muestra en la figura 4.2.

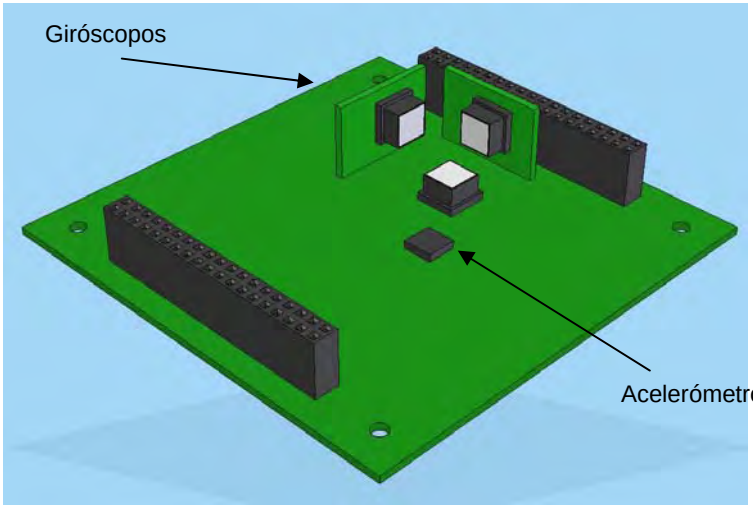


Figura 4.2 UMI de SIMSAT en la tarjeta de sensores de estabilización

Esta disposición tan peculiar de los giróscopos se debe a que cada uno de ellos solo mide respecto a un eje del circuito, este eje es perpendicular a su base. Para cubrir los tres ejes de medición que necesitamos se emplean 2 pequeñas tarjetas adicionales colocadas perpendicularmente a la placa base para llevar a cabo la medición en los ejes restantes.

La desventaja de las UMIs es que por más calidad que tengan sus componentes, no son una tecnología perfectamente precisa debido a efectos que presentan sus componentes principalmente los giróscopos que sufren de un fenómeno llamado deriva, este consiste en que los giróscopos no mantienen adecuadamente su rigidez giroscópica y se mueven del lugar del que fueron fijados provocando acumulación del error, además de que si utiliza componentes MEM también son sensibles a la temperatura.

Las UMIs son solo un componente del sistema de navegación. Es por eso que se utilizan otros sistemas para corregir las imprecisiones de las UMIs, en este caso un sensor de referencia que es la brújula electrónica.

4.3.2 Estimación angular mediante el acelerómetro

Un acelerómetro mide como su nombre lo indica, aceleración a lo largo de un eje predefinido en el dispositivo. La gravedad de la Tierra es también aceleración por lo que un acelerómetro también puede medir la gravedad terrestre.

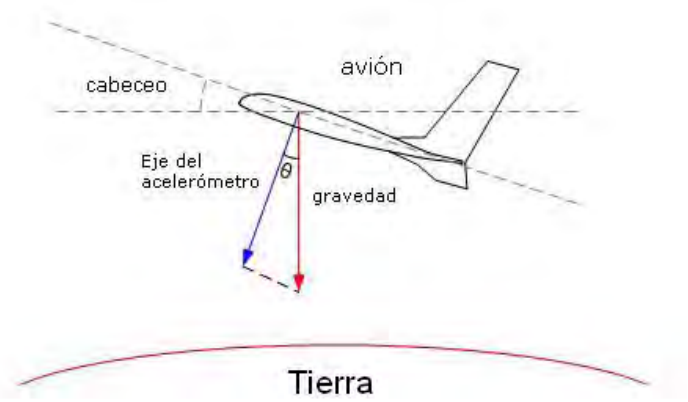


Figura 4.3 Uso del acelerómetro como inclinómetro

Como se muestra en la figura 4.3 se indican las direcciones de la gravedad y el eje del acelerómetro que es perpendicular a la plataforma del avión. El ángulo θ formado entre los vectores de la gravedad y el eje de referencia del acelerómetro se relacionan con el cabeceo del avión, si θ es conocido puede determinarse el ángulo de cabeceo como se ve en la formula ºcabeceo = θ+90º. Dado que se conoce la fuerza de gravedad de la Tierra un simple cálculo da el ángulo de cabeceo como sigue:

$$a = g\cos\theta$$

$$\theta = \arccos\left(\frac{a}{g}\right)$$

donde:

a = aceleración sobre un eje del acelerómetro
g = aceleración gravitacional

Y entonces ya que:

$$°cabeceo = \theta + 90°$$

$$°cabeceo = \arcsin\left(\frac{a}{g}\right)$$

El cálculo del ángulo de alabeo es muy similar solo que se necesita un acelerómetro extra (si es que con el que se cuenta es de un solo eje), o en nuestro caso usar el eje perpendicular al eje de cabeceo.

En realidad el proceso es un poco diferente del proceso mostrado anteriormente debido a que el seno inverso no puede darnos un rango de medición de 360°. Un solo eje no puede distinguir entre varios ángulos como por ejemplo de un ángulo de 45° de uno de 135° debido a que la función de seno inverso nos da el mismo valor es por eso que hay que basarse en los resultados de otro eje para establecer el cuadrante y de ahí conocer el ángulo adecuado.

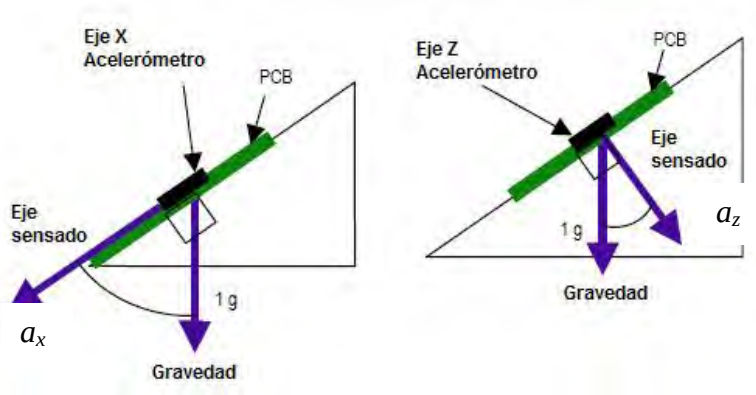


Figura 4.4 Estimación basada en dos ejes

Por eso, haciendo las consideraciones anteriores es necesario el uso de otra función:

$$^{o}cabeceo = \arctan\left(\frac{a_Z}{a_X}\right)$$

y extendido al otro eje

$$^{o}alabeo = \arctan\left(\frac{a_Z}{a_Y}\right)$$

donde:

a_X = aceleración en el eje X del acelerómetro
a_Y = aceleración en el eje Y del acelerómetro
a_Z = aceleración en el eje Z del acelerómetro

Dado que los ejes de X e Y se encuentran sobre el plano horizontal el resultado de estos para ofrecernos el ángulo de guiñada no está definido.

Las mediciones que nos ofrecen los acelerómetros son relevantes dentro de los sensores del SIMSAT debido que si hay una inclinación en la brújula electrónica esta presentará un error en su medición por lo que también se hace una medición de la inclinación para realizar una compensación en la medida de la brújula.

Aunque estos sensores parecen ser bastante versátiles contienen debilidades, la primera es que este sensor es sensible a la aceleración de todo el sistema, esto es que para obtener las medidas angulares correctas este tiene que estar estático o moviéndose a velocidad constante, ya que los cambios de aceleración producirían estimaciones erróneas. La segunda debilidad es que son sensores cuya salida es bastante ruidosa y si está en un vehículo que está sometido a leves vibraciones el resultado es peor obligando a que se implemente un filtro paso bajas que reduzca el ruido.

La medición de la inclinación representa una de las pruebas hechas en la interfaz de prueba que será descrita en un capítulo posterior.

4.3.3 Estimación angular mediante el giróscopo

Un giróscopo es un dispositivo capaz de medir velocidad angular en grados por segundo. Algunos, se usan en aplicaciones críticas (como satélites de comunicaciones, aviación)

son muy avanzados y complicados. Afortunadamente hoy en día podemos conseguir giróscopos de bajo costo que son suficientemente buenos. Son de fabricación MEM y los producen varias compañías como Analog Devices, este tipo de giróscopos se usan en el SIMSAT y tienen un rango de ±300 [°/s].

Con los giróscopos es posible medir la posición del sistema integrando las mediciones que obtenemos a través de él, estos sensores ofrecen la ventaja de hacer mediciones angulares sobre los tres ejes sin depender de otros parámetros, como el gradiente gravitacional en el caso del acelerómetro.

Una de las desventajas de este dispositivo es que presenta varios errores al momento de su medición como el “bias” que es un offset o nivel de referencia distinto de cero y un efecto llamado deriva, que consiste en la variación del “bias”, a comparación de cuando está inmóvil y genera un error que se va acumulado en las mediciones, como se puede apreciar en la figura 4.5.

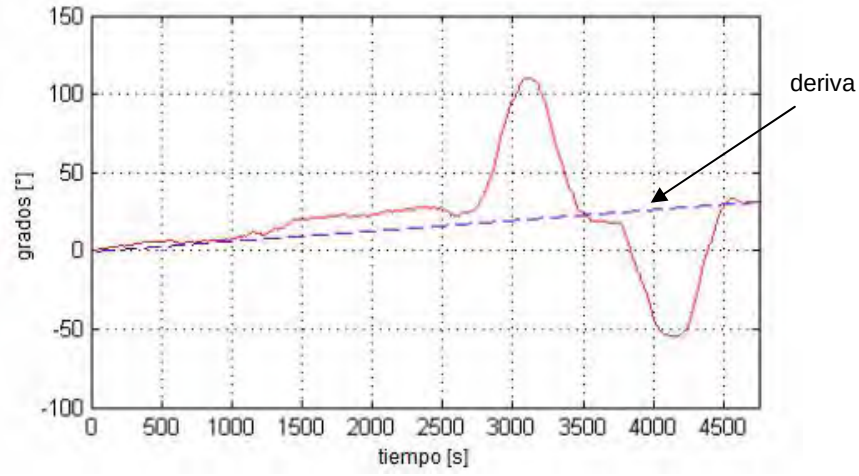


Figura 4.5 Error causado por la deriva en un giróscopo

Para obtener una gráfica como en la figura 4.5 es necesario integrar los datos que se reciben de los giróscopos ya que debido al bias en la serie de mediciones los datos son muy dispares y se obtendría una gráfica como en la figura 4.6.

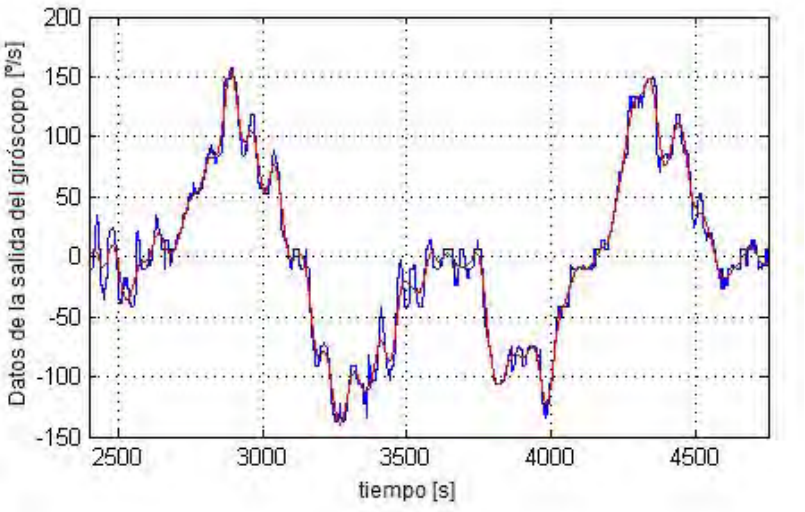


Figura 4.6 Datos del giróscopo sin procesar

Para la correcta interpretación de los datos es necesario integrar el conjunto de datos,

$$a(t) = \int_{t_0}^{t} g(t)dt + a(t_0)$$

donde:

a(t) = ángulo
g(t) = señal de salida del giróscopo

Estas integrales se tienen que realizar para cada una de los ejes alrededor de los cuales se mide la posición angular (X, Y y Z). Debido al error introducido en cada medición por el bias, también se está integrando este error introduciendo así un error que crece con el paso del tiempo en cada medición. Este bias no es posible cancelarlo, solo se puede restar de la medición, ya que no es un valor constante, de modo que todo esto resulta en una deriva del giróscopo desde unas pocas centésimas o décimas de grado hasta varios grados por minuto, todo esto depende de la calidad de los giróscopos.

En el SIMSAT estas correcciones se llevan a cabo en el software que recibe los datos, el software de prueba que fue hecho para el subsistema solo corrobora que funcionen adecuadamente, ya que hacer las correcciones a través de otros métodos más exactos como un filtro Kalman en las mediciones, va mas allá de los objetivos de esta tesis.

4.3.4 Estimación angular mediante la conjunción de acelerómetros y giróscopos

Como se ha visto, el obtener una estimación angular a través de un solo tipo de sensor como los giróscopos o los acelerómetros es muy difícil y no provee buenos resultados. Sin embargo, estos dos sensores son complementarios debido a que mientras los giróscopos proveen una buena estimación angular de manera dinámica los acelerómetros proveen una buena referencia angular de manera estática.

Teniendo en cuenta las características de estos sensores, es posible realizar un algoritmo capaz de usar la información de ambos para obtener un buen sistema de medición inercial, este algoritmo se deja como una propuesta ya que va mas allá de los alcances de esta tesis.

Bibliografía

1. Clifforf, M. y Gomez, L. *AN3107 "Measuring Tilt with Low-g Accelerometers".* Hoja de aplicación Freescale Semiconductor. EEUU, 2005.
2. Fortescue, P., Stark, J. y Swinerd, G. "*Spacecraft Systems Engineering".* 3ª edición. Ed. Wiley and Sons. EEUU, 2003.
3. Pallas, R. "*Sensores y acondicionadores de señal".* 4ª edición. Ed. Marcombo. España, 2007.
4. Kerhuel, L. *"Miniature Inertial Measurement Unit (IMU)",* 2008. [http://www.kerhuel.eu/wiki/index.php5?title=Miniature_Inertial_Measurement_Unit_-_IMU]
5. Pycke, T. *"Accelerometer to pitch and roll"*, 2008. [http://tom.pycke.be/mav/69/accelerometer-to-attitude]
6. Pycke, T. *"Gyroscope to roll, pitch and yaw "*, 2008. [http://tom.pycke.be/mav/70/gyroscope-to-roll-pitch-and-yaw]

Capítulo 5

Desarrollo de las tarjetas de estabilización activa y sensores para el SIMSAT

5.1 Introducción

Como se explicó ya en capítulos anteriores las tareas del subsistema de estabilización son cruciales para que la carga útil o experimento de un satélite pueda desarrollar adecuadamente sus tareas. Inclusive pueden ser decisivas para que un satélite pueda sobrevivir en el espacio, todo esto mediante un programa de control complementario cargado en la computadora de vuelo. De la misma importancia es el sistema de sensores, sin el cual el control de apuntamiento de un satélite no sería posible, debido a que el sistema de estabilización necesita saber desde dónde y hacia dónde debe desplazarse.

En el presente capítulo se presentan los detalles generales acerca del desarrollo de hardware para las tarjetas de estabilización activa y de sensores para el SIMSAT, cuyas etapas y consideraciones de diseño se describen a continuación.

Así también, en el presente capítulo se habla de los aspectos mecánicos relacionados a este subsistema, las pruebas preliminares que fueron realizadas en "protoboard" previas a la fabricación de la tarjeta electrónica, así como la forma en que se cargan nuevos programas al subsistema de estabilización.

5.2 Arquitectura electrónica de la tarjeta de estabilización y su diseño electrónico asociado

La tarjeta está compuesta por un pequeño tacómetro, un puente H para el motor vinculado a la rueda inercial, 3 circuitos de puentes H para el manejo de las BTMS, un transductor de corriente a voltaje, dos amplificadores operacionales y el microcontrolador que da las ordenes a los elementos de la tarjeta. Adicionalmente se anexaron LEDs para indicar el funcionamiento de algunas partes del subsistema de estabilización.

5.2.1 Tacómetro

El tacómetro va acoplado a la parte trasera del motor seleccionado para manejar la rueda inercial, está compuesto por un sensor de efecto Hall y un disco magnético, que en conjunto envían 16 pulsos digitales por vuelta y a través de 2 canales.

Figura 5.1 Encoder MEnc13 de Maxon Motors

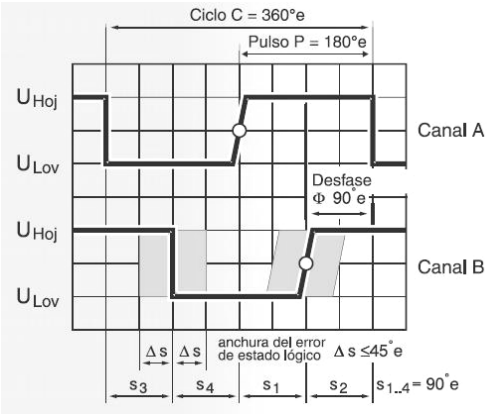


Figura 5.2 Pulsos del tacómetro

El efecto Hall fue descubierto por E. R. Hall en 1879. Su principio de operación se basa en que cuando un haz de partículas atraviesa un campo magnético existen fuerzas que actúan sobre las partículas ocasionando que la trayectoria del haz se deforme. Cuando un corriente fluye a través de un conductor se comporta como un haz de partículas en movimiento, por lo que al pasar por un campo magnético esta corriente se puede desviar. Si consideremos que el conductor se vuelve una placa a la que se le aplica un campo magnético perpendicular a su plano, como consecuencia los electrones que se desplazan se desvían hacia uno de los lados de la placa, con lo cual se carga negativamente mientras el lado opuesto se carga de manera positiva. Esta separación de cargas produce un campo eléctrico en el material. Esta separación dura hasta que las fuerzas a las que están sujetas las partículas cargadas del campo eléctrico compensan las fuerzas producidas por el campo magnético. Entonces, el resultado es una diferencia de potencial.

Hay 2 tipos de sensores de efecto Hall, los lineales y los de umbral. En los lineales la salida de voltaje varía en forma proporcional al flujo de campo magnético. En los que se basan en umbral, como el caso del tacómetro, la salida cambia en forma digital ante un campo magnético determinado. Los sensores de efecto Hall tienen la ventaja de funcionar como interruptores con una frecuencia de repetición de hasta 100 kHz, son económicos,

prácticamente inmunes a fuentes eléctricas contaminantes ambientales y capaces de trabajar en condiciones de trabajo severas.

Diseño electrónico asociado al tacómetro

El codificador fue conectado al microcontrolador (para registrar la velocidad del motor) siguiendo el diagrama de su conector indicado en la hoja de especificaciones que se muestra en la figura 5.3.

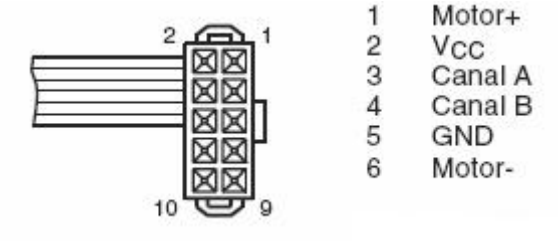


Figura 5.3 Patigrama del encoder

Para acoplarlo al motor cuenta con 2 terminales de energización para el motor, por lo que ambos sistemas se energizan del mismo conector lo que se traduce en mayor facilidad para desmontar al motor del sistema, facilitando con ello a su vez el diseño de su respectivo circuito impreso. Al codificador se le anexó un transistor en uno de sus canales para verificar su funcionamiento así como la recepción de datos por parte del microcontrolador. Además, se le anexó un arreglo de resistencias que operan como resistencias de “pull-up” (10kΩ) y como protección a la entrada (1kΩ). Las resistencias de “pull-up” se utilizan para asegurar que no haya entradas falsas en el microcontrolador cuando el codificador esté desconectado. De esta forma, la conexión empleada con el microcontrolador se muestra en la figura 5.4.

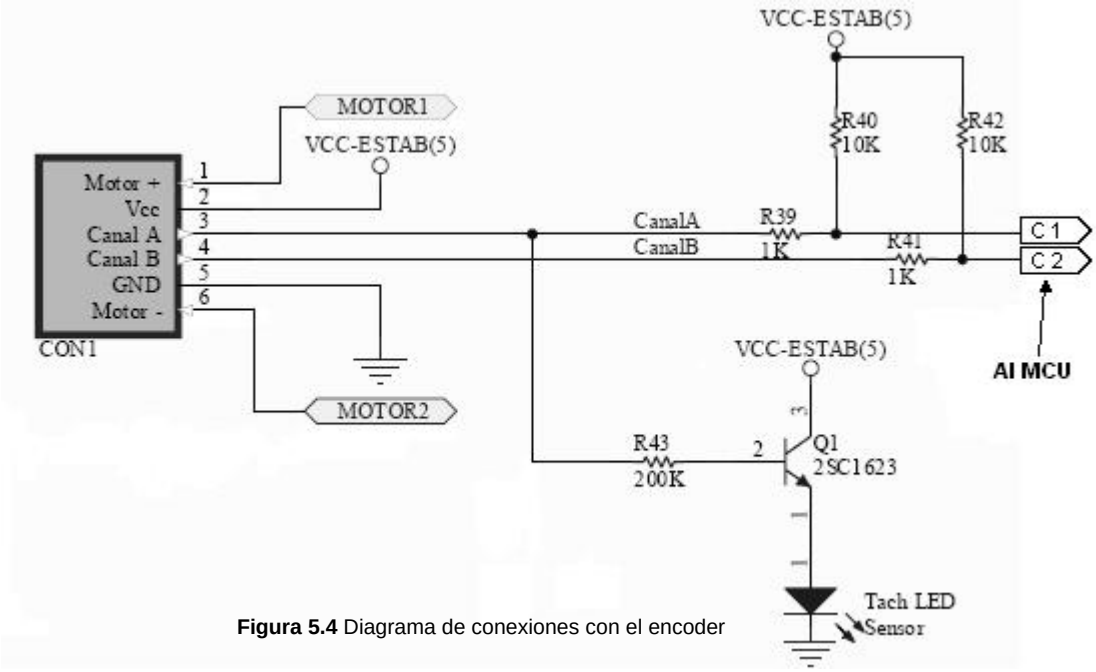


Figura 5.4 Diagrama de conexiones con el encoder

5.2.2 Puentes H empleados en el SIMSAT

El puente H se denomina así por su apariencia en un circuito esquemático, y es capaz de dirigir la corriente a través de una carga en ambas direcciones. Particularmente, el control bidireccional de un motor requiere de un puente H. Para entender el funcionamiento del puente H, éste se debe de dividir en dos partes, o medios puentes.

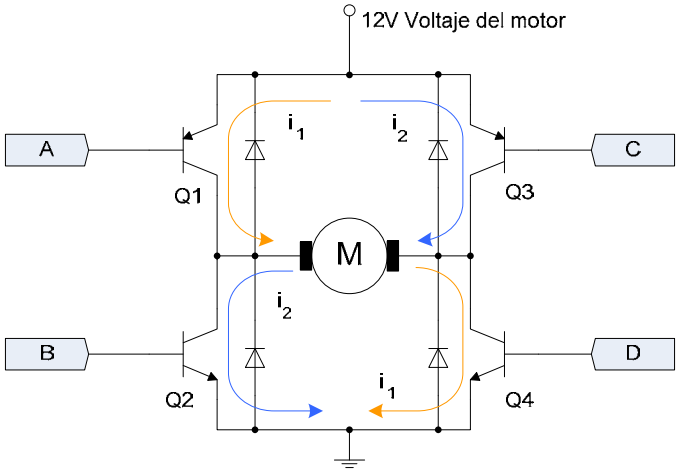


Figura 5.5 Diagrama esquemático del puente H

Refiriéndonos a la figura 5.5, los transistores Q1 y Q2 hacen un medio puente mientras Q3 y Q4 hacen otro medio puente. Cada uno de estos medios puentes es capaz de conectar un lado del motor a tierra o a la fuente de voltaje. Por ejemplo, cuando se activa a Q1 y se desactiva a Q2 el lado del motor está conectado a la fuente de voltaje. Encendiendo Q4 y dejando a Q3 apagado se conecta el lado opuesto del motor a tierra.

La corriente i_1 mostrada en la figura muestra el flujo de corriente resultante de esta configuración. La configuración inversa de encendidos de los transistores hace que el motor gire en sentido inverso redirigiendo la corriente como se ve en la figura 5.5 con la corriente i_2. Apagando los transistores el motor se detiene, en tanto que en el modo de freno las terminales del motor se aterrizan. El motor se comporta como un generador cuando está rotando y al cortocircuitar las terminales del motor se presenta una carga de magnitud infinita llevando al motor a que se detenga rápidamente.

	Q1 (A)	Q2 (B)	Q3 (C)	Q4 (D)
Giro izquierda	Encendido	Apagado	Apagado	Encendido
Giro derecha	Apagado	Encendido	Encendido	Apagado
Alto	Apagado	Apagado	Apagado	Apagado
Frenado	Apagado	Encendido	Apagado	Encendido

Tabla 5.1 Tabla de funcionamiento del puente H

Los diodos conectados entre cada transistor los protegen de picos de corriente generados por la fuerza contraelectromotriz (FCEM) cuando son apagados. También es aconsejable que se coloque un capacitor de tipo cerámico no polarizado entre las terminales del motor para reducir la emisión de ruido de radio frecuencia (RF) que se produce por el arqueo de los conmutadores en el motor.

Los puentes H empleados en el SIMSAT son de dos tipos, uno se usa en el motor y el otro en las bobinas de torque magnético. El puente usado en el motor es el TA7291S de la compañía Toshiba. La desventaja del puente H radica en que los transistores a los que se encuentra conectado el motor generan una caída de voltaje por cada transistor en este caso. De acuerdo con la hoja de datos del puente éste tiene una caída de entre 1 V y 1.35 V, lo que significa que si tenemos un voltaje de alimentación de 12 V, al motor llegarán entre 11 V y 10.65 V haciendo que se vea limitado en su velocidad. Sin embargo, entre las ventajas que tiene este puente se puede citar que ya tiene integrados los diodos para evitar los picos de corriente generados por FCEM.

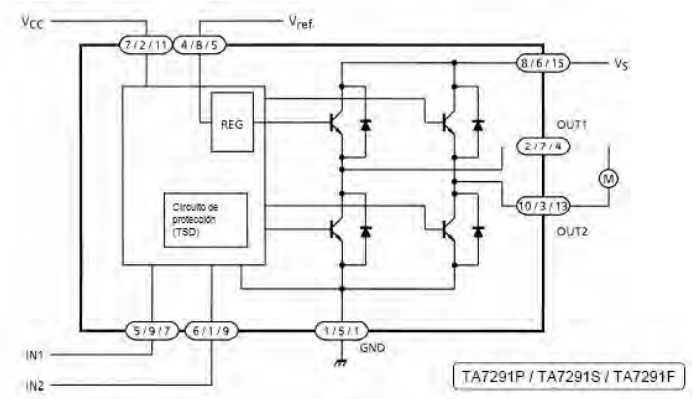


Figura 5.6 Diagrama de bloques del puente H, TA7291S de Toshiba usado en el motor

Para las bobinas de torque magnético se empleó el puente L293DD, de la compañía ST Electronics. En este caso, la caída de voltaje a la salida es de entre 1.4 V y 1.8 V, la cual no resulta tan relevante como en el caso del motor que ve mermada su velocidad al no tener suficiente voltaje. En el caso de las bobinas de torque magnético resulta más importante la corriente. Este puente H igualmente posee los diodos de manera interna ahorrando componentes y espacio en el circuito impreso.

Diseño electrónico asociado a puentes H para el SIMSAT

Al puente H del motor se le conectan dos voltajes de alimentación, 5V para su lógica y 12V para alimentar el arreglo de transistores y para energizar el motor. Adicionalmente, al puente H se le colocan un capacitor de 0.1μF para suprimir el ruido de RF que emite el motor y 2 capacitores extra de desacoplo, uno para cada fuente de alimentación en el circuito. Las señales de control de dirección para puentes se generan desde un microcontrolador, en tanto que la salida de PWM del microcontrolador fue conectada al control del voltaje del puente. Las conexiones se muestran en la figura 5.7.

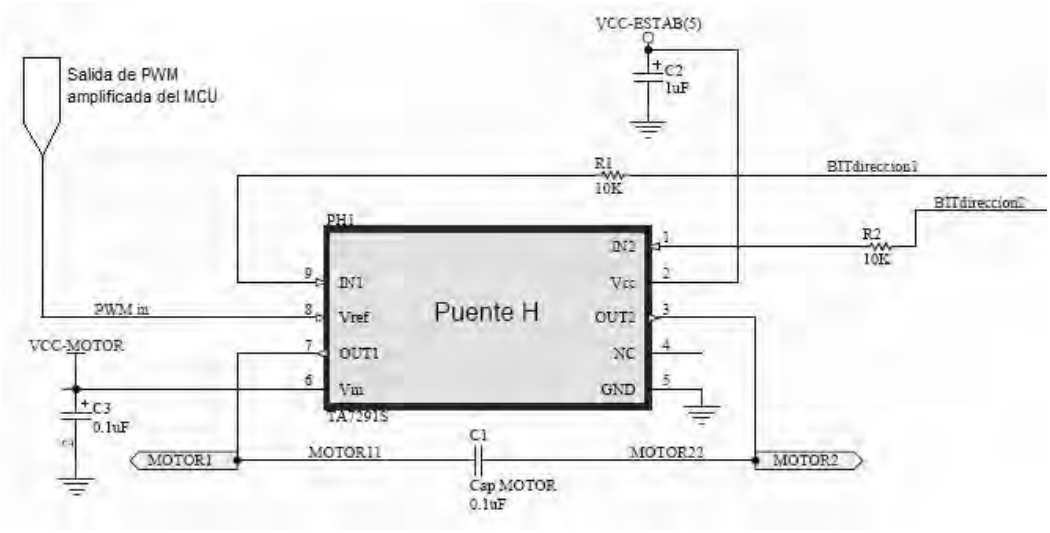


Figura 5.7 Conexión del puente H en SIMSAT

Los puentes H de las BTMs, al igual que el puente H del motor, usan 2 voltajes de alimentación. Debido a que este circuito en especial no posee una entrada de control de voltaje, la señal de PWM fue conectada a los habilitadores del puente H para hacer una forma de encendido y apagado de manera modulada y así controlar su voltaje a la salida. También se anexaron capacitores entre las salidas de las BTMs para eliminar el ruido de RF que producen. Adicionalmente, las entradas de control de dirección de la corriente se conectan al microcontrolador, se usan 6 salidas, la conexión se muestra en la figura 5.8.

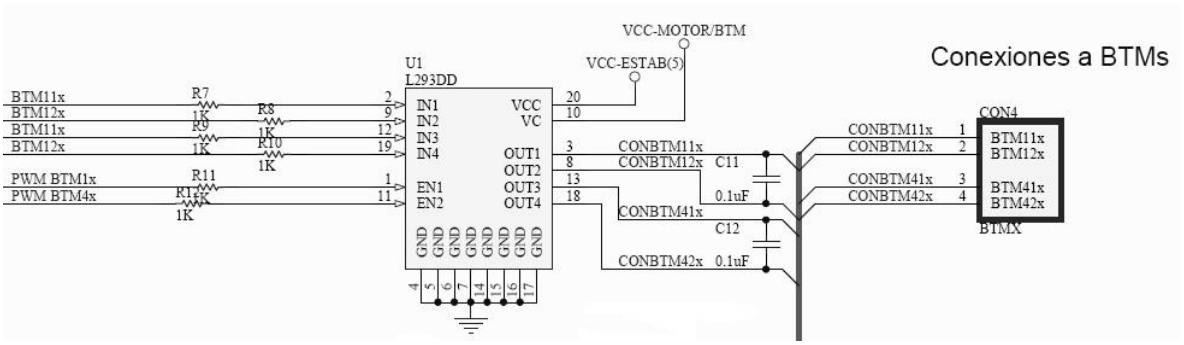


Figura 5.8 Conexión de BTMs al puente H en SIMSAT

5.2.3 Amplificadores operacionales del subsistema de Estabilización

Un amplificador operacional, es un amplificador diferencial de muy alta ganancia que posee una gran impedancia a la entrada y baja impedancia a la salida. En general el amplificador operacional se utiliza para proporcionar cambios en la amplitud del voltaje (amplitud y polaridad) en osciladores, filtros y muchos circuitos de instrumentación. Un amplificador operacional tiene muchas etapas de amplificación diferencial para obtener una ganancia de voltaje muy alta.

Los amplificadores operacionales pueden ser utilizados en un gran número de circuitos y ofrecer distintas características de operación.

Diseño electrónico asociado a amplificadores operacionales de Estabilización

En el SIMSAT se usan dos amplificadores operacionales, uno de ellos se emplea como comparador (en conjunto con el sensor de corriente) para accionar el apagado del motor ante un gran drenado de corriente que puede significar que su rotor haya quedado atorado. El amplificador usado es el LM6511 de la compañía Nacional Semiconductor, este amplificador permite hacer una interfaz entre circuitos analógicos y digitales, llegando a trabajar incluso con circuitos cuando solo se tiene una fuente de +3 ó +3.3V. Su salida en colector abierto permite la compatibilidad con varias familias de circuitos como TTL y CMOS. Las características que llevaron a la selección de este comparador en este circuito fueron su bajo consumo de potencia y gran velocidad de respuesta que es de solo 180 ns.

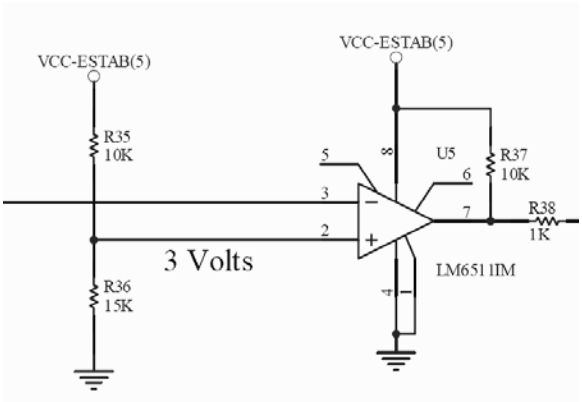
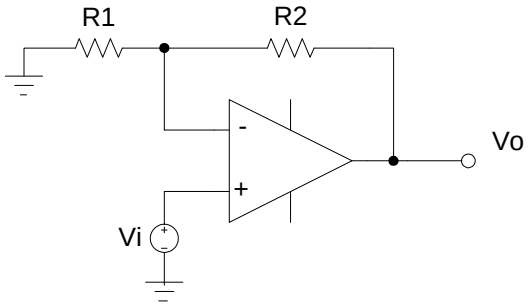


Figura 5.9 Configuración del circuito LM6511 como comparador en SIMSAT

El otro amplificador operacional se usa como amplificador no inversor, particularmente se usa para amplificar la señal PWM que va del microcontrolador hasta el puente H que controla al motor, ya que se necesita que la señal PWM tenga una amplitud igual a la del voltaje de alimentación del motor para variar adecuadamente su velocidad.

La configuración no inversora de un amplificador operacional es capaz de amplificar un voltaje sin invertirlo de polaridad a través de una malla de retroalimentación. En nuestro caso, el armado del amplificador tiene la siguiente configuración:



Donde :

Vo = Voltaje de salida

Vi = Voltaje de entrada

Su ecuación de amplificación de voltaje es:

$$V_o = \left(1 + \frac{R_2}{R_1}\right) \cdot V_i$$

Para el SIMSAT, R_2 = 110 kΩ y R_1 = 75 kΩ, y sustituyendo en la fórmula:

$$V_o = \left(1 + \frac{110k\Omega}{75k\Omega}\right) \cdot (5V) = 12.33V$$

Aunque el voltaje resultante de la malla externa de retroalimentación es mayor que el de la fuente que alimenta al amplificador, un amplificador operacional no puede amplificar a un voltaje mayor con el que se energiza, por lo que en este caso amplificará hasta los 12V.

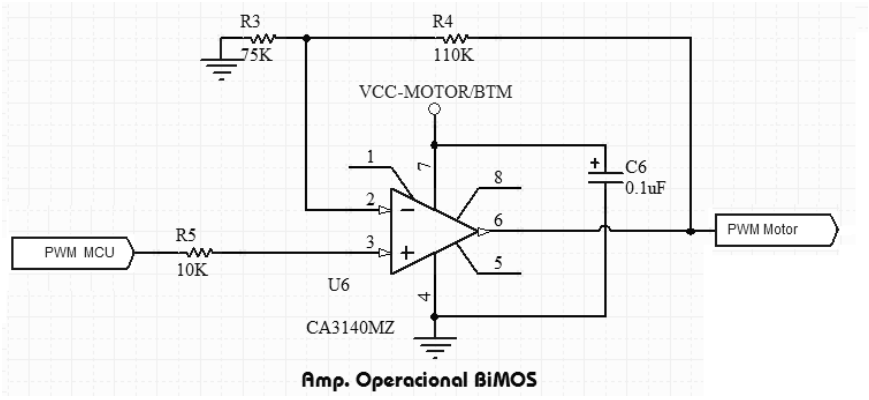


Figura 5.10 Configuración del circuito CA3140 como amplificador no inversor en SIMSAT

El amplificador usado es el CA3140 de la compañía Intersil, que es un amplificador de tecnología BiMOS que posee transistores MOSFETs protegidos (PMOS) en el circuito para proveer una impedancia muy alta a la entrada, con una corriente de entrada muy baja (aprox. 10 [pA]), así como una velocidad de respuesta muy alta. Otra característica que llevó a la selección del amplificador es que fue hecho para equipo que utiliza una fuente referida a tierra, como en automóviles y equipo portátil de instrumentación. En el SIMSAT se tienen fuentes referidas a tierra por lo que se requieren amplificadores que funcionen con fuentes unipolares, pero además, que la señal PWM sea seguida con la fidelidad más alta posible.

Se hicieron pruebas con otros amplificadores para amplificar la señal, en esos casos pudieron seguirla pero tenían un ligero offset que llegaba a alterar las rutinas de control.

En este amplificador debido a que usa transistores de efecto de campo PMOS, en la etapa de entrada tiene una capacidad de voltaje de entrada en modo común de 0.5V por debajo de la terminal de voltaje negativo lo que hace que el nivel de voltaje que pudiera llegar a existir en la entrada no sea amplificado.

5.2.4 Microcontrolador PIC18F4431 del subsistema de Estabilización

Este componente es el “alma” del subsistema, se encarga de decodificar y ejecutar los comandos de estabilización provenientes de la CV, haciendo que esta no desperdicie recursos para activar los actuadores de la tarjeta. El microcontrolador seleccionado fue el PIC18F4431 de la compañía Microchip, dada la popularidad y la facilidad de cargado de programas en estos microcontroladores. La familia 18 de los microcontroladores PIC de Microchip son microcontroladores de 8 bits, ideales para aplicaciones que requieren de un desempeño máximo de 10 a 16 MIPS (millones de instrucciones por segundo) y un espacio en memoria de programa de hasta 128 kb. Poseen una arquitectura Harvard y trabajan con una memoria de programa de 16 bits y memoria de datos de 8 bits, entre otras de sus características principales se encuentran:

- Eficiencia en compiladores de lenguaje C.
- Arquitectura de alto desempeño.

- Maneja el firmware y periféricos lideres de la industria como CAN, USB, ZigBee, TCP/IP.
- Compatibilidad entre periféricos, admitiendo escalabilidad para diseños embebidos complejos.

El microcontrolador PIC18F4431 tiene la particularidad de poseer un módulo dedicado de control de potencia por PWM, con 4 generadores de PWM y hasta 8 canales de salida, además de un módulo de retroalimentación de movimiento que permite monitorear la velocidad de un motor. Estas características fueron ideales para el propósito que se tenía de controlar la velocidad del motor así como de regular el voltaje en las BTMs para regular su fuerza al interactuar con otro campo magnético, esta fue la principal razón por la que se eligió este microcontrolador.

Entre sus características generales se encuentran:

- Un módulo de control de PWM con resolución de hasta 14 bits, con inserción programable de “tiempo muerto”.
- Un módulo de retroalimentación de movimiento, ya sea de 3 entradas de captura o una interfaz de encoder en cuadratura.
- Convertidor analógico-digital de alta velocidad (hasta 200 000 muestras/segundo), con 9 canales disponibles.
- Tecnología nanoWatt, que son modos de operación del microcontrolador para disminuir su consumo.
- Múltiples opciones de oscilador, usando osciladores de cristal, RC, o incluso uno interno.
- Capacidad de ejecutar hasta 10 millones de instrucciones por segundo con un reloj de 40 Mhz.
- Un puerto de comunicación serial síncrona y asíncrona.
- “Watch Dog Timer” con tiempo extendido.
- 16 kb de memoria flash, 768 bytes de RAM y 256 bytes de memoria EEPROM.
- 36 entradas o salidas, 1 timer de 8 bits y 3 timers de 16 bits.
- Rango de operación de temperatura de -40°C a +85°C.

En la figura 5.12 se encuentra el diagrama de bloques del microcontrolador elegido.

Diseño electrónico asociado al Microcontrolador PIC18F4431

Para el uso del microcontrolador en la tarjeta fue necesario asignar sus pines para que interactuara con la demás electrónica, asignación que se muestra en la tabla 5.2. Para su adecuado funcionamiento se le agregaron dos capacitores de tantalio de 0.1µF, en tanto que el oscilador usado para trabajar con el microcontrolador fue de 10 MHz, del cual se usa su PLL interno para aumentar la frecuencia y trabajar a 10 MIPS (Millones de Instrucciones por segundo).

Pin	*Mnemónico*	*Uso*
1	Rx	Receptor de comunicación serial con la CV
2	FLTA	Señal de sobrecorriente en el motor
3	PWM4	Salida de PWM a BTM en X
4	PWM6	Salida de PWM a BTM en Z
5	PWM7	Salida de PWM a BTM en Z

6	Vss	Tierra
7	Vdd	Voltaje de alimentación
9	PWM1	Salida de PWM para el motor
10	PWM2	Salida de PWM a BTM en Y
11	PWM3	Salida de PWM a BTM en Y
14	PWM5	Salida de PWM a BTM en X
16	PGC	Interfaz para el programador
17	PGD	
18	Vpp/MCLR	Usada para inyectar el voltaje de programación y al botón de RESET
19	RA0	Control de dirección del motor
20	RA1	
22	CAP2	Entradas de captura para la señal del encoder
23	CAP3	
24	AN5	Se usa el convertidor A/D para medir la corriente en el motor
28	AVdd	Voltaje de alimentación
29	AVss	Tierra
30	OSC1	Entrada del reloj
32	RC0	Control de dirección de la corriente en la BTMs en X
35	RC1	
36	RC2	Control de dirección de la corriente en la BTMs en Y
37	RC3	
42	RC4	Control de dirección de la corriente en la BTMs en Z
43	RC5	
44	TX	Transmisor de comunicación serial con la CV

Tabla 5.2 Uso de pines del PIC18F4431

Los pines 12, 13, 33 y 34 del microcontrolador no tienen ningún uso en la presentación de circuito escogida. El pin 31 que es OSC2 es una salida de frecuencia del reloj dividida entre 2, y no se usa en el diseño. Los pines 8, 15, 21, 25 al 27 y 38 al 41, son entradas/salidas que quedaron libres en el microcontrolador.

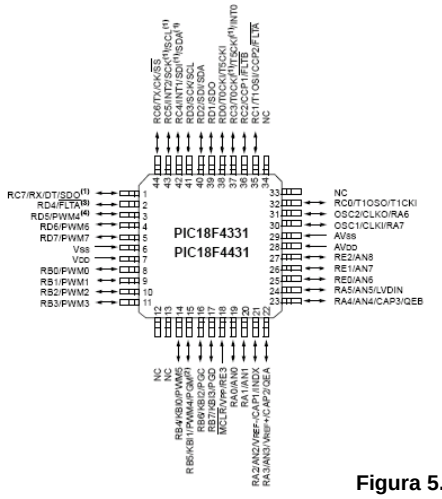


Figura 5.11 Patigrama PIC18F4431 en empaque TQFP-44

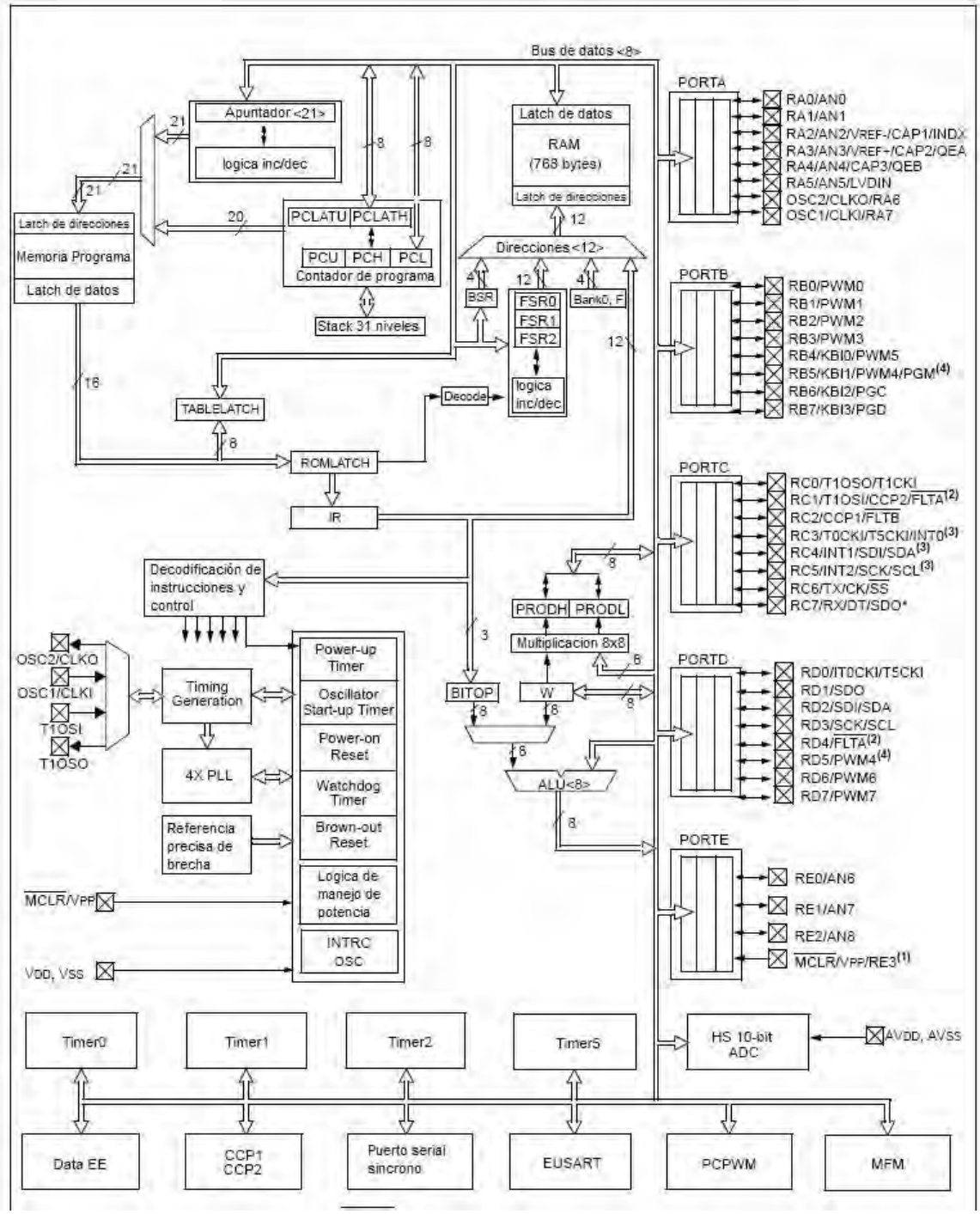


Figura 5.12 Diagrama de bloques del microcontrolador PIC18F4431

5.2.5 Selección del motor de la tarjeta de estabilización activa

Para seleccionar el motor que se utilizaría en conjunto con la rueda inercial para la tarjeta de estabilización activa del SIMSAT fue necesario analizar los tipos de motor que podrían

usarse. Debido a que SIMSAT emplea corriente directa, se necesitaba un motor también de corriente directa (CD).

5.2.5.1 Tipos de Motor de CD

Existen varios tipos de motor de CD, los más populares dentro de sistemas de control son los de imán permanente. Los motores de CD de imán permanente pueden clasificarse de acuerdo a su esquema de conmutación y al diseño de la armadura. En los motores de CD convencionales, como es el caso de los de imán permanente, la conmutación se realiza de manera mecánica a través de las escobillas. Sin embargo hay una clase de motores de CD que realiza la conmutación de manera electrónica, este es el motor de CD sin escobillas.

Motor de CD de imán permanente de núcleo de hierro. La configuración del rotor y estator de un motor de CD de imán permanente puede ser bario-ferrita, Alnico, o un compuesto de tierras raras. El flujo magnético producido por el imán pasa a la estructura del rotor laminado que tiene ranuras. Los conductores de la armadura están localizados en las ranuras del rotor. Este tipo de motor de CD está caracterizado por una inercia del rotor relativamente alta (ya que la parte giratoria está formada por las bobinas de la armadura), una inductancia alta, bajo costo y alta confiabilidad.

Motor de CD de devanado superficial. En este motor los conductores de la armadura están pegados a la superficie de la estructura cilíndrica del rotor, la cual está hecha de discos laminados sujetados al eje del motor. Debido a que en este diseño no se emplean ranuras sobre el rotor, la armadura no presenta el efecto de "rueda dentada". Puesto que los conductores están proyectados entre el hierro de aire que está entre el rotor y el campo de imán permanente, este tipo de motor tiene menor inductancia que el de estructura de núcleo de hierro.

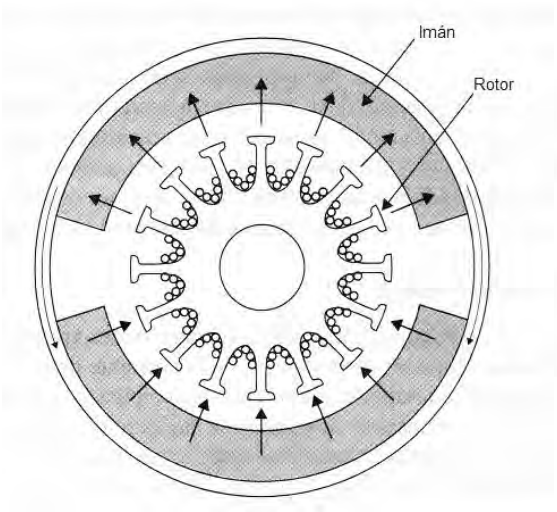


Figura 5.13 Sección transversal de un motor de CD de imán permanente de núcleo de hierro

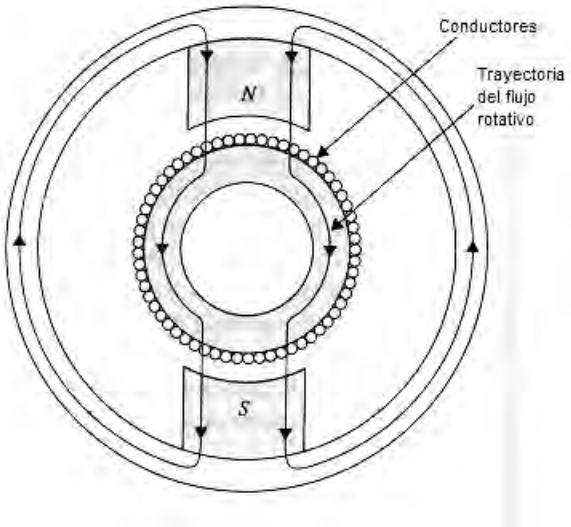


Figura 5.14 Sección transversal de un motor de CD de devanado superficial

Motor de CD de bobina móvil. Los motores de bobina móvil están diseñados para tener momentos de inercia muy bajos e inductancia de armadura también muy baja. Esto se logra al colocar los conductores de la armadura de hierro entre la trayectoria de regreso

de flujo estacionario y la estructura de imán permanente. En este caso, la estructura del conductor está soportada por un material no magnético (normalmente resinas epóxicas o fibra de vidrio) para formar un cilindro hueco. Uno de los extremos del cilindro forma un eje, el cual está conectado al eje del motor. Una vista de la sección transversal de este tipo de motor se muestra en la Fig. 5.15. Ya que se han eliminado todos los elementos no necesarios de la armadura del motor de bobina móvil, su momento de inercia es muy bajo. Como los conductores en la armadura de bobina móvil no están en contacto directo con el hierro, la inductancia del motor es muy baja; valores de menos de 100 μ*H* son comunes en este tipo de motor. Las propiedades de inercia e inductancia bajas hacen que el motor de bobina móvil sea una de las mejores elecciones de actuadores para sistemas de control de alto desempeño.

Motores de CD sin escobillas. Los motores de CD sin escobillas difieren de los mencionados anteriormente en que emplean conmutación eléctrica (en lugar de mecánica) de la corriente de la armadura. La configuración del motor de CD sin escobillas más comúnmente empleado (especialmente para aplicaciones de movimiento incremental) es una en la que el rotor consta de imanes y un soporte de hierro, en el que las bobinas conmutadas están localizadas en forma externa a las partes giratorias, como se muestra en la Fig. 5.16. En comparación con los motores de CD convencionales, presenta una configuración "interna-externa". En función de la aplicación específica, los motores de CD sin escobillas se pueden usar cuando se requiere un momento de inercia bajo, como en el manejo del eje de unidades de disco de alto desempeño empleados en computadoras.

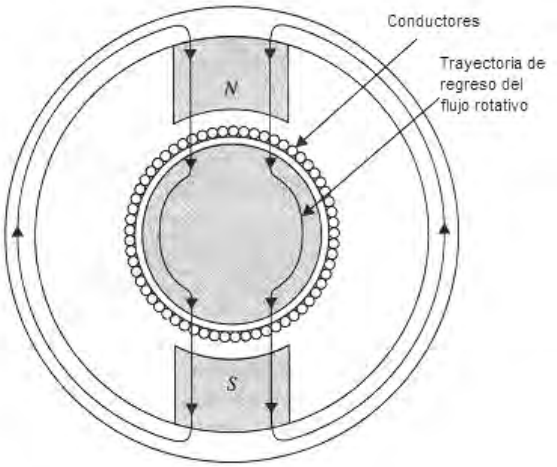


Figura 5.15 Sección transversal de un motor de CD de bobina móvil

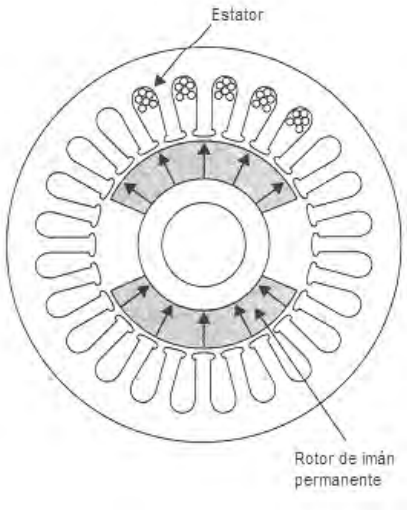


Figura 5.16 Sección transversal de un motor de CD sin escobillas

5.2.5.2 Motor empleado en el SIMSAT

El motor seleccionado para el SIMSAT fue del tipo de devanado superficial, en particular el modelo A2520 de la marca Maxon Motors, se escogió este modelo por el balance que tiene su torque y el número máximo de revoluciones por minuto (14700 rpm) que podía ofrecer. Entre las ventajas adicionales que ofrece este motor es que el fabricante nos proporciona los datos que fueron necesarios para su simulación matemática en un modelo de control y además posee un encoder que se ajusta a la parte trasera del motor

permitiendo mediante electrónica adicional controlar su velocidad y posición, eliminando la necesidad de colocar electrónica y realizar detalles extra de fabricación en el circuito impreso.

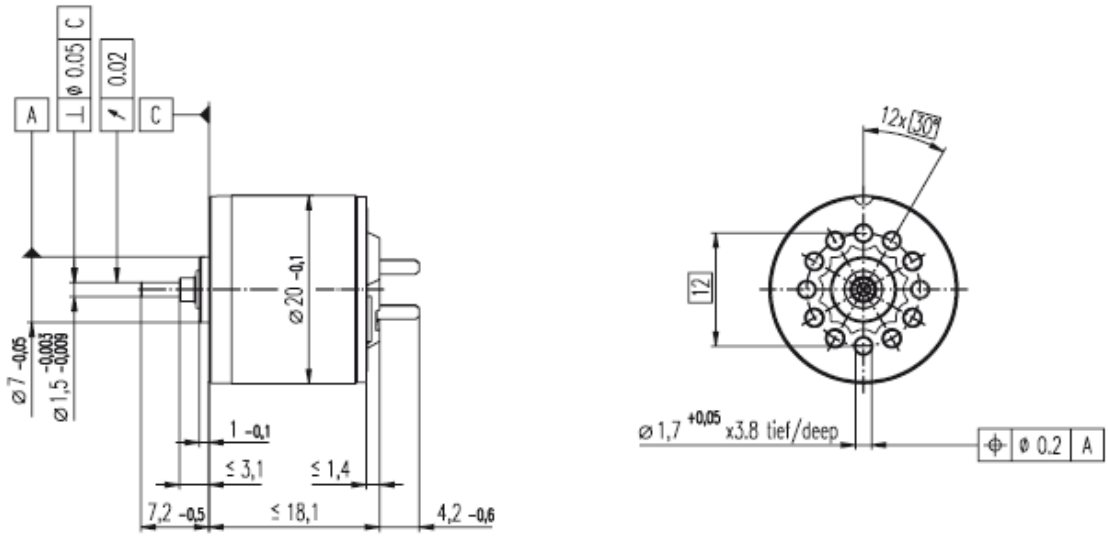


Figura 5.17 Motor A2520 de Maxon Motors

5.2.6 Transductor de corriente en el motor

El transductor nos permite medir la corriente que consume el motor, de tal modo que al percibir un gran aumento de la misma (que podría deberse a que quedara atorado o a que tuviera cualquier otra falla que desencadenara un gran consumo de corriente) envía un pulso al microcontrolador que permite apagar todas las salidas de PWM, desenergizando al motor y las BTMs para evitar que las baterías se descarguen inútilmente.

Este transductor es un circuito integrado de la marca Maxim Semiconductors modelo MAX4071, consiste de un arreglo de 2 amplificadores operacionales de bajo costo capaces de medir la corriente de una carga en ambas direcciones. Este circuito convierte la corriente que circula por la carga en un voltaje. Para que el circuito pueda funcionar adecuadamente es necesario que se le coloque una resistencia externa y seleccionar el nivel de ganancia que tendrá el arreglo de amplificadores operacionales.

Diseño electrónico asociado al transductor de corriente

Para escoger el valor de la ganancia y la resistencia a usar en el sistema, primeramente se usó la siguiente tabla:

Corriente[A]	Resistencia de sensado [mΩ]	Ganancia [V/V]	V_{SENS}[mV]	Voltaje de salida respecto a la referencia [V]
0.075	1000	50	75	3.75
0.05		100	50	5.0
0.75	100	50	75	3.75
0.5		100	50	5.0
3.75	20	50	75	3.75

2.5		100	50	5.0
7.5	10	50	75	3.75
5.0		100	50	5.0
15	5	50	75	3.75
10		100	50	5.0

Tabla 5.3 Tabla de resistencias recomendadas para la medición

El valor seleccionado fue de 100 mΩ (0.1 Ω) y la ganancia seleccionada fue de 50 V/V. Una vez que se tuvieron los valores seleccionados se tiene un voltaje de salida que varía respecto a un voltaje de referencia fijo o ajustable, que en el caso del MAX4071 es fijo y es de 1.5 V. Esta variación se da hacia arriba o por debajo del voltaje de referencia de acuerdo con el sentido del flujo de la corriente a través de la resistencia de sensado. En el caso del SIMSAT va solo por arriba de esta referencia, debido a que la corriente fluye en una sola dirección puesto que la medición de la corriente se hace a través del puente H. Para tener la relación entre la corriente sensada y el voltaje de salida, se usó la siguiente fórmula:

$$V_O = R_S \cdot I \cdot A_V + V_{ref}$$

Donde V_O = Voltaje de salida
R_S = Resistencia de sensado
I = corriente en la carga (el motor y el puente H)
A_V = la ganancia de voltaje
V_{ref} = Voltaje de referencia

Se fijó la corriente máxima en 300 mA, por lo que al sustituir en la fórmula se obtuvo el voltaje que debía ir en el comparador para enviar el pulso al microcontrolador para apagar el motor.

$$V_O = (0.1\Omega)(0.3A)(50V/V) + 1.5V = 3V$$

Para obtener las lecturas de corriente a través del microcontrolador, solo se lee el voltaje de salida a través de un convertidor analógico-digital (A/D) y el programa que interpreta la telemetría en una PC se encarga de hacer la conversión, esto con la finalidad de que el microcontrolador no desperdicie tiempo de procesamiento.

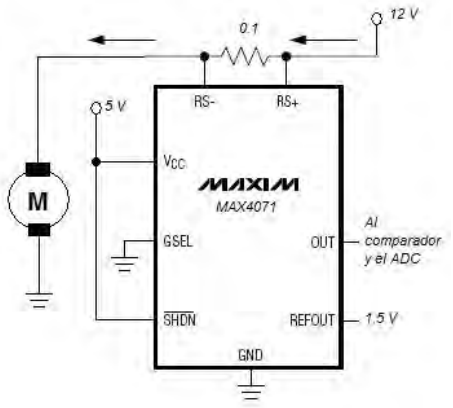


Figura 5.18 Configuración usada del circuito MAX4071

5.3 Arquitectura de la tarjeta de electrónica de sensores y su diseño electrónico asociado

La tarjeta de sensores está compuesta principalmente por:

- 3 Giróscopos de un eje ADIS16100 de Analog Devices.
- 1 Acelerómetro triaxial MA7260QT de Freescale Motorola.
- 1 Brújula electrónica CMPS03 desarrollada por Devantech.
- 1 Microcontrolador PIC18F2520 de Microchip.

La electrónica restante que requiere para su adecuado funcionamiento, tiene LEDs indicadores al igual que la tarjeta de estabilización, para indicar algunos estados de funcionamiento.

5.3.1 Giróscopo

El giróscopo usado en el SIMSAT es el ADIS16100 de la compañía Analog Devices, este giróscopo tiene la característica de ser un MEMS (del inglés, Micro Electro-Mechanical System), un sistema micro electromecánico. Los giróscopos anteriormente resultaban ser muy voluminosos, complejos en su fabricación, además de costosos. Hoy día se tiene una nueva opción para experimentar con ellos gracias a los MEMS logrando unir los sistemas mecánicos y electrónicos en un solo sustrato para ambos sistemas, logrando que la construcción de estos en gran escala sea barata. La nueva desventaja, es que estos sistemas presentan problemas que eran despreciables e imperceptibles en una escala mayor como son la temperatura y la carga electroestática, ahora estando en micro escala estos efectos son muy notorios.

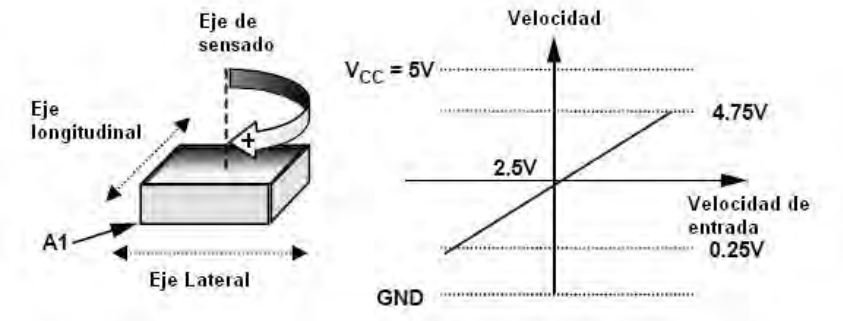


Figura 5.19 Eje de sensado del giróscopo

El ADIS16100 es un giróscopo capaz de medir un rango de velocidad angular de ±300 [º/s] sobre un solo eje, perpendicular a la base del giróscopo. El giróscopo está compuesto de manera interna por el elemento de sensado giroscópico, un sensor de temperatura, 2 canales de conversión analógica-digital y el módulo que se encarga de leer y enviar las lecturas de manera serial a través de un módulo SPI.

El principio de operación del giróscopo se basa en dos estructuras sensibles, que contienen membranas osciladoras de polisilicio que son llevadas electrostáticamente hasta la resonancia. Esto produce la velocidad necesaria para producir la fuerza de Coriolis mientras giran. Dos de los extremos de cada cuadro, ortogonales al movimiento

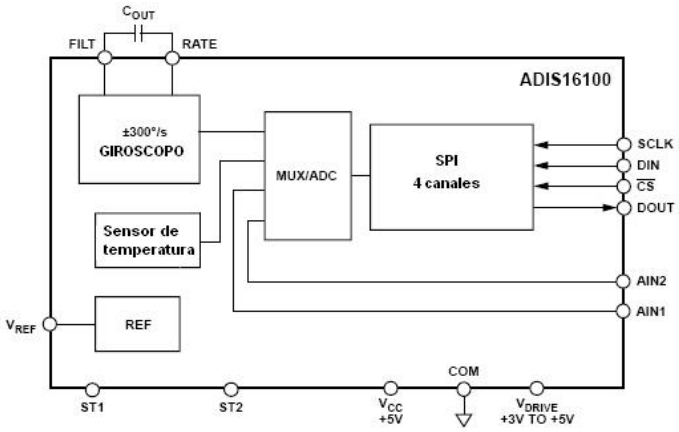


Figura 5.20 Diagrama a bloques del giróscopo ADIS16100

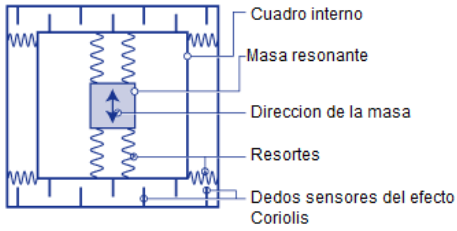


Figura 5.21 Esquema mecánico del giróscopo

de vibración de las membranas, son “dedos” movibles que se colocan entre “dedos” fijos interceptores para formar una estructura capacitiva que sensa el efecto Coriolis (Figura 5.21). Así, cuando el giróscopo no está girando, la distancia entre los dos elementos se mantiene, y por tanto la capacitancia de esta estructura no varía. Sin embargo cuando está girando se producirá el siguiente efecto: el elemento resonante por su movimiento de vibración se encuentra desplazándose hacia fuera del eje de rotación o bien hacia adentro, experimentando por tanto una aceleración o una desaceleración respectivamente producida por el efecto Coriolis (Figura 5.23). Estas aceleraciones/desaceleraciones se traducen en fuerzas en sentidos contrarios que afectan a la masa resonante. Estas fuerzas empujan a la masa más cerca o lejos del elemento fijo, cambiando la capacidad de la estructura capacitiva de forma proporcional a la velocidad de rotación. Estos cambios de capacidad son detectados por elementos sensores que permiten determinar la velocidad de rotación del giroscopio y expresarla en forma de un voltaje de salida. El resultado es alimentado a una serie de etapas de ganancia y demodulación que producen la señal eléctrica de salida. Después de la etapa de demodulación el chip incluye un filtro paso bajas de un polo, el cual filtra las interferencias de altas frecuencias antes de la etapa de amplificación final. Un segundo filtro paso bajas se instala de manera externa por medio del capacitor que limita el ancho de banda C_{out}, como se ve en la figura 5.20. La señal de velocidad es convertida en una representación digital y se transmite por el bus SPI. El diseño de sensor doble en el giróscopo rechaza fuerzas externas como la de gravedad y la vibración.

Su resonador electroestático requiere de 14 a 16V para su operación, dado que solo se alimenta con 5 V es necesario un capacitor de carga que se encuentra integrado en el chip. Su patigrama se ve en la figura 5.22.

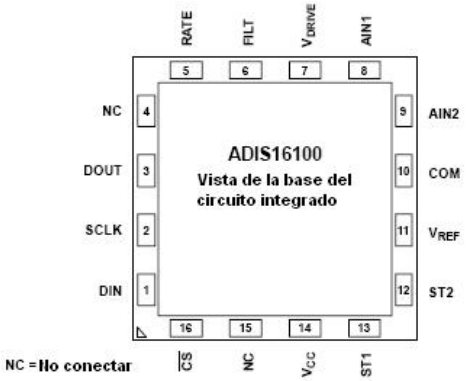


Figura 5.22 Patigrama del giróscopo

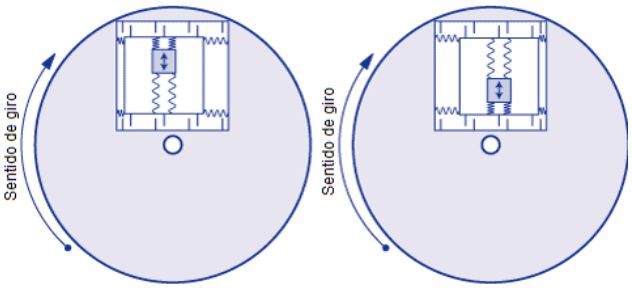


Figura 5.23 Desplazamiento debido al efecto Coriolis

Pin	*Mnemónico*	*Tipo*	*Función*
1	DIN	Entrada	Datos de entrada por el bus SPI
2	SCLK	Entrada	Reloj serial, provee la señal de reloj necesaria para leer y escribir los datos en el bus SPI. También se usa como fuente de reloj para el proceso de conversión del giróscopo.
3	DOUT	Salida	Datos de salida por el bus SPI.
4	NC		No conectar.
5	RATE	Salida	Salida analógica representando la señal de velocidad angular.
6	FILT	Entrada	Conexión al capacitor externo para controlar el ancho de banda.
7	V_{DRIVE}	A	Alimentación del bus SPI. El voltaje alimentado por este pin determina el voltaje al cual funcionará el bus SPI.
8	AIN1	Entrada	Entrada del convertidor A/D 1.
9	AIN2	Entrada	Entrada del convertidor A/D 2.
10	COM	A	Tierra.
11	V_{REF}	Salida	Referencia de 2.5 V
12	ST2	Entrada	Entrada de prueba 1
13	ST1	Entrada	Entrada de prueba 2
14	V_{CC}	A	Voltaje de alimentación
15	NC		No conectar
16	CS	Entrada	Selección de chip. Este pin indica el inicio de transmisión de datos por el bus SPI.

Tabla 5.4 Patigrama del giróscopo ADIS16100

Diseño electrónico asociado al giróscopo

El giróscopo se utilizó 3 veces en la tarjeta de sensores, cada uno de los 3 giróscopos fue utilizado para sensar el movimiento de cada eje geométrico. La forma en que se conectaron estos no fue muy complicada en hardware, solo se conectaron los 3 dispositivos al bus SPI y se emplearon 2 líneas para mandar señales de prueba. El protocolo respectivo será abordado en el siguiente capítulo. Entre los giróscopos hay 3 líneas en común para el bus SPI y la única que no lo es se refiere a la selección de chip (CS, pin 16), esta línea sirve para habilitar solo a un giróscopo en especial cada vez que se quiera tomar una lectura de cada eje. De estar habilitados los 3 habría choques de bits y ninguna de las lecturas seria tomada adecuadamente. Adicionalmente se conectó un capacitor entre los pines 5 y 6 para formar conjuntamente con una resistencia interna en el circuito, un filtro paso bajas para filtrar el ruido de alta frecuencia. Las conexiones se realizaron como lo indica la figura 5.24.

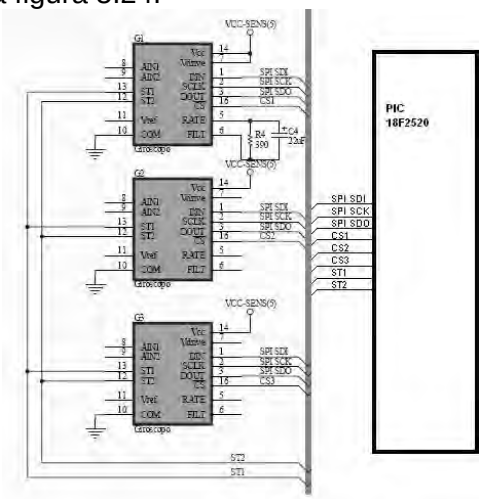


Figura 5.24 Conexión de los giróscopos en el diagrama

5.3.2 Acelerómetro

El acelerómetro usado en el SIMSAT es el MMA7260QT de la compañía Freescale Semiconductor, el cual es un acelerómetro de 3 ejes de tecnología MEMS de bajo costo. Este dispositivo consiste de 2 celdas capacitivamente sensibles y micromaquinadas (celdas “g”) y un acondicionador de señal contenido en un pequeño circuito integrado. Los elementos sensibles están herméticamente sellados al nivel de la oblea de silicio.

La celda g es una estructura formada por materiales semiconductores (polisilicio) usando procesos de fabricación para semiconductores. Este puede ser modelado como un conjunto de varillas adjuntas a una masa central movible que se mueve entre varillas fijas. Las varillas fijas pueden doblarse al someter el sistema a una aceleración.

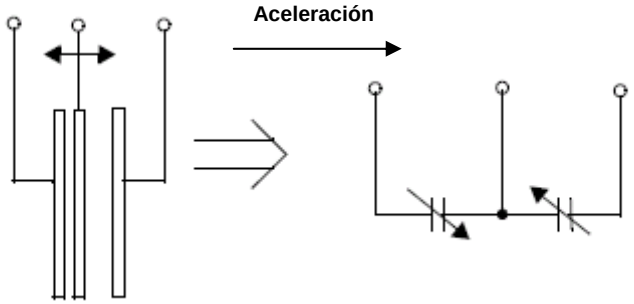


Figura 5.25 Modelo simplificado del transductor de aceleración

Las varillas de la celda g forman un par de capacitores encontrados como se ve en la figura 5.25. La varilla central se mueve con la aceleración y con ello su distancia con las otras varillas entre las que se encuentra, y con esto varía el valor de la capacitancia que forman.

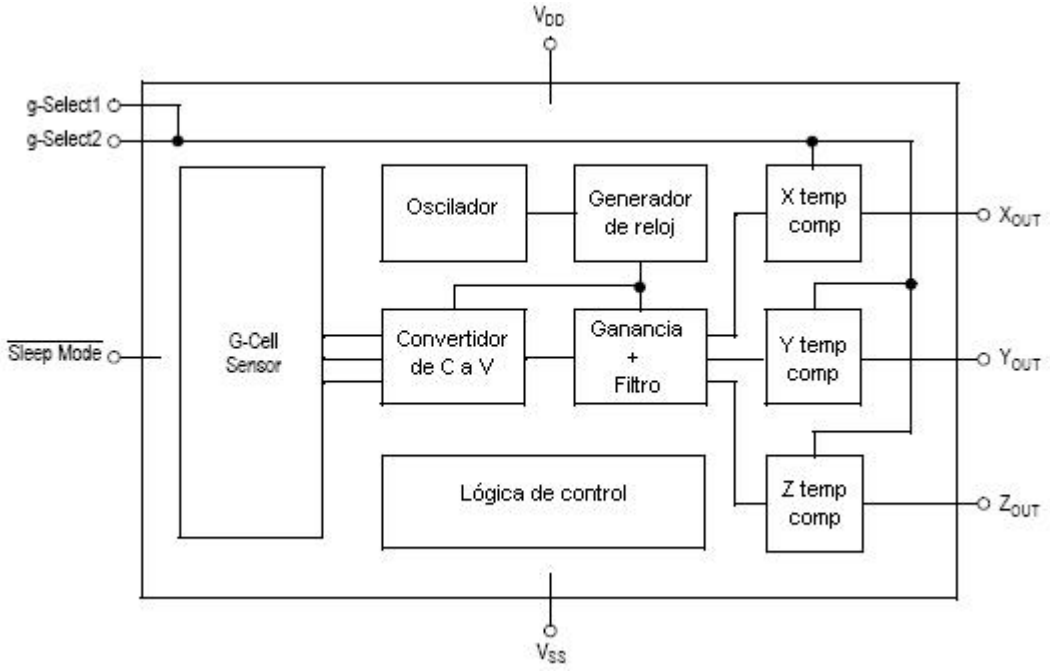


Figura 5.26 Diagrama a bloques del acelerómetro

El acondicionador de señal que posee usa técnicas de capacitores conmutados para medir las capacitancias de las celdas g y para medir la aceleración por medio de la diferencia de capacitancias que genera la celda.

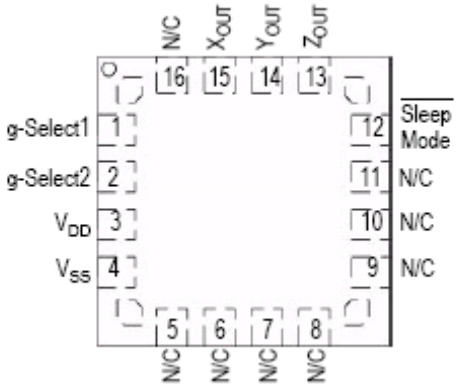


Figura 5.27 Esquema del acelerómetro

Pin	*Mnemónico*	*Descripción*
1	g-select1	Entradas lógicas para seleccionar la sensibilidad
2	g-select2	
3	V_{DD}	Entrada de alimentación
4	V_{SS}	Tierra
5-11	NC	No conectar
12	Sleep Mode	Entrada para habilitar al acelerómetro
13	Z_{out}	Salidas de las direcciones en Z, Y y X respectivamente
14	Y_{out}	
15	X_{out}	
16	NC	No conectar

Tabal 5.5 Patigrama del acelerómetro

Este dispositivo también tiene la posibilidad de ajustar su sensibilidad, dependiendo de la lógica insertada en sus pines 1 y 2 su sensibilidad puede variar de 1.5 g, 2 g, 4 g a 6 g, como se ve en la tabla 5.6. Esta característica es ideal para cuando se requieren diferentes sensibilidades para un mejor desempeño y puede ser cambiada en cualquier momento.

g-select1	*g-select2*	*Rango [g]*	*Sensibilidad [mV/g]*
0	0	1.5	800
0	1	2	600
1	0	4	300
1	1	6	200

Tabla 5.6 Descripción de selección de sensibilidad

Diseño electrónico asociado al acelerómetro

El acelerómetro fue conectado al microcontrolador como se ve en la figura 5.28, los pines X_{out}, Y_{out} y Z_{out} fueron conectados en sus entradas 2, 3 y 4 en las que se utilizó el convertidor A/D para tomar las mediciones de voltaje en cada eje del acelerómetro. Antes de conectar directamente las salidas del acelerómetro al microcontrolador el fabricante recomienda usar un filtro RC con una resistencia de 1 kΩ y un capacitor de 0.1 µF después de la salida del acelerómetro para reducir el ruido del reloj usado en el acelerómetro (causado por la técnica de capacitores conmutados del circuito).

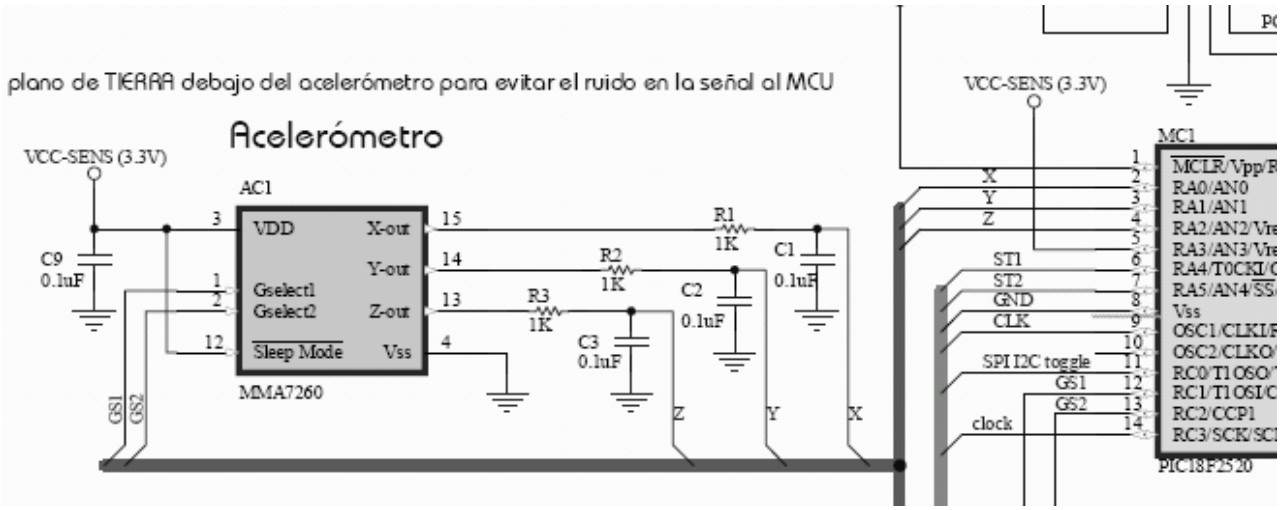


Figura 5.28 Conexión del acelerómetro en el diagrama

5.3.3 Brújula electrónica

La brújula electrónica CMPS03 de Devantech, está formada por un circuito electrónico montado en un pequeño circuito impreso de 34 x 32 mm. Está compuesto por un microcontrolador PIC18F2321, dos sensores magnéticos KMZ51 de Philips y un amplificador operacional dual LMC6032 de Nacional Semiconductor. Los sensores magnéticos que posee sensan el campo magnético de la Tierra. La salida de los dos circuitos sensores (montados de manera ortogonal) se usa para calcular la dirección de la componente horizontal del campo magnético terrestre y así servir como una brújula.

Su patigrama y las funciones de sus pines se muestran en la siguiente figura y tabla.

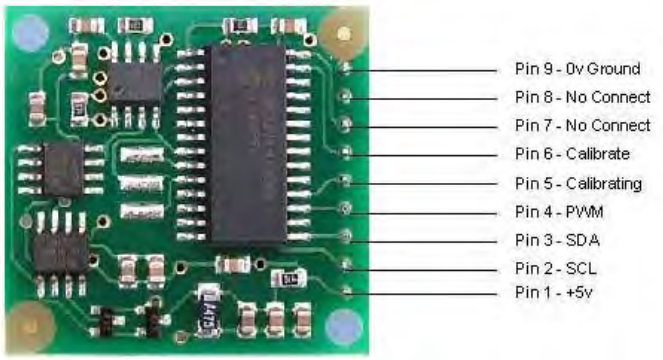


Figura 5.29 Brújula electrónica

Pin	Mnemónico	Función
1	+5V	Voltaje de alimentación, que requiere una salida de 25mA
2	SCL	Pines de interfaz al bus I2C para la entrega de datos
3	SDA	
4	PWM	Salida modulada representando el ángulo.
5	Calibrating	Indica si la calibración de la brújula está en proceso.
6	Calibrate	Utilizado para calibrar la brújula de forma manual.
7	NC	No conectar
8	NC	
9	Ground	Tierra

Tabla 5.7 Funciones de los pines de la brújula

Diseño electrónico asociado a la brújula electrónica

La conexión de la brújula se hizo mediante un bus de comunicaciones serial llamado I^2C (del inglés, Inter-Integrated Circuit), este bus se verá con más detalle en el siguiente capítulo. Para interconectar de manera adecuada este bus con el microcontrolador, se necesita un par de resistencias de "pull-up", el fabricante recomienda algunos valores de resistencias dependiendo de la velocidad a la que se va a trabajar. Como en este caso se va a trabajar a 100kHz se eligió un valor de 4.7kΩ.

Adicionalmente, en esta brújula surgió un contratiempo debido a que al utilizar los protocolos de comunicaciones SPI e I^2C en el microcontrolador estos comparten los mismos pines por lo que se añadió un multiplexor analógico 2 a 1 entre el microcontrolador y ambos sensores, el multiplexor usado fue el 74HC4053. Para cambiar de bus de comunicaciones, se usó un pin del microcontrolador para controlar las salidas en el multiplexor cuando se requiera. La conexión se muestra en la figura siguiente.

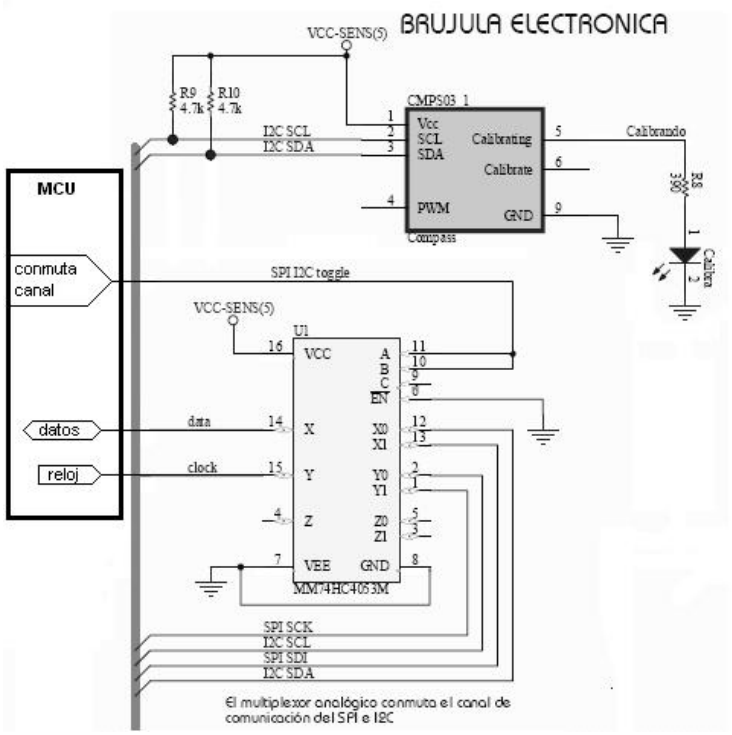


Figura 5.30 Brújula electrónica conexión con el MCU

5.3.4 Microcontrolador PIC18F2520 del subsistema de sensores

Este microcontrolador al igual que en el subsistema de estabilización, desempeña las funciones principales del sistema. En el caso del subsistema de sensores, colecta las mediciones de los sensores que se encuentran en la tarjeta y empaca los datos para enviarlos a la CV. Este microcontrolador posee características similares al usado en la parte de los actuadores, tiene una arquitectura Harvard y trabaja con una memoria de programa de 16 bits y memoria de datos de 8 bits. También tiene la característica de ser bastante eficiente al usarlo con compiladores de lenguaje C conjuntados con una arquitectura de alto desempeño.

El PIC18F2520, figura 5.31, es un circuito integrado de 28 pines, con hasta 25 pines de entrada/salida, 32 Kb en memoria flash para 1536 bytes de memoria RAM y 256 bytes de memoria EEPROM. Otras características que lo distinguen son:

- Módulo ECCP (Enhanced Capture/Compare/PWM), compara, captura y crea una señal de PWM con resolución de hasta 10 bit.
- Módulo MSSP (Master Synchronous serial Port), se encarga de controlar los buses de comunicaciones sincronas SPI e I^2C, ya sea en modo maestro o esclavo.
- Módulo EUSART (Enhanced Universal Synchronous Receiver Transmitter), se encarga de manejar el Puerto serial, ya sea de manera sincrona o asincrona.
- 10 canales de convertidores A/D de 10 bit.
- 2 comparadores analógicos con entradas multiplexadas.
- 4 “timers”, 1 de 8 bit y 3 de 16 bit.
- “Watchdog timer” de tiempo extendido.
- Estructura de oscilador flexible, hasta 4 modos de oscilador.

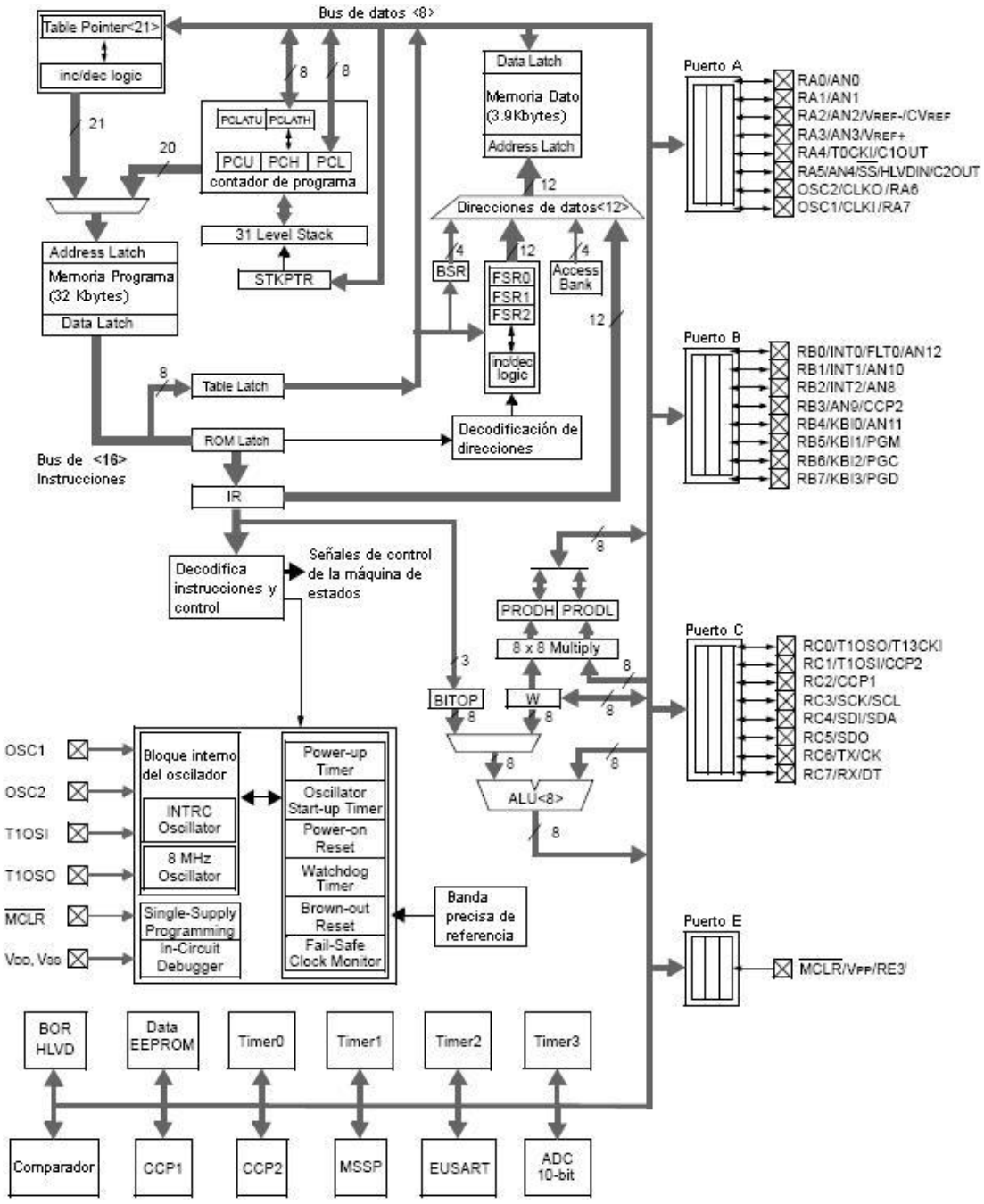


Figura 5.31 Diagrama a bloques del microcontrolador PIC18F2520

Diseño electrónico asociado al Microcontrolador PIC18F2520

De manera similar al microcontrolador en la parte de actuadores del sistema de estabilización, fue necesario asignar los pines de salida, la frecuencia de trabajo y adicionar un capacitor de desacoplo de 0.1 µF de tantalio. El oscilador del microcontrolador es de 10 MHz, este microcontrolador también posee un PLL interno el

cual permite elevar la frecuencia hasta los 40 MHz y trabajar a 10 MIPS. La asignación de pines se muestra en la siguiente tabla.

Pin	*Mnemónico*	*Uso*
1	Vpp/MCLR	Usada para inyectar el voltaje de programación y el botón de RESET
2	AN0	Entradas hacia el convertidor A/D para leer las lecturas del acelerómetro
3	AN1	
4	AN2	
5	Vref+	Voltaje de referencia para realizar las lecturas del acelerómetro
6	RA4	Señales de prueba usadas en los giróscopos para probar que estos funcionen adecuadamente.
7	RA5	
8	Vss	Tierra
9	OSC1	Entrada del reloj
11	RC0	Cambio entre las conexiones con los giróscopos o la brújula
12	RC1	Son utilizadas para seleccionar la sensibilidad del acelerómetro
13	RC2	
14	SCK/SCL	Emite la señal de reloj de sincronía para transferencia de datos en los buses I^2C y SPI
15	SDI/SDA	En el bus SPI es la entrada de datos, y en el bus I^2C es la línea de intercambio de datos
17	TX	Transmisor de comunicación serial con la CV
18	RX	Receptor de comunicación serial con la CV
19	Vdd	Voltaje de alimentación
20	Vss	Tierra
24	RB3	Las líneas de selección de lectura de cada giróscopo
25	RB4	
26	RB5	
27	PGD	Interfaz para el programador
28	PGC	

Tabla 5.8 Uso de pines del PIC18F4431

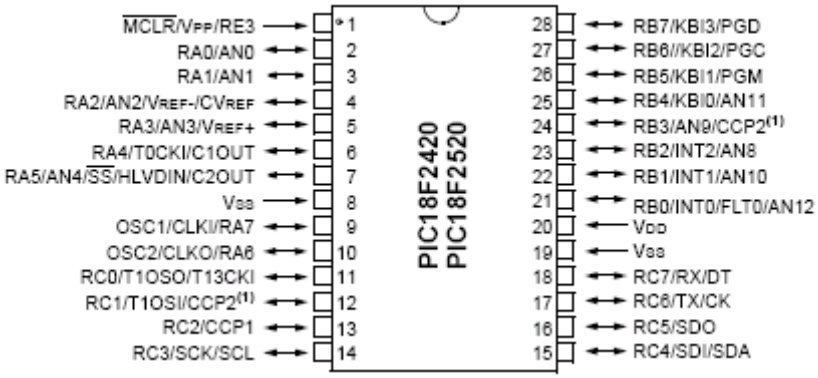


Figura 5.32 Patigrama PIC18F2520 en empaque SOIC-28

5.4 Desarrollo de los circuitos impresos de las tarjetas de estabilización activa y sensores

El SIMSAT se desarrolla como un sistema digital de alta calidad tanto en hardware como en software, por tal razón, cada uno de sus subsistemas se debe desarrollar con base en herramientas que generen productos de alta calidad. En el caso del hardware, se emplean

herramientas de diseño asistido por computadora para generar circuitos impresos de alta calidad, los cuales se envían a producción con un proveedor que ha dado servicio al Instituto de Ingeniería por bastantes años.

5.4.1 Plataforma de diseño Protel DXP

El desarrollo de la tarjeta impresa de estabilización activa se realizó con un programa de diseño llamado Protel DXP, debido a la gran cantidad de características favorables que tiene su entorno de diseño y a que ya se habían realizado varios diseños anteriores de tarjetas electrónicas con muy buenos resultados. Este software de diseño fue hecho por la compañía Altium [www.altium.com], aunque ya tiene nuevas herramientas de diseño, el entorno es básicamente el mismo, dentro de sus características principales se encuentran:

- Es una aplicación integrada a sistemas operativos Windows 2000 y XP.
- Todos los planos de diseño se integran bajo el mismo espacio de trabajo (esquemáticos, PCBs, simulaciones, bibliotecas adicionales de componentes, etc.).
- Amplias bibliotecas de símbolos de componentes.
- Ruteador manual y automático capaz de realizar la mayoría del ruteo necesario en un PCB.
- Sincronización entre diagrama esquemático y PCB.
- Reglas de ruteo.
- Generación de archivos de fabricación de PCBs, etc.

El proceso de diseño y manufactura de un tabloide electrónico mediante el uso del software Protel DXP, se puede describir de manera general como sigue:

1. Creación del proyecto de la tarjeta electrónica.
2. Captura del esquemático.
3. Verificación del diseño eléctrico del esquemático.
4. Generación de lista de redes.
5. Generación de la tarjeta impresa con las dimensiones adecuadas.
6. Posicionamiento de componentes en la tarjeta.
7. Ruteo de redes.
8. Verificación final de las reglas de diseño.
9. Generación de archivos de salida para la manufactura.

El proceso de diseño lleva etapas intermedias como probar previamente los componentes en un diseño preliminar hecho en protoboard, correcciones posteriores, actualizaciones, etc. Los temas descritos anteriormente resumen brevemente el proceso que lleva el diseñar un circuito impreso funcional.

5.4.2 Desarrollo del circuito impreso de la tarjeta de estabilización activa y la tarjeta de sensores

El desarrollo del circuito impreso de la tarjeta de estabilización comprende varias etapas, pero las principales son la captura del circuito esquemático y la definición del PCB, estas etapas se describen a continuación para la tarjeta de estabilización activa.

5.4.2.1 Captura del circuito esquemático

Como se ha dicho ya en el presente capítulo, la tarjeta de estabilización cuenta con un microcontrolador PIC18F4431, 1 transductor de corriente, un oscilador, 2 amplificadores operacionales, 1 puente H completo, 3 circuitos conteniendo 4 medios puentes y demás componentes pasivos, todos estos componentes fueron unidos simbólicamente a través del circuito esquemático. Los símbolos usados para representar a todos los componentes dentro de la tarjeta de estabilización fueron agregados de bibliotecas existentes dentro de Protel DXP o bien creados a partir de un editor de símbolos que posee, con este editor se pueden crear bibliotecas propias y asociar a ese símbolo su huella que va dentro del circuito impreso.

El circuito esquemático es una figura meramente representativa de los componentes del PCB y de cómo fueron conectados, esto nos ayuda posteriormente para ver detalles de su funcionamiento y dar cuenta de las fallas que hubiera de manera rápida. Los esquemáticos del sistema de estabilización se muestran en las figuras 5.34 a 5.38.

5.4.2.2 Verificación del diseño electrónico en el circuito esquemático

Para verificar que los componentes hayan sido adecuadamente conectados en las terminales correspondientes y que no se hayan realizado conexiones inadecuadas entre pines (como puede ser que dos terminales de diferente nombre estén conectadas al mismo punto, lo que podría implicar un corto circuito) Protel DXP incorpora una herramienta que verifica que las conexiones se hayan trazado adecuadamente, (monitoreando de manera permanente el diseño esquemático e indicando cuando ocurre un posible error) esta herramienta se configura a través de una matriz de reglas que puede definirse de acuerdo con las necesidades del diseñador.

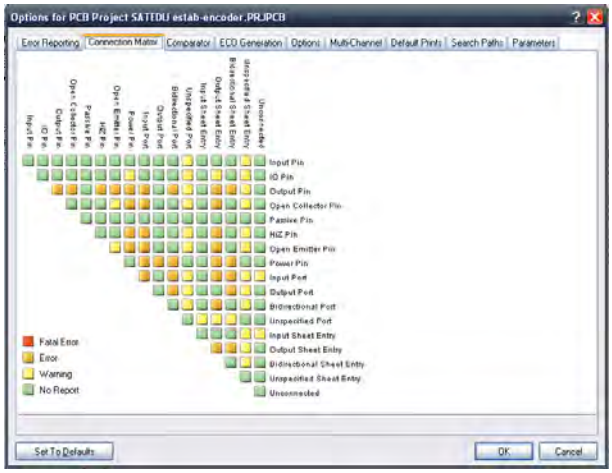


Figura 5.33 Matríz de reglas de conexión

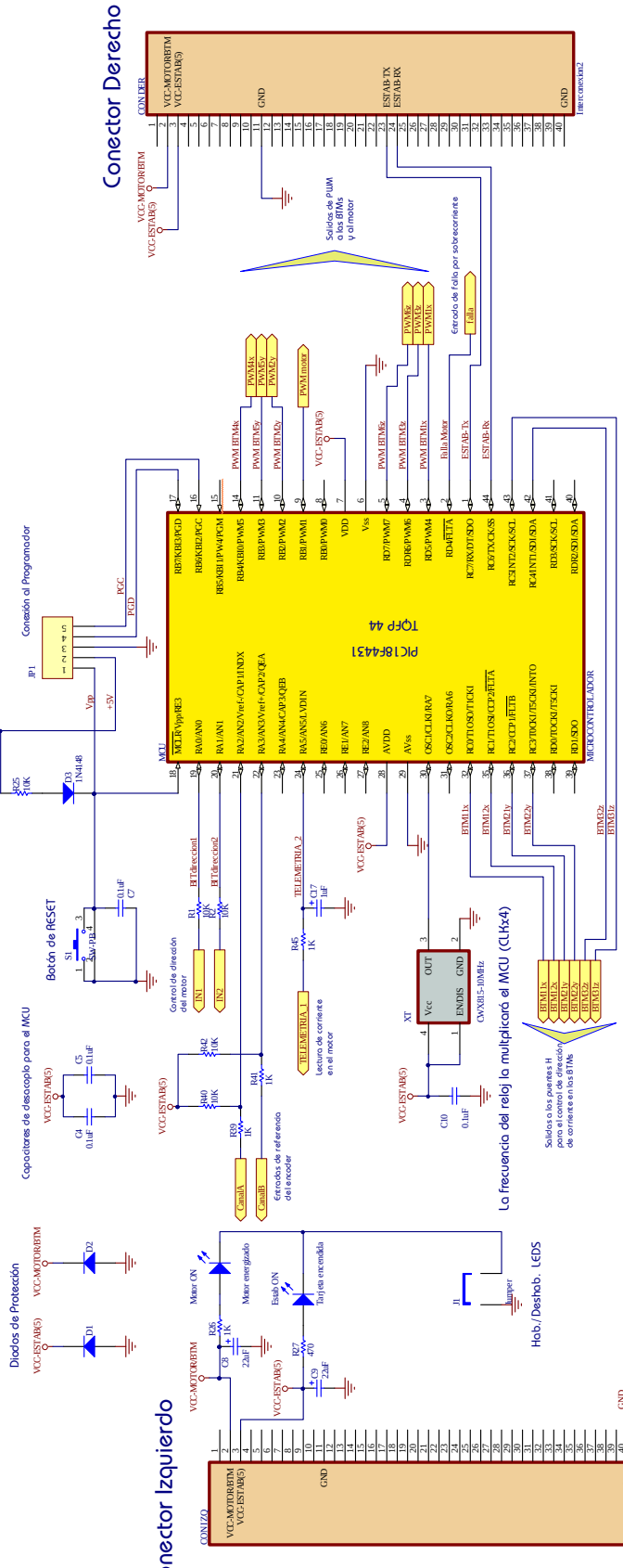


Figura 5.34 Diagrama esquemático de la tarjeta de estabilización del SIMSAT, parte del microcontrolador

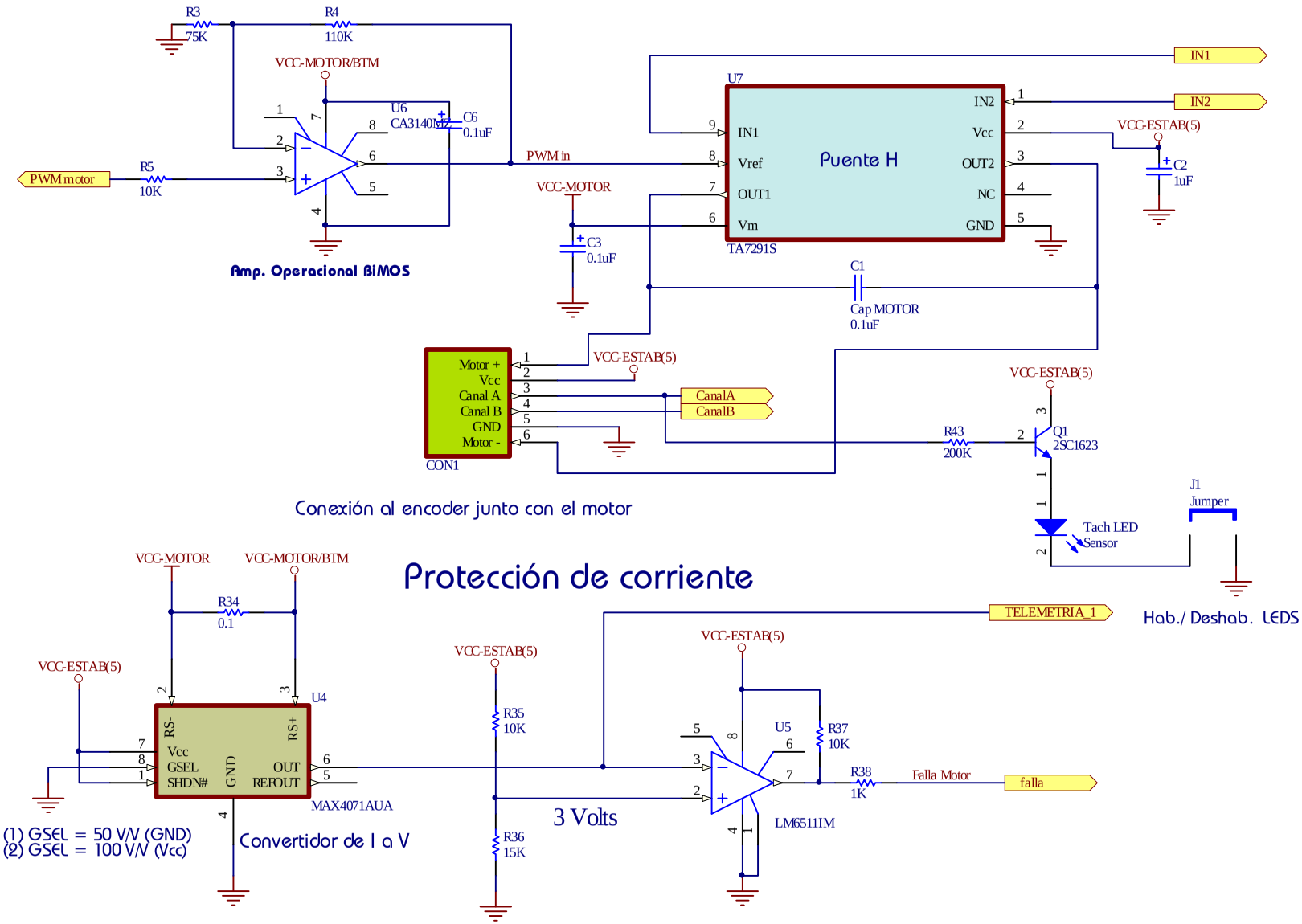


Figura 5.35 Diagrama esquemático de estabilización, parte de control del motor de CD

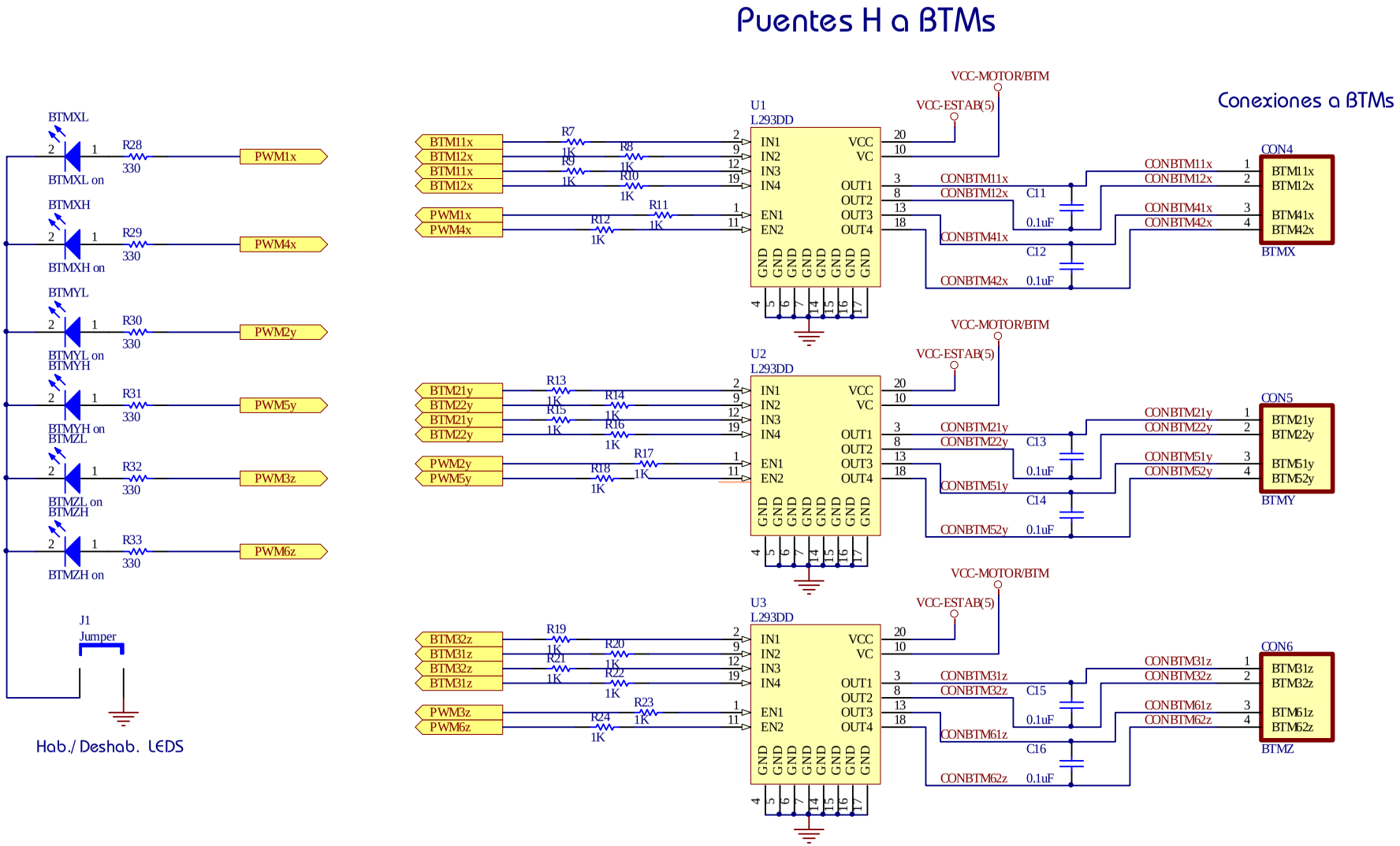


Figura 5.36 Diagrama esquemático de estabilización, conexiones con BTMs y LEDS indicadores

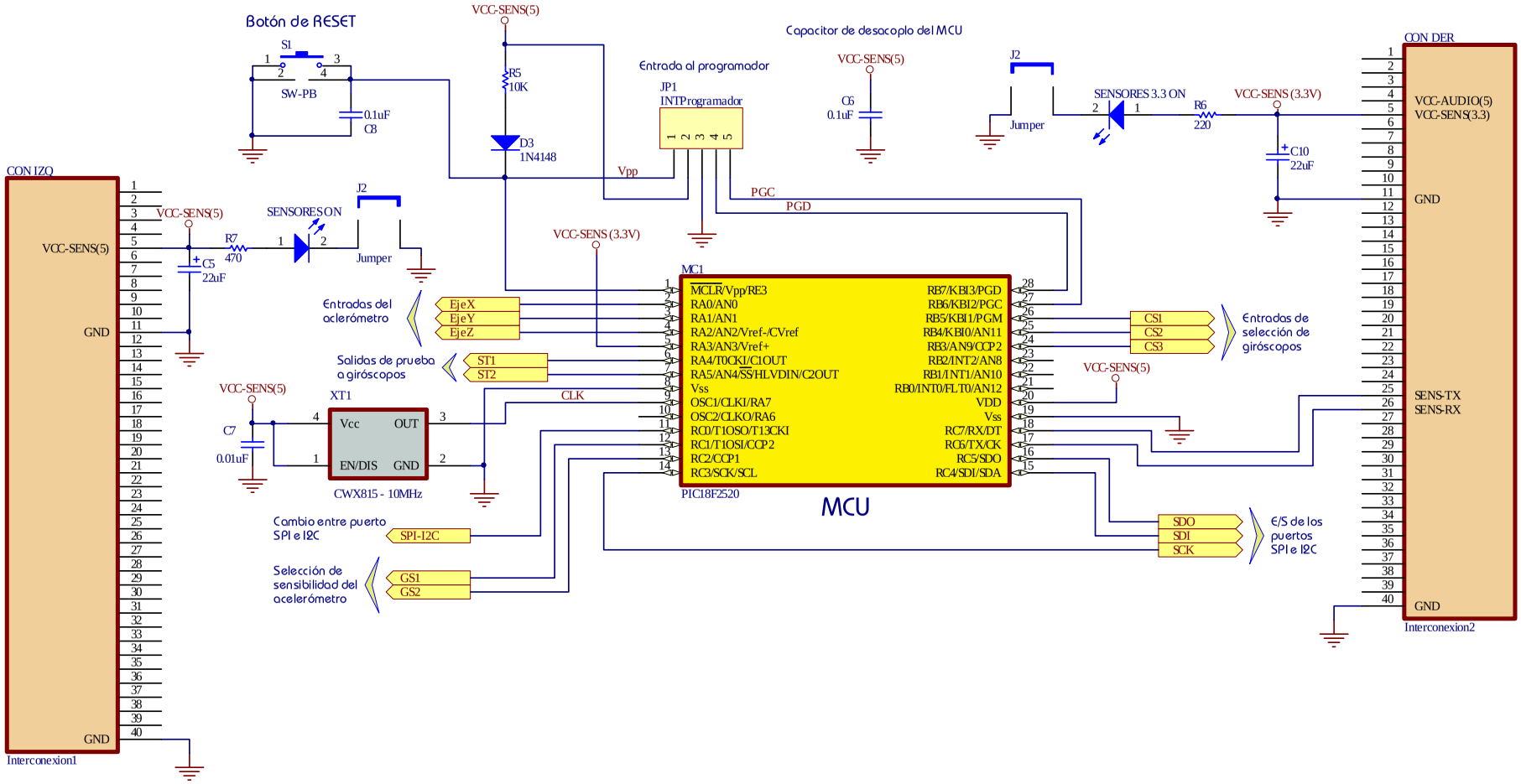


Figura 5.37 Diagrama esquemático de la tarjeta de sensores, parte de MCU y conectores

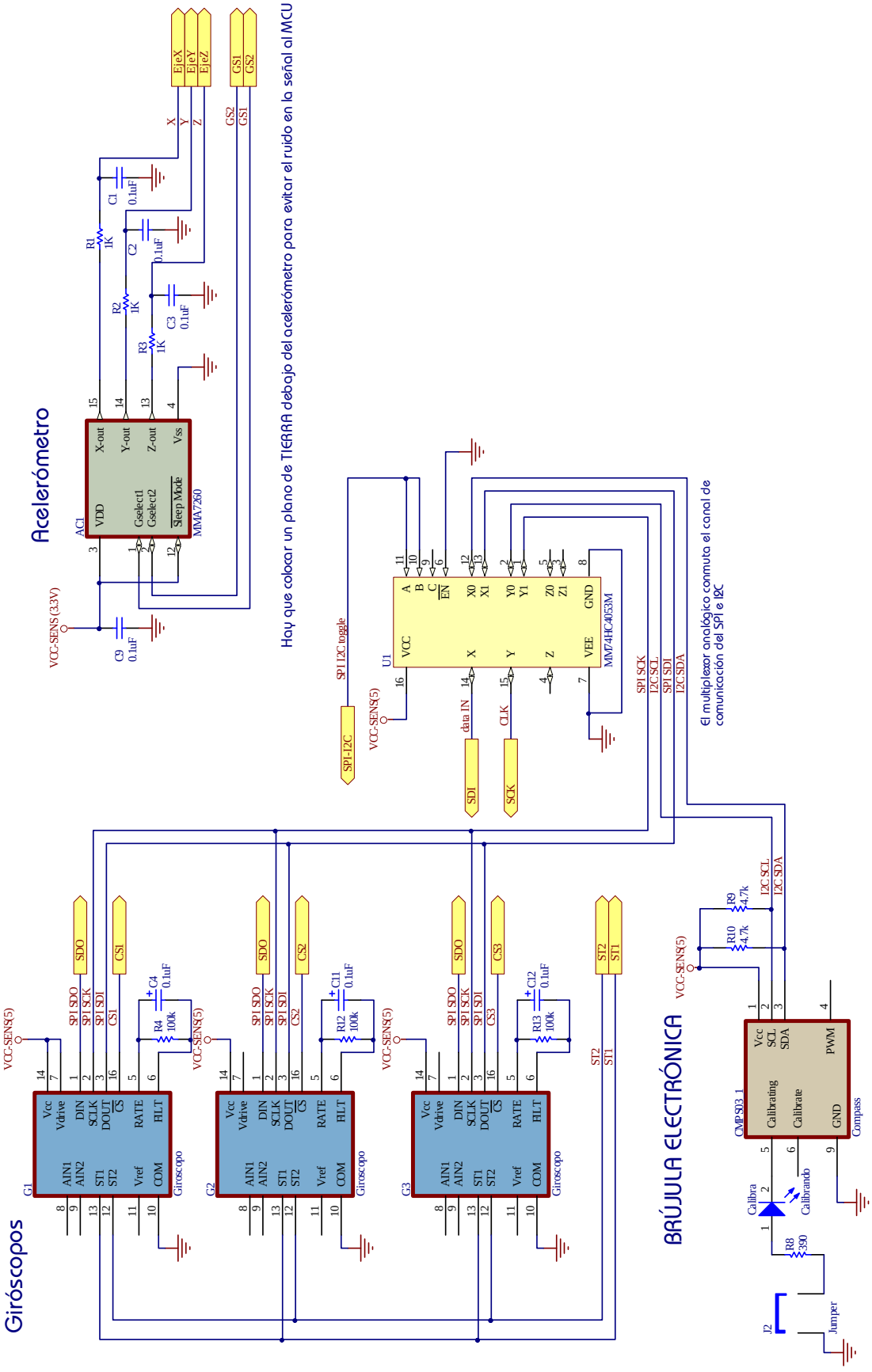


Figura 5.38 Diagrama esquemático de la tarjeta de sensores., conexión de los sensores

5.4.2.3 Generación de plataforma de la tarjeta impresa

Una vez que se tiene perfectamente verificada la conexión adecuada de los componentes, se procede a generar la plataforma en donde serán colocados los componentes correspondientes al circuito esquemático. En esta plataforma se especifican el número de capas del impreso, sus dimensiones, forma, el tipo predominante de componentes que serán colocados, el tipo de vías, el ancho y separación entre pistas, etc.

Para ayudar en la generación de la plataforma, Protel DXP cuenta con un ayudante para la creación de dicha plataforma en donde con pocos pasos se definen rápidamente todas sus características. Una vez generada la plataforma ésta debe anexarse al proyecto que se creó en Protel DXP para que la asocie al circuito esquemático de la tarjeta.

5.4.2.4 Posicionamiento de los componentes en la tarjeta

Una vez generada la plataforma y antes de comenzar a rutear el circuito impreso, es necesario efectuar el posicionamiento de los componentes en la tarjeta. En Protel DXP una vez que se tiene la tarjeta asociada al proyecto, desde la pantalla del esquemático se efectúa el cargado de los componentes y de la red. En este paso, los componentes son colocados de forma aleatoria fuera de la tarjeta. Para hacer un ruteado eficiente de la tarjeta, es muy importante una buena colocación de los componentes, una buena práctica para lograrlo es colocar los componentes de tal forma que las líneas que unen (y que simbolizan la conexión entre componentes) se crucen lo menos posible. Aunado a lo anterior se recomienda seguir las siguientes pautas para la colocación de los componentes:

- Colocar los componentes lógicos que trabajen con señales de mayor frecuencia, cerca del centro de la tarjeta.
- Reunir los componentes en bloques que efectúen funciones semejantes.
- Colocar los componentes analógicos juntos y de ser posible en la periferia de la tarjeta.
- Colocar los capacitores de desacoplo de polarización, lo más cercano posible a los circuitos integrados de mayor frecuencia.
- No colocar componentes de alta frecuencia muy cercanos a los bordes de la tarjeta impresa.

En esta etapa, Protel DXP ofrece una herramienta de posicionamiento automático de componentes llamada “Auto Placer”. Para el caso de la tarjeta de estabilización el posicionamiento fue hecho de manera manual debido a que no eran muchos los componentes y a que se tenía que considerar el espacio que se iba a dejar para la colocación de la rueda inercial sobre la tarjeta. Los componentes que trabajan a mayor frecuencia son solamente 2 el microcontrolador a 40 MHz y el cristal a 10 MHz, estos se colocaron en la parte central de la tarjeta. En las figuras 5.39 y 5.40 se muestran las tarjetas con los componentes colocados y listos para rutear.

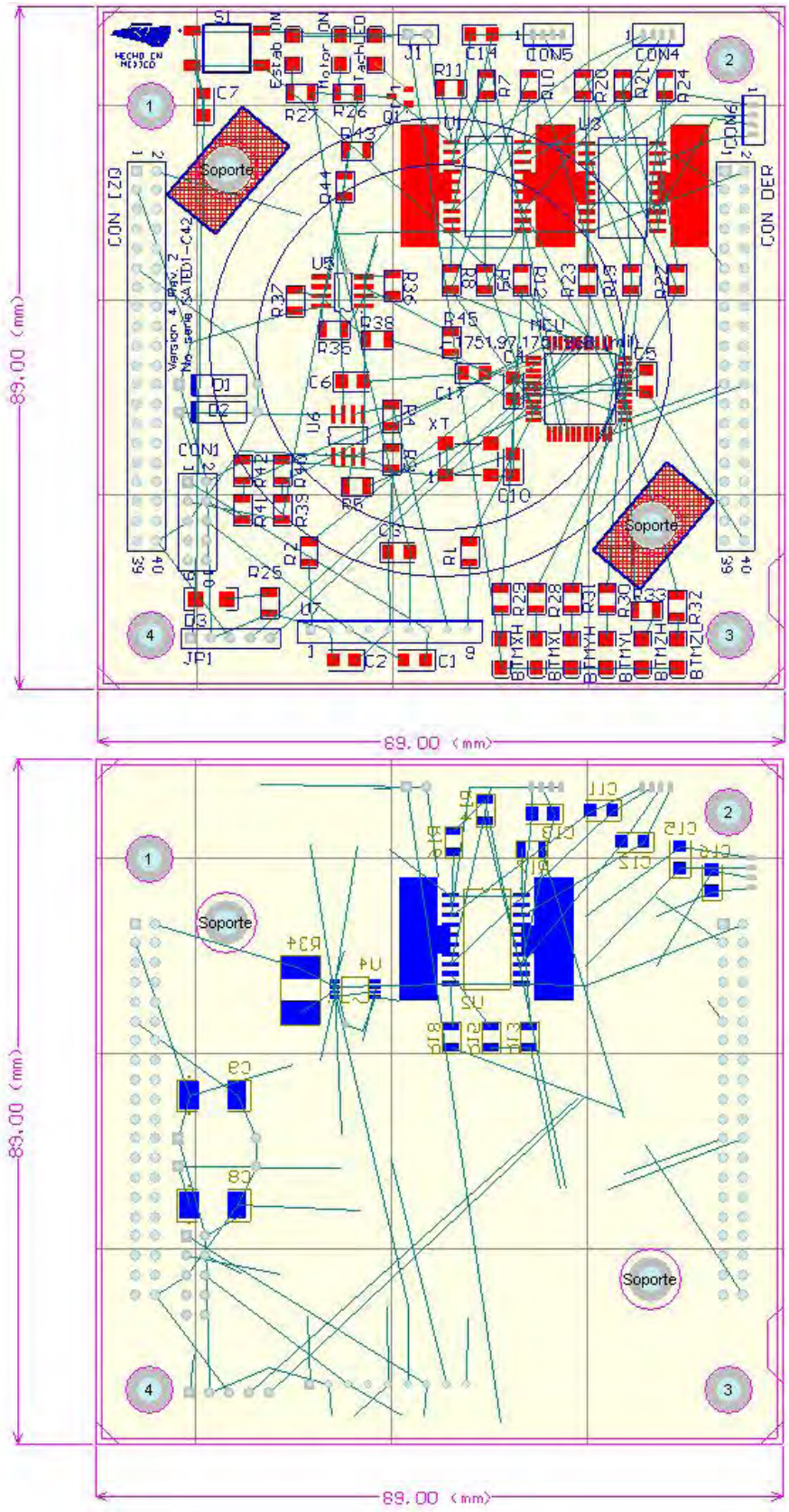


Figura 5.39 Vistas superior e inferior del circuito impreso de estabilización activa antes del ruteo con la relación de conexiones

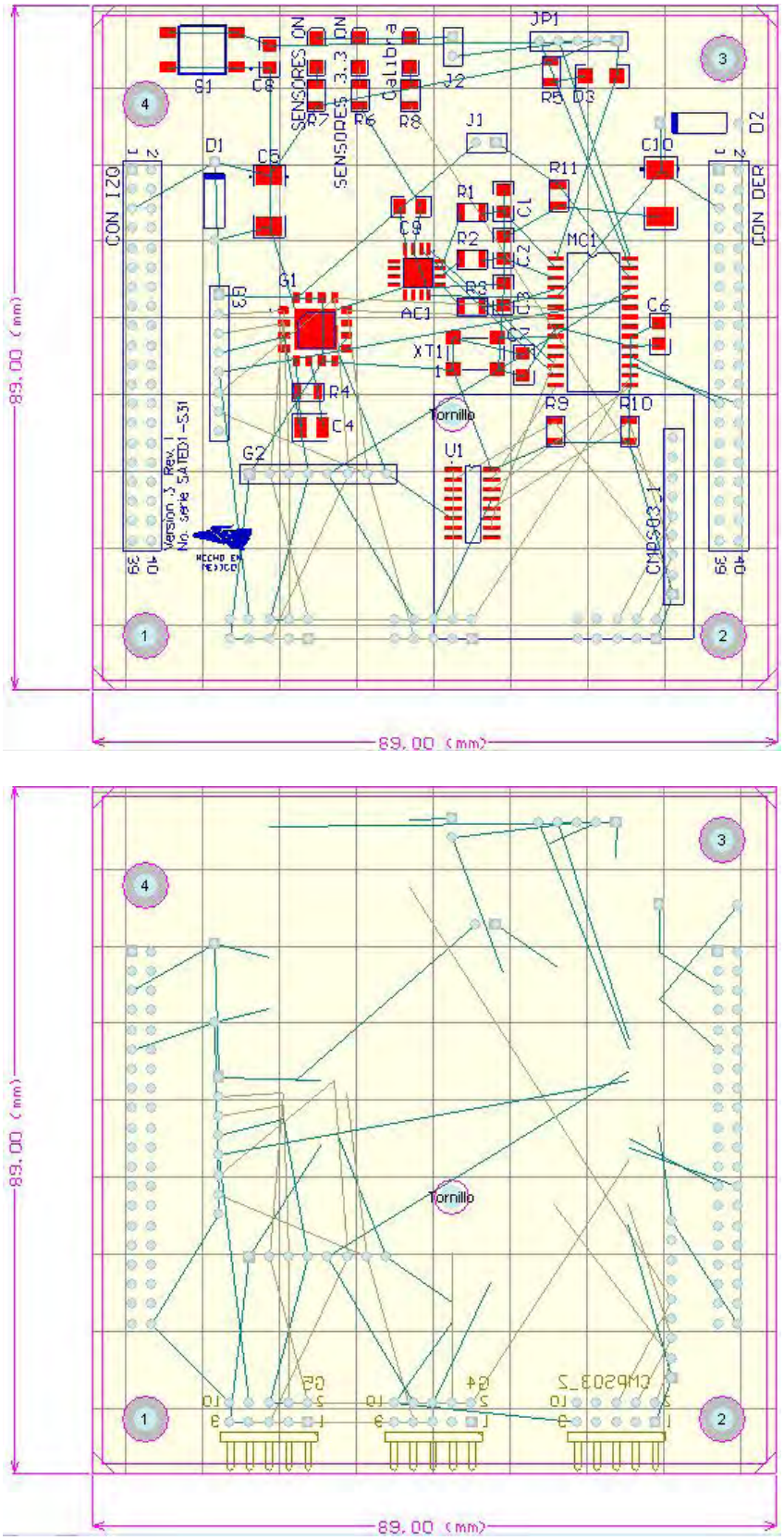


Figura 5.40 Vistas superior e inferior del circuito impreso de sensores antes del ruteo con la relación de conexiones

5.4.2.5 Ruteo de las tarjetas impresas

El ruteo, consiste en trazar las conexiones físicas de los componentes dentro de la tarjeta mediante el uso de pistas y vías. Regularmente, esta etapa resulta muy complicada en función de la cantidad de componentes y de conexiones que contenga una tarjeta. Por tanto, esta etapa es la que consume más tiempo de diseño en las tarjetas impresas, sin embargo, Protel DXP presenta una gran ayuda en este aspecto, ya que su herramienta de ruteo automático ofrece muy buenos resultados en cuanto a calidad y terminación del mismo (más del 90% o 100% en todos los casos analizados).

La herramienta de ruteo Automático que emplea Protel está basada en el uso de algoritmos de ruteo por forma (Shape Based), que han demostrado ser más efectivos que los basados en laberinto (Maze Based).

Para el ruteo de las tarjetas impresas se usan reglas que permiten definir características especiales de ruteo y restricciones para que el ruteo automático que realiza Protel DXP se haga de la manera más adecuada y conveniente en las placas o tarjetas. En el caso de la tarjeta de estabilización se usaron las reglas mostradas en la tabla 5.9 y para la tarjeta de sensores se usaron las reglas mostradas en la tabla 5.10.

Regla	Aplicada a	Propiedades
Regla de ancho de pistas	Polarización del motor	Min. = 10, Max. = 15, Preferido = 15 mils
	Polarización 12V	Min. = 10, Max. = 25, Preferido = 20 mils
	Polarización 5V	Min. = 10, Max. = 20, Preferido = 15 mils
	Conectores al motor y BTMs	Min. = 15, Max. = 25, Preferido = 20 mils
	Tierra	Min. = 5, Max. = 30, Preferido = 25 mils
	Todas las demás pistas	Min. = 8, Max. = 12, Preferido = 10 mils
Capas de ruteo	Toda la tarjeta	Top = Horizontal, Bottom = Vertical.
Separación entre objetos	Entre polígonos de diferentes redes	25mils
	Toda la tarjeta	10mils
Prioridad de ruteo (entre más alto valor, tiene mayor prioridad)	Reloj del MCU	100
	Transmisión serial	90
	Recepción serial	80
	Toda la tarjeta	0
Estilo de vías de ruteo	Toda la tarjeta	Ø interno = 25mil Ø externo = 40 mil

Tabla 5.9 Reglas de ruteo en la tarjeta de estabilización

Una vez que el ruteo automático termina, a veces suele no terminarlo del todo o no terminarlo bien, comete pequeños errores en los acabados, como por ejemplo rutear repetitivamente la misma línea, o hacer un ruteo por caminos largos e innecesarios, por lo que suele ser recomendable primero rutear la rutas consideradas críticas y al final permitir que el ruteo automático termine el trabajo. En este caso no fue necesario, todo el ruteado se hizo bastante bien y solo hubo que remediar pequeños detalles de acabado. Las

figuras 5.41 y 5.42 muestran los PCBs diseñados para el subsistema de Estabilización de SIMSAT.

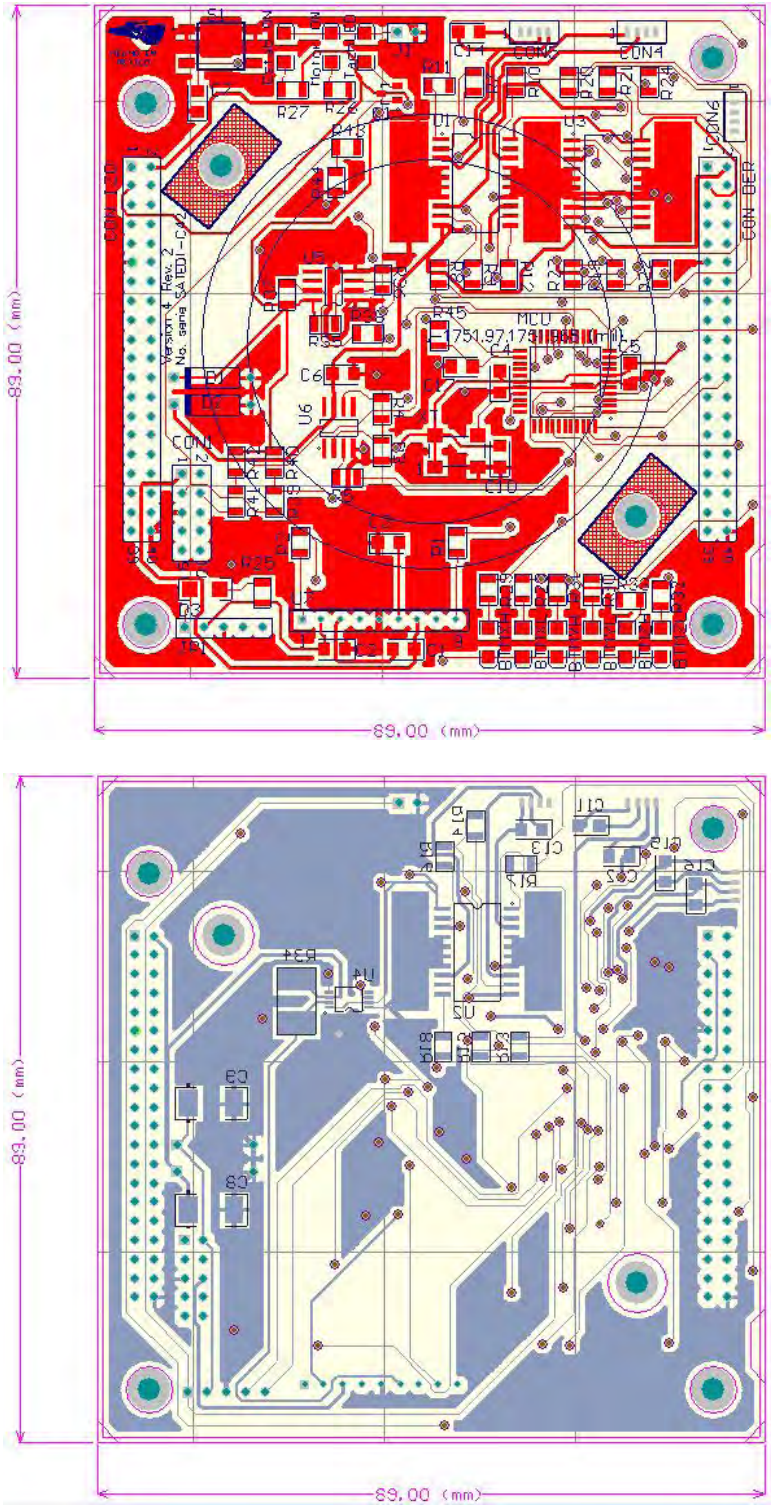


Figura 5.41 Vistas superior e inferior del circuito impreso de estabilización ruteadas

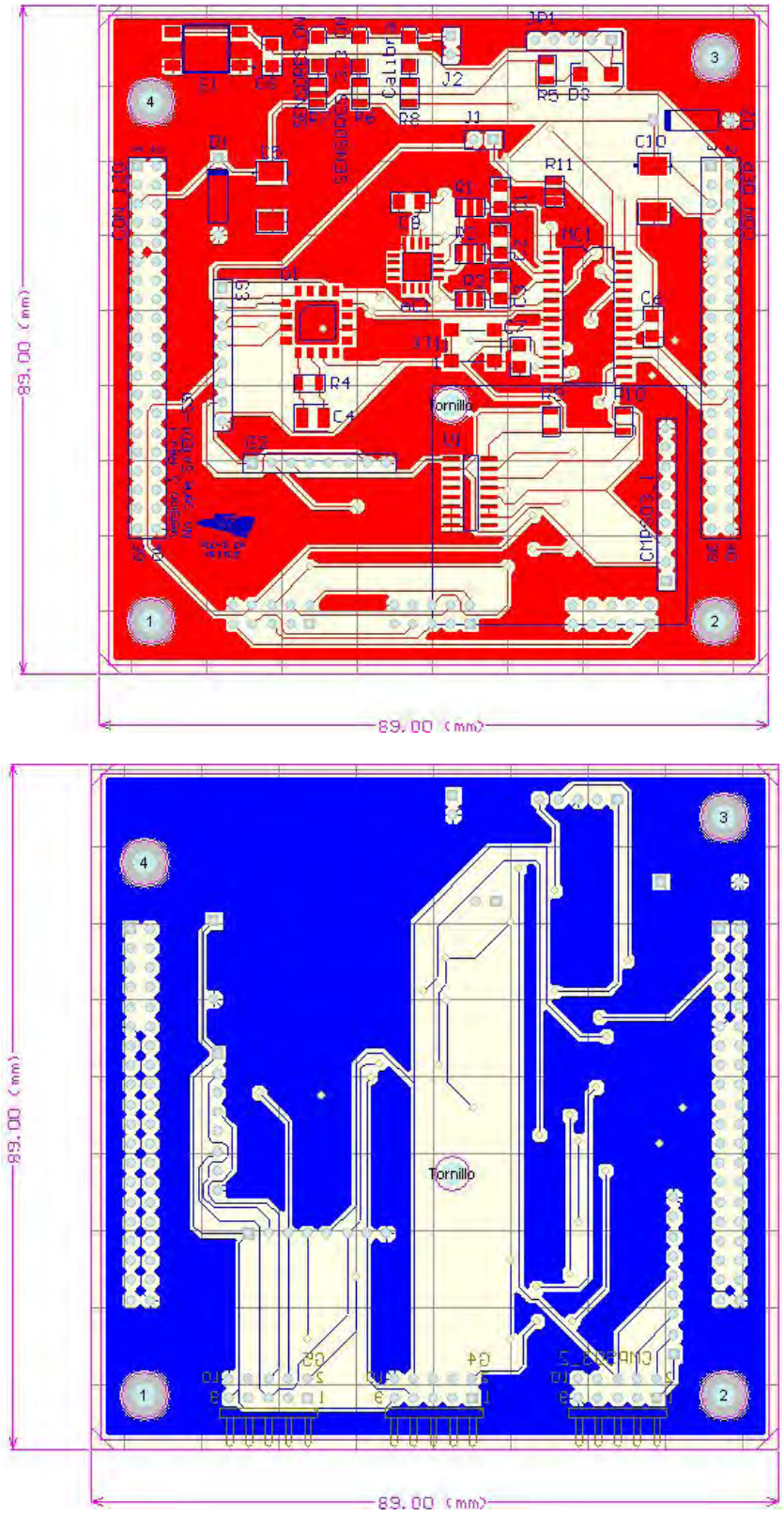


Figura 5.42 Vistas superior e inferior del circuito impreso de sensores ruteadas

Regla	Aplicada a	Propiedades
Regla de ancho de pistas	Todas las pistas	Min. = 8, Max. = 12, Preferido = 10 mils
Capas de ruteo	Toda la tarjeta	Top = Horizontal, Bottom = Vertical.
Separación entre objetos	Entre polígonos de diferentes redes	25mils
	Toda la tarjeta	10mils
Prioridad de ruteo (entre más alto valor, tiene mayor prioridad)	Reloj del MCU	100
	Transmisión serial	90
	Recepción serial	80
	Toda la tarjeta	0
Estilo de vías de ruteo	Toda la tarjeta	Ø interno Min. = 20, Preferido = 25mil Ø externo Min. = 38, Preferido = 45mil

Tabla 5.10 Reglas de ruteo en la tarjeta de sensores

5.5 Manufactura y ensamble de la tarjetas de estabilización activa y sensores

Las tarjetas de estabilización activa y de sensores, como se había mencionado anteriormente, estarán conectadas una sobre la otra. Debido a problemas de manufactura la rueda inercial no ha sido terminada, sin embargo, ambas tarjetas se han empleado ya para la validación del hardware.

Una vez fabricados los circuitos impresos de la computadora de vuelo se llevó a cabo una verificación ocular y de dimensiones de las tarjetas impresas de forma previa al montaje de los componentes, con la finalidad de detectar defectos en el proceso de fabricación de las mismas. También de manera previa al soldado de los componentes, se llevó a cabo una verificación de la continuidad entre las pistas de los impresos con el objeto de detectar posibles cortocircuitos o fallas en el ruteo de las tarjetas, con especial énfasis en la verificación de las líneas que alimentan a los circuitos integrados, que en la tarjeta de estabilización son las líneas de 12V, 5V y tierra, mientras que en la tarjetas de sensores son las líneas de 5V, 3.3V y tierra.

Posteriormente se procedió a soldar los componentes de cada una de las tarjetas, este proceso fue efectuado de forma modular, siguiendo los diagramas de las arquitecturas de la tarjeta de estabilización y de sensores, primero ensamblando los componentes pasivos y de menor tamaño (resistencias, capacitores y LEDs) y al final los componentes activos y de mayor tamaño (circuitos integrados, conectores), de tal forma que cada una de las tarjetas estuviese completamente armada antes de comenzar con la siguiente.

Durante el proceso de ensamblado de las tarjetas, fue de suma importancia tomar ciertas precauciones y recomendaciones, entre las que podemos señalar:

- ✓ Utilizar una pulsera de aterrizamiento eléctrico, durante la manipulación de las tarjetas impresas y/o los componentes.
- ✓ Proteger las tarjetas impresas y los componentes del polvo.
- ✓ No aplicar calor excesivo durante el soldado de los componentes.

- ✓ Verificar la posición correcta de los componentes antes del soldado.
- ✓ Verificar con lupa o cuenta hilos la correcta soldadura de cada uno de los componentes.
- ✓ Utilizar en cantidades moderadas el líquido o pasta para soldar.
- ✓ Verificar constantemente el estado de la punta de cautín, ya que es una herramienta sumamente importante durante el proceso de soldado.

Además, se efectúo un proceso de limpieza (con thinner) y de revisión bajo un microscopio estereoscópico, con equipo e instalaciones prestadas por el Departamento de Ingeniería Ambiental de la División de Estudios de Postgrado de la Facultad de Ingeniería, UNAM.

En las figuras 5.43 y 5.44 se muestran fotografías de las tarjetas antes de ensamblarlas, en tanto que las figuras 5.45 y 5.46 muestran fotografías de las tarjetas después de ensamblarlas.

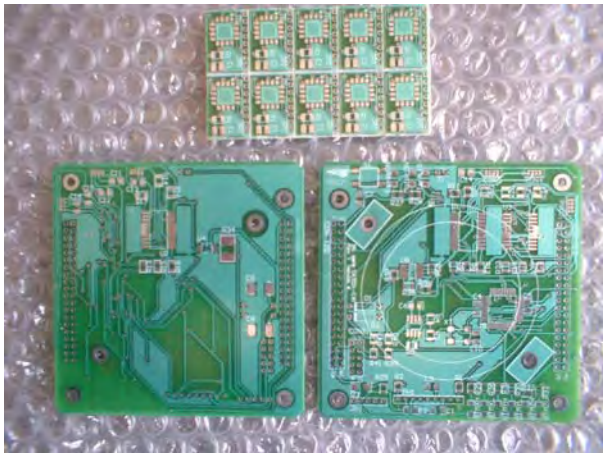


Figura 5.43. Circuito impreso de la tarjeta de estabilización activa

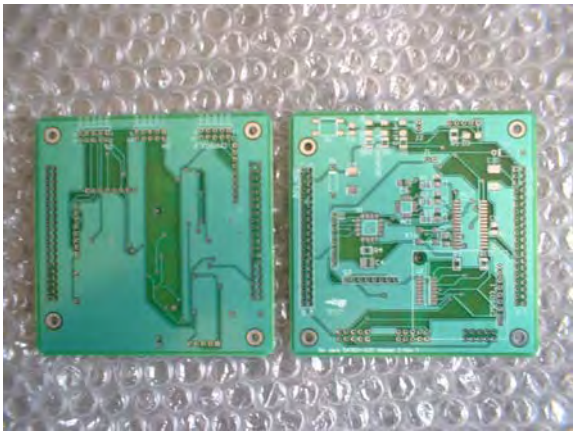


Figura 5.44 Circuito impreso de la tarjeta de sensores de SIMSAT

Figura 5.45 Ensamblado del circuito impreso de estabilización activa

Figura 5.46 Ensamblado de la tarjeta de sensores

5.6 Pruebas preliminares de funcionamiento de las tarjetas de estabilización activa y sensores

Antes de tener el diseño definitivo del circuito impreso fue preciso hacer una serie de pruebas en “protoboard” con los microcontroladores y demás circuitos de empaquetados en línea doble, todos los que fue posible conseguir, el único circuito con el que no fue posible experimentar fue el convertidor de corriente a voltaje (MAX4071).

5.6.1 Cargador de programas al microcontrolador

En la puesta en operación de la tarjeta de medios de estabilización, el primer aspecto que había que verificar en funcionamiento era el microcontrolador de la tarjeta, para ello se procedió a cargarle un programa a través del programador que fue construido para este propósito y con el cual fue posible hacer las pruebas de puesta en operación previas al diseño final del circuito impreso.

El programador construido fue hecho a partir de un diseño publicado en la página de Blueroom Electronics correspondiente a una pequeña empresa canadiense que amablemente comparte sus diseños con todos aquellos interesados en el tema. A su vez, pone a la venta sus diseños ya armados o los kits para armar. El programador es el que nombran INCHWORM, un programador y emulador en circuito compatible con el programador ICD2 de Microchip, este programador puede programar la mayoría de los microcontroladores PIC basados en memoria FLASH. Este programador representó un medio económico y práctico para programar los microcontroladores usados en el proyecto SATEDU.

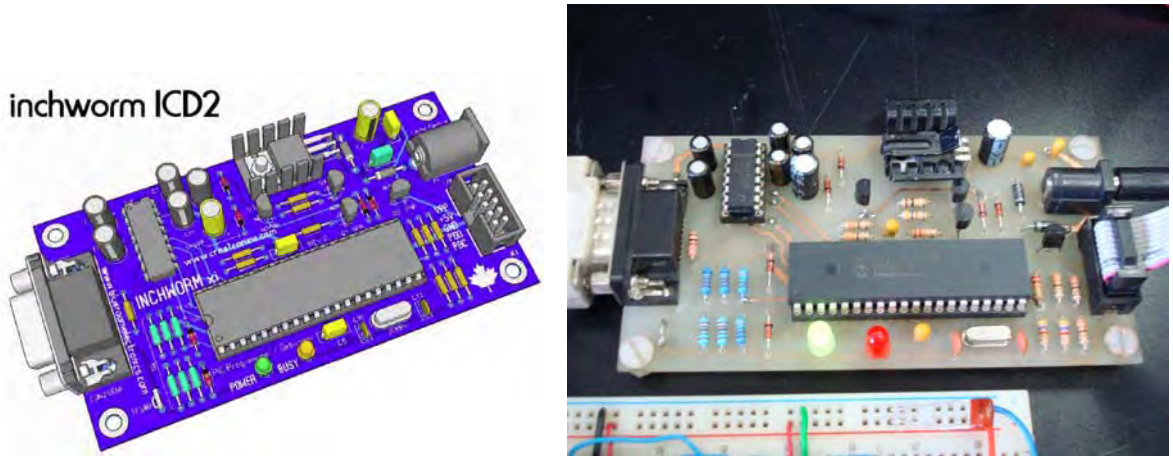


Figura 5.47 Programador Inchworm

5.6.2 Pruebas en protoboard

Las pruebas en protoboard fueron realizadas de manera previa a la construcción del circuito impreso para corroborar la funcionalidad del circuito que se deseaba implantar en circuito impreso. La primera parte hecha con los protoboards fue realizada para validar el esquema de estabilización. Los experimentos validados en los protoboards para cumplir con los objetivos del subsistema fueron:

1. Prueba de cargado de programas y funcionamiento del microcontrolador PIC18F4431 (encendido y apagado de un LED), así como la selección del cristal.
2. Prueba de funcionamiento del puerto serie del microcontrolador.
3. Prueba de funcionamiento del módulo PWM, conectado al motor a través del puente H.
4. Prueba de funcionamiento del módulo de retroalimentación, usado con el motor y un sensor de efecto Hall.
5. Prueba de uso de convertidor A/D para la lectura de corriente a través del convertidor de corriente en voltaje.
6. Prueba de módulo PWM con los puentes H en las BTMs.
7. Pruebas de envío, recepción, construcción y decodificación de telemetría.
8. Pruebas de integración de software.

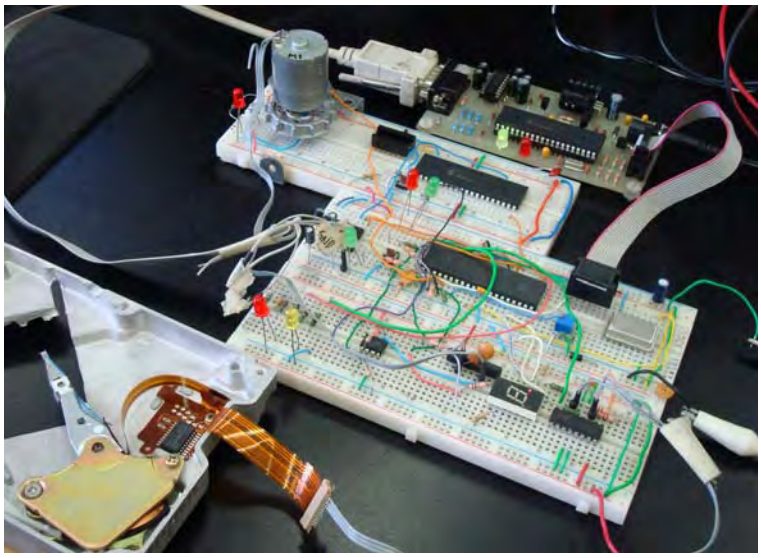


Figura 5.48 Protoboards de prueba del subsistema de estabilización activa

De igual manera para el subsistema de sensores varias pruebas se realizaron en protoboard, pero debido a que se carecía de varios sensores no fue posible realizar todas las pruebas hasta que se tuvieron las tarjetas impresas armadas. Las pruebas que fue posible hacer en protoboard fueron:

1. Prueba de cargado de programas y funcionamiento del microcontrolador PIC18F2520 (encendido y apagado de un LED) así como la selección del cristal.
2. Prueba de funcionamiento del puerto serie del microcontrolador.
3. Prueba de uso múltiple de convertidores A/D para la lectura del acelerómetro.
4. Prueba de lectura de la brújula digital a través del bus I^2C.
5. Prueba de uso del bus SPI.
6. Pruebas de envío, recepción, construcción y decodificación de telemetría.
7. Pruebas de integración de software.

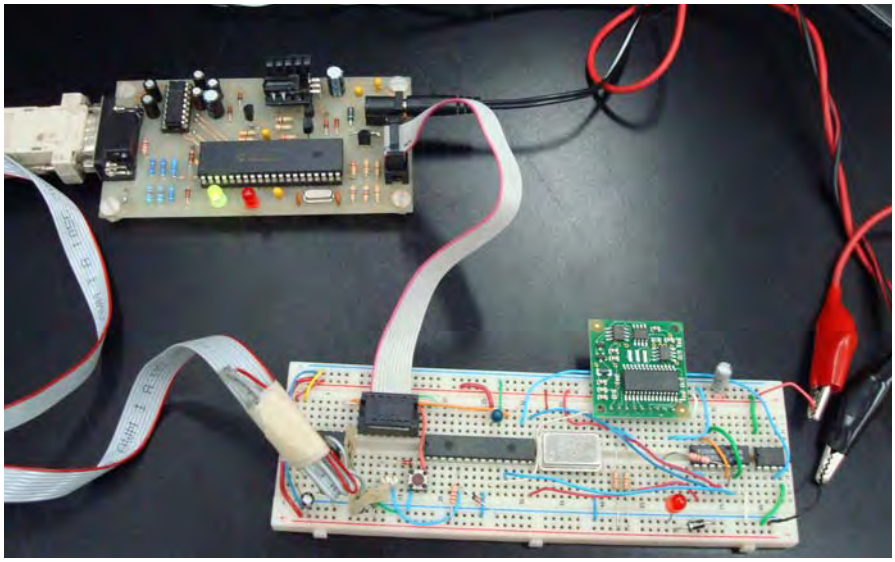


Figura 5.49 Protoboard de prueba del subsistema de sensores

Bibliografía

1. Ginsberg, G. "*Printed Circuit Design"*, Ed. McGraw-Hill. EEUU, 1991.
2. Franco, S., "*Diseño con amplificadores operacionales y circuitos integrados analógicos"*. 3ª edición. Ed. McGraw-Hill, México, 2005.
3. Kuo, B. "*Control Automático"*. 7ª edición. Ed. Prentice Hall, México, 2005.
4. Altium Board-level design system. *"Introducing Protel DXP Tutorial"*. Hoja de tutorial, 2002. [www.altium.com]
5. Analog Devices. *"ADIS16100 ±300 º/s Yaw rate Gyro with SPI Interface*", Hoja de datos, 2006.
6. Freescale. **"***MMA7260QT ±1.5-6g Three Axis Low-g Micromachined Accelerometer",* Hoja de datos, 2006.
7. Microchip Technology. **"***PIC18F4431, Enhanced Flash Microcontroller with nanoWatt Technology and High Performance PWM and A/D",* Hoja de datos, 2003.
8. Microchip Technology. *"PIC18F2520, Enhanced Flash Microcontroller with 10-bit A/D and nanoWatt Technology".* Hoja de datos, Microchip, 2004.
9. Maxon Motors. Catálogo en línea, 2006. [www.maxonmotors.com]
10. Devantech. *"CMPS03 Compass Module"***.** Hoja de datos técnica, 2007. [http://www.robot-electronics.co.uk/htm/cmps3tech.htm]
11. Blueroom Electronics. **"***Inchworm ICD2 Assembly Manual*". Hoja de datos, 2007. [http://www.blueroomelectronics.com]
12. Geen, J. y Krakauer, D. *"New iMEMS Angular-Rate Sensing Gyroscope"*. Analog Dialogue, Vol. 37, pp 1-4, ADI Micromachined Devices Division, EEUU, marzo, 2003.

Capítulo 6

Capítulo 6

Software de operación para las tarjetas de estabilización activa y sensores para el SIMSAT

6.1 Introducción

Debido a la naturaleza de algunos de los componentes que han sido incorporados en las tarjetas como el par de microcontroladores PIC, uno en la tarjeta de estabilización y otro en la tarjeta de sensores, estos necesitan de cierta programación para ejecutar sus tareas asignadas.

En el presente capítulo se habla sobre la programación que fue desarrollada para que los microcontroladores (montados en los subsistemas de estabilización activa y sensores) pudieran realizar sus tareas. También se detallan las herramientas que fueron utilizadas para programar los microcontroladores y la interfaz de prueba (hecha en Visual Basic 6) para probar el adecuado funcionamiento e interacción entre las tarjetas de estabilización y sensores con la CV.

6.2 Plataformas de desarrollo de software

Las plataformas de desarrollo de software en el presente trabajo fueron tres:

- MPLAB IDE de Microchip para programar a los microcontroladores.
- Un complemento para compilar los programas en lenguaje C en lugar de lenguaje ensamblador, esto debido a la longitud y complejidad del programa por desarrollar.
- Visual Basic 6, para desarrollar una interfaz lo bastante amigable para aplicar pruebas fáciles y rápidas entre las tarjetas desarrolladas y la CV.

6.2.1 MPLAB IDE

MPLAB IDE es un programa que corre bajo el sistema operativo Windows con el objeto de desarrollar aplicaciones para los microcontroladores y controladores digitales de señales de Microchip. Es llamado un Ambiente de Desarrollo Integrado (IDE, del inglés Integrated Development Enviroment) porque provee un solo ambiente integrado para desarrollar código de programación para los microcontroladores PIC.

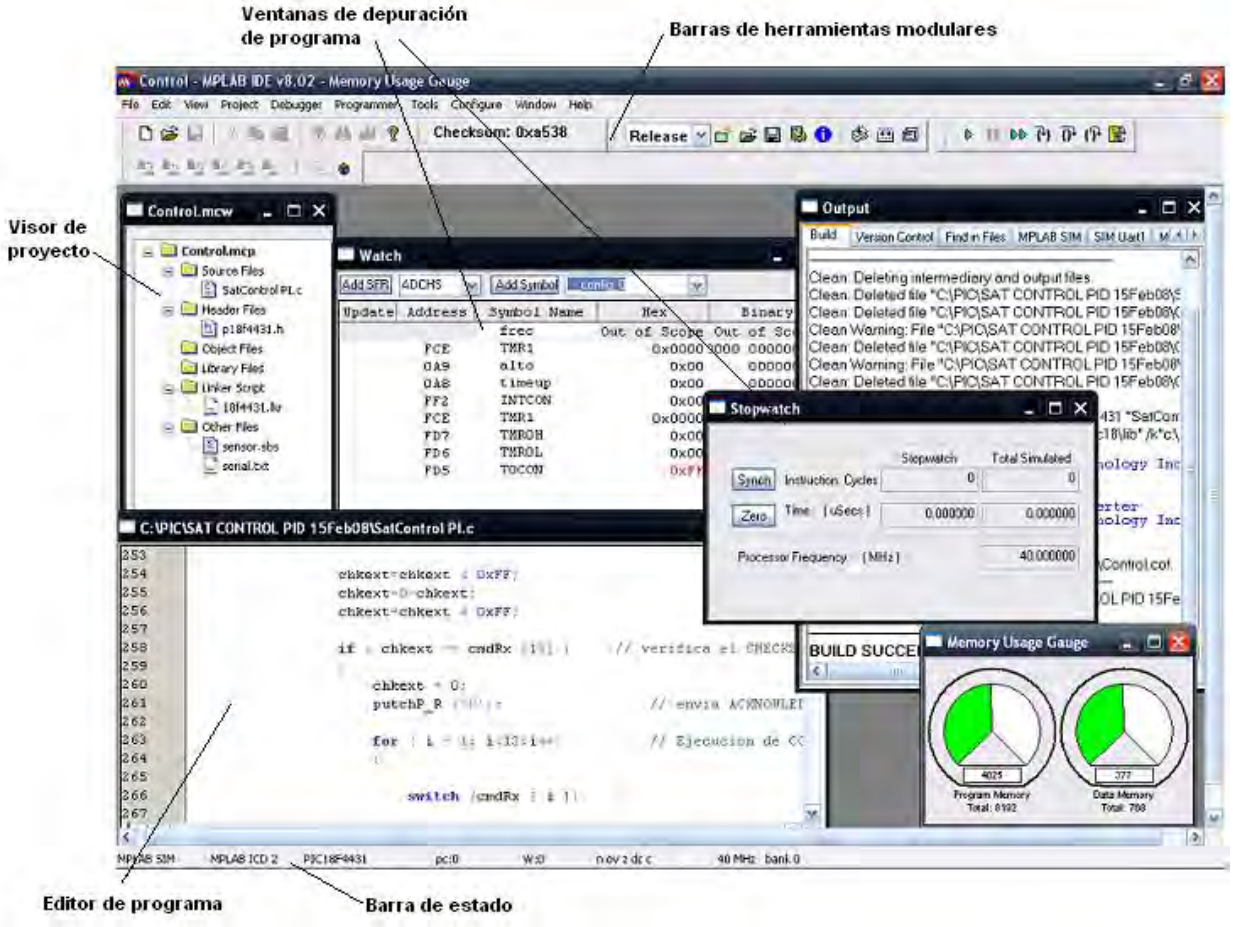


Figura 6.1 Entorno de programación en MPLAB

6.2.1.1 Sistemas embebidos

Un sistema embebido es básicamente un diseño electrónico que emplea un pequeño microcontrolador como los PIC de Microchip. Estos microcontroladores combinan una unidad de procesamiento con algunos periféricos adicionales, además de algunos circuitos adicionales para crear un pequeño módulo de control que requiere pocos dispositivos externos. Este dispositivo puede entonces ser embebido en otros dispositivos electrónicos y mecánicos para tener un control digital de bajo costo. Frecuentemente, un sistema embebido es una parte invisible o un submódulo de un producto, como por ejemplo en teléfono celular, o como en SIMSAT donde constituyen una opción muy buena y económica para desempeñar las tareas que requieran sus subsistemas; en este caso estabilización y sensores. Los diseños embebidos usan microcontroladores de bajo costo para agregar inteligencia a un sinfín de aplicaciones.

6.2.1.2 Componentes de MPLAB IDE

MPLAB IDE tiene tanto componentes instalados como aplicaciones extras que pueden ser integradas en el entorno de desarrollo de MPLAB para aplicaciones de microcontroladores sobre diferentes plataformas de programación.

Los componentes instalados en MPLAB IDE son:

- ***Manejador de proyectos:*** provee integración y comunicación entre el ambiente de desarrollo integrado y las herramientas del lenguaje de programación. Es un organizador de archivos a ser editados y otros archivos asociados, de modo que puedan enviarse a las herramientas de lenguaje de programación para ensamblarlos o compilarlos.

- ***Editor:*** es un cuadro de texto que ofrece señalizaciones adecuadas del correspondiente lenguaje de programación, que además sirve como ventana señalizadora en el depurador de código.

- ***Ensamblador/ligador y herramientas de lenguaje:*** el ensamblador puede ser usado individualmente para ensamblar un solo archivo, o puede ser usado con un ligador para construir un proyecto desde archivos fuente separados, bibliotecas y objetos recompilados. El ligador es responsable de posicionar el código compilado dentro de las áreas de memoria del microcontrolador elegido para el proyecto.

- ***Depurador:*** permite definir “breakpoints”, ejecución paso a paso, ventanas de monitoreo y todas las características de un depurador moderno para el MPLAB IDE. Este trabaja en conjunto con el editor para referenciar información desde el objeto en depuración hasta el código fuente.

- ***Motores de ejecución:*** son simuladores de software en MPLAB IDE para todos los microcontroladores PIC y dsPIC. Estos simuladores usan la PC para simular las instrucciones y algunos periféricos del microcontrolador. También se encuentran emuladores y depuradores en circuito opcionales para la verificación del código en el hardware.

Existen también componentes extra que pueden comprarse y adicionarse al entorno de MPLAB como:

- Compiladores de lenguaje de programación, como CCS Compiler, HI-TECH, o IAR que son capaces de compilar en lenguajes diferentes al de ensamblador como C o Basic.

- Programadores, como MPLAB ICD2 o PICSTART Plus que programan una gran cantidad de microcontroladores PIC de Microchip.

- Emuladores en circuito como MPLAB ICE 2000, que permite el control total sobre un microcontrolador emulado directamente sobre la aplicación física.

- Depuradores en circuito, similar al emulador pero es una alternativa más económica.

Figura 6.2 Herramientas de programación de Microchip

6.2.2 MPLAB C18

MPLAB C18 es un complemento externo de MPLAB, es una aplicación que corre en Windows compatible con el entorno MPLAB IDE. Es también un compilador de lenguaje ANSI C con optimización para microcontroladores de la familia PIC18 de Microchip. El compilador se desvía del estándar ANSI X3.159-1989 solo donde el estándar entra en conflicto con la eficiencia en código del microcontrolador. Este lenguaje puede depurarse en conjunto con las herramientas de desarrollo de Microchip como son el programador-depurador MPLAB ICD2 y el simulador MPLAB SIM de Microchip.

El compilador MPLAB MC18 tiene las siguientes características:

- Compatibilidad con el estándar ANSI 89.
- Integración con el entorno MPLAB IDE para su fácil manejo en la creación y depuración de proyectos de software.
- Generación de módulos de objetos reubicables para un reuso de código mejorado.
- Compatibilidad con módulos de objetos generados por el ensamblador MPASM, permitiendo completa libertad al mezclar código en lenguaje C y ensamblador, en un solo proyecto.
- Acceso transparente a la lectura y escritura de memoria externa.
- Fuerte apoyo para el ensamble en línea cuando se requiere un control total del proyecto.
- Generador de eficiencia de código con optimización en niveles múltiples.

- Extensas bibliotecas, incluyendo manipulación de periféricos como generadores de PWM, módulos I^2C, puerto serial, puerto SPI, manipulación de cadenas y bibliotecas matemáticas.
- Control completo del usuario sobre la localización de datos y código.

6.2.3 Visual Basic 6

Visual Basic 6 constituye un IDE (Ambiente de Desarrollo Integrado o en inglés Integrated Development Enviroment) que ha sido empaquetado como un programa de aplicación, es decir, consiste de un editor de código (programa donde se escribe el código fuente), un depurador (programa que corrige errores en el código fuente para que pueda compilarse correctamente), un compilador (que traduce el código fuente a lenguaje de máquina) y un constructor de interfaz gráfica o GUI (forma de programación visual que no requiere la escritura de código para la parte gráfica del programa).

Es también un lenguaje de fácil aprendizaje pensado tanto para programadores principiantes como expertos, guiado por eventos y centrado en un motor de formularios que facilita el rápido desarrollo de aplicaciones gráficas. Su sintaxis, derivada del antiguo BASIC, ha sido ampliada con el tiempo al agregarse las características típicas de los lenguajes estructurados modernos. No requiere de manejo de punteros y posee un manejo muy sencillo de cadenas de caracteres. Posee varias bibliotecas para manejo de bases de datos y se puede conectar con cualquier base de datos.

Visual Basic 6 se utiliza principalmente para aplicaciones de gestión de empresas, debido a la rapidez con la que se puede desarrollar un programa que utilice una base de datos sencilla, además de la abundancia de programadores que existe para este lenguaje.

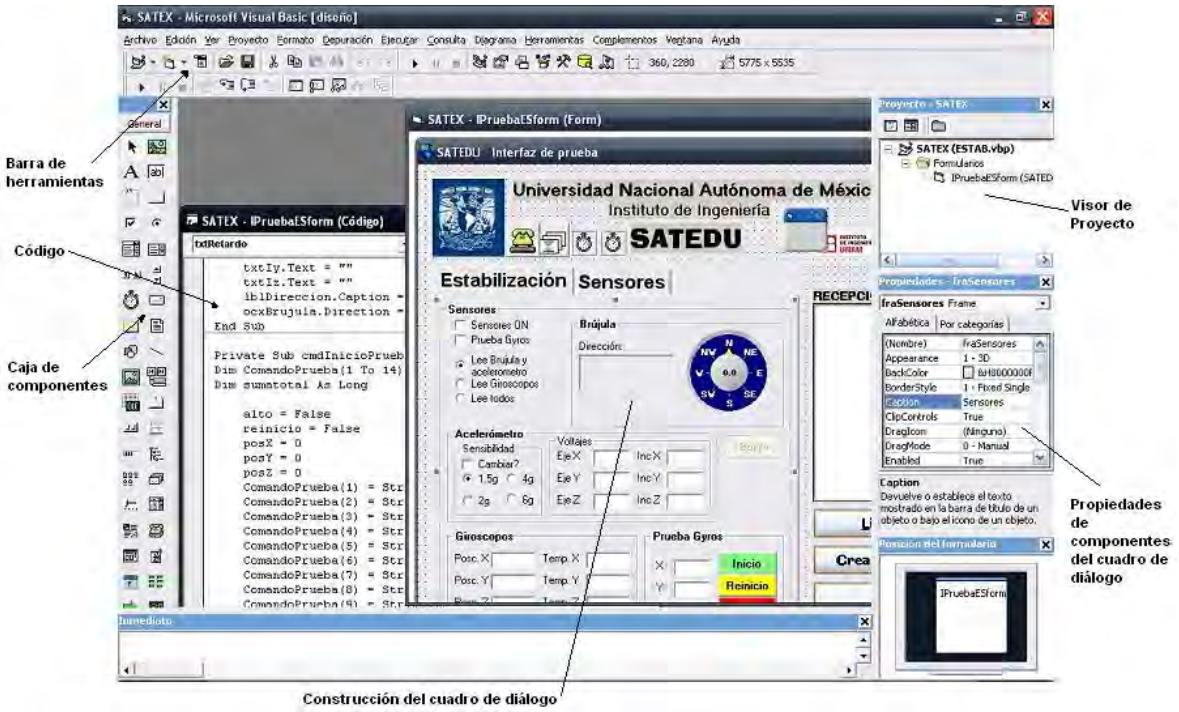


Figura 6.3 Entorno de Visual Basic 6

6.3 Programación de los microcontroladores en los subsistemas de estabilización activa y sensores

La programación, como se había mencionado ya, es esencial para el buen desempeño del SIMSAT en sus tareas, y más aun debido a que éste se construyó como un sistema de inteligencia distribuida donde las decisiones principales las toma la CV, a través de un programa principal.

6.3.1 Sistemas distribuidos

Los sistemas distribuidos tienen su origen en los sistemas de computo, donde varios equipos se conectaban entre si, coordinando sus acciones a través de mensajes para el logro de un objetivo común y empleando un esquema de cliente-servidor. Debido a la evolución de la tecnología en el área de desarrollo de circuitos de alta escala de integración (como los microcontroladores) se ha hecho factible la creación de sistemas distribuidos de menor tamaño. Para ello la industria actual emplea esquemas de operación similares a los precedentes pero usando ahora procesadores independientes (microcontroladores, FPGAs[1] o DSPs[2]) en lugar de PCs. Tal es el caso de la empresa *National Instruments* que se dedica a la creación de esta clase de hardware para la automatización industrial. En la figura 6.4 se muestra el sistema distribuido implantado en SIMSAT.

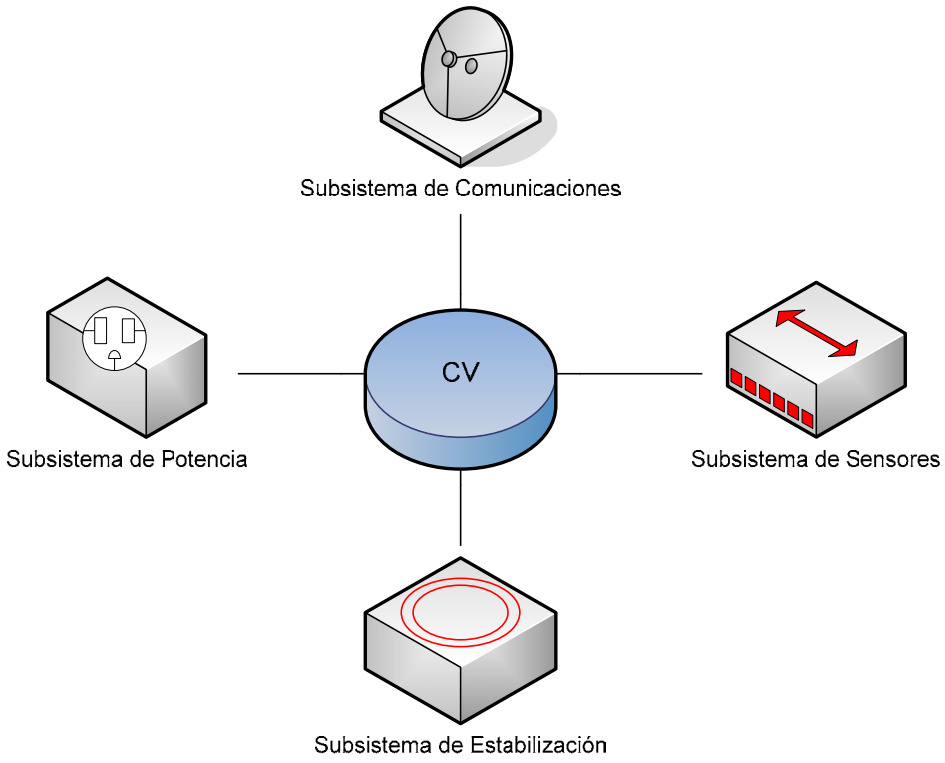


Figura 6.4 Sistema distribuido implantado en SIMSAT

[1] Field Programmable Gate Array (campo de arreglo de compuertas programables), son dispositivos con centenas o centenas de miles de bloques de lógica cuya interconexión es programable para lograr desde funciones de lógica sencillas, como el de una compuerta hasta las muy complejas como en un procesador.

[2] Digital Signal Processor, es un procesador optimizado para aplicaciones numéricas que requieran gran velocidad.

Al igual que en un sistema de computo distribuido, en SIMSAT la CV es el servidor mientras los demás subsistemas se comportan como clientes. Cada subsistema tiene su propio procesador que realiza tareas independientes para liberar la carga de procesamiento al procesador de la CV. De otra forma la CV tendría que atender a cada tarea y adicionalmente su programación sería más extensa. Estas son las principales ventajas de los sistemas distribuidos. Ahora bien, los microcontroladores, como se señaló anteriormente, deben llevar una cierta programación para desempeñar sus tareas asignadas, por lo que el nuevo reto de un sistema distribuido se vuelve su programación, debido a que estos sistemas suelen tener microprocesadores de arquitecturas distintas, lo cual hace que la programación total del sistema no necesariamente se desarrolle con un solo lenguaje o estilo de programación.

En SIMSAT el aspecto de la programación trato de cuidarse haciendo que la mayoría de los microcontroladores pertenecieran a la misma familia y que las herramientas de programación fueran iguales, para que el desarrollo de la programación de la mayoría del sistema fuera uniforme. Para ello, la programación de todo el SIMSAT se basa en C, con la diferencia que el estilo de programación en C del procesador de la CV (SAB80C166), el PIC16 de la CV y los PIC18 de los subsistemas son un poco diferentes entre sí, además de que se compilan con herramientas distintas.

6.3.2 Esquemas generales de ejecución de comandos en las tarjetas de estabilización activa y sensores

En vista de que los microcontroladores usados en los subsistemas de estabilización y sensores pertenecen a la misma familia (PIC18) y a que se usa un compilador común (MPLAB C18) fue posible realizar un esquema general sobre la forma en que tendrían que recibir los comandos, según lo requirieran las funciones que realizarán los subsistemas.

En la figura 6.5 se muestra el esquema general de recepción y ejecución de comandos de las tarjetas de estabilización activa y sensores. En ambas tarjetas el diagrama de flujo general es el mismo y la programación es casi idéntica, solo cambian los archivos de cabecera y la forma en que se inicializa cada microcontrolador. Esta es la ventaja de programar bajo los mismos lenguajes de programación y con las mismas herramientas. Un punto a recalcar dentro de este esquema de programación es el ahorro de energía que fue considerado, pues como se ve al final del diagrama de flujo cada vez que se termina de procesar la línea de comandos se envía a dormir al CPU. En el caso de estabilización solo se envía a dormir al CPU en tanto que sus periféricos quedan energizados debido a que habrá veces que se necesitará que la rueda inercial quede girando (para disminuir perturbaciones en el SIMSAT durante algún experimento).

En la figura 6.6, está representada la interrupción principal que se genera en ambos subsistemas a partir de la recepción de nuevos datos por el canal serial de comunicaciones. Esta rutina se da o se puede dar mientras el microcontrolador se encuentra durmiendo, y en ella se reciben los 14 bytes que componen una trama de comunicaciones. Para que la trama quede almacenada se verifica que el encabezado sea el indicado (está predefinido), posteriormente el microcontrolador ejecuta el comando, figura 6.5.

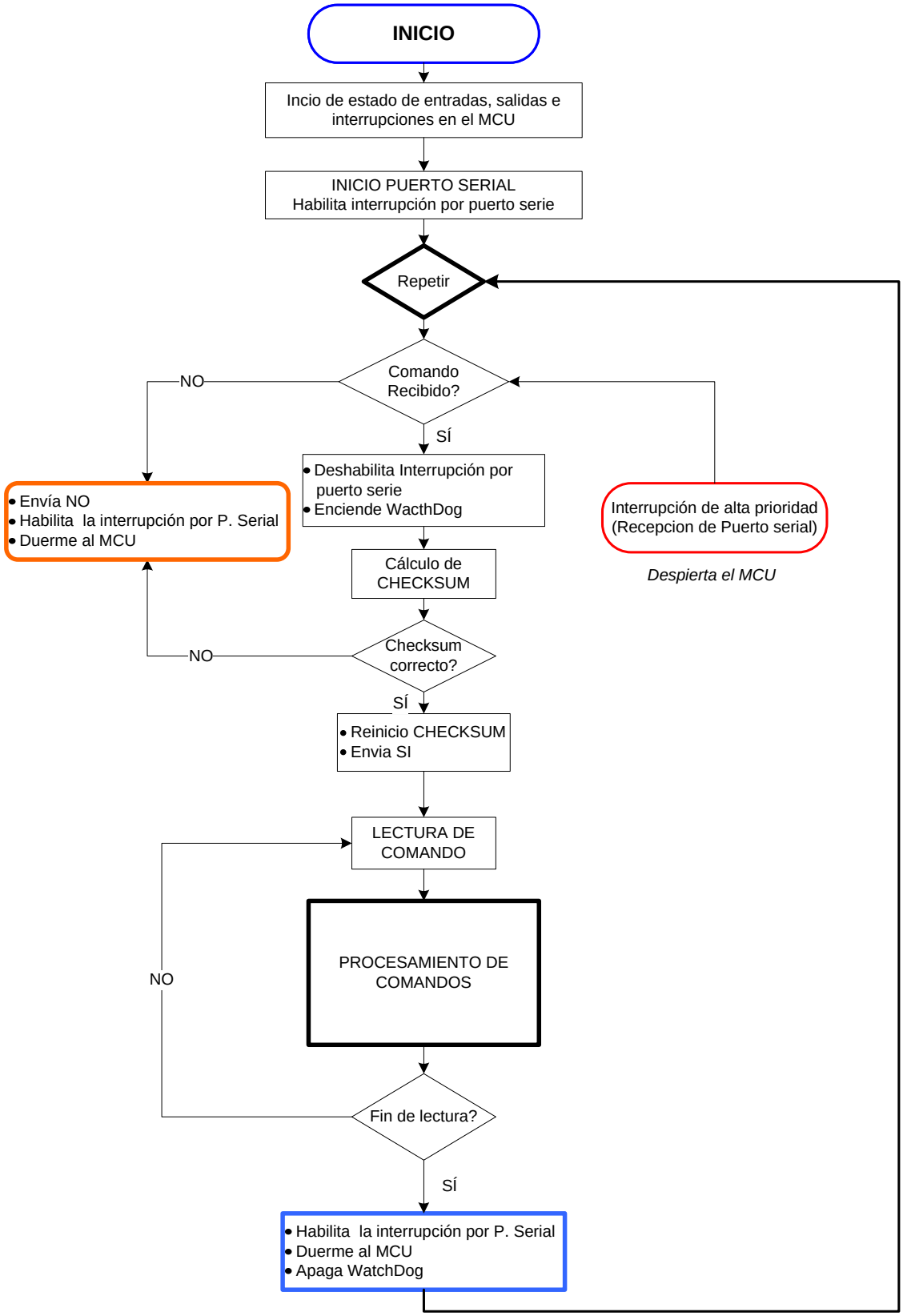


Figura 6.5 Esquema general de recepción y evaluación de comandos

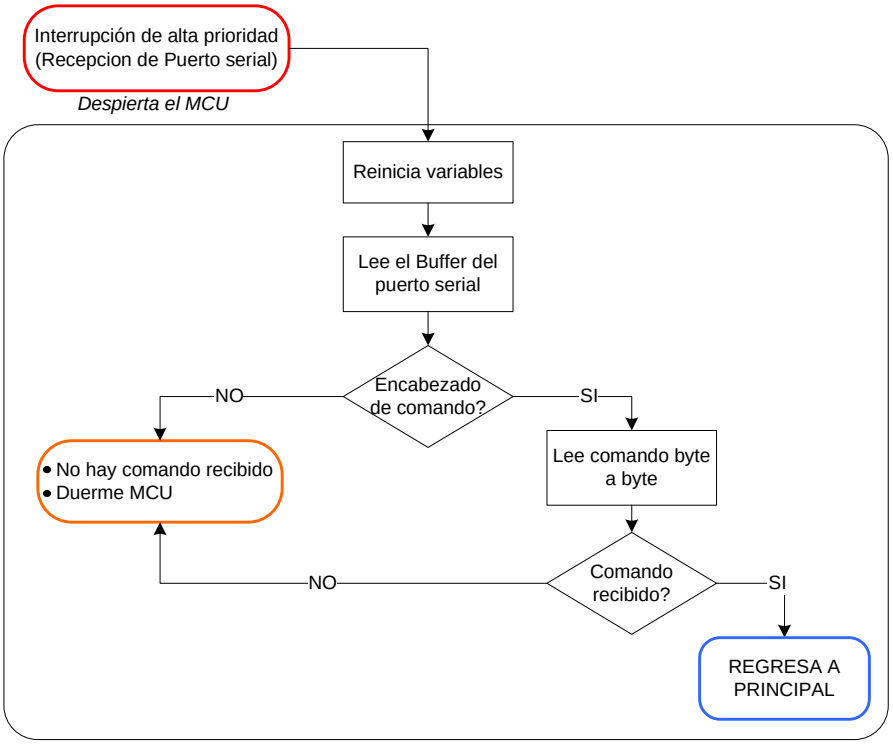


Figura 6.6 Esquema general de interrupción por recepción en el puerto

6.4 Comandos creados para la interacción entre la computadora de vuelo y las tarjetas de estabilización activa y sensores

Antes de empezar a programar los microcontroladores se determinaron las acciones que desempeñarían las tarjetas al recibir las tramas de comandos de la CV. Los comandos definidos para la tarjeta de estabilización activa fueron:

- Simulación de ondas: senoidal, triangular y diente de sierra.
- Fijar PWM para 6 BTMs en sus tres ejes.
- Apagado de bobinas de manera individual y total.
- Fijar PWM del motor de la rueda inercial y definir sentido de giro.
- Llevar el motor de rueda inercial a cierta velocidad.
- Cambiar la velocidad del motor de rueda inercial sin alterar su sentido.
- Apagar el motor de la rueda inercial.
- Frenar el motor de la rueda inercial.
- Enviar de telemetría del motor de la rueda inercial.

Los comandos definidos para la tarjeta de sensores fueron:

- Lectura de brújula y acelerómetro en tres ejes.
- Lectura de giróscopos.
- Lectura de todos los sensores.
- Cambio de sensibilidad del acelerómetro en tres ejes.

- Generación del flujo de datos (stream) para la acción de seguimiento en tiempo real con software de realidad virtual.

6.4.1 Estructura de los comandos de los subsistemas

Las tramas de comandos empleadas para ordenar las acciones requeridas en cada subsistema tienen la misma longitud, 14 bytes cada una. En tanto que el camino que siguen así como el contenido de las tramas que llegan a los subsistemas se indica en la figura 6.7.

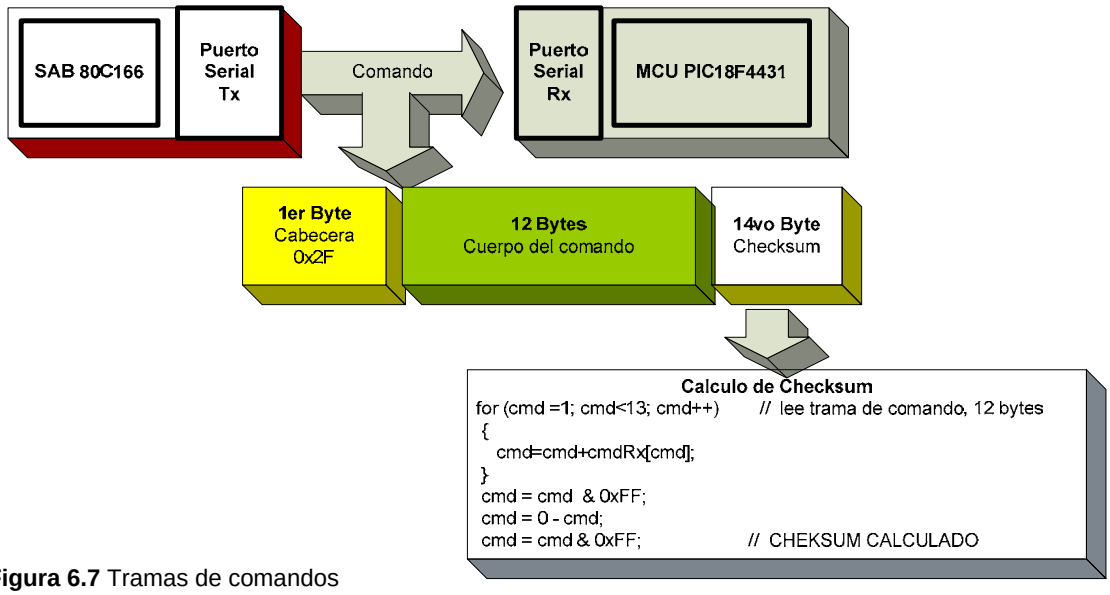


Figura 6.7 Tramas de comandos

Como se puede observar en la figura 6.7, el primer byte es un 0x2F que fue elegido como el byte que marca la diferencia entre cada trama de comunicaciones recibida o enviada por los subsistemas. Los 12 bytes complementarios son el cuerpo del comando que son bytes que definen los argumentos asociados a la función del comando y que se asocian a la acción procesada y ejecutada en cada subsistema. El último byte es el *checksum* o suma de verificación, que es una forma de control de redundancia de los datos y sirve para verificar la integridad de los datos recibidos en el comando (que no lleguen corrompidos). El *checksum* lo calcula el emisor del comando, en este caso la CV, después de llenar la trama de comunicaciones de 14 bytes; al llegar al receptor se realiza la misma operación y se comparan para verificar la integridad del paquete de datos. Esta suma de verificación o *checksum* consiste en sumar los 12 bytes que componen el cuerpo del comando, que son los importantes, despreciando la cabecera del comando, adicionalmente se realiza un complemento a 2 de esta suma y se trunca a 8 bits.

6.4.2 Forma de envío de un comando desde un subsistema a la CV

Fue necesario planear un procedimiento para crear y enviar un comando desde los subsistemas que lo requirieran hasta la CV, que se usan a manera de respuesta a comandos enviados por la CV. Estas respuestas pueden ser requeridas por la estación terrena (ET) para conocer el estado total del sistema. Para ello se creó la rutina mostrada en el diagrama de flujo de la figura 6.8.

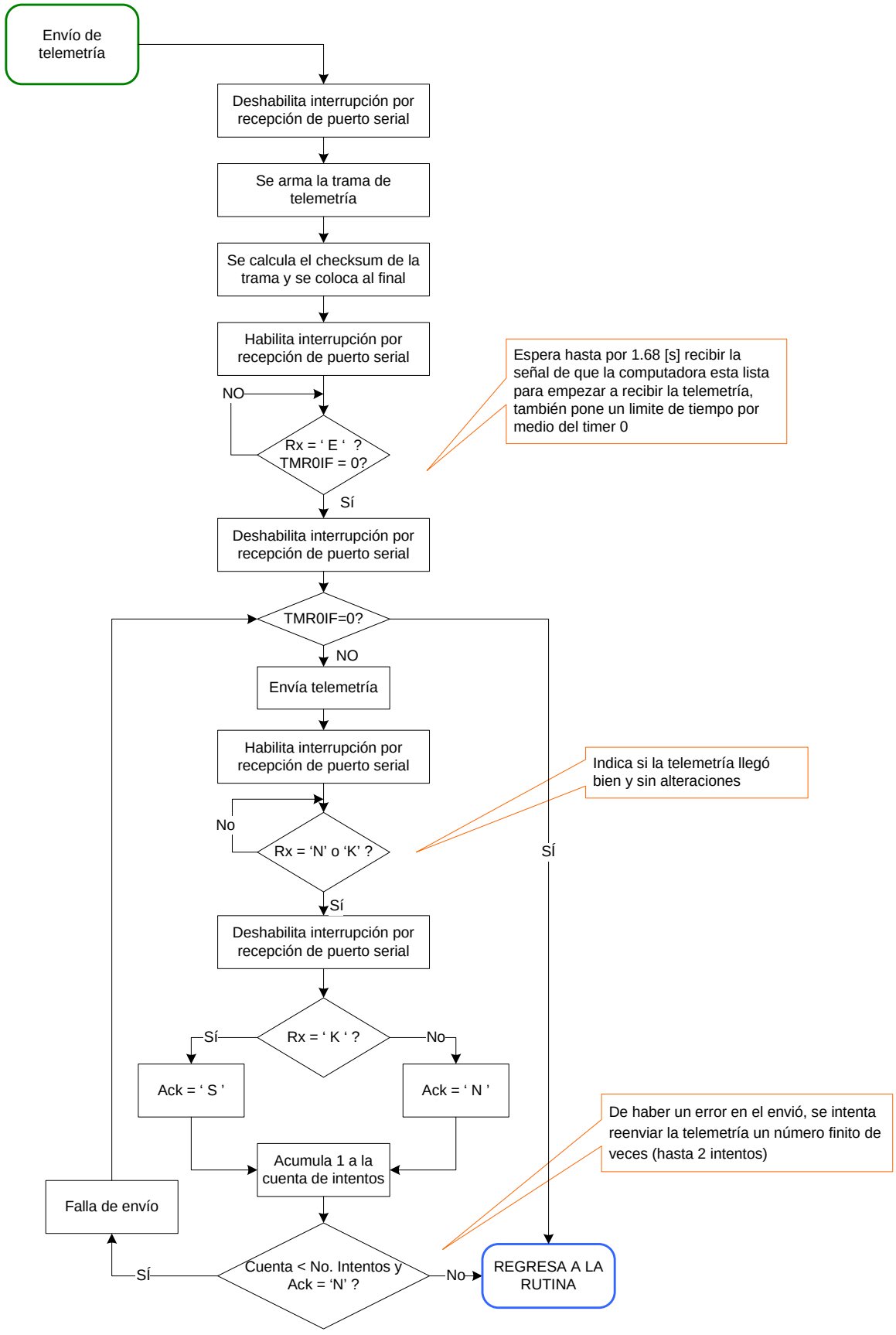


Figura 6.8 Diagrama de flujo para el envío de telemetría

Como se puede ver en la figura anterior, se hacen varias consideraciones para evitar que los algoritmos se congelen y evitar el reset, entre ellas el tiempo máximo de espera para recibir un byte por el puerto serial del microcontrolador, así como el reenvío de la telemetría un número determinado de veces.

6.4.3 Comandos para la tarjeta de estabilización activa

Como una forma de estructurar los comandos de la tarjeta de estabilización activa, estos se dividieron en 3 categorías:

1. Comandos de emulación de señales en lazo abierto en el motor de DC.
2. Comandos de BTMs.
3. Comandos del motor y la rueda inercial.

6.4.3.1 Comandos de emulación de señales en lazo abierto en el motor de CD

Para los comandos de emulación de señales en lazo abierto, se asignaron 3 señales: senoidal, triangular y diente de sierra, emuladas todas con el motor en lazo abierto.

Señal senoidal: para emular esta señal en el motor, se cargó previamente una tabla de valores en el microcontrolador que corresponden sólo a medio período. Estos valores se adaptaron a los registros que corresponden al ciclo de trabajo que se asigna al motor, debido a que se puede variar con una resolución de 14 bits. De acuerdo con la configuración realizada, los valores variaron de entre 0 y 16383, de forma senoidal en un total de 200 puntos asignados.

El comando de la señal senoidal está compuesto de la siguiente forma:

1er byte	2do byte
A0	XX
Cabecera de identificación de la señal Seno	Va de 0 a 0xFF. Define el retardo entre la carga de punto a punto de la señal lo que permite variar en un rango la frecuencia de la señal.

Señal triangular y diente de sierra: para emular estas dos señales se realizó un proceso un poco distinto, los puntos de las señales se generaban al momento debido a que eran lineales y es fácil generar cada punto al instante solo con sumas. Para emular la señal estos puntos se cargaron al registro que corresponde al ciclo de trabajo del generador PWM conectado al motor.

Los comandos de las señales están compuestos de manera similar a la señal Senoidal, de la siguiente forma:

1er byte	2do byte
A1	XX
A2	XX
Cabecera de identificación de la señal triangular y diente de sierra respectivamente.	Va de 0 a 0xFF. Define el retardo entre la carga de punto a punto de la señal, lo que permite variar en un rango la frecuencia de la señal

6.4.3.2 Comandos de BTMs

La segunda sección de comandos está relacionado con el control de las BTMs, regulando, su encendido, apagado y el sentido e intensidad de la corriente que pasa por ellas. Las BTMs dobles pueden manejar diferentes intensidades de campo magnético debido a los 2 arrollamientos diferentes que la componen.

Los comandos que controlan a las BTMs permiten: regular su encendido, la intensidad y el sentido del campo que generan. El diagrama de flujo de carga de estos es común a las 6 funciones siguientes, que se ven en la figura 6.9.

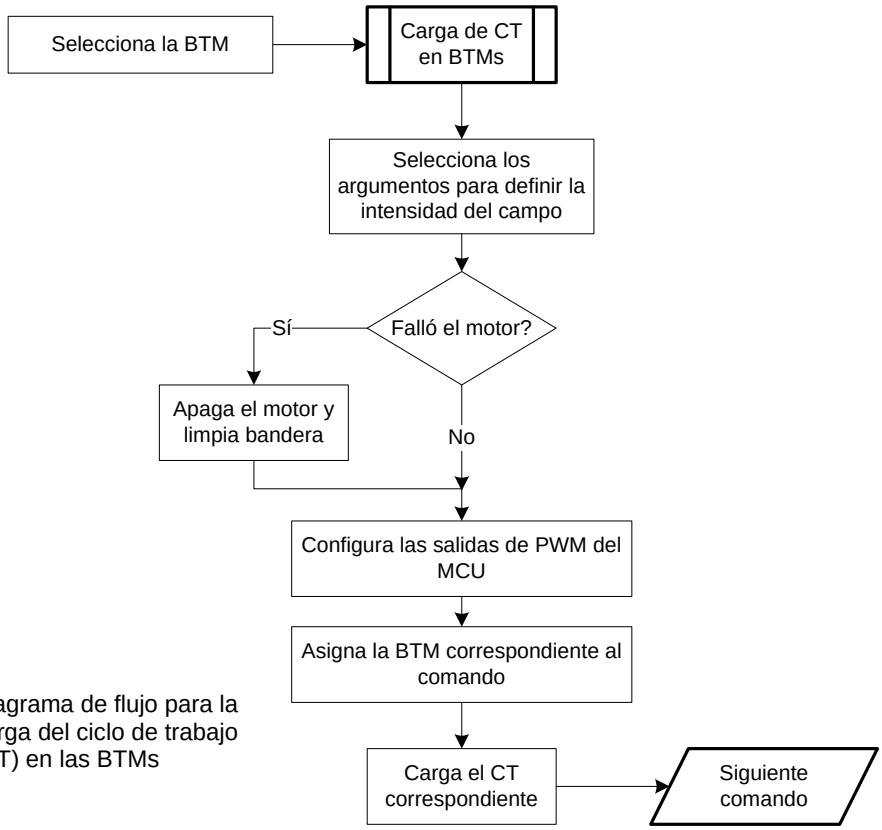


Figura 6.9 Diagrama de flujo para la carga del ciclo de trabajo (CT) en las BTMs

La estructura del comando que asigna el ciclo de trabajo a las BTMs es:

1er byte	2do byte	3er byte
B0 (eje X bobina pequeña)	XX	XX
B1 (eje X bobina grande)	XX	XX
B2 (eje Y bobina pequeña)	XX	XX
B3 (eje Y bobina grande)	XX	XX
B4 (eje Z bobina pequeña)	XX	XX
B5 (eje Z bobina grande)	XX	XX
Cabecera de identificación	Definen el CT de la BTM	

El segundo y tercer byte contienen los siguientes parámetros:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2	Determinan el sentido de la corriente		Determinan el CT de la señal de PWM que va a la bobina a través de los puentes H					
3	Determinan el CT de la señal de PWM que va a la bobina a través de los puentes H							

6.4.3.3 Comandos del conjunto del motor y rueda inercial

Los comandos que se creyeron pertinentes para el control del conjunto del motor-rueda se dan en seguida. El primero de ellos es similar al de las BTMs solo hay que fijar un ciclo de trabajo y un sentido de circulación de corriente al puente H que controla al motor; el objetivo es que éste se pueda mover a una cierta velocidad. El comando que se envía es tiene la siguiente forma:

1er byte	2do byte	3er byte
C0	XX	XX
Cabecera de identificación	Definen el Ciclo de Trabajo del motor	

El segundo y tercer byte están constituidos de la siguiente manera:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2	Determinan el sentido de la corriente		Determinan el CT de la señal de PWM que va al motor					
3	Determinan el CT de la señal de PWM que va a la bobina a través de los puentes H							

En el siguiente comando se implanta una rutina de control de la velocidad de giro de la rueda inercial de forma retroalimentada, se implementó un control PI que se analizó en el capítulo 2 y en el cual fueron definidos sus parámetros. El diagrama de flujo que siguió la rutina se muestra en la figura 6.10.

El comando que se envía tiene la siguiente forma:

1er byte	2do byte	3er byte
C1	XX	XX
Cabecera de identificación	Definen la velocidad y sentido de giro del motor	

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2	Determina el sentido de la corriente	Determinan la velocidad en RPMs que debe de alcanzar la rueda y el motor						
3	Determinan la velocidad en RPMs que debe de alcanzar la rueda y el motor							

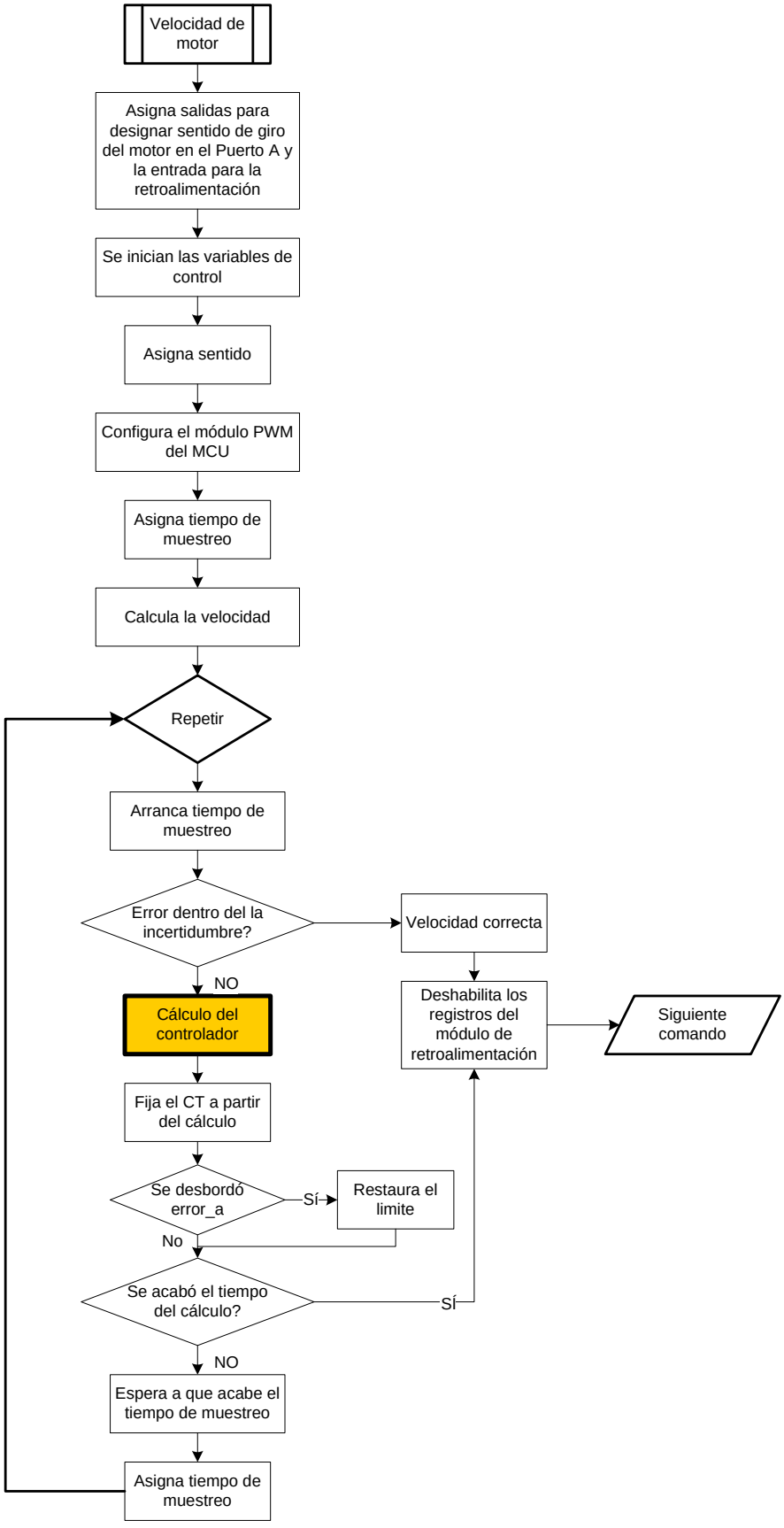


Figura 6.10 Diagrama de flujo para la carga de nueva velocidad en la rueda

La forma en que es implementado un algoritmo de control dentro de un microcontrolador no es única. En el capítulo 2, la ecuación que conforma el controlador de la planta, en este caso el conjunto del motor y la rueda inercial, no es una ecuación que pueda ser colocada de manera directa dentro del microcontrolador, tiene que ser una rutina capaz de emular a la ecuación durante un cierto tiempo hasta que la velocidad deseada se alcance.

Recapitulando un poco la representación del controlador PI es mediante la ecuación:

$$C(t) = Kp \cdot E(t) + Ki \cdot \int_0^t E(t)dt$$

Donde:

C(t) = respuesta del controlador
E(t) = error
Kp = constante proporcional
Ki = constante integral

Esta ecuación está representada como puede observarse en función del tiempo, pero para poder manejarla dentro de una rutina en un microcontrolador se debe de transformar a una ecuación en diferencias, y aunque tal vez no sea posible generar un resultado exacto, al menos este será muy aproximado al deseado, la ecuación queda como:

$$c(n) = Kp \cdot e(n) + Ki \cdot \sum_0^N e(n)$$

Donde:

n = número de muestra
c(n) = respuesta del controlador
e(n) = error

A partir de la ecuación en diferencias es posible generar un diagrama de flujo, mostrado en la figura 6.11.

Ya que se tiene el diagrama de flujo, en primer lugar se tiene al término proporcional que es el más simple, donde la ganancia kp es multiplicada por el error obtenido. La suma de la corrección aplicada al sistema es directamente proporcional al error y conforme la ganancia kp se aumente, la corrección sometida al sistema se vuelve cada vez más agresiva. Este tipo de controlador es común usarlo para llevar al error a un valor pequeño pero diferente de cero, dejando un error de estado estacionario. Para su implementación solamente hay que multiplicar el error obtenido por la ganancia kp especificada, esto se logra en una rutina de multiplicación de 16 x 16 bits y el resultado se almacena en una variable de 24 bits ya que limitamos la variable kp a un término de máximo 8 bits comprendido en valores entre 0 y 15, y el error es una variable de 16 bits comprendido en un valor entre 0 y 65535.

En cuanto a la parte del control integral solo le interesan los errores pasados, a diferencia del control proporcional al que solo le interesaba el error actual. Por lo tanto es necesario sumar el conjunto de errores que se hayan presentado al ejecutarse el controlador de modo que al final de cada tiempo de muestreo un nuevo valor de error se sume (en este caso en la variable error_a).

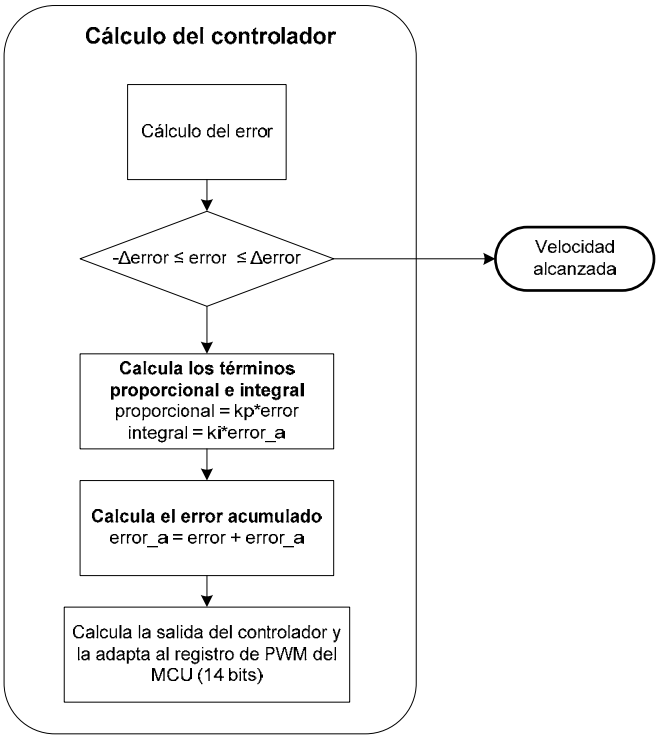


Figura 6.11 Diagrama de flujo del cálculo del controlador

Una consideración a tener dentro del sistema es respecto al tiempo de muestreo, y es que dependiendo del tipo de sistema y su capacidad de respuesta es que es posible asignar el adecuado. Si el sistema tiene un tiempo de respuesta muy lento y el tiempo de muestreo es muy rápido como para darle oportunidad de responder el error se acumulará dentro de una variable, y debido a que ésta tiene una capacidad limitada dentro del microcontrolador ya que le fue asignada sólo cierta capacidad de almacenamiento, al final terminará desbordándose provocando fallas en el control y que la planta no se estabilice. Otro factor que hay que tener en cuenta para el controlador de tipo integral es llamado '*wind-up',* ocurre cuando la acumulación del error se mantiene en incremento debido a que la salida de la planta está saturada. Este evento puede ser corregido estableciendo un límite a la variable del error acumulado o bien simplemente dejando de ejecutar la acción integral del controlador sobre la planta cuando está saturada. En este caso se decidió poner un límite al error acumulado.

El siguiente comando especifica cambio de velocidad en la rueda inercial, este comando es básicamente el mismo que el del algoritmo de velocidad, lo que varía son las variables del controlador PI que ahora son restauradas. Por lo que ahora el cambio se da de una manera más suave y no tiene que empezar de cero para alcanzar la nueva velocidad requerida. Hay que puntualizar que el cambio de velocidad solo se da en una misma dirección, por lo que el bit de sentido no se considera. El comando está constituido de la misma forma que el anterior solo que ahora no se toma en cuenta el sentido, como se muestra enseguida:

1er byte	2do byte	3er byte
C2	XX	XX
Cabecera de identificación	Definen la nueva velocidad del motor	

Los siguientes 2 comandos tienen como objetivo ya sea frenar o apagar al motor según se desee, estos comandos no necesitan de datos adicionales, la representación que tienen es la siguiente:

1er byte	Función
C3	Apaga el motor
C4	Frena el motor

El último comando que también está compuesto de un byte envía una solicitud de envío de la corriente que circula por el motor. Debido a que no hay lectores de corriente en el microcontrolador lo que se lee es un voltaje proporcional a la corriente y éste se interpreta ya sea por la CV o por el software de ET. El byte que se envía y la estructura del comando es como sigue:

1er byte	Función
C5	Solicitud de lectura de corriente

La trama enviada está conformada de la siguiente manera:

No de Byte	Byte	Función
1	2F	Cabecera de comando
2	XX	Los 2 bits mas significativos del convertidor A/D
3	XX	Los 8 bits restantes del convertidor A/D
4	41	Dato de relleno
5	41	Dato de relleno
6	41	Dato de relleno
7	41	Dato de relleno
9	41	Dato de relleno
10	41	Dato de relleno
11	41	Dato de relleno
12	41	Dato de relleno
13	41	Dato de relleno
14	XX	Suma de verificación o “checksum”

6.4.4 Comandos para la tarjeta de sensores

En el caso de los comandos de la tarjeta de sensores, básicamente comprende el regreso de datos de sensores del subsistema hacia la CV. Para este subsistema en especial se usaron 3 protocolos de comunicaciones diferentes, los cuales se dan en seguida.

Comunicación serial (RS232): La EIA (Electronics Industry Association) elaboró la norma RS-232 que define la interfaz mecánica, los pines, las señales y los protocolos que debe cumplir la comunicación serial.

El RS232 es un protocolo de comunicación serie orientado a caracteres, es decir, un protocolo donde toda la información es enviada por un solo canal bit a bit (un canal

para enviar información y otro para recibirla), y donde lo que se envían son caracteres. Por ejemplo, si se desea enviar el número 123, primero se tendrá que enviar el carácter 1, seguidamente el 2 y para finalizar el 3, y no el byte que represente el número 123.

Este protocolo está diseñado para distancias cortas, de unos 15 metros, y se puede trabajar de forma asíncrona o síncrona y con tipos de canal simplex, halfduplex y fullduplex. En SIMSAT se trabajó de la forma asíncrona y halfduplex.

Los conectores más utilizados para una conexión RS232 son los DB9 y los DB25. En la tabla 6.1 se pueden observar las señales más comunes en RS232 según los pines del conector asignados:

- GND: Valor a 0V.
- TD: Línea de datos del transmisor al receptor.
- RD: Línea de datos del receptor al transmisor.
- DTR: Línea por donde el receptor informa al transmisor que está vivo y bien.
- DSR: Línea por donde el transmisor informa al receptor que está vivo y bien.
- RTS: Línea en la que el transmisor indica que quiere enviar algo al receptor.
- CTS: Línea en la que se informa que el receptor está preparado para recibir datos.
- DCD: Línea por la que el receptor informa al transmisor que tiene una portadora entrante.
- RI: Línea en la que se indica que se ha detectado una portadora.

Señal	DB-25	DB-9
GND	7	5
Transmisión de datos (TD)	2	3
Recepción de datos (RD)	3	2
Terminal de datos preparado (DTR)	20	4
Datos preparados (DSR)	6	6
Petición de envió (RTS)	4	7
Limpieza de envió (CTS)	5	8
Portadora de datos detectada (DCD)	8	1
Indicador de tono (RI)	22	9

Tabla 6.1 Señales mas comunes en RS232 y sus respectivos pines en los conectores.

La conexión más sencilla se puede realizar con 3 cables (TD, RD, y GND). En el presente trabajo se utilizó esta configuración de 3 cables.

Los parámetros a configurar en una comunicación RS232 son los siguientes:

- ✓ **Protocolo serie (numero de bits - paridad - bits de parada):** La paridad se utiliza para comprobar la calidad de los datos recibidos. Los bits de datos pueden estar entre los 5 bits y los 8, y los bits de parada consisten en uno o dos bits puestos a ’1’. En el SIMSAT se utiliza la configuración 8, N, 1 (8 bits de datos sin paridad y con un bit de parada).

- ✓ **Velocidad del puerto:** RS232 puede transmitir los datos a unas velocidades determinadas (normalmente entre 4800 y 115200 bps). En SIMSAT se usa una velocidad entre tarjetas de **9600 bps**.

- ✓ **Protocolo de control de flujo:** El control de flujo puede ser mediante hardware gracias al llamado 'handshaking' entre las líneas RTS y CTS, o por software mediante el XON/XOFF. En SIMSAT no se usa control de flujo.

Comunicación SPI (Serial Peripherial Interface): El Bus SPI es un estándar de comunicaciones desarrollado por Motorola, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos es del tipo fullduplex, síncrono de tipo serial. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier electrónica digital que acepte un flujo de bits serie regulado por un reloj.

Este tipo de bus necesita de un dispositivo maestro y uno o más esclavos por lo que requiere de 4 líneas de conexión:

MISO (Master In / Slave Out): Esta línea envía los datos desde el esclavo hasta el maestro. Cuando el dispositivo con el que se desea la comunicación no está habilitado permanece en alta impedancia para evitar interferencias.

MOSI (Master Out / Slave In): Envía los datos del maestro al esclavo. El circuito maestro pone la línea MOSI medio ciclo antes del final del flanco de alta impedancia para evitar interferencias.

SCLK (Serial Clock): El reloj con el que los datos son sincronizados en las líneas anteriores.

CS (Chip Select): Los esclavos son seleccionados por el dispositivo maestro a través de esta línea. Para habilitar el dispositivo se tiene que poner en bajo esta línea. Solo puede haber un dispositivo habilitado.

Las ventajas de un bus serie son la reducción del número de conductores, pines y el tamaño del circuito integrado. Esto reduce el costo de fabricar, montar y probar la electrónica. Un bus de periféricos serie es la opción más flexible cuando están presentes muchos tipos diferentes de periféricos serie. Casi cualquier dispositivo digital puede ser controlado con la combinación de señales de SPI. Los dispositivos se diferencian en un número predecible de formas. Algunos leen los datos en el flanco de subida del reloj y otros en el flanco de bajada. La escritura se realiza casi siempre en la dirección opuesta de la dirección de movimiento del reloj. Algunos dispositivos tienen dos relojes. Uno para capturar o mostrar los datos y el otro para el dispositivo interno.

En el SIMSAT se utilizan tres dispositivos de esta naturaleza, los giróscopos, uno en cada eje coordenado. No es posible obtener la información de los 3 giróscopos al mismo tiempo por las condiciones que posee el bus SPI, por lo que hay que hacerlo uno a uno, dependiendo de la velocidad del dispositivo maestro es que es posible accesar al bus SPI con una velocidad máxima de hasta 10 Mbps en el caso del PIC debido a que se usa una velocidad de 40 MHz en su oscilador, pero para no usarlo a su velocidad máxima

se emplea una velocidad de **2.5 Mbps** que es suficiente; con ello, la lectura de los giróscopos se puede hacer de una manera muy rápida.

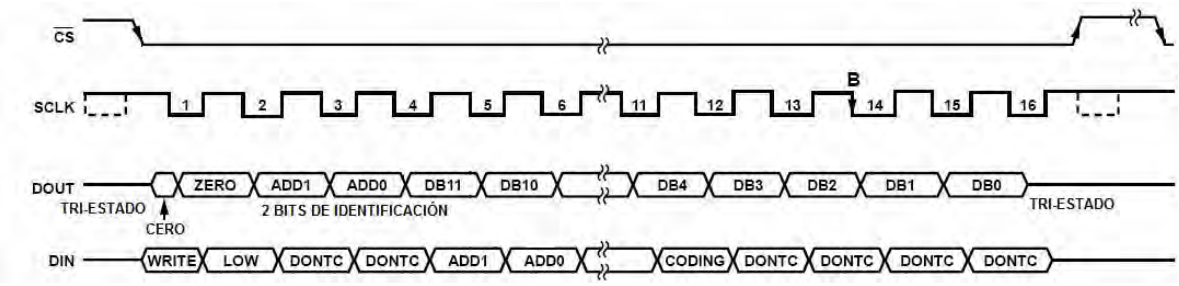


Figura 6.12 Diagrama de pulsos para el SPI del ADIS16100

Las comunicaciones entre el PIC y el ADIS, que se encuentran permanentemente a la escucha del registro de entrada de datos (DIN), se realizan a través de comandos de 16 bits. Una vez que el dispositivo ha sido seleccionado a través del CS, se inicia la comunicación y cuando se han recibido los 16 bits, el chip lee los datos y los procesa. En función del código recibido, realizará ciertas operaciones, en el caso del giróscopo estas operaciones pueden ser la lectura del giróscopo, su temperatura, o la lectura de alguno de los 2 canales convertidores A/D que tiene.

Para acceder a la lectura de los datos que puede medir el giróscopo, primero es necesario configurar el giróscopo a través de un primer comando que consiste en una trama de 12 bits, como se ve a continuación:

MSB (11)											LSB (0)
WRITE	LOW	DONTC	DONTC	ADD1	ADD0	HIGH	HIGH	DONTC	DONTC	LOW	CODING

No de Byte	Nemónico	Función
11	WRITE	Si esta en 1, escribe sobre el giro la configuración nueva
10	LOW	Debe estar en BAJO
9,8	DONTC	No importan
7,6	ADD1,ADD2	Modo de trabajo: 00: Lee el giróscopo 01: Lee temperatura 10: Convertidor A/D 1 11: Convertidor A/D 2
5,4	HIGH	Debe estar en ALTO
3,2	DONTC	No importa
1	LOW	Debe estar en BAJO
0	CODING	Formato de datos de salida 0: Complemento a dos 1: Offset binario

Como el PIC tiene un buffer de 8 bits es necesario configurar al giróscopo en dos pasos, mandando 16 bits, de modo que los últimos 4 tampoco importan. En el software del SIMSAT se usan 2 configuraciones para leer al giróscopo y a su temperatura. Posteriormente se envian los siguientes bytes 0x8300(giróscopo) y 0x8700(temperatura), ambas lecturas se toman en modo de complemento a dos.

Las lecturas tomadas son en palabras de 16 bits, y depende del formato elegido la forma en que tienen que ser descifradas, como se ve en la siguiente tabla:

	Representación en º/s	Representación decimal[1]	Patrón binario
Velocidad angular en complemento a dos	0.2439	1	0000**000000000001**
	0	0	0000**000000000000**
	-0.2439	4095	0000**111111111111**
Velocidad angular en representación binaria	0.2439	2049	0000**100000000001**
	0	2048	0000**100000000000**
	-0.2439	2047	0000**011111111111**
	[ºC]		
Temperatura en complemento a dos	25.1453	1	0001**000000000001**
	25	0	0001**000000000000**
	24.8547	4095	0001**111111111111**
Temperatura en representación binaria	25.1453	2049	0001**100000000001**
	25	2048	0001**100000000000**
	24.8547	2047	0001**011111111111**

Comunicación I^2C (Inter-Integrated Circuit): Durante los 80's, Philips (Koninklijke Philips Electronics N.V.) desarrolló el bus de dos alambres entre circuitos integrados (del inglés Inter-Integrated Circuit, I2C) para conectar de manera sencilla periféricos múltiples a una unidad de procesamiento central (CPU o un MCU) en aplicaciones de televisión.

Como los circuitos se fueron haciendo más complejos con muchas conexiones entre periféricos, se necesitó de un método para simplificar los diseños y reducir los costos. El protocolo I2C cumplió con este requisito por medio de la limitación de pistas en el circuito impreso (PCB) y reduciendo el uso de entradas-salidas en el microprocesador.

En seguida se dan las principales características del Bus:

- Requiere un maestro (microcontrolador) y uno o más dispositivos esclavos.
- Está formado por dos hilos SDA (Serial Data) y SCL (Serial Clock) formando una forma de comunicación halfduplex.
- Cada dispositivo tiene una dirección única.
- El bus está limitado a una carga de dispositivos que no exceda los 400 pF.
- Es un bus serial bidireccional de 8 bits.
- Trabaja a velocidades desde los 100 kbps (estándar), 400 kbps (modo rápido) y hasta 3.4 Mbps (alta velocidad).
- Las líneas de conexión necesitan de 2 resistencias de pull-up para polarizarse y permanecer estables.

En el SIMSAT se tiene que la brújula digital usa esta clase de interfaz y se realiza la secuencia mostrada en la figura 6.13 para poder acceder a las lecturas de la brújula electrónica con base en el protocolo I2C, y se configuró la velocidad a **100 kbps**.

[1] Se tomaron en cuenta los números en negritas del patrón binario para esta interpretación.

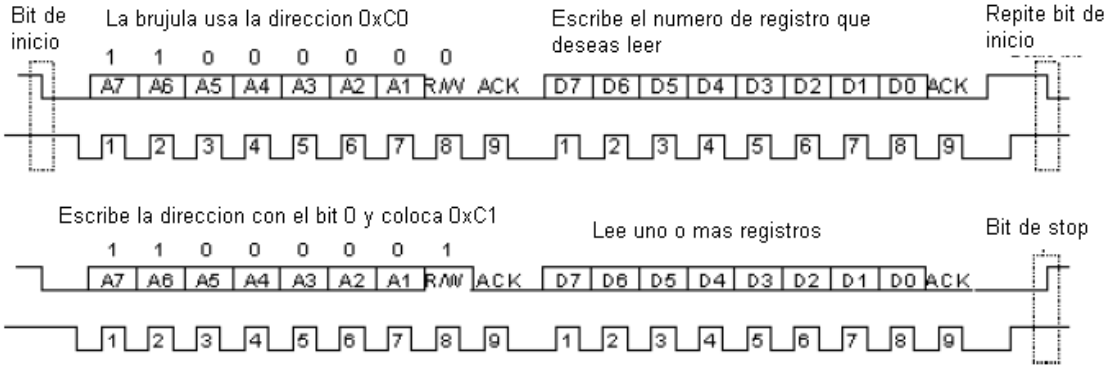


Figura 6.13 Diagrama de pulsos para el bus I2C de la brújula electrónica

Primero se envía un bit de inicio, después se envía la dirección que se desea leer o escribir que corresponde al dispositivo, en este caso la brújula electrónica cuya dirección es 0xC0; con el octavo bit que es el de lectura o escritura (R/W) en bajo para indicar escritura; el noveno bit corresponde a un bit de acknowledge indicando el inicio del siguiente byte a enviar, tal como puede apreciarse en la figura 6.13, y por último el número de registro a partir del cual se desea leer la brújula. Acto seguido, se repite el bit de inicio y ahora se envía de nuevo la dirección sumando el bit R/W en alto para indicar la lectura del dispositivo esclavo, enviando 0xC1 en el primer byte, enseguida es posible leer una serie de registros en la brújula hasta que sea enviado el bit de stop en el bus.

Los bytes que se leen son los siguientes:

Registro	Función
0	Número de revisión de software
1	Lectura de brújula, resolución de 0 a 255 (1.41º)
2,3	Lectura de brújula, resolución de 0 a 3599, representando 0 a 359.9º
4,5	Prueba interna
6,7	
8,9	Prueba interna, lectura directa desde el sensor 1
10,11	Prueba interna, lectura directa desde el sensor 2
12	Código de desbloqueo 1, requeridos para cambiar la dirección o reestablecer la calibración de fabrica
13	Código de desbloqueo 2
14	Código de desbloqueo 3
15	Registro de comando

Dado que la lectura se realiza de manera secuencial solo se toman los bytes necesarios, y se toman desde el 0 al 3 tomando como dato solo los bytes 2 y 3 que son los que brindan la lectura de la brújula con resolución de 0.1º.

Comandos: Los anteriores protocolos de comunicaciones interactuaron en el subsistema de la forma mostrada en la figura 6.14.

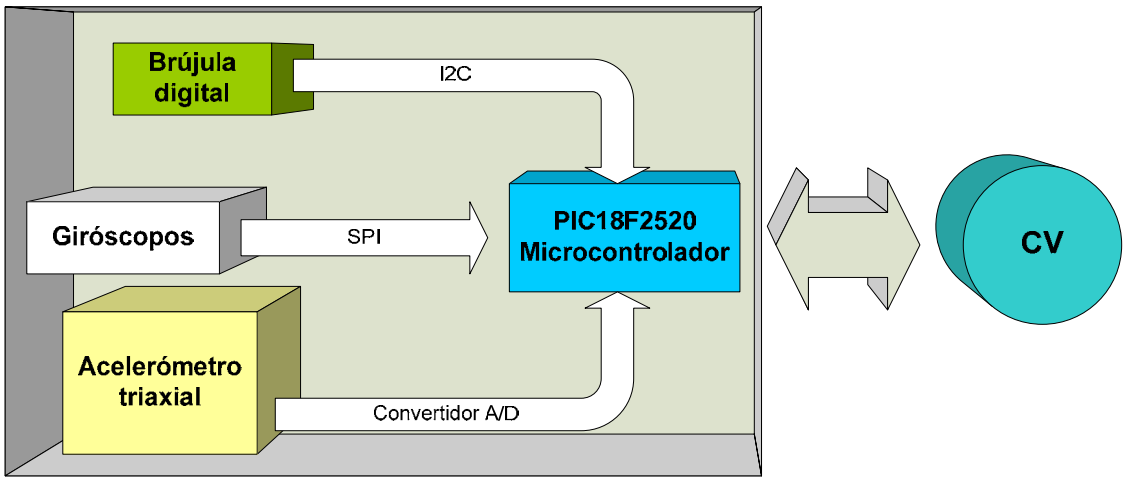


Figura 6.14 Protocolos empleados entre la tarjeta de sensores y la CV

Los primeros 3 comandos entre CV y la tarjeta de sensores piden la lectura de algún conjunto en particular de sensores o bien de todos ellos, la estructura que tienen es la siguiente:

1er byte	Función
B0	Lee brújula digital y acelerómetro
B1	Lee giróscopos
B2	Lee todos los sensores

Y los últimos dos comandos, son de reconfiguración del acelerómetro y envío de datos para la simulación de movimiento en tiempo real, como sigue:

1er byte	2do byte	Función
B3	1 - 4	Cambio de sensibilidad del acelerómetro 1 - 1.5 g 2 - 2 g 3 - 4 g 4 - 6 g
B4	No hay	Envío continuo de datos de sensores

Por ahora el envío continuo solo es realizado a partir de los datos generados por los giroscopios.

Los datos que se envían como telemetría (para la brújula y el acelerómetro) se transfieren en 1 ó 2 comandos:

No de Byte	Byte	Función
1	2F	Cabecera de comando
2	XX	Lectura de la brújula electrónica
3	XX	
4	XX	Lectura de acelerómetro en el eje X
5	XX	
6	XX	Lectura de acelerómetro en el eje Y
7	XX	
8	XX	Lectura de acelerómetro en el eje Z
9	XX	
10	41	Dato de relleno
11	41	Dato de relleno
12	41	Dato de relleno
13	41	Dato de relleno
14	XX	Suma de verificación o “checksum”

Para la lectura de los giróscopos:

No de Byte	Byte	Función
1	2F	Cabecera de comando
2	XX	Lectura de giróscopo X
3	XX	
4	XX	Lectura de giróscopo Y
5	XX	
6	XX	Lectura de giróscopo Z
7	XX	
8	XX	Lectura de temperatura en el giróscopo X
9	XX	
10	XX	Lectura de temperatura en el giróscopo Y
11	XX	
12	XX	Lectura de temperatura en el giróscopo Z
13	XX	
14	XX	Suma de verificación o “checksum”

La lectura de todo el bloque se hace por medio de los dos comandos anteriores.

6.5 Programa de simulación de la CV

Este programa de simulación de la CV se desarrolló en Visual Basic 6.0 para comprobar el adecuado funcionamiento de las tarjetas de estabilización y de sensores por medio de comunicación serial. Fue necesario elaborar este software debido a que la computadora de vuelo aún no estaba lista para emplearse en combinación con las tarjetas referidas.

Es necesario señalar que este software solo puede comunicarse con un subsistema a la vez ya que solo hay un puerto serial en la PC, en la CV esto es similar ya que solo hay un puerto serie por donde se comunican los subsistemas y ésta conmuta el canal de forma electrónica.

En la interfaz desarrollada en VB6 se armó una interfaz gráfica que indica cada una de las cosas que es capaz de hacer cada subsistema como se ve en las figuras 6.15 y 6.16.

Figura 6.15 Interfaz de estabilización

Figura 6.16 Interfaz de sensores

En la ventana se elige el subsistema sobre el cual se harán las pruebas seleccionando una “pestaña” que es la que indica el subsistema a probar. Una vez seleccionado se le indicarán las funciones que requiere realizar a través del chequeo derecuadros y la indicación numérica de lo que se quiere realizar.

Una vez seleccionadas todas las funciones deseadas es necesario presionar el botón “Crea Comando” que arma el comando asociado que la CV enviaría al subsistema. Al presionar el botón anterior, y ya que VB6 es un software activado por eventos, verifica que todas las elecciones hechas en la interfaz quepan en una trama de comando, de no ser así el software notifica al usuario para que reduzca la cantidad de comandos que desea ejecutar en el subsistema.

Cuando se tiene el comando creado, este puede visualizarse en la parte inferior de la interfaz, para enviar este comando se presiona “ENVIAR” y se envía por el canal serial COM1. En el lado derecho se encuentra una ventana de texto llamada “Recepción”; ésta monitorea el envió y recepción de mensajes del puerto serie, y realiza indicaciones a cerca de la recepción correcta o errónea de los comandos a cada subsistema.

En el caso de recepción de telemetría proveniente de la lectura de datos de los subsistemas como la corriente en el motor en el subsistema de estabilización o la lectura de todos los sensores en el subsistema de sensores, se desarrolló una forma de recepción y notificación de la correcta recepción de los datos, si alguno estuviera

incorrecto el subsistema lo notificaría por medio de la CV y se enviaría la telemetría una segunda ocasión. La forma de envió-recepción se ilustra en la figura 6.17.

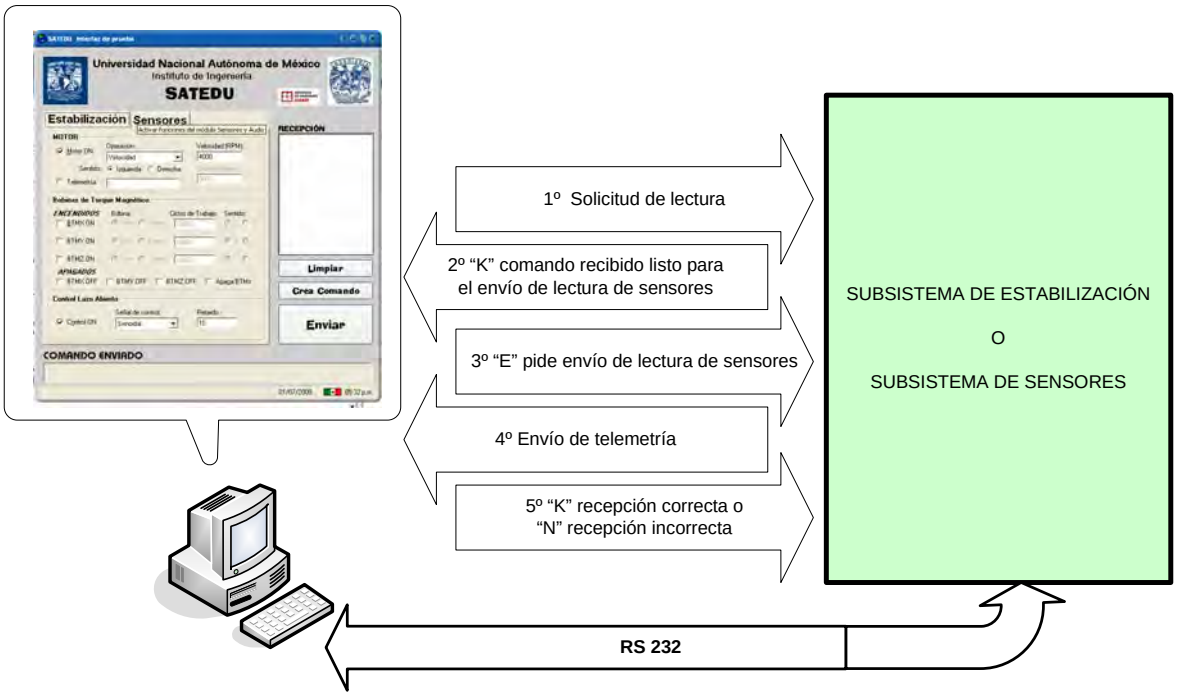


Figura 6.17 Envío recepción de telemetría

En el caso de sensores para el envío de datos para el monitoreo en tiempo real se envía un comando que le indica al subsistema que debe enviar continuamente la lectura de sensores para que la estación terrena pueda proyectar el movimiento del sistema. El envío de lecturas de sensores se hace cada 100 [ms], con lo que se tienen 10 muestras cada segundo.

La manera que se envían las lecturas es la siguiente:

Bytes	***1***	***2***	***3***	***4***	***5***	***6***	***7***
	0x2F	XX	XX	XX	XX	XX	XX
	Indica el inicio de la secuencia	Lectura del Gyro X		Lectura del Gyro Y		Lectura del Gyro Z	

Bytes	***8***	***9***	***10***	***11***	***12***	***13***
	XX	XX	XX	XX	XX	XX
	Lectura del acelerómetro eje X (10 bits)		Lectura del acelerómetro eje Y (10 bits)		Lectura del acelerómetro eje Z (10 bits)	

Por ahora se envía una trama solamente con las lecturas de los giróscopos y el acelerómetro cada 100 [ms], como se aprecia se elimina el checksum y solo se distingue el inicio de las lecturas, esto es para agilizar las lecturas de los sensores.

Las lecturas de los giróscopos son adaptadas, ya que los datos que envían no corresponden a lo que ellos son capaces de medir (grados por segundo), en la lectura estos poseen un bit de signo que indica el cambio de sentido de giro, a su vez que este número tiene que ser dividido entre 4.1 debido a que este número es el que representa, el numero de bits significativos por cada unidad de velocidad angular (grados por segundo [º/s]) , según lo especifica el fabricante en su hoja de datos. Los fenómenos que ocurren en el giróscopo discutidos en el capítulo 4 se trataron de reducir en la medida de lo posible dentro de este software para obtener resultados aceptables. Y los datos recopilados de los acelerómetros se dejaron para una posterior mejora en el desarrollo de un mejor algoritmo que en conjunto con los datos de los giróscopos puedan dar una medición precisa de la posición del sistema.

Bibliografía

1. Angulo, J. M., Romero, S. y Angulo, I. "*Microcontroladores PIC Diseño practico de aplicaciones Segunda Parte: PIC16F87x"*. 2ª edición. Ed. McGraw-Hill Interamericana. Mexico, 2003.
2. Microchip Technology. *"MPLAB IDE User's Guide"*, EEUU, 2006.
3. Microchip Technology*. "MPLAB C18 User's Guide",* EEUU, 2006.
4. Devantech. *"CMPS03 Compass Module"***.** Hoja de datos técnica, 2007. [http://www.robot-electronics.co.uk/htm/cmps3tech.htm]
5. Analog Devices. *"ADIS16100 ±300 º/s Yaw rate Gyro with SPI Interface*", Hoja de datos, 2006.
6. Valenti, C. *"AN937 Implementing a PID controller using a PIC18 MCU"*, Hoja de aplicación, Microchip Technology, EEUU, 2004.
7. Charais, J., y Lourens, R. *"AN964 Control of an Inverted Pendulum Using the PIC16F684",* Hoja de aplicación, Microchip Technology, EEUU, 2004.
8. Bolton, B. "*Mecatrónica: Sistemas de control electrónico en la ingeniería mecánica y eléctrica"*. 2ª edición. Ed. Alfaomega, México, 2001.
9. García, J., Rodríguez, J. y Brazález, A. "*Aprenda Visual Basic 6 como si estuviera en primero"*. Colección de Informática, Universidad de Navarra, 1999.

Capítulo 7

Capítulo 7

Pruebas de validación realizadas a las tarjetas de estabilización activa y sensores del SIMSAT

7.1 Introducción

Fue posible hacer muchas pruebas en protoboard para verificar la validez de la mayoría de la electrónica desarrollada en los subsistemas de estabilización activa y sensores, y con base en ello y a las hojas de datos de varios de los componentes seleccionados es que fue posible desarrollar un circuito impreso donde estos componentes serían montados. Debido a que la electrónica no fue probada en su totalidad, las pruebas definitivas se hicieron hasta tener el impreso fabricado y con los componentes montados.

En este capítulo se pretende describir las pruebas que se realizaron a los impresos que fueron fabricados para montar los sistemas de estabilización activa y sensores, y los pormenores que hubo con su diseño en hardware, así como de los detalles de consumo de los sistemas en funcionamiento.

7.2 Pruebas en la tarjeta de estabilización activa

Para verificar que la totalidad de los componentes funcionara adecuadamente se hizo una prueba parcial en cada componente del subsistema para verificar al final el adecuado funcionamiento del sistema en conjunto.

Se dividió a la tarjeta en partes para verificar su funcionamiento, como se describe a continuación:

1. ***Prueba de funcionamiento del microcontrolador y el software simulador de la CV:*** Ya se habían realizado pruebas de cargado de programa en protoboard, solo que para comprobar que el microcontrolador no se haya dañado en el proceso de soldado se le conectó el programador para que verificara su identidad y cargó un programa sencillo para que encendiera y apagara un LED. Posteriormente se le cargó el

programa principal y se hicieron pruebas de comunicaciones de comandos entre la tarjeta y la PC para verificar con el programa en Visual Basic funcionaba en conjunto con la tarjeta. Para conectar el subsistema a la PC se utilizó un circuito MAX232 montado en una protoborad ya que los voltajes en una PC por su puerto serial son de entre -12V y 12V al contrario del subsistema que son entre 0 y 5V. Finalmente, en la parte de Visual Basic hubo que depurar algunos problemas con los tiempos de recepción y procesamiento de los comandos ya que la PC tiene una velocidad de recepción y cálculo mayor que la del subsistema.

2. ***Prueba de funcionamiento del puente H del motor y su encoder:*** Para probar el funcionamiento del puente H se envío un comando que accionaba al motor, se fijó el comando de aplicar un ciclo de trabajo específico y para ver el funcionamiento del encoder se verificó en el osciloscopio. Además, se verificó la configuración adecuada de la salida PWM que se fijó en una frecuencia de **2.4 kHz** con una resolución de **14 bits**. El funcionamiento de ambos fue exitoso. Dado que aún no se contaban con la rueda inercial definitiva no fue posible corroborar los resultados teóricos obtenidos en el capítulo 2 para el algoritmo de velocidad.

3. ***Prueba de funcionamiento del circuito de sobrecorriente en el motor:*** Para esta prueba se puso en funcionamiento al motor en una velocidad alta y se bloqueó el eje del motor para forzar a que este empezara a drenar más y más corriente por sus embobinados, hasta que este se detuviera por acción del circuito instalado.

   Hubo algunos contratiempos con este circuito ya que el nivel de voltaje que otorgaba el circuito MAX4071 no aumentaba, sino que disminuía por lo que el circuito comparador era inservible. Para resolver el problema, hubo que abrir las pistas de la resistencia de sensado del circuito e invertirlas para que el voltaje proporcional a la corriente aumentara, demostrando un error en el circuito impreso. Adicionalmente se notó que el voltaje era en forma de una onda cuadrada por lo que se agregó un circuito RC para rectificarlo y se hizo una modificación en el software para apagar las salidas de PWM de manera definitiva hasta que se borre el bit que indica la falla, en lugar que ocurra un alto en el pin del microcontrolador, ya que esto provocaba un encendido y apagado intermitente en el motor.

4. ***Prueba de funcionamiento de los puentes H para las BTMs:*** Para esta prueba solo se configuraron los ciclos de trabajo para las 3 BTMs y se puede ver en la intensidad de luz en los LEDs indicadores, comprobando el funcionamiento de los generadores de PWM sobre las BTMs y se verificó la salida amplificada en los puentes H por el osciloscopio, confirmando que estaban funcionales. Ya que la configuración de salida del módulo PWM del microcontrolador está asociada con las salidas de PWM de las bobinas, estas también adoptan la misma frecuencia y resolución de salida, **2.4 kHz** y **14 bits** respectivamente.

7.3 Pruebas en la tarjeta de sensores

Al igual que en la parte anterior se hicieron pruebas parciales de funcionamiento, y se hicieron funcionar poco a poco los sensores que componían a esta tarjeta. Para esta tarjeta en particular no fue posible hacer pruebas preliminares en protoboard con dos de los tres tipos de sensores que se tenía pensado integrar, el acelerómetro y los giróscopos, dado que la lectura del acelerómetro solo consiste en la lectura de variaciones de voltaje

en sus salidas, fue posible simular esto en una protoboard sin problema a través del uso de varios potenciómetros. La única prueba que no fue posible realizar fue la de los giróscopos debido a que no se contaba con ninguno y ningún dispositivo similar.

Igualmente se dividió al subsistema en partes para verificar su funcionamiento, como sigue:

1. ***Prueba de funcionamiento del microcontrolador y el software simulador de la CV:*** Se realizaron pruebas semejantes a las que se realizaron en el subsistema de estabilización, básicamente se verificó el funcionamiento del microcontrolador y la comunicación con la interfaz en la PC.

2. ***Prueba de funcionamiento del acelerómetro:*** Para probar el acelerómetro se usó el programa de simulación de CV para leer todas las salidas del acelerómetro en un principio resultó en lecturas erróneas por cada eje según la hoja de datos, solo la del eje X eran correctas. Al verificar el impreso se observó que el ruteo estaba erróneo se conectó el pin 9 del acelerómetro, que corresponde a un pin que debe dejarse sin conectar, con la entrada del ADC del microcontrolador, dejando la salida 14 del acelerómetro desconectada, que era la que correspondía al eje Y, y cruzando la entrada que correspondía al eje Z con la del eje Y en el microcontrolador por lo que hubo que hacer una modificación en el software para que la reparación en la tarjeta impresa fuera mínima, cruzando las lecturas que correspondían al eje Z por las del eje Y y viceversa.

   Una vez hechas las adecuaciones se verificó que las lecturas fueran correctas, teniendo éxito esta vez.

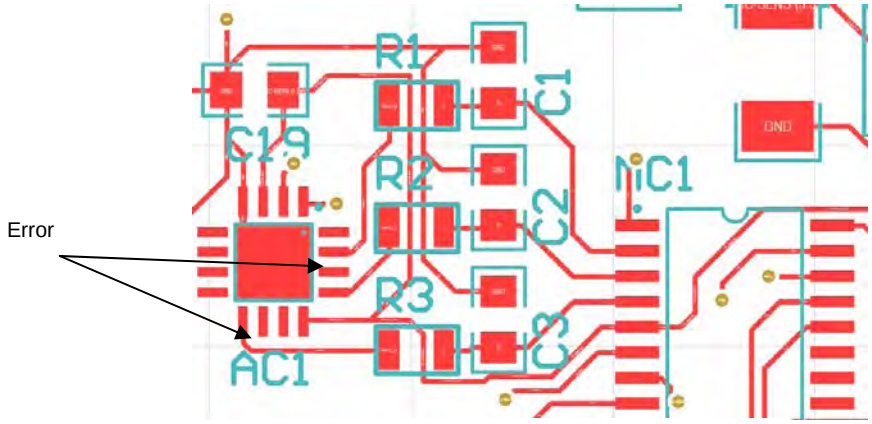


Fig. 7.1 Error en el primer circuito impreso

Para obtener las lecturas de la tarjeta se logró hacer a través del programa emulador de la CV. En este programa se reciben las lecturas del ADC del MCU ya que las salidas del acelerómetro son voltajes proporcionales a la aceleración para una sensibilidad específica. Para no hacer perder tiempo al MCU en la interpretación de estos voltajes como aceleración o inclinación esto lo hace el programa hecho en Visual Basic 6 ya que una PC tiene más velocidad y capacidad de procesamiento. En la interfaz se presentan los datos en su forma original (un número en decimal que arroja el ADC del MCU) y los grados de inclinación en cada eje pero usando una equivalencia de la expresión mostrada en el capítulo 4, donde por ejemplo para

obtener el ángulo de cabeceo basándose en el eje Z del acelerómetro se obtenía como,

$$^{o}cabeceo = \arcsin\left(\frac{a}{g}\right)$$

, y para obtener el ángulo a partir de los datos que da el acelerómetro, se realiza de la siguiente manera:

$$^{o}cabeceo = \arcsin\left(\frac{V_{medicion}[V] - V_{offset}[V]}{sensibilidad\left[\frac{V}{g}\right]\cdot 1g}\right)$$

Donde:

$V_{medicion}$ = Voltaje medido desde el ADC

V_{offset} = Voltaje de offset del acelerómetro ya que es posible detectar la dirección de la aceleración

sensibilidad = la sensibilidad escogida en el acelerómetro

Cabe mencionar que por ahora no se implantó en la interfaz el rango de medición de 360º propuesto en el capítulo 4.

3. ***Prueba de funcionamiento de la brújula electrónica:*** Para probar la brújula electrónica se usó un programa hecho para regresar lecturas continuas a una interfaz en Visual Basic, figura 7.2 para visualizar la orientación que indica la brújula, para verificar la adecuada programación del bus I2C, esta prueba fue exitosa y posteriormente anexada al software general de funcionamiento.

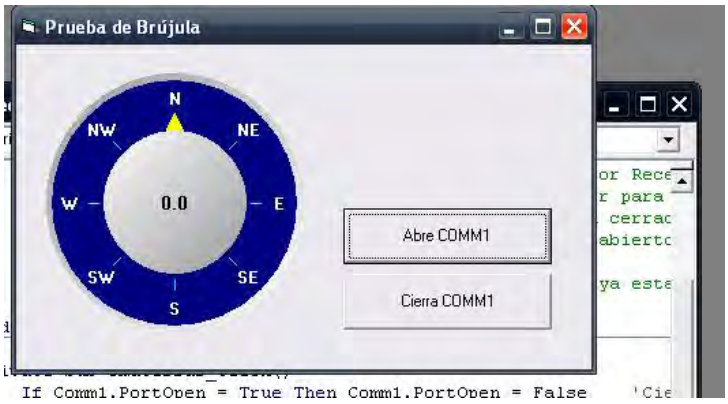


Fig. 7.2 Interfaz de prueba de la brújula electrónica

4. ***Prueba de funcionamiento del giróscopo:*** Para probar el giróscopo se hizo usó del programa de simulación de CV para leer las lecturas de un solo giróscopo que fue soldado a la placa, en un principio este no arrojaba lectura alguna, hubo algunos

errores en la forma de lectura por el puerto SPI, aun al corregirlos arrojaba lecturas erróneas, así se observó que las indicaciones en la hoja de datos del giróscopo eran erróneas, en la forma que tenía que ser configurado el bus SPI, ya que según la hoja de datos el bus del microcontrolador se configura en modo 2 como se observa en la figura 7.3. Pero al configurar el giróscopo en el modo 1, la lectura sí correspondía con la magnitud que se quería medir, la temperatura o la velocidad angular. Esto corresponde a la hoja de datos del ADIS16100 Rev. 0.

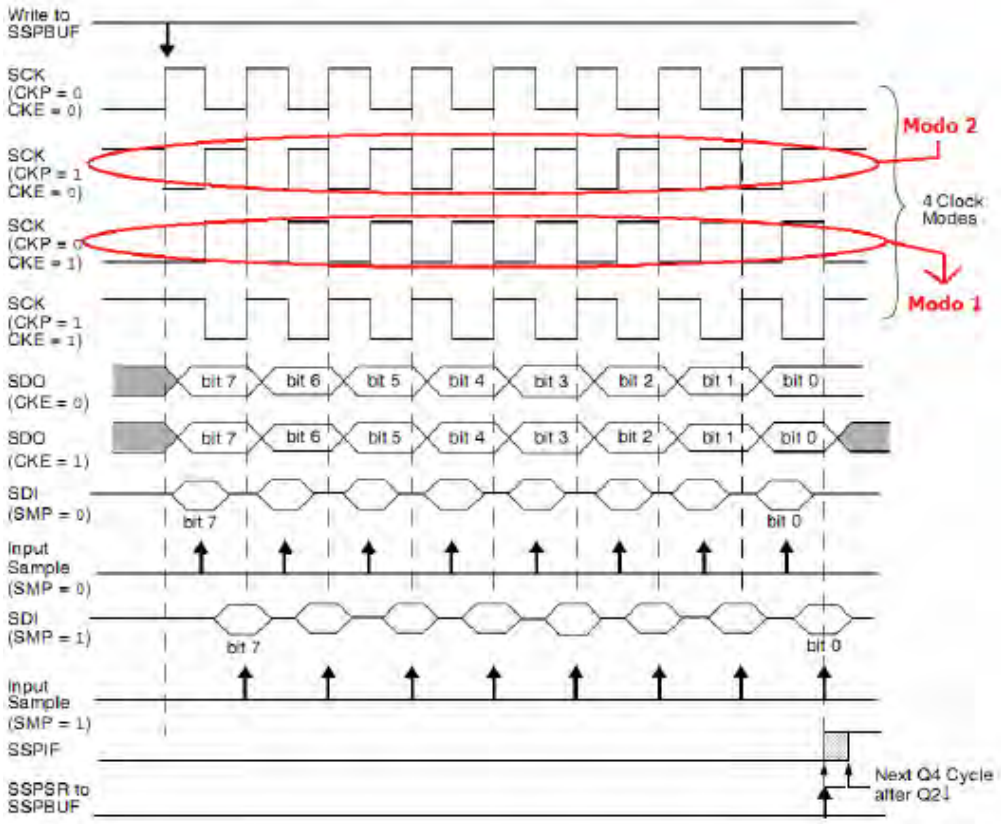
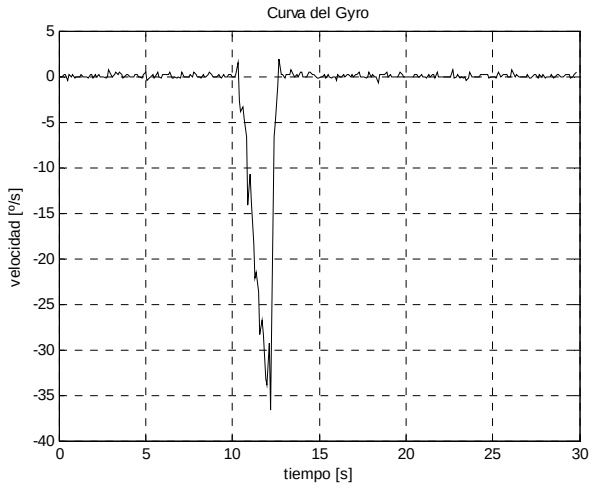


Fig. 7.3 Diagrama de pulsos de los modos en el bus SPI

Una vez resueltos los problemas que se tenían con el giróscopo se realizó otra prueba para poder interpretar adecuadamente los datos que arrojaba con el fin de posteriormente poder utilizarlos como una primera aproximación en el programa de visualización en tiempo real, para ello se realizaron 3 programas, el primero era un programa en el MCU que lo único que hacia era tomar un grupo de 300 muestras cada una tomada cada 100 [ms], una vez que el microcontrolador las tomaba, las enviaba por el puerto serie al segundo programa que fue hecho en Visual Basic 6, este interpretaba la serie de mediciones y las pasaba a forma decimal para poder ser guardadas en un archivo que tomaba el tercer programa hecho en MATLAB para poder graficar la respuesta del giróscopo en su manera original y mostrar como se vería este después de que fueran procesadas por un algoritmo.

Recapitulando un poco de lo visto en el capítulo 4, como ya se sabe no es posible obtener la posición de manera directa a través de las mediciones de los giróscopos debido a que estos son datos que se interpretan a manera de velocidad angular en grados por segundo [º/s], por lo que es necesario integrar los datos, esta integración se realiza de manera discreta, esto es, la suma de todos los datos que se hayan recolectado, en la figura 7.4 se observan los datos recolectados directamente del giróscopo en un muestreo, se hicieron dos muestreos con dos clases de movimientos uno donde solo se gira un poco la tarjeta con el sensor y el otro donde se le aplican una serie constante de movimientos.

a)



b)

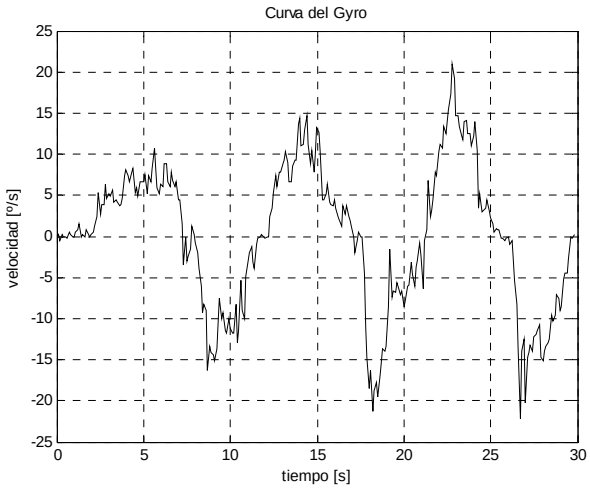


Fig. 7.4 Gráficas de datos del giróscopo, (a) con un movimiento y (b) con movimientos continuos

Como se observa en la figura 7.4 si se usan los datos directamente sobre el modelo virtual el efecto de movimiento se apreciaría de alguna manera cuando se moviera continuamente pero si solo se realiza un movimiento, el modelo tendría que quedarse fijo en esa posición pero como se observa en la gráfica de la figura 7.4a el modelo regresaría a su posición neutral después de un tiempo siendo que solo se realizó un movimiento y se quedó fijo en esa posición. Por ello hay que realizar una integración para poder interpretarlo como posición. El integrador más sencillo es:

$$integral\ (n) = integral\ (n\text{-}1) + valor\ inicial$$

Y aunque es fácil de aplicar este algoritmo hay otros que pueden producir mejores resultados suavizando las curvas de la posición debido a los cambios de latencia que existen en el muestreo, como es el método de Runge-Kutta (RK4) y está dado por:

$$i(n) = i(n-1) + \frac{y(n-3) + 2y(n-2) + 2y(n-1) + y(n)}{6}$$

Donde:

n = número de muestra
i = valor de la integral
y = muestra obtenida del giróscopo

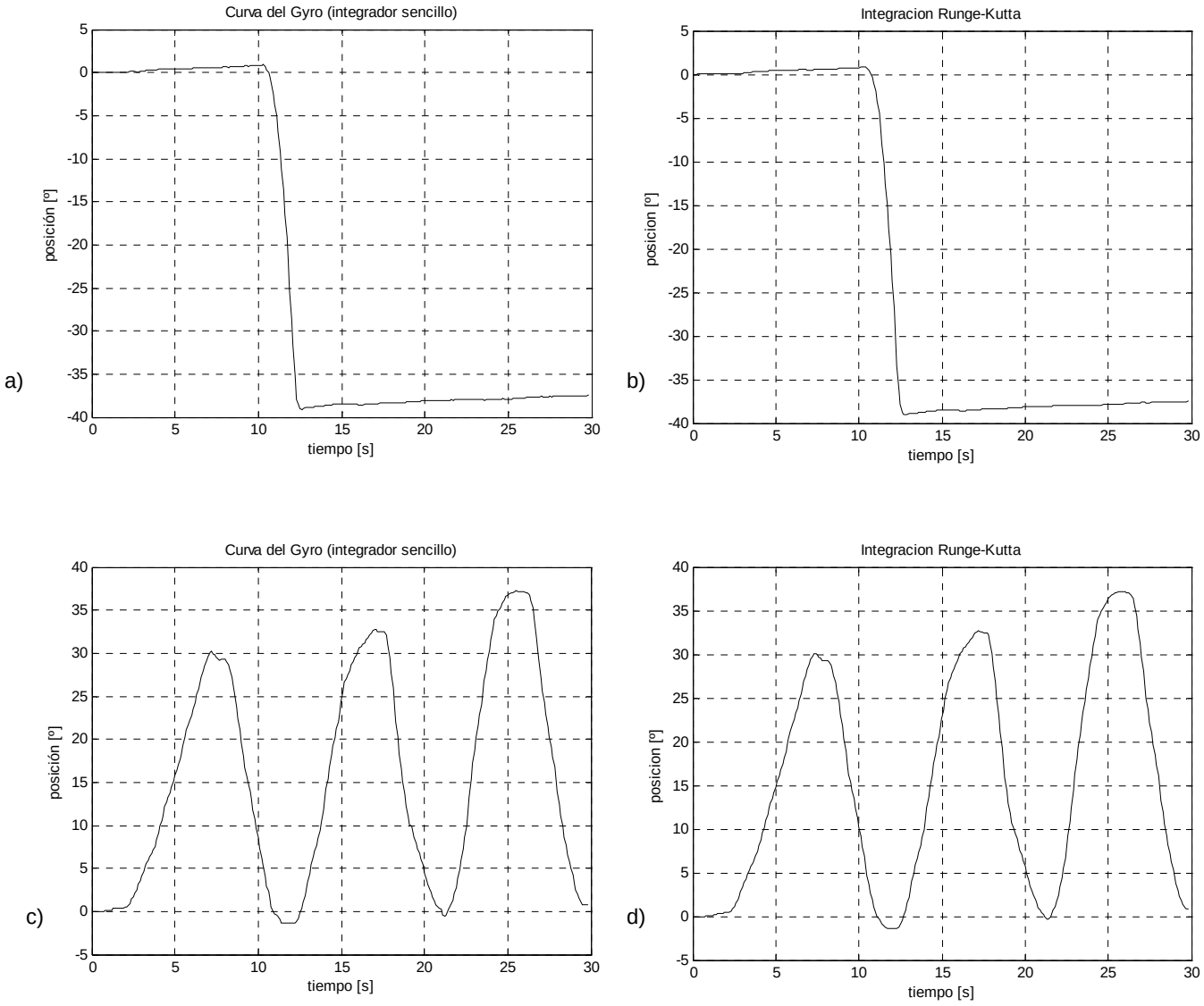


Fig. 7.5 Gráficas de datos del giróscopo integradas, (a y c) por el integrador sencillo y (b y d) por medio del integrador Runge-Kutta

Ambos métodos de integración fueron aplicados para comparar cual es mejor pero como se observa en las gráficas de la figura 7.5, las diferencias son poco perceptibles a primera vista, pero el método de Runge-Kutta ofrece una salida mas suavizada.

7.4 Consumos de energía en los subsistemas

En las siguientes tablas se presentan los consumos de corriente en cada subsistema, uno estimado por los datos de las hojas de especificaciones y el registrado en el laboratorio.

	Componentes	Hojas de datos
	PIC18F4431	Max 200 [mA] (depende) Uso aprox. 100mA
	Oscilador 10MHz	45 [mA]
	Puente H TA7291S	6.5 - 8 [mA]
	3 Puentes H L293DD	3 x 16-44 [mA]
	Max4071	0.1 - 0.25 [mA]
	OPAM CA3140	1.6 [mA]
	OPAM LM6511	2.7[mA]
	Motor	Max 169 [mA]
	Encoger	8 [mA]
	3 LEDs	3 x 10 [mA]
Total estimado	**444.5 [mA] (funcionando al máximo)**	
Valor experimental	Midiendo desde la fuente de 5 V	150 [mA]
	Midiendo desde la fuente de 12 V	110[mA]
Total experimental	**260 [mA][1]**	

	Componentes	Hojas de datos
	PIC18F2520	Max 200 [mA] (depende) Uso aprox. 100 [mA]
	Oscilador 10MHz	45 [mA]
	Acelerómetro MMA7260	0.5 – 0.8 [mA]
	3 Gyros ADIS16100	3 x 7-9 [mA]
	Multiplexor 74HC4053	15 [µA]
	Brújula CMPS03	Max. 25 [mA]
	3 LEDs	3 x 10 [mA]
Total estimado	**209.815 [mA] (funcionando al máximo)**	
Valor experimental	Midiendo desde la fuente de 3.3 V	30 [mA]
	Midiendo desde la fuente de 5 V	100[mA]
Total experimental	**130 [mA]**	

[1] El motor que se usó para esta prueba no tenía el diseño final de la rueda sino un sustituto para ponerle carga.

Bibliografía

1. Analog Devices. *"ADIS16100 ±300 º/s Yaw rate Gyro with SPI Interface*", Hoja de datos, 2006.
2. Miranda, C. y Ronquillo, J. "*Diseño y construcción de bus de datos y sensores para las prácticas de NACC (Navegación Aérea, Cartografía y Cosmografía)"*. Tesis de Licenciatura. Universidad Politécnica de Cataluña. España, 2008.
3. Pycke, T. *"Gyroscope to roll, pitch and yaw"*, 2008.
[http://tom.pycke.be/mav/70/gyroscope-to-roll-pitch-and-yaw]

Capítulo 8

Conclusiones y Recomendaciones

8.1 Conclusiones

De la tesis desarrollada se concluye:

- ✓ Se realizó de manera exitosa el diseño y prueba de los subsistemas de estabilización activa y sensores pudiendo depurar prácticamente todos los errores que surgieron durante su instalación final en el circuito impreso, haciéndola totalmente compatible con la computadora de vuelo del SIMSAT.

- ✓ Se pudieron desarrollar perturbaciones a través del motor emulando una onda senoidal, triangular y diente de sierra, generando movimientos de posición en el SIMSAT, los cuales son fáciles de observar por los usuarios.

- ✓ Se pudo realizar de manera parcial un algoritmo de control de velocidad en lazo cerrado, especificada al motor de manera digital, ya que aunque se programó no se realizaron pruebas en el sistema definitivo.

- ✓ El software realizado para ambas tarjetas está completamente funcional, además de constituir una interfaz a través de Visual Basic que permite emular a la CV, para ser independiente de esta última. Adicionalmente, la interfaz simuladora de la CV tiene la posibilidad de actualizarse y de anexar más subsistemas.

- ✓ Fue posible realizar un sistema de seguimiento del movimiento del SIMSAT a través de los sensores, que pudo ser verificado en forma parcial.

- ✓ Fue posible adecuar el Watchdog Timer dentro del microcontrolador de manera tal que si el programa llegara a entrar en un bucle infinito en segmento del mismo, este pueda auto inicializarse de manera automática.

- ✓ Fue posible minimizar el uso del CPU de los microcontroladores para ahorrar energía, ya que como ambos subsistemas siempre están a la espera de las ordenes de la computadora de vuelo, caso de no recibir ninguna, los CPUs de los microcontroladores se ponen en modo de bajo consumo y solo se dejan funcionando algunos de sus periféricos como es el módulo de PWM o el módulo de comunicación serial en caso de recibir un nuevo comando y encender de nuevo el CPU.

8.2 Recomendaciones

- Usar puentes H de fabricación MOSFET en lugar de aquellos fabricados con transistores, ya que la salida que da el puente se ve disminuida en tensión debido a la caída producida en cada transistor, algo que no ocurre en los MOSFET.

- Usar capacitores especiales en los puentes H entre las terminales del motor para un mejor filtrado del ruido EMI.

- Anexar un circuito MAX233 (interfaz RS232) y un conector adicional para que se pueda dar una comunicación serial entre la PC y las tarjetas, y evitar así la reprogramación de la computadora de vuelo o el uso de una tarjeta adicional o una protoboard en caso de que se deseen realizar pruebas individuales de los subsistemas.

- Reevaluar el motor usado y el uso de una tercera BTM.

- Mejorar el algoritmo de control de velocidad del motor en base a las pruebas que se realicen.

- Tratar de anexar algunos sensores (navegación inercial) a la tarjeta de estabilización.

- Mejorar el software de interpretación de lecturas de los sensores en la parte de la interfaz de la PC por medio de Visual Basic 6 ya que, en la interfaz realizada al recibir los datos de los sensores, la interpretación de la posición angular que se hace a través de estos solo se hace por medio de los giróscopos y no se consideran las lecturas del acelerómetro triaxial para corregir los problemas que presentan los giróscopos como el bias y la deriva además de que tampoco se considera la influencia de la temperatura sobre la medición de los mismos.

- Anexar más sensores en la tarjeta como puede ser un sensor de sol o sensar el voltaje de los paneles solares para simular un seguimiento de sol.

- Cambiar del bus de comunicaciones de red interna entre subsistemas (RS-232, a 9.6 kbauds) por uno más rápido y con menor cantidad de líneas de conexión entre la CV y los demás subsistemas como el bus I2C, que es capaz de funcionar a una velocidad estándar de 100 kbauds, además de que mediante 2 líneas seria capaz de conectar a todos los subsistemas del SIMSAT sin tener que recurrir al uso multiplexores en la computadora de vuelo y la programación extra que esto conlleva con el uso de los mismos.

Apéndice A

Cargado de programa en los microcontroladores PIC

El cargado de programas a los microcontroladores PIC se hizo a través de un programador llamado INCHWORM desarrollado por la empresa BLUEROOM ELECTRONICS que amablemente comparte su diseño, solo hubo que diseñar la placa y colocar los componentes, además es compatible con la interfaz para MPLAB.

En la figura A.1, puede apreciarse el programador fabricado, este se conecta al microcontrolador por medio de un cable, comunicando 5 pines al microcontrolador, Vpp, VDD, Vss, PGD y PGC, ademas es necesario agregar algunos componentes adicionales en nuestro PIC para poder programar una resistencia y un diodo para poder mantener el voltaje de programación en la Terminal Vpp.

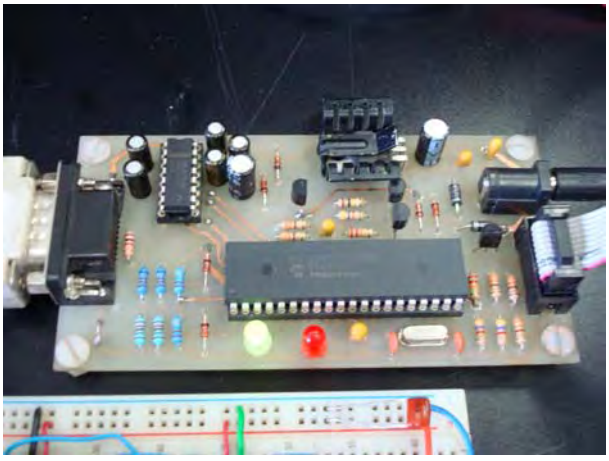


Figura A.1 Programador construido a partir del diagrama del Inchworm

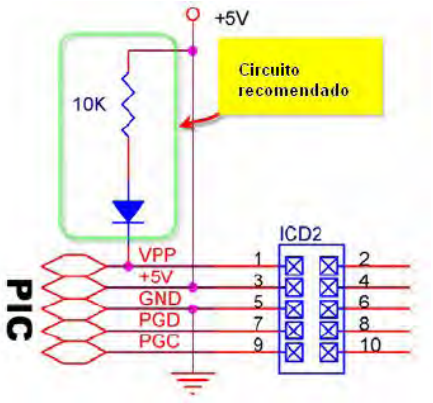


Figura A.2 Componentes adicionales para programar el PIC en el circuito

Ya conectado al microcontrolador, se abre MPLAB IDE para poder programarlo, se siguen los siguientes pasos:

1. Al conectarlo por primera vez es necesario asegurarse en el Administrador de dispositivos que la configuración del puerto serial conectado al programador tenga el control de flujo por Hardware y en opciones avanzadas los búferes FIFO estén deshabilitados.

2. Primero hay que abrir el proyecto y compilarlo para que genere el archivo en formato .HEX el cual comprende los datos que serán grabados en el PIC.

3. En la barra de MPLAB en la parte de programadores, se selecciona MPLAB ICD2 y aparecerá una pequeña barra de herramientas.
4. Ya que se tiene todo el hardware conectado solo hay que presionar “Reset and Connect ICD”, y MPLAB se empezará a “dialogar” con el dispositivo, dependiendo del PIC el programador necesitara descargar un sistema operativo para poder programarlo, hay que esperar un momento para MPLAB nos indique que esta listo.

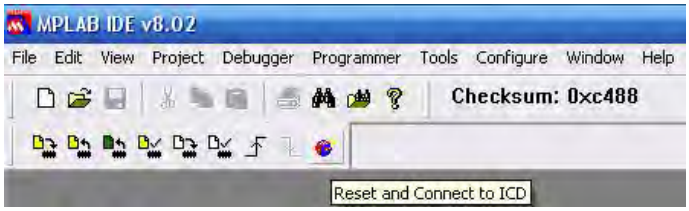


Figura A.3 Barra del programador ICD2

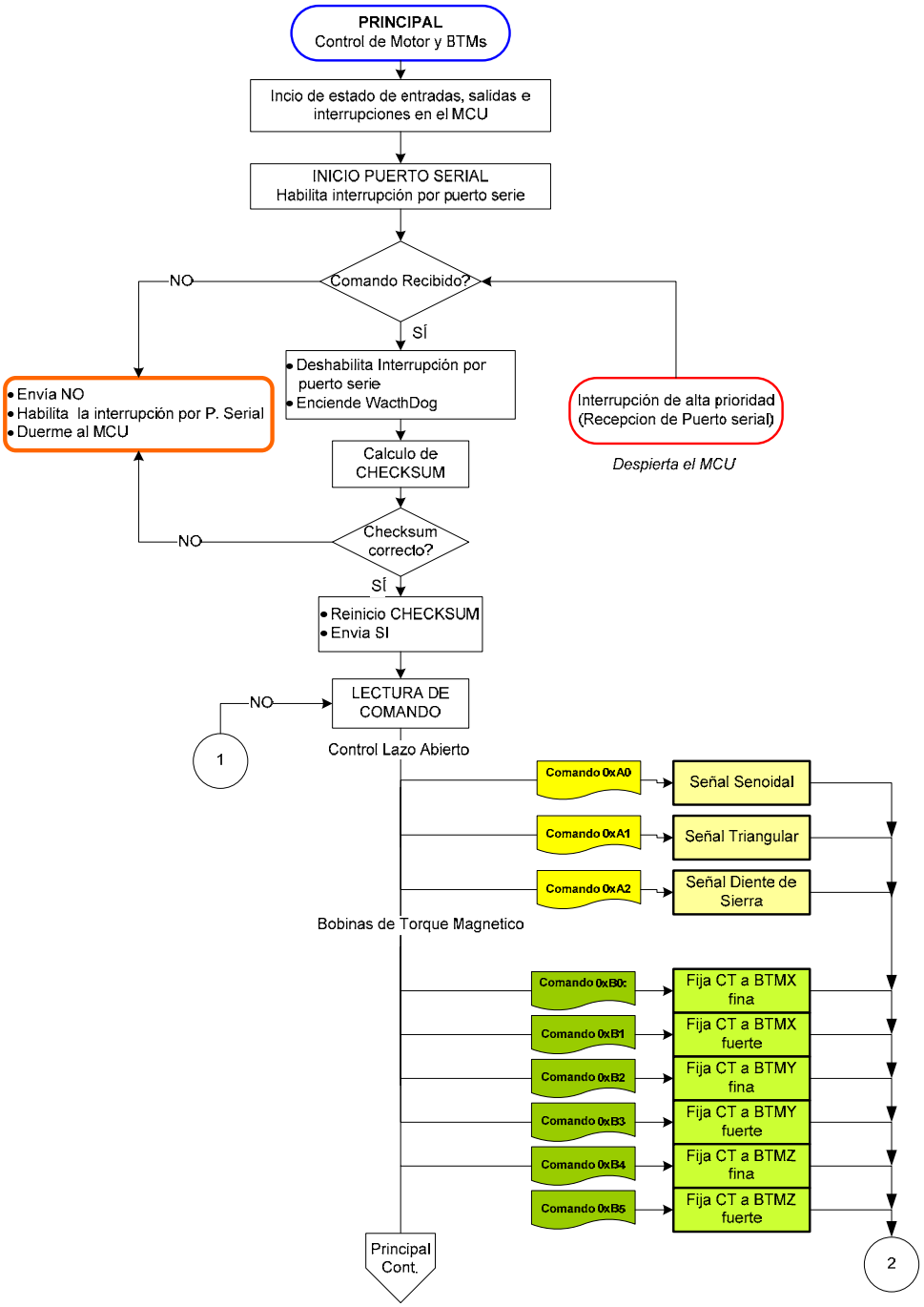
5. Ya que esta listo los demás botones de la barra de herramientas se habilitarán y ahora es posible programar al PIC, entre otras funciones. Se pulsa programar todo, y hay que esperar un momento a que termine, una vez hecho es posible ver si el código funciona quitando el reset en MPLAB.

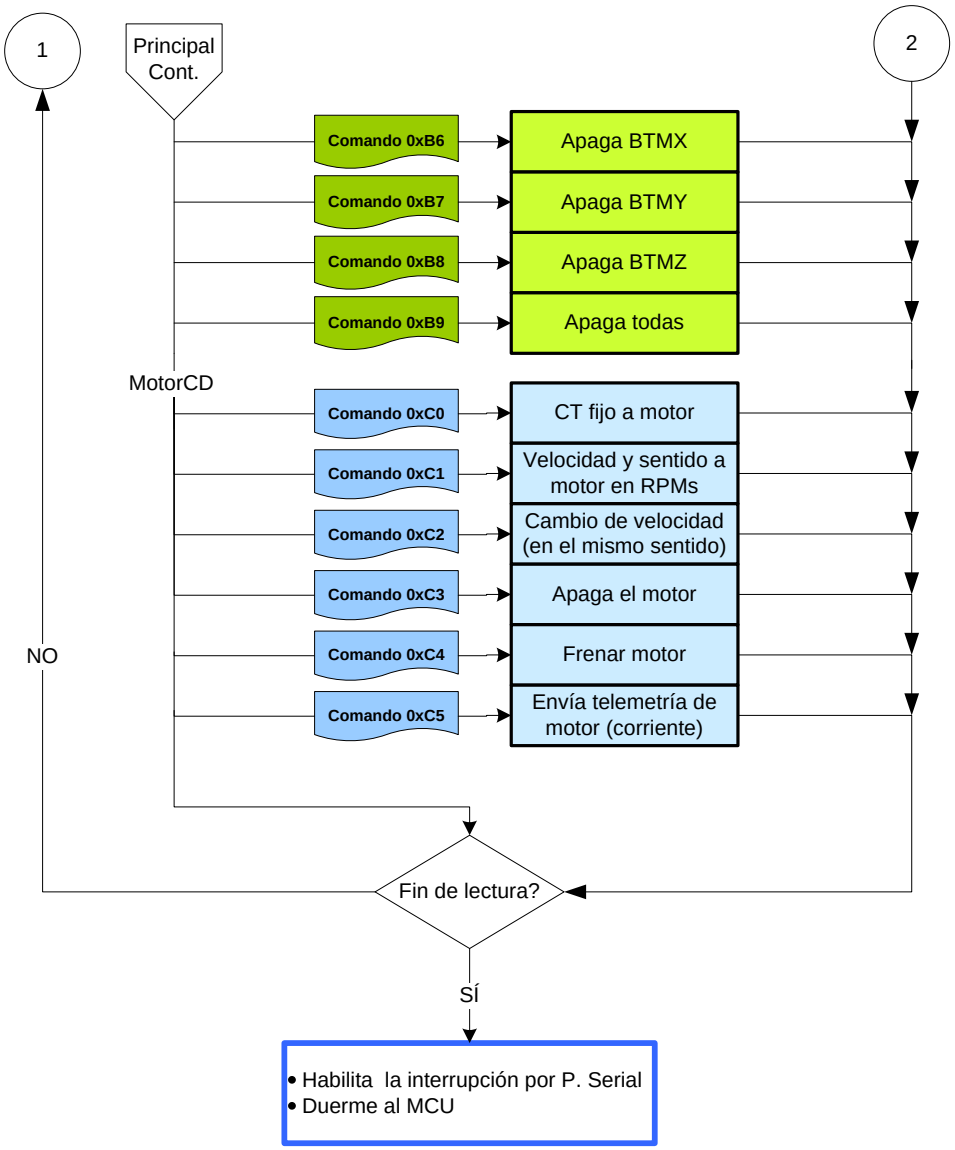
La ventaja principal de este programador yace en que sin desconectar nada del circuito es posible energizar y probar el circuito conectado al microcontrolador, si la carga no es demasiado grande, ya que para poder accionar otros dispositivos como pueden ser motores se vuelve necesario usar una fuente adicional para accionar la carga sin que se produzca una disminución del voltaje y causar que el circuito llegue a fallar.

Apéndice B

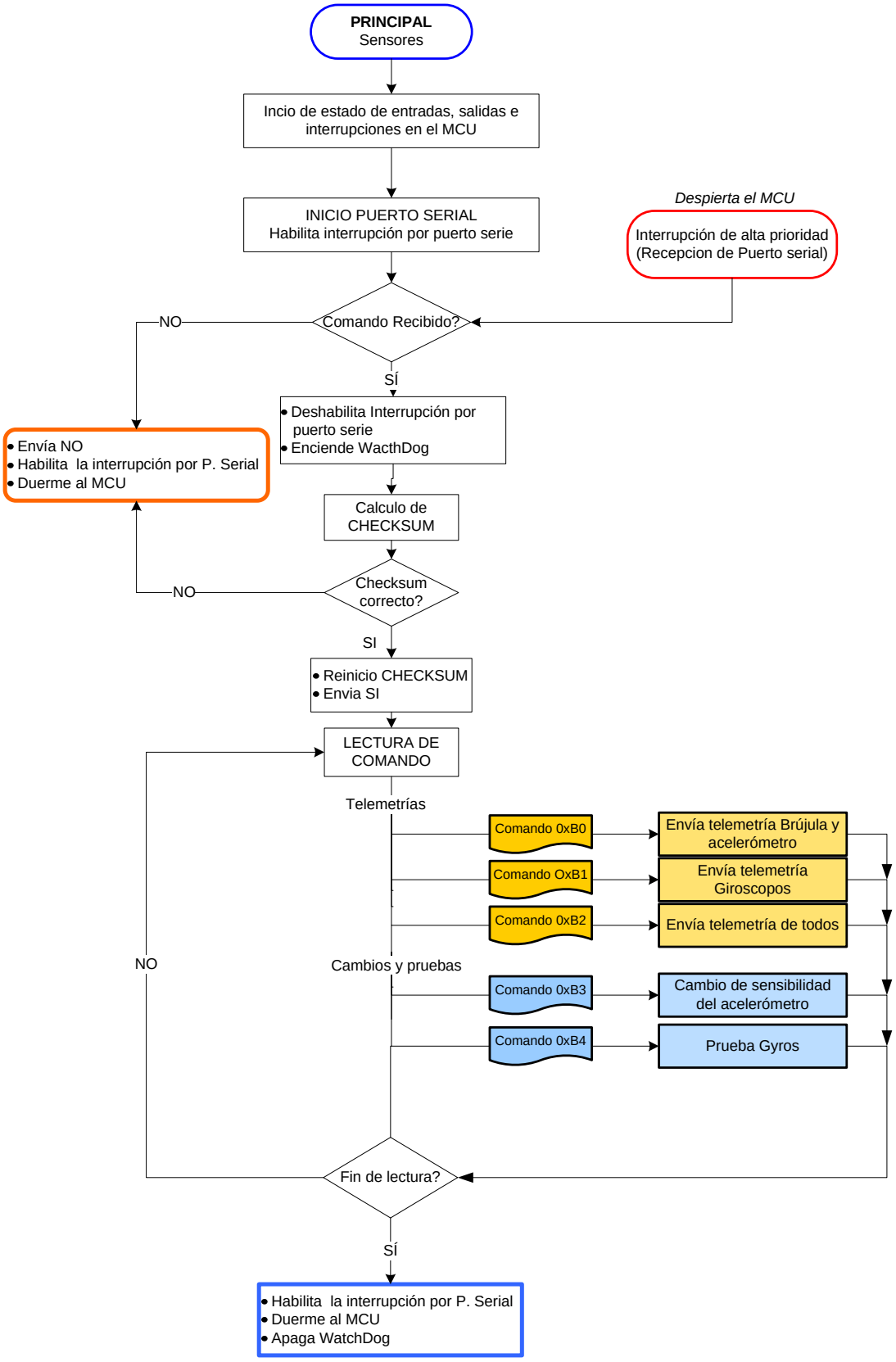
Diagramas de flujo generales

Estabilización

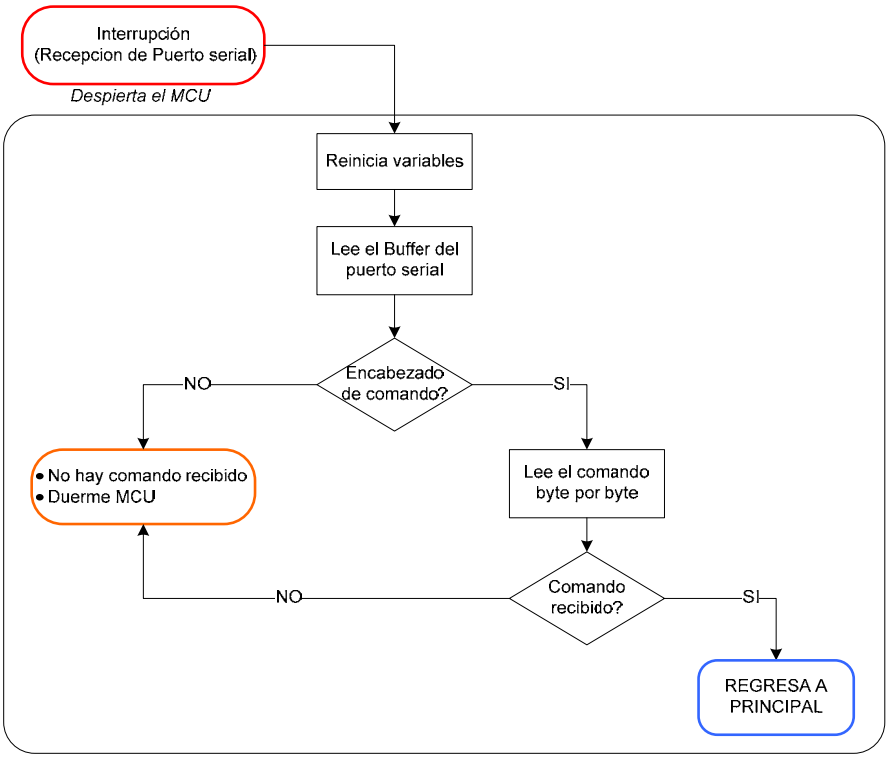



1
Principal Cont.
2
Comando 0xB6
Comando 0xB7
Comando 0xB8
Comando 0xB9
Apaga BTMX
Apaga BTMY
Apaga BTMZ
Apaga todas
MotorCD
Comando 0xC0
Comando 0xC1
Comando 0xC2
Comando 0xC3
Comando 0xC4
Comando 0xC5
CT fijo a motor
Velocidad y sentido a motor en RPMs
Cambio de velocidad (en el mismo sentido)
Apaga el motor
Frenar motor
Envía telemetría de motor (corriente)
NO
Fin de lectura?
SÍ
• Habilita la interrupción por P. Serial
• Duerme al MCU

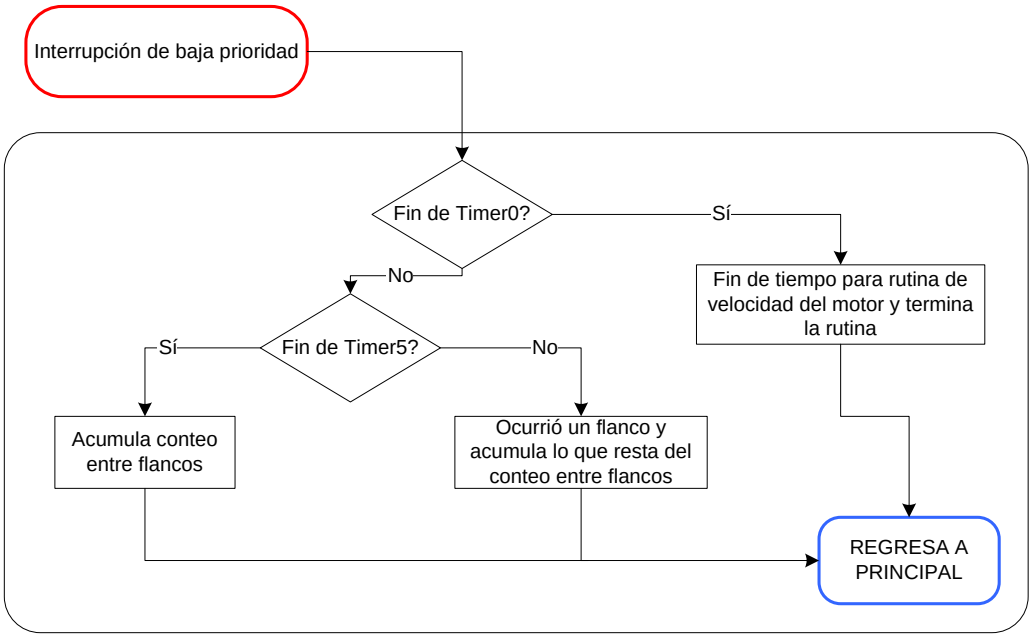
Sensores


PRINCIPAL
Sensores
Incio de estado de entradas, salidas e interrupciones en el MCU
INICIO PUERTO SERIAL
Habilita interrupción por puerto serie
Despierta el MCU
Interrupción de alta prioridad (Recepcion de Puerto serial)
Comando Recibido?
NO
SÍ
Deshabilita Interrupción por puerto serie
Enciende WacthDog
Envía NO
Habilita la interrupción por P. Serial
Duerme al MCU
Calculo de CHECKSUM
Checksum correcto?
NO
SI
Reinicio CHECKSUM
Envia SI
LECTURA DE COMANDO
Telemetrías
Comando 0xB0
Envía telemetría Brújula y acelerómetro
Comando OxB1
Envía telemetría Giroscopos
Comando 0xB2
Envía telemetría de todos
NO
Cambios y pruebas
Comando 0xB3
Cambio de sensibilidad del acelerómetro
Comando 0xB4
Prueba Gyros
Fin de lectura?
SÍ
Habilita la interrupción por P. Serial
Duerme al MCU
Apaga WatchDog

Interrupción por puerto serie en ambos diseños



Interrupción de baja prioridad en estabilización



Apéndice C

Programas de los microcontroladores PIC

Programa en el MCU de estabilización

```
/*********** Software Control del Modulo de Estabilizacion *****************/

//      Este software interfacea a la computadora de vuelo con el modulo de control /////
       encargado del control y el intercambio de informaccion entre ambos modulos

// Librerias de inicio
#include <p18f4431.h>
#include <usart.h>
#include <delays.h>
#include <adc.h>

//  CONFIGURACIONES
#pragma config WDTEN = OFF    // deshabilita el WATCHDOG TIMER
#pragma config WDPS = 256     // watch dog relacion 1:2048, reset en aprox. 8 seg
#pragma config OSC = HSPLL    // habilita el PLLx4 con un oscilador de 10 Mhz (HSPLL)
#pragma config PWM4MX = RD5   // coloca la salida de PWM4 por RB5
#pragma config PWMPIN = OFF   // se enciende el modulo de pwm automaticamente despues del
                              // reset == NO
#pragma config FLTAMX = RD4   // pin de falla de MOTOR

#pragma idata         //  almacenamiento en memoria estatica

// FUNCIONES AGREGADAS

       // DATOS

void maneja_IntRx (void);                   // funcion para manejar interrupciones
void overflow (void);
char Lee1b_Redint_PoR (void);               // lee 1 byte
char putchP_R(char c);                      // Transmite por red interna
void checksum_calc(void);
void esc_cmd_redint_PoR (void);             // envio de TELEMETRIA
void rellenaCOM(char fill);                 // rellena comando de envio con 0
void WDT_ON (void);                         // watchdog encendido y reiniciado
void WDT_OFF (void);                        // watchdog apagado

       //CONTROL LAZO ABIERTO
void senoidal ( unsigned char frec );
void triangular ( unsigned char frec );
void dientesierra (unsigned char frec );

       // MOTOR
void AbreMotor (void);// fijado del periodo
void SetDCMotor (unsigned int ciclotrabajo);  // fija ciclo de trabajo
void Velocidad (unsigned int vel);
void CambVeloc (unsigned int vel);
void ApagaMotor (void);
void FrenaMotor (void);
```

```
void telemetria (void);
void FallaMotorON(void);
void FallaMotorOFF(void);

       // BOBINAS TM
void AbreBTMs (void);         // abrir bobinas de TM
void BTMx_fina (void);        // SW de bobinas finas y fuertes
void BTMx_fuerte(void);
void BTMy_fina (void);
void BTMy_fuerte(void);
void BTMz_fina (void);
void BTMz_fuerte(void);
void SetBobinaX(unsigned int ciclotrabajo_dircorr);        // asigna valores a BTM's
void SetBobinaY(unsigned int ciclotrabajo_dircorr);
void SetBobinaZ(unsigned int ciclotrabajo_dircorr);
void ApagaBTMx(void);               // apagado de bobinas
void ApagaBTMy(void);
void ApagaBTMz(void);
void ApagaBobinas(void);
void falla_motor(void);             // prueba si el motor esta en falla

// VARIABLES GLOBALES
unsigned char CmdRecibido;    // solo algunas interrupciones hacen 'S' esta bandera
char Fallo_TOTAL_RedInt;      // byte de indicacion de falla en transimision y recepcion //
                              de datos,    => bandera
char nvodato;                 // byte leido de red interna
int chkext;
unsigned char cmdRx [14];      // comando de red interna Rx
unsigned char cmdTx [14];      // comando de red interna Tx

char Num_IntentosTxRx;  // numero de intentos
int aux_comando;      // variable auxiliar en el procesamiento de comandos

unsigned char ctk,hay_dato,bandera_dato;
int i;
char r;

union pwmct{
       unsigned char lpwm;
       unsigned char hpwm; };

//     MOTOR
unsigned int velocidad;
signed short long medida_sensor;
unsigned char sentido;
signed short long medicion;
signed int salida_pi;
signed short long vel_deseada,error,a_error;
signed short long proporcional,integral;
unsigned char mem_sentido;
unsigned int sent;
unsigned char kp,ki;
unsigned char timeup; // tiempo limite de estabilizacion
unsigned char alto;           // bandera detiene el motor

#pragma code

//     HAB / DESHAB  INTERRUPCIONES
#define Deshabil_IntRx()            PIE1bits.RCIE = 0;
#define Habil_IntRx()               PIE1bits.RCIE = 1;

// Deshabilita las interrupciones por recepcion de datos
#define Deshabil_IntActualRI()  PIE1bits.RCIE = 0;
#define Habilita_IntActualRI()  PIE1bits.RCIE = 1;  // Habilita las interrupciones

#define Idle()  OSCCONbits.IDLEN = 1;  // modo idle CPU apagado y perifericos encendidos

// etiquetas rutina de velocidad
#define velocidad_minima     1200
#define delta_error          20     // umbral de incertidumbre para errar la medicion
#define periodo_vel          37500000
```

```
#define t_asentamiento       5      // tiempo de asentamiento de la rutina de velocidad
#define p_control            1
#define i_control            1
#define kp_dec               2
#define ki_dec               16


#define t_muestreo1   0x0b
// timer0 1:16, interrupt despues del OVERFLOW, po lo que 0xFFFF-(tiempo deseado)
#define t_muestreo2   0xdb
// 0xbdb => 50ms, 0x85ed => 25ms, 0xe795 => 10ms, 0xf3ca => 5ms

#define entrada_captura      0x45
// configuracion del registro de control de captura  0x45 => cada pulso, 0x44 => cada 16
pulsos

#define SatMax        8388000
#define SatMin        -8388000


//     Tabla del Seno 200 puntos

const rom int t_seno [] =
{
       0,257,514,771,1028,1285,1541,1797,2052,2307,
       2562,2816,3069,3321,3573,3823,4073,4322,4569,4816,
       5061,5305,5548,5789,6029,6268,6505,6740,6974,7206,
       7436,7664,7891,8115,8338,8558,8776,8992,9206,9418,
       9627,9834,10039,10241,10441,10637,10832,11023,11212,11399,
       11582,11762,11940,12115,12286,12455,12621,12783,12942,13098,
       13251,13401,13547,13690,13830,13966,14098,14228,14353,14476,
       14594,14709,14821,14928,15032,15133,15229,15322,15411,15496,
       15578,15655,15729,15799,15865,15927,15985,16039,16089,16136,
       16178,16216,16250,16281,16307,16329,16347,16361,16371,16377,
       16380,16377,16371,16361,16347,16329,16307,16281,16250,16216,
       16178,16136,16089,16039,15985,15927,15865,15799,15729,15655,
       15578,15496,15411,15322,15229,15133,15032,14928,14821,14709,
       14594,14476,14353,14228,14098,13966,13830,13690,13547,13401,
       13251,13098,12942,12783,12621,12455,12286,12115,11940,11762,
       11582,11399,11212,11023,10832,10637,10441,10241,10039,9834,
       9627,9418,9206,8992,8776,8558,8338,8115,7891,7664,
       7436,7206,6974,6740,6505,6268,6029,5789,5548,5305,
       5061,4816,4569,4322,4073,3823,3573,3321,3069,2816,
       2562,2307,2052,1797,1541,1285,1028,771,514,257
};

#pragma code



// PROGRAMA PRINCIPAL

void main(void)
{
   // Habilita interrupciones y niveles de interrupcion
  RCONbits.IPEN = 1;                 // habilita niveles de interrupcion
  INTCONbits.GIE = 1;                // hab interrupcion global
  INTCONbits.PEIE = 1;        // hab interrupcion de perifericos
  INTCONbits.TMR0IE = 0;      // hab interrupt en TMR0
  PIE3bits.TMR5IE = 1;        // mascara de interrupcion a TMR5
  PIE3bits.IC2QEIE = 1;              // interrupcion al recibir un pulso del encoder
  IPR1 = 0x20;                       // interrupcion de ALTA prioridad SOLO en RX
  IPR2 = 0;
  IPR3 = 0;                   // interrupcion de BAJA prioridad en lo demas

  INTCONbits.TMR0IF = 0;             // baja bandera por OVERFLOW TMR0
  PIR1bits.TMR1IF = 0;        // baja bandera por OVERFLOW TMR1
  PIR3bits.PTIF = 0;          // baja bandera interrupcion por OVERFLOW TIMER PWM

  // DESACTIVA ENTRADAS ANALOGICAS
  ANSEL0 =  0;
```

```
ANSEL1bits.ANS8 = 0;

// ACTIVA ENTRADA ANALOGICA PARA LECTURA DE TELEMETRIA
ANSEL0bits.ANS5 = 1;

// INICIALIZACION DE VARIABLES
nvodato = 0;
Num_IntentosTxRx = 2;       // Intentos de comunicacion por red interna
CmdRecibido = 'N';                 // solo la interrupcion lo hace 'S'

TRISAbits.TRISA0 = 0;              // INICIA salidas para direccion de motor
TRISAbits.TRISA1 = 0;

TRISAbits.TRISA3 = 1;       // INICIA entradas para retroalimentacion del Motor
TRISAbits.TRISA4 = 1;
TRISAbits.TRISA5 = 1;       // Entrada de telemetria

PORTA = 0;          // limpia las salidas y entradas

TRISCbits.TRISC0 = 0;       // INICIA salidas para la direccion de corriente en BTM
TRISCbits.TRISC1 = 0;
TRISCbits.TRISC2 = 0;
TRISCbits.TRISC3 = 0;
TRISCbits.TRISC4 = 0;
TRISCbits.TRISC5 = 0;
PORTC = 0;

TRISDbits.TRISD4 = 1;       // ENTRADA de falla del motor
TRISDbits.TRISD2 = 0;       // salida de prueba
PORTDbits.RD2=0;

PORTB &= 0xC0;              // limpia las salidas

// INICIALIZACION DE PUERTO SERIAL
OpenUSART (  USART_TX_INT_OFF &           // apaga interrupcion por transmision
             USART_RX_INT_OFF &  // enciende interrrupcion por recepcion
             USART_ASYNCH_MODE &   // modo asincrono
             USART_EIGHT_BIT &     // trans/recep en 8 bit
             USART_CONT_RX &       // modo continuo de recepcion
             USART_BRGH_HIGH,      // ALTO intervalo de BAUD
             255  ) ;  // De tablas => 9600 bauds, XT = 40MHz    (255)

Habil_IntRx();
FallaMotorON();                    // habilita modo de fallo  catastrofico en el motor
Delay10KTCYx(50);

while (1)
{
   if (CmdRecibido=='S')
   {
     CmdRecibido='N';      // Reset a bandera
     Deshabil_IntRx();     // deshabilita la interrupcion por PUERTO SERIE
     WDT_ON();             // habilita el WatchDog

     chkext = 0;
     /*lee trama de comando, 12 bytes */
     for ( ctk=1; ctk<13; ctk++ )
     {
       chkext = chkext + cmdRx [ctk];
     }

     chkext=chkext & 0xFF;
     chkext=0-chkext;
     chkext=chkext & 0xFF;        /* CHEKSUM CALCULADO */

     if ( chkext == cmdRx [13] )   // verifica el CHECKSUM
     {
       chkext = 0;
       putchP_R ('K');     // envia ACKNOWLEDGE

       for ( i = 1; i<13;i++)      // Ejecucion de COMANDOS
```

```
        {

           switch (cmdRx [ i ])    {
             //     COMANDOS CONTROL LAZO ABIERTO
             case 0xA0:     if ( cmdRx[i+1] == 0) break;
                            WDT_OFF();
                            senoidal( cmdRx [i+1] );
                            checksum_calc();
                            WDT_ON();
                            break;
 // sale del bucle infinito y carga i = 0 y da el RESET a la bandera CmdRecibido

             case 0xA1:     if ( cmdRx[i+1] == 0) break;
                            WDT_OFF();
                            triangular(cmdRx [i+1]);
                            checksum_calc();
                            WDT_ON();
                            break;
// sale del bucle infinito y carga i = 0 y da el RESET a la bandera CmdRecibido

             case 0xA2:     if ( cmdRx[i+1] == 0) break;
                            WDT_OFF();
                            dientesierra(cmdRx [i+1]);
                            checksum_calc();
                            WDT_ON();
                            break;
// sale del bucle infinito y carga i = 0 y da el RESET a la bandera CmdRecibido


             //  COMANDOS BOBINAS
             case 0xB0:     aux_comando = cmdRx [i+1];    //BTMXFina
                            aux_comando = (aux_comando<<8)+cmdRx[i+2];
                            falla_motor();              // hay falla en el motor?
                            AbreBTMs();
                            BTMx_fina();
                            SetBobinaX(aux_comando);
                            i=i+2; break;
             case 0xB1:     aux_comando = cmdRx [i+1];    //BTMXFuerte
                            aux_comando = (aux_comando<<8)+cmdRx[i+2];
                            falla_motor();              // hay falla en el motor?
                            AbreBTMs();
                            BTMx_fuerte();
                            SetBobinaX(aux_comando);
                            i=i+2; break;
             case 0xB2:     aux_comando = cmdRx [i+1];           //BTMYFina
                            aux_comando = (aux_comando<<8)+cmdRx[i+2];
                            falla_motor();              // hay falla en el motor?
                            AbreBTMs();
                            BTMy_fina();
                            SetBobinaY(aux_comando);
                            i=i+2; break;
             case 0xB3:     aux_comando = cmdRx [i+1];    //BTMYFuerte
                            aux_comando = (aux_comando<<8)+cmdRx[i+2];
                            falla_motor();              // hay falla en el motor?
                            AbreBTMs();
                            BTMy_fuerte();
                            SetBobinaY(aux_comando);
                            i=i+2; break;
             case 0xB4:     aux_comando = cmdRx [i+1];    //BTMZFina
                            aux_comando = (aux_comando<<8)+cmdRx[i+2];
                            falla_motor();           // hay falla en el motor?
                            AbreBTMs();
                            BTMz_fina();
                            SetBobinaZ(aux_comando);
                            i=i+2; break;
             case 0xB5:     aux_comando = cmdRx [i+1];    //BTMZFuerte
                            aux_comando = (aux_comando<<8)+cmdRx[i+2];
                            falla_motor();              // hay falla en el motor?
                            AbreBTMs();
                            BTMz_fuerte();
                            SetBobinaZ(aux_comando);
```

```
                    i=i+2; break;
              case 0xB6: ApagaBTMx(); i++; break; // Apagado de BTMs
              case 0xB7: ApagaBTMy(); i++; break;
              case 0xB8: ApagaBTMz(); i++; break;
              case 0xB9: ApagaBobinas(); i++; break;

              //   COMANDOS MOTOR
              case 0xC0:     aux_comando = cmdRx [i+1];
                             aux_comando = (aux_comando<<8)+cmdRx[i+2];
                             AbreMotor();
                             if ( (cmdRx[i+1] & 0x80) == 0x80 )
                             {
                               PORTAbits.RA0 = 0;  // gira a la izquierda
                               PORTAbits.RA1 = 1;
                               sentido = 2;
                             }
                             else
                             {
                               PORTAbits.RA0 = 1;  // gira a la derecha
                               PORTAbits.RA1 = 0;
                               sentido = 1;
                             }
                             variables.mem_sentido = sentido;
                             SetDCMotor(aux_comando & 0x3FFF);
                             i=i+2; break;
              // Coloca al motor en una velocidad fija en RPMs
              case 0xC1:     aux_comando = cmdRx [i+1];
                             aux_comando = (aux_comando<<8)+cmdRx[i+2];
                             if ((aux_comando^0x80)==0) ApagaMotor();
                             WDT_OFF();
                             Velocidad(aux_comando);
                             WDT_ON();
                             i=i+2; break;
              // Cambia de velocidad al motor sin cambiar de sentido
              case 0xC2:     aux_comando = cmdRx [i+1];
                             aux_comando = (aux_comando<<8)+cmdRx[i+2];
                             if ((aux_comando^0x80)==0) ApagaMotor();
                             WDT_OFF();
                             CambVeloc(aux_comando);
                             WDT_ON();
                             i=i+2; break;
              case 0xC3:     ApagaMotor(); break;  //     Apaga el motor
              case 0xC4:     FrenaMotor(); break;  //  Frena el motor
              case 0xC5:     WDT_OFF();telemetria();WDT_ON(); break;
       //     Envia los datos de la corriente del motor
              case 0xC6:     FallaMotorON(); break;
       //     Enciende la condicion de falla del motor
              case 0xC7:     FallaMotorOFF(); break;
       //     Apaga la condicion de falla

              default: Nop(); break;}

       }

    }
    else
    {
      chkext = 0;
      putchP_R ('N');        // envia NO ACKNOWLEDGE
    }
  }
  else putchP_R ('N');       // envia NO ACKNOWLEDGE

  Habil_IntRx();             // habilita de nuevo la interrupcion por Rx
  WDT_OFF();          // Apaga WatchDog
  Idle();             // Modo Idle para apagar CPU y dejar encendidos los perifericos
  Sleep();            // Duerme al CPU del MCU
  }
}
```

```
/********************** DATOS ******************************/

void WDT_ON (void)
{
       WDTCONbits.SWDTEN = 1;        // enciende Watch Dog Timer
       ClrWdt();      // limpia el timer del WATCHDOG
}

void WDT_OFF (void)
{
       WDTCONbits.SWDTEN = 0;
}


char Lee1b_Redint_PoR(void)
{

       r=0;
       TMR0H = 0;                    // borra timer0
       TMR0L = 0;
       T0CON = 0x83;                 // corre el timer0 en 16 bit, prescalador 1:256
                                     // espera un comando por un tiempo de 1.68 seg

       while( ! ( PIR1bits.RCIF | INTCONbits.TMR0IF) );      /*    espera dato por un
tiempo "p"*/

       if ( PIR1bits.RCIF )
              {
                     nvodato = RCREG;              /* si hubo dato red interna */
                     r=1;
              }
       else r=0;     // no hubo dato pero ocurrio RETRASO, muchos llamados se hacen a esta
rutina solo por este RETRASO

       INTCONbits.TMR0IF = 0;        // reset a bandera
       T0CONbits.TMR0ON = 0;         // detiene el timer
       return(r);    // regresa "r"
}


char putchP_R(char c)    /* Transmite por red interna PRINCIPAL O REDUNDANTE */
{
       Nop();
       WriteUSART ( c );             // Transmite caracter
       while ( BusyUSART() );        /* transmit interrupt request flag      */
       Nop();
       return ( c );
}

void checksum_calc (void)
{
       unsigned char borra;

       for ( ctk=1; ctk<13; ctk++ )             //lee trama de comando, 12 bytes
       {
              chkext = chkext + cmdRx [ctk];
       }
       chkext=chkext & 0xFF;
       chkext=0-chkext;
       chkext=chkext & 0xFF;                     // CHEKSUM CALCULADO
       if ( chkext == cmdRx [13] )
       {
              chkext = 0;
              putchP_R('K');
              i = 0;
       }
       else
       {
              chkext = 0;
              putchP_R('N');
              i = 20;
```

```
        }
        PORTA = 0;
        borra = RCREG;        // lee el registro para borrar la bandera
}

void rellenaCOM(char fill)
{
        char cuenta1;
        for (cuenta1 = 1; cuenta1 <= fill; cuenta1++)
        {
                cmdTx[12 - fill + cuenta1] = 0;
        }
}

void esc_cmd_redint_PoR (void)
{
        char Num_pasos,cuenta;
        char ack;

        union {        int si;
                       int difck;
                       int ctk;        } aux;

        unsigned int resp2=0;
        unsigned int resp1=0;

        Deshabil_IntActualRI();  /*Deshabilita la interrup. por canal principal en la
RedInt*/

        Delay10KTCYx(3);
        Num_pasos=0;

        /************* CALCULO DE CHECKSUM *****************/

        cmdTx[0]= 0x2F;       // byte de despertar

        for( aux.si=1;aux.si<13;aux.si++)    /* para cheksum se excluye despertar [0] y se
excluye el casillero para cheksum [13] */
        {
                resp1=resp1+ cmdTx[aux.si];   /*suma partes de comando*/
        }
        resp1= resp1 & 0x00FF;
        resp2 = 0 - (unsigned long)resp1;
        resp2 = resp2 & 0x0FF;
        aux.difck = resp2;
        cmdTx [13] = resp2;

        /************* CHECKSUM calculado ******************/

                       TMR0H = 0;     // borra timer0
                       TMR0L = 0;
                       T0CON = 0x83;  // corre el timer0 en 16 bit, prescalador 1:256
                       // espera un comando por un tiempo de 1.68 seg y marca la bandera

                       Habilita_IntActualRI();       // habilita la interrupcion por Rx
                       INTCONbits.TMR0IF = 0;        // baja bandera
                       while( !INTCONbits.TMR0IF )
                       {
                              if (RCREG == 'E')
                              {
                                     T0CONbits.TMR0ON = 0;         // detiene el timer
                                     INTCONbits.TMR0IF = 0;// baja la bandera
                                     break;  // espera un tiempo la requisicion de envio
                              }
                       }


                       Deshabil_IntActualRI();  // deshabilita la interrupcion para evitar
interferencias
```

```
                        cuenta=0;        /* transmite un maximo de Num_IntentosTxRx o hasta
recibir "K" */
                        ack = 'N';

                        do {
                                        if (INTCONbits.TMR0IF == 1)   break;
        // sale de la rutina, sin hacer nada

                                        nvodato=0;
                                        for (ctk=0; ctk<14; ctk++)           /* envio incluye
Despertar y cheksum */
                                        {
                                                putchP_R(cmdTx[ctk] & 0xFF);
                                                /* ENVIA COMANDO */
                                        }

        /*************************************************************/
        /*lee ACKNOWLEDGE si es que hay, y lo guarda en 'nvodato' : */
        /*************************************************************/
                                        // habilita la interrupcion por RX recibe 'K' o 'N')
                                                Habilita_IntActualRI();
                                                while( 1 )
                                                {
                                                        if (RCREG == 'K' || RCREG == 'N')
                                                        {
                                                                nvodato=RCREG; break;
                                                        }
                                                }
                                        // deshabilita la interrupcion para evitar interferencias
                                                Deshabil_IntActualRI();
                                                if (nvodato=='K')
                                                {
        /* Habilita interrupt canal princ. de Red Interna*/
                                                        ack='S';
                                        // Se recibio la lectura de sensores
                                                }
                                                else
                                                {
                // Si no hubo respuesta o la respuesta fue "no ack"
                                                        ack='N';
        // No se recibieron las lecturas de los sensores y lo volvera a intentar
                                                }
                                                cuenta++;

                                        Delay10KTCYx(3);             // delay de 3ms

/*****************************************************************************/
/* retransmite hasta tener acK o un maximo de Num_IntentosTxRx veces antes de conmutar
   de canal         */
/*****************************************************************************/
                } while ( (cuenta < Num_IntentosTxRx) && ack=='N'); // nuevo intento si NO se
reciben y cuenta es menor que el numero de intentos permitidos


        if (ack=='S')  Fallo_TOTAL_RedInt='N';              // se han enviado las lecturas
correctamente
        else Fallo_TOTAL_RedInt='S';                        // no se pudo!

        /*si no ocurrio ACK => que red interna princ. y redund. no operan*/

}

/************************    CONTROL LAZO ABIERTO ***************************/

// SENO
// Se cargan los valores del seno previamente cargados en una tabla en memoria dato

void senoidal (unsigned char frec)
{
        int i_seno=0;
        int seno_aux;
```

```
	ANSEL0 = 0;
	AbreMotor();

	TRISAbits.TRISA0 = 0; // establece salidas para designar sentido
	TRISAbits.TRISA1 = 0;

	PORTA = 0;		//limpia puerto A
	PORTAbits.RA0 = 1;		// gira derecha
	Habil_IntRx();		// habilita interrupcion por recepcion

	while (1)
	{
		for (i_seno=0; i_seno < 200; i_seno++)
		{
			seno_aux=t_seno[i_seno]>>1;
			// Carga los datos de la tabla de senos sucesivamente
			SetDCMotor( seno_aux );
			Delay1KTCYx(frec);
			Delay1KTCYx(25);		// retraso de ajuste
			Delay10TCYx(10);
		}
		if (CmdRecibido == 'S') break;
		PORTA ^= 0x03;		// cambio de direccion
	}

	Deshabil_IntRx();		// deshabilita interrupcion por recepcion
	SetDCMotor( 0 );		// detiene el motor
	PORTA = 0;
}


void triangular (unsigned char frec)
{
	int i_triang=0;
	PORTA = 0;		//limpia puerto A
	AbreMotor();
	PORTAbits.RA0 = 1;			// gira derecha
	Habil_IntRx();				// habilita interrupcion por recepcion
	while(1)
	{
		while ( i_triang < 8191 )		// señal triangular amptotal = 16383
		{
			SetDCMotor(i_triang);
			Delay1KTCYx(frec);
			Delay100TCYx(55);		// retraso de ajuste , con
			Delay10TCYx(9);
			i_triang+=41;		// amptotal = 81
		}
		if (CmdRecibido == 'S') break;
		while ( i_triang > 0)
		{
			SetDCMotor(i_triang);
			Delay1KTCYx(frec);
			Delay100TCYx(55);		// retraso de ajuste
			Delay10TCYx(9);
			i_triang=i_triang-41;
		}
		if (CmdRecibido == 'S') break;
		PORTA ^= 0x03;		// cambio de direccion
	}
	Deshabil_IntRx();		// deshabilita interrupcion por recepcion
	SetDCMotor( 0 );		// detiene el motor
	PORTA = 0;
}

void dientesierra (unsigned char frec)
{
	int i_sierra;
	PORTA = 0;		//limpia puerto A
	AbreMotor();
	PORTAbits.RA0 = 1;			// gira derecha
```

```
        PIE1bits.RCIE = 1;           // habilita interrupcion por recepcion
        i_sierra = 0;
        while(1)
        {

                while ( i_sierra < 8191 )           // señal triangular
                {
                        SetDCMotor(i_sierra);
                        Delay1KTCYx(frec);
                        Delay100TCYx(55);           // retraso de ajuste
                        Delay10TCYx(9);
                        i_sierra+=41;
                }
                if (CmdRecibido == 'S') break;
                i_sierra=16383;
                PORTA ^= 0x03;        // cambio de direccion

                while ( i_sierra > 0)
                {
                        SetDCMotor(i_sierra);
                        Delay1KTCYx(frec);
                        Delay100TCYx(55);           // retraso de ajuste
                        Delay10TCYx(9);
                        i_sierra = i_sierra - 41;
                }
                if (CmdRecibido == 'S') break;
                PORTA ^= 0x03;        // cambio de direccion

        }
        Deshabil_IntRx();            // deshabilita interrupcion por recepcion
        SetDCMotor( 0 );             // detiene el motor
        PORTA = 0;
}


/*********************MOTOR *********************************/

// Inicializacion del modulo de PWM PIC18FXX31
// El MCU contiene hasta 8 canales de salida de PWM
// en el 4331 y 4431 mientras en el 2331 y 2431 hay solo 6
// SINTAXIS
//
//     AbreMotor (int periodo);
//     periodo => numero 0x0000 a  0x3FFF
void AbreMotor(void)
{

        PTCON0 = 0x00;               // postscale =1:1, prescale=1:1, modo free-running
(alinea el PWM en el borde)
        PTCON1 = 0x80;        // base de tiempo encendida, cuenta hacia arriba ('C' cuenta
hacia abajo)

        PWMCON0 = 0x5F;       // se pone en modo independiente todas las salidas de PWM y se
habilita solo PWM 0,1

        PWMCON1 = 0x00;       // Post escala = 1:1, actualizacion del CT y buffer de
periodo, base de tiempo sincronizada con la base de tiempo de pwm

        DTCON = 0x00;         // tiempo muerto = 0


        OVDCONDbits.POVD0 = 1;       // la salida de PWM es controlada por el valor en el
reg de CT  y la base tiempo de PWM
        OVDCONDbits.POVD1 = 1;
        OVDCONS = 0x00;       // la salida de PWM esta INACTIVA cuando el correspondiente
bit en OVDCONS se borra junto con el bit en OVDCOND

        // PERIODO  => ( CD = 14 bits )

        PTPERL = 0xFF; // asigna valor al reg de periodo bajo
        PTPERH = 0x3F; // asigna valor al reg de periodo alto
```

```
}


// PIC18FXX31
// Puesta del ciclo de trabajo en el modulo de PWM
// el limite del numero entero que se puede colocar en esta funcion es de 14 bits
// de 0 a 16383, si el entero es mas grande los bits que se encuentren mas arriba
// seran descartados
// SINTAXIS
//
//            SetDCPWM1(int ciclotrabajo);
//            ciclotrabajo => numero que define el ancho de pulso de PWM de hasta
//                                          14 bit (0 - 14,383)

void SetDCMotor(unsigned int ciclotrabajo)
{
       union pwmct ctrabajo;
       ctrabajo.lpwm = ciclotrabajo;               // obtenemos los MSB de ciclo de trabajo
       PDC0L = ctrabajo.lpwm;//  asignamos el valor al registro de asignacion de CT bajo
canal 1
       ctrabajo.hpwm = (ciclotrabajo >>8) & 0x3F;    // obtenemos los LSB de ciclo de
trabajo
       PDC0H = ctrabajo.hpwm;   // asignamos el valor al registro de asignaacion de CT alto
canal 1

       while(!PIR3bits.PTIF);       // Espera a que el timer termine de contar para colocar
el nuevo valor de CT
       PIR3bits.PTIF = 0;           // baja bandera

}

// ALGORITMO DE VELOCIDAD
// Control PI de la velocidad del motor
// Fijando el tiempo de muestreo (ts) en 100ms a traves del TIMER 0
// usando el postscaler 1:16 => 0xF424 (c. max.)
// precargar 0xBDB en el TIMER ya que no tiene registro de periodo

void Velocidad (unsigned int vel)
{

       TRISAbits.TRISA0=0;   // asigna salidas para la direccion
       TRISAbits.TRISA1=0;
       TRISAbits.TRISA2=1;          // asigna la entrada al sensor

///////INICIANDO VARIABLES DE CONTROL/////////////////
       medicion = 0;
       vel_deseada = 0;
       proporcional = 0;
       integral = 0;
       salida_pi = 0;
       velocidad = 0;
////////////////////////////////////////////////////

       alto = 0;             // bit de alto del ALGORITMO
       timeup = 0;

       sent = vel&0x8000;                  // filtramos el sentido
       if (sent == 0)        sentido = 1;
       if (sent == 0x8000)   sentido = 2;

       if ( mem_sentido != sentido && mem_sentido != 0 )   // comprueba el sentido y ...
       {
                     PORTAbits.RA0 = 0;            // hay un tiempo de espera en caso de
que haya un cambio de sentido
                     PORTAbits.RA1 = 0;
                     Delay10KTCYx(255);
                     Delay10KTCYx(255);
                     Delay10KTCYx(255);
                     Delay10KTCYx(255);
                     Delay10KTCYx(255);
```

```
                Delay10KTCYx(255);
        }

        switch (sent)         // asigna el sentido
        {
               case 0x0000:   PORTAbits.RA0 = 0;    // gira a la derecha
                              PORTAbits.RA1 = 1;
                              break;
               case 0x8000:   PORTAbits.RA0 = 1;    // gira a la izquierda
                              PORTAbits.RA1 = 0;
                              break;
         }

        mem_sentido = sentido;// despues de comprobar el sentido guarda el dato de sentido

        velocidad = 0x7FFF&vel;                              // quita los bits de sentido
        AbreMotor();                 // se fija la frecuencia del CT

        DFLTCON = 0x1E;              // filtro digital en relacion 1:4 entradas 1 y 2
        T5CONbits.T5SEN = 1;  // HABILITA DURANTE SLEEP
        T5CONbits.RESEN = 1;  // habilita el reset especial
        T5CONbits.T5MOD = 0;  // modo de conteo continuo
        if (velocidad < velocidad_minima)
        {
               T5CONbits.T5PS1 = 1;  // preescalador 1:8
               T5CONbits.T5PS0 = 1;
        }
        else
        {
               T5CONbits.T5PS1 = 0;  // preescalador 1:1
               T5CONbits.T5PS0 = 0;
        }
        T5CONbits.T5SYNC = 0; // se ignora este bit
        T5CONbits.TMR5CS = 0; // usa el reloj interno
        T5CONbits.TMR5ON = 1; // habilita timer5
        CAP2CON = entrada_captura;    // modo de medir periodo y reset al timer al terminar


        T0CON = 0x07;         // TIMER0 postscaler 1:256
        TMR0H = 0;
        TMR0L = 0;

               ////////////////////////////////////////////
        //                          CONTROL                                 //
               ////////////////////////////////////////////

//      (1) Fijar el valor de Kp, Ki
        kp = p_control;
        ki = i_control;

//      (2) Fijar los valores iniciales de error,a_error y ts
        error = 11;
        a_error = 0;

        /* Tiempo de muestreo (ts) 50 ms   */
        T1CON = 0x30;         // TMR0 prescaler 1:8, RW el registro TMR1 en 16 bits
        TMR1H = t_muestreo1;
        TMR1L = t_muestreo2;

        PIR1bits.TMR1IF = 0;
        ////////////////////////////////////

//      (3) Calculo de la velocidad
        medida_sensor = 0;
        CAP2BUFH = 0;
        CAP2BUFL = 0;
//      incertidumbre = 15;
        vel_deseada = periodo_vel / velocidad;
        // conversion de rps a vel (2 imanes => se dividio / 2 los 10,000,000)
        // para rpm se multiplica por 60  => [ (10e6)*60 ] / (# pulsos por vuelta)
        // para 16 ppv => 37,500,000 en rpms
```

```
        if ( velocidad < velocidad_minima )
        {
                vel_deseada /= 8;
        }

        T0CONbits.TMR0ON = 1;         // empieza la validacion del ALGORITMO

//      (4) COMIENZA CONTROL
        while (1)
        {
                T1CONbits.TMR1ON = 1; // arranca tiempo de muestreo
                // bit de prueba REG curva de resp de control en lazo cerrado
                PORTDbits.RD2 = 1;
                error = vel_deseada - medicion;             // calculo del error

                if ( error <= delta_error && error >= -delta_error )       break;
// verifica el error, si esta dentro del rango aceptado, ya NO toma acciones de control

                // error de + - 1rpm
                else
                {

                        proporcional=kp*error/kp_dec; // calculo del termino proporcional
                        integral=ki*a_error/ki_dec;   // calculo del termino integral

                        a_error += error;             // suma el error
                        salida_pi = (proporcional + integral) / 512;
                        // se obtiene la salida final y se le da un factor de escala

                        if ( salida_pi < 0 )
                        {
                                salida_pi = 0xFFFF - salida_pi + 1;
                                // realiza el complemento a 1
                        }

                        SetDCMotor (salida_pi);              // coloca el ciclo de trabajo
                }

                // da un limite superior e inferior para evitar el 'windup'
                if (a_error > SatMax)         a_error = 8388000;
                else if (a_error < SatMin)    a_error = -8388000;

                if (alto == 1) break; // sobrepasa 3.3 seg, no lo logra estabilizar

                while (!PIR1bits.TMR1IF); //espera a que acabe el ts para hacer otro muestreo
                PIR1bits.TMR1IF = 0;

                ////    Recarga tiempo de muestreo   ////
                T1CON = 0x30;         // TMR1 preescaler  1:8
                TMR1H = t_muestreo1;
                TMR1L = t_muestreo2;
        }

        if (alto == 1)        putchP_R('N');
        else   putchP_R('V');

        CAP2CON = 0;   // deshabilita los registros para evitar interrupciones en falso
        T5CON = 0;
        T1CON = 0;
        T0CON = 0;
        PORTDbits.RD2 = 0;    // apaga bits de prueba
}

void CambVeloc (unsigned int vel)
{
  if (PORTAbits.RA0^PORTAbits.RA1)
  {
///////INICIANDO VARIABLES DE CONTROL/////////////////
        vel_deseada = 0;
        salida_pi = 0;
//////////////////////////////////////////////////////
```

```
        AbreMotor();                                    // se abre el canal del motor
        velocidad = 0x7FFF&vel;                         // quita los bits de sentido

        T5CONbits.T5SEN = 1;          // HABILITA DURANTE SLEEP
        T5CONbits.RESEN = 1;  // habilita el reset especial
        T5CONbits.T5MOD = 0;  // modo de conteo continuo
        if (velocidad < velocidad_minima)
        {
                T5CONbits.T5PS1 = 1;  // preescalador 1:8
                T5CONbits.T5PS0 = 1;
        }
        else
        {
                T5CONbits.T5PS1 = 0;  // preescalador 1:1
                T5CONbits.T5PS0 = 0;
        }
        T5CONbits.T5SYNC = 0; // se ignora este bit
        T5CONbits.TMR5CS = 0; // usa el reloj interno
        T5CONbits.TMR5ON = 1; // habilita timer5
        CAP2CON = entrada_captura;    // modo de medir periodo y reset al timer al terminar

        T0CON = 0x07;         // TIMER0 postscaler 1:256
        TMR0H = 0;
        TMR0L = 0;
        INTCONbits.TMR0IF = 0;
        alto = 0;             // bit de alto del ALGORITMO
        timeup = 0;

                    ////////////////////////////////////////////
                    //             CONTROL                    //
                    ////////////////////////////////////////////

//      (2) Fijar los valores iniciales de error,a_error y ts
        error = 11;

        /* Tiempo de muestreo (ts) 10 ms   */
        T1CON = 0x30;
        TMR1H = t_muestreo1;
        TMR1L = t_muestreo2;
        PIR1bits.TMR1IF = 0;          // baja bandera de interrupcion

        ////////////////////////////////////

        medida_sensor = 0;
// conversion de rps a vel (2 imanes => se dividio / 2 los 10,000,000)
// para rpm se multiplica por 60  => [ (10e6)*60 ] / (#imanes)
        vel_deseada = periodo_vel / velocidad;

        if ( velocidad < velocidad_minima )
        {
                vel_deseada /= 8;
        }

        T0CONbits.TMR0ON = 1;         // empieza la validacion del ALGORITMO

        while ( 1 )
        {
                T1CONbits.TMR1ON = 1;         // arranca tiempo de muestreo
                PORTDbits.RD2 = 1;                    // bit de prueba
                error = vel_deseada - medicion;                     // calculo del error

                if ( error <= delta_error && error >= -delta_error )       break;
// verifica el error, si esta dentro del rango aceptado, ya NO toma acciones de control
                else
                            // error de + - 1rpm
                {

                        proporcional=kp*error/kp_dec;  // calculo del termino proporcional
                        integral=ki*a_error/ki_dec;    // calculo del termino integral
```

```
                        a_error += error;            // suma el error

                        // se obtiene la salida final y se le da un factor de escala
                        salida_pi = (proporcional + integral) / 512;

                        if ( salida_pi < 0 )
                        {
                                // realiza el complemento a 1
                                salida_pi = 0xFFFF - salida_pi + 1;
                        }

                        SetDCMotor (salida_pi);      // coloca el ciclo de trabajo
                }

                // da un limite superior e inferior para evitar el 'windup'
                if (a_error > SatMax)        a_error = 8388000;
                else if (a_error < SatMin)   a_error = -8388000;

                if (alto == 1) break;

                while (!PIR1bits.TMR1IF);// espera a que acabe el ts para hacer otro muestreo
                PIR1bits.TMR1IF = 0;

                ////   Recarga tiempo de muestreo   ////
                T1CON = 0x30;
                TMR1H = t_muestreo1;
                TMR1L = t_muestreo2;
        }

        if (alto == 1)        putchP_R('N');
        else   putchP_R('V');

        CAP2CON = 0;   // deshabilita los registros para evitar interrupciones en falso
        T5CON = 0;
        T1CON = 0;
        T0CON = 0;
        PORTDbits.RD2=0;      // bit de prueba
  }
}

void ApagaMotor ( void )
{
        PORTAbits.RA0 = 0;    // en cero la direccion
        PORTAbits.RA1 = 0;
        PDC0H = 0;                    // en cero el CT
        PDC0L = 0;
        while(!PIR3bits.PTIF);        // Espera a que el timer termine de contar para colocar
el nuevo valor de CT
        PIR3bits.PTIF = 0;                    // baja bandera
        OVDCONS = 0;                          // deshabilitada la salida de PWM
        OVDCOND &= 0xFC;
        FLTCONFIGbits.FLTAS  = 0;     // baja la bandera de la condicion de falla
}

void FrenaMotor ( void )
{
        PORTAbits.RA0 = 1;    // segun el driver 1 y 1 para frenar
        PORTAbits.RA1 = 1;
        PDC0H = 0;                    // en cero el CT
        PDC0L = 0;
        OVDCONS = 0;          // deshabilitada la salida de PWM
        OVDCOND &= 0xFC;
        FLTCONFIGbits.FLTAS  = 0;     // baja la bandera de la condicion de falla
}

void telemetria (void)
{
        // ABRE y CONFIGURA CONVERTIDOR A/D
        ADCON0 = 0b00000100;  // single shot, Grupo B (1,5)
        ADCON1 = 0b00000000;  // sin activar FIFO buffer
```

```
        ADCON2 = 0b10110010;    // justificado a la derecha, 12 TAD (retardo de conversion),
Fosc/32
        ADCON3 = 0b00000000;  // disparadores de conversion desactivados
        ADCHS = 0b00010000;           // selecciona el canal 5

        ADCON0bits.ADON = 1;  // enciende el convertidor
        Delay10TCYx(10);              // retardo de 10us
        ConvertADC();                 // convierte
        while( BusyADC() );           // espera la conversion
        Nop();
        cmdTx[1] = ADRESH;    // lee los registros de conversion y los coloca en el comando
a enviar
        cmdTx[2] = ADRESL;
        CloseADC();                         // cierra el convertidor

        rellenaCOM(10);                     // rellena con 0
        esc_cmd_redint_PoR();   // envia el comando haciendo su checksum y CHARLA
}

// Enciende la condicion de falla del motor
void FallaMotorON(void)
{
        FLTCONFIGbits.FLTCON = 1;                   // habilita modo de fallo  catastrofico
en el motor
        FLTCONFIGbits.FLTAMOD = 0;                  // hasta que el bit FLTAS sea borrado en
software
        FLTCONFIGbits.FLTAEN = 1;                   // habilita condicion de falla en A
        FLTCONFIGbits.FLTAS  = 0;                   // baja la bandera de falla
}

// Apaga la condicion de falla del motor
void FallaMotorOFF(void)
{
        FLTCONFIGbits.FLTAEN = 0;           // deshabilita condicion de falla en A
        FLTCONFIGbits.FLTAS  = 0;
        FLTCONFIGbits.FLTAMOD = 0;    // hasta que el bit FLTAS sea borrado en software
}

/******************   BOBINAS DE TORQUE MAGNETICO ***************************/

void AbreBTMs(void)
{
        PTCON0 = 0x00;        // (BIEN)postscale =1:1, prescale=1:1, modo free-running
(alinea el PWM en el borde)
        PTCON1 = 0x80;        // (BIEN)base de tiempo encendida, cuenta hacia arriba ('0xC0'
cuenta hacia abajo)

        PWMCON0 = 0x5F; // PWMx = ON ,se pone en modo independiente todas las salidas de PWM
        PWMCON1 = 0x01;       //(BIEN) base de tiempo asincrona a la base de tiempo de PWM

        // PERIODO DE LA SEÑAL PWM (BIEN)

        PTPERL = 0xFF;        //asigna valor al reg de periodo bajo
        PTPERH = 0x0F;        //asigna valor al reg de periodo alto
}

void  BTMx_fina (void)              // APAGA bobina fuerte y ENCIENDE la fina
{
        OVDCONSbits.POUT4 = 0;       // Estado del pin de salida en el canal PWM2 y 3
        OVDCONSbits.POUT5 = 0;
        OVDCONDbits.POVD5 = 0;       // Control del pin de salida mediante el generador de
PWM o registro OVDCONS
        OVDCONDbits.POVD4 = 1;
}

void BTMx_fuerte (void)             //APAGA bobina fine y ENCIENDE la fuerte
{
        OVDCONSbits.POUT4 = 0;
        OVDCONSbits.POUT5 = 0;
        OVDCONDbits.POVD4 = 0;
        OVDCONDbits.POVD5 = 1;
```

```
}

void  BTMy_fina (void)
{
       OVDCONSbits.POUT2 = 0;
       OVDCONSbits.POUT3 = 0;
       OVDCONDbits.POVD3 = 0;
       OVDCONDbits.POVD2 = 1;
}

void BTMy_fuerte (void)
{
       OVDCONSbits.POUT2 = 0;
       OVDCONSbits.POUT3 = 0;
       OVDCONDbits.POVD2 = 0;
       OVDCONDbits.POVD3 = 1;
}

void  BTMz_fina (void)
{
       OVDCONSbits.POUT6 = 0;
       OVDCONSbits.POUT7 = 0;
       OVDCONDbits.POVD7 = 0;
       OVDCONDbits.POVD6 = 1;
}

void BTMz_fuerte (void)
{
       OVDCONSbits.POUT6 = 0;
       OVDCONSbits.POUT7 = 0;
       OVDCONDbits.POVD6 = 0;
       OVDCONDbits.POVD7 = 1;
}

void SetBobinaX(unsigned int ciclotrabajo_dircorr)
{
       TRISCbits.TRISC0=0;   // asigna salidas para la direccion
       TRISCbits.TRISC1=0;

       //     Designa el sentido de la corriente
       switch ( ciclotrabajo_dircorr & 0xC000 )
       {
              case  0x4000:  PORTCbits.RC0 = 1;  // corriente izq
                             PORTCbits.RC1 = 0;
                             break;

              case  0x8000:  PORTCbits.RC0 = 0;  // corriente der
                             PORTCbits.RC1 = 1;
                             break;
        }

       PDC2L = ciclotrabajo_dircorr & 0xFF;                      // obtenemos los LSB de
ciclo de trabajo y asignamos el valor al registro de asignacion de CT bajo canal 1
       PDC2H = ( ciclotrabajo_dircorr >>8 ) & 0x3F;    // obtenemos el MSB de ciclo de
trabajoctrabajo.hpwm y asignamos el valor al registro de asignacion de CT alto canal 1

       while(!PIR3bits.PTIF);       // Espera a que el timer termine de contar para colocar
el nuevo valor de CT
       PIR3bits.PTIF = 0;           // baja bandera
}

void SetBobinaY(unsigned int ciclotrabajo_dircorr)
{
       //     Designa el sentido de la corriente
       switch (ciclotrabajo_dircorr & 0xC000)
       {
              case  0x4000:  PORTCbits.RC2 = 1;  // corriente izq
                             PORTCbits.RC3 = 0;
                             break;
              case  0x8000:  PORTCbits.RC2 = 0;  // corriente der
                             PORTCbits.RC3 = 1;
```

```
                        break;
        }

       PDC1L = ciclotrabajo_dircorr & 0xFF;                    // obtenemos los LSB de
ciclo de trabajo y asignamos el valor al registro de asignacion de CT bajo canal 2
       PDC1H = ( ciclotrabajo_dircorr >>8 ) & 0x3F;    // obtenemos el MSB de ciclo de
trabajoctrabajo.hpwm y asignamos el valor al registro de asignacion de CT alto canal 2

       while(!PIR3bits.PTIF);       // Espera a que el timer termine de contar para colocar
el nuevo valor de CT
       PIR3bits.PTIF = 0;           // baja bandera
}


void SetBobinaZ(unsigned int ciclotrabajo_dircorr)
{
       //     Designa el sentido de la corriente
       switch (ciclotrabajo_dircorr & 0xC000)
       {
              case  0x4000:  PORTCbits.RC4 = 1;  // corriente izq
                             PORTCbits.RC5 = 0;
                             break;
              case  0x8000:  PORTCbits.RC4 = 0;  // corriente der
                             PORTCbits.RC5 = 1;
                             break;
        }

       PDC3L = ciclotrabajo_dircorr & 0xFF;                    // obtenemos los LSB de
ciclo de trabajo y asignamos el valor al registro de asignacion de CT bajo canal 3
       PDC3H = ( ciclotrabajo_dircorr >>8 ) & 0x3F;    // obtenemos el MSB de ciclo de
trabajoctrabajo.hpwm y asignamos el valor al registro de asignacion de CT alto canal 3

       while(!PIR3bits.PTIF);       // Espera a que el timer termine de contar para colocar
el nuevo valor de CT
       PIR3bits.PTIF = 0;           // baja bandera
}

//     BobinaX
void ApagaBTMx(void)
{
       PDC2L = 0x00;
       PDC2H = 0x00;
       while(!PIR3bits.PTIF);       // Espera a que el timer termine de contar para colocar
el nuevo valor de CT
       PIR3bits.PTIF = 0;           // baja bandera
       PORTCbits.RC0=0;
       PORTCbits.RC1=0;
       OVDCONS = 0;          // Estado del pin de salida en el canal PWM2 y 3
       OVDCOND &= 0xCF;      // Control del pin de salida mediante el generador de PWM(1) o
registro OVDCONS(0)
}

// BobinaY
void ApagaBTMy(void)
{
       PDC1L = 0x00;
       PDC1H = 0x00;
       while(!PIR3bits.PTIF);       // Espera a que el timer termine de contar para colocar
el nuevo valor de CT
       PIR3bits.PTIF = 0;           // baja bandera
       PORTCbits.RC2=0;
       PORTCbits.RC3=0;
       OVDCONS = 0;
       OVDCOND &= 0xF3;
}

// Bobina Z
void ApagaBTMz(void)
{
       PDC3L = 0x00;
       PDC3H = 0x00;
```

```
// Espera a que el timer termine de contar para colocar el nuevo valor de CT
        while(!PIR3bits.PTIF);
        PIR3bits.PTIF = 0;           // baja bandera
        PORTCbits.RC4=0;
        PORTCbits.RC5=0;
        OVDCONS = 0;
        OVDCOND &= 0x3F;
}

// Todas las Bobinas
void ApagaBobinas(void)
{
        PORTCbits.RC0=0;             // Apaga bits de sentido de la corriente
        PORTCbits.RC1=0;
        PORTCbits.RC2=0;
        PORTCbits.RC3=0;
        PORTCbits.RC4=0;
        PORTCbits.RC5=0;
        PDC1L = 0x00;
        PDC1H = 0x00;
        PDC2L = 0x00;
        PDC2H = 0x00;
        PDC3L = 0x00;
        PDC3H = 0x00;
        // Espera a que el timer termine de contar para colocar el nuevo valor de CT
        while(!PIR3bits.PTIF);
        PIR3bits.PTIF = 0;           // baja bandera
        OVDCONS = 0;
        OVDCOND &= 0x03;
}

void falla_motor (void)
{
        if (FLTCONFIGbits.FLTAS  == 1)     // hay falla en el motor??
        {
                ApagaMotor();         // apaga el motor
        }
}

/******************* INTERRUPCIONES *********************************/

#pragma code rx_interrupt = 0x08                          // vector de interrupcion
void rx_int ( void )
{
        _asm GOTO maneja_IntRx _endasm
}
#pragma code

#pragma interrupt maneja_IntRx                            // rutina de interrupcion
void maneja_IntRx (void)
{
/*                      Define si hay dato en el canal principal              */
        nvodato = 0;
        hay_dato = 0;
        hay_dato = Lee1b_Redint_PoR();
        cmdRx [0] = nvodato;                  // byte de  despertar = cmdRx[0]
        if ( cmdRx [0]== 0x2F )       // Verifica la cabecera del comando (se pierde)
        {
                for (ctk=1; ctk<=13; ctk++) // lee trama de comando, 14 bytes
                {
        // Verifica POR POLEO si llega algun dato al puerto serie
                        hay_dato = Lee1b_Redint_PoR();
                        if ( hay_dato ) // Mientras haya algun dato en el canal
                        {
                                cmdRx [ctk] = nvodato;        // almacena comando
                        }
                        else  ctk=20;  // Si no hay dato obliga a salir del for
                }

                if (ctk==20)
                {
```

```
                    CmdRecibido = 'N';
                    Habil_IntRx();
             }
             else  CmdRecibido='S';
      }
 }

////////////////INTERRUPCION BAJA PRIORIDAD/////////////////
// las interrupciones son producidas por overflow en el TIMER 5 y al producirse una captura
en el
// la entrada del modulo de captura de movimiento CAP2

#pragma code interrupcion_baja = 0x18
void flanco_int (void)
{
      _asm  GOTO overflow  _endasm
}
#pragma code

#pragma interruptlow overflow
void overflow (void)
{
      if (INTCONbits.TMR0IF == 1)
      {
             timeup += 1;          // se acaba el tiempo de estabilizacion
             if (timeup > t_asentamiento)  alto = 1;
             INTCONbits.TMR0IF = 0;
      }
      if (PIR3bits.TMR5IF == 1)
      {      medida_sensor += 0xFFFF;      // acumula conteo, hasta detectar el siguiente
flanco
             PIR3bits.TMR5IF = 0;  // baja la bandera
      }
      else
      // espera que haya una bandera en el registro, para acumular la medicion restante
      {      medicion = CAP2BUFH;  // toma el valor del bufer de captura 1 y ...
             medicion = (medicion<<8)|CAP2BUFL;
             // lo suma al acumulador auxiliar para lograr un conteo mayor
             medicion += medida_sensor;
             medida_sensor = 0;           // Reset al registro auxiliar de conteo
             PIR3bits.IC2QEIF = 0;
      }
}
```

Programa en el MCU de sensores

```
/****************** Software Control del Modulo de Sensores ***********************/

// Este software interfacea a la computadora de vuelo con el modulo de control encargado
// del control y el intercambio de informaccion entre ambos modulos

// NOTA: SE HIZO UN CAMBIO EN EL SOFTWARE INTERCAMBIANDO ACELEROMETROY Y ACELEROMETROZ
//              DEBIDO A ERRORES DE RUTEO EN LA TARJETA SATED1-S31

// Librerias de inicio
#include <p18f2520.h>
#include <usart.h>
#include <delays.h>
#include <adc.h>
#include <spi.h>
#include <i2c.h>

//     CONFIGURACIONES
#pragma config WDT = OFF      // deshabilita el WATCHDOG TIMER
#pragma config WDTPS = 256    // post-escala de 1:1024 aprox. 4.096 seg
#pragma config OSC = HSPLL           // habilita el PLLx4 con un oscilador de 10 Mhz
#pragma config PBADEN = OFF          // PORTB salida configuradas para I/O
#pragma config DEBUG = OFF
#pragma config LVP = OFF
```

```
#pragma idata         //  almacenamiento en memoria estatica

// FUNCIONES AGREGADAS

       // DATOS
void maneja_IntRx (void);           // funcion para manejar interrupciones
char Lee1b_Redint_PoR (void);       // lee 1 byte
char putchP_R(char c);              // Transmite por red interna
// transmite cadena de comandos para la lectura de sensores
void esc_cmd_redint_PoR (void);
void rellenaCOM(char fill);
void WDT_ON (void);                 // encendido y apagado de WatchDog Timer
void WDT_OFF (void);

       // SENSORES
void habSPI (void);                 // habilita el SPI por la compuerta
void habI2C (void);
void CambioSPI(void);               // cambio de puerto
void CambioI2C(void);
void AbrirCanalSPI(char canal);     // selecciona el canal SPI
void CerrarCanalSPI(char canal);

       // BRUJULA
void Abre_i2c(void);
void Inicia_i2c(void);
void Reinicia_i2c(void);
void Escribe_i2c(unsigned char dato);
unsigned char Lee_i2c(void);
void Ack_i2c(void);
void NoAck_i2c(void);
void alto_i2c(void);
void lee_brujula (unsigned char *buffer);   // funcion que lee la brujula electronica


       // ACELEROMETRO
void AbreADC(void);                         // configura el ADC
void Sensibilidad (char g_sel);             // cambia rango en g's del acelerometro
void AcelerometroX(void);                   // lecturas del acelerometro
void AcelerometroY(void);
void AcelerometroZ(void);
       //     GIROSCOPO
// lecturas del gyro
void Gyro(unsigned char canal,unsigned char *dato,unsigned char numero);
void TempGyro(unsigned char canal,unsigned char *dato,unsigned char numero);
// configuracion del gyro
void ConfigGyro(unsigned char canal,unsigned int config);
void PruebaGyro(void);

// VARIABLES
       // Principal
char CmdRecibido;             // solo algunas interrupciones hacen 'S' esta bandera
// byte de indicacion de falla en transimision y recepcion de datos,    => bandera
char Fallo_TOTAL_RedInt;
char nvodato;                 // byte leido de red interna
int chkext;
char ctk,r;
unsigned char cmdRx [14];     // comando de red interna Rx
unsigned char cmdTx [14];     //comando de red interna Tx
char Num_IntentosTxRx;        // numero de intentos
int aux_comando;              // variable auxiliar en el procesamiento de comandos
unsigned char i;              // contadores auxiliares de comando
// Interrupcion
char hay_dato;
// Brujula
unsigned char l_brujula[4];

// Gyro
unsigned char gyroX[2];
unsigned char gyroY[2];
unsigned char gyroZ[2];
unsigned char t_gyroX[2];
```

```
unsigned char t_gyroY[2];
unsigned char t_gyroZ[2];
char var;

#pragma code

//     HAB / DESHAB  INTERRUPCIONES
#define Deshabil_IntRx()            PIE1bits.RCIE = 0;
#define Habil_IntRx()               PIE1bits.RCIE = 1;
#define Idle()                                    OSCCONbits.IDLEN = 1;
#define CS1                                       PORTBbits.RB5
#define CS2                                       PORTBbits.RB4
#define CS3                                       PORTBbits.RB3
#define ST1                                       PORTAbits.RA4
#define ST2                                       PORTAbits.RA5

// Deshabilita las interrupciones por recepcion de datos
#define Deshabil_IntActualRI()  PIE1bits.RCIE = 0;
#define Habilita_IntActualRI()  PIE1bits.RCIE = 1;  // Habilita las interrupciones
// lectura de gyro, en modo comp a 2
#define c_Gyro                              0x8300
// lectura de temperatura de gyro, en modo comp a 2
#define c_TempGyro                          0x8700

/************************************************************/
/***************** PROGRAMA PRINCIPAL  **********************/
/************************************************************/

void main(void)
{

  // Habilita interrupciones y niveles de interrupcion
  RCONbits.IPEN = 1;                // habilita niveles de interrupción
  INTCONbits.GIE = 1;               // hab interrupción global
  INTCONbits.PEIE = 1;        // hab interrupción de periféricos
  IPR1 = 0x20;                      // interrupción de ALTA prioridad SOLO en RX
  IPR2 = 0;                   // interrupción de BAJA prioridad en lo demas

  // INICIALIZACIÓN DE VARIABLES
  nvodato = 0;
  Num_IntentosTxRx = 2;       // Intentos de comunicación por red interna
  CmdRecibido = 'N';          // solo la interrupción lo hace 'S'
  l_brujula[0] = 0;   // inicia la lectura de la brujula
  l_brujula[1] = 0;
  i = 0;                                    // inicia contadores

  TRISAbits.TRISA0 = 1;       // ASIGNA entradas para acelerómetro
  TRISAbits.TRISA1 = 1;
  TRISAbits.TRISA2 = 1;

  TRISAbits.TRISA4 = 0;       // selftest gyro ST1
  TRISAbits.TRISA5 = 0;       // selftest gyro ST2

  TRISBbits.TRISB3 = 0;       //     CS3    seleccion de GYROs, en SPI
  TRISBbits.TRISB4 = 0;       //     CS2
  TRISBbits.TRISB5 = 0;       //     CS1

  TRISCbits.TRISC0 = 0;       //     Toggle SPI-I2C, 0 - I2C, 1 - SPI
  // selectores de nivel de sensibilidad del acelerometro
  TRISCbits.TRISC1 = 0;       // (GS1)
  TRISCbits.TRISC2 = 0;       // (GS2)

  TRISCbits.TRISC3 = 0;       //  I/O del SPI
  TRISCbits.TRISC4 = 1;
  TRISCbits.TRISC5 = 0;

  PORTA = 0;                  // limpia puertos
  PORTB = 0;
  PORTC = 0;

  // INICIALIZACIÓN DE PUERTO SERIAL
```

```
OpenUSART (  USART_TX_INT_OFF &    // apaga interrupcion por transmision
             USART_RX_INT_OFF &
             USART_ASYNCH_MODE &   // modo asincrono
             USART_EIGHT_BIT &     // trans/recep en 8 bit
             USART_CONT_RX &       // modo continuo de recepcion
             USART_BRGH_HIGH,      // ALTO intervalo de BAUD
             255  ) ;      // De tablas => 9600 bauds, XT = 40MHz    (255)
                           //      => 115200 bauds, (21), => 57600 bauds (42)
Delay10KTCYx(50);          // inicio de puerto serie

while (1)
{
  Habil_IntRx();           // habilita la interrupcion por Rx del canal serial

  Idle();           // configuracion de consumo
            // Duerme al CPU del MCU, y espera a que lo despierte una interrupcion
  Sleep();
  if (CmdRecibido=='S')
  {
      Deshabil_IntRx(); //deshabilita la interrupcion por PUERTO SERIE
      CmdRecibido='N';                     /*Reset a bandera*/
      WDT_ON();      // enciende watch dog en caso de que el programa se atore

      for ( ctk=1; ctk<13; ctk++ )   /*lee trama de comando, 12 bytes */
      {
       chkext = chkext + cmdRx [ctk];
      }

      chkext=chkext & 0xFF;
      chkext=0-chkext;
      chkext=chkext & 0xFF;                    /* CHEKSUM CALCULADO */

      if ( chkext == cmdRx [13] )          // verifica el CHECKSUM
      {
       chkext = 0;
       putchP_R ('K');             // envia ACKNOWLEDGE

       for ( i = 1; i<14;i++)              // Ejecución de COMANDOS
       {

        switch (cmdRx [ i ]){

              //  SENSORES
              case 0xB0:     // DIRECCION de la brujula
              CambioI2C();
              lee_brujula(l_brujula);                     // lee el COMPASS
              cmdTx[1] = l_brujula[2];
              cmdTx[2] = l_brujula[3];
              StopI2C();
              AcelerometroX();                            // lee acelerometro
              cmdTx[3] = ADRESH;
              cmdTx[4] = ADRESL;

              AcelerometroY();
              cmdTx[5] = ADRESH;
              cmdTx[6] = ADRESL;
              AcelerometroZ();
              cmdTx[7] = ADRESH;
              cmdTx[8] = ADRESL;
              CloseADC();
              rellenaCOM(4);               // rellena lo que sobro del comando
              esc_cmd_redint_PoR ();       // envia lecturas de brujula y acelerometro
              break;

              case 0xB1:     CambioSPI();
              ConfigGyro('X',c_Gyro);
              Gyro('X',gyroX,2);
              cmdTx[1] = gyroX [0];
              cmdTx[2] = gyroX [1];

              ConfigGyro('Y',c_Gyro);
```

```
Gyro('Y',gyroY,2);
cmdTx[3] = gyroY [0];
cmdTx[4] = gyroY [1];

ConfigGyro('Z',c_Gyro);
Gyro('Z',gyroZ,2);
cmdTx[5] = gyroZ [0];
cmdTx[6] = gyroZ [1];

ConfigGyro('X',c_TempGyro);
TempGyro('X',t_gyroX,2);
cmdTx[7] = t_gyroX [0];
cmdTx[8] = t_gyroX [1];

ConfigGyro('Y',c_TempGyro);
TempGyro('Y',t_gyroY,2);
cmdTx[9] = t_gyroY [0];
cmdTx[10] = t_gyroY [1];

ConfigGyro('Z',c_TempGyro);
TempGyro('Z',t_gyroZ,2);
cmdTx[11] = t_gyroZ [0];
cmdTx[12] = t_gyroZ [1];

esc_cmd_redint_PoR ();       // envia lecturas de brujula y acelerometro
break;

case 0xB2:     CambioI2C();
lee_brujula(l_brujula);                    // lee el COMPASS
cmdTx[1] = l_brujula[2];
cmdTx[2] = l_brujula[3];
StopI2C();

AcelerometroX();                           // lee acelerometro
cmdTx[3] = ADRESH;
cmdTx[4] = ADRESL;

AcelerometroY();
cmdTx[5] = ADRESH;
cmdTx[6] = ADRESL;
AcelerometroZ();
cmdTx[7] = ADRESH;
cmdTx[8] = ADRESL;
CloseADC();

CambioSPI();
ConfigGyro('X',c_Gyro);
Gyro('X',gyroX,2);
cmdTx[9] = gyroX [0];
cmdTx[10] = gyroX [1];

ConfigGyro('Y',c_Gyro);
Gyro('Y',gyroY,2);
cmdTx[11] = gyroY [0];
cmdTx[12] = gyroY [1];

esc_cmd_redint_PoR ();// envia lecturas de brujula y acelerometro

ConfigGyro('Z',c_Gyro);
Gyro('Z',gyroZ,2);
cmdTx[1] = gyroZ [0];
cmdTx[2] = gyroZ [1];

ConfigGyro('X',c_TempGyro);
TempGyro('X',t_gyroX,2);
cmdTx[3] = t_gyroX [0];
cmdTx[4] = t_gyroX [1];
//     TgyroY();
ConfigGyro('Y',c_TempGyro);
TempGyro('Y',t_gyroY,2);
cmdTx[5] = t_gyroY [0];
```

```
                cmdTx[6] = t_gyroY [1];

                //     TgyroZ();
                ConfigGyro('Z',c_TempGyro);
                TempGyro('Z',t_gyroZ,2);
                cmdTx[7] = t_gyroZ [0];
                cmdTx[8] = t_gyroZ [1];

                rellenaCOM(4);              // rellena lo que sobro del comando
                esc_cmd_redint_PoR ();// envia lecturas de brujula y acelerometro
                break;

                // Cambio de SENSIBILIDAD en acelerometro
                case 0xB3:     Sensibilidad(cmdRx[i+1]);
                               i++; break;
                case 0xB4:     WDT_OFF();
                               PruebaGyro();
                               WDT_ON();
                               break;

                               default: Nop(); break;}
                }      // FIN DE FOR
        }
        else
        {
          chkext = 0;
          putchP_R ('N');     // envia NO ACKNOWLEDGE, falla de checksum
        }
    }   // FIN DE IF(CmdRecibido=='S')

    else       putchP_R ('N');       // envia NO ACKNOWLEDGE, falla de acknowledge
    WDT_OFF();                // apaga WATCHDOG
  }     //  FIN DE WHILE(1)
}


/******************  SENSORES       *********************************/
// Cambios en compuerta
void habSPI (void)
{
       PORTCbits.RC0 = 1;           // bit de hab para el MUX 2 a 1
       Delay10TCYx(1);
}
void habI2C (void)
{
       PORTCbits.RC0 = 0;           // bit de hab para el MUX 2 a 1
       Delay10TCYx(1);
}

void CambioSPI(void)
{
       CloseI2C();           // Cierra el I2C
       Delay10KTCYx(1);                    // Delay de 1ms para estabilizarse
       habSPI();             // habilita la compuerta
       OpenSPI ( SPI_FOSC_16 , MODE_01, SMPMID);          // habilita las funciones de
MAESTRO, el pin SS se manejara aparte
       Delay10KTCYx(1);                    // Delay de 1ms para estabilizarse
}
void CambioI2C(void)
{
       CloseSPI();
       Delay10KTCYx(1);                    // Delay de 1ms para estabilizarse
       habI2C();
       OpenI2C(MASTER,SLEW_OFF); // I2C como maestro, SLEW OFF para transmitir a 100KHz
       SSPADD = 99;          // baud rate 100k en 40Mhz, clock = Fosc/(4*(SSPADD+1))
       Delay10KTCYx(1);      // Delay de 1ms para estabilizarse
}

void AbrirCanalSPI(char canal)
{
       switch (canal){
```

```
        case 'X':             CS1 = 0;
                              CS2 = 1;
                              CS3 = 1;break;
        case 'Y':             CS1 = 1;
                              CS2 = 0;
                              CS3 = 1;break;
        case 'Z':             CS1 = 1;
                              CS2 = 1;
                              CS3 = 0;break;
        }
}
void CerrarCanalSPI(char canal)
{
        switch (canal){
        case 'X':             CS1 = 1;break;
        case 'Y':             CS2 = 1;break;
        case 'Z':             CS3 = 1;break;
        }
}

/**************       BRUJULA        ******************/
void Abre_i2c(void)
{
        OpenI2C(MASTER,SLEW_OFF);  // I2C como maestro, SLEW OFF para transmitir a 100KHz
        SSPADD = 99;          // baud rate 100k en 40Mhz, clock = Fosc/(4*(SSPADD+1))
}

void Inicia_i2c(void)
{
        StartI2C();                     // condicion de inicio del bus
        while(SSPCON2bits.SEN);       // espera a que este listo
}

void Reinicia_i2c(void)
{
        SSPCON2bits.RSEN = 1;  // envia reinicio
        while(SSPCON2bits.RSEN); // espera el reinicio
}

void Escribe_i2c(unsigned char dato)
{
        PIR1bits.SSPIF = 0;                // baja bandera
        SSPBUF = dato;                     // escribe el dato
        while(!PIR1bits.SSPIF);       // espera a que se complete la trasmision
        PIR1bits.SSPIF = 0;           // baja bandera
}

unsigned char Lee_i2c(void)
{
        SSPCON2bits.RCEN = 1;         // comienza a recibir
        while(!SSPSTATbits.BF);       // espera datos
        return (SSPBUF);                   // los regresa
}

void Ack_i2c(void)
{
        SSPCON2bits.ACKEN = 1;             // comienza secuencia de Acknowledge
        while(SSPCON2bits.ACKEN);     // espera el acknowledge
}

void NoAck_i2c(void)
{
        SSPCON2bits.ACKDT = 1;             // No acknowledge para el ultimo byte
        SSPCON2bits.ACKEN = 1;             // comienza secuencia
        while(SSPCON2bits.ACKEN);     // espera el acknowledge y termina
}
void alto_i2c(void)
{
        SSPCON2bits.PEN = 1;          // envia bit de alto
        while(SSPCON2bits.PEN);            // espera el bit
}
```

```
void lee_brujula(unsigned char *buffer)
{
	Inicia_i2c();			// condicion de inicio del bus
	SSPCON2bits.ACKDT = 0;		// bit de acknowledge

	Escribe_i2c(0xc0);	// esribe la direccion
	Escribe_i2c(0);

	Reinicia_i2c();			// comienza de nuevo
	Escribe_i2c(0xc1);	// direcion

	buffer[0] = Lee_i2c();		// lee version,
	Ack_i2c();

	buffer[1] = Lee_i2c();		// dato de 0 a 255,
	Ack_i2c();
					// datos de 0 a 3599
	buffer[2] = Lee_i2c();		// MSB
	Ack_i2c();

	buffer[3] = Lee_i2c();		//LSB
	NoAck_i2c();

	alto_i2c();			// detiene el bus
}

/***************  ACELEROMETRO  ****************************/

/*****************************************************************
*	En este acelerometro hay que considerar durante las lecturas,
*	que siempre dara la medicion del punto medio de 3.3V que
*	es con el voltaje con el cual se alimenta el transductor,
*	es decir 1.65 V pero en su forma digital, para indicarnos
*	que NO ha registrado movimiento, al moverse la lectura
*	dependera de si esta es mayor o menor a este punto medio
*	y nos indicara que se ha movido en un sentido u otro,
*	ya que se ha fijado el valor del voltaje el ADC divide 3.3V
*	entre el valor del convertidor 1024(10 bits), dandonos
*	3.2227mV (aprox.) por bit de diferencia
*****************************************************************/
// CONFIGURA EL CONVERTIDOR A/D
void AbreADC(void)
{
	ADCON1 = 0b00011100;  // Vref- = GND, Vref+ = Externo(3.3V),salidas analogicas de la
AN0-AN2
	ADCON2 = 0b10110010;  // Justificado a la DERECHA, 16 TAD, Fosc/32
	ADCON0bits.ADON = 1;  // enciende el convertidor
}

void Sensibilidad(char g_sel)
{
	switch (g_sel){
		case 1: PORTCbits.RC1=0;
				PORTCbits.RC2=0;
				break;
		case 2: PORTCbits.RC1=1;
				PORTCbits.RC2=0;
				break;
		case 3: PORTCbits.RC1=0;
				PORTCbits.RC2=1;
				break;
		case 4: PORTCbits.RC1=1;
				PORTCbits.RC2=1;
				break;
		default: Nop(); break;
	}
}

void AcelerometroX(void)
{
```

```
	ADCON0 = 0;				// selecciona el canal 0
	AbreADC();
	Delay10TCYx(1);
	ConvertADC();			// convierte y ...
	while(BusyADC());			// mientras este ocupado no puede enviar

}

void AcelerometroZ(void)
{
	ADCON0 = 0b00000100;  // selecciona el canal 1
	AbreADC();
	Delay10TCYx(1);
	ConvertADC();			// convierte y ...
	while(BusyADC());			// mientras este ocupado no puede enviar
}

void AcelerometroY(void)
{
	ADCON0 = 0b00001000;  // selecciona el canal 2
	AbreADC();
	Delay10TCYx(1);
	ConvertADC();			// convierte y ...
	while(BusyADC());			// mientras este ocupado no puede enviar
}

/******************   GIROSCOPO      ******************************/
// Configuiracion de los Gyros

void ConfigGyro(unsigned char canal,unsigned int config)
{

	AbrirCanalSPI(canal);		// habilita el gyro
	putcSPI(config>>8);		// envia los bytes de config
	putcSPI(config);
	CerrarCanalSPI(canal);

	AbrirCanalSPI(canal);		// habilita el gyro
	getcSPI();			// tira la basura
	getcSPI();
	CerrarCanalSPI(canal);

}

//     LECTURA DE GYROS
void Gyro(unsigned char canal,unsigned char *dato,unsigned char numero)   // lectura del
Gyro en X
{

	AbrirCanalSPI(canal);		// habilita el gyro
	var = putcSPI(0x03);		// envia los bytes de lectura
	var = putcSPI(0x00);
	CerrarCanalSPI(canal);

	AbrirCanalSPI(canal);		// habilita el gyro

	while(numero)			// recibe la lectura
	{
		*dato++ = getcSPI();  // lee dato a dato aumentando el apuntador
		numero--;
	}
	CerrarCanalSPI(canal);

}

// temperatura del gyro

void TempGyro(unsigned char canal,unsigned char *dato,unsigned char numero)      // lectura
de la temperatura del Gyro
{
```

```
        AbrirCanalSPI(canal);         // habilita el gyro
        var = putcSPI(0x07);          // envia los bytes de lectura
        var = putcSPI(0x00);
        CerrarCanalSPI(canal);

        AbrirCanalSPI(canal);         // habilita el gyro

        while(numero)                 // recibe la lectura
        {
                *dato++ = getcSPI();
                numero--;
        }
        CerrarCanalSPI(canal);

}

void PruebaGyro(void)
{
        Deshabil_IntRx();             // Deshabilita la interrupcion por recepcion
        CambioSPI();                  // Cambia al puerto SPI

        T0CON = 0x07;                         // timer0 config. 16 bit, preescaler 1:256
        INTCONbits.TMR0IF = 0;        //baja bandera

        ConfigGyro('X',c_Gyro);               // configura los Gyros para recibir lecturas
        ConfigGyro('Y',c_Gyro);
        ConfigGyro('Z',c_Gyro);

        while ( RCREG != 'G')
        {
                TMR0H = 0xf0;         // carga para tiempo a 100 ms (0xf0bd),
                TMR0L = 0xbd;
                T0CONbits.TMR0ON = 1;

                while(INTCONbits.TMR0IF == 0);
                INTCONbits.TMR0IF = 0;                //baja bandera

                WriteUSART('/');              // encabezado de recepcion de datos
                while(BusyUSART());

                Gyro('X',gyroX,2);
                WriteUSART(gyroX [0]);
                while(BusyUSART());
                WriteUSART(gyroX [1]);
                while(BusyUSART());

                Gyro('Y',gyroY,2);
                WriteUSART(gyroY [0]);
                while(BusyUSART());
                WriteUSART(gyroY [1]);
                while(BusyUSART());

                Gyro('Z',gyroX,2);
                WriteUSART(gyroZ [0]);
                while(BusyUSART());
                WriteUSART(gyroZ [1]);
                while(BusyUSART());

                AcelerometroX();                              // lee acelerometro eje X
                WriteUSART(ADRESH);
                while(BusyUSART());
                WriteUSART(ADRESL);
                while(BusyUSART());

                AcelerometroY();                              // lee acelerometro eje Y
                WriteUSART(ADRESH);
                while(BusyUSART());
                WriteUSART(ADRESL);
                while(BusyUSART());

                AcelerometroZ();                              // lee acelerometro eje Z
```

```
               WriteUSART(ADRESH);
               while(BusyUSART());
               WriteUSART(ADRESL);
       }

       T0CONbits.TMR0ON = 0;
       Habil_IntRx();                      // Habilita la interrupcion por Rx

}



// INTERRUPCIONES
#pragma code rx_interrupt = 0x08                          // vector de interrupcion
void rx_int ( void )
{
       _asm GOTO maneja_IntRx _endasm
}
#pragma code

#pragma interrupt maneja_IntRx                            // rutina de interrupcion
void maneja_IntRx (void)
{
    /* Define si hay dato en el canal principal   */
    nvodato = 0;
    hay_dato = 0;

    hay_dato = Lee1b_Redint_PoR();
    cmdRx [0] = nvodato;                    // byte de  despertar = cmdRx[0]

    if ( cmdRx [0]== 0x2F )   /*Mientras haya algun dato en el canal principal:*/
    {
      for (ctk=1; ctk<=13; ctk++)           /*lee trama de comando, 14 bytes */
      {        /*Verifica POR POLEO si llega algun dato al puerto serie*/
        hay_dato = Lee1b_Redint_PoR();
        if ( hay_dato ) /*Mientras haya algun dato en el canal principal:*/
        {
          cmdRx [ctk] = nvodato;            /* almacena comando */
        }
        else  ctk=40;
               /*Si no hay dato obliga a salir del for para seguir leyendo*/
       }

       if (ctk==40)  CmdRecibido = 'N';
       else  CmdRecibido='S';
    }
    else       CmdRecibido = 'N';
 }
```

Apéndice D

Software en Visual Basic 6

Interfaz de pruebas de los subsistemas

```
Option Explicit
Dim regRx As String                  ' Byte(s) recibido(s)
Dim CmdRx(1 To 14) As Integer
Dim inclinacion As Integer
Dim comando(1 To 16) As Byte            ' Comando a enviar
Dim i, j As Byte                        ' conteo auxiliar
Dim cont As Integer
Public velocidad As String
Public ct As String
Public btmx, btmy, btmz As String
Public frec As String
Public aux, straux As String
Dim numaux As Byte
Dim sensibilidad As Byte                ' sensibilidad del acelerometro
Dim long_mensaje As Integer             ' longitud
Dim frase(1 To 14) As String
Dim ban_sensores As Boolean             ' bandera de recepcion del modulo de audio y
sensores
Dim dato_leido(1 To 7) As Byte          ' grupo de datos leidos

Dim tiempoespera As Integer         ' conteo delay
Dim FALLA As Boolean                ' falla de CHARLA
Dim long_trama As Byte              ' longitud de la trama

Dim trama As String                 ' trama de datos de CHARLA
Dim trama2 As String

Dim respuesta As Byte
Public com_motor, com_audio As String

Dim alto, reinicio As Boolean           ' alto a prueba de Gyros
Dim pausa As Byte                       ' pausa de prueba de gyros


Function BrujulayAcelerometro()
    fraBrujula.Enabled = True
    lblDir.Enabled = True
    fraAcelerometro.Enabled = True
    fraSensibilidad.Enabled = True
    fraEjes.Enabled = True
    lblex.Enabled = True
    lbley.Enabled = True
    lblez.Enabled = True
    opt1G.Enabled = True
    opt2G.Enabled = True
    opt4G.Enabled = True
    opt6G.Enabled = True
    chkCambiarSens.Enabled = True
    txtVX.Enabled = True
    txtVY.Enabled = True
    txtVZ.Enabled = True

    fraGyro.Enabled = False
    lblPx.Enabled = False
    lblPy.Enabled = False
```

```
    lblPz.Enabled = False
    lblTx.Enabled = False
    lblTy.Enabled = False
    lblTz.Enabled = False
    txtPX.Enabled = False
    txtPY.Enabled = False
    txtPZ.Enabled = False
    txtTX.Enabled = False
    txtTY.Enabled = False
    txtTZ.Enabled = False

End Function
Function Gyro()
    fraBrujula.Enabled = False
    lblDir.Enabled = False
    fraAcelerometro.Enabled = False
    fraSensibilidad.Enabled = False
    fraEjes.Enabled = False
    lblex.Enabled = False
    lbley.Enabled = False
    lblez.Enabled = False
    opt1G.Enabled = False
    opt2G.Enabled = False
    opt4G.Enabled = False
    opt6G.Enabled = False
    chkCambiarSens.Enabled = False
    txtVX.Enabled = False
    txtVY.Enabled = False
    txtVZ.Enabled = False

    fraGyro.Enabled = True
    lblPx.Enabled = True
    lblPy.Enabled = True
    lblPz.Enabled = True
    lblTx.Enabled = True
    lblTy.Enabled = True
    lblTz.Enabled = True
    txtPX.Enabled = True
    txtPY.Enabled = True
    txtPZ.Enabled = True
    txtTX.Enabled = True
    txtTY.Enabled = True
    txtTZ.Enabled = True
End Function
Function TodoSensores()
    fraBrujula.Enabled = True
    lblDir.Enabled = True
    fraAcelerometro.Enabled = True
    fraSensibilidad.Enabled = True
    fraEjes.Enabled = True
    lblex.Enabled = True
    lbley.Enabled = True
    lblez.Enabled = True
    opt1G.Enabled = True
    opt2G.Enabled = True
    opt4G.Enabled = True
    opt6G.Enabled = True
    chkCambiarSens.Enabled = True
    txtVX.Enabled = True
    txtVY.Enabled = True
    txtVZ.Enabled = True

    fraGyro.Enabled = True
    lblPx.Enabled = True
    lblPy.Enabled = True
    lblPz.Enabled = True
    lblTx.Enabled = True
    lblTy.Enabled = True
    lblTz.Enabled = True
    txtPX.Enabled = True
    txtPY.Enabled = True
```

```
    txtPZ.Enabled = True
    txtTX.Enabled = True
    txtTY.Enabled = True
    txtTZ.Enabled = True
End Function
Sub NoSens()
    fraBrujula.Enabled = False
    lblDir.Enabled = False
    fraAcelerometro.Enabled = False
    fraSensibilidad.Enabled = False
    fraEjes.Enabled = False
    lblex.Enabled = False
    lbley.Enabled = False
    lblez.Enabled = False
    opt1G.Enabled = False
    opt2G.Enabled = False
    opt4G.Enabled = False
    opt6G.Enabled = False
    chkCambiarSens.Enabled = False
    fraGyro.Enabled = False
    lblPx.Enabled = False
    lblPy.Enabled = False
    lblPz.Enabled = False
    lblTx.Enabled = False
    lblTy.Enabled = False
    lblTz.Enabled = False
End Sub

Private Sub botonCrea_Click()
Dim respuesta As String
Dim estimacion As Byte
Dim api As Byte     ' contador de mensaje
    comando(1) = &H2F    ' cabecera del comando 0x2F
    i = 2

If tabsModulos.SelectedItem.Index = 1 Then

    If chkTelemetria.Value = 1 Then
        comando(i) = &HC5
        i = i + 1
    End If

    If chkMotor.Value = 1 Then
        Select Case ComboMotor.Text
            Case "Fijar CT a motor"
                If txtCT.Text = "" Then       ' verifica que haya un valor en la casilla
                    respuesta = MsgBox("ASIGNA UN VALOR A CT", 16, "ERROR")
                    txtCT.SetFocus
                    Exit Sub
                End If
                ct = Hex(txtCT.Text)          ' cambia el valor de CT en hex

                comando(i) = &HC0
                aux = ct

                While Len(aux) < 4      ' si la cadena es menor que 4 le aumento un cero
                    aux = 0 & aux
                Wend
                straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
                numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
                If optDer.Value = True Then   ' determina el sentido del motor
                    comando(i + 1) = &H80 Or numaux
                Else
                    comando(i + 1) = numaux
                End If
                straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
                numaux = Val("&h" & straux)   ' convierte en enteros
                comando(i + 2) = numaux
                i = i + 3                     ' aumenta contador
            Case "Velocidad"
                If txtVelocidad.Text = "" Then
```

```
                respuesta = MsgBox("ASIGNA UN VALOR A VELOCIDAD", 16, "ERROR")
                txtVelocidad.SetFocus
                Exit Sub
            End If

            comando(i) = &HC1
            velocidad = Hex(txtVelocidad.Text)
            aux = velocidad

            While Len(aux) < 4      ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
            numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
            If optDer.Value = True Then   ' determina el sentido del motor
                comando(i + 1) = &H80 Or numaux
            Else
                comando(i + 1) = numaux
            End If
            straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
            numaux = Val("&h" & straux)   ' convierte en enteros
            comando(i + 2) = numaux
            i = i + 3                     ' aumenta contador
        Case "Cambia Velocidad"
            comando(i) = &HC2
            If txtVelocidad.Text = "" Then
                respuesta = MsgBox("ASIGNA UN VALOR A VELOCIDAD", 16, "ERROR")
                txtVelocidad.SetFocus
                Exit Sub
            End If
            velocidad = Hex(txtVelocidad.Text)
            aux = velocidad
            While Len(aux) < 4      ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
            numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
            comando(i + 1) = numaux
            straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
            numaux = Val("&h" & straux)   ' convierte en enteros
            comando(i + 2) = numaux
            i = i + 3                     ' aumenta contador
        Case "Apagar Motor"
           comando(i) = &HC3
           i = i + 1
        Case "Frenar Motor"
           comando(i) = &HC4
           i = i + 1
        Case Else
            respuesta = MsgBox("ASIGNA UNA FUNCION AL MOTOR", 16, "Advertencia")
            ComboMotor.SetFocus
    End Select
End If

If chkBTMX.Value = 1 Then
    If txtCTX.Text = "" Then
         respuesta = MsgBox("ASIGNA UN VALOR A CT EN BTMX", 48, "AVISO")
         txtCTX.SetFocus
         Exit Sub
    End If

    If optbutXfina.Value = True Then
        comando(i) = &HB0
            btmx = Hex(txtCTX.Text)
            aux = btmx

            While Len(aux) < 4     ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
            numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
```

```
            If optX1.Value = True Then    ' determina el sentido del motor
                comando(i + 1) = &H80 Or numaux
            Else
                comando(i + 1) = &H40 Or numaux
            End If
            straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
            numaux = Val("&h" & straux)   ' convierte en enteros
            comando(i + 2) = numaux
            i = i + 3                     ' aumenta contador
    Else
        comando(i) = &HB1
            btmx = Hex(txtCTX.Text)
            aux = btmx

            While Len(aux) < 4     ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
            numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
            If optX1.Value = True Then    ' determina el sentido del motor
                comando(i + 1) = &H80 Or numaux
            Else
                comando(i + 1) = &H40 Or numaux
            End If
            straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
            numaux = Val("&h" & straux)   ' convierte en enteros
            comando(i + 2) = numaux
            i = i + 3                     ' aumenta contador
    End If
End If

If chkBTMY.Value = 1 Then
    If txtCTY.Text = "" Then
           respuesta = MsgBox("ASIGNA UN VALOR A CT EN BTMY", 48, "AVISO")
           txtCTY.SetFocus
           Exit Sub
    End If

    If optbutYfina.Value = True Then
        comando(i) = &HB2
            btmy = Hex(txtCTY.Text)
            aux = btmy

            While Len(aux) < 4     ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
            numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
            If optY1.Value = True Then    ' determina el sentido del motor
                comando(i + 1) = &H80 Or numaux
            Else
                comando(i + 1) = &H40 Or numaux
            End If
            straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
            numaux = Val("&h" & straux)   ' convierte en enteros
            comando(i + 2) = numaux
            i = i + 3                     ' aumenta contador
    Else
        comando(i) = &HB3
            btmy = Hex(txtCTY.Text)
            aux = btmy

            While Len(aux) < 4      ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
            numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
            If optY1.Value = True Then    ' determina el sentido del motor
                comando(i + 1) = &H80 Or numaux
            Else
                comando(i + 1) = &H40 Or numaux
```

```
                End If
                straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
                numaux = Val("&h" & straux)    ' convierte en enteros
                comando(i + 2) = numaux
                i = i + 3                      ' aumenta contador
            End If
        End If

        If chkBTMZ.Value = 1 Then
            If txtCTZ.Text = "" Then
                respuesta = MsgBox("ASIGNA UN VALOR A CT EN BTMZ", 48, "AVISO")
                txtCTZ.SetFocus
                Exit Sub
            End If

            If optbutZfina.Value = True Then
                comando(i) = &HB4
                    btmz = Hex(txtCTZ.Text)
                    aux = btmz

                    While Len(aux) < 4      ' si la cadena es menor que 4 le aumento un cero
                        aux = 0 & aux
                    Wend
                    straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
                    numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
                    If optZ1.Value = True Then    ' determina el sentido del motor
                        comando(i + 1) = &H80 Or numaux
                    Else
                        comando(i + 1) = &H40 Or numaux
                    End If
                    straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
                    numaux = Val("&h" & straux)   ' convierte en enteros
                    comando(i + 2) = numaux
                    i = i + 3                     ' aumenta contador
            Else
                comando(i) = &HB5
                    btmz = Hex(txtCTZ.Text)
                    aux = btmz

                    While Len(aux) < 4      ' si la cadena es menor que 4 le aumento un cero
                        aux = 0 & aux
                    Wend
                    straux = Left(aux, 2)         ' toma los 2 primeros de la cadena(hex)
                    numaux = Val("&h" & straux)   ' convierte de numeros hex en numeros(dec)
                    If optZ1.Value = True Then    ' determina el sentido del motor
                        comando(i + 1) = &H80 Or numaux
                    Else
                        comando(i + 1) = &H40 Or numaux
                    End If
                    straux = Right(aux, 2)         ' toma los 2 ultimos de la cadena en hex
                    numaux = Val("&h" & straux)   ' convierte en enteros
                    comando(i + 2) = numaux
                    i = i + 3                     ' aumenta contador
            End If
            If i > 14 Then
                respuesta = MsgBox("DEMASIADOS COMANDOS", 16, "ERROR")
                Exit Sub
            End If
        End If

        If chkBtmxoff.Value = 1 Then
            VerificaLongComando         ' subrutina que verifica la longitud del comando no sea
excesiva
            comando(i) = &HB6
            i = i + 1
        End If

        If chkBtmyoff.Value = 1 Then
            VerificaLongComando
            comando(i) = &HB7
            i = i + 1
```

```
End If

If chkBtmzoff.Value = 1 Then
    VerificaLongComando
    comando(i) = &HB8
    i = i + 1
End If

If chkBtmsoff.Value = 1 Then
    VerificaLongComando
    comando(i) = &HB9
    i = i + 1
End If

If chkControl.Value = 1 Then
    If txtFrecuencia.Text = "" Then
        respuesta = MsgBox("ASIGNA UN VALOR A LA FRECUENCIA DE LA SEÑAL", 48, "AVISO")
        txtFrecuencia.SetFocus
        Exit Sub
    End If

    Select Case ComboControl.Text
        Case "Senoidal"
            comando(i) = &HA0
            frec = Hex(txtFrecuencia.Text)
            aux = frec
            While Len(aux) < 2      ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            numaux = Val("&h" & aux)   ' convierte de numeros hex en numeros(dec)
            comando(i + 1) = numaux
            i = i + 2                    ' aumenta contador

        Case "Triangular"
            comando(i) = &HA1
            frec = Hex(txtFrecuencia.Text)
            aux = frec
            While Len(aux) < 2      ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            numaux = Val("&h" & aux)   ' convierte de numeros hex en numeros(dec)
            comando(i + 1) = numaux
            i = i + 2                    ' aumenta contador

        Case "Diente de Sierra"
            comando(i) = &HA2
            frec = Hex(txtFrecuencia.Text)
            aux = frec
            While Len(aux) < 2      ' si la cadena es menor que 4 le aumento un cero
                aux = 0 & aux
            Wend
            numaux = Val("&h" & aux)   ' convierte de numeros hex en numeros(dec)
            comando(i + 1) = numaux
            i = i + 2                    ' aumenta contador

        Case Else
            respuesta = MsgBox("ASIGNA UNA SEÑAL", 48, "Advertencia")
            ComboControl.SetFocus
            Exit Sub
    End Select
    If i > 14 Then
        respuesta = MsgBox("DEMASIADOS COMANDOS", 16, "ERROR")
        Exit Sub
    End If
End If

While i < 14
    comando(i) = &H41           ' si no se llena el comando se rellena con AA
    i = i + 1
Wend
```

```
    comando(14) = CHECKSUM          ' calcula checksum

    txtComando.Enabled = True       'habilita la ventana de muestra del comando
        txtComando.Text = ""        ' limpia la ventana
    For i = 1 To 14 Step 1
        txtComando.Text = txtComando.Text & Hex(comando(i)) & " "    ' imprime el comando
    Next i
    com_motor = txtComando.Text     ' graba el comando anterior
    txtComando.Enabled = False
End If

If tabsModulos.SelectedItem.Index = 2 Then
'se encuentra seleccionado el modulo de sensores

    ban_sensores = False       ' baja la bandera de audio recepcion
    comando(1) = &H2F    ' cabecera del comando 0x2F
    j = 2

    For cont = 1 To 7 Step 1   ' borra memoria de datos a leer, para asignarlos otra vez
        dato_leido(cont) = 0
    Next cont

    If chkSensores.Value = 1 Then   ' lectura de sensores en 3 modalidades
                                    ' * brujula y acelerometro
                                    ' * gyro
                                    ' * todos los sensores
        If optBrujAcel.Value = True Then
            comando(j) = &HB0
            j = j + 1
            VerificaSensibilidad
        ElseIf optGyro.Value = True Then
            comando(j) = &HB1
            j = j + 1
        ElseIf optTodos.Value = True Then
            comando(j) = &HB2
            j = j + 1
            VerificaSensibilidad
        End If
    End If

    While j < 14
        comando(j) = &H41               ' si no se llena el comando se rellena con AA
        j = j + 1
    Wend

    comando(14) = CHECKSUM          ' calcula el checksum

    txtComando.Enabled = True       ' habilita la ventana de muestra del comando
    txtComando.Text = ""            ' limpia la ventana
    For j = 1 To 14 Step 1
        txtComando.Text = txtComando.Text & Hex(comando(j)) & " "
       ' imprime el comando
    Next j
    com_audio = txtComando.Text     ' graba el comando anterior
    txtComando.Enabled = False

End If

End Sub

Function CHECKSUM() As Integer
Dim chks As Byte
Dim chksum As Integer

    For chks = 2 To 13 Step 1           ' suma los terminos del comando
        chksum = chksum + comando(chks)
    Next chks

    chksum = chksum And &HFF
    chksum = 0 - chksum
    chksum = chksum And &HFF            ' checksum calculado
```

```
    CHECKSUM = chksum

End Function

Sub VerificaLongComando()
        If i > 13 Then
            respuesta = MsgBox("DEMASIADOS COMANDOS", 16, "ERROR")
            Exit Sub
        End If
End Sub

' verifica la sensibilidad del acelerometro y la modifica en el comando
Sub VerificaSensibilidad()
        If chkCambiarSens.Value = 1 Then
            comando(j) = &HB3
            j = j + 1
            Select Case sensibilidad
            Case 1
                comando(j) = sensibilidad
                j = j + 1
            Case 2
                comando(j) = sensibilidad
                j = j + 1
            Case 3
                comando(j) = sensibilidad
                j = j + 1
            Case 4
                comando(j) = sensibilidad
                j = j + 1
            End Select
        End If
End Sub

Private Sub botonEnviar_Click()
Dim corriente As Single
Dim chksum As Integer
Dim chkext As Byte
Dim respuesta As String
Dim otravez As Byte
Dim valor As Long
    stbarAyuda.Panels("mensaje").Text = "Enviando..."
    stbarAyuda.Panels("mensaje").Picture = imglstRx.ListImages("enviando").Picture
    For i = 1 To 14 Step 1
        COMM1.Output = Chr(comando(i))      ' envia los comandos uno por uno
    Next i

    otravez = 1

'   CHARLA C0N SENSORES

' Hay sensores va a modo de CHARLA

    If chkSensores.Value = 1 And tabsModulos.SelectedItem.Index = 2 Then

        saludo (1)      ' espera N o K para ver si llego bien el comando
        If FALLA = True Then
            TimerEspera.Enabled = False ' se acabo el tiempo
            FALLA = False               ' reset a acondicion de falla
            Exit Sub
        Else
            If Len(trama) = 0 Then
                richtxtRx.Text = richtxtRx.Text & "FALLA " & Time & Chr(10)
                Exit Sub
            End If

            SeparaBytes trama, 1
            Select Case Chr(CmdRx(1))
                Case "N"        ' seleccion de dato recibidos
                    richtxtRx.Text = richtxtRx.Text & "<N> Fallo " & Time & Chr(10)
'Si recibe una N no se recibio el comando
                    stbarAyuda.Panels("mensaje").Text = "No llegó..."
```

```
                    stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("noentregado").Picture
                    Exit Sub
                Case "K"
                    richtxtRx.Text = richtxtRx.Text & "<K> Recibido " & Time & Chr(10)
                    stbarAyuda.Panels("mensaje").Text = "Enviado"
                    stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("entregado").Picture
                    COMM1.Output = "E"  ' pide el envio
                Case Else
                    richtxtRx.Text = richtxtRx.Text & "<0x" & Hex(CmdRx(1)) & "> Dato
Desconocido " & Time & Chr(10)
                    stbarAyuda.Panels("mensaje").Text = "Dato falso..."
                    stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("falso").Picture
                    Exit Sub
            End Select
        End If
        richtxtRx.Text = richtxtRx.Text & "Esperando Telemetría ..." & Chr(10)
        ' Se prepara a recibir comando

OTRA:   saludo (14)         ' selecciona los primeros 14 bytes
        If FALLA = True Then
            TimerEspera.Enabled = False ' se acabo el tiempo
            FALLA = False               ' reset a condicion de falla
            Exit Sub
        Else
            If Len(trama) = 0 Then
                richtxtRx.Text = richtxtRx.Text & "FALLA " & Time & Chr(10)
                Exit Sub
            End If

            SeparaBytes trama, 14   ' introduce la trama y separa los datos en cmdRx
        End If

        chksum = 0
        For chkext = 2 To 13 Step 1           ' suma los terminos del comando
            chksum = chksum + CmdRx(chkext)
        Next chkext
        chksum = chksum And &HFF
        chksum = 0 - chksum
        chksum = chksum And &HFF            ' checksum calculado

        If otravez > 2 Then
                Exit Sub
        End If

        If chksum <> CmdRx(14) Then         ' el comando no llego bien
            COMM1.Output = "N"
            otravez = otravez + 1
            GoTo OTRA
        Else
            COMM1.Output = "K"              ' el comando llego bien
            richtxtRx.Text = richtxtRx.Text & "Recibida Telemetria " & Time & Chr(10)
            stbarAyuda.Panels("mensaje").Text = "Recibido"
            stbarAyuda.Panels("mensaje").Picture = imglstRx.ListImages("telemetria").Picture
        End If

        If optBrujAcel.Value = True Then
            BRUJULA                         ' lecturas de la brujula
            ACELEROMETRO                    ' lect acelerometro
        ElseIf optGyro.Value = True Then
            valor = CmdRx(3)
            txtPX.Text = valor * 256 + CmdRx(2)  'lects gyro
            valor = CmdRx(5)
            txtPY.Text = valor * 256 + CmdRx(4)
            valor = CmdRx(7)
            txtPZ.Text = valor * 256 + CmdRx(6)
            valor = CmdRx(9)
            txtTX.Text = valor * 256 + CmdRx(8)
            valor = CmdRx(11)
```

```
            txtTY.Text = valor * 256 + CmdRx(10)
            valor = CmdRx(13)
            txtTZ.Text = valor * 256 + CmdRx(12)
        ElseIf optTodos.Value = True Then
            BRUJULA
            ACELEROMETRO
            valor = CmdRx(11)
            txtPX.Text = valor * 256 + CmdRx(10)  'lects gyro
            valor = CmdRx(13)
            txtPY.Text = valor * 256 + CmdRx(12)


            ' Se prepara a recibir comando
            COMM1.Output = "E"  ' solicita 2a trama
OTRA2:      saludo (14)         ' selecciona los primeros 14 bytes
            If FALLA = True Then
                TimerEspera.Enabled = False ' se acabo el tiempo
                FALLA = False
                Exit Sub
            Else
                If Len(trama) = 0 Then
                    richtxtRx.Text = richtxtRx.Text & "FALLA 2a TRAMA " & Time & Chr(10)
                    Exit Sub
                End If

                SeparaBytes trama, 14   ' introduce la trama y separa los datos en cmdRx
            End If

            chksum = 0
            For chkext = 2 To 13 Step 1           ' suma los terminos del comando
                chksum = chksum + CmdRx(chkext)
            Next chkext
            chksum = chksum And &HFF
            chksum = 0 - chksum
            chksum = chksum And &HFF            ' checksum calculado

            If otravez > 2 Then
                Exit Sub
            End If

            If chksum <> CmdRx(14) Then         ' el comando no llego bien
                COMM1.Output = "N"
                otravez = otravez + 1
                GoTo OTRA2
            Else
                COMM1.Output = "K"              ' el comando llego bien
                richtxtRx.Text = richtxtRx.Text & "Recibida 2a Telemetria " & Time & Chr(10)
                stbarAyuda.Panels("mensaje").Text = "Recibido"
                stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("telemetria").Picture
            End If
            valor = CmdRx(3)
            txtPZ.Text = valor * 256 + CmdRx(2)
            valor = CmdRx(5)
            txtTX.Text = valor * 256 + CmdRx(4)
            valor = CmdRx(7)
            txtTY.Text = valor * 256 + CmdRx(6)
            valor = CmdRx(9)
            txtTZ.Text = valor * 256 + CmdRx(8)
        End If
    End If


' CHARLA CON SENSORES DE ESTABILIZACION

    If chkTelemetria.Value = 1 And tabsModulos.SelectedItem.Index = 1 Then

        saludo (1)      ' espera N o K para ver si llego bien el comando
        If FALLA = True Then
            TimerEspera.Enabled = False ' se acabo el tiempo
            FALLA = False
```

```
            Exit Sub
        Else
            If Len(trama) = 0 Then
                richtxtRx.Text = richtxtRx.Text & "FALLA " & Time & Chr(10)
                Exit Sub
            End If

            SeparaBytes trama, 1
            Select Case Chr(CmdRx(1))
                Case "N"        ' seleccion de dato recibidos
                    richtxtRx.Text = richtxtRx.Text & "<N> Fallo " & Time & Chr(10)      'Si
recibe una N no se recibio el comando
                    stbarAyuda.Panels("mensaje").Text = "No llegó..."
                    stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("noentregado").Picture  ' coloca imagen en la barra de tarea
                    Exit Sub
                Case "K"
                    richtxtRx.Text = richtxtRx.Text & "<K> Recibido " & Time & Chr(10)
                    stbarAyuda.Panels("mensaje").Text = "Enviado"
                    stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("entregado").Picture
                    COMM1.Output = "E"  ' pide el envio
                Case Else
                    richtxtRx.Text = richtxtRx.Text & "<0x" & Hex(CmdRx(1)) & "> Dato
Desconocido " & Time & Chr(10)
                    stbarAyuda.Panels("mensaje").Text = "Dato falso..."
                    stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("falso").Picture
                    Exit Sub
            End Select
        End If
        richtxtRx.Text = richtxtRx.Text & "Esperando Telemetría ..." & Chr(10)
        ' Se prepara a recibir comando

OTRA3:  saludo (14)         ' selecciona los primeros 14 bytes
        If FALLA = True Then
            TimerEspera.Enabled = False ' se acabo el tiempo
            FALLA = False
            Exit Sub
        Else
            If Len(trama) = 0 Then
                richtxtRx.Text = richtxtRx.Text & "FALLA " & Time & Chr(10)
                Exit Sub
            End If

            SeparaBytes trama, 14   ' introduce la trama y separa los datos en cmdRx
        End If

        chksum = 0
        For chkext = 2 To 13 Step 1           ' suma los terminos del comando
            chksum = chksum + CmdRx(chkext)
        Next chkext
        chksum = chksum And &HFF
        chksum = 0 - chksum
        chksum = chksum And &HFF            ' checksum calculado

        If otravez > 2 Then
                Exit Sub
        End If

        If chksum <> CmdRx(14) Then         ' el comando no llego bien
            COMM1.Output = "N"
            otravez = otravez + 1
            GoTo OTRA3
        Else
            COMM1.Output = "K"              ' el comando llego bien
            richtxtRx.Text = richtxtRx.Text & "Recibida Telemetria " & Time & Chr(10)
            stbarAyuda.Panels("mensaje").Text = "Recibido"
            stbarAyuda.Panels("mensaje").Picture = imglstRx.ListImages("telemetria").Picture
        End If
```

```
        valor = CmdRx(2)
        valor = valor * 256 + CmdRx(3)
        corriente = (valor / 1024) - 0.3 ' 50 => ganancia en el max4071, (5-1.5V) en la ref
del voltaje
        txtTelemetria.Text = CStr(Round(corriente, 3)) & " [A] " & Time

    End If

    COMM1.Settings = "9600,N,8,1"     '9600 baud, Sin Paridad, 8 bits
    COMM1.RThreshold = 1              'Interrupción por Recepción
    COMM1.InputLen = 1                'Datos a recibir para interrumpir
    If COMM1.PortOpen = False Then COMM1.PortOpen = True
End Sub

Sub BRUJULA()
Dim bruj_med  As Double
Dim sumaux1, sumaux2 As Integer
    sumaux1 = CmdRx(2)
    sumaux1 = sumaux1 * 256
    sumaux2 = CmdRx(3)     'numero de 0 a 3599
    bruj_med = (sumaux1 + sumaux2) / 10
    ocxBrujula.Direction = bruj_med
    If bruj_med = 0 Then
        lblDireccion.Caption = "NORTE "
    ElseIf bruj_med < 90 And 0 < bruj_med Then
        lblDireccion.Caption = "NE " & CStr(bruj_med)
    ElseIf bruj_med = 90 Then
        lblDireccion.Caption = "ESTE "
    ElseIf bruj_med > 90 And bruj_med < 180 Then
        lblDireccion.Caption = "SE " & CStr(bruj_med)
    ElseIf bruj_med = 180 Then
        lblDireccion.Caption = "SUR "
    ElseIf bruj_med > 180 And bruj_med < 270 Then
        lblDireccion.Caption = "SO " & CStr(bruj_med)
    ElseIf bruj_med = 270 Then
        lblDireccion.Caption = "OESTE "
    ElseIf bruj_med > 270 And bruj_med <= 359.9 Then
        lblDireccion.Caption = "NO " & CStr(bruj_med)
    End If
End Sub

Sub ACELEROMETRO()

    txtVX.Text = CmdRx(4) * 256 + CmdRx(3)
    txtVY.Text = CmdRx(6) * 256 + CmdRx(5)
    txtVZ.Text = CmdRx(8) * 256 + CmdRx(7)

End Sub

Function saludo(longitud As Byte)

    COMM1.InputLen = longitud
    COMM1.InBufferCount = 0     ' limpia el buffer de entrada
    trama = ""                  ' limpia la trama
    TimerEspera.Enabled = True  ' comienza el timer de espera
    long_trama = longitud

    Do While Len(trama) <> longitud

        If FALLA = True Then
            richtxtRx.Text = richtxtRx.Text & "FALLA " & Time & Chr(10) ' hay falla
            TimerEspera.Enabled = False
            Exit Function
        End If

        DoEvents        ' atiende el timer
    Loop

    TimerEspera.Enabled = False

End Function
```

```
' tram => la cadena de datos, fin => longitud de la cadena
Function SeparaBytes(tram As String, fin As Byte)
Dim cuentatrama As Byte
    For cuentatrama = 1 To fin Step 1
        CmdRx(cuentatrama) = Asc(Mid(tram, cuentatrama, 1))
         ' transforma a decimal el valor del byte
    Next cuentatrama
End Function

Private Sub botonLimpiar_Click()
    richtxtRx = ""                  ' limpia la ventana de mensajes recibidos
End Sub

Private Sub chkBTMX_Click()         ' Habilita o deshabilita los controles de BTMX
    If chkBTMX.Value = 1 Then
        optbutXfina.Enabled = True
        optbutXfuerte.Enabled = True
        txtCTX.Enabled = True
        optX1.Enabled = True
        optX2.Enabled = True
    Else
        optbutXfina.Enabled = False
        optbutXfuerte.Enabled = False
        txtCTX.Enabled = False
        optX1.Enabled = False
        optX2.Enabled = False
    End If
End Sub

Private Sub chkBTMY_Click()         ' Habilita o deshabilita los controles de BTMY
    If chkBTMY.Value = 1 Then
        optbutYfina.Enabled = True
        optbutYfuerte.Enabled = True
        txtCTY.Enabled = True
        optY1.Enabled = True
        optY2.Enabled = True
    Else
        optbutYfina.Enabled = False
        optbutYfuerte.Enabled = False
        txtCTY.Enabled = False
        optY1.Enabled = False
        optY2.Enabled = False
    End If
End Sub

Private Sub chkBTMZ_Click()         ' Habilita o deshabilita los controles de BTMZ
    If chkBTMZ.Value = 1 Then
        optbutZfina.Enabled = True
        optbutZfuerte.Enabled = True
        txtCTZ.Enabled = True
        optZ1.Enabled = True
        optZ2.Enabled = True
    Else
        optbutZfina.Enabled = False
        optbutZfuerte.Enabled = False
        txtCTZ.Enabled = False
        optZ1.Enabled = False
        optZ2.Enabled = False
    End If
End Sub

Private Sub chkCambiarSens_Click()
    If chkCambiarSens.Value = 1 Then
        opt1G.Enabled = True
        opt2G.Enabled = True
        opt4G.Enabled = True
        opt6G.Enabled = True
    Else
        opt1G.Enabled = False
        opt2G.Enabled = False
```

```
        opt4G.Enabled = False
        opt6G.Enabled = False
    End If
End Sub

Private Sub chkControl_Click()    ' Habilita o deshabilita los controles de lazo abierto
    If chkControl.Value = 1 Then
        ComboControl.Enabled = True
        lblSeñal.Enabled = True
        Select Case ComboControl.Text
            Case ""
                txtFrecuencia.Enabled = False
                lblFrec.Enabled = False
            Case Else
                txtFrecuencia.Enabled = True
                lblFrec.Enabled = True
        End Select
    Else
        ComboControl.Enabled = False
        txtFrecuencia.Enabled = False
        lblSeñal.Enabled = False
        lblFrec.Enabled = False
    End If
End Sub

Private Sub chkMotor_Click()           ' Habilita o deshabilita los controles de motor
    If chkMotor.Value = 1 Then
        ComboMotor.Enabled = True
        lblOperacion.Enabled = True
        Select Case ComboMotor.Text
            Case "Fijar CT a motor"
                txtVelocidad.Enabled = False
                txtCT.Enabled = True
                optIzq.Enabled = True
                optDer.Enabled = True
                lblCiclo.Enabled = True
                lblSentido.Enabled = True
                lblVelocidad.Enabled = False
            Case "Velocidad"
                txtVelocidad.Enabled = True
                txtCT.Enabled = False
                optIzq.Enabled = True
                optDer.Enabled = True
                lblCiclo.Enabled = False
                lblSentido.Enabled = True
                lblVelocidad.Enabled = False
            Case "Cambia Velocidad"
                txtVelocidad.Enabled = True
                txtCT.Enabled = False
                optIzq.Enabled = False
                optDer.Enabled = False
                lblCiclo.Enabled = False
                lblSentido.Enabled = False
                lblVelocidad.Enabled = False
            Case Else
                txtVelocidad.Enabled = False
                txtCT.Enabled = False
                optIzq.Enabled = False
                optDer.Enabled = False
                lblCiclo.Enabled = False
                lblSentido.Enabled = False
                lblVelocidad.Enabled = False
        End Select
    Else
        lblOperacion.Enabled = False
        ComboMotor.Enabled = False
        txtVelocidad.Enabled = False
        txtCT.Enabled = False
        optIzq.Enabled = False
        optDer.Enabled = False
        lblCiclo.Enabled = False
```

```
        lblSentido.Enabled = False
        lblVelocidad.Enabled = False
    End If
End Sub

Private Sub chkPruebaGyros_Click()
    If chkPruebaGyros.Value = 1 Then
        fraPruebaGyros.Enabled = True
        GyroX.Enabled = True
        GyroY.Enabled = True
        GyroZ.Enabled = True
        txtXP.Enabled = True
        txtYP.Enabled = True
        txtZP.Enabled = True
        cmdInicioPrueba.Enabled = True
        cmdAltoPrueba.Enabled = True
    Else
        fraPruebaGyros.Enabled = False
        GyroX.Enabled = False
        GyroY.Enabled = False
        GyroZ.Enabled = False
        txtXP.Enabled = False
        txtYP.Enabled = False
        txtZP.Enabled = False
        cmdInicioPrueba.Enabled = False
        cmdAltoPrueba.Enabled = False
    End If
End Sub

Private Sub chkSensores_Click()
    If chkSensores.Value = 1 Then
        optBrujAcel.Enabled = True
        optGyro.Enabled = True
        optTodos.Enabled = True
        cmdBorraSens.Enabled = True
        If optBrujAcel.Value = True Then
            BrujulayAcelerometro        ' subrutinas de indicadores
        ElseIf optGyro.Value = True Then
            Gyro
        ElseIf optTodos.Value = True Then
            TodoSensores
        End If
    Else
        optBrujAcel.Enabled = False
        optGyro.Enabled = False
        optTodos.Enabled = False
        cmdBorraSens.Enabled = False
        NoSens              ' subrutina apaga indicadores
    End If
End Sub

Private Sub chkTelemetria_Click()
    If chkTelemetria.Value = 1 Then
        txtTelemetria.Enabled = True
    Else
        txtTelemetria.Enabled = False
    End If
End Sub

Private Sub cmdBorraMensaje_Click()
    txtMensaje.Text = ""
    txtMensaje.SetFocus
End Sub

Private Sub cmdAltoPrueba_Click()
    alto = True
    COMM1.Output = "G"
End Sub

Private Sub cmdBorraSens_Click()
    txtVX.Text = ""
```

```
    txtVY.Text = ""
    txtVZ.Text = ""
    txtPX.Text = ""
    txtPY.Text = ""
    txtPZ.Text = ""
    txtTX.Text = ""
    txtTY.Text = ""
    txtTZ.Text = ""
    lblDireccion.Caption = ""
    ocxBrujula.Direction = 0
End Sub

Private Sub cmdInicioPrueba_Click()
Dim ComandoPrueba(1 To 14) As Byte
Dim sumatotal As Integer

    alto = False
    reinicio = False

    ComandoPrueba(1) = Str(&H2F)        ' envia comando requiriendo prueba
    ComandoPrueba(2) = Str(&HB4)
    ComandoPrueba(3) = Str(&H41)
    ComandoPrueba(4) = Str(&H41)
    ComandoPrueba(5) = Str(&H41)
    ComandoPrueba(6) = Str(&H41)
    ComandoPrueba(7) = Str(&H41)
    ComandoPrueba(8) = Str(&H41)
    ComandoPrueba(9) = Str(&H41)
    ComandoPrueba(10) = Str(&H41)
    ComandoPrueba(11) = Str(&H41)
    ComandoPrueba(12) = Str(&H41)
    ComandoPrueba(13) = Str(&H41)
    ComandoPrueba(14) = Str(&H81)

    For i = 1 To 14 Step 1
        COMM1.Output = Chr(ComandoPrueba(i))      ' envia los comandos uno por uno
    Next i

    saludo (1)      ' espera N o K para ver si llego bien el comando
    If FALLA = True Then
        TimerEspera.Enabled = False ' se acabo el tiempo
        FALLA = False               ' reset a acondicion de falla
        Exit Sub
    Else
        If Len(trama) = 0 Then
            richtxtRx.Text = richtxtRx.Text & "FALLA " & Time & Chr(10)
            Exit Sub
        End If

        SeparaBytes trama, 1
        Select Case Chr(CmdRx(1))
            Case "N"        ' seleccion de dato recibidos
                richtxtRx.Text = richtxtRx.Text & "<N> Fallo " & Time & Chr(10)      'Si
recibe una N no se recibio el comando
                stbarAyuda.Panels("mensaje").Text = "No llegó..."
                stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("noentregado").Picture  ' hay dos formas de adquirir imagen
                Exit Sub
            Case "K"
                richtxtRx.Text = richtxtRx.Text & "<K> Recibido " & Time & Chr(10)
                stbarAyuda.Panels("mensaje").Text = "Enviado"
                stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("entregado").Picture  ' hay dos formas de adquirir imagen
                COMM1.Output = "E"  ' pide el envio
            Case Else
                richtxtRx.Text = richtxtRx.Text & "<0x" & Hex(CmdRx(1)) & "> Dato
Desconocido " & Time & Chr(10)
                stbarAyuda.Panels("mensaje").Text = "Dato falso..."
                stbarAyuda.Panels("mensaje").Picture = imglstRx.ListImages("falso").Picture
' hay dos formas de adquirir imagen
                Exit Sub
```

```
        End Select

        While alto = False
            saludo (13)         ' recibe los 13 bytes a leer de los gyros y acelerometro

            pausa = 0
            TimerPausa.Enabled = True       ' pausa para recolectar datos
            While pausa < 5
                DoEvents
            Wend
            TimerPausa.Enabled = False

            SeparaBytes trama, 13   'separa la trama en 13 y lo pone en el arreglo CmdRx
            sumatotal = CmdRx(2) + CmdRx(3) * 256
            txtXP.Text = sumatotal
            sumatotal = CmdRx(4) + CmdRx(5) * 256
            txtYP.Text = sumatotal
            sumatotal = CmdRx(6) + CmdRx(7) * 256
            txtZP.Text = sumatotal
            txtVX.Text = CmdRx(9) * 256 + CmdRx(8)
            txtVY.Text = CmdRx(11) * 256 + CmdRx(10)
            txtVZ.Text = CmdRx(13) * 256 + CmdRx(12)

            DoEvents
        Wend
    End If

End Sub

Private Sub ComboControl_Click()
    Select Case ComboControl.Text       ' habilita la asignacion de frecuencia
        Case ""
            txtFrecuencia.Enabled = False
            lblFrec.Enabled = False
        Case Else
            txtFrecuencia.Enabled = True
            lblFrec.Enabled = True
    End Select
End Sub

Private Sub ComboMotor_Click()

    Select Case ComboMotor.Text
        Case "Fijar CT a motor"                     ' fija CT a motor
            txtCT.Enabled = True
            txtVelocidad.Enabled = False
            optIzq.Enabled = True
            optDer.Enabled = True
            lblCiclo.Enabled = True
            lblSentido.Enabled = True
            lblVelocidad.Enabled = False
        Case "Velocidad"                      ' Velocidad
            txtCT.Enabled = False
            txtVelocidad.Enabled = True
            optIzq.Enabled = True
            optDer.Enabled = True
            lblCiclo.Enabled = False
            lblSentido.Enabled = True
            lblVelocidad.Enabled = True
        Case "Cambia Velocidad"                     ' Cambia velocidad
            txtCT.Enabled = False
            txtVelocidad.Enabled = True
            optIzq.Enabled = False
            optDer.Enabled = False
            lblCiclo.Enabled = False
            lblSentido.Enabled = False
            lblVelocidad.Enabled = True
        Case Else
            txtCT.Enabled = False
            txtVelocidad.Enabled = False
            optIzq.Enabled = False
```

```
            optDer.Enabled = False
            lblCiclo.Enabled = False
            lblSentido.Enabled = False
            lblVelocidad.Enabled = False
    End Select

End Sub

Private Sub COMM1_OnComm()
Dim j As Byte
Dim respuesta As String
tiempoespera = 0
j = 1

If tabsModulos.SelectedItem.Index = 1 Then
     ' almacena las tramas del comando
    If chkTelemetria.Value = 1 Or chkPruebaGyros.Value = 1 Then

        If long_trama > 1 Then
            trama = trama & COMM1.Input
        Else
            trama = COMM1.Input
        End If

    Else
        regRx = COMM1.Input           'Carga el valor en variable

        If regRx = "K" Then           'Si recibe una K se recibio correctamente
            richtxtRx.Text = richtxtRx.Text & "<" & regRx & "> Recibido " & Time & Chr(10)
            stbarAyuda.Panels("mensaje").Text = "Enviado"
            stbarAyuda.Panels("mensaje").Picture = imglstRx.ListImages("entregado").Picture
' muestra imagen en la barra

        ElseIf regRx = "N" Then
            richtxtRx.Text = richtxtRx.Text & "<" & regRx & "> Fallo " & Time & Chr(10)
'Si recibe una N no se recibio el comando
            stbarAyuda.Panels("mensaje").Text = "No llegó..."
            stbarAyuda.Panels("mensaje").Picture =
imglstRx.ListImages("noentregado").Picture

        ElseIf regRx = "V" Then
            richtxtRx.Text = richtxtRx.Text & "<" & regRx & "> Vel. correcta " & Time &
Chr(10)      'Si recibe una V al alcanzar la velocidad
            stbarAyuda.Panels("mensaje").Text = Val("&h" & velocidad) & " RPM"
            stbarAyuda.Panels("mensaje").Picture = imglstRx.ListImages("velocidad").Picture

        Else
            richtxtRx.Text = richtxtRx.Text & "<" & CStr(Hex(Asc(regRx))) & "> Dato
Desconocido " & Time & Chr(10)
            stbarAyuda.Panels("mensaje").Text = "Dato falso..."
            stbarAyuda.Panels("mensaje").Picture = imglstRx.ListImages("falso").Picture
        End If
        Beep
    End If
Else

        If long_trama > 1 Then
            trama = trama & COMM1.Input
        Else
            trama = COMM1.Input
        End If

End If

End Sub

Private Sub Form_Load()
' UNAM II proyecto SATEDU
' Software de prueba para subsistemas
' Jimenez Madrigal Emilio Augusto
' Iniciando puerto serial
```

```
    Set Borde.Forms = Forms
    COMM1.CommPort = 1                'COM1
    COMM1.Settings = "9600,N,8,1"     '9600 baud, Sin Paridad, 8 bits
    COMM1.RThreshold = 1              'Interrupción por Recepción
    COMM1.InputLen = 1                'Datos a recibir para interrumpir
    If COMM1.PortOpen = False Then COMM1.PortOpen = True
    For i = 1 To 7 Step 1
        dato_leido(i) = False
    Next i
    sensibilidad = 1       ' inicio del comando de sensibilidad
End Sub

Private Sub Form_Unload(Cancel As Integer)
    If COMM1.PortOpen = True Then COMM1.PortOpen = False
End Sub

Private Sub fraBTM_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    stbarAyuda.Panels("status").Text = ""
End Sub

Private Sub fraControl_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    stbarAyuda.Panels("status").Text = ""
End Sub

Private Sub fraMotor_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    stbarAyuda.Panels("status").Text = ""
End Sub

Private Sub opt1G_Click()
sensibilidad = 1   'sens 1.5g
End Sub

Private Sub opt2G_Click()
sensibilidad = 2  'sens 2g
End Sub

Private Sub opt4G_Click()
sensibilidad = 3  'sens 4g
End Sub

Private Sub opt6G_Click()
sensibilidad = 4  'sens 6g
End Sub

Private Sub optBrujAcel_Click()
    BrujulayAcelerometro
End Sub

Private Sub optGyro_Click()
 Gyro
End Sub

Private Sub optTodos_Click()
    TodoSensores
End Sub

Private Sub tabsModulos_Click()
    If tabsModulos.SelectedItem.Index = 1 Then
        fraControl.Visible = True
        fraMotor.Visible = True
        fraBTM.Visible = True
        fraSensores.Visible = False
        txtComando.Text = com_motor     ' carga el anterior
    ElseIf tabsModulos.SelectedItem.Index = 2 Then
        fraControl.Visible = False
        fraMotor.Visible = False
        fraBTM.Visible = False
        fraSensores.Visible = True
        txtComando.Text = com_audio     ' carga el anterior
```

```
        com_audio = txtComando.Text     ' guarda antes de borrar
    End If
End Sub

Private Sub TimerEspera_Timer()
    tiempoespera = tiempoespera + 1   'Incrementa contador
    If tiempoespera > 10 Then
        FALLA = True
    End If
End Sub

Private Sub TimerPausa_Timer()
    pausa = pausa + 1
End Sub

Private Sub txtCT_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    stbarAyuda.Panels("status").Text = "CT en 14 bits de 0 a 16383 como máximo"
End Sub

Private Sub txtCTX_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    stbarAyuda.Panels("status").Text = "CT en 14 bits de 0 a 16383 como máximo"
End Sub

Private Sub txtCTY_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    stbarAyuda.Panels("status").Text = "CT en 14 bits de 0 a 16383 como máximo"
End Sub

Private Sub txtCTZ_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    stbarAyuda.Panels("status").Text = "CT en 14 bits de 0 a 16383 como máximo"
End Sub

Private Sub txtFrecuencia_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    stbarAyuda.Panels("status").Text = "Frecuencia de la señal variable hasta 15 Hz"
End Sub

Private Sub txtTelemetria_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    stbarAyuda.Panels("status").Text = "TELEMETRIA Corriente en el motor"
End Sub

Private Sub txtVelocidad_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    stbarAyuda.Panels("status").Text = "Velocidad minima de 600 RPM"
End Sub
```