UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – TELECOMUNICACIONES

"DESARROLLO DEL SUBSISTEMA DE ESTABILIZACIÓN BASADO EN RUEDAS INERCIALES Y BOBINAS DE TORQUE PARA EL SATÉLITE EDUCATIVO (SATEDU)."

MODALIDAD DE GRADUACIÓN: TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
DIERK LUEDERS MONSIVAIS

TUTOR PRINCIPAL
DR. ESAÚ VIVENTE VIVAS
INSTITUTO DE INGENIERIA

MÉXICO, D. F. Mayo, 2013

Jurado Asignado

Presidente:	Dr. Tinoco Magaña Julio César
Secretario:	Dr. Landeros Ayala Salvador
Vocal:	Dr. Vicente Vivas Esaú
1 er. Suplente:	Dr. Martinez López José Ismael
2 d o. Suplente:	Dr. Rangel Licea Víctor

Lugar donde se realizo la tesis:

Coordinacion de electrónica y computación, Laboratorio de Sistemas Aereoespaciales, Instituto de Ingenieria, UNAM, México.

TUTOR DE TESIS
Dr. Esaú Vicente Vivas

FIRMA

Dedicatoria

Al Universo y sus componentes fotones, electrones, protones, neutrones, neutrinos, quark up, down, higgs, etc., que componen el todo y a la gravedad por ser la causa de que se haya formado nuestra Galaxia y nuestro sistema solar que permitió que existiera un planeta dónde se diera la vida.

A las ondas electromagnéticas por ser la base de esta Maestría en Ingeniería Eléctrica en Telecomunicaciones. Gracias a los átomos que me componen, a las moléculas y cadenas de ADN que me hacen ser quien soy y estar aquí.

A mis padres Laura y Dierk, por haberme dado la vida. En especial a mi madre Laura Monsivais por ser una mujer luchadora que siempre ha estado presente para apoyarme, comprenderme y educarme a lo largo de toda mi vida. Gracias a su solidaridad y consejos, he llegado a realizar una de mis grandes metas. Lo cual constituye la herencia más valiosa que pudiera recibir.

A mis hermanos Ken y Ayari por compartir tantos momentos y experiencias juntos; por su compañía y apoyo, dándome fuerza para superarme, y poder ser un mejor ser humano. Ellos a quienes jamás encontraré la forma de agradecer el cariño y comprensión brindados en los momentos buenos y malos de mi vida.

A mis abuelos Guillermo Monsivais y Carmen Galindo por todo su cariño y animos en todo momento quienes siempre creyeron en mí y fueron un gran pilar en la familia.

A mi tío Guillermo por ser una fuente de inspiración para seguir en el ámbito universitario, y a mi tía Erika. A mis tíos Jorge, Raúl y en especial a Patricia, que además de su apoyo, ha sido una gran madrina buscando convertirme en un hombre de provecho.

A mis primos Ian, Manuel, Omar, Daniel, Karla, Bibi, Gus, Gonzalo por los alientos, la amistad, el apoyo y el cariño a lo largo de mi carrera y mi vida.

A César Balcázar por haberme enseñado tantas cosas. Y a los que me acompañaron en el teatro: Héctor García, Carlos Ortiz, Armando Souza, Hixem, Jairo, Zoraida y Eleazar Gómez.

A mis compañeros de proyecto del Instituto de Ingeniería: Dr. Esaú Vicente Vivas, Mario Alberto Bárcenas, Paul Domínguez, Rodrigo Córdova, Ignacio Mendoza, Rodrigo Alva, Eduardo Vizcaíno, Francisco Osorio (Enkor), Paloma Pedrajas, Alberto Ramírez, Mario Alberto Hernández, Alejandro Córdoba, Luis Zepeda, Erika Mendoza, Genaro Islas, Alejandro Castilla, Miguel Alvarado, Eddy Vino, pero en particular y sobre todo a Emilio Jiménez que, sin sus enseñanzas y conocimiento, no sería posible esta tesis.

A mis amigos Ismael Mendoza y Fabián Fernández por los gratos momentos en la facultad y la apreciación musical. A Mauricio García que conozco desde la preparatoria. A mis amigos de la facultad Argelia Rosales, Alejandro Enríquez, Esteban Arrangóiz, que además de amigos son mis colegas profesores en la Facultad de Medicina, así como al Dr. Israel

Martínez, a quienes he tenido el honor de conocer, deseándoles que tengan todo el éxito en sus vidas.

A mis compañeros y amigos de la Maestría: Helio Gonzales, Hugo Martínez, Jovanni Silva, Antonio Uribe y Vero Rivera, por haber compartido este viaje de conocimiento durante la Maestría.

A mis amigos de toda la vida con quienes he recorrido tantos caminos de diferentes lugares desde la Preparatoria Num.6, como: Nasser, Ranyus, Choke, Rubén Montedorico, Eduardo Rojas, Rodrigo “Dhink” Padilla, León Francisco Coronado, Juan Carlos “Montana” y toda la familia Mictlán , a Pavel Granados, Edgar Valero, sabiendo que jamás encontraré la forma de agradecer su constante apoyo y confianza, sólo espero que comprendan que mis ideales, esfuerzos y logros han sido también suyos e inspirados en ustedes.

También a Diana y Ximena Fernández, mi mejor amiga, a Irene, Teresa y Christina Due, a Alicia Maldonado, a los que han trabajado en Las Yardas, así como en el Mala Idea, al “Puma” Valdez, Carlos Antonio “Dengue”, al colectivo Colmena, Andrés Ávila, Javier, Enrique y Diana Bastida, Maritania, Renata Rendón, Pablo Tonda, Fernando Kruxty, Claudia Domínguez, Fernanda Rojas, a quienes jamás encontraré la forma de agradecer el que me hayan brindado su mano en las derrotas y logros de mi vida, haciendo de este triunfo más suyo que mío por la forma en la que guiaron mi vida con amor y energía.

Y a todos aquellos amigos y compañeros con los que pude he podido convivir, compartir y aprender . Por todos estos años de amistad, diversión, lucha, comprensión y apoyo moral en cada instante que hemos pasado juntos. A todas esas personas que me han animado y apoyado, impulsándome a concluir esta etapa de mi vida, y que comparten conmigo este logro.

A Sherkhar, Katy y Freide compañeros entrañables.

Y a esas magnificas vidas que, directa o indirectamente, me han ayudado en forma moral y económica para mi formación, como ser humano, y en el ámbito profesional.

Agradecimientos

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería por la oportunidad de formar parte de este recinto que es la máxima casa de estudios y todas las oportunidades que me brindaron para no sólo ser un buen profesionista si no también un buen ser humano y poderme desarrollar, personal, académica profesionalmente.

Al mi asesor y amigo el Dr. Esaú Vicente Vivas por la oportunidad de participar dentro de este proyecto. Por sus consejos y paciencia dentro y fuera del desarrollo de esta tesis; pero sobre todo por su confianza, apoyo incondicional y tolerancia para que terminara esta tesis.

A mis compañeros y amigos del Instituto de Ingeniería especialmente a Mario Alberto Mendoza, Rodrigo Córdova, Paul Domínguez y Emilio Jiménez, por sus consejos, apoyo, aportaciones y participación en este proyecto.

Al todo el departamento de Informática Biomédica de la Facultad de Medicina, UNAM, en especial al Dr. Israel Martínez Franco y al Ing. Fabián Fernández, por haberme dado la oportunidad de ser profesor de asignatura de esa división de estudios profesionales y poder desempeñarme como académico de la UNAM, lo cual me llena de orgullo.

A los miembros del jurado, los doctores Julio César Tinoco Magaña, Salvador Landeros Ayala, José Ismael Martínez López y Víctor Rangel Licea por sus comentarios y críticas que engrandecen este trabajo y por revisar satisfactoriamente otorgando los votos a favor del mismo.

A mis profesores de la facultad porque de ellos aprendí a ser Ingeniero. A mis profesores del Posgrado de la Facultad de Ingeniería por compartir sus experiencias y conocimiento.

Al Posgrado y al Instituto de Ingeniería porque gracias a ellos puede crecer en todos los aspectos de mi persona.

"Limitar nuestra atención a cuestiones terrestres sería limitar el espíritu humano."
Stephen Hawking

"Si estamos solos en el Universo, seguro que sería una terrible pérdida de espacio"
Carl Sagan

Resumen

Se presenta el diseño y el desarrollo del subsistema de estabilización basado en Ruedas Inerciales (RIs) y Bobinas de Torque Magnético (BTMs) para el satélite educativo (SATEDU), satélite para la formación de recursos humanos en el área satelital, bajo proceso de patente diseñado, fabricado y validado en el Instituto de Ingeniería de la UNAM.

El subsistema de estabilización es capaz de controlar tres ruedas inerciales y 6 bobinas de torque magnético, y sustituye a la Computadora de Vuelo (CV).

La CV puede recibir los datos de una tarjeta de sensores que está compuesta por un giróscopo triaxial, un acelerómetro triaxial y un magnetómetro triaxial, capaces de tomar lecturas de su orientación actual.

Se utiliza una mesa suspendida en aire (MSA) como plataforma de simulación que genera un medio con fricción despreciable, simulando el ambiente espacial. Las ruedas inerciales están colocadas en los tres ejes ortogonales de la plataforma, están constituidas por discos metálicos sujetos a los ejes de motores de corriente directa.

Se presenta también el Filtro de Kalman Unscented y su implementación para filtrar las señales de los sensores de navegación.

Se presenta adicionalmente, el diseño y desarrollo de los circuitos impresos para las tarjetas electrónicas, así como una descripción de los componentes electrónicos que las componen. Se describen las herramientas de programación utilizadas en el microprocesador y se presentan las pruebas realizadas a las tarjetas impresas.

Para el diseño del nuevo modelo de Computadora de Vuelo (CV) se retomaron experiencias y trabajo de tesis previas realizadas en 2007 y en 2009 para la Plataforma Satelital SATEDU, con los siguientes resultados:

Primero, se propuso un nuevo diseño en hardware para la CV que provee al subsistema de interfaces adecuadas para realizar la conectividad con la plataforma SATEDU, pero con recursos actualizados suficientes que permiten su aplicación directa para manejar la ruedas inerciales, además de su interacción con diferentes recursos periféricos.

Segundo, se rediseñó el *firmware* original de la CV de SATEDU, para ejecutarlo en un procesador de Texas Instruments Stellaris de 32 bits, el cual es el núcleo del nuevo prototipo de CV.

Abstract

This thesis shows the design and validation of the Stabilization subsystem, based on Inertial Wheels (IW) and Magnetic Torque Coils (MTC) for the Educative Satellite Platform called SATEDU, which supports human resources formation in the satellite field. SATEDU was designed, manufactured and validated at the Institute of Engineering UNAM and at present time is under patent process at the same institution.

The Stabilization subsystem allows controlling three IW and 6 MTC, and replacing the previous Flight Computer (FC) Subsystem.

The FC is able to receive data from an inertial navigation sensor board which contains a set of accelerometers, gyroscopes and magnetometers, all of them in three axes that allow data acquisition from the current orientation.

An air bearing system (ABS) is used as a simulation tool, as it allows a cost-effective simulation of free space frictionless conditions to perform attitude control, testing and validation in terrestrial laboratories. The IW are integrated into the ABS platform with basic equipment to perform the referred tests, and placed in orthogonal axes. The IW are metallic discs drived by Direct Current motors.

The thesis work also describes the Unscented Kalman Filter (UFC) to estimate and improve the orientation signals of the navigation sensors.

This thesis also presents the design and development of the new hardware boards, and its associated electronic components description. The programation tools used for the processor are described as well. The validation testing process for these boards is also shown.

The design of the new model of FC took advantage of previous thesis work performed in 2007 and 2009 for SATEDU satellite. This achieved the following results:

First, it was proposed a new FC hardware design which allows SATEDU to integrate better connectivity interfaces. In addition it will add adequate resources for the operation of the three IW, and its peripheral subsystems.

Second, the original SATEDU FC firmware was redesigned to run on a Stellaris Texas Instruments 32-bit processor, which is the core system for the new FC.

Indice

DESARROLLO DEL SUBSISTEMA DE ESTABILIZACIÓN BASADO EN RUEDAS INERCIALES (RI) Y BOBINAS DE TORQUE (BT) PARA EL SATÉLITE EDUCATIVO (SATEDU).

Indice de figuras

Indice de tablas

Capítulo 1. Introducción

1.1 Objetivo de la tesis

La presente tesis comprende el diseño, construcción y validación funcional del hardware y software del subsistema de estabilización del satélite educativo SATEDU, que ha desarrollado el Instituto de Ingeniería de la UNAM. El subsistema cuenta con actuadores activos y con el hardware de soporte necesario para el manejo de 3 ruedas inerciales (RIs) y 3 bobinas de torque magnético (BTMs). El elemento de cómputo y gestión de control a bordo del subsistema de estabilización está basado en un microcontrolador MCU LM4F2315QHr Stellaris ARM cortex 4, que le da gran flexibilidad para resolver problemas de estabilización satelital y le otorga amplios recursos actualizados a la nueva computadora de vuelo que se desarrolló en esta tesis.

El sistema que se desarrolló en esta tesis está contenido en 2 tarjetas, una que contiene el MCU LM4F2315QHR Stellaris ARM cortex 4, un Microcontrolador para reprogramación y un banco de memoria flash de 4 Mb. En tanto que la segunda contiene principalmente la parte de potencia (puentes H, reguladores, elementos pasivos, etc.) para manejar 3 ruedas inerciales externas, conectores al exterior, así como una etapa de reguladores de voltaje tanto para las ruedas inerciales como para tarjetas externas que forman parte de la instrumentación del satélite SATEDU.

Esta configuración de dos tarjetas hará que la segunda se pueda usar tanto en versiones anteriores de SATEDU como en versiones nuevas, o en otras aplicaciones que la usen como periférico externo, entre ellos, sistemas de desarrollo FPGA como la Spartan 3E o la Spartan 6.

Otro objetivo de esta tesis fue implementar al MCU Stellaris un método de filtrado para las señales de los sensores de orientación y navegación inercial. Estos sensores ayudan a determinar la orientación del satélite con base en tecnologías MEMS (Micro Electro Mechanical Systems). El filtrado implantado fue el filtro de Kalman Unscented, que es un método de filtrado ampliamente validado y usado en gran cantidad de tareas, el cual fue propuesto por un integrante del Grupo de Desarrollo de Sistemas Aéreoespaciales del Instituto de Ingeniería en su tesis de maestría.

1.2 Definición del problema

El IINGEN-UNAM ha desarrollado durante los últimos años un sistema de bajo costo para el entrenamiento de recursos humanos en técnicas y temas de tecnología satelital, proyecto que finalizó exitosamente en una primera etapa en Enero de 2009 y que continúa actualizándose de forma permanente en todos sus subsistemas. SATEDU emplea partes de uso comercial, debido a que no se pretende orbitarlo en el espacio; debe señalarse que este equipo se emplea a su vez para generar proyectos alternos de tecnología y conocimiento en

el campo satelital. Adicionalmente, se pretende que SATEDU sea empleado en centros educativos (universidades, tecnológicos, etc.) para acercar el tema tecnológico satelital no solo a la juventud, sino también a personal docente y de investigación en el campo satelital y tecnologías de la información.

La primera versión de SATEDU sólo tiene una rueda inercial, por lo cual sólo puede emplearse para realizar pruebas de estabilización en un solo eje, sin embargo, para que el sistema se utilice en temas de desarrollo de tecnología satelital avanzada, se requiere de una nueva versión que permita controlarlo en tres ejes, es decir, que contenga 3 ruedas inerciales. Este es precisamente el objetivo de la presente tesis, desarrollar el nuevo subsistema de estabilización para la nueva versión de SATEDU, lo cual dará pauta para generar posteriormente tecnología satelital estabilizada en tres ejes para uso en órbita espacial.

Se pretende además, que el prototipo de la tarjeta de estabilización que se desarrolle en esta tesis continúe preservando las características de SATEDU, es decir, que continúe siendo amigable para utilizarlo en laboratorios y que sea así mismo, suficientemente flexible y versátil para ofrecer cursos de entrenamiento y aprendizaje (atractivos, modulares y rápidos) en el campo satelital. De este modo, en esta tesis se persiguió desarrollar un subsistema de una nueva versión de SATEDU, que continúe acercando a las nuevas generaciones al mundo apasionante de los satélites, del espacio y de las tecnologías de la información, y en general, que continúe apoyando la difusión y el avance de La Ciencia y La Tecnología en México.

SATEDU contiene todos los subsistemas de un satélite comercial, pero en tamaño reducido, incluyendo: el estructural, de potencia, de comunicaciones, de computadora de vuelo, de estabilización, de sensores de estabilización, de sensores de plataforma y la carga útil.

Particularmente, el subsistema de estabilización que se propone desarrollar en esta tesis se diferencia de las versiones anteriores en que esta nueva versión tendrá las 3 ruedas inerciales incorporadas tanto en su hardware como en su software de estabilización. Este sistema permitirá que la nueva versión de SATEDU, pueda emplearse para validar modelos de estabilización satelital en 3 ejes, así como para validar algoritmos de control de apuntamiento. Esta característica hará único a nivel internacional al sistema SATEDU, debido a que el único sistema de entrenamiento satelital comercial (estadounidense) que existe en el mundo no incorpora este tipo de funciones.

Para validar la operación del subsistema desarrollado en esta tesis se empleó una plataforma de simulación satelital suspendida en aire Mesa Suspendida en Aire (MSA), la cual fue desarrollada en el Instituto. Cabe señalar que para realizar este tipo de validación, se empleó el siguiente equipo:

- El prototipo SATEDU con sus dos nuevas tarjetas de estabilización. Ello incluye computadora de vuelo, comunicaciones inalámbricas, sensores de navegación inercial y subsistema de potencia.
- La plataforma de simulación satelital o Mesa Suspendida en Aire (MSA).

- Tres RIs instaladas en la MSA, conectadas a SATEDU.
- Una batería de tecnología LiPo de alta potencia instalada en la MSA.

Para la implementación del filtro de Kalman Uncented (FKU) se utilizó la tarjeta de sensores de la compañía *Sparkfun,* la cual contiene tres clases de sensores requeridas para experimentar con métodos de estimación de orientación. La tarjeta integra los siguientes sensores:

- Magnetómetro triaxial (en tres ejes).
- Acelerómetro triaxial (en tres ejes).
- Giróscopo triaxial (en tres ejes).

Debido a que ésta tesis se enfoca sólo a generar la instrumentación básica para realizar pruebas de estabilización en 3 ejes, no resulto necesario tomar en cuenta todas las consideraciones necesarias para vuelo orbital (en cuanto al uso transformaciones para hacer cambios de coordenadas y en cuanto al magnetómetro, al no considerar un modelo de campo magnético terrestre debido que al hacer pruebas en Tierra se supone que el campo magnético es fijo). Es decir, los requerimientos del FKU se reducen notablemente para los propósitos de esta tesis.

1.3 Justificación

El equipo desarrollado en esta tesis será empleado tanto en el satélite educativo SATEDU, que ya fue sometido a patente en septiembre de 2012 por el IINGEN-UNAM, para entrenar a recursos humanos en todo el país. Y se espera que pueda emplearse en uno de los primeros satélites pequeños de manufactura totalmente nacional, que serán fabricados dentro del esquema de trabajo de la Agencia Espacial Mexicana.

Estos satélites que empezará a fabricar nuestro país estarán orientados a obtener imágenes de todo nuestro territorio para realizar estudios de percepción remota, con énfasis en la atención a todo tipo de desastres naturales que ocurren en nuestro país. Entre ellos inundaciones, huracanes terremotos, incendios, mancha urbana, desastres ecológicos, contaminación ambiental, etc.

Por estas razones esta infraestructura de alta tecnología mexicana será de gran ayuda no solo para la Ciudad de México sino para todo el país.

1.4 Contribución y Metodología utilizada

La principal contribución del presente trabajo de tesis, constituye la actualización de la computadora de vuelo tanto del SATEDU, como del nanosatélite que actualmente desarrolla el IINGEN-UNAM. Esta computadora realizará las funciones de automatización del subsistema de estabilización y control de apuntamiento satelital, por lo cual integra todos los recursos necesarios para controlar digitalmente actuadores activos como son Ruedas Inerciales y Bobinas de Torque Magnetico. De esta forma, la presente tesis propone que esta computadora de vuelo realice las operaciones globales del satélite y que adicionalmente

pueda realizar automáticamente las funciones de estabilización y control de apuntamiento satelital.

El presente trabajo de tesis solo aborda el diseño y desarrollo de hardware y de software básico de validación, por lo que otro tipo de funciones quedan como trabajo subsecuente para nuevas tesis de maestria. Particularmente, la nueva computadora resuelve importantes problemas que se presentaban en la computadora de vuelo de la primera y segunda versiones de SATEDU, entre los cuales destacaban:

- Alto consumo de energía.
- Poca diversidad de periféricos.
- Medios de comunicación y red obsoletos.
- Nula posibilidad de incluir un sitema operativo en tiempo real.
- Poca capacidad de almacenamiento de datos (tanto temporal como permanente).
- La necesidad de multiplexar canales de comunicación serial ante la imposibilidad de usar comunicaciones tipo I²C, SPI, etc.
- El uso de conectores de costilla con amplio numero de líneas, lo cual representaba espacio y peso adicional en el satélite. Este aspecto es de vital importancia para el satélite real que fabricara la UNAM, en vista de que se lograra un ahorro significativo en su masa final y por tanto impactará el costo final de lanzamiento espacial.

En cuanto a la metodología empleada, esta contempló las siguientes actividades y metas durante el desarrollo de la tesis:

Se diseñaron dos tarjetas empleando software Altium Protel tanto el la parte de circuitos esquematicos como de circuitos impresos. La primera de ellas constituye la computadora de vuelo del satélite, que integra al MCU Stellaris un microcontrolador de 32 bits de Texas Instruments. La segunda fue una tarjeta de potencia que permite manejar 3 ruedas inerciales cuando es controlada con señales de salida PWM provenientes del MCU Stellaris.

Para efectos de validación del hardware y software desarrollados en esta tesis, se emplearon diversos recursos disponibles en nuestro laboratorio, entre ellas tres ruedas inerciales con sus respectivos motores de corriente directa y bobinas de torque magnetico. Adicionalmente, se empleo la mesa suspendida en aire MSA que ha instrumentado el IINGEN en los últimos tres años. Con estos recursos se validaron los sistemas desarrollados en esta tesis, cuyos resultados se presentan en el ultimo capitulo.

Se fabricó, ensambló, validó y se realizaron pruebas a la tarjeta de potencia definida en el primer punto.

En tanto que el diseño completo de la computadora de vuelo quedo concluido y actualmente el IINGEN esta por mandar a manufacturar tanto el circuito impreso como los componentes electrónicos respectivos.

De igual forma se desarrollo el software necesario para validar las operaciones tanto de la computadora de vuelo como de la tarjeta de potencia que maneja a las ruedas

inerciales. Particularmente se utilizo el software Code Composer Studio (CCS) para diseñar, desarrollar y verificar el software elaborado para el MCU Stellaris de la computadora de vuelo.

Se realizaron bastantes pruebas operativas y exitosas del funcionamiento de las ruedas inerciales empleando la tarjeta de potencia desarrollada en esta tesis y el software de la computadora de vuelo, el cual se emuló preliminarmente con un sistema de desarrollo comercial DSP. Para este propósito, se empleó también la herramienta CCS, la cual permite generar código para diversas plataformas de Texas Instruments.

De forma más detallada, en seguida se presenta la metodología que se siguió en los tres semestres de investigación que llevaron a la conclusión de la presente tesis.

Trabajo de Investigación I

Completar la familiarización con la arquitectura global del proyecto SATEDU. Se adquirirá conocimiento de los mecanismos para incorporar nuevas interfaces y subsistemas al SATEDU. Se realizará una investigación de publicaciones relacionadas, profundizando en los temas de ruedas inerciales, su electrónica de potencia y bobinas de torque magnético.

Familiarizarse con diversos aspectos relacionados al tema de FPGAs, sistemas de desarrollo tales como Spartan 3E de Xilinx y la suite de diseño ISE (desarrollo de arquitecturas en VHDL, diseño e integración de sistemas embebidos, simuladores de desempeño de arquitecturas).
Realizar el diseño preliminar de las dos tarjetas de estabilización, así como elaborar la estrategia y diseño de los medios para subir nuevos programas al FPGA.

Redactar el objetivo, definición del problema e introducción de la tesis, así como los capítulos 1 y 2.

Trabajo de Investigación II

Realizar el diseño final de los circuitos esquemáticos de las dos tarjetas electrónicas de estabilización referidas anteriormente.

Realizar el diseño final de los circuitos impresos de las dos tarjetas electrónicas de estabilización referidas anteriormente.

Ensamblar, verificar y realizar primeras pruebas operativas a las 2 tarjetas de estabilización de SATEDU.

Redactar los capítulos de la tesis que estén involucrados con las metas y trabajo planeado para el trabajo de investigación II.

Trabajo de Investigación III

Elaborar el software básico para el control de los motores de CD para las ruedas inerciales y el software básico de operación para las bobinas de torque magnético por medio de PWM. Verificar y validar dicho software, con énfasis en su interacción con el software de operación satelital y el software de estación terrestre, tanto para el sistema SATEDU.
De igual forma se analizarán aspectos básicos de software para poder reprogramar al FPGA.

La validación funcional básica de las tarjetas de estabilización se realizará con el SATEDU montado en la plataforma de simulación suspendida en aire para observar el movimiento de la plataforma con la acción de los actuadores referidos.

Redactar los capítulos de la tesis relacionados con la elaboración del software propuesto, y con las etapas de validación y pruebas de los subsistemas desarrollados, así como los resultados obtenidos y las conclusiones.

1.5 Estructura de la tesis.

En el primer capitulo se da una intruduccion al trabajo de desarrollo de tecnología satelital que realiza el IINGEN. En este modulo también se plantea el problema de estabilización y control de apuntamiento satelital que se aborda en dicha institución, haciendo énfasis en las necesidades de hardware y software que se requieren para resolver dicho problema y por tanto para definir los objetivos de trabajo de la presente tesis, con lo cual se hace una justificación del tema de trabajo.

En el capitulo dos se describen con detalle los subsistemas que componen al satélite educativo SATEDU que ha desarrollado y que se encuentra patentando el IINGEN. A su vez se subraya la necesidad de integrarle nuevos recursos de estabilización, los cuales forman parte del trabajo de la presente tesis, haciendo énfasis en la reducción de costos como base global de trabajo.

El tercer capitulo trata la revisión del estado del arte en los Cubesats. Se presentan las ideas mayores en el estado del arte (el análisis viene un poco más tarde), hasta las propias ideas personales (pero sin incluirlas). Esta sección esta organizada por *idea* y no por autor o por publicación. Se presentan también las ruedas inerciales, la bobina de torque magnetico, la mesa suspendida en aire y los sensores de navegación inercial.

En el capitulo cuatro se describe el subsistema de estabilización activa en tres ejes por ruedas inerciales y bobinas de torque magnetico. Se ven diferentes técnicas de estabilización en satélites en tres ejes, desde estabilización activa a pasiva.

En el capitulo cinco se detalla el estimador utilizado en el MCU Stellaris el cual es el Filtro de Kalman Unscented. Se presenta la historia del filtro asi como su análisis matematico.

El capitulo seis trata el desarrollo de las tarjetas de estabilización activa en tres ejes para SATEDU. Se describe como funciona la adquisición de datos de los sensores, asi como la arquitectura del hardware de control de estabilización por ruedas inerciales y bobinas de torque en la MSA. Se denota la arquitectura electrónica tanto de la tarjeta de reguladores y potencia como de la tarjeta de la nueva CV, con la MCU Stellaris. También en este capitulo se ve la manufactura, ensamble y pruebas a las tarjetas.

En el capitulo siete se puede como se implemento el software básico de operación que fue desarrollado para las tarjetas de estabilización activa. Se describe el funcionamiento de un bootloader, de como se generan señales de PWM, asi como de la implementación del Filtro Square-Root Unscented Kalman Filter (SR-UKF).

El capitulo ocho trata sobre las pruebas de validación y resultados obtenidos. En tanto que en el capitulo nueve se pueden ver las conclusiones asi como recomendaciones para trabajos futuros.

Al final se provee un apéndice con los programas utilizados asi como un glosario de los acrónimos utilizados.

Capítulo 2. Antecedentes: Orígenes del Satélite Educativo, SATEDU ver1

2.1 Introducción

México cuenta con servicios satelitales desde 1968, cuando se construyó la primera estación terrena, en el estado de Hidalgo, con el objetivo de llevar a cabo la transmisión a color de los Juego Olímpicos de verano celebrados en México; y a partir de entonces, México cuenta con estos servicios, principalmente para enlaces transatlánticos.

En 1981 estableció un sistema doméstico, con ayuda de un satélite extranjero. Este sistema llevó a México a adquirir su propio sistema satelital, siendo el 17 de junio de 1985 la fecha en que se lanza el primer satélite mexicano: Morelos I, un HS 376 construido por la empresa Hughes Aircraft Corp., (actualmente Boeing) y llevado hasta su órbita (en arco geoestacionario a 113.5º de longitud oeste) por la NASA en la misión tripulada 51-G, habiéndose construido para entonces, el Centro de Control Satelital de Iztapalapa en México D.F. Posteriormente sería lanzado el Moleros II, operando a 116.8º de longitud oeste. Representando todo lo anterior, el principio de un sistema satelital doméstico (Satmex), administrado y operado por la Secretaria de Comunicaciones y Transportes (SCT), que luego se separo en la empresa estatal Telecomm (Telecomunicaciones de México).

De 1985 a 1997, se construyeron los satélites Solidaridad 1 y 2, y Morelos III (o Satmex 5), y se llevó a cabo la construcción del Centro de Control en Hermosillo, Sonora. En noviembre de 1997 la alianza Loral Space & Communications y Principia adquieren el 75% del capital de la sección de operación satelital de Telecomm: Satmex. El 5 de diciembre de 1998 es lanzado el Satmex 5 desde Kourou, Guyana Francesa.

Actualmente, Satmex opera 3 satélites: Solidaridad 2, Satmex 5 y Satmex 6. El sistema Solidaridad 1 fue lanzado el 19 de noviembre de 1993, y Solidaridad 2 el 7 de octubre de 1994, ambos puestos en órbita en un vehículo de lanzamiento Ariane 4, desde la plataforma espacial de Arianespace en Kourou, Guyana Francesa. En la banda Ku, los Solidaridad cuentan con 16 transpondedores de 54 MHz, dando servicio a un amplio rango de aplicaciones, desde broadcast hasta InterNet, con cobertura de América Central, regiones del sur de Estados Unidos y del norte de Sudamérica. Los satélites cuentan con un poder de transmisión de hasta 37.5 dBW. Solidaridad 1 y 2 operan a 109.2º y 113º de Longitud Oeste respectivamente. En marzo de 2006 el satélite Solidaridad 2 se migró a una órbita inclinada a los 114.9º oeste para alargar su vida útil hasta 2013.

Satmex 5 fue el primer satélite comercial mexicano lanzado desde manos privadas. Satmex 5 es un satélite geosíncrono modelo HS 601 HP, único satélite Latinoamericano que ofrece cobertura continental desde Canadá hasta Argentina, en una sola huella satelital. Satmex 5 transmite con una potencia de 49 dBW y transporta una carga útil de 24 transpondedores en banda C y 24 en banda Ku y esta ubicado a los 116.8º oeste.

El Satmex 6 fue solicitado a SpaceSystems Loral, con base en el satélite LS-1300X, construido en Palo Alto, California. Pesa 5,700 Kg, cuenta con 36 bandas C y 24 bandas Ku. Fue lanzado el 27 de mayo de 2006 por un cohete Ariane-5ECA y puesto en órbita geoestática en los 109.2º de latitud oeste.

El Satmex 7 es un satélite solicitado en junio de 2008 a SpaceSystemsLoral, con base en el satélite LS-1300, que incorporará el sistema Fixed Satellite Services (FSS) al continente. Es un satélite de última generación con alta capacidad en transmisión en bandas C y Ku que cubrirá el servicio de HDTV. Esta diseñado para ocupar la posición orbital de 114.9º que actualmente ocupa el Solidaridad 2 y se realizó su lanzamiento en diciembre de 2012, [1].

El Satmex 8 fue anunciado en mayo de 2010 el cual reemplazará el Satmex 5 que incorporará el sistema Fixed Satellite Services (FSS). Se espera que tenga capacidad para 64 bandas C y 64 bandas Ku. Tendría una vida útil de 15 años. En abril de 2010, SATMEX hizo un pago inicial de $2 millones de dólares a SpaceSystems/Loral para comenzar la fabricación del SATMEX 8, el cual sería el reemplazo del SATMEX 5 que tiene combustible suficiente para operar hasta el 2013. Se planea lanzar este satélite a finales de 2013, [2].

Todos estos satélites se construyeron, se validaron y se lanzaron al espacio fuera de nuestro país, por lo cual somos usuarios de la mejor tecnología satelital a nivel mundial, pero, hasta el momento aun no hemos logrado colocar un satélite hecho en México en el espacio que opere exitosamente, que además sirva como palanca para encauzar el desarrollo de alta tecnología mexicana.

Dentro de las grandes experiencias que ha tenido México hasta el momento, se encuentran aquellas donde ha participado la UNAM: los Unamsats A y B, el proyecto microsatelital Satex y otras iniciativas en proceso de gestación en diversas dependencias educativas mexicanas.

El UNAMSAT-1 fue concluido en 1993 en Ciudad Universitaria. De forma cúbica de 23 centímetros, con 10 litros de volumen y con un peso de 17 kilos. Compuesto por 5 módulos cuadrangulares, celdas solares y baterías, con una vida útil calculada de 4 años y medio. Su sistema de telecomunicación trabajó bajo el sistema amateur de 0.2-0.3 UHF, con un sistema de protocolo PACSAT que usaron los satélites Oscar 16 y 19. Para ponerlo en órbita polar, se estableció un convenio de colaboración del PUIDE con el Instituto Sternberg de la Universidad estatal de Moscú y la empresa espacial Progress, con el fin de tener un costo menor en la puesta en órbita del satélite, en comparación con lo que sería un acuerdo comercial. El convenio permitió establecer con el Grupo de Tecnología Espacial del Instituto Sternberg el diseño del acoplamiento entre el satélite y el cohete ruso. El UNAMSAT-1 fue catalogado como carga secundaria durante el lanzamiento de un satélite de comunicaciones Ruso, por lo que el proyecto sufrió aplazamientos según el propio programa espacial Ruso. La primera fecha programada fue en diciembre de 1993 y la segunda en junio de 1994.

Finalmente el UNAMSAT-1 fue lanzado desde una base militar en Plasestsk, Rusia, el 28 de marzo de 1995, pero una falla en el IV estadio del Start 1 Ruso malogró el lanzamiento, [3].

El UNAMSAT-2 comenzó a construirse a la par del UNAMSAT-1 como un satélite gemelo para quedarse en Tierra como simulador de la operación en órbita, el cual se pensaba concluir en 1995. Sin embargo, tras la fallida puesta en órbita del UNAMSAT-1, se decidió poner en órbita a su gemelo, bajo el nombre de UNAMSAT-B, realizándose las adecuaciones correspondientes para ello y añadiéndosele mejoras en relación con su antecesor. Sus 5 paneles solares de Arseniuro de Galio fueron adquiridos a mitad de precio a la empresa italiana FIAR, S.p.A. El lanzamiento se negoció con la ayuda del Instituto Aeronáutico de Moscú (MAI), con la Empresa Espacial LavochkinAssociation, con quienes se diseñó y construyó el acoplamiento del UNAMSAT-B a un satélite militar Ruso Kosmos 2334 (Parus #86), que sería la carga principal del cohete Kosmos-3M de la empresa Rusa Polyot. El lanzamiento se realizó el 5 de septiembre de 1996, desde el cosmódromo de Plesetsk. Cinco horas después el UNAMSAT-B se separó exitosamente del satélite militar Ruso para alcanzar una órbita de mil kilómetros de altura a 83º de inclinación con respecto del ecuador; e inició transmisiones a la estación portátil instalada en Plesetsk a las 11:00 P.M. En agosto de 1997 se reportó que comenzaron a tenerse problemas en el sistema de alimentación de poder, con lo que se dio por perdido al satélite, [4].

El proyecto del microsatélite experimental SATEX 1 fue desarrollado por la Benemérita Universidad Autónoma de Puebla, en coordinación con la Comisión Federal de Telecomunicaciones y con la participación de diversos centros de investigación mexicanos: entre ellas el Instituto de Ingeniería UNAM. La finalización oficial de las actividades encargadas al Instituto de Ingeniería ocurrió en 2004 cuando se dieron por terminados los sistemas de la computadora de vuelo, el subsistema de sensores, protocolos de comunicaciones, el software operativo del satélite y el software de estación terrena encargado de monitorear el satélite. Desafortunadamente, el satélite nunca fue lanzado debido a que no se llegó a integrarlo ni a aplicarle por tanto pruebas operativas, [5].

Los proyectos citados anteriormente han sido y son muy importantes para México, no sólo para tratar de reducir la enorme brecha tecnológica que tenemos en relación con los países altamente industrializados, sino para conformar un programa doméstico de desarrollo y de investigación en el campo satelital. Sin embargo, para conducir tal línea del trabajo, se requiere la formación, la calificación y el entrenamiento de recursos humanos en estas áreas tecnológicas. De acuerdo con la experiencia del Instituto de Ingeniería de la UNAM, se ha constatado que el desarrollo de un satélite experimental en nuestro país requiere de 4 a 8 años de trabajo, dependiendo tanto del apoyo financiero obtenido, como de la magnitud de los objetivos tecnológicos y científicos que se persigan, también se ha visto que este tipo de proyectos puede requerir la participación de 25 a 200 personas dependiendo de los retos y de la magnitud de desarrollo tecnológico que pretenda una misión satelital.

El análisis de esta problemática en México durante los últimos 15 años llevó a elaborar la idea de desarrollar un sistema de bajo costo para el entrenamiento de recursos humanos en técnicas y temas de tecnología satelital. Aplicándose esto en el desarrollo del SATEDU ver1 y SATEDU ver 2 (versiones 1 y 2), éste proyecto finalizó exitosamente en Enero de 2009. SATEDU ver1 y ver2 emplea partes de uso comercial, debido a que no se pretende orbitarlo en el espacio. Debe señalarse que este modelo se empleará a su vez para generar proyectos alternos de tecnología y conocimiento, que puedan desarrollarse para uso espacial. Adicionalmente, se pretende que SATEDU ver1, ver2 y versiones posteriores se

empleen en centros educativos (universidades, tecnológicos, etc.) para acercar el tema tecnológico satelital no solo a la juventud, sino también a personal docente y de investigación en el campo satelital y tecnologías de la información, [6]

SATEDU ver 1 y ver2, sólo tienen una rueda inercial, por lo cual sólo pueden emplearse para realizar pruebas de estabilización en un solo eje, sin embargo, para que el sistema SATEDU se utilice en el desarrollo de tecnología satelital avanzada, se requiere de una nueva versión que permita controlarlo en tres ejes, es decir, que contenga 3 ruedas inerciales. Este precisamente es el objetivo de la presente propuesta de tesis, desarrollar la nueva computadora de vuelo que a su vez se empleará como tarjeta de estabilización para SATEDU ver3, lo cual dará pauta para generar posteriormente tecnología satelital para uso en órbita terrestre.

Se pretende además, que el prototipo de la tarjeta de estabilización que se desarrolle en esta propuesta, así como todo el sistema SATEDU ver3, sean suficientemente económicos para utilizarlos en laboratorios y que sean así mismo, suficientemente flexibles, versátiles y amigables para incluirse en cursos de entrenamiento y aprendizaje (atractivos, modulares y rápidos) en el campo satelital. De este modo, se persigue desarrollar un subsistema del nuevo SATEDU ver3, que permita acercar a las nuevas generaciones al mundo apasionante de los satélites, del espacio y de las tecnologías de la información, y en general, para apoyar la difusión y el avance de La Ciencia y La Tecnología en México.

Al igual que SATEDU ver1 y ver2, la versión 3 estará constituida fundamentalmente por un pequeño prototipo satelital totalmente instrumentado (dentro de un volumen de 1.5 a 2 litros), además de un software interactivo que correrá en computadoras personales, para supervisar y controlar inalámbricamente al prototipo.

Por lo anterior, SATEDU contiene todos los subsistemas de un satélite comercial, pero en tamaño reducido, incluyendo: el estructural, de potencia, de comunicaciones, de computadora de vuelo, de estabilización, de sensores de estabilización, de sensores de plataforma y la carga útil.

Particularmente, el subsistema de estabilización que se propone desarrollar en esta tesis se diferencia de las versiones anteriores en que esta nueva versión tendrá las 3 ruedas inerciales incorporadas tanto en su hardware como en su software de estabilización. Este sistema permitirá que SATEDU ver3 pueda emplearse para validar modelos de estabilización satelital en 3 ejes, así como para validar algoritmos de control de apuntamiento en 3 dimensiones.

2.2 Diferencias entre un sistema satelital real y SATEDU

Las diferencias entre un sistema satelital real y SATEDU, radican principalmente en los diseños de los equipos, en la calidad y en el tamaño de sus subsistemas. En el caso de SATEDU se pretende que se puedan desarrollar en paralelo varios subsistemas que serian idénticos tanto en SATEDU como en un satélite real. Así la electrónica para el equipo espacial, debe ser de clasificación militar o espacial en tanto que, para SATEDU su electrónica es de uso comercial.

Además de la fabricación de la electrónica, otra de las diferencias es que algunas tarjetas tienen circuitos de protección del efecto latch-up. Este fenómeno ocasiona que los componentes electrónicos consuman altas y destructivas corrientes en lapsos muy cortos de tiempo, que los llevan a la destrucción cuando no existen protecciones para ello. SATEDU ver1 cuenta con protecciones para efecto latch-up tanto en la computadora de vuelo como en su subsistema de potencia.

Sin embargo, en SATEDU ver2 se omitieron estas protecciones debido a que no tienen ningún uso en SATEDU mientras sea utilizado en salones de clase o en laboratorios ya que este efecto no se presenta, pero serán de gran utilidad cuando la tarjeta se pretenda utilizar en un picosatélite real.

2.3 Arquitectura de SATEDU

Lo primero que se definió al iniciar el diseño de SATEDU fue el tamaño que tendrían las placas de circuito impreso donde se colocaría toda la electrónica que daría vida al satélite educativo. Por cuestiones de compatibilidad física con un Picosatélite, se adoptaron las dimensiones del estándar generado por la Universidad de Stanford, EEUU para el Picosatélite conocido como Cubesat. Las dimensiones externas de éste no deben rebasar los 10x10x10 cm, por lo cual, los circuitos impresos deben ser de menor tamaño. Debido a que uno de los subsistemas más importantes es la computadora de vuelo, esta tarjeta fue la base para definir el tamaño de las demás, y de esta forma el tamaño elegido fue de 8.9 x 8.9 cm.

La forma elegida para interconectar los circuitos impresos se basó a su vez en el principio utilizado por la computadora de vuelo de SATEX [7], es decir, se interconectan uno sobre otro con la ayuda de dos conectores de terminales largos, ubicados en los extremos de cada tabloide, figura 2.1.

Figura 2.1. Vista de SATEDU.

De esta forma, SATEDU está conformado por los siguientes subsistemas [8]:

- Subsistema Estructural,
- 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 futuros desarrollos.

2.3.1 Subsistema estructural

En cuanto a la estructura de SATEDU, figuras 2.1 y 2.2, y debido a que es un simulador satelital, no fue necesario asignarle una forma cúbica o similar, por ello se eligió un envase plástico muy común, de buena presentación, muy económico y además, fácil de conseguir.

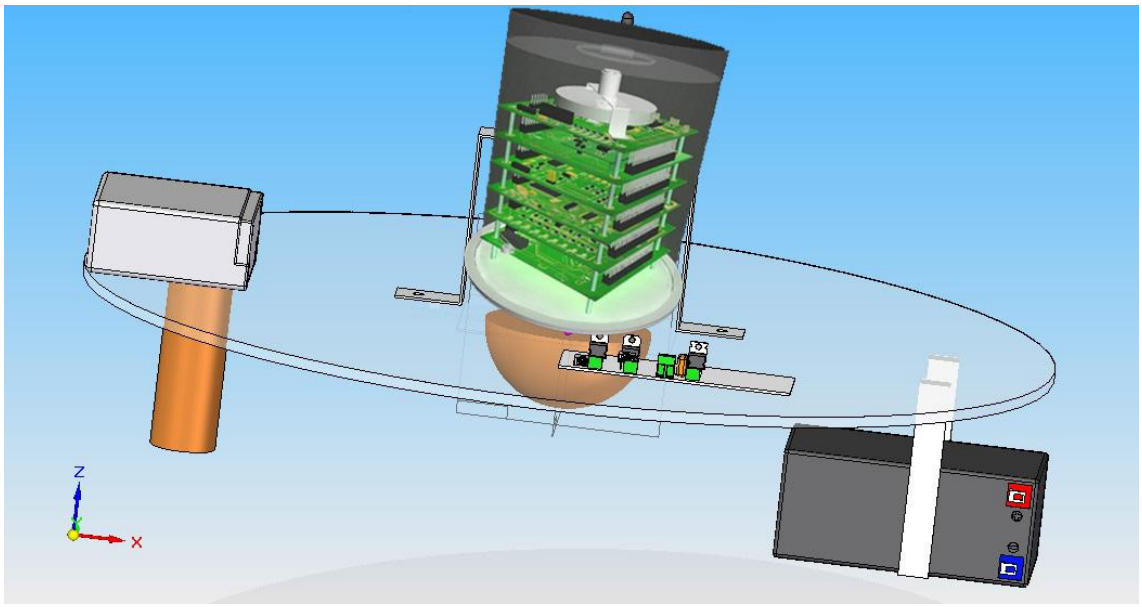


Figura 2.2 Estructura de SATEDU sobre la mesa de aire.

2.3.2 Computadora de vuelo

La Computadora de Vuelo CV reside en un circuito impreso de 2 capas de 8.9 x 8.9 cm. Para comunicarse con las demás tarjetas se colocaron dos conectores en lados opuestos haciendo que otras tarjetas puedan montarse tanto por la parte superior como por la parte inferior, y del mismo modo establecen conexiones eléctricas a dos buses. Adicionalmente para darle mayor firmeza a la estructura se le agregaron espacios para tornillos en sus 4 esquinas, los cuales permitirán que el bloque de circuitos impresos pueda fijarse a la estructura, figura 2.3.

El objetivo de la presente tesis es desarrollar la nueva versión de esta computadora de vuelo, estos detalles se presentan en los siguientes capítulos.

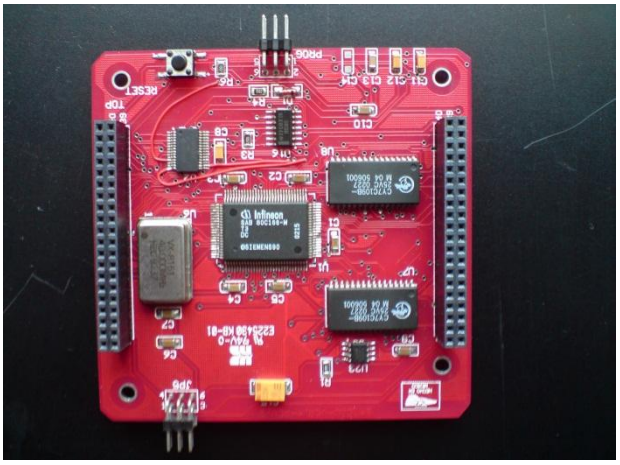


Figura 2.3 Circuito impreso de la CV realizado en Protel DXP.

2.3.3 Subsistema de potencia

El Subsistema de Potencia (SP) está constituido por 2 tarjetas, una de ellas contiene electrónica del sistema, que además fue diseñada para emplearla en un Picosatélite real por lo que tiene una serie de circuitos adicionales para protegerla del efecto “latch-up” y para asegurar su encendido al momento de que el satélite sea liberado en el espacio. También contiene interruptores que requiere el Picosatélite real, en este caso, un interruptor para remover antes vuelo y el llamado “killswitch” que mantiene totalmente apagado al Picosatélite hasta que ocurre la liberación. La segunda tarjeta básicamente contiene baterías Li+ y algún otro circuito que no pudo alojarse en la primera.

La electrónica del SP está constituida por: un microcontrolador PIC18F2321 de Microchip, que se comunica con la CV para recibir comandos de apagado y encendido de las fuentes de energía para los demás subsistemas; una serie de reguladores; un convertidor DC-DC que suministra energía a los subsistemas; un sistema de recarga de voltaje para baterías; las fuentes de energía primaria y secundaria de SATEDU que son 4 baterías Li-+ recargables y celdas solares que son flexibles y auto adheribles.

2.3.4 Subsistema de comunicaciones

En cuanto al Subsistema de Comunicaciones (SC), figura 2.4, el SATEDU emplea dos subsistemas diferentes. Uno dentro del sándwich de impresos SATEDU y otro que se emplea en la computadora personal que hace las veces de estación terrestre y en donde se encuentra el software para manipular a SATEDU. El primero está constituido por una sola tarjeta, en tanto que el segundo tiene una electrónica similar más componentes adicionales, así como un contenedor y un conector USB hacia la PC para propósitos de comunicación y energización. Ambos subsistemas son de uso exclusivo de SATEDU en laboratorio, es decir, estos subsistemas no fueron desarrollados para emplearse en un Picosatélite.

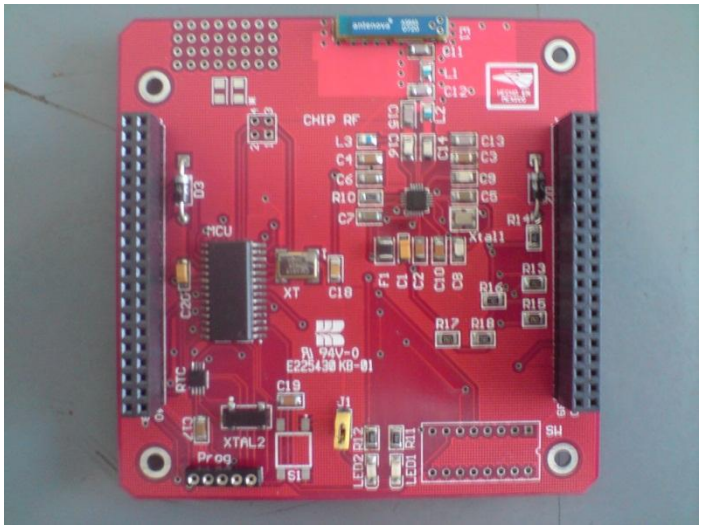


Figura 2.4 Tarjeta de Comunicaciones.

Ambos SC o 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 desarrollado para la PC usa un transceptor adicional que es utilizado para comunicarse con la PC vía USB. De esta forma al usar una Laptop junto con SATEDU se permite obtener una completa movilidad o portabilidad del sistema completo.

2.3.5 Subsistema de estabilización

El sistema completo está contenido en 2 tarjetas, una que contiene el MCU LM4F2315QHR Stellaris ARM cortex 4, un Microcontrolador para reprogramación, un banco de memoria flash de 4 Mb y bobinas de torque magnético.

La segunda contiene principalmente la parte de potencia (puentes H, transistores de potencia y elementos pasivos) para manejar 3 ruedas inerciales externas y conectores al exterior y reguladores para alimentar otras tarjetas. Las ruedas inerciales se muestran en la figura1.4.5, son masas circulares sujetas a motores, mientras que las BTM son embobinados puestos uno en cada eje.

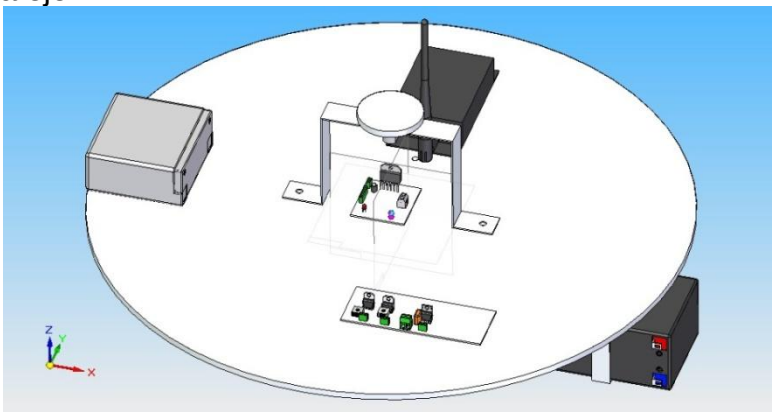


Figura 2.5 Vista artística de la rueda inercial y su motor.

Los componentes y desarrollo de este subsistema serán discutidos más a fondo en los siguientes capítulos de esta tesis.

2.3.6 Subsistema de sensores de navegación inercial

El desarrollo de la CV que se presenta en esta tesis, tiene un conector para instalar una tarjeta diminuta de la compañía Sparkfun, la cual integra tres giróscopos, tres magnetómetros, y tres acelerómetros, cada sensor en ejes perpendiculares. A la integración de estos tres sensores se le conoce como MARG por sus siglas en ingles (Magnetic, Angular Rate and Gravity), a estos sensores los referiremos como los sensores de navegación inercial del satélite, figura 2.6

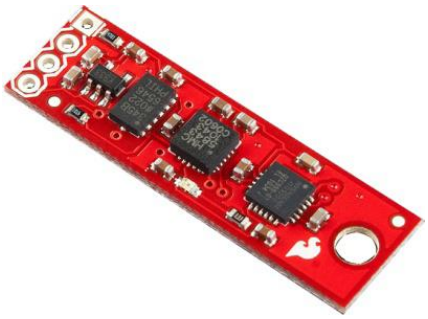


Figura 2.6 Sensores de navegación inercial Sparkfun.

Los tres sensores utilizan un bus de comunicaciones I²C, con el que se comuninican a la CV, lo que reduce los costos ya que el costo es menor al comprar esta tarjeta que adquirirlos por separado.

2.3.7 Subsistemas de nuevos desarrollos (carga útil)

SATEDU ha sido diseñado y desarrollado para desarrollar expansiones que permitan adaptarlo a diversas necesidades ya sean de capacitación, enseñanza o investigación, todo esto sin necesidad de hacer cambios a los circuitos impresos que lo componen. Para este propósito, se han dejado señales eléctricas pertinentes dentro de los conectores laterales para que se puedan desarrollar nuevos subsistemas e integrarlos sin grandes dificultades a la operatividad de SATEDU.

Uno de los nuevos desarrollos que se pensaron fue la implantación de una pequeña cámara digital que sea capaz de almacenar fotografías en la memoria flash de la CV, para posteriormente transferir los datos de imagen a la computadora que simula ser la estación terrena. Por estas razones, el nuevo diseño de la CV que se presenta en esta tesis ofrece recursos mayores y mayor rapidez de procesamiento para poder implantar de forma más sencilla este tipo de aplicaciones.

2.3.8 Segmento terrestre para supervisión y mando de SATEDU

Este segmento de supervisión comprende a una computadora donde se encuentra el software de interfaz de comunicación con SATEDU. Este subsistema permite desplegar la telemetría de SATEDU, enviar nuevos programas operativos a la CV de SATEDU, así como la visualización de telemetría en tiempo real.

El desarrollo de este software estará basado parcialmente en el software desarrollado para el microsatélite SATEX debido entre otros aspectos a que éste software podría emplearse en un Picosatélite real y es precisamente el sistema SATEDU quien permitirá depurar y validar gran parte de su desarrollo. Este subsistema se espera desarrollar en su nueva versión durante 2013, y sus detalles quedan fuera del alcance de esta tesis.

2.4 La necesidad de incluir medios de estabilización activa para SATEDU

En un proyecto anterior (microsatélite experimental SATEX) los medios de estabilización eran bobinas de torque magnético y un gradiente gravitacional, los cuales son medios de estabilización muy lentos y limitados. En tal proyecto se tenía calculado que su proceso de estabilización en el espacio hubiera tomado varias semanas antes de empezar a utilizar sus cargas útiles.

Por ello, en el IINGEN-UNAM se han desarrollado los medios mecánicos y electrónicos para manejar ruedas inerciales, las cuales constituyen actuadores activos que son económicos y eficientes para realizar maniobras de estabilización y control de apuntamiento. Los medios de estabilización son de vital importancia, ya sea para que el satélite genere energía eléctrica de forma eficiente (orientando las celdas solares hacia el sol), para permitir que los experimentos realicen sus tareas, o bien, para establecer comunicaciones directivas con Tierra, de amplio de ancho de banda.

Otro medio de estabilización y de propulsión que se emplea por tanto para modificar la altitud orbital de satélites de forma más rápida, 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 aun en SATEDU, sin embargo, se espera que en el futuro se pueda desarrollar un sistema sencillo con estas características, [9]

2.5 La reducción de costos como base global de diseño y desarrollo

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

Aunque actualmente la empresa Colorado Satellite Services (EUA), comercializa un sistema satelital educativo, este tiene un costo considerablemente elevado que es de USD$7995, [10], este costo es debido en principal medida a su manufactura. Entre otros aspectos, está fabricado con circuitos impresos multicapa, tiene una estructura de acrílico, espaciadores de tarjetas hechos a la medida y un subsistema de comunicaciones comercial.

Por otro lado, SATEDU está compuesto por circuitos impresos de dos capas, una estructura sumamente económica 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 real, es decir se han diseñado para emplearse en vuelos orbitales.

Esto hace que SATEDU, además de ser más económico, sea un sistema de mayor valor agregado, por las alternativas de uso que tiene y por sus posibilidades de expansión.

Capitulo 3. Estado del Arte

3.1 Introducción a los CubeSats

El desarrollo de los pequeños satélites llama la atención a nivel mundial de fabricantes y diseñadores de los mismos, debido a sus capacidades económicas y tecnológicas. En las universidades y centros de investigación el primer foco de atención son los satélites clasificados como micro, nano y pico satélites.

Dentro de estos satélites se ha adoptado el estándar CubeSat (acrónimo de Cube Satellite), diseñado en 1999 por el Dr. Robert Twiggs de la Universidad de Stanford y por el Dr. Jordi Puig-Suari de la Universidad Politécnica del Estado de California, el primero de ellos también creador del concepto CanSat, [11]. Desde entonces se ha dado un crecimiento en el desarrollo de estos pequeños satélites ya que conforme avanza la miniaturización electrónica que permite tener cargas útiles (payloads) e instrumentos para misiones espaciales de menor tamaño y peso, también un consumo de energía más eficiente, y reducción de costos lo que los ha hecho viables para academias, institutos y universidades. Estas instituciones académicas son las que han hecho la mayor parte del desarrollo de los mismos, [12].

3.1.1 XATCobeo

XATCobeo es un proyecto español desarrollado en la Universidad de Vigo. El proyecto es dirigido por el Dr. Fernando Aguado en colaboración con el Instituto Nacional de Técnica Aeroespacial (INTA), el apoyo de la empresa Retegal y la junta de Galicia. Se estima que la vida útil del dispositivo aeroespacial será de 6 a 12 meses y su costo aproximado fue de 1.2 millones de euros. El principal objetivo de la misión es el desarrollo del satélite en sí mismo bajo el estándar CubeSat, junto a una Estación en Tierra en la Universidad de Vigo.

La carga útil del satélite consiste en un programa definido de radio reconfigurable (*software defined reconfigurable radio o SRAD*) y un sistema de medición de cantidad de radiación ionizante *(ionizing radiation system* o RDS). Se incluye también un sistema experimental de despliegue de paneles solares. Con el desarrollo de este proyecto se favorece un escenario donde profesores, investigadores y estudiantes pueden enfrentarse a la resolución de problemas en áreas como electrónica, ingeniería, comunicación y desarrollo de software, para la construcción del vehículo espacial.

El diseño del satélite se apega al estándar CubeSat con una dimensión de $10cm^3$ y aproximadamente un kilogramo de peso. Además, ha sido necesario que se apegue a estándares de documentación y seguimiento de proyectos propuestos por la Agencia Espacial Europea para garantizar su correcto funcionamiento en el espacio así como su seguridad durante el vuelo, [13].

3.2 Sistemas De Entrenamiento en Recursos Humanos

Una parte importante en el desarrollo de los pequeños satélites es la creación de Satélites Entrenadores de Recursos Humanos ya que estos están diseñados con todos los sistemas de un Satélite de vuelo pero son para uso en tierra, específicamente en salones de clase para enseñar a alumnos de ingeniería de una manera económica los subsistemas que componen a un satélite real.

Actualmente hay una versión comercial de un satélite para salones de clase que fue desarrollado por la US Air Force, el Eyassat, figura 3.1, [10] y [14], este cuenta con subsistemas de:

Computadora de vuelo y manejo de datos: comandos para las comunicaciones entre módulos y para telemetría.
Estabilización y control: incluye una rueda inercial y bobinas de torque magnético, cuenta con control de lazo cerrado para mantener los paneles solares apuntando hacia el sol.
Comunicaciones. Canal inalámbrico para conectarse a funciones de telemetría y control pueden conectarse hasta 10 de estos.
Potencia. Este genera distintos voltajes para las tarjetas del satélite.
Estructural. Cuenta con paneles solares, sensores de movimiento, bobinas de torque magnético.
GPS (opcional): da la localización con el sistema Global Positioning System.
FlexX (opcional): para poner sensores extras.

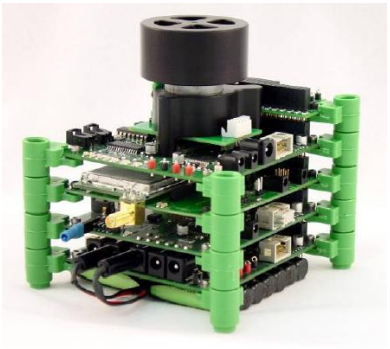


Figura 3.1 Eyassat.

Otro satélite de entrenamiento de recursos humanos en desarrollo es el Sensat (acrónimo de Self-explore Nano Satellite), que es un proyecto del CISESE, [16] y [17], de la universidad de Ensenada Baja California, que tiene como objetivo, realizar investigación, diseño, desarrollo y construcción de satélites pequeños educacionales, como instrumentos tecnológicos para la formación de recursos humanos especializados en tecnología aeronáutica y aeroespacial, tanto en la industria como en las instituciones de educación superior de la región Sonora-Baja California.

El sistema está en proceso de desarrollo, los subsistemas y cuenta con los siguientes subsistemas:

Sensores.

Computadora de vuelo.
Potencia.
Estructural.
Estabilización en tres ejes.

El Instituto de Ingeniería ha desarrollado con la misma idea un Sistema de Entrenamiento de Recursos Humanos en Tecnología Satelital, SATEDU, que es un sistema de bajo costo para el entrenamiento de recursos humanos en técnicas y temas de tecnología satelital, proyecto que finalizó exitosamente en Enero de 2009, [18].

SATEDU cuenta prácticamente con todos los subsistemas de un Satélite Comercial, sólo que en nuestro caso se tienen sistemas muy pequeños y portátiles, además de inteligentes, pues integran cada uno de ellos a procesadores digitales. Entre sus subsistemas se encuentran los siguientes, figura 3.2, [9]:

- Estructura,
- Potencia,
- Computadora de Vuelo,
- Comunicaciones Inalámbricas,
- Sensores de Plataforma Satelital,
- Estabilización por Rueda Inercial
- Sensores de Navegación Inercial.

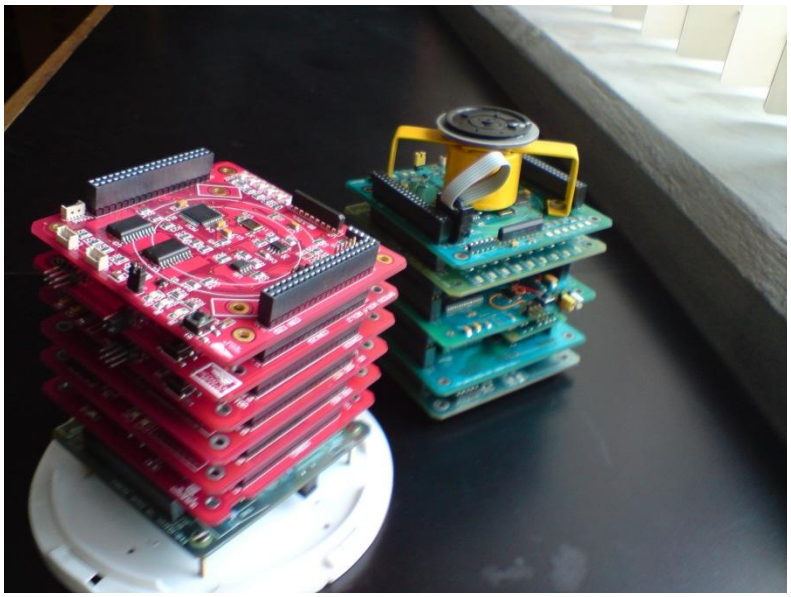


Figura 3.2 Prototipos de Satedu, versión 1 y versión 2.

Adicionalmente cuenta con software distribuido en cada uno de sus subsistemas, con los cuales realiza tareas de alta complejidad. De igual forma su Computadora de Vuelo contiene el Software de Comunicaciones entre SATEDU y su Estación Terrena, que en este caso está constituida por Software que se ejecuta en una PC, [8].

Otra de las principales características de SATEDU es que en términos de costo de partes oscila alrededor de los $20,000 pesos MN. Este costo contrasta con el precio de adquisición del único sistema comercial que existe en el mundo y que fue desarrollado originalmente por la Fuerza Aérea Norteamericana, y que actualmente comercializa la compañía Colorado Satellite Services, EU, el cual es de aproximadamente $120,000 pesos MN, [10], [14].

Ambas plataformas cuentan solo con una rueda inercial por lo que se pueden mover únicamente en un eje; por ello, el propósito de esta tesis es integrarle tres ruedas para darle la capacidad de movimiento en tres ejes.

3.3 Ruedas Inerciales y Bobinas de Toque Magnético

Las Bobinas de Torque Magnético han sido utilizadas en muchos CubeSats, mientras que son pocos los que implementan Ruedas Inerciales para control de apuntamiento. El BEESAT, [19], de Alemania cuenta con 3 pequeñas ruedas y es capaz de hacer giros de 180° en tres minutos, [20].

3.4 Mesa Suspendida en aire

El problema del apuntamiento satelital es constantemente tratado en gran cantidad de bibliografía referente al tema de control satelital, y al mencionar apuntamiento nos referimos al estado de orientación de un satélite con respecto a la Tierra, aunque también puede aplicarse respecto a otro objeto. Tal es el caso de satélites astronómicos o de satélites espías, los cuales pueden tener objetivos diferentes y específicos para su objeto de análisis, [21].

Para realizar pruebas que validen en tierra (laboratorio) el control de orientación satelital recreando lo mejor posible las condiciones dinámicas que tendrá en el espacio exterior es necesario del uso de una mesa suspendida en aire (MSA).

La mesa suspendida en aire, figura 3.3, es un sistema desarrollado para recrear ambientes de baja fricción, aunque es obvio que en tierra es muy difícil cancelar la gravedad, por lo que solo se recrea la parte del efecto de la flotación con recursos que son asequibles y con equipo bastante compacto y factible de tener prácticamente en cualquier laboratorio.

La mesa suspendida en aire consiste de 3 partes básicas:

• *Mesa o plataforma*: es la base en la cual se montan todos los actuadores y electrónica necesaria para realizar el control de apuntamiento, como son las ruedas inerciales, bobinas de torque magnético y el sistema electrónico de control.

• *Cojinete neumático esférico:* Es la parte principal y consta de una semiesfera, una copa, la brida y el soporte. La copa es una base donde embona la semiesfera, a través de la cual sale expulsado aire a presión por varios orificios capilares de diferentes dimensiones ubicados de

manera distribuida. Todo esto calculado de manera tal que cuando el aire choca con la copa hace que esta se eleve centésimas de milímetro que es suficiente para recrear la cero fricción. La brida es la parte que va debajo de la copa que se encarga de encauzar el flujo de aire a la copa y también hacia una válvula de desahogo que permite que el aire no se aglutine en la copa ya que causaría vibraciones sobre la semiesfera. El soporte sostiene a las partes anteriores a cierta altura.

• *Un compresor de aire*: que inyecta aire a presión al cojinete neumático, este debe de tener de preferencia un sistema de filtrado de agua, aceite y polvo ya que pueden llegar a obstruir el flujo constante de aire sobre el cojinete neumático.

Figura 3. 3 Mesa suspendida en aire.

3.5 Subsistema de sensores de navegación inercial de SATEDU

Los sensores de estabilización de SATEDU permiten conocer la orientación del satélite. Las categorías de sensores de posición y orientación en un satélite se subdividen en 2, [14]:

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 la 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 Navigation Satellite Systems), conocido como GPS

Sensores de inercia: Estos miden continuamente los cambios en posición, sin embargo, necesitan un ajuste y una calibración de los sensores de referencia. Los giróscopos representan el único sensor conocido hasta ahora como sensor de inercia y las únicas variantes que tiene se refieren a su fabricación.

Para realizar un buen sistema de navegación y orientación satelital se deben de combinar ambas categorías de sensores, de referencia e inercia, ya que es muy difícil que baste un solo tipo de sensor para medir adecuadamente el apuntamiento de un satélite.

En SATEDU se utilizó un sensor de referencia, representado por la brújula electrónica y sensores inerciales mencionados anteriormente, en este caso 3 giróscopos de un eje dispuestos de manera ortogonal y un acelerómetro de 3 ejes, figura 3.4.

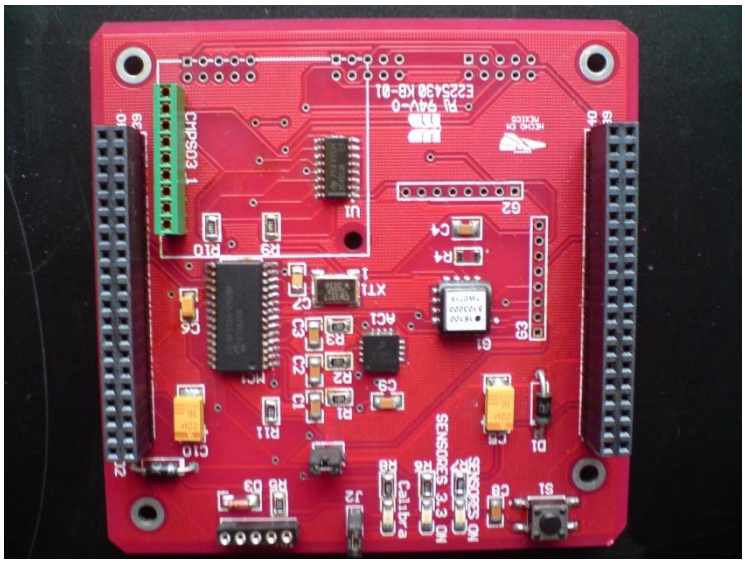


Figura 3.4 Tarjeta de sensores de navegación inercial de SATEDU.

3.6 Sistemas de navegación inercial

Un sistema de navegación inercial (INS - Inertial Navigation System) es un asistente de navegación que emplea una computadora y sensores de movimiento (acelerómetros y giróscopos) para calcular continuamente (a través de la estimación de la posición) la orientación y la velocidad de un objeto en movimiento, sin la necesidad de emplear referencias externas. Estos sistemas fueron desarrollados originalmente para usarlos en cohetes, siendo Robert Goddard el pionero en la integración de estos sistemas giroscópicos.

Un INS puede detectar cambios en su posición geográfica, cambios de velocidad y cambios de orientación, lo cual logra por medio de la medición de las aceleraciones y velocidades lineales y angulares aplicadas al sistema.

Los INS se utilizan en muchos sistemas móviles tales como: vehículos, aeronaves, submarinos, vehículos espaciales, satélites y misiles guiados. Sin embargo, su complejidad restringe los ambientes en que pueden emplearse.

Un INS está compuesto por unidades de medición inercial (IMU – Inertial Measurement Unit) compuestas por acelerómetros y giróscopos, que realizan tanto mediciones lineales (acelerómetros) como mediciones angulares (giróscopos), las que cuentan mínimamente con un sensor de cada tipo para cada eje. Por cada grado de libertad,

de los 6 que se tienen en traslación y rotación (x, y, z y θ_x, θ_y, θ_z) se integran a través del tiempo las variables medidas hasta obtener la posición y orientación. La creación de los INS en sus inicios fue una tarea muy difícil ya que no se contaba con la capacidad de cómputo necesaria para ello.

Aunque la tarea de obtener la posición y orientación de un sistema pareciera sencilla esto no es así porque cada sensor tiene una tasa de error debido a que su fabricación no es perfecta. Todos los sistemas de navegación sufren de un problema llamado “deriva”, donde los pequeños errores de la medición de aceleración y velocidad angular al integrarse progresivamente terminan convirtiéndose en errores muy grandes con el paso del tiempo. Es por esto que se usan otras clases de sensores u otros sistemas para corregir periódicamente el error, tales como magnetómetros y GPS (Global Positioning System). Hoy día existen técnicas para obtener una gran precisión en los INS, como es el filtro Kalman y variaciones del mismo, que permiten fusionar las mediciones de distintos tipos de sensores.

3.6.1 Sistemas de navegación inercial en satélites

En satélites, los INS son parte fundamental del sistema de control de apuntamiento, que permiten que los satélites desempeñen tareas críticas para algunos subsistemas o experimentos que demanden estabilización y apuntamiento. Estas tareas pueden ser el control de orientación de las antenas para sistemas de comunicaciones directivos o el apuntamiento de sistemas de percepción remota. Un INS satelital no es igual a uno instalado en Tierra, el cual varía en varios aspectos, uno de ellos la calidad de su electrónica. Debido a las condiciones adversas que existen en el espacio (rayos cósmicos, temperaturas extremas, vacío, etc.) la calidad de los componentes electrónicos de uso espacial debe ser mejor a los de uso terrestre, o bien, deben contar con protecciones electrónicas espaciales para uso orbital. Otro aspecto en el que cambia la electrónica de un INS satelital es que debe recurrir a otra clase de sensores de referencia, que un sistema terrestre no suele necesitar, como el sensor de horizonte, sensor de estrellas y los sensores de sol que en un satélite son muy útiles.

Aunque en satélites se recomienda el uso de sensores de calificación espacial, estos suelen ser muy caros, espaciosos, pesados y con un consumo de energía considerable, que son aspectos que un nanosatélite como SATEDU no debe tener. Por ello es que en sensores se está recurriendo a componentes de tipo comercial de calificación industrial y se le anexan protecciones a través de electrónica adicional para protegerlos de las condiciones mencionadas anteriormente, y de preferencia empleando partes que hayan sido validadas previamente y con éxito en el espacio por otros desarrolladores.

3.6.1.1 Acelerómetro

El acelerómetro es un sensor que mide la aceleración de un cuerpo relativa a su base. Esta no es necesariamente del mismo tipo que la aceleración coordinada (cambio de velocidad del dispositivo en el espacio), sino que es el tipo de aceleración asociada con el fenómeno de peso experimentada por la masa de prueba que se encuentra en el marco de referencia del acelerómetro. Los acelerómetros miden por tanto, peso por unidad de masa, una cantidad

también conocida como fuerza especifica o *fuerza g.* Otra forma de decir esto es que al medir el peso, un acelerómetro mide la aceleración del marco de referencia de caída libre (sistema de referencia inercial) con respecto a si mismo [22].

Actualmente existen acelerómetros que son capaces de medir desde un solo eje hasta 3 ejes, los cuales se usan para determinar la orientación, aceleración de coordenadas (siempre y cuando haya cambios de aceleración), detectar vibraciones, golpes y caídas. Su fabricación es variada pero actualmente se está incrementando su producción debido a la evolución de los MEMS.

En satélites estos dispositivos son utilizados como sensores de referencia ya que el marco inercial utilizado para estimar la orientación es la Tierra, pero dependerá de la distancia a la que estos se encuentren ya que entre más alejado se este de la Tierra la fuerza de gravedad disminuye hasta el punto en que prácticamente es nula y por tanto también la medición en el sensor, por lo que deja de ser útil en su función de sensor de referencia. Usualmente debido a esta falta de confiabilidad no son usados en el espacio (solo se usan durante la fase de lanzamiento orbital) y son reemplazados por otra clase de sensor.

En la mesa suspendida en aire que se usa para validar los algoritmos de control de apuntamiento satelital se usa un acelerómetro triaxial debido a que es fácil conseguir y de trabajar. En el satélite real este se puede reemplazar por otro tipo de sensor y solo hay que adecuar de manera mínima el modelo de control de apuntamiento.

3.6.1.2 Giróscopo

Los giróscopos son sensores que miden la rotación. Hay 2 clases de giróscopos, los que miden velocidad angular y los que miden el ángulo de desplazamiento respecto de un marco inercial de referencia. Para satélites se usan giróscopos que miden velocidad angular y se utilizan tanto para medir la velocidad de giro como para formar parte del control de apuntamiento. Los giróscopos por si mismos son incapaces de sostener el control de apuntamiento ya que no poseen la capacidad de obtener una referencia de orientación es por eso que se usan para ayudar en esta tarea a otros sensores, [23] y [24].

Existen distintos tipos de giróscopos, los cuales se clasifican de acuerdo a su uso en los siguientes tipos, tabla 3.1:

Giróscopos	Uso			
Tipo	Estratégico	Navegación	Táctico	Consumidor
Flotado de un grado de libertad	X			
Flotado de 2 grados de libertad	X			
Giróscopo con ajuste dinámico		X		
Giróscopo electroestáticamente suspendido	X	X		
Giróscopo de anillo laser		X	X	
Giróscopo antibloqueo		X	X	
Giróscopo interferométrico de fibra óptica		X		
Giróscopo con resonador hemisférico		X		
MEM			X	X
Sensor de vibración de disco				X
Giróscopo de resonancia magnética nuclear				X

Tabla 3.1 Clasificación de los giróscopos.

Dentro de la clasificación de la tabla 3.1, cada tipo de giróscopo tiene una forma distinta de fabricación, hoy día al igual que los acelerómetros los giróscopos de fabricación MEMs son más accesibles, aunque aun no son los más adecuados para sistemas de navegación. La forma en que avanza esta tecnología y la reducción de su precio, consumo de energía y tamaño en comparación de otros sensores los convierten en una buena opción para instrumentar satélites pequeños. Actualmente varias misiones de nanosatélites experimentales han reportado su uso, como ha sido el satélite experimental AAUSAT-II de la Universidad de Aalborg en Dinamarca o CUTE-I del Instituto de Tecnología de Tokio en Japón, [25].

Los giróscopos dependiendo de su complejidad y fabricación presentan diferentes grados de desempeño, entre mejor sea éste, se logrará una mejor precisión al momento de obtener estimaciones de orientación del satélite, como se muestra en la tabla 3.2 [24].

	Grados de desempeño			
Parámetro de desempeño	**Unidades**	**inercial**	**Intermedio**	**Moderado**
Máxima entrada	$^\circ/h$	10^2 - 10^6	10^2 - 10^6	10^2 - 10^6
	$^\circ/s$	10^{-2} - 10^2	10^{-2} - 10^2	10^{-2} - 10^2
Factor de escala	parte/parte	10^{-6} - 10^{-4}	10^{-4} - 10^{-3}	10^{-3} - 10^{-2}
Estabilidad de bias	$^\circ/h$	10^{-4} - 10^{-2}	10^{-2} - 10	$10 - 10^2$
	$^\circ/s$	10^{-8} - 10^{-6}	10^{-6} - 10^{-3}	10^{-3} - 10^{-2}
Deriva	$^\circ/\sqrt{h}$	10^{-4} - 10^{-3}	10^{-2} - 10^{-1}	1 – 10
	$^\circ/\sqrt{s}$	10^{-6} - 10^{-5}	10^{-5} - 10^{-4}	10^{-4} - 10^{-3}

Tabla 3.2 Grados de desempeño de los giróscopos.

3.6.1.3 Magnetómetro

Los magnetómetros miden la magnitud de campo magnético. En satélites de órbita baja se utilizan como sensores de referencia en el control de apuntamiento de un satélite. Actualmente los magnetómetros pueden tener desde uno hasta 3 ejes de medición, y se utilizan en varios dispositivos móviles. Al combinar la magnitud con la que el campo magnético incide sobre el sensor, teniendo como marco de referencia el modelo magnético de la Tierra y conociendo la ubicación geográfica del satélite es posible determinar su orientación, [26].

Los magnetómetros también se emplean para ajustar la ganancia de las bobinas de torque magnético de un satélite. Debido a la susceptibilidad de los magnetómetros, éstos deben colocarse en una posición alejada de los campos generados por otros dispositivos como bobinas de torque magnético, motores, o materiales ferro magnéticos, generalmente estos sensores se montan en extensiones fuera del satélite.

Capítulo 4. Subsistema de estabilización activa por ruedas inerciales y bobinas de torque magnético

4.1 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 de SATEDU. 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.

4.2 Técnicas de estabilización de satélites en 3 ejes

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

4.3 Estabilización pasiva

Los métodos pasivos no emplean energía eléctrica continua o 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 arcosegundo = 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. [9]

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. 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. [9]

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 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. [27]

4.4 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. 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, [28].

Bobinas de torque magnético (BTM):

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, [28].

Toberas de propulsión (thrusters):

Las toberas de propulsión 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.

4.5 Estabilización activa por medio de RIs y BTMs

El subsistema de estabilización activa a través de ruedas inerciales 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 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.

En el caso de la mesa suspendida en aire MSA se instalaron 3 ruedas inerciales en ejes ortogonales sobre esta, mientras que el anterior Satedu solo contaba con una rueda inercial en su tarjeta de estabilización y por tanto solo se podía 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 la mesa o SATEDU giran en sentido opuesto al que lo hace la rueda debido a la transferencia de momento angular.

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 un sentido u otro partiendo del reposo de modo que al acelerar la rueda hasta una velocidad deseada esta generará una fuerza 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: *F= ma*, esta ley que aunque enuncia que es aplicada a trayectorias lineales también puede ser extrapolada a movimientos 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 al subsistema de estabilización.[28]

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:

$$\tau = J\,w$$

Donde:

τ = torque de reacción en la rueda
J = inercia de la rueda
w = aceleración angular de la rueda

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, [29].

4.6 Las ruedas inerciales de SATEDU

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 4.4, 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.

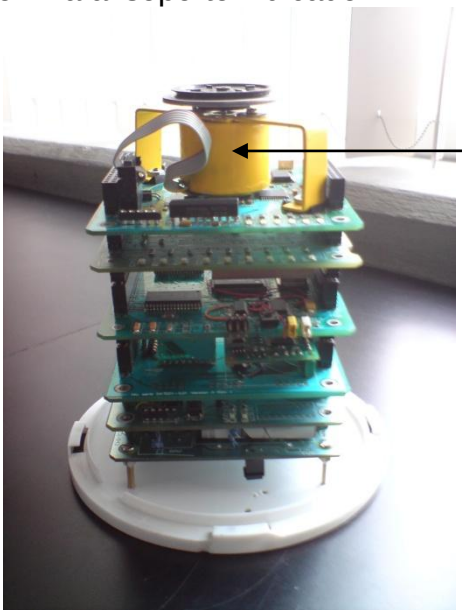


Figura 4.1 SATEDU y su rueda inercial.

4.7 Las Ruedas de la mesa suspendida en aire

Las ruedas inerciales para la MSA son fabricada con una de las aleaciones más comunes de aluminio [31], en este caso la 6061 T6, cuya composición química y propiedades físicas se presentan en las tablas 4.1 y 4.2.

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 4.1 Composición química del Aluminio 6061.

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 4.2 Propiedades físicas del Aluminio 6061.

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.

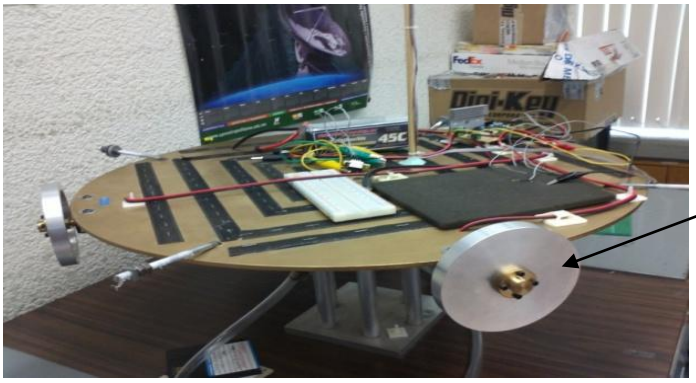


Figura 4.2 Ruedas Inerciales instaladas en la MSA.

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.

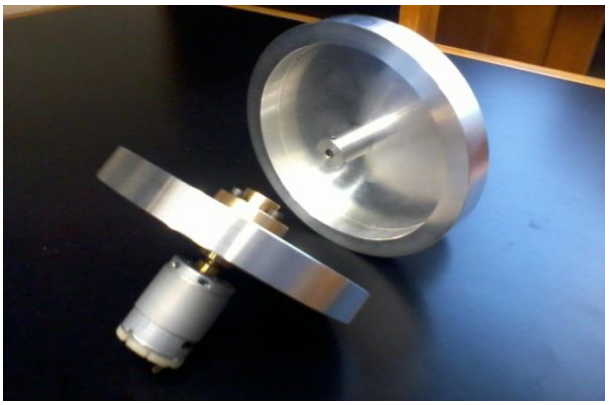


Figura 4.3 Ruedas Inerciales de la MSA.

4.8 Las bobinas de torque magnético de SATEDU

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. 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.

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 satélite 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.

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.

En un satélite real se usan tres tipos de BTMs colocadas ortogonalmente para tener un campo de control en tres planos, figura 4.5, o bien, en un sistema tridimensional, X, Y y Z. De esta manera se puede tener un buen sistema para control de apuntamiento.

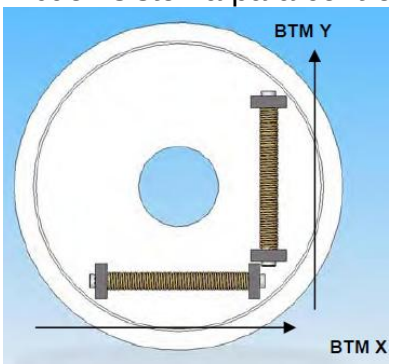


Figura 4.4 BTMs colocadas ortogonalmente.

Capítulo 5. Filtrado de Kalman en el MCU LM4F2315QHR Stellaris

5.1 Introducción

Al usar el sistema de sensores de navegación, estos se ven influenciados por ruido lo que genera a su vez ruido en la medición final de la orientación. Uno de los métodos para filtrar este ruido de las señales son los algoritmos de estimación analíticos. A continuación se presenta el filtro de Kalman, particularmente una de sus variaciones el SR-UKF (Square Root Unscented Kalman Filter) que es el que se va a implementar en esta tesis.
En el siguiente capitulo se explicará un método para estimar los estados de un sistema estocástico. El método fue descrito por Rudolf E. Kalman en 1960.

En un sistema determinístico trabajaríamos de la siguiente forma. En primer lugar, construiríamos un modelo del sistema a partir de las leyes físicas que definen las dinámicas. En segundo lugar, estudiaríamos su estructura y su respuesta, [32]. Por último, y si fuera necesario, se diseñarían compensadores para alterar las características del sistema o bien reguladores. Para ello tendríamos además un amplio conjunto de experiencias que han sido aplicadas con anterioridad sobre estos sistemas. En la realidad ocurre que los sistemas determinísticos son aproximaciones de lo que realmente son, principalmente por tres motivos:

- No existe un modelo matemático perfecto de un sistema real.
- Existen perturbaciones que no se pueden modelar de una forma determinística.
- Los sensores no son perfectos.

Esto nos lleva a plantearnos una serie de preguntas:

- Cómo desarrollar modelos de sistemas que tengan en cuenta las incertidumbres.
- Cómo estimo de una forma óptima los datos que me interesan de un sistema mal modelado y con datos alterados por el ruido.
- Cómo controlo de una forma óptima sistemas estocásticos, [33].

5.2 Filtro de Kalman

En 1960, Rudolph Emil Kalman publica un trabajo titulado “A New Approach to Linear Filtering and Prediction Problems” (Una Nueva Aproximación a Problemas de Predicción y Filtrado Lineal) en este presentaba un nuevo método para separar señales aleatorias y definidas del ruido aleatorio.

Es un algoritmo de procesado de datos óptimo recursivo. Óptimo porque minimiza un criterio determinado y porque incorpora toda la información que se le suministra para determinar el filtrado. Recursivo porque no precisa mantener los datos previos, lo que facilita

su implementación en sistemas de procesado en tiempo real. Por último, es un algoritmo de procesado de datos, ya que es un filtro pensado para sistemas discretos.

El objetivo del filtro de Kalman es estimar los estados de una manera óptima, de manera que se minimiza el índice del error cuadrático medio, [34].

En principio R. E. Kalman empezó a describir esta nueva aproximación del problema de filtrado y predicción de manera discreta, describiendo una solución recursiva. En el filtro de Kalman es necesario recurrir a dos modelos, el modelo del proceso, que describe como un sistema cambia a través del tiempo, y el modelo de medición, que describe la relación entre las cantidades medibles y los estados del sistema. A estos modelos les son agregados un modelo de los ruidos del proceso y el ruido de la medición, [35]. Es muy complejo obtener un modelo del ruido entonces lo que generalmente se asume es que el ruido tiene la propiedad de ser un proceso de ruido blanco gaussiano con media igual a cero.

donde *x* es el vector de estado, cada variable que compone este vector describe un estado del sistema, F es la matriz del sistema y describe la forma en que el sistema varía con el tiempo y otros parámetros, *w* es el proceso de ruido que describe el nivel de incertidumbre que se tiene con respecto al modelo del sistema, en el modelo de medición *z* es el vector de medición cada variable representa una cantidad medible por los sensores, H es la matriz de medición esta matriz representa la relación entre las cantidades medibles y los estados del sistema, y *v* es un vector que describe el ruido de los sensores. Estas ecuaciones representan modelos continuos pero en la realidad se trabaja con maquinas digitales por lo que hay que discretizarlo, y queda como:

Donde x_k y z_k son los vectores de estado y medición en el tiempo t_k, es la matriz de transición que describe como el sistema cambia en el tiempo t_{k+1} de los anteriores estados del sistema, H_k es la matriz de medición y por último, w_k y v_k son los procesos de ruido, estos últimos deben de ser descritos como procesos estocásticos a través de dos matrices de covarianza Q_k y R_k indicando el grado de aleatoriedad del ruido, [36] y [37].

El filtro de Kalman es iterativo y sigue el siguiente proceso:

1. Obtener los modelos del proceso del sistema y el proceso de medición como en (1.2).
2. Previo a empezar el proceso de iteraciones, se debe definir un vector de estado inicial x_0 y una matriz de error de covarianza inicial P_0.
3. Los estados estimados (denota que corresponde a una estimación) y la matriz de error de covarianzas junto con y Q_k que son usados para realizar las predicciones del vector de estado del sistema para el siguiente escalón de tiempo y también para la predicción de la matriz de covarianzas , (¨ este

símbolo representa una predicción o estado *a priori* que indica que se da antes de una medición), a esto usualmente se le conoce como propagación.

4. Se usa y para calcular la ganancia del filtro .

5. Se obtiene un nuevo conjunto de mediciones y junto con , y para calcular el vector .

6. De las matrices y se calcula la matriz de estimación de covarianza del error s , y se repite este proceso desde el punto 3, [38].

Este proceso se ilustra en la figura 5.1 en forma de diagrama de flujo, [39] y [44].

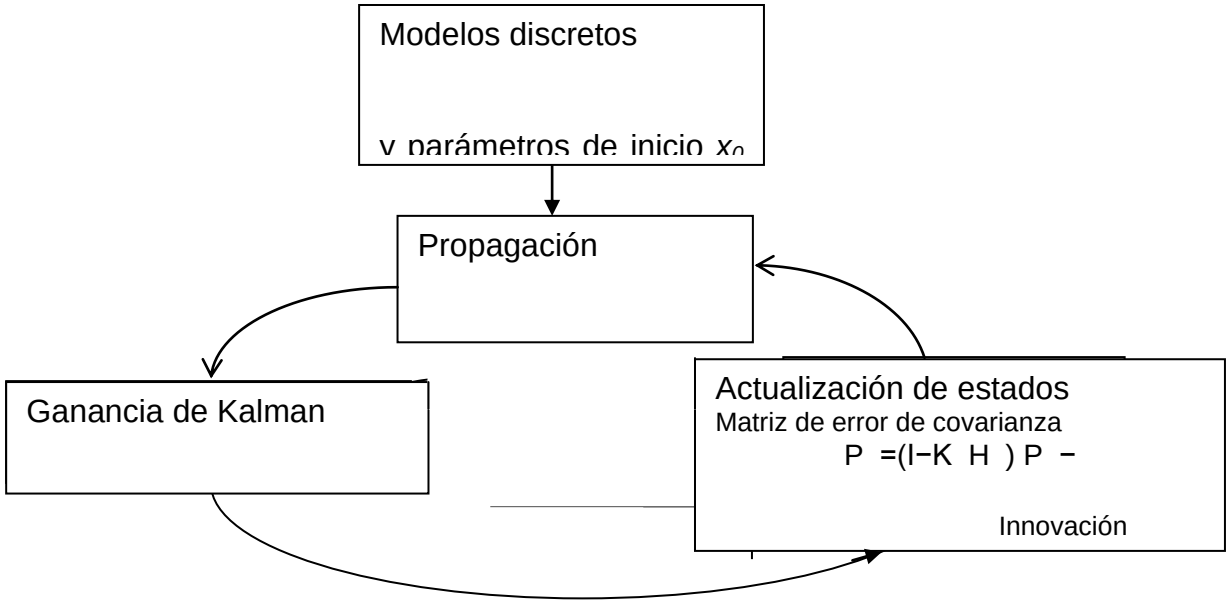


Figura 5. 1 Filtro de Kalman discreto.

Cabe recalcar que este proceso del filtro de Kalman es aplicable solo sobre sistemas lineales, [41] y [42]. Desgraciadamente esencialmente todo sistema que se desee modelar en la realidad implica que es un sistema no lineal, por lo que el método del filtro de Kalman tal cual no podría trabajar adecuadamente con un sistema real, es por ello que han surgido variaciones del filtro que permiten trabajar con él. Una de las variaciones del filtro de Kalman que ha surgido es Filtro de Kalman Extendido (EKF, Extended Kalman Filter), este método ha

sido múltiplemente utilizado por muchas décadas en una gran cantidad de aplicaciones incluyendo el tratado del problema de la estimación de la orientación de satélites. El filtro EKF guarda la misma estructura que el método lineal, solo que linealiza todas las transformaciones no lineales y substituye por matrices Jacobianas las transformaciones lineales que se presentan en el filtro de Kalman, [40].

Aunque el EKF ha presentado mucho tiempo un buen desempeño en diversas aplicaciones, [44] sufre de varias limitaciones importantes como:

- Las transformaciones lineales solo son confiables si la propagación del error puede ser bien aproximada por una función. Si esta condición no se mantiene la aproximación lineal será muy pobre.
- Linearizar el proceso solo es posible si existe la matriz Jacobiana. Y no suele ser siempre el caso porque aunque exista la matriz el sistema puede tener discontinuidades, puede haber un cambio abrupto de parámetros y otros factores que afecten de manera importante al filtro.
- El cálculo de las matrices Jacobianas puede ser un proceso muy difícil y propenso a errores. El uso de jacobianos implica un desarrollo muy grande en código que puede hacerlo propenso a fallar o que al retomar el código sea difícil entenderlo.

Es por ello, que con el tiempo se desarrollaron técnicas para quitar aquellas deficiencias que traían los métodos de liberalización, es por ello que para el desarrollo de esta tesis se escogió el método del filtrado de Kalman Unscented, que se trata a continuación.

5.3 Filtro Kalman Unscented

El filtro de Kalman Unscented (UKF) fue propuesto por Simon J. Julier y Jeffrey K. Uhlmann en 1997, se propone una variante del filtro de Kalman que trata directamente con los modelos no lineales a través de la transformada Unscented (UT, Unscented Transform) que fue desarrollada para subsanar las deficiencias de la liberalización en el EKF, proveyendo un mecanismo más directo y explícito para transformar la información de la media y la covarianza, donde básicamente la filosofía de esta transformación está basada en la premisa de que es más fácil aproximar una distribución de probabilidad de lo que es aproximar una función no lineal arbitraria o transformación, [45].

Lo que hace la UT es escoger un conjunto de puntos, llamados *puntos sigma*, de modo que en conjunto su media y su covarianza sean características del conjunto. La función no lineal del modelo del sistema se aplica sobre cada punto escogido para formar una “nube” de puntos transformados. La estadística de los puntos transformados puede ser calculada para formar estimaciones de la media y la covarianza no linealmente transformadas. Este método guarda un parecido superficial con los filtros de partículas, pero guarda varias diferencias fundamentales, como que los puntos sigma no son dibujados aleatoriamente sino que son escogidos de manera determinística y de tal forma que exhiban ciertas características.

A pesar de su aparente simplicidad, la UT tiene un número importante de propiedades, [46].

El algoritmo trabaja con un número de puntos sigma finito, que naturalmente se presta para ser usado como “caja negra” en una biblioteca de filtrado. Dado un modelo (con las salidas y entradas apropiadamente definidas), una rutina estándar puede ser utilizada para calcular las cantidades predichas como necesarias para cualquier transformación dada.
El costo computacional del algoritmo es del mismo orden de magnitud que el del EKF.
Cualquier conjunto de puntos sigma que codifica correctamente la media y la covarianza correctamente, calcula correctamente la media y la covarianza proyectadas a un segundo orden.
El algoritmo puede ser usado con transformaciones discontinuas.

De forma conceptual el filtro UKF es conceptualmente idéntico al filtro de Kalman, es un método iterativo de estimación. La excepción radica en que este método utiliza la transformada Unscented y es necesaria la obtención de puntos sigma, así como que se utilizan de manera directa las ecuaciones no lineales del sistema para la predicción de los estados y las mediciones, y de las matrices de covarianzas asociadas. Se siguen los siguientes pasos:

Se inicializa el UKF con y , como ya se hacia en el método anterior.
Se calcula la predicción de los estados .
Se definen los *2L* puntos sigma

dónde L es el número de estados del sistema, y es un parámetro de escala.

Se usan los puntos sigma y la ecuación no lineal del proceso para transformar a los *2L* puntos obtenidos

Se calcula el vector de predicciones del estado

donde W_i es un conjunto de pesos escalares definidos como , , , y , son parámetros escalares, L corresponde a la dimensión de las variables de estado y α determina la dispersión de los puntos sigma alrededor de y está ubicado entre . β es usada para incorporar el conocimiento previo de la distribución de (para el caso de distribuciones gaussianas β = 2 es óptimo).
Se calcula la matriz de predicción de la covarianza del error:

donde es la matriz de covarianza del ruido en el proceso.

Se calcula la predicción de las mediciones y las matrices de covarianza asociadas por medio de:
Usando para encontrar los 2L puntos sigma para las mediciones como:

Se usa para calcular predicciones de las mediciones :

Se usa y para calcular la matriz de predicción de la covarianza del error del modelo de medición como:

donde es la matriz de covarianza del ruido de medición.

Se usan , , y , para calcular la matriz de predicción de la covarianza del error cruzado como:

Cuando un nuevo conjunto de mediciones se obtiene, ya es posible obtener la estimación de los estados mediante:
Se usan y para calcular la matriz de ganancias:

Se usan , , y como sigue:

Se usa , y para calcular la estimación de la matriz de covarianzas del error como sigue:

Este proceso se repite desde el paso 2 hasta el 4, [47].

Dentro del proceso de estimación cabe hacer notar que se agrega un nuevo tipo de operación que consiste en sumar un vector columna a una matriz ( ), esta operación solo consiste en sumar el mismo vector en cada columna de la matriz. Aunque el proceso del UKF es documentado como muy similar al EKF en cuanto a procesos de cálculo, al momento de pensar en llevarlo a una implementación presenta el problema de cómo realizar una raíz cuadrada de una matriz cuadrada que se requiere en el paso 2 del algoritmo, esto se vuelve complicado y es precisamente esta parte la que vuelve computacionalmente muy pesado al algoritmo, [48]. Es por ello que se creó una variante de este método de filtrado llamado

Square Root Unscented Kalman Filter (SR-UKF), el cual se decidió seria implementado finalmente y se trata a continuación.

Square Root Unscented Kalman Filter (SR-UKF)

Este método que es una variante del UKF fue propuesto por Rudolph van der Merwe y Eric A. Wan, [49]. Debido a la problemática que tenía el calcular de manera recursiva la raíz cuadrada de una matriz este método replanteo el algoritmo. Básicamente, lo que plantea es que dado que una matriz está dada como , lo que hace es que se trabaja directamente con la matriz S sin la necesidad de refactorizar para obtener la matriz **P**, esto lo logra usando tres herramientas del algebra lineal, que son la descomposición QR, la actualización del factor de Cholesky y la obtención de mínimos cuadrados eficientes. El flujo del algoritmo es muy parecido al UKF, la forma del cálculo de los parámetros es el mismo y también la de los pesos $W_i^{(c)}$ y $W_i^{(m)}$. El algoritmo sigue los siguientes pasos:

Se inicializa el SR-UKF con y .
Se calcula la predicción de los estados .
Se definen los *2L* puntos sigma :

dónde L es el número de estados del sistema, y es un parámetro de escala.

Se usan los puntos sigma y la ecuación no lineal del proceso para transformar a los *2L* puntos obtenidos:

Se calcula el vector de predicciones del estado

Se calcula la matriz de predicción de la covarianza del error:

dónde es la matriz de covarianza del ruido en el proceso, y su raíz se obtiene a partir de la descomposición de Cholesky.

Se calcula la predicción de las mediciones y las matrices de covarianza asociadas por medio de:
Usando para encontrar los 2L puntos sigma para las mediciones como:

Se usa para calcular predicciones de las mediciones .

Se usa y para calcular la matriz de predicción de la covarianza del error del modelo de medición como:

dónde es la matriz de covarianza del ruido de medición y su raíz se calcula previamente a través de la descomposición de Cholesky.

Se usan , , y , para calcular la matriz de predicción de la covarianza del error cruzado como:

Cuando un nuevo conjunto de mediciones se obtiene, ya es posible obtener la estimación de los estados mediante:
Se usan y para calcular la matriz de ganancias.

Se usan , , y como sigue:

Se usa , y para calcular la estimación de la matriz de covarianzas del error como sigue:

Este proceso se repite desde el paso 2 hasta el 4, [49] y [50].

Planteamiento del problema de estimación de la orientación usando el SR-UKF

Para un satélite real, como ya se ha visto en el capítulo 3, es necesario el uso de varios sensores divididos en dos clases de sensores, los sensores de referencia y los sensores inerciales, pero cada sensor al no ser perfecto agrega un factor de ruido dentro de cada lectura, y si cada lectura fuera utilizada directamente dentro de algún método determinístico para obtener la orientación del satélite esta medida estaría contaminada por el ruido de cada lectura realizada de los sensores. El SR-UKF como ya se ha visto es un método analítico que una vez debidamente ajustado, es capaz de fusionar las medidas de sensores de distinta clase para obtener la orientación sin los procesos de ruido que aquejan a los sensores.

Dado que el estimador va de la mano con el controlador del sistema de orientación es necesario plantear los estados necesarios a estimar a partir de las necesidades del control. Para realizar el control de orientación es necesario estudiar la dinámica y la cinemática del satélite en el espacio, el control no es objeto de esta tesis y estas necesidades ya se han cubierto en otros trabajos. De trabajos de tesis anteriores se ha llegado a la conclusión de que el vector de estados necesarios a estimar son las velocidades angulares del satélite en

sus 3 ejes, así como obtener su representación de la orientación a través de un cuaternión unitario, por lo que el vector de estado queda como:

donde ω corresponde a las velocidades angulares del satélite sobre sus diferentes ejes y $\boldsymbol{q}$ corresponde al cuaternión que representa su orientación o actitud, por lo que se tienen 7 estados (3 componentes de la velocidad angular y 4 componentes del cuaternión). Para el inicio del algoritmo es necesario establecer un valor inicial en el vector de estados y por ahora se supone que este inicia de 0, pero lo mejor es agregar una pequeña rutina previa que tome un par de muestreos de los sensores para establecer el vector de estado inicial . La matriz S_0 corresponde a la matriz de covarianzas que corresponden al ruido del proceso, este valor inicial es propuesto y puede darse en función del factor de ruido de los sensores si es que el fabricante ofrece un número aproximado de este factor, o bien obtener la matriz de manera experimental.

Para el modelo del proceso consideramos que este es un sistema discreto y que posee parámetros fijos, entonces la matriz de transición de estados () puede expresarse como

donde corresponde al tiempo de muestreo, [51].

Para la tarea de la estimación de la sola orientación del satélite solo es necesario estudiar su cinemática, como se vio anteriormente, por lo que la matriz F se determinó a través del modelo cinemático que corresponde a la rotación de un cuerpo rígido, por lo que el modelo de proceso queda como

–

En la parte del modelo de medición del sistema, las mediciones se presentan de manera directa, ya que todos los parámetros que era necesario medir se presentaban directamente a través de los sensores, la parte de velocidad angular directamente a través de los giróscopos y la parte del cuaternión directamente del magnetómetro y el acelerómetro triaxiales mediante un método determinístico para obtener un cuaternión como medición por lo que la matriz que describe al proceso de medición es una matriz identidad, es decir

Para más información sobre la implementación de este método ver la referencia [39]

Capítulo 6. Diseño y desarrollo de las tarjetas de estabilización activa en tres ejes para SATEDU

6.1 Introducción

El presente capítulo describe los requerimientos y características del subsistema de estabilización activa en tres ejes, así como de los módulos que lo forman. Se describe la implementación del algoritmo de estimación y filtrado SR-UKF y como se adquieren los datos de los sensores de navegación. Se presentan los detalles generales acerca del desarrollo de hardware para las tarjetas de estabilización activa y de drivers de potencia, cuyas etapas y consideraciones de diseño se describen a continuación. Primero revisaremos la tarjeta de drivers de potencia para los motores y más adelante la tarjeta para el MCU Stellaris.

6.2 Adquisición de datos de los sensores

Para utilizar el MCU stellaris como computadora se requiere que este obtenga los datos de los sensores los procese y en base a este determine la orientación del satélite, posteriormente debe generar salidas PWM para controlar las ruedas inerciales.

Figura 6.1 Adquisición de datos.

Los sensores de navegación inercial se encuentran dentro de una tarjeta comercial de la compañía *Sparkfun* figura 6.1, la cual contiene los tres sensores requeridos para experimentar con métodos de estimación. Estos son los siguientes sensores:

- ***Magnetómetro triaxial (HMC5843, Honeywell):*** es un circuito de montaje superficial, que contiene sensores magnetoresistivos más un circuito integrado de aplicación especifica desarrollado por Honeywell que contiene una fase de amplificación, cancelación de offset, un ADC de 12 bits y un bus serial I2C. Este dispositivo utiliza la tecnología Magnetoresistiva Anisotropica de Honeywell que provee ventajas sobre

otros dispositivos magnéticos. Estos sensores se caracterizan por su sensibilidad y linealidad sobre cada eje. Son de rango ajustable, uno de ellos de hasta ±6.5 gauss, y operan con un voltaje de 3.3V. La lectura de datos se hace por cada eje y tienen una resolución digital de 12 bits en forma de complemento a 2. Ofrece también 2 modos de lectura, una sola lectura y modo de lectura continua, de los cuales se seleccionó el último modo debido a que el circuito obtiene las lecturas de manera automática. Adicionalmente, es posible ajustar el muestreo en 0.5, 1, 2, 5, 10, 20 y 50 Hz, de los cuales se seleccionó el modo de 50 Hz [52].

Acelerómetro triaxial (ADXL345, Analog Devices): es un dispositivo de bajo consumo, se alimenta con 3.3V, tiene una resolución de 10 bits y un rango de medición de hasta ±16 *g*. La salida se da en modo de 16 bit con complemento a 2 y es posible acceder al circuito ya sea por el modo SPI o I2C, figura 5.6. Este dispositivo es muy utilizado dentro de aplicaciones móviles, por ejemplo en dispositivos donde se requiere medir la inclinación, o bien donde es necesario medir la aceleración producida por el movimiento. Debido a su rango ajustable de medición se puede obtener una resolución de 4 [m*g*/LSB] que le permite realizar mediciones de inclinación de menos de 1˚.

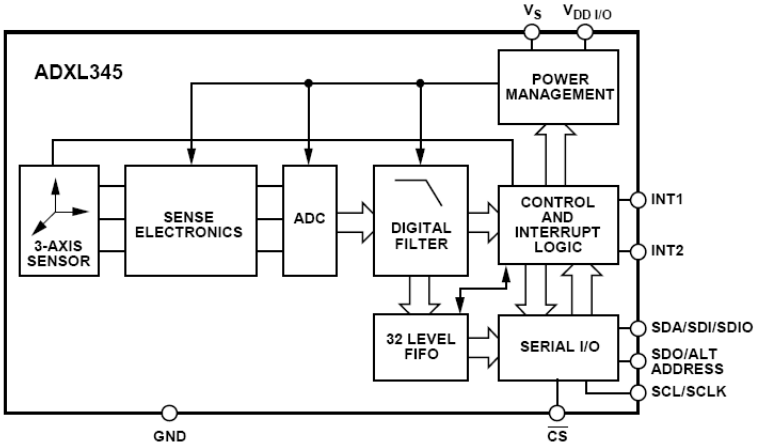


Figura 6.2.Diagrama de bloques del acelerómetro ADXL345.

Giróscopo triaxial (ITG-3200, InvenSense): este circuito fue uno de los primeros giróscopos electrónicos en integrar 3 ejes de sensado con una salida digital para aplicaciones en juegos electrónicos y en dispositivos cursores en 3D. Parte de las características importantes son el mejoramiento del *bias* y la estabilidad del sensor ante cambios de temperatura reduciendo la necesidad de calibrar el sensor. Presenta una disminución en el ruido de baja frecuencia con respecto a dispositivos anteriores simplificando el desarrollo de alguna aplicación sencilla al tiempo que los controles tienen mejores respuestas. Este circuito integra en su parte digital a convertidores ADCs de 16 bit para digitalizar las entradas de los giróscopos, un filtro paso-bajas con ancho de banda ajustable de manera digital y una interface I2C de hasta 400 [kHz] de velocidad, figura 5.7. Además incluye un sensor de temperatura y un oscilador interno. Las mediciones en cada eje de los giróscopos se dan en una velocidad angular de

[°/s] en modo de complemento a 2. Opera con rangos de voltaje de 2.2 a 3.6V y posee un rango de medición de ±2000 [°/s] con una sensibilidad de 14.375 LSBs por [°/s].

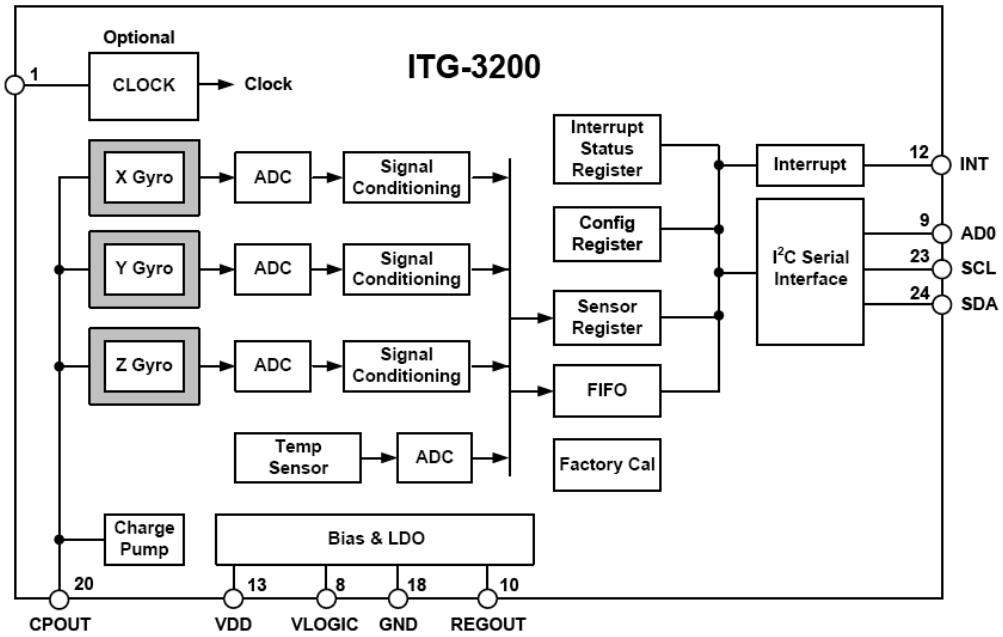


Figura 6.3 Diagrama de bloques del giróscopo ITG-3200.

Para conocer más acerca del protocolo I²C se recomienda consultar la referencia [52] y el apéndice.

6.3 Arquitectura del hardware de control de estabilización por ruedas inerciales y bobinas de torque magnético en la MSA

Se utilizaron motores con escobillas Faulhaber, coreless, 12 V, 112mA y 9900 RPM. La mesa suspendida en aire es ligera ya que esta hecha de madera MDF en lugar de aluminio.

Se Fabricaron 3 ruedas inerciales y son completamente operativas. Las ruedas inerciales fueron diseñadas para mover la mesa suspendida en aire en tiempos pequeños. Estas miden 10 cm de diámetro. Se obtuvieron resultados teóricos para estas ruedas con tiempos de reacción para mover la mesa suspendida en aire de 21-28 segundos con ruedas de 40 a 47 gramos.

6.4 Arquitectura electrónica de la tarjeta de reguladores y drivers de potencia

Esta tarjeta es el elemento que dota de potencia a los motores de CD para su funcionamiento, también cuenta con un regulador para alimentar tarjetas externas a 5V.

6.4.1 Características de la tarjeta de reguladores y drivers de Potencia

La tarjeta de reguladores y drivers de potencia posee una forma cuadrada de 8.9cm x 8.9cm con componentes por ambos lados. La tarjeta está compuesta por tres circuitos de puentes H para los motores vinculados a las ruedas inerciales, un regulador de voltaje a 5V y cuenta con conectores de costilla para ser compatible con SATEDU. Adicionalmente se anexaron LEDs para indicar el funcionamiento de algunas partes de esta tarjeta.

Puentes H empleados

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.

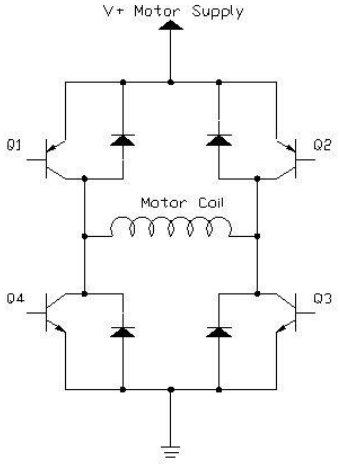


Figura 6.4 Puente H.

Refiriéndonos a la figura 6.4, 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 I1 es 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 6.4 con la corriente i2. 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.

Inputs			Output Mosfets (*)
	IN1	IN2	
V_{EN} = H	L L H H	L H L H	Sink 1, Sink 2 Sink 1, Source 2 Source 1, Sink 2 Source 1, Source 2
V_{EN} = L	X	X	All transistors turned oFF

L = Low H = High X = DON't care
(*) Numbers referred to INPUT1 or INPUT2 controlled output stages

Tabla 6.1 Entradas y salidas de los puentes H.

Los Puentes H utilizados en esta tarjeta son de tipo Full-Bridge, o sea puente completo, y son los LM3202, que nos ofrecen varias ventajas como no requerir demasiados componentes extra solo utilizan unos capacitores de *bootstrap* para la salida de la corriente. Pueden operar a voltajes de 42V y muy altas velocidades, y sus entradas lógicas son compatibles con TTL, CMOS y mC.

Diseño electrónico asociado a puentes H para SATEDU

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, [53]. 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 6.5

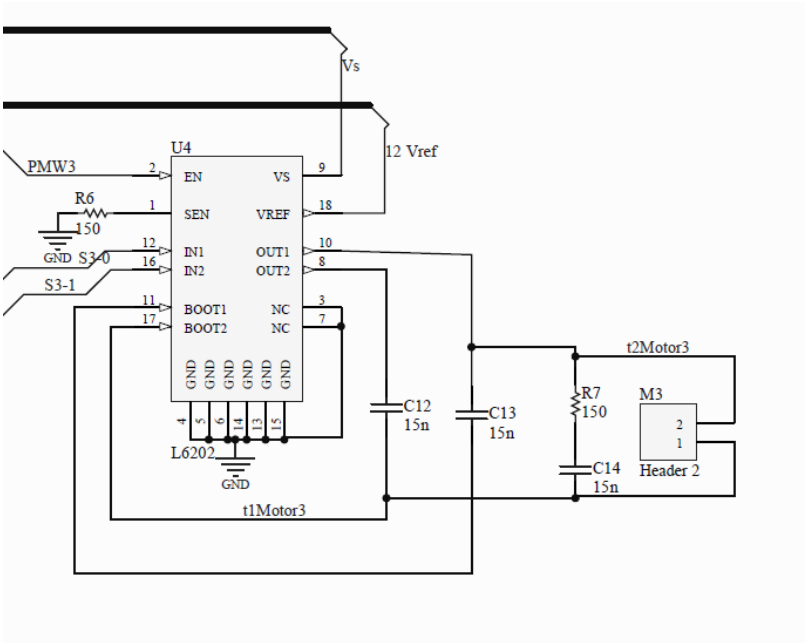


Figura 6.5 Esquemático de los puentes H L602.

Regulador de Voltaje

Este regulador se utiliza para alimentar tarjetas externas que requieran un voltaje constante de 5V, como son las tarjetas de desarrollo comerciales, tanto para DSPs, como para FPGAs, también puede dar alimentación a SATEDU. Como uno de los requisitos es que el voltaje se mantenga constante se optó por un regulador fijo a 5V.

Se escogió el LM2575, ya que este al ser switcheado no requiere muchos componentes extras además de no calentarse y disipar mejor el calor, especialmente con entradas altas de voltaje como es nuestro caso ya que el voltaje de la fuente proviene de una batería a 14V. Utilizando la aplicación típica y su diagrama de bloques para obtener los 5V, figura 6.6.

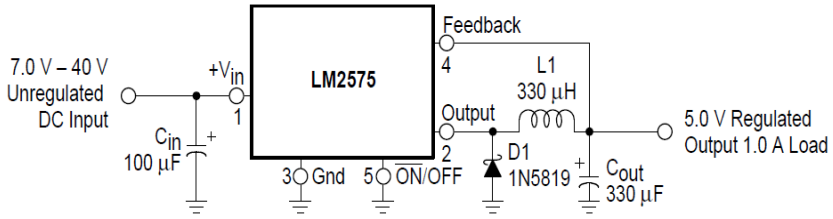


Figura 6.6 Esquemático del Regulador LM2575.

A este diagrama solo se le agregó un led para saber cuando esta prendido el regulador. Figura 6.7

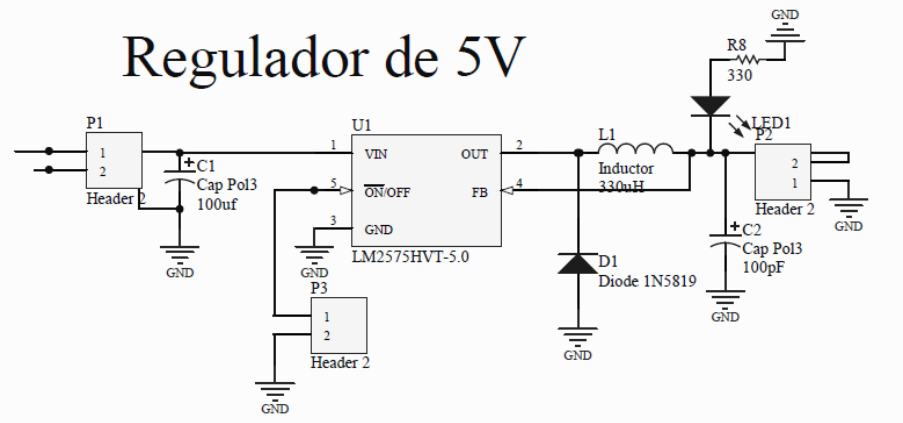


Figura 6.7 Esquemático del Regulador LM2575.

6.4.2 Arquitectura electrónica de la tarjeta MCU Stellaris

La tarjeta que alberga el MCU Stellaris de Texas Instrumentes posee una forma cuadrada de 8.9cm x 8.9cm con componentes por ambos lados y conectores de costilla para garantizar su compatibilidad con SATEDU.

El microcontrolador principal es un Texas Instruments Stellaris LM4F231H5QR, el cual posee un núcleo ARM Cortex 4 con dos bancos de memoria Flash de 128 KB y 32 KB de memoria SRAM, además de diversos periféricos para comunicaciones.

6.4.2.1 Procesador

La serie Stellaris forma parte de la familia de microcontroladores con memoria Flash de Texas Instruments basados en los procesadores RISC de 32 bits de alto rendimiento ARM Cortex-M4. En particular se seleccionó este modelo LM4F231 por la facilidad de traspasar los programas en C de un DSP a este, por que cuenta con varias librerías que se pueden utilizar ya que hay muchos desarrollos para estas lo que implica una mejora a nivel de hardware e implementación del firmware y por que puede realizar operaciones con punto flotante además de contar con 16 líneas de PWM, [55].

También se eligió debido a que ya se tenían experiencias con dispositivos similares. Actualmente la mayoría de los entornos de desarrollo de software de estos dispositivos ya poseen bibliotecas para el manejo del punto fijo por lo que realizar programas para estos microcontroladores resulta más sencillo. A continuación en la figura 6.6 se muestra el diagrama de bloques de la familia Stellaris LM4F2315QHR.

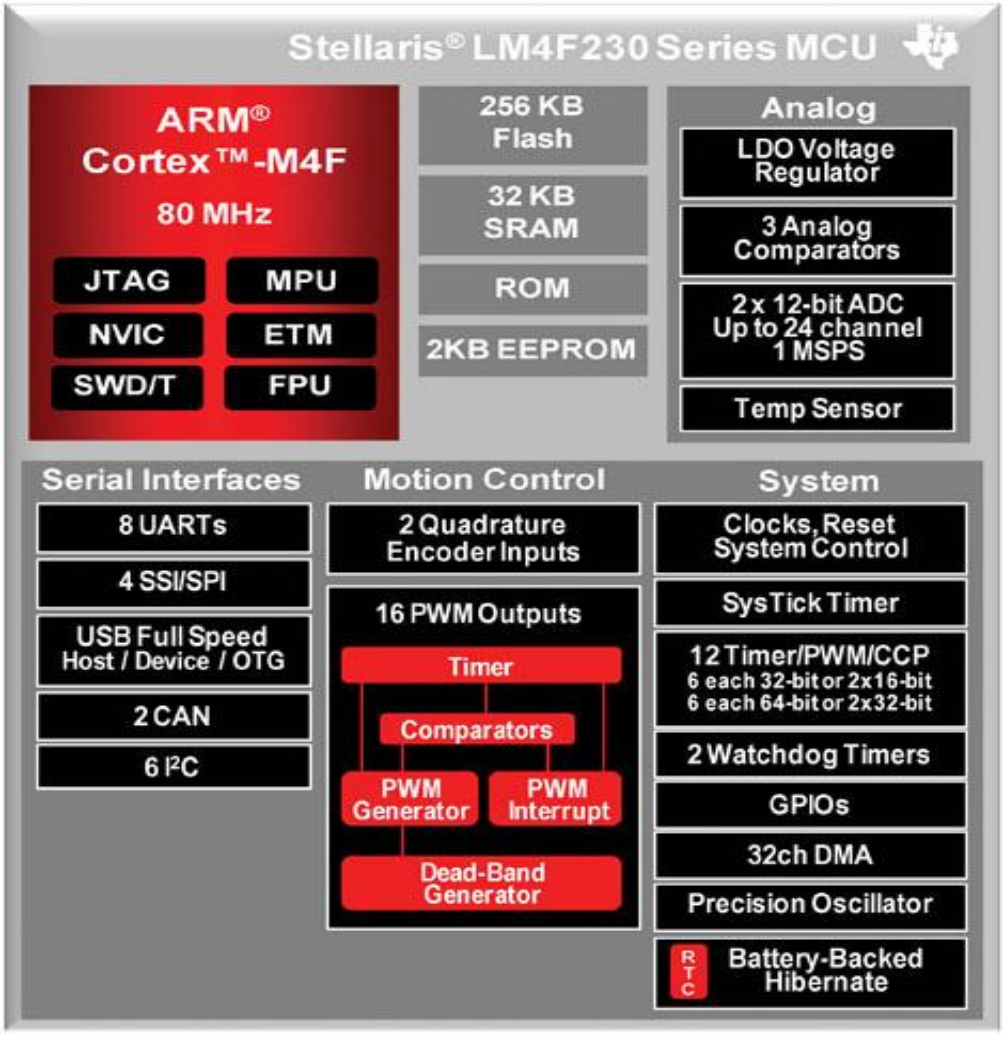


Figura 6.8 Familia de microcontroladores Stellaris.

Otra ventaja es que incluye 2 KB de EEPROM, un empaquetado de 64-LQFP, 16 canales de PWM, y este MCU puede soportar el StellarisWare que son pequeños programas preprogramados en la memoria. La tabla 6.2 resume las principales características.

	LM4F231H5QR
Pin/Package	64LQFP
I2C	6
CPU	ARM Cortex M4F
ADC Resolution (Bits)	12
ADC Channels	12
Flash (KB)	256
RTC	Yes
Max Speed (MHz)	80
QEI	2
I2S	No
Battery-Backed Hibernation Module	No
Internal LDO Voltage Regulator	Yes
EPI/EMIF	No
Ethernet (10/100 MAC+PHY)	No
IEEE 1588	No
USB D, H/D, or OTG	OTG
Memory Protection Unit (MPU)	Yes
Motion PWM	16
Boot Loader in ROM	Yes
Watchdog Timers	2
ADC Sample Rate (kSPS)	1000
Analog Comparators	2
SSI/SPI	4
Digital Comparators	16
Maximum 5-V Tolerant GPIOs	49
ADC Units	2
Internal Temp Sensor	1

CAN MAC	2
16 MHz Precision Oscillators	Yes
UART	8
GPIOs	49
ROM Software Libraries	Yes
SysTick	Yes
10/100 MAC with MII Interface	No
SRAM (kB)	32

Tabla 6.2 Características del MCU LM4F231H5QR.

El procesador cuenta con 32-bit ARM Cortex-M4F, opera a 80-MHz; y puede ejecutar hasta 100 DMIPS.

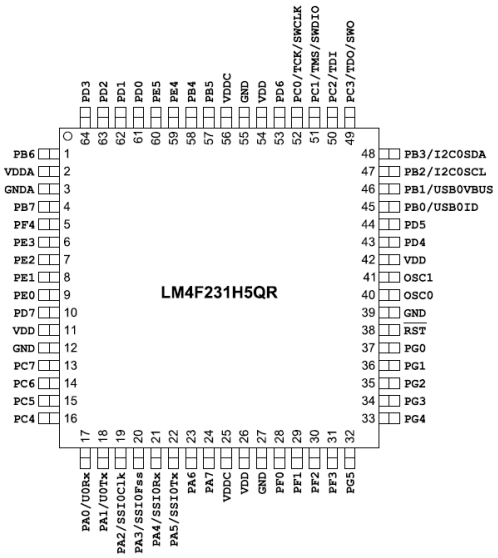


Figura 6.9 Patigrama del MCU Stellaris LM4F2315QHr.

6.4.2.2 Protecciones de efecto latch up para MCU stellaris

Cuando un dispositivo semiconductor, como un circuito integrado, se expone a la radiación del espacio, en función de su dosis de carga eléctrica que acumule, podrá experimentar un incremento en su consumo de corriente eléctrica. Al presentarse tal efecto, llamado Latch-Up, y de no detenerlo a tiempo, provocará que el dispositivo deje de funcionar permanentemente.

Una manera de detener el efecto latch-up es desenergizando el dispositivo. En la CV la electrónica necesaria para detectar el efecto latch-up consiste de un sensor de corriente

MAX4071 con salida de voltaje y un comparador de voltaje LM6511 que genera una señal eléctrica para que la tarjeta de potencia desenergize el dispositivo que presenta el efecto latch-up.

La tarjeta con el MCU Stellaris cuenta con protección para el microcontrolador pues es el componente de mayor escala de integración y por tanto más susceptible de presentar el efecto latch-up ya que puede estar trabajando por largos periodos de tiempo. La lectura de la corriente consumida se realiza por medio del convertidor analógico digital A/D del MCU, [56].

6.4.2.3 Sensor de corriente MAX4071

El sensor MAX4071, permite medir la cantidad de corriente que circula por una carga. Para esto, la carga se conecta en serie con una resistencia de sensado (Rsense) haciendo que la corriente que circula por dicha resistencia sea la misma que circula por la carga. La fuente que alimenta Rsense y la carga es la misma, Figura 6.8.

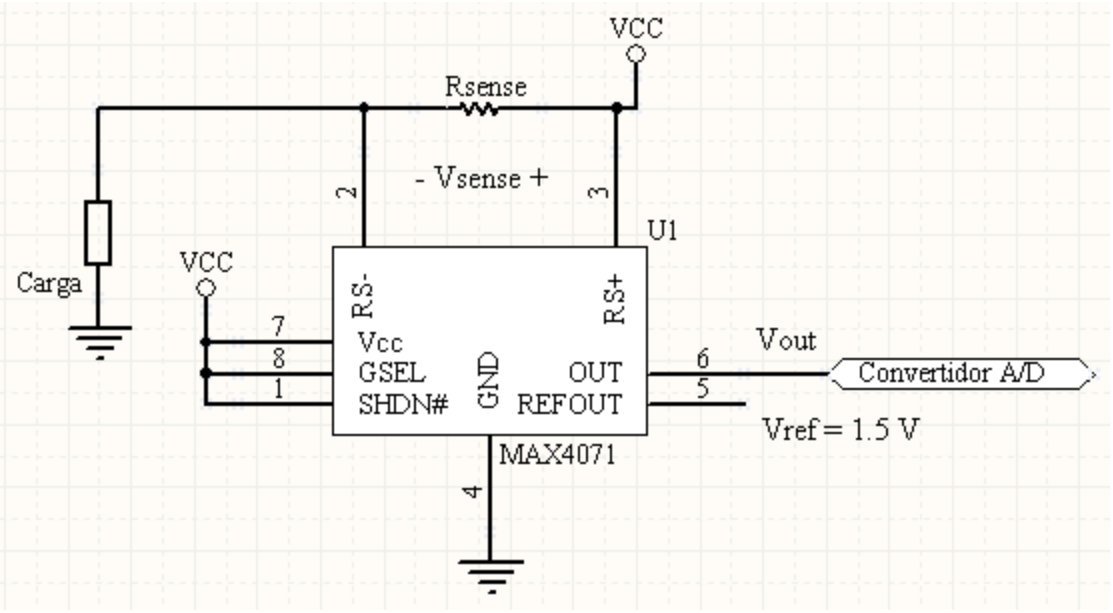


Figura 6.10 Esquemático sensor MAX4071.

Como salidas se tienen: un voltaje de referencia en REFOUT de 1.5 V y un voltaje medido con respecto a tierra, Vout, que es directamente proporcional a la corriente que fluye por la resistencia Rsense y esta dado por:

V out = G sel R sense I Rsense + 1.5[V]--------(1)

Donde Gsel representa una ganancia que es igual a 100 V/V, cuando el pin GSEL es conectado a la fuente de 3.3 V.

En la tabla 6.6 se muestran los valores recomendados por el fabricante para la selección de Rsense de acuerdo al valor de corriente que se desea monitorear y para el caso de una ganancia de 100V/V. Al seleccionar el valor de Rsense se debe elegir el mínimo valor

posible para que la caída de voltaje en Rsense sea mínima. La resistencia Rsense debe tener una inductancia pequeña si la corriente a monitorear varía mucho, [57].

Corriente máxima [A]	Rsense [mΩ]	Vsense [mV]	Vout-Vref [V]
10	5	50	5
5	10	50	5
2.5	20	50	5
0.5	100	50	5
0.05	1000	50	5

Tabla 6.3 Valores Rsense.

6.4.5.4 Comparador LM6511

Con una fuente de alimentación de 2.7 a 3.6 V y tiempo de respuesta de 180 ns, el LM6511 funciona como comparador de un voltaje de referencia y el voltaje de salida del sensor de corriente MAX4071.

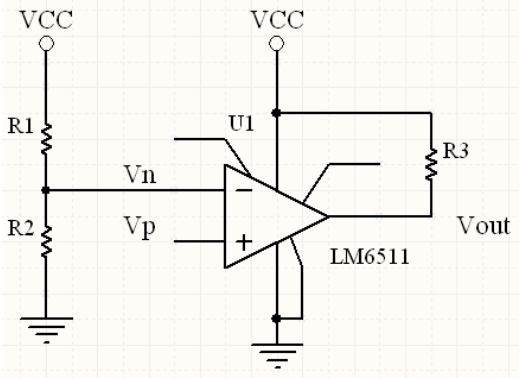


Figura 6.11 comparador LM6511.

Con esta ecuación se obtiene el valor de R1.

$$R_1 = R_2\left(\frac{V_{cc}}{V_{ref}} - 1\right)[\Omega]\text{--------}(2)$$

De esta manera dado un valor de R2 se obtiene el de R1. Si Vp es menor al voltaje de referencia Vref entonces tenemos a la salida un voltaje igual a 0 y si Vp es mayor al voltaje Vref tenemos una salida igual a Vcc.

El MCU Stellaris trabaja con una corriente máxima de 40 mA. De la tabla 6.6 se observa que la resistencia de sensado más conveniente es la de 100 mΩ teniendo la capacidad de sensar hasta 500 mA. La de 1Ω implica un Vout mayor de 3.3 V, voltaje de saturación, que impide el uso correcto del comparador.

De la ecuación (1) el valor Vout para una corriente de 30 mA es:
Vout = (100)(0.1)(0.03) + 1.5
Vout = 1.8 [V]

Sustituyendo en la ecuación (2) Vref = 2.2 V y R2 = 2.7 kΩ tenemos que R1 vale:
R1 = 2.7k (3.3/1.8 -1)
R1 = 2.26 [kΩ]

La protección contra efecto latch-up del MCU queda implantada de la siguiente forma:

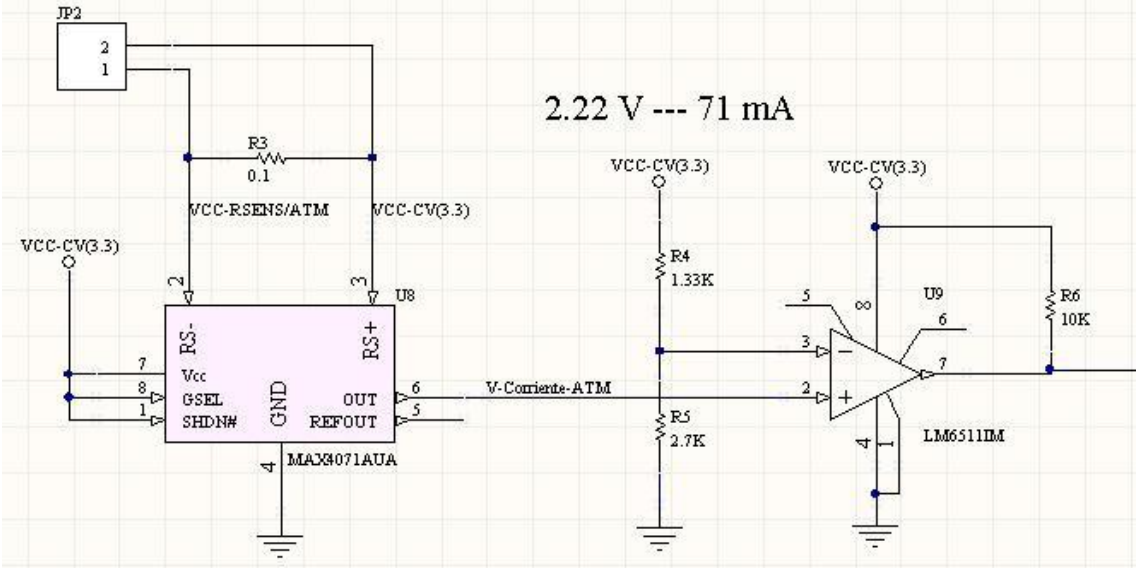


Figura 6.12 Protección de efecto latch-up para MCU.

Donde Vcc-Rsens/RAM se conecta a la carga, el MCU y V-Corriente-RAM se envía al convertidor A/D para tomar la lectura de la corriente consumida, [58].

6.5 Desarrollo de circuitos impresos de las tarjetas de estabilización activa utilizando Protel DXP

El desarrollo de las tarjetas impresas para 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], [59] 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, XP, Vista, 7.
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.

Ruteado 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.

Una vez que se tiene el esquemático de las tarjetas como se muestra en las figuras 6.8 y 6.9 se procede a generar el PCB o sea ya como quedará la tarjeta impresa.

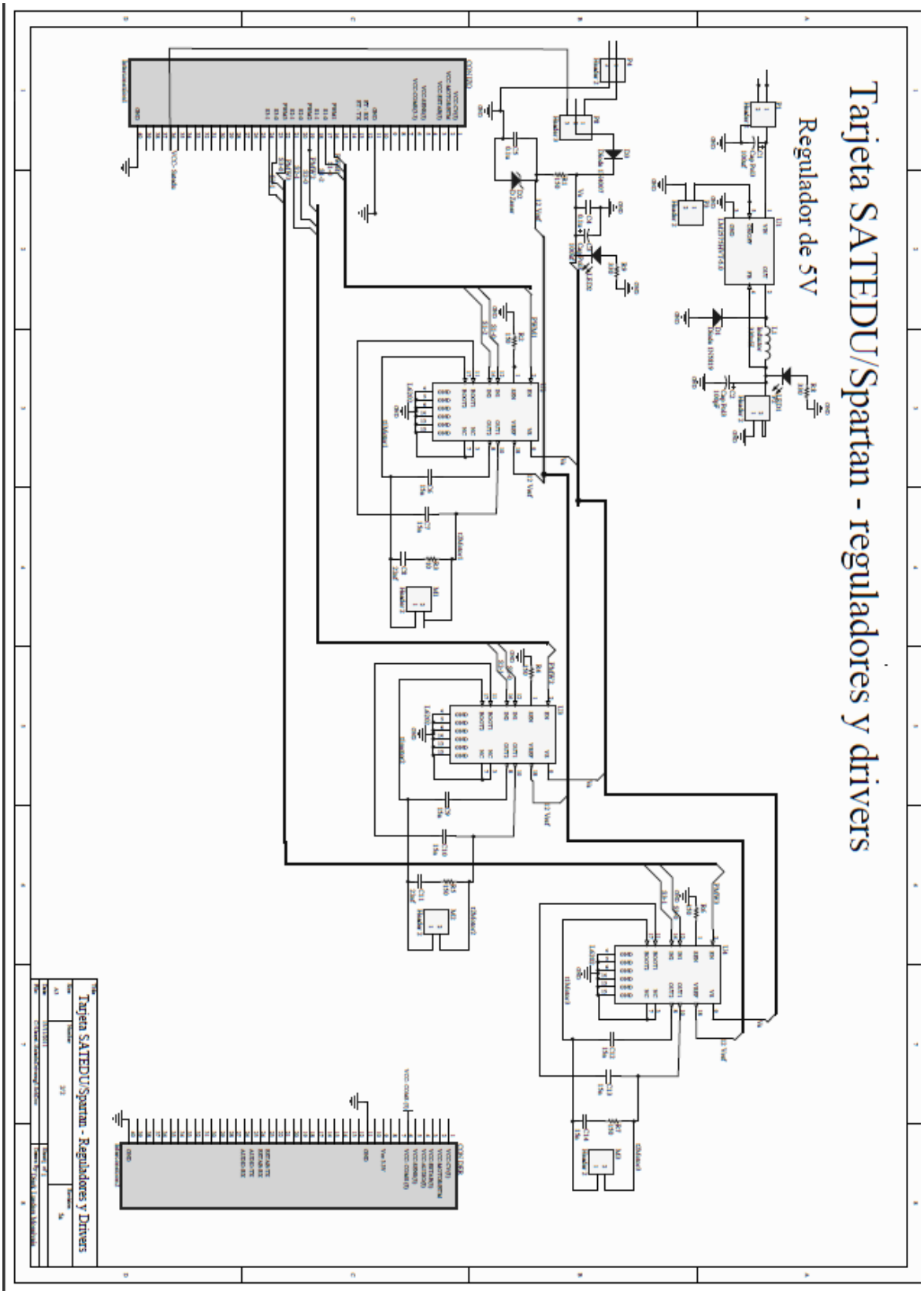


Figura 6.13 Esquemático de la Tarjeta de reguladores y drivers de potencia.

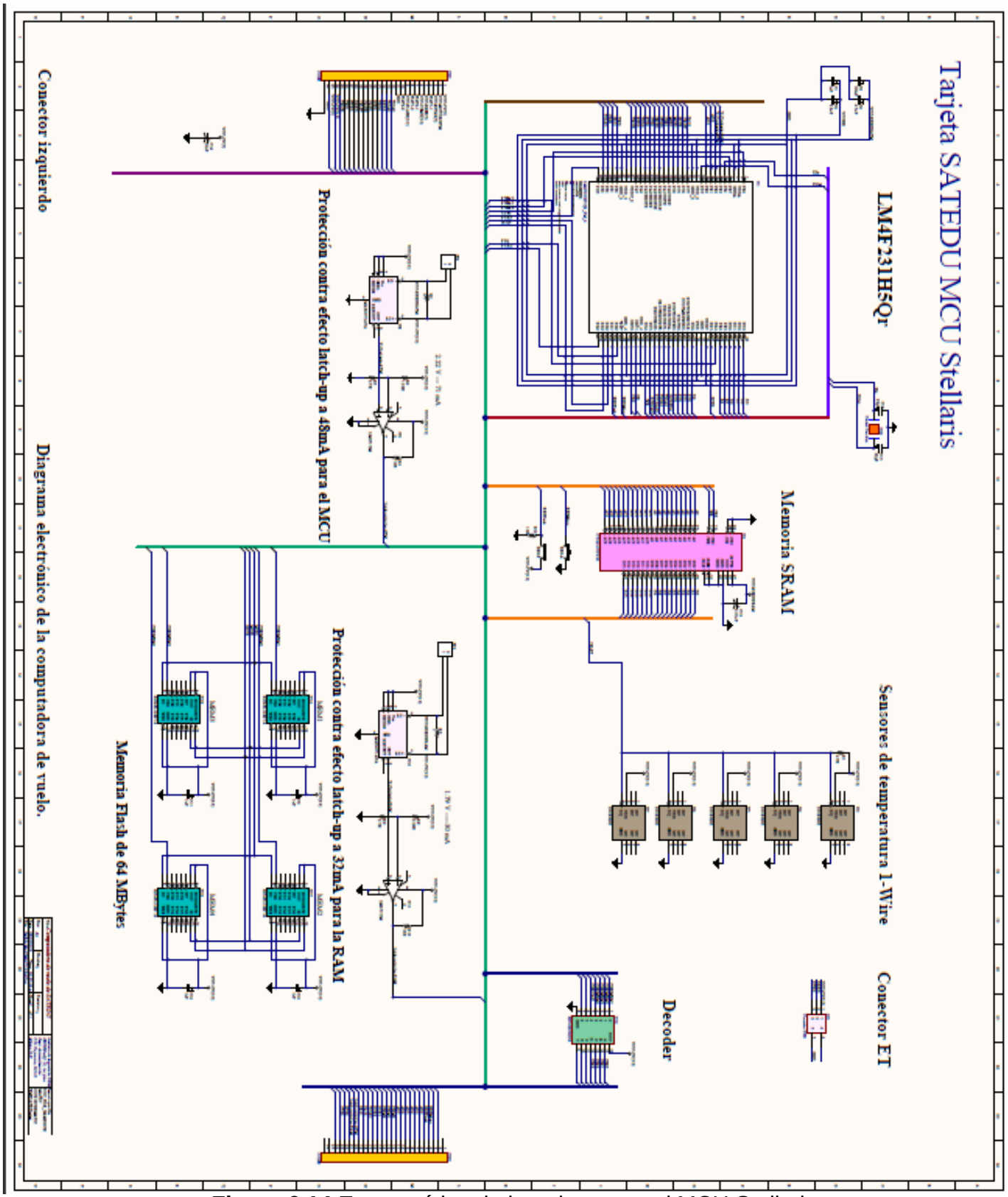


Figura 6.14 Esquemático de la tarjeta para el MCU Stellaris.

6.6 Manufactura, ensamble y pruebas de las tarjetas de estabilización activa

Las tarjetas de estabilización activa y la de reguladores y drivers, como se había mencionado anteriormente, estarán conectadas una sobre la otra siguiendo la configuración de SATEDU. Debido a problemas de manufactura la tarjeta para el MCU Stellaris no ha sido terminada, sin embargo la de drivers esta completamente funcional.

Una vez fabricado el circuito impreso de la tarjeta 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 la misma. 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 en las tarjetas, este proceso fue efectuado de forma modular, siguiendo los diagramas de las arquitecturas de la tarjeta de estabilización, 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, [9] y [39].
En la figura 6.10 se muestra la tarjeta antes y en la figura 6.11 después de ensamblarla.

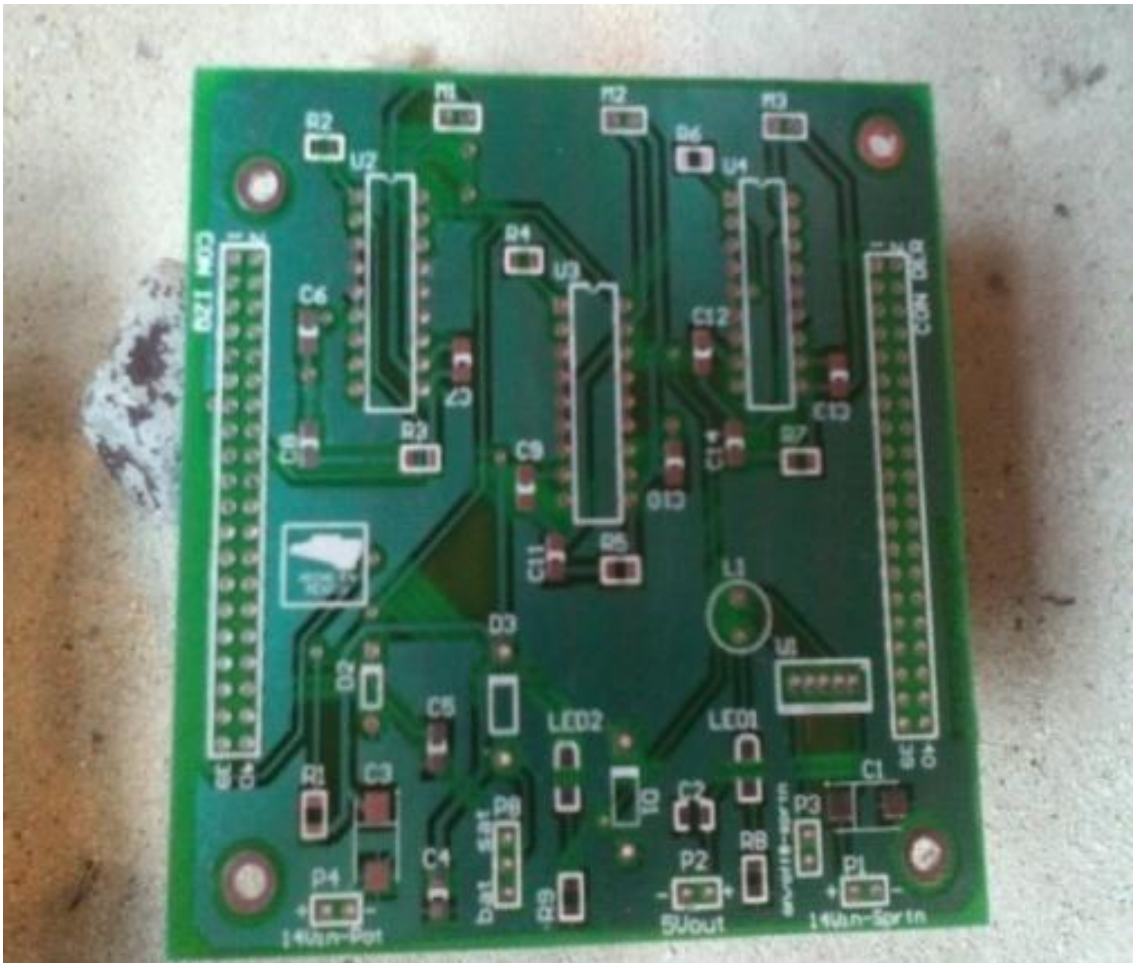


Figura 6.15 Tarjeta de drivers elaborada en esta tesis antes del ensamble.

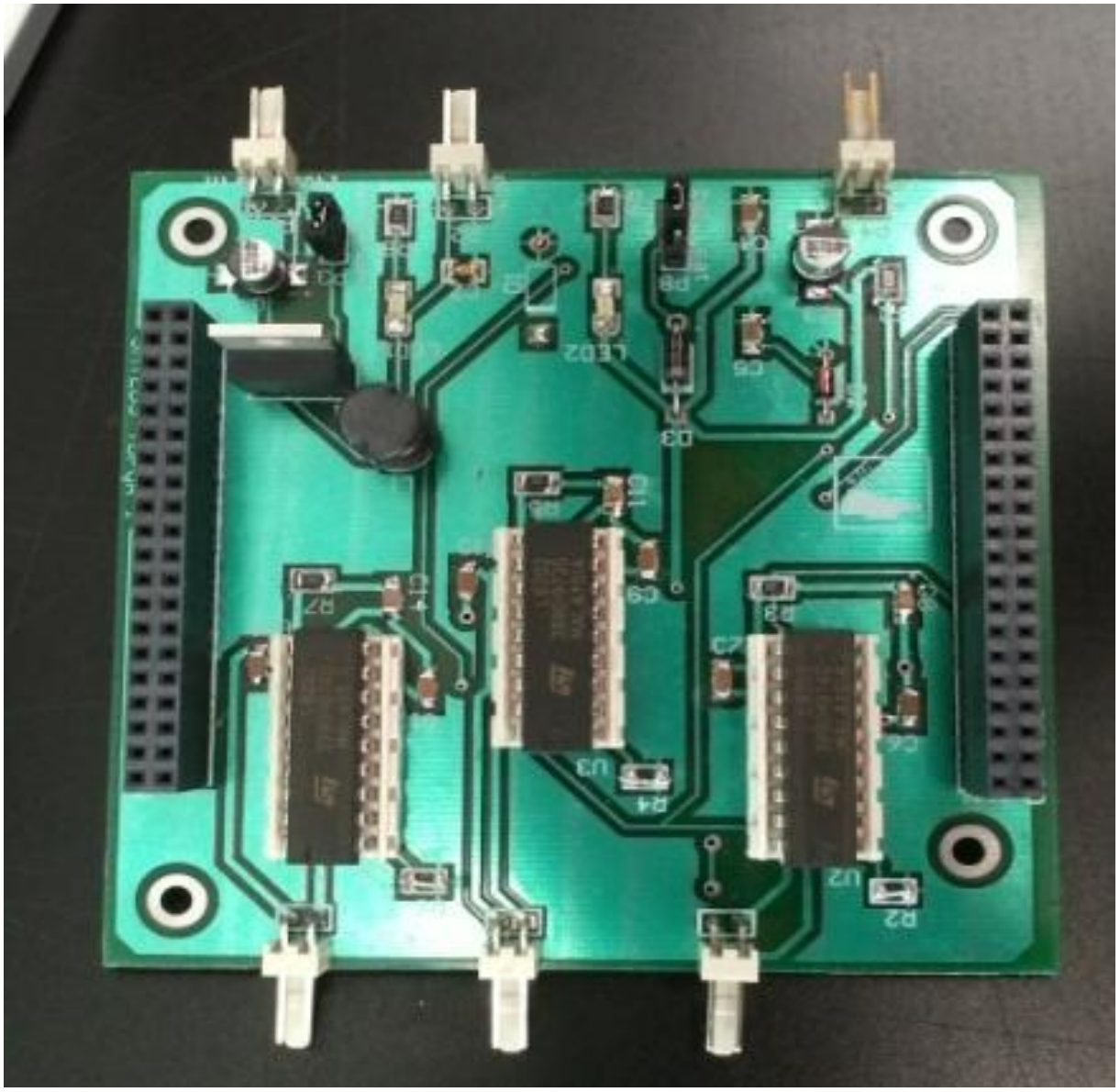


Figura 6.16 Tarjeta de drivers elaborada en esta tesis después del ensamble.

El mismo software Altium DXP permite generar una vista en 3D de como quedará la tarjeta final de estabilización con el MCU Stellaris que se muestra a continuación, figura 6.12 y figura 6.13.

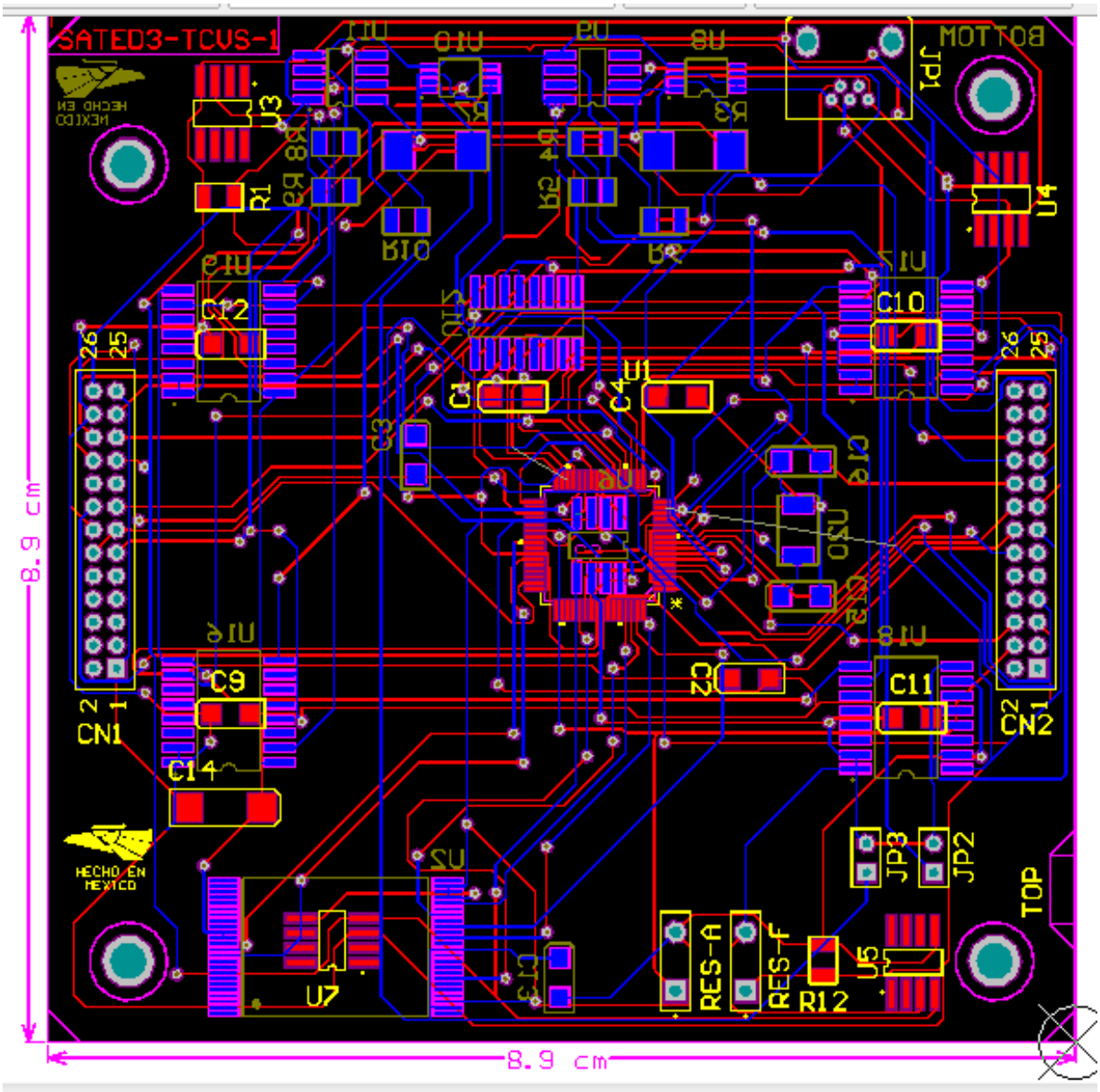


Figura 6.17 Tarjeta final de estabilización desarrollada para el MCU Stellaris.

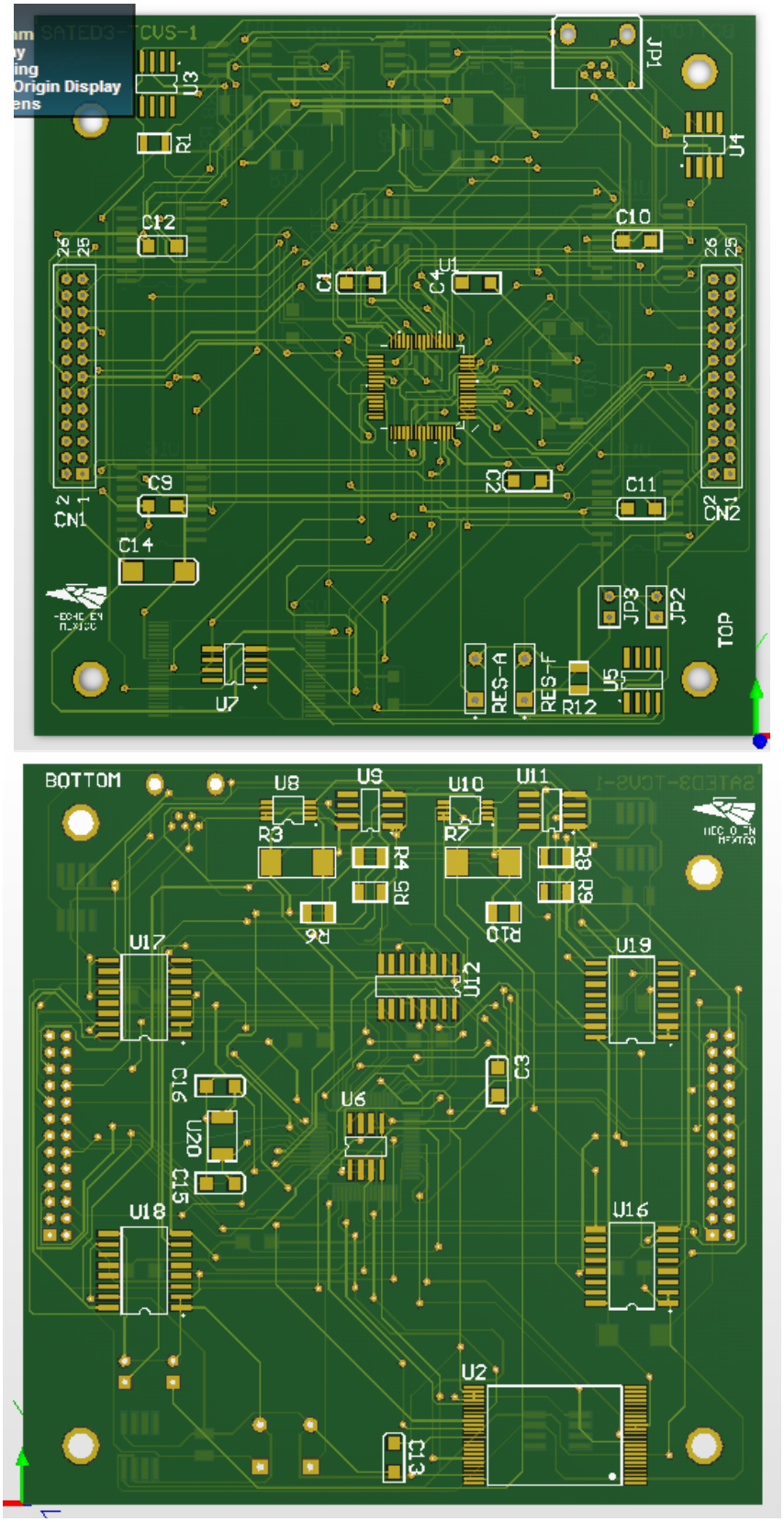


Figura 6.18 Vista por adelante y por atrás de la tarjeta final de estabilización desarrollada para el MCU Stellaris.

Capitulo 7 Implementación del Software básico de operación elaborado para las tarjetas de estabilización activa de SATEDU

7.1 Introducción

El MCU Stellaris necesita de cierta programación para ejecutar sus tareas asignadas. En el presente capítulo se habla sobre la programación que fue desarrollada para que el microcontrolador pueda realizar sus tareas. También se detallan las herramientas que fueron utilizadas para programar al MCU.

7.2 Plataforma de software

El software desarrollado en esta tesis para el MCU fue programado sobre la interfaz de desarrollo que también proporciona Texas Instruments para la programación de sus dispositivos, llamada *Code Composer Studio* (CCS), figura 7.1. Este software provee una interfaz gráfica para usar herramientas de generación de código y relacionar toda la información necesaria para construir un programa o una biblioteca. Cuando se construye un programa dentro del CCS, este invoca todas las herramientas correspondientes para la compilación, el ensamblado, el ligado y además se asocian las bibliotecas de código correspondientes usadas en el proyecto.

El CCS también cuenta con una serie de herramientas que ayudan a la depuración del código como los puntos de interrupción de programa (breakpoints), un visor de variables (watch), visor de memoria tipo dato y memoria tipo programa, visor gráfico de la memoria dato para la graficación de una variable, además de que es posible visualizar lenguaje mixto entre ensamblador y lenguaje C, etc.

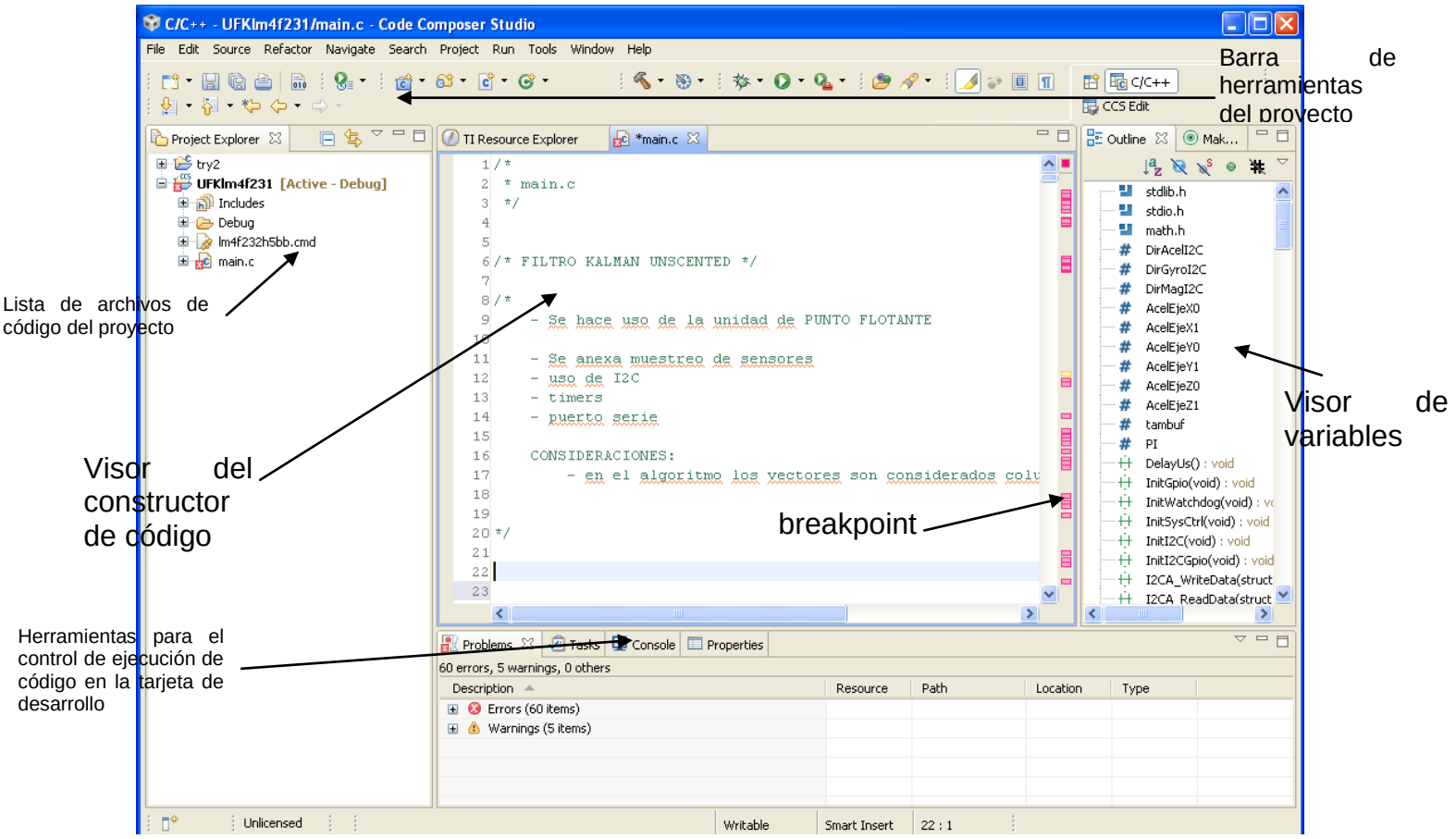


Figura 7.1 Interfaz del Code Composer Studio (CCS).

Dentro del ambiente CCS es posible realizar la programación tanto en lenguaje ensamblador como en lenguaje C o bien mezclar ambos lenguajes dentro del mismo proyecto. El compilador CCS utilizado fue la versión 5 que incluye el mismo fabricante, una de las grandes ventajas de este sistema es que es posible depurar el código al mismo tiempo que se prueba en la tarjeta, lo cual permite una rápida identificación de problemas y errores de código, [39].

7.3 Bootloader

La técnica más usada hoy en día para realizar una reconfiguración en software de un sistema embebido, ya ssea para corregir errores de programación o extender sus prestaciones.
Para hacer reconfiguración en campo se siguen los siguientes pasos:

1. El fabricante diseña un dispositivo con una versión inicial de *firmware*
2. El dispositivo es vendido al cliente
3. El fabricante desarrolla una nueva versión de *firmware*
4. La nueva versión es distribuida al cliente
5. El cliente actualiza su dispositivo con la nueva versión del *firmware*

Si no se toman ciertas medidas, este proceso puede resultar muy peligroso y dejar el dispositivo inservible. Algunas situaciones comunes que se generan durante el proceso de reconfiguración y que pueden generar un error son la perdida de energía o de la conexión durante la transmisión del nuevo *firmware*. Este problema es llamado problema de seguridad del dispositivo *(safety of the device*), otro problema de seguridad suele ser, donde la integridad del *firmware* es importante por lo cual puede agregarse código de detección de errores pero también de codificación para que no pueda ser usado por un competidor.

Desarrollar un *firmware* para un microcontrolador se hace con las herramientas necesarias y un banco de pruebas. Entre el conjunto de estas herramientas pueden enumerarse puertos de depuración como JTAG, depuradores en circuito, dispositivos programadores entre otros.

Sin embargo, estas herramientas no suelen ser usadas para actualizar el *firmware* de un dispositivo por el usuario final, ya que pueden presentarse varios escenarios bastante obvios que incluyen desde el no contar con estas herramientas hasta la falta de experiencia técnica por parte del usuario. Por esta razón, la actualización en campo suele realizarse mediante otros mecanismos.

En resumen el *bootloader* es un pequeño código que se programa al principio de la *flash* para actuar como un lanzador de aplicaciones así como mecanismo de actualización de algún firmware que este corriendo en el MCU Stellaris, utilizando ya sea UART, I2C, SSI, Ethernet o USB. El proceso de actualización usando un *bootloader* queda resumido como sigue:

1. El fabricante libera una nueva versión del *firmware*
2. La nueva versión del *firmware* es distribuida a los usuarios
3. La condición de disparo es activada por el usuario
4. El *bootloader* se conecta con el anfitrión
5. El anfitrión enviar el nuevo *firmware*
6. La aplicación existente es reemplazada
7. La nueva aplicación es ejecutada

Una vez que la transferencia termina, el *bootloader* ha reemplazado la versión antigua del *firmware* por la nueva versión, por lo que la nueva aplicación ha sido cargada. La figura 7.2 muestra el proceso descrito, [60].

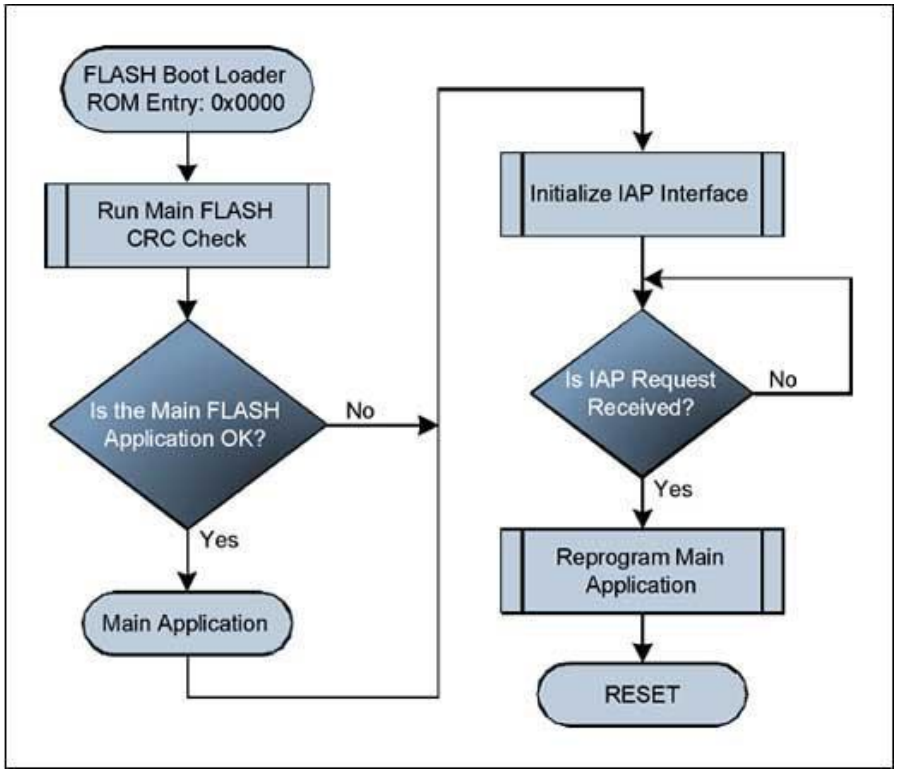


Figura 7.2 Diagrama operativo del *bootloader.*

7.4 Generación de PWM

Como se ha explicado anteriormente para que se muevan las ruedas se utiliza la tarjeta de *drivers* y potencia, pero la señal que los mueve proviene del MCU Stellaris, el cual proporciona las salidas de PWM y además de dos señales para dar el sentido de giro, así como el frenado.

Para esto se debe inicializar el modulo PWM, y generar las salidas del sentido de giro. El código de generación de PWM se puede ver en el apéndice.

7.5 Filtro Square Root - Kalman Unscented (SR-UKF)

En esta sección se presenta el diagrama de flujo que se utilizó para aplicar el UKF de manera recursiva en el MCU Stellaris, este diagrama se muestra de manera general en la figura 7.4. En el método SR-UKF implementado en el MCU Stellaris de esta tesis también se anexó un método determinístico para realizar el proceso de estimación de la orientación.

El procedimiento del SR-UKF es similar al EKF solo que se anexa el uso de la transformada *Unscented* para obtener los puntos sigma y para realizar el proceso de filtrado.

Inicialización de variables: previo a empezar las iteraciones es necesario tener una serie de condiciones iníciales para el algoritmo y también se calculan de manera previa algunas constantes que no es necesario calcular con cada iteración.

Método TRIAD: El método TRIAD es un método determinístico empleado en la estimación de orientación, y es útil siempre y cuando se tengan dos vectores referencia y dos vectores de medición. Para este caso, los vectores de medición se adquieren a través de dos sensores, el acelerómetro y el magnetómetro, a los que es posible definir un vector de referencia. En cuanto al magnetómetro, se tiene el problema de que el vector de Norte magnético no es fijo en toda la superficie de la Tierra lo que hace que el vector de referencia también esté cambiando, por lo cual sería necesario emplear otro método para renovar continuamente este vector de referencia
En el caso del vector de referencia usado para el acelerómetro corresponde al vector de gravedad, y en el caso del magnetómetro al Norte magnético de la Tierra. Cabe señalar que esto será solo aplicado para la mesa suspendida en aire ya que en el caso de un satélite real no es posible usar acelerómetro debido a que el vector de referencia de gravedad es más débil y debido a que las aceleraciones que se presentaran se confundirían con el vector disminuido de la gravedad, para ello sería necesario emplear otro tipo de sensor como puede ser el sensor de sol o el sensor de estrellas.
.
El método TRIAD se usa en el SR-UKF como parte del medio de medición del sistema. A través de este método es posible obtener el cuaternión de medición que será usado en la parte de innovación del filtro.

Para mayor información sobre el código ver la referencia [39]. La implementación del código en el MCU Stellaris se puede ver dentro del apéndice.

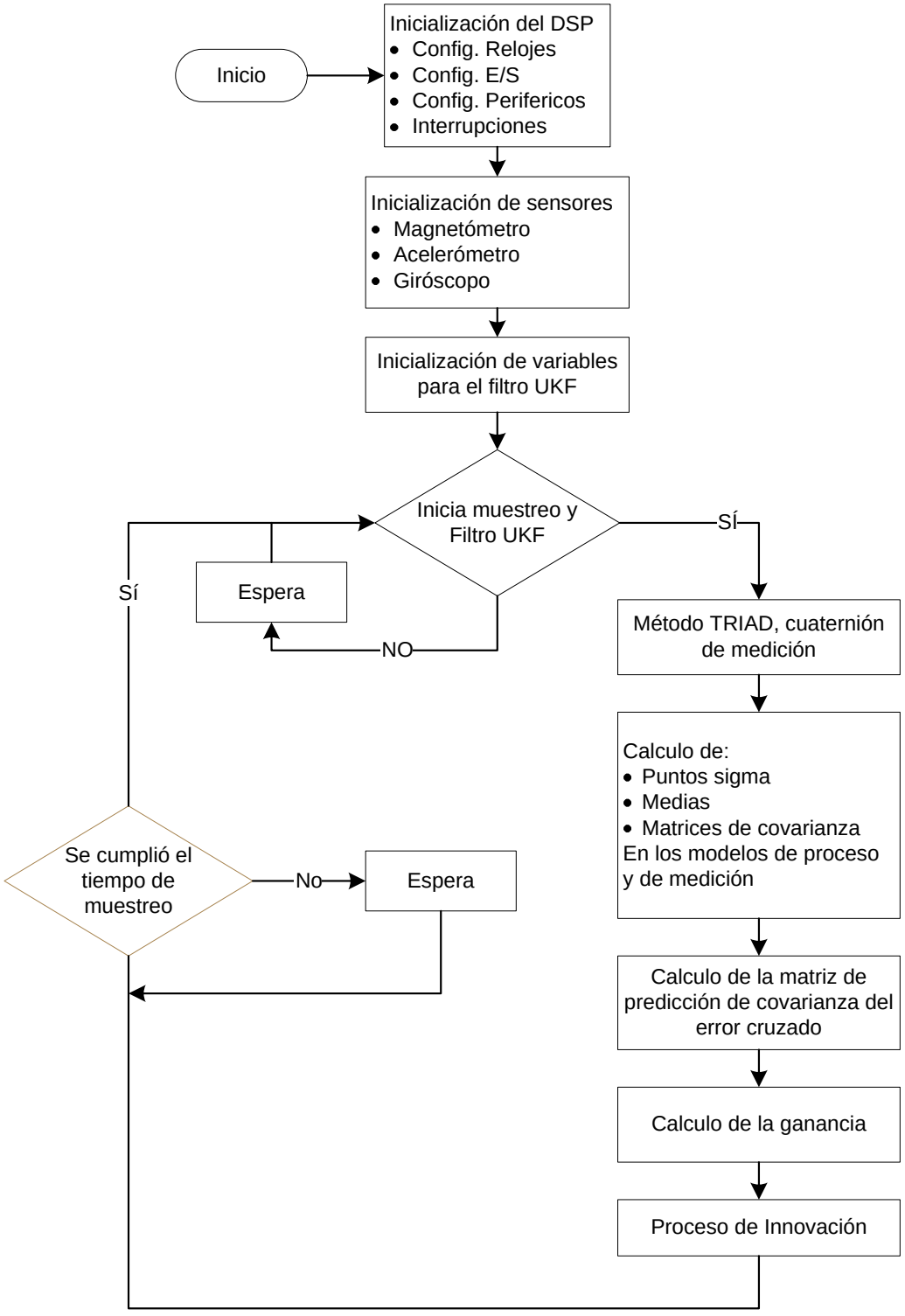


Figura 7.4 Diagrama de flujo del filtro de Kalman Uncented.

Capítulo 8. Pruebas de validación realizadas a las tarjetas de estabilización activa con el sistema SATEDU

8.1 Introducción

El presente capítulo tiene por objetivo mostrar y resumir los resultados obtenidos en este proyecto, los cuales abarcan desde las pruebas de validación hechas en las tarjetas hasta los resultados de la implementación de algoritmos en sistema embebido del MCU Stellaris.

8.2 Pruebas de validación desarrolladas y resultados obtenidos con las tarjetas de estabilización activa en tres ejes de SATEDU

La tarjeta de Reguladores y Drivers fue montada en la mesa suspendida en aire MSA, y validada con un FPGA, corriendo algoritmos de estabilización, probando su funcionamiento para mover las ruedas inerciales colocadas sobre la mesa.

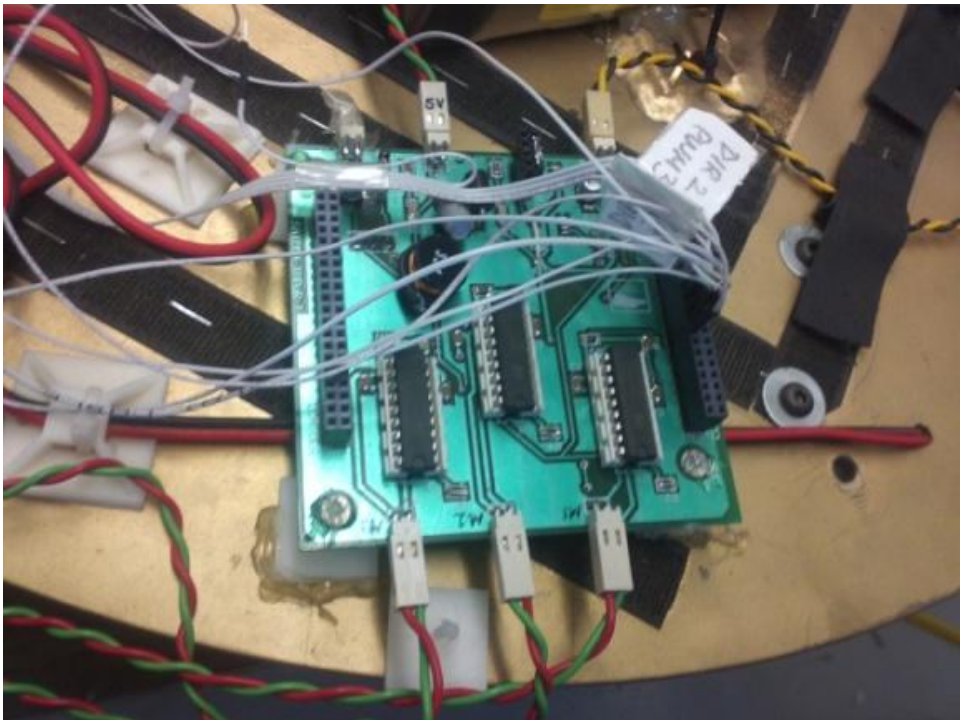


Figura 8.1 Tarjeta de drivers elaborada en esta tesis para Ruedas Inerciales sobre la MSA.

Figura 8.2 Tarjeta de drivers elaborada en esta tesis para Ruedas Inerciales sobre la MSA.

- Se efectuó el diseño del prototipo de la tarjeta de estabilización activa basada en MCU Stellaris para el satélite educativo SATEDU cumpliendo los objetivos planteados al inicio de su desarrollo. Se obtuvieron resultados en el ahorro de energía, ampliación de los recursos de hardware y en compatibilidad con los subsistemas de la plataforma satelital SATEDU.
- Se sustituyó el microcontrolador principal de la plataforma satelital SATEDU SAB80C166 por el microcontrolador LM4F231 Stellaris de Texas Instruments, basado en un núcleo Cortex 4 de 32 bits de ARM, con juego de instrucciones RISC, arquitectura Harvard y una alimentación de 3.3V para sus periféricos y de 1.8V para su núcleo.
- Se mantiene compatibilidad en el canal de comunicación de la plataforma SATEDU vía RS-232,
- Se diseñó un sistema de reconfiguración remota basado en un microcontrolador PIC18LF2550 para realizar el cargado de nuevo *firmware* a través de un *bootloader* compatible con el MCU Stellaris.
- Se implementó software para leer los datos de la tarjeta de sensores desde el MCU.
- Se modifico el código de SR-UKF para se utilizado en el MCU.
- Se implemento un segundo bus de comunicaciones para la adquisición de datos de sensores utilizando el protocolo TWI (I2C).
- Se genero código para el manejo de las ruedas inerciales a través de salidas PWM.
- Se realizaron pruebas experimentales de maniobras con ruedas inerciales sobre la mesa suspendida en aire.

Capitulo 9. Conclusiones del trabajo desarrollado

9.1 Introducción

En este capítulo se exponen las conclusiones y recomendaciones mas importantes que emanan del presente trabajo de tesis de maestria.

9.2 Conclusiones

Durante el desarrollo de esta tesis se lograron validar las tarjetas que componen el sistema de estabilización por medio de ruedas inerciales y BTMs, las cuales servirán como referencia y punto de partida para proyectos satelitales inmediatos que sigan esta línea de investigación.

El diseño propuesto para la tarjeta de estabilización MCU Stellaris para el satélite SATEDU cumple preliminarmente con los objetivos marcados al principio del proyecto.

Con este nuevo diseño se puede sustituir a la computadora de vuelo antecesora y se plantea una base para una nueva plataforma satelital basada en SATEDU que mejora las prestaciones ofrecidas por este último al contar con un nuevo bus de comunicaciones en red, un mejor núcleo para procesamiento, menor consumo de energía y mejores recursos de hardware.

De igual forma, se desarrolló y validó competamente la tarjeta de potencia para manejar hasta tres ruedas inerciales del sistema de estabilización activa que se ha desarrollado en el IINGEN-UNAM.

Respecto al satélite SATEDU, las tarjetas que componen el sistema de estabilización activa harán posible como producto adicional a este trabajo de tesis, formar recursos humanos especializados en este campo, gracias a la experiencia y conocimientos adquiridos durante su desarrollo.

Se implementó el filtro SR-UKF, un método orientado a resolver el problema de estimación de la orientación en el satélite SATEDU como parte del problema del control de orientación en su primera aproximación para ser validado en la mesa suspendida en aire MSA.

9.3 Recomendaciones y trabajo futuro

Algunas recomendaciones para la mejora de la tarjeta de estabilización activa MCU Stellaris son las siguientes.

En caso de utilizar sistemas operativos, para el desarrollo de aplicaciones más complejas, es recomendable integrar un sistema operativo de tiempo real (RTOS) por su relativa sencillez en comparación con otros. Existen en el mercado sistemas operativos de tiempo real que incluyen el manejo de protocolos de red, sistemas de archivos y herramientas que administran de forma eficiente los recursos de la unidad de procesamiento. Texas Instruments cuenta y proporciona algunos de estos sistemas en tiempo real.

Posteriormente, se deberá integrar a la implementación los algoritmos de control de orientación necesarios para este propósito. Actualmente ya también se tiene una primera propuesta en proceso de implantación en hardware usando técnicas de *hardware-in-the-loop* en el Instituto de Ingeniería, el cual será validado en la mesa suspendida en aire

Referencias

[1] Historia, Pagina de Satmex, http://www.satmex.com.mx/, consultada diciembre 2012.

[2] Chris Bergin, ILS Proton-M makes successful return with Satmex 8
ILS Proton-M makes successful return with Satmex 8, NASASpaceflight, 2013, http://www.nasaspaceflight.com/2013/03/ils-proton-m-launch-satmex-8/

[3] Juan Carlos Plata , Tendrá México satélite que ayudaría a predecir terremotos , Boletin UV, 2005, http://www.uv.mx/boletines/banner/vertical/noviembre05/141105/satelite.htm

[4] Acerca de la UNAM, historia 1990,
http://www.unam.mx/acercaunam/es/unam_tiempo/unam/1990.html

[5] Pacheco, E.; Conte, R.; Mendieta, F.J.; Muraoka, R.; Medina, J.L.; Arvizu, A. The satex project: a mexican effort. The development of a micro-satellite platform for space technologies knowledge and human resources preparation, Aerospace Conference, 2003. Proceedings. 2003 IEEE, CICESE Research Center. DOI: 10.1109/AERO.2003.1235477.

[6] Vivas, E. et al. “Steps towards the development of a portable and cost-effective system for human resources training in small satellite technology”, Magazine: Research On Computer Science, Control, Virtual Instrumentation and Digital Systems, Vol. 24, pp 117-113, ISSM: 1870-4069, Centro de Investigacion en Computacion, IPN, Mexico DF, Noviembre 2006.

[7] Ortiz, H. “Implantación de técnicas de tolerancia a fallas, ensamble y validación operativa de la computadora de vuelo del microsatelite experimental SATEX1”. Tesis de Licenciatura, Facultad de Ingeniería, UNAM, México, 2003

[8] Página del SATEDU http://proyectos.iingen.unam.mx/satedu/publicaciones.html

[9] Jiménez Madrigal, Emilio Augusto; Subsistemas de estabilización activa y sensores para un simulador satelital, Tesis de licenciatura, Facultad de Ingeniería, UNAM, México, 2009.

[10] Colorado Satellite Service, EEUU. “EyasSat The Classroom Satellite”, http://www.eyassat.com, consultada en junio 2011.

[11] Diseño y Fabricación De Un Pico-Satélite Artificial Mediante Las Normas CanSat, Israel Zúñiga de la Mora, Facultad de Telemática, Universidad de Colima tesis de licenciatura, junio 2011

[12] Proyectos de Cubesats.
http://en.wikipedia.org/wiki/CubeSat#Current_%20running_projects, consultada Junio 2011

[13] Pagina de XatCobeo http://www.xatcobeo.com/cms/index.php consultada en Agosto 2012

[14] Jerry-Sellers, EyasSAT: A Classroom Nanosatellite for Teaching Space Systems Engineering, presentación Centro de Tecnologia e Informacion, Universidad de Tokio, http://park.itc.u-tokyo.ac.jp/nsat/NS1/files/10th.PM/Session_4/Presentation_Jerry-Sellers.pdf, 2010.

[15] EyasSat User's Manual (C4.1), David J. Barnhart, U. S. Air Force Academy, Colorado, USA, 10 June 2008

[16] Senstat, http://mysensat.org, consultada en Diciembre, 2012.

[17] F.J. Mendieta, E. Pacheco, A. Serrano, "Viculacion academica industria en tecnología satelital: el proyecto Sensat en la región Noroeste", CICESE, Ensenada, B.C., Presentación para el foro de la AEM, diciembre 2010.

[18] Dr. Esaú Vicente-Vivas, E. A. Jiménez, Z. Carrizales , C.A. Sánchez, J.R. Córdova, R. Alva, M.A.García and G. Islas"Successful Development of a Portable Didactic Satellite for Training andResearch in Satellite Technology", CORE-2009, 10th Computing Congress, CIC-IPN, Mexico City, May, 27-29, 2009.

[19] Espen Oland and Rune Schlanbusch, "Reaction Wheel Design for CubeSats"
Department of Scientific Computing, Electrical Engineering and Space Technology
Narvik University College, Narvik, Norway

[20] Kayal, H., Baumann, F., and Briess, K., "BEESAT - A Pico Satellite
of TU Berlin for the In-Orbit Verification of Miniaturised Wheels," International Astronautical Conference, 2008.

[21] Prado, J. Sistema de simulación para pruebas de algoritmos de orientación y control de satélites pequeños. Tesis de doctorado UNAM, México, 2008.

[22] Fortescue, P., Stark, J. y Swinerd, G. Spacecraft Systems Engineering. 3a edición. Ed. Wiley and Sons. EEUU, 2003.

[23] Paluszek, M., et al. Spacecraft Attitude and Orbit Control. 2a edición. Princeton. EEUU 2009.

[24] Grewal, M. How Good is your Gyro?. Revista IEEE Control Systems Magazine. Febrero, 2010.

[25] Michael's List of Cubesat Satellite Missions. Sitio Web, [http://mtech.dk/thomsen/space/cubesat.php]

[26] E.A. Jimenez, Filtrado Digital de señales de navegación inercial para el satélite humsat, tesis de maestría, UNAM, 2012

[27] Howley, B. “Overview of Spacecraft Attitude Determination and Control”. Articulo de Lockheed Martin Space Systems Company.

[28] Tohami, S. “Sensor Modeling, attitude determination and control for micro-satellite”. Tesis de Maestria. Universidad Noruega de la Ciencia y tecnologia, 2005.

[29] Contreras Carrasco Fabiola, Pruebas de Control de Estabilización para Satélite Pequeño empleando bobinas magnéticas y ruedas inerciales, 2004, Tesis de licenciatura.

[30] Lappas, V. “Control Techniques for Aerospace Systems”. Surrey Space Centre METU Aerospace Seminar. Artículo del seminario realizado el 8 de Mayo de 2003.

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

[32] Rafael Molina Soriano. Bases Del Filtro de Kalman. http://decsai.ugr.es/vip/doctorado/pvd/T7bn.pdf

[33] Inelmatic- El filtro de Kalman http://www.inelmatic.com/web/files/downloads/filtrodekalman.pdf

[34] Tema XVII – El filtro de Kalman http://www.depeca.uah.es/wwwnueva/docencia/ ING-CA/ctr_avz/VVEE17.PDF

[35] Greg Welch, Gary Bishop. Introduction to the Kalman Filter.

[36] Peter S. Haybeck. Stochastic models, estimation and control. Volume 1

[37] Greg Welch, Gary Bishop. SCAAT. Incremental tracking with Incomplete Information http://knaan.com/Lev/SIGGRAPH2001_CoursePack_08.pdf

[38] Greg Welch, Gary Bishop, Department of Computer Science at theUniversity of North Carolina at Chapel Hill, The Kalman Filter http://www.cs.unc.edu/~welch/kalman/

[39] E.A. Jimenez, Filtrado Digital de señales de navegación inercial para el satélite humsat, tesis de maestria, UNAM, 2012

[40] Kalman Filter Derivation. http://www.aticourses.com/kalman_filter.pdf

[41] Astolfi, A., Karagiannis, D. y Ortega, R. Nonlinear and Adaptive Control with Applications. Ed. Springer. EEUU, 2008.

[42] Brown, H. G. *Introduction to Random Signals Analysis and Kalman Filtering.* Ed. Wiley & Sons. EEUU, 1983.

[43] Estrada, I. Control basado en pasividad y estimación de la orientación de un robot móvil omnidireccional. Tesis de maestría Centro de Investigaciones y Estudios Avanzados del IPN, México, 2009.

[44] Haykin, S. Kalman Filtering and Neural Networks. Ed. Wiley & Sons. EEUU, 2001.

[45] Julier, S. J. y Uhlmann, J. K. Unscented Filtering and Nonlinear Estimation. En Proceedings of the IEEE, Vol. 92, No. 3, pp 401-422, EEUU, Noviembre 2004.

[46] Julier, S. J. y Uhlmann, J. K. A new extension of the Kalman Filter to Nonlinear Systems. En Proceedings of Aerosense: The 11th Int. Symp. on Aerospace/Defense Sensing, Simulation and Controls, EEUU, 1997.

[47] Rowlett, P. The unplanned impact of mathematics. Revista Nature, Vol. 475, pp 166-169.

[48] Tewari, A. Atmospheric and Space Flight Dynamics Modeling and simulation with MATLAB and Simulink. Ed. Birkhäuser. EEUU, 2007.

[49] Van der Merwe, R. y Wan, E. A. The Square-root Unscented Kalman Filter for State and parameter-estimation. En Proceedings of the 2001 International Conference in Acoustics, Speech and Signal Processing, Vol. VI, pp 3461-3464 ,Salt Lake City, EEUU, Mayo 2001.

[50] Van Verth, J. y Bishop, L. Essential Mathematics for Games and Interactive Applications. 2a edición. Ed. Morgan Kaufmann. EEUU, 2008.

[51] Yun, X., et al. An Improved Quaternion-Based Kalman Filter for Real-Time Tracking of Rigid Body Orientation. En Proceedings of the 2003 IEEE International Conference on Intelligent Robots and Systems, pp 1074-1078, Las Vegas, EEUU, Octubre 2003.

[52] Domínguez Mendoza Paul, Procesamiento de señales de sensores de navegación para satélites sobre una plataforma FPGA, tesis de maestría, 2011

[53] Hoja de aplicación del L602, Microchip Technology, EEUU, 2004.

[54] AN1593, hoja de aplicación de Regulador LM2575, Motorola, EEUU, 1997.

[55] Stellaris LM4F231H5Qr, hoja de datos tecnica de LM4F2315QHR, Texas Instruments, 2012.

[56] García Illescas, Miguel, Actualización De Una Computadora De Vuelo Para Un Satélite Educativo, 2009

[57] Maxim MAX4071, hoja de datos técnica, EEUU, 2003.

[58] Comparador LM6511, hoja de datos técnica, Texas Instruments, EEUU, 1999.

[59] Altium Board-level design system. “Introducing Protel DXP Tutorial”. Hoja de tutorial, 2002. [www.altium.com]

[60] Nucamendi, Ignacio, Desarrollo de computadora de Vuelo y software terrestre de Operaciones para el satélite Humsat, tesis de maestría, 2011

Glosario

3D: Tercera dimension.

A: Amperes.

AAUSAT: Aalborg University Satellite, Denmark.

ABS: Air Bearing System.

ADC: *Analog-to-Digital Converter, Convertidor Analógico-Digital.*

EEA: Agencia Espacial Europea.

ARM: Arquitectura ARM, una familia de microprocesadores producidos por la empresa ARM Holdings.

BEESAT: Berlin Experimental and Educational Satellite.

BSDL: Boundary Scan Description Language, lenguaje de escaneo de límites.

BTM: Bobinas de Torque Magnético.

CanSat: Satélite del tamaño de una lata de refresco.

CCS: Code Composer Studio.

CD: Corriente Directa.

CMG: Control Moment Gyro, Giroscopio de control de momentos.

Cubesat: Satélite de 10x10 cm con forma de cubo.

CV: Computadora de Vuelo. Véase FC.

DMIPS: Decenas de Millones de Instrucciones Por Segundo.

DSP: Digital Signal Procesor, procesador digital de señales.

EEPROM: Electrically Erasable Programmable Read-Only Memory, ROM programable y borrada eléctricamente.

EKF: Extended Kalman Filter.

EUA: Estados Unidos de America.

Eyassat: Satellite Eyas, satelite halcón.
FC: Flight Computer.

FKU: Filtro de Kalman Unscented. Véase UFC.

FKE: Filtro de Kalman Extendido vease EKF.

FPGA: *Field Programmable Gate Array, arreglo de compuertas lógicas programables en campo.*

FSS: Fixed Satellite Services, servicios de satélite fijo.

GDSA: Grupo de Desarrollo de Sistemas Aéreoespaciales del Instituto de Ingeniería.

GNSS: Global Navigation Satellite Systems, satelites de los sistemas de navegacion global.

GPIO: General Purpose Input/Output, entradas y salidas de proposito general.

GPS: *Global Positioning System,* sistema de posicionamiento global

HDTV: High Definition Television, televisión de alta definición.

Hz: Hertz

I²C: Inter-Integrated Circuit, Inter-Circuitos Integrados, un bus de comunicaciones en serie.

IINGEN-UNAM: Instituto de Ingeniería de la UNAM.

IMU – Inertial Measurement Unit

INS - Inertial Navigation System

INTA: Instituto Nacional de Técnica Aeroespacial.

JTAG: Joint Test Action Group, se utiliza para testear PCBs utilizando también como mecanismo para depuración de aplicaciones empotradas, puesto que provee una puerta trasera hacia dentro del sistema.

KB: KiloBytes:

LED: Light-Emitting Diode, diodo emisor de luz.

LiPo*: Litio*-ion *Polímero*, ion *de litio* en *polímero.*

LQFP: Low-profile Quad Flat Package, encapsulado cuadrado plano de perfil bajo.

LSB: *Least Significant Bit,* el bit menos significativo.

M: metros.

MARG: Magnetic, Angular Rate and Gravity, magnetico, razon angular, y gravedad.

MCU: microcontrolador.

MEMS: Micro Electro Mechanical Systems, micro sistemas electromecánicos.

MPU: Memory Protection Unit, unidad de protección a la memoria.

MSA: Mesa Suspendida en Aire. Véase ABS.

PACSAT: Manufacturador de antenas en Australia.

PC: Personal Computer, computadora personal.

PCB: *Printed Circuit Board*, circuito impreso.

PWM: *pulse-width modulation*, modulación por ancho de pulsos.

Q: transistores.

RDS: ionizing radiation system, radiación ionizante *(*o

SRAD: Software Defined Reconfigurable Radio, radio reconfigurable definido por software.

RF: radio frecuencia.

RIs: Ruedas Inerciales.

RISC*: Reduced Instruction Set Computer*, Computadora con Conjunto de Instrucciones Reducidas.

ROM: Read Only Memory, memoria de solo lectura.

RPM: Revoluciones Por Minuto.

SATEDU: Satélite Educativo. Educative Satellite Platform

Satex: Satelite Mexicano.

SC: Subsistema de Comunicaciones.

SCT: Secretaria de Comunicaciones y Transportes.

SE: Subsistema de estabilización.

Sensat: Self-explore Nano Satellite.

CISESE: *Centro de Investigación Científica y de Educación Superior de Ensenada*, Baja California.

SP: Subsistema de Potencia.

SRAM: Static Random Access Memory, Memoria Estática de Acceso Aleatorio, tipo de memoria basada en semiconductores que a diferencia de la memoria DRAM, es capaz de mantener los datos, mientras esté alimentada, sin necesidad de circuito de refresco.

SR-UKF: Filtro Square-Root Unscented Kalman Filter.

SSE: Subsistema de sensores de estabilización.

SSI/SPI: Serial Peripheral Interface Bus, puerto serie que opera en modo full duplex. SSI, Synchronous Serial Interface, interface síncrona serial.

Telecomm: Telecomunicaciones de México.

TI: Texas Instruments.

TRIAD: Triada, el método TRIAD es un método determinístico empleado en la estimación de orientación, y es útil siempre y cuando se tengan dos vectores referencia y dos vectores de medición.

UART: *Universal Asynchronous Receiver-Transmitter,* Transmisor-Receptor Asíncrono Universal.

UFC: Unscented Kalman Filter.

UNAM: Universidad Autónoma de México.

UNAMSAT: Satelite de la UNAM.

USB: *Universal Serial Bus,* bus universal en serie.

USD: United States Dollar, dolar estadounidense.

V: Voltios.

VHDL: Representa la combinación de VHSIC y HDL, donde VHSIC es el acrónimo de Very High Speed Integrated Circuit y HDL es a su vez el acrónimo de Hardware Description Language.

Apendice A

I^2C

I^2C es un protocolo de comunicación serie diseñado por Philips que se utiliza esencialmente entre dispositivos que pertenecen al mismo circuito, por ejemplo, sensores con un microcontrolador.

TWI
Aunque las patentes de I^2C ya han expirado, algunos vendedores utilizan los nombres TWI y TWSI para referirse a I^2C. Es exactamente lo mismo.

Características del protocolo:

- Velocidad standard de **100Kbit/s** (100kbaudios). Se puede cambiar al modo de alta velocidad (400Kbit/s)
- Configuración maestro/esclavo. La direccion del esclavo se configura con software
- Solo se necesitan **dos** lineas:
  - **SDA (Serial Data Line):** Linea de datos.
  - **SCL/CLK (Serial Clock Line):** Linea de reloj, será el que marque el tiempo de RW (Lectura/Escritura)
  - *Nota: Suponemos que todos los dispositivos tienen masa común, si no fuera así hay que uncluir una linea de masa.*
- Los comunicación siempre tiene la estructura siguiente:
  - Transmisor: Byte de datos (8 Bits)
  - Receptor: Bit llamado ACK de confirmación.

¿Cómo se realizan las conexiones?
SDA y SCL van a su pin correspondiente en cada dispositivo, de manera que todos quedan en paralelo.

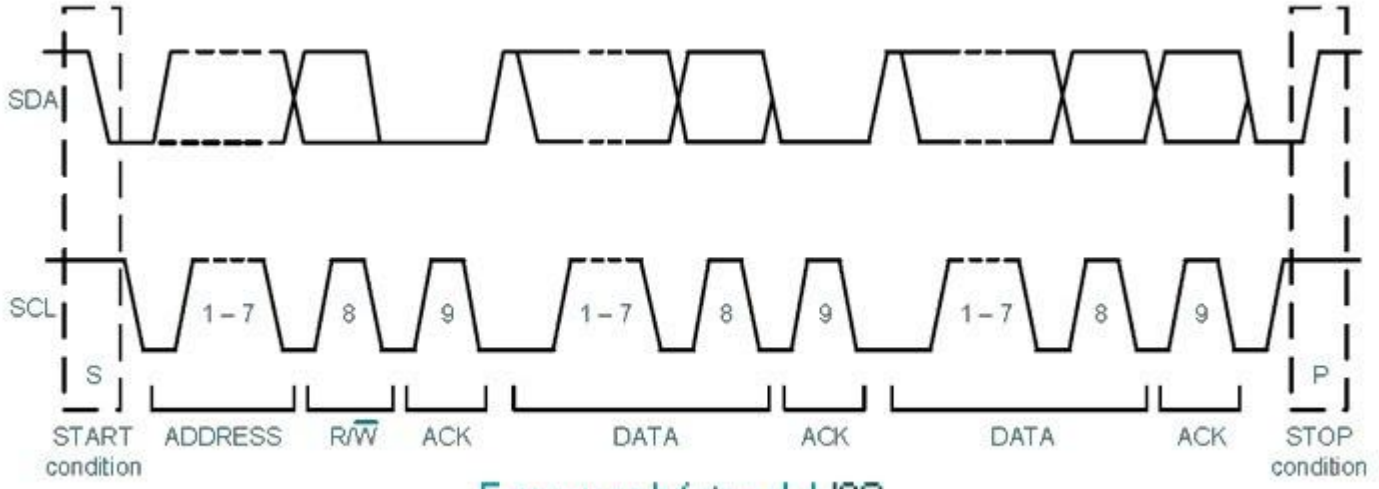
Las lineas SDA y SCL estan independientemente conectadas a dos resistores Pull-Up que se encargaran de que el valor logico siempre sea alto a no ser que un dispositivo lo ponga a valor lógico bajo.

¿Qué tipo de comunicación es?
Es una comunicación de tipo **half duplex**. Comunicación bidireccional por la misma linea pero no simultáneamente bidireccional.

¿Cual es la estructura de la comunicación?
La estructura de la comunicación básica es la siguiente:

1. START condition *(Master)*
2. 7 Bits de direccion de esclavo *(Master)*
3. 1 Bit de RW, 0 es Leer y 1 Escribir. *(Master)*
4. 1 Bit de Acknowledge *(Slave)*
5. Byte de dirección de memoria *(Master)*
6. 1 Bit de Acknowledge *(Slave)*
7. Byte de datos *(Master/Slave (Escritura/Lectura))*
8. 1 Bit de Acknowledge *(Slave/Master (Escritura/Lectura))*
9. STOP condition *(Master)*

Esta es la base de la comunicación pero para leer o escribir, según el dispositivo con el que se comunica el Master la comunicación tendrá una estructura específica.



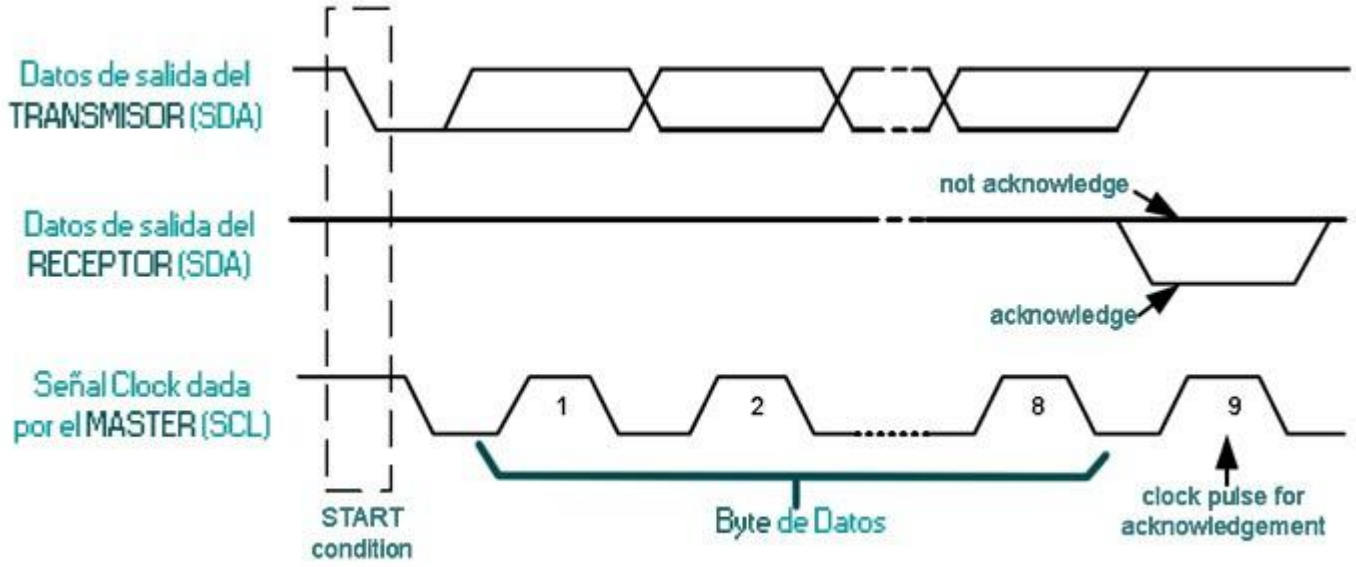
Estructura básica del I2C

¿Qué es el bit de Acknowledge (ACK)?

Este bit es una respuesta del receptor al transmisor. Es una parte básica de la comunicación y tiene diferentes sentidos según el contexto.

Se utiliza principalmente con dos propósitos:

1. Conocer si el transmisor ha sido escuchado
2. **Lecturas multidatos:**
   Cuando se esta leyendo de un esclavo, si el master realiza un ACK, es decir, que responde; el esclavo pasa al siguiente valor de registro y lo envía también, hasta recibir un NACK (Not Acknowledge), es decir, ninguna respuesta del master. Esto sirve para hacer múltiples lecturas. Por ejemplo, nuestro acelerómetro o giroscopio, que tienen valores de X, Y y Z.



ACKnowledge en protocolo I2C

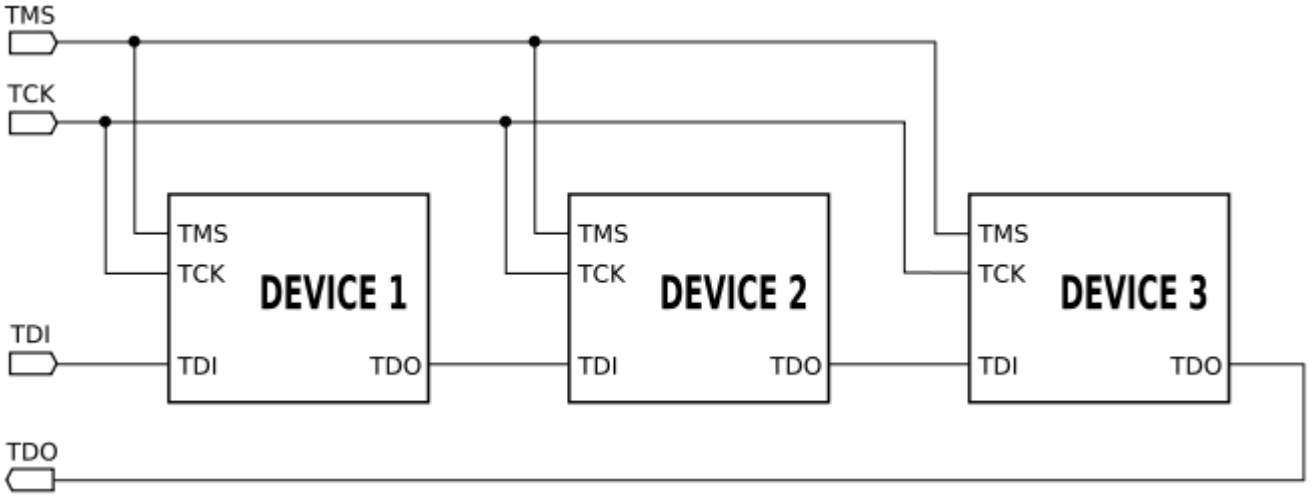
En esencia este es el funcionamiento del protocolo de comunicación con el que nos comunicaremos con múltiples sensores.

Apendice B

JTAG

Una interfaz JTAG es una interfaz especial de cuatro o cinco pines agregadas a un chip, diseñada de tal manera que varios chips en una tarjeta puedan tener sus líneas JTAG conectadas en daisy chain, de manera tal que una sonda de testeo JTAG necesita conectarse a un solo "puerto JTAG" para acceder a todos los chips en un circuito impreso. Los pines del conector son

1. TDI (Entrada de Datos de Testeo)
2. TDO (Salida de Datos de Testeo)
3. TCK (Reloj de Testeo)
4. TMS (Selector de Modo de Testeo)
5. TRST (Reset de Testeo) es opcional.



Ya que posee una sola línea de datos, el protocolo es necesariamente serial, como el Serial Peripheral Interface. La entrada de la señal de reloj es por el pin TCK. La configuración del dispositivo se realiza manipulando una máquina de estados de un bit empleando el pin TMS. Un bit de datos es cargado en TDI y otro sacado en TDO por cada pulso de reloj de la señal TCK. Se pueden cargar diferentes modo de instrucción como leer el ID del chip, muestrear el valor de pines de entrada/salida, manejar pines de salida, manipular funciones del chip, o funciones de bypass que unen el pin TDI con TDO para lógicamente unir cadenas de varios chips (chips en cascada). La frecuencia de trabajo de la señal de reloj del pin TCK varía en función de cada chip, pero típicamente está en el rango de 10-100 MHz (10-100ns/bit).

Cuando se hace la operación de boundary scan en circuitos integrados, las señales manipuladas están entre diferentes bloques funcionales del chip, más que entre diferentes chips.

El pin TRST es una señal opcional bajo-activa para reseteo o reinicio de la prueba lógica (por lo general asíncrona, pero que a veces está sincronizada con el reloj, dependiendo del chip). Si no se dispone de dicho pin, la prueba lógica puede reiniciarse mediante una instrucción reset.

Existen productos de consumo que tienen un puerto JTAG integrado, por lo que las conexiones están a menudo disponibles en la PCB como parte de la fase de prototipado del producto. Estas conexiones pueden proporcionar una sencilla forma de realizar ingeniería inversa.

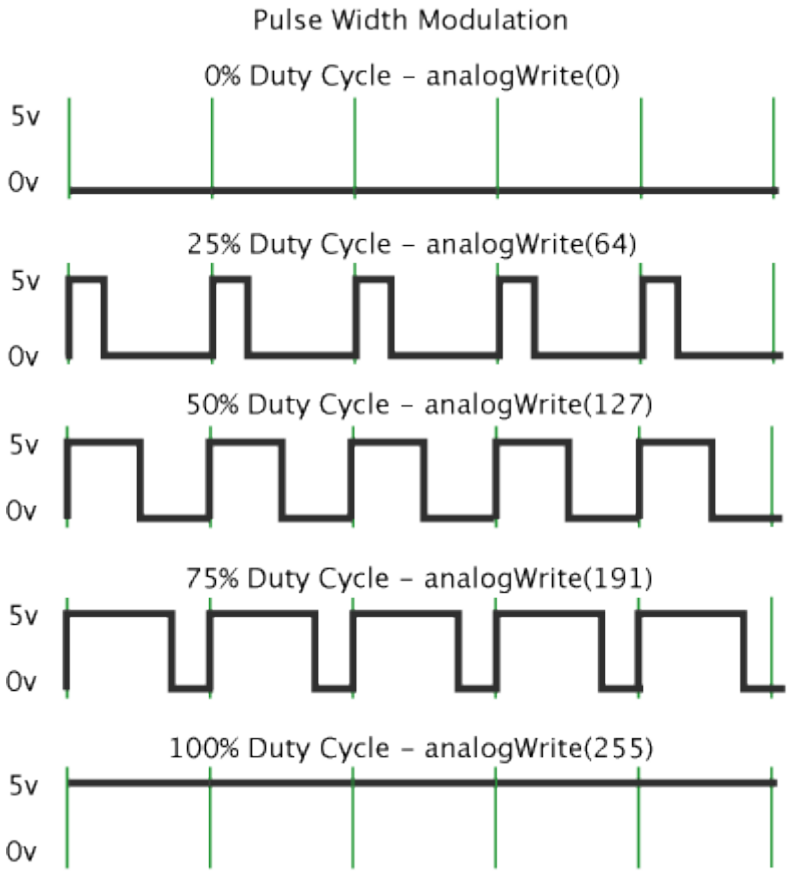
Apendice C

PWM

El ejemplo "Fading" demuestra el uso de una salida analógica (PWM) para atenuar y aumentar la luminosidad de un LED. Lo tienes disponible en la opción "File->Sketchbook->Examples->Analog" del menú del software de Arduino.

La Modulación por Ancho de Pulso (PWM = Pulse Width Modulation) es una tecnica para simular una salida analógica con una salida digital. El control digital se usa para crear una onda cuadrada, una señal que conmuta constantemente entre encendido y apagado. Este patron de encendido-apagado puede simular voltajes entre 0 (siempre apagado) y 5 voltios (siempre encendido) simplemente variando la proporción de tiempo entre encendido y apagado. A la duración del tiempo de encendido (ON) se le llama Ancho de Pulso (pulse width). Para variar el valor analógico cambiamos, o modulamos, ese ancho de pulso. Si repetimos este patrón de encendido-apagado lo suficientemente rapido por ejemplo con un LED el resultado es como si la señal variara entre 0 y 5 voltios controlando el brillo del LED.

En el grafico de abajo las lineas verdes representan un periodo regular. Esta duración o periodo es la inversa de la frecuencia del PWM. En otras palabras, con la Arduino la frecuencia PWM es bastante proxima a 500Hz lo que equivale a periodos de 2 milisegundos cada uno. La llamada a la función analogWrite() debe ser en la escala desde 0 a 255, siendo 255 el 100% de ciclo (siempre encendido), el valor 127 será el 50% del ciclo (la mitad del tiempo encendido), etc.



Una vez cargado y ejecutado el ejemplo mueve la arduino de un lado a otro, lo que ves es esencialmente un mapeado del tiempo a lo largo del espacio. A nuestros ojos el movimiento difumina cada parpadeo del LED en una linea. A medida que la luminosidad del LED se incrementa o atenua esas pequeñas lineas crecen o se reducen. Ahora estas viendo el ancho de pulso (pulse width).

En los motores

La modulación por ancho de pulsos es una técnica utilizada para regular la velocidad de giro de los motores eléctricos de inducción o asíncronos. Mantiene el par motor constante y no supone un desaprovechamiento de la energía eléctrica. Se utiliza tanto en corriente continua como en alterna, como su nombre lo indica, al controlar: un momento alto (encendido o alimentado) y un momento bajo (apagado o desconectado), controlado normalmente por relevadores (baja frecuencia) o MOSFET o tiristores (alta frecuencia).

Otros sistemas para regular la velocidad modifican la tensión eléctrica, con lo que disminuye el par motor; o interponen una resistencia eléctrica, con lo que se pierde energía en forma de calor en esta resistencia.

Otra forma de regular el giro del motor es variando el tiempo entre pulsos de duración constante, lo que se llama modulación por frecuencia de pulsos.

En los motores de corriente alterna también se puede utilizar la variación de frecuencia.

La modulación por ancho de pulsos también se usa para controlar servomotores, los cuales modifican su posición de acuerdo al ancho del pulso enviado cada un cierto período que depende de cada servo motor. Esta información puede ser enviada utilizando un microprocesador como el Z80, o un microcontrolador (por ejemplo, un PIC 16F877A de la empresa Microchip).

Apendice D

PROGRAMAS EN C

```
/* FILTRO KALMAN UNSCENTED */

/*
        - Se hace uso de la unidad de PUNTO FLOTANTE

        - Se anexa muestreo de sensores
        - uso de I2C
        - timers
        - puerto serie

        CONSIDERACIONES:
                - en el algoritmo los vectores son considerados columnas de las matrices


*/


#include "LM4F231x_Device.h"
#include " LM4F231x_I2C_defines.h"          // Macros used for I2C examples.
#include "stdlib.h"
#include "stdio.h"

#include "math.h"


// #define          GLOBAL_Q                20              //  valor Q de manera local
// #include "IQmathLib.h"


/*************************************/
/*********** ETIQUETAS ***************/

// sensores: direcciones de I2C
#define  DirAcelI2C          0x53    /* direcciones de los dispositivos (7bits MSB) */
#define  DirGyroI2C                  0x68
#define  DirMagI2C           0x1E

        /* acelerometro */
#define  AcelEjeX0           0x32
#define  AcelEjeX1           0x33
#define  AcelEjeY0           0x34
#define  AcelEjeY1           0x35
#define  AcelEjeZ0           0x36
#define  AcelEjeZ1           0x37


/* datos */

#define  tambuf                  300             // tamaño del buffer de datos


#define  PI                  3.141592653589793


/************** FUNCIONES  ***************/

/*** Delay ***/
extern void DelayUs(Uint16);    /* Delay (en ensamblador) */
```

```
/*** INicializacion del Micro ***/
extern void InitGpio(void);                 /* inicializacion de I/O  generales */
extern void InitWatchdog(void);             /* funcion de inicializacion de Watchdog */
extern void InitSysCtrl(void);              /* inicia el CPU  */


/*** Puerto I2C ***/
extern void InitI2C(void);
extern void InitI2CGpio(void);
Uint16 I2CA_WriteData(struct I2CMSG2 *msg);
Uint16 I2CA_ReadData(struct I2CMSG2 *msg);
extern Uint16 LecturaI2C(Uint16 disp, Uint16 registro, Uint16 nbytes, Uint16 *data);    // lee datos del Acelerometro ADXL345
//extern void EscrituraAcel(Uint16 registro, Uint16 nbytes, Uint16 *dataw);             // escribe datos del Acelerometro ADXL345
extern void EscrituraI2C(Uint16 disp, Uint16 registro, Uint16 dataw);

void InitAcel(void);                // inicia el acelerometro
void InitGyro(void);   // inicia el gyro
void InitMag(void);                 // inicia el Magnetometro

void ResetI2C(void);


/*** Pueto serie ***/
extern void InitSciaGpio(void);             /* funciones para puerto serie */
extern void scia_init(void);
extern Uint16 BusyTXA(void);
extern Uint16 DataRdyA(void);
extern unsigned char ReadSciA(void);
extern void scia_xmit(int a);
extern Uint16 BusyTXA(void);


/***** Sensores conversion ******/
void convAcel(int16 ax, int16 ay, int16 az, float gs[3]);
void convGyro(int16 gx, int16 gy, int16 gz, float ws[3]);
void convMag(float mx, float my, float mz, float mags[3]);


/***** MATEMATICAS  *********/
float modulo( float vector[3]);
void crossp( float vec1[3], float vec2[3], float prod[3]);
void prodMat3x1(float vect[3], float vect_trans[3], float mat3x3[3][3]);
void divVector(float vector[3], float escalar, float result[3]);

void dcm2quat(float Mat[3][3], float qc[4]);// convierte de Matriz de rot a quaternion
void escXmatrix(float escalar, float Mat[7][7], float res[7][7]);       // escalar por matriz

        // para kalman
void puntosSigmaX(float vector[7], float mat[7][7], float resp[7][15]);          //obtiene los puntos sigma del vector de estado
void proceso(float sigma[7][15], float sigmaUT[7][15], Uint16 columna);  //modelo del proceso
void medicion(float sigma[7][15], float sigmaUT[7][15], Uint16 columna);  // modelo de medicion

        // mats para Kalman
float prod_intM21(float mat1[21][7], Uint16 col1, float mat2[21][7], Uint16 col2);
void qr_mR( float mat[21][7], float R[7][7] );                          // obtiene solo la matriz R
void multVects7x1MatrixEsc(float escalar, float vec[7], float mat[7][7]);       // mat = escalar * (vec * vec')
void multMatrixTT(float mat[7][7], float resp[7][7]);    //  S'*S (S -> matriz triangular superior)
void multMatVecDiag(float mat[7][15], float vec[7], float diag[15], float res[7][15]);      //
void matrixPxy(float mat1[7][15], float mat2[7][15], float vec[7], float res[7][7]); // mat1-> normal (7x15), mat2(transpuesta) -> (15x7), res(7x7), vec ->
media y

void obtenK(float X[7][7], float A[7][7], float B[7][7]);  // despeja la matriz de ganancias
void qr_new( float mat[7][7],float Q[7][7],float R[7][7] );
```

```
float prod_intM7(float mat1[7][7], Uint16 col1, float mat2[7][7], Uint16 col2);



// Descomposicion de Cholesky
void chol(float mat[7][7], float cholmat[7][7]);

        // borrado
void borraMatriz7x15(float mat[7][15]);


/*** INTERRUPCIONES  ***/
        /* rutina de interrupcion */
interrupt void Int_timer0(void);

        /* interrupciones */
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);         /* tabla de vectores de interrupcion */



/*****************************************/
/*****************************************/


/*************** VARIABLES ****************/

// long GlobalQ = 20;


// Uint16 i=0;   // conteo

unsigned char bandera_lectura;  /* lectura de sensores */

// Lecturas de sensores
int16 x_mag,y_mag,z_mag;                // lecturas del Magnetometro
int16 x_acel,y_acel,z_acel;             // acelerometro
int16 x_gyro,y_gyro,z_gyro;             // gyro

Uint16 datos[6];
Uint16 datosGyro[6];
Uint16 datosMag[6];

// angulos
//float phi[tambuf];
//float theta[tambuf];
//float psi[tambuf];

int16 xa[tambuf],ya[tambuf],za[tambuf];
int16 xg[tambuf],yg[tambuf],zg[tambuf];
int16 xm[tambuf],ymag[tambuf],zm[tambuf];

// estimacion temporales PRUEBA
// quaternion
/*
float n[tambuf];
float e1[tambuf];
float e2[tambuf];
float e3[tambuf];
*/

// varibales TRIAD
float r_temp[3];
float r1[3], r2[3], r3[3];
```

```
float s1[3], s2[3], s3[3];

float acelerometro[3];              // vectores moviles
float magnetometro[3];

float gyro[3];

float rotMat[3][3], mat1[3][3], mat2[3][3], mat3[3][3];              // matrices auxiliares y de rotacion

float ref_mag[3] = {(0.26923076923), (-0.04769230769), (0.238461538461)};              // vectores de referencia [350 -62 310] / 1300
float ref_acel[3] = {0,0, (-9.81)};



// variables del UKF

#define    ts                              0.02
#define    tsm                             0.01        // ts / 2

float q_aux[4];         // cuaternion auxiliar
float aux;

float S[7][7] = { {(0.5),0,0,0,0,0,0},
                                           {0,(0.5),0,0,0,0,0},
                                           {0,0, (0.5),0,0,0,0},
                                           {0,0,0,(0.5),0,0,0},
                                           {0,0,0,0,(0.5),0,0},
                                           {0,0,0,0,0,(0.5),0},
                                           {0,0,0,0,0,0,(0.5)} };

/*
float sqR[7][7] = { {(20),0,0,0,0,0,0},          // matriz de covarianzas de la medicion
                                           {0,(20),0,0,0,0,0},
                                           {0,0, (20),0,0,0,0},
                                           {0,0,0,(0.5),0,0,0},
                                           {0,0,0,0,(0.5),0,0},
                                           {0,0,0,0,0,(0.5),0},
                                           {0,0,0,0,0,0,(0.5)} };
*/
float sqR[7]= {(2),(2),(2),(0.5),(0.5),(0.5),(0.5)};

/*
float sqQ[7][7] = { {(0.1732),0,0,0,0,0,0},     // matriz de covarianzas del estado
                                           {0,(0.1732),0,0,0,0,0},
                                           {0,0, (0.1732),0,0,0,0},
                                           {0,0,0,(0.5),0,0,0},
                                           {0,0,0,0,(0.5),0,0},
                                           {0,0,0,0,0,(0.5),0},
                                           {0,0,0,0,0,0,(0.5)} };
*/
float sqQ[7]= {(0.1732),(0.1732),(0.1732),(0.5),(0.5),(0.5),(0.5)};


float alfa = (0.5);        // factores de ajuste
float beta = (2);

// vectores de peso
float Wm[15] = {(0.75),(0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),
                                    (0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),
                                    (0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857)};

float Wc[15] = {(3.5),(0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),
                                    (0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),
                                    (0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857),(0.017857142857)};
```

```
float sqrtWc1 = (1.3677823999);
float sqrtWc2 = (0.1336306210);

float eta = (1.322875655532);                   // factor de escala


float x[7]={0,0,0,0,0,0,0};           // vector de estado
float ym[7];              // vector de medicion

float A[7][7]; // matriz usada para multiplicar el factor de ajuste eta por S (matriz de covarianzas)
float B[7][7],C[7][7];   // A,B y C son utilizadas como auxiliares

float xSigmaPts[7][15];                         // matriz de puntos sigma
float xSigmaPtsUT[7][15];                       // matriz de puntos sigma con transformada Unscented
float xmeanUT[7];


// Variables

float Sy[7][7];
float ymeanUT[7];
float ySigmaPtsUT[7][15];                       // puntos sigma del modelo de medicion

float Pxy[7][7];                                // matriz de covarianzas cruzadas
float K[7][7];                                  // matriz de ganancias

//  Matrices auxiliares
float Raux[21][7]={0};              // matriz de apoyo
float matAux[7][7]={0};
float vec_aux[7],vec_aux2[7];                               //vector auxiliar
float Paux[7][15];

float moduloq;

/*****************************************/
/*****************************************/


float theta, phi, psi;



void main(void)
{

        Uint16 i,j,k,l;                 // conteo

//      Uint16 cont;

        InitSysCtrl();                  // config de relojes
        InitWatchdog();                 // config watchdog
        InitGpio();                     // config I/O


        DINT;       // deshabilita las interrupciones en el CPU

        InitPieCtrl();  // Inicializa los registros de interrupcion

        // Deshabilita las interrupciones en el CPU y limpia todas las banderas
        IER = 0x0000;
        IFR = 0x0000;
```

```
        // Inicializa la tabla de vectores con apuntadores a las rutinas ISR
        // llenara la tabla incluso si no es usada la interrupcion
        // util para procesos de despuracion de codigo
        InitPieVectTable();

        // redirecciona a la rutina Int_timer0 al ocurrir la interrupcion
        EALLOW;
        PieVectTable.TINT0 = &Int_timer0;
        EDIS;

        // inicio de timers en CPU
        InitCpuTimers();
        // Configure CPU-Timer 0 to interrupt every 20 milliseconds:
        // 150MHz CPU Freq, 20 millisecond Period (in uSeconds)
        ConfigCpuTimer(&CpuTimer0, 150, 20000);

        CpuTimer0Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 0


// Habilita CPU INT1 que esat conectado al CPU-Timer 0:
        IER |= M_INT1;

// Habilita TINT0 en el PIE: Grupo 1 interrupcion 7
        PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

// Habilita Interrupciones Gloables y alta prioridad a los real-time debug events:
        EINT;   // Habilita Global interrupt INTM
        ERTM;   // Habilita Global realtime interrupt DBGM


        InitI2CGpio();                  // Inicio de las entradas y salidas el bus I2C
        InitI2C();              //inicio de periferico

        // PUERTO SERIE
        InitSciaGpio();
        scia_init();


        ResetI2C();             // reset a banderas de aviso en el BUS
        InitAcel();
        InitGyro();
        InitMag();



        //  borra matriz Raux
        for(i=0;i<21;i++)
                for(j=0;j<7;j++)        Raux[i][j]=0;

        // borra Sy
        for(i=0;i<7;i++)
                for(j=0;j<7;j++)        Sy[i][j]=0;



        DelayUs(10000);


        divVector(ref_acel, modulo(ref_acel), s1);              // s1 = ref_acel / mod(ref_acel)  -> constante
        crossp( s1, ref_mag, r_temp);                                   // r_temp = r1 x magnetometro
        divVector(r_temp, modulo(r_temp), s2);                  // s2 = r1 x ref_mag / mod(r1 x ref_mag)
        crossp( s1, s2, s3);                                    // s3 = s1 x s2;
```

```
        while(1)
//      for (cont=0; cont<tambuf; cont++)
        {

                while(!bandera_lectura);        // espera la lectura de sensores
                bandera_lectura = 0;

//              x_acel=xa[cont]; y_acel=ya[cont]; z_acel=za[cont];
//              x_mag=xm[cont]; y_mag=ymag[cont]; z_mag=zm[cont];
//              x_gyro=xg[cont]; y_gyro=yg[cont]; z_gyro=zg[cont];

                // conversiones de los sensores
                convAcel(x_acel, y_acel, z_acel, acelerometro);
                convMag(x_mag, y_mag, z_mag, magnetometro);
                convGyro(x_gyro,y_gyro,z_gyro, gyro);

        //      magX[i]=x_mag; magY[i]=y_mag; magZ[i]=z_mag;

                divVector(acelerometro, modulo(acelerometro), r1);  // r1 = acel / mod(acel) ;

                crossp( r1, magnetometro, r_temp);                              // r_temp = r1 x magnetometro
                divVector(r_temp, modulo(r_temp), r2);                  // r2 = r1 x magnetometro / mod(r1 x magnetometro)

                crossp( r1, r2, r3);                                    // r3 = r1 x r2;

                prodMat3x1(r1, s1, mat1);
                prodMat3x1(r2, s2, mat2);
                prodMat3x1(r3, s3, mat3);

                // matriz de rotacion   Arm = r1*s1_T + r2 * s2_T + r3 * s3_T;
                rotMat[0][0] = mat1[0][0]+mat2[0][0]+mat3[0][0]; rotMat[0][1] = mat1[0][1]+mat2[0][1]+mat3[0][1]; rotMat[0][2] =
mat1[0][2]+mat2[0][2]+mat3[0][2];
                rotMat[1][0] = mat1[1][0]+mat2[1][0]+mat3[1][0]; rotMat[1][1] = mat1[1][1]+mat2[1][1]+mat3[1][1]; rotMat[1][2] =
mat1[1][2]+mat2[1][2]+mat3[1][2];
                rotMat[2][0] = mat1[2][0]+mat2[2][0]+mat3[2][0]; rotMat[2][1] = mat1[2][1]+mat2[2][1]+mat3[2][1]; rotMat[2][2] =
mat1[2][2]+mat2[2][2]+mat3[2][2];

                dcm2quat(rotMat,q_aux);                 // obtiene cuaternion

                ym[0]=gyro[0];
                ym[1]=gyro[1];
                ym[2]=gyro[2];
                ym[3]=q_aux[0];
                ym[4]=q_aux[1];
                ym[5]=q_aux[2];
                ym[6]=q_aux[3];

                /** Obtiene puntos SIGMA ***/
                escXmatrix(eta,S,A);   // A = eta*S;
                puntosSigmaX(x,A,xSigmaPts);    // obtiene puntos sigma  xSigmaPts (7x15)

                // ***************************
                //   PREDICCION DE LOS ESTADOS
                // ***************************

                xmeanUT[0] = 0; xmeanUT[1] = 0; xmeanUT[2] = 0; xmeanUT[3] = 0;
                xmeanUT[4] = 0; xmeanUT[5] = 0; xmeanUT[6] = 0;

                borraMatriz7x15(xSigmaPtsUT);


                for(i=0; i<15 ;i++)
                {
                        proceso(xSigmaPts,xSigmaPtsUT,i);               // ejecuta el proceso sobre los puntos sigma
```

```
		// obtiene la media de los puntos sigma
		xmeanUT[0] += Wm[i] * xSigmaPtsUT[0][i];
		xmeanUT[1] += Wm[i] * xSigmaPtsUT[1][i];
		xmeanUT[2] += Wm[i] * xSigmaPtsUT[2][i];
		xmeanUT[3] += Wm[i] * xSigmaPtsUT[3][i];
		xmeanUT[4] += Wm[i] * xSigmaPtsUT[4][i];
		xmeanUT[5] += Wm[i] * xSigmaPtsUT[5][i];
		xmeanUT[6] += Wm[i] * xSigmaPtsUT[6][i];
}

for (i=0; i<7; i++)
{
		for(j=1; j<15; j++)
		{
			Raux[j-1][i] = sqrtWc2 * (xSigmaPtsUT[i][j] - xmeanUT[i]);
		}
}

Raux[14][0] = sqQ[0]; Raux[15][1] = sqQ[1]; Raux[16][2] = sqQ[2];
Raux[17][3] = sqQ[3]; Raux[18][4] = sqQ[4]; Raux[19][5] = sqQ[5];
Raux[20][6] = sqQ[6];


qr_mR(Raux, S);

multMatrixTT(S,A);			// S' * S

vec_aux[0] = xSigmaPtsUT[0][0] - xmeanUT[0];		// vector
vec_aux[1] = xSigmaPtsUT[1][0] - xmeanUT[1];
vec_aux[2] = xSigmaPtsUT[2][0] - xmeanUT[2];
vec_aux[3] = xSigmaPtsUT[3][0] - xmeanUT[3];
vec_aux[4] = xSigmaPtsUT[4][0] - xmeanUT[4];
vec_aux[5] = xSigmaPtsUT[5][0] - xmeanUT[5];
vec_aux[6] = xSigmaPtsUT[6][0] - xmeanUT[6];

multVects7x1MatrixEsc( sqrtWc1, vec_aux, B);

// suma las matrices A y B
for (i=0;i<7;i++)
{
		for(j=0;j<7;j++)
		{
			C[i][j] = A[i][j] + B[i][j];
		}
}

for (i=0;i<7;i++)
		for(j=0;j<7;j++)  S[i][j]=0;  // borra matriz S


chol(C, S);			// se guarda la matriz de Cholesky

// ********************************
//   PREDICCION DE LAS MEDICIONES
// ********************************
ymeanUT[0] = 0; ymeanUT[1] = 0; ymeanUT[2] = 0; ymeanUT[3] = 0;
ymeanUT[4] = 0; ymeanUT[5] = 0; ymeanUT[6] = 0;

borraMatriz7x15(ySigmaPtsUT);


for(i=0; i<15 ;i++)
{
```

```
            medicion(xSigmaPts,ySigmaPtsUT,i);                  // ejecuta el medicion sobre los puntos sigma

            // obtiene la media de los puntos sigma
            ymeanUT[0] += Wm[i] * ySigmaPtsUT[0][i];
            ymeanUT[1] += Wm[i] * ySigmaPtsUT[1][i];
            ymeanUT[2] += Wm[i] * ySigmaPtsUT[2][i];
            ymeanUT[3] += Wm[i] * ySigmaPtsUT[3][i];
            ymeanUT[4] += Wm[i] * ySigmaPtsUT[4][i];
            ymeanUT[5] += Wm[i] * ySigmaPtsUT[5][i];
            ymeanUT[6] += Wm[i] * ySigmaPtsUT[6][i];
}

for (i=0; i<7; i++)
{
            for(j=1; j<15; j++)
            {
                        Raux[j-1][i] = sqrtWc2 * (ySigmaPtsUT[i][j] - ymeanUT[i]);
            }
}

Raux[14][0] = sqR[0]; Raux[15][1] = sqR[1]; Raux[16][2] = sqR[2];
Raux[17][3] = sqR[3]; Raux[18][4] = sqR[4]; Raux[19][5] = sqR[5];
Raux[20][6] = sqR[6];


qr_mR(Raux, Sy);

multMatrixTT(Sy,A);             // S' * S

vec_aux[0] = ySigmaPtsUT[0][0] - ymeanUT[0];        // vector
vec_aux[1] = ySigmaPtsUT[1][0] - ymeanUT[1];
vec_aux[2] = ySigmaPtsUT[2][0] - ymeanUT[2];
vec_aux[3] = ySigmaPtsUT[3][0] - ymeanUT[3];
vec_aux[4] = ySigmaPtsUT[4][0] - ymeanUT[4];
vec_aux[5] = ySigmaPtsUT[5][0] - ymeanUT[5];
vec_aux[6] = ySigmaPtsUT[6][0] - ymeanUT[6];

multVects7x1MatrixEsc( sqrtWc1, vec_aux, B);

// suma las matrices A y B
for (i=0;i<7;i++)
{
            for(j=0;j<7;j++)
            {
                        C[i][j] = A[i][j] + B[i][j];
            }
}

for (i=0;i<7;i++)
            for(j=0;j<7;j++)  Sy[i][j]=0;  // borra matriz S

chol(C, Sy);                    // se guarda la matriz de Cholesky
                                                // en matlab esta se transpone pero para ahorrar
                                                // operaciones se transpone al operarla en las operaciones
                                                // que la usa


multMatVecDiag(xSigmaPtsUT,xmeanUT,Wc,Paux);

matrixPxy(Paux, ySigmaPtsUT, ymeanUT, Pxy);

for (i=0; i<7; i++)  // multiplicacion  mats:  Sy' * Sy
{
            for(j=0; j<7; j++)
```

```
		{
			matAux[i][j] = 0;
			for(k=0;k<7;k++)
			{
				matAux[i][j] += Sy[k][i] * Sy[k][j];
			}
		}
	}

	for (i=0;i<7;i++)
		for(j=0;j<7;j++)  K[i][j]=0;  // borra matriz S

	obtenK(K, matAux, Pxy);

	vec_aux2[0]=ym[0]-ymeanUT[0];
	vec_aux2[1]=ym[1]-ymeanUT[1];
	vec_aux2[2]=ym[2]-ymeanUT[2];
	vec_aux2[3]=ym[3]-ymeanUT[3];
	vec_aux2[4]=ym[4]-ymeanUT[4];
	vec_aux2[5]=ym[5]-ymeanUT[5];
	vec_aux2[6]=ym[6]-ymeanUT[6];

	for (i=0; i<7; i++)
	{
		vec_aux[i]=0;
		for(j=0; j<7; j++)
		{
			vec_aux[i] += K[i][j] * vec_aux2[j];
		}
	}

	// innovacion del estado
	x[0] = xmeanUT[0] + vec_aux[0];
	x[1] = xmeanUT[1] + vec_aux[1];
	x[2] = xmeanUT[2] + vec_aux[2];
	x[3] = xmeanUT[3] + vec_aux[3];
	x[4] = xmeanUT[4] + vec_aux[4];
	x[5] = xmeanUT[5] + vec_aux[5];
	x[6] = xmeanUT[6] + vec_aux[6];

	// normalizando el cuaternion del vector de estado
	q_aux[0] = x[3]*x[3];
	q_aux[1] = x[4]*x[4];
	q_aux[2] = x[5]*x[5];
	q_aux[3] = x[6]*x[6];

	aux = q_aux[0] + q_aux[1] + q_aux[2] + q_aux[3];

	moduloq = sqrt(aux);

	x[3] = (x[3]/moduloq);		// cuaternion normalizado
	x[4] = (x[4]/moduloq);
	x[5] = (x[5]/moduloq);
	x[6] = (x[6]/moduloq);

/*******************************/
/********** PRUEBA *************/
/*******************************/
/*	n[cont] = x[3];
	e1[cont] = x[4];
	e2[cont] = x[5];
	e3[cont] = x[6];*/
/*******************************/
```

```
		for (i=0; i<7 ; i++)			// multiplica, K * Sy' (upMat en MATLAB)
		{
			for(j=0; j <7 ; j++)
			{
				matAux[i][j] = 0;
				for(k=0; k<7 ; k++)
				{
					matAux[i][j] += K[i][k] * Sy[j][k];
				}
			}
		}

/*
		for (i=0; i<7 ; i++)			// multiplica, S' * S
		{
			for(j=0; j <7 ; j++)
			{
				A[i][j] = 0;
				for(k=0; k<7 ; k++)
				{
					A[i][j] += S[k][i]*S[k][j];
				}
			}
		}
*/
		for(i=0;i<7;i++)			// copia matriz S -> C
			for(j=0;j<7;j++)	C[i][j]=S[i][j];


		for (l=0; l<7; l++)
		{
			multMatrixTT(C,A);

			for (i=0; i< 7; i++)
			{
				for (j=0; j<7; j++)
				{
					B[i][j] = A[i][j] - (matAux[i][l] * matAux[j][l]);
				}
			}

			for(i=0;i<7;i++)			// borra C
				for(j=0;j<7;j++)	C[i][j]=0;

			chol(B, C);				// Obtiene al final: S = chol(S'*S - upMat(:,k)*upMat(:,k)'); (MATLAB)
		}


		for (i=0; i< 7; i++)			// obtiene S'
		{
			for (j=0; j<7; j++)
			{
				S[i][j] = C[j][i];
			}
		}




/*
		for (k=0; k<7; k++)
		{
```

```
                                for(j=0; j<7; j++)
                                {
                                        for(i=0;i<k;i++)
                                        {
                                                [i][j] += _IQmpy( [i][j], [j][i]);
                                        }
                                }
                        }

                        for (i=0; i<7; i++)
                        {
                                for (j=i+1; j<7; j++)
                                {
                                        [i+1][j] = [j][i+1];
                                }
                        }s
*/

                        phi = atan2(2*(x[3]*x[4] + x[5]*x[6]), 1-2*(x[4]*x[4] + x[5]*x[5] )) * 57.2958;
                        theta = asin(2*(x[3]*x[5] - x[6]*x[4])) * 57.2958;
                        psi = atan2(2*(x[3]*x[6] + x[4]*x[5]), 1-2*(x[5]*x[5] + x[6]*x[6] )) * 57.2958;


        }               // end WHILE(1) o FOR(1:tambuf)


        while(1);


}
```

```
/******** OPERACIONES EXTRA *************/

        // producto interno, toma las columnas como vectores de una matriz M
float prod_intM7(float mat1[7][7], Uint16 col1, float mat2[7][7], Uint16 col2)
{
        Uint16 i;
        float resultado=0;

        for(i=0; i < 7; i++)
        {
                resultado += mat1[i][col1] * mat2[i][col2];

        }
        return resultado;
}

void qr_new( float mat[7][7], float Q[7][7], float R[7][7] )
{
        Uint16 i=0,j;
```

```
	Uint16 a=0,b=0,c=0,d=0;

	float temp;
//	float v[orden][orden];				// puede quitarse
//	float u2[orden],u3[orden],u4[orden],u5[orden],u6[orden],u7[orden];
	float u[7][7];				// puede quitarse
//	float c21, c32, c31, c43, c42, c41;

	float coef_c[7-1][7-1];
	float modulos[7];	// modulos de los vectores

	float prodint[7-1];

	for (i=0; i<7;i++)	u[i][0] = mat[i][0];	// u1 = v1

	for( a=0; a < 7-1; a++ )
	{
		prodint[a] = prod_intM7(u,a,u,a);		// obtiene coeficientes C

		for( b=0; b<=a; b++)
		{
			coef_c[a][b] = (prod_intM7(mat,a+1,u,b) / prodint[b]);
		}

		for( d=0; d<7; d++)
		{
			u[d][a+1] = mat[d][a+1];
			for( c=0; c<(a+1); c++)
			{
				temp = coef_c[a][c] * u[d][c];
				u[d][a+1] = u[d][a+1] - temp;
			}
		}
	}

	for (j=0; j<7;j++)
	{
		modulos[j]=0;
		for (i=0; i<7;i++)
		{
			modulos[j] = modulos[j] + u[i][j] * u[i][j];
		}
		modulos[j] = sqrt(modulos[j]);
	}

	// MATRIZ "Q"
	for (j=0; j<7;j++)
	{
		for (i=0; i<7;i++)
		{
			Q[i][j] = (u[i][j] / modulos[j]);
		}
	}

	// MATRIZ "R"
	R[0][0] = modulos[0];
	R[0][1] = coef_c[0][0]*modulos[0];
	R[0][2] = coef_c[1][0]*modulos[0];
	R[0][3] = coef_c[2][0]*modulos[0];
	R[0][4] = coef_c[3][0]*modulos[0];
	R[0][5] = coef_c[4][0]*modulos[0];
	R[0][6] = coef_c[5][0]*modulos[0];

	R[1][0] = 0;
```

```
	R[1][1] = modulos[1];
	R[1][2] = coef_c[1][1]*modulos[1];
	R[1][3] = coef_c[2][1]*modulos[1];
	R[1][4] = coef_c[3][1]*modulos[1];
	R[1][5] = coef_c[4][1]*modulos[1];
	R[1][6] = coef_c[5][1]*modulos[1];

	R[2][0] = 0;
	R[2][1] = 0;
	R[2][2] = modulos[2];
	R[2][3] = coef_c[2][2]*modulos[2];
	R[2][3] = coef_c[3][2]*modulos[2];
	R[2][3] = coef_c[4][2]*modulos[2];
	R[2][3] = coef_c[5][2]*modulos[2];

	R[3][0] = 0;
	R[3][1] = 0;
	R[3][2] = 0;
	R[3][3] = modulos[3];
	R[3][4] = coef_c[3][3]*modulos[3];
	R[3][5] = coef_c[4][3]*modulos[3];
	R[3][6] = coef_c[5][3]*modulos[3];

	R[4][0] = 0;
	R[4][1] = 0;
	R[4][2] = 0;
	R[4][3] = 0;
	R[4][4] = modulos[4];
	R[4][5] = coef_c[4][4]*modulos[4];
	R[4][6] = coef_c[5][4]*modulos[4];

	R[5][0] = 0;
	R[5][1] = 0;
	R[5][2] = 0;
	R[5][3] = 0;
	R[5][4] = 0;
	R[5][5] = modulos[5];
	R[5][6] = coef_c[5][5]*modulos[5];

	R[6][0] = 0;
	R[6][1] = 0;
	R[6][2] = 0;
	R[6][3] = 0;
	R[6][4] = 0;
	R[6][5] = 0;
	R[6][6] = modulos[4];

}

void obtenK(float X[7][7], float A[7][7], float B[7][7])
{
	int16 i,j,k;

	float Qtemp[7][7];
	float Rtemp[7][7];
	float tempMat[7][7];

	qr_new(A, Qtemp, Rtemp);	// de A obtiene R y Q


	// Multiplicacion de las matrices Q' * B' (transpuestas)
	for (k=0; k<7;k++)
	{
		for (j=0; j<7;j++)
```

```
                {
                        tempMat[j][k] = 0;
                        for (i=0; i<7;i++)
                        {
                                tempMat[j][k] += B[k][i]*Qtemp[i][j];
                        }
                }
        }


        //      Obtiene K'
/*      for (k=0; k<orden;k++)
        {
                for (j=orden-1; j>=0;j--)
                {
                        X[j][k] = 0;
                        for (i=orden-1; i>=j+1;i--)
                        {
                                X[j][k] += _IQmpy(Rtemp[j][i],X[i][k]);
                        }
                        X[j][k] = _IQdiv(tempMat[j][k]-X[j][k], Rtemp[j][j]);
                }
        }
*/

        //      Obtiene K'
        for (k=0; k<7;k++)
        {
                for (j=7-1; j>=0;j--)
                {
                        X[k][j] = 0;
                        for (i=7-1; i>=j+1;i--)
                        {
                                X[k][j] += Rtemp[j][i]*X[k][i];
                        }
                        X[k][j] = (tempMat[j][k]-X[k][j]) / Rtemp[j][j];
                }
        }

}

void matrixPxy(float mat1[7][15], float mat2[7][15], float vec[7], float res[7][7]) // mat1-> normal (7x15), mat2(transpuesta) -> (15x7), res(7x7), vec ->
media y
{
        Uint16 i,j,k;

        for(i=0; i<7  ; i++)
        {
                for(j=0; j<7  ;j++)
                {
                        res[i][j] = 0;

                        for(k=0; k<15 ; k++)
                        {
                                res[i][j] += mat1[i][k]*(mat2[j][k] - vec[j]);
                        }
                }
        }
}


void multMatVecDiag(float mat[7][15], float vec[7], float diag[15], float res[7][15])
{
        Uint16 i,j;
```

```
		for(i=0; i<7 ;i++)
		{
			for(j=0; j<15 ; j++)
			{
				res[i][j] = (mat[i][j]-vec[i])* diag[j];
			}
		}
}


void medicion(float sigma[7][15], float sigmaUT[7][15], Uint16 columna)
{
		sigmaUT[0][columna] = sigma[0][columna];
		sigmaUT[1][columna] = sigma[1][columna];
		sigmaUT[2][columna] = sigma[2][columna];
		sigmaUT[3][columna] = sigma[3][columna];
		sigmaUT[4][columna] = sigma[4][columna];
		sigmaUT[5][columna] = sigma[5][columna];
		sigmaUT[6][columna] = sigma[6][columna];
}

// Descomposicion de Cholesky
void chol(float mat[7][7], float cholmat[7][7])
{
		Uint16 i,j,k;
		float temp;

		for(i=0; i < 7; i++ )
		{
			temp=0;
			for (k=0; k < 7; k++ )   // obtiene matriz rango 1 a partir de renglones
			{
				temp = temp + cholmat[i][k]*cholmat[i][k];
			}

			cholmat[i][i] = sqrt(mat[i][i] - temp);

			for (j=i+1; j < 7; j++)
			{

				temp=0;
				for (k=0; k < 7; k++ )   // obtiene matriz rango 1 a partir de renglones
				{
					temp = temp + cholmat[i][k] * cholmat[j][k];
				}

				cholmat[j][i] = (mat[i][j]-temp) * cholmat[i][i];
			}
		}
}


void multMatrixTT(float mat[7][7], float resp[7][7])      //  S'*S (S -> matriz triangular superior)
{
		Uint16 i,j,k;

			for (i=0; i<7; i++)                    // obtiene una matriz triangular superior
			{
				for(j=i; j<7; j++)
				{
					resp[i][j] = 0;
					for(k=0;k<=i;k++)
					{
```

```
                                        resp[i][j] += mat[k][i]*mat[k][j];
                                }
                        }
                }

                for (i=0; i<7; i++)             // copia los valores al triangulo inferior
                {
                        for (j=i+1; j<7; j++)
                        {
                                resp[j][i] = resp[i][j];
                        }
                }
}


void multVects7x1MatrixEsc(float escalar, float vec[7], float mat[7][7])        // mat = escalar * (vec * vec')
{
        Uint16 i,j;
        for (i=0;i<7;i++)
        {
                for(j=0;j<7;j++)
                {
                        mat[i][j] = vec[i]*vec[j]*escalar;
                }
        }
}


        // producto interno, toma las columnas como vectores de una matriz M
float prod_intM21(float mat1[21][7], Uint16 col1, float mat2[21][7], Uint16 col2)
{
        Uint16 i;
        float resultado=0;

        for(i=0; i < 21; i++)
        {
                resultado += mat1[i][col1]*mat2[i][col2];

        }
        return resultado;
}


void qr_mR( float mat[21][7], float R[7][7] )           // obtiene solo la matriz R
{
        Uint16 i,j;
        Uint16 a,b,c,d;

        float temp;
//      float v[orden][orden];                  // puede quitarse
//      float u2[orden],u3[orden],u4[orden],u5[orden],u6[orden],u7[orden];
        float u[21][7];                 // puede quitarse
//      float c21, c32, c31, c43, c42, c41;

        float coef_c[7-1][7-1];         // se forma una matriz triangular inferior
        float modulos[7];       // modulos de los vectores

        float prodint[7-1];

        for (i=0; i<21;i++)     u[i][0] = mat[i][0];    // u1 = v1  (21 -> #renglones )

        for( a=0; a < 7-1; a++ )  // 7-1 -> #columnas-1
```

```
{
        prodint[a] = prod_intM21(u,a,u,a);                              // obtiene coeficientes C y los vectores u

        for( b=0; b<=a; b++)
        {
                coef_c[a][b] = (prod_intM21(mat,a+1,u,b)/prodint[b]);
        }

        for( d=0; d<21; d++)  // 21 -> #renglones
        {
                u[d][a+1] = mat[d][a+1];
                for( c=0; c<(a+1); c++)         //          -> ALERTA 2*
                {
                        temp = coef_c[a][c]*u[d][c];
                        u[d][a+1] = u[d][a+1] - temp;
                }
        }
}

for (j=0; j<7;j++)                      //  modulo de los vectores u (7 -> #columnas)
{
        modulos[j]=0;
        for (i=0; i<21;i++)   // 21 -> #renglones
        {
                modulos[j] = modulos[j] + u[i][j]*u[i][j];
        }
        modulos[j] = sqrt(modulos[j]);
}

// MATRIZ "R"

// MATRIZ "R"
R[0][0] = modulos[0];
R[0][1] = coef_c[0][0]*modulos[0];
R[0][2] = coef_c[1][0]*modulos[0];
R[0][3] = coef_c[2][0]*modulos[0];
R[0][4] = coef_c[3][0]*modulos[0];
R[0][5] = coef_c[4][0]*modulos[0];
R[0][6] = coef_c[5][0]*modulos[0];

R[1][0] = 0;
R[1][1] = modulos[1];
R[1][2] = coef_c[1][1]*modulos[1];
R[1][3] = coef_c[2][1]*modulos[1];
R[1][4] = coef_c[3][1]*modulos[1];
R[1][5] = coef_c[4][1]*modulos[1];
R[1][6] = coef_c[5][1]*modulos[1];

R[2][0] = 0;
R[2][1] = 0;
R[2][2] = modulos[2];
R[2][3] = coef_c[2][2]*modulos[2];
R[2][3] = coef_c[3][2]*modulos[2];
R[2][3] = coef_c[4][2]*modulos[2];
R[2][3] = coef_c[5][2]*modulos[2];

R[3][0] = 0;
R[3][1] = 0;
R[3][2] = 0;
R[3][3] = modulos[3];
R[3][4] = coef_c[3][3]*modulos[3];
R[3][5] = coef_c[4][3]*modulos[3];
R[3][6] = coef_c[5][3]*modulos[3];
```

```
		R[4][0] = 0;
		R[4][1] = 0;
		R[4][2] = 0;
		R[4][3] = 0;
		R[4][4] = modulos[4];
		R[4][5] = coef_c[4][4]*modulos[4];
		R[4][6] = coef_c[5][4]*modulos[4];

		R[5][0] = 0;
		R[5][1] = 0;
		R[5][2] = 0;
		R[5][3] = 0;
		R[5][4] = 0;
		R[5][5] = modulos[5];
		R[5][6] = coef_c[5][5]*modulos[5];

		R[6][0] = 0;
		R[6][1] = 0;
		R[6][2] = 0;
		R[6][3] = 0;
		R[6][4] = 0;
		R[6][5] = 0;
		R[6][6] = modulos[4];
}


void proceso(float sigma[7][15], float sigmaUT[7][15], Uint16 columna)
{
		float temp;

		sigmaUT[0][columna] = sigma[0][columna];
		sigmaUT[1][columna] = sigma[1][columna];
		sigmaUT[2][columna] = sigma[2][columna];

		temp = (sigma[0][columna]*sigma[4][columna]) + (sigma[1][columna]*sigma[5][columna]) + (sigma[2][columna]*sigma[6][columna]);
		sigmaUT[3][columna] = sigma[3][columna] - (temp*tsm);

		temp = (sigma[0][columna]*sigma[3][columna]) - (sigma[1][columna]*sigma[6][columna]) + (sigma[2][columna]*sigma[5][columna]);
		sigmaUT[4][columna] = sigma[4][columna] + (temp*tsm);

		temp = (sigma[0][columna]*sigma[6][columna]) + (sigma[1][columna]*sigma[3][columna]) - (sigma[2][columna]*sigma[4][columna]);
		sigmaUT[5][columna] = sigma[5][columna] + (temp*tsm);

		temp = (sigma[1][columna]*sigma[4][columna]) + (sigma[2][columna]*sigma[3][columna]) - (sigma[0][columna]*sigma[5][columna]);
		sigmaUT[6][columna] = sigma[6][columna] - (temp*tsm);
}



void borraMatriz7x15(float mat[7][15])
{
		Uint16 i,j;
		for(i=0;i<7;i++)
		{
			for(j=0;j<15;j++)	mat[i][j] = 0;
		}

}


		// Puntos sigma del estado
void puntosSigmaX(float vector[7], float mat[7][7], float resp[7][15])
{
		Uint16 i,j;
```

```
		resp[0][0]=vector[0];
		resp[1][0]=vector[1];
		resp[2][0]=vector[2];
		resp[3][0]=vector[3];
		resp[4][0]=vector[4];
		resp[5][0]=vector[5];
		resp[6][0]=vector[6];

		// reduccion de operaciones SUPRIMIENDO los for

		for(i=0;i<7;i++)
		{
			for(j=0;j<7;j++)
			{
				resp[i][j+1] = vector[i] + mat[i][j];
			}
		}

		for(i=0;i<7;i++)
		{
			for(j=0;j<7;j++)
			{
				resp[i][j+8] = vector[i] - mat[i][j];
			}
		}

}


		// escalar por matriz 7x7
void escXmatrix(float escalar, float Mat[7][7], float res[7][7])
{
		Uint16 i,j;
		for(i=0;i<7;i++)
		{
			for(j=0;j<7;j++)
			{
				res[i][j] = escalar*Mat[i][j];
			}
		}
}



/*  Paso de DCM a quaternion */
/* hay 4 casos */
void dcm2quat(float Mat[3][3], float qc[4])
{
		float aux1, aux2;
		float dif[3];
		float temp;
		float d[3];

		aux1 = Mat[0][0] + Mat[1][1] + Mat[2][2];

		dif[0]=Mat[0][1] - Mat[1][0];
		dif[1]=Mat[1][2] - Mat[2][1];
		dif[2]=Mat[2][0] - Mat[0][2];

		if(aux1 > 0)
		{
			aux2 = sqrt(aux1 + (1));
			temp = aux2*2;
```

```
                qc[0] = 0.5*aux2;
                qc[1] = (dif[1]/temp);
                qc[2] = (dif[2]/temp);
                qc[3] = (dif[0]/temp);
        }
        else
        {
                d[0]=Mat[0][0]; d[1]=Mat[1][1]; d[2]=Mat[2][2];

                if( (d[1] > d[0]) && (d[1] > d[2]) )
                {
                        // valor maximo en Mat[1][1]
                        aux2 = d[1] - d[0] - d[2] + (1);
                        aux1 = sqrt(aux2);
                        qc[2] = 0.5*aux1;

                        if( aux1 != 0)          aux1=((0.5)/aux1);

                        qc[0] = (dif[2]*aux1);
                        qc[1] = (dif[0]*aux1);
                        qc[3] = (dif[1]*aux1);
                }
                else if (d[2] > d[0])
                {
                        aux2 = d[2] - d[0] - d[1] + (1);
                        aux1 = sqrt(aux2);
                        qc[3] = ((0.5)*aux1);

                        if( aux1 != 0)          aux1=((0.5)/aux1);

                        qc[0] = (dif[0]*aux1);
                        qc[1] = (dif[2]*aux1);
                        qc[2] = (dif[1]*aux1);
                }
                else
                {
                        aux2 = d[0] - d[1] - d[2] + (1);
                        aux1 = sqrt(aux2);
                        qc[3] = ((0.5)*aux1);

                        if( aux1 != 0)          aux1=((0.5)/aux1);

                        qc[0] = (dif[1]*aux1);
                        qc[1] = (dif[0]*aux1);
                        qc[2] = (dif[2]*aux1);
                }
        }
}

        // producto cruz
void crossp( float vec1[3], float vec2[3], float prod[3])
{

        prod[0] = (vec2[2]*vec1[1]) - (vec2[1]*vec1[2]);
        prod[1] = (vec2[0]*vec1[2]) - (vec2[2]*vec1[0]);
        prod[2] = (vec2[1]*vec1[0]) - (vec2[0]*vec1[1]);

}

        // modulo de un vector de 3x1
float modulo( float vector[3])
{
        float mod;
```

```
	mod = sqrt( (vector[0]*vector[0]) + (vector[1]*vector[1]) + (vector[2]*vector[2]) );

	return mod;
}

	// producto de un vector y un vector transpuesto
	// se obtiene una matriz de 3x3, considerando que vect es columna
void prodMat3x1(float vect[3], float vect_trans[3], float mat3x3[3][3])
{
	mat3x3[0][0] = (vect[0]*vect_trans[0]); mat3x3[0][1] = (vect[0]*vect_trans[1]); mat3x3[0][2] = (vect[0]*vect_trans[2]);
	mat3x3[1][0] = (vect[1]*vect_trans[0]); mat3x3[1][1] = (vect[1]*vect_trans[1]); mat3x3[1][2] = (vect[1]*vect_trans[2]);
	mat3x3[2][0] = (vect[2]*vect_trans[0]); mat3x3[2][1] = (vect[2]*vect_trans[1]); mat3x3[2][2] = (vect[2]*vect_trans[2]);
}


	// division vector - escalar
void divVector(float vector[3], float escalar, float result[3])
{
	result[0] = (vector[0]/escalar);
	result[1] = (vector[1]/escalar);
	result[2] = (vector[2]/escalar);
}



/******* SENSORES **********/

void InitAcel(void)
{
	EscrituraI2C(DirAcelI2C,0x2d,0);		// pw ctl
	EscrituraI2C(DirAcelI2C,0x31,0x0b);
	EscrituraI2C(DirAcelI2C,0x25,16);
	EscrituraI2C(DirAcelI2C,0x24,8);
	EscrituraI2C(DirAcelI2C,0x26,5);
	EscrituraI2C(DirAcelI2C,0x2c,0x0a);
	EscrituraI2C(DirAcelI2C,0x2d,0x28);
}

void ResetI2C(void)
{
	I2caRegs.I2CMDR.bit.IRS = 0;	// reset a banderas de aviso en el BUS
	DelayUs(10);
	I2caRegs.I2CMDR.bit.IRS = 1;
}


void InitGyro(void)
{
	EscrituraI2C(DirGyroI2C,0x3e,0x80);			// RESET al gyro
	EscrituraI2C(DirGyroI2C,0x15,9);		// sample rate to 100Hz
	EscrituraI2C(DirGyroI2C,0x16,0x1b);			// 42Hz low pass, 1kHz internal sample rate
	EscrituraI2C(DirGyroI2C,0x17,0x25);			//interrupt for data available and new clock
	EscrituraI2C(DirGyroI2C,0x3e,0x00);			// RESET al gyro

}

void InitMag(void)
{
	EscrituraI2C(DirMagI2C, 0, 0x18);		// Config A, normal Measure, rate 50 Hz
	EscrituraI2C(DirMagI2C, 1, 0x20);		// Config B, default(+- 1Gauss-1300count/mGauss)
	EscrituraI2C(DirMagI2C, 2, 0);			// modo continuo (0), modo simple (1)
}
```

```
// convierte y ajusta de los datos del acelerometro a m/s2
void convAcel(int16 ax, int16 ay, int16 az, float gs[3])
{
	// Realiza una conversion a unidades de [m/s^2]
	// y quita el bias que posee el eje del sensor

	float temp;
	temp = (ax) - (6);	// se le resta el zero-bias
	gs[0] = (temp * (0.0383203125));		// 9.81 / 256

	temp = (ay) - (1);
	gs[1] = (temp * (0.0383203125));

	temp = (az) + (6);
	gs[2] = (temp * (0.0383203125));
}

// convierte datos del gyro
void convGyro(int16 gx, int16 gy, int16 gz, float ws[3])
{
	// Realiza una conversion a unidades de [m/s]
	// y quita el bias que posee el eje del sensor para evitar el "drift"

	float temp;
//	temp = gx - 48;
	temp = (gx) - (39);
//	ws[0] = temp*PI/14.375/180;	// g * (PI / 180 / 14.375) = 0.00121414208834
	ws[0] = (temp * (0.00121414208834));

//	temp = gy + 20;
	temp = (gy) + (10);
//	ws[1] = temp*PI/14.375/180;
	ws[1] = (temp * (0.00121414208834));

//	temp = gz - 9;
	temp = (gz);
//	ws[2] = temp*PI/14.375/180;
	ws[2] = (temp * (0.00121414208834));
}

void convMag(float mx, float my, float mz, float mags[3])
{
	mags[0] = mx/1300;
	mags[1] = my/1300;
	mags[2] = mz/1300;
}


/****** INTERRUPCION *********/
interrupt void Int_timer0(void)
{
//	CpuTimer0.InterruptCount++;
//	LED ^= 1;  // prende y apaga LED

	// lectura de sensores y acopla en localidades de 16 bits

		LecturaI2C(DirAcelI2C, AcelEjeX0, 6, datos);

		x_acel = (datos[1] << 8) | datos[0];
		y_acel = (datos[3] << 8) | datos[2];
		z_acel = (datos[5] << 8) | datos[4];
```

```
                LecturaI2C(DirGyroI2C, 0x1d, 6, datosGyro);

                x_gyro = (datosGyro[0] << 8) | datosGyro[1];
                y_gyro = (datosGyro[2] << 8) | datosGyro[3];
                z_gyro = (datosGyro[4] << 8) | datosGyro[5];


                LecturaI2C(DirMagI2C, 0x03, 6, datosMag);

                y_mag = (datosMag[0] << 8) | datosMag[1];          // X  e Y: volteados en el sensor
                x_mag = (datosMag[2] << 8) | datosMag[3];
                z_mag = (datosMag[4] << 8) | datosMag[5];

                bandera_lectura = 1;

        PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; //Ack de interrupcion para recibir mas interrupciones
}
```

Bootloader

```
//*****************************************************************************
//
// bl_main.c - The file holds the main control loop of the boot loader.
//
// Copyright (c) 2006-2010 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 5961 of the Stellaris Firmware Development Package.
//
//*****************************************************************************

#include "inc/hw_gpio.h"
#include "inc/hw_flash.h"
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_ssi.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include "bl_config.h"
#include "boot_loader/bl_commands.h"
#include "boot_loader/bl_decrypt.h"
#include "boot_loader/bl_flash.h"
#include "boot_loader/bl_hooks.h"
#include "boot_loader/bl_i2c.h"
#include "boot_loader/bl_packet.h"
#include "boot_loader/bl_ssi.h"
#include "boot_loader/bl_uart.h"

//*****************************************************************************
//
// Make sure that the application start address falls on a flash page boundary
//
//*****************************************************************************
#if (APP_START_ADDRESS & (FLASH_PAGE_SIZE - 1))
#error ERROR: APP_START_ADDRESS must be a multiple of FLASH_PAGE_SIZE bytes!
#endif

//*****************************************************************************
//
// Make sure that the flash reserved space is a multiple of flash pages.
//
//*****************************************************************************
#if (FLASH_RSVD_SPACE & (FLASH_PAGE_SIZE - 1))
#error ERROR: FLASH_RSVD_SPACE must be a multiple of FLASH_PAGE_SIZE bytes!
#endif

//*****************************************************************************
//
//! \addtogroup bl_main_api
```

```
//! @{
//
//*****************************************************************************
#if defined(I2C_ENABLE_UPDATE) || defined(SSI_ENABLE_UPDATE) || \
    defined(UART_ENABLE_UPDATE) || defined(DOXYGEN)

//*****************************************************************************
//
// A prototype for the function (in the startup code) for calling the
// application.
//
//*****************************************************************************
extern void CallApplication(unsigned long ulBase);

//*****************************************************************************
//
// A prototype for the function (in the startup code) for a predictable length
// delay.
//
//*****************************************************************************
extern void Delay(unsigned long ulCount);

//*****************************************************************************
//
// Holds the current status of the last command that was issued to the boot
// loader.
//
//*****************************************************************************
unsigned char g_ucStatus;

//*****************************************************************************
//
// This holds the current remaining size in bytes to be downloaded.
//
//*****************************************************************************
unsigned long g_ulTransferSize;

//*****************************************************************************
//
// This holds the total size of the firmware image being downloaded (if the
// protocol in use provides this).
//
//*****************************************************************************
#ifdef BL_PROGRESS_FN_HOOK
unsigned long g_ulImageSize;
#endif

//*****************************************************************************
//
// This holds the current address that is being written to during a download
// command.
//
//*****************************************************************************
unsigned long g_ulTransferAddress;

//*****************************************************************************
//
// This is the data buffer used during transfers to the boot loader.
//
//*****************************************************************************
unsigned long g_pulDataBuffer[BUFFER_SIZE];

//*****************************************************************************
//
```

```
// This is an specially aligned buffer pointer to g_pulDataBuffer to make
// copying to the buffer simpler.  It must be offset to end on an address that
// ends with 3.
//
//*****************************************************************************
unsigned char *g_pucDataBuffer;

//*****************************************************************************
//
// Converts a word from big endian to little endian.  This macro uses compiler-
// specific constructs to perform an inline insertion of the "rev" instruction,
// which performs the byte swap directly.
//
//*****************************************************************************
#if defined(ewarm)
#include <intrinsics.h>
#define SwapWord(x)           __REV(x)
#endif
#if defined(codered) || defined(gcc) || defined(sourcerygxx)
#define SwapWord(x) __extension__                           \
      ({                                        \
          register unsigned long __ret, __inp = x;          \
          __asm__("rev %0, %1" : "=r" (__ret) : "r" (__inp)); \
          __ret;                                \
      })
#endif
#if defined(rvmdk) || defined(__ARMCC_VERSION)
#define SwapWord(x)           __rev(x)
#endif
#if defined(ccs)
unsigned long
SwapWord(unsigned long x)
{
   __asm("    rev     r0, r0\n"
         "    bx      lr\n"); // need this to make sure r0 is returned
   return(x + 1); // return makes compiler happy - ignored
}
#endif

//*****************************************************************************
//
//! Configures the microcontroller.
//!
//! This function configures the peripherals and GPIOs of the microcontroller,
//! preparing it for use by the boot loader.  The interface that has been
//! selected as the update port will be configured, and auto-baud will be
//! performed if required.
//!
//! \return None.
//
//*****************************************************************************
void
ConfigureDevice(void)
{
#ifdef UART_ENABLE_UPDATE
   unsigned long ulProcRatio;
#endif

#ifdef CRYSTAL_FREQ
   //
   // Since the crystal frequency was specified, enable the main oscillator
   // and clock the processor from it.
   //
   HWREG(SYSCTL_RCC) &= ~(SYSCTL_RCC_MOSCDIS);
```

```
    Delay(524288);
    HWREG(SYSCTL_RCC) = ((HWREG(SYSCTL_RCC) & ~(SYSCTL_RCC_OSCSRC_M)) |
                         SYSCTL_RCC_OSCSRC_MAIN);

    //
    // Set the flash programming time based on the specified crystal frequency.
    //
    HWREG(FLASH_USECRL) = ((CRYSTAL_FREQ + 999999) / 1000000) - 1;
#else
    //
    // Set the flash to program at 16 MHz since that is just beyond the fastest
    // that we could run.
    //
    HWREG(FLASH_USECRL) = 15;
#endif

#ifdef I2C_ENABLE_UPDATE
    //
    // Enable the clocks to the I2C and GPIO modules.
    //
    HWREG(SYSCTL_RCGC2) |= SYSCTL_RCGC2_GPIOB;
    HWREG(SYSCTL_RCGC1) |= SYSCTL_RCGC1_I2C0;

    //
    // Configure the GPIO pins for hardware control, open drain with pull-up,
    // and enable them.
    //
    HWREG(GPIO_PORTB_BASE + GPIO_O_AFSEL) |= (1 << 7) | I2C_PINS;
    HWREG(GPIO_PORTB_BASE + GPIO_O_DEN) |= (1 << 7) | I2C_PINS;
    HWREG(GPIO_PORTB_BASE + GPIO_O_ODR) |= I2C_PINS;

    //
    // Enable the I2C Slave Mode.
    //
    HWREG(I2C0_MASTER_BASE + I2C_O_MCR) = I2C_MCR_MFE | I2C_MCR_SFE;

    //
    // Setup the I2C Slave Address.
    //
    HWREG(I2C0_SLAVE_BASE + I2C_O_SOAR) = I2C_SLAVE_ADDR;

    //
    // Enable the I2C Slave Device on the I2C bus.
    //
    HWREG(I2C0_SLAVE_BASE + I2C_O_SCSR) = I2C_SCSR_DA;
#endif

#ifdef SSI_ENABLE_UPDATE
    //
    // Enable the clocks to the SSI and GPIO modules.
    //
    HWREG(SYSCTL_RCGC2) |= SYSCTL_RCGC2_GPIOA;
    HWREG(SYSCTL_RCGC1) |= SYSCTL_RCGC1_SSI0;

    //
    // Make the pin be peripheral controlled.
    //
    HWREG(GPIO_PORTA_BASE + GPIO_O_AFSEL) |= SSI_PINS;
    HWREG(GPIO_PORTA_BASE + GPIO_O_DEN) |= SSI_PINS;

    //
    // Set the SSI protocol to Motorola with default clock high and data
    // valid on the rising edge.
    //
```

```
    HWREG(SSI0_BASE + SSI_O_CR0) = (SSI_CR0_SPH | SSI_CR0_SPO |
                            (DATA_BITS_SSI - 1));

    //
    // Enable the SSI interface in slave mode.
    //
    HWREG(SSI0_BASE + SSI_O_CR1) = SSI_CR1_MS | SSI_CR1_SSE;
#endif

#ifdef UART_ENABLE_UPDATE
    //
    // Enable the the clocks to the UART and GPIO modules.
    //
    HWREG(SYSCTL_RCGC2) |= SYSCTL_RCGC2_GPIOA;
    HWREG(SYSCTL_RCGC1) |= SYSCTL_RCGC1_UART0;

    //
    // Keep attempting to sync until we are successful.
    //
#ifdef UART_AUTOBAUD
    while(UARTAutoBaud(&ulProcRatio) < 0)
    {
    }
#else
    ulProcRatio = UART_BAUD_RATIO(UART_FIXED_BAUDRATE);
#endif

    //
    // Set GPIO A0 and A1 as UART pins.
    //
    HWREG(GPIO_PORTA_BASE + GPIO_O_AFSEL) |= UART_PINS;

    //
    // Set the pin type.
    //
    HWREG(GPIO_PORTA_BASE + GPIO_O_DEN) |= UART_PINS;

    //
    // Set the baud rate.
    //
    HWREG(UART0_BASE + UART_O_IBRD) = ulProcRatio >> 6;
    HWREG(UART0_BASE + UART_O_FBRD) = ulProcRatio & UART_FBRD_DIVFRAC_M;

    //
    // Set data length, parity, and number of stop bits to 8-N-1.
    //
    HWREG(UART0_BASE + UART_O_LCRH) = UART_LCRH_WLEN_8 | UART_LCRH_FEN;

    //
    // Enable RX, TX, and the UART.
    //
    HWREG(UART0_BASE + UART_O_CTL) = (UART_CTL_UARTEN | UART_CTL_TXE |
                              UART_CTL_RXE);

#ifdef UART_AUTOBAUD
    //
    // Need to ack in the UART case to hold it up while we get things set up.
    //
    AckPacket();
#endif
#endif
}

//*****************************************************************************
```

```
//
//! This function performs the update on the selected port.
//!
//! This function is called directly by the boot loader or it is called as a
//! result of an update request from the application.
//!
//! \return Never returns.
//
//*****************************************************************************
void
Updater(void)
{
    unsigned long ulSize, ulTemp, ulFlashSize;

    //
    // This ensures proper alignment of the global buffer so that the one byte
    // size parameter used by the packetized format is easily skipped for data
    // transfers.
    //
    g_pucDataBuffer = ((unsigned char *)g_pulDataBuffer) + 3;

    //
    // Insure that the COMMAND_SEND_DATA cannot be sent to erase the boot
    // loader before the application is erased.
    //
    g_ulTransferAddress = 0xffffffff;

    //
    // Read any data from the serial port in use.
    //
    while(1)
    {
        //
        // Receive a packet from the port in use.
        //
        ulSize = sizeof(g_pulDataBuffer) - 3;
        if(ReceivePacket(g_pucDataBuffer, &ulSize) != 0)
        {
            continue;
        }

        //
        // The first byte of the data buffer has the command and determines
        // the format of the rest of the bytes.
        //
        switch(g_pucDataBuffer[0])
        {
            //
            // This was a simple ping command.
            //
            case COMMAND_PING:
            {
                //
                // This command always sets the status to COMMAND_RET_SUCCESS.
                //
                g_ucStatus = COMMAND_RET_SUCCESS;

                //
                // Just acknowledge that the command was received.
                //
                AckPacket();

                //
                // Go back and wait for a new command.
```

```
            //
            break;
        }

        //
        // This command indicates the start of a download sequence.
        //
        case COMMAND_DOWNLOAD:
        {
            //
            // Until determined otherwise, the command status is success.
            //
            g_ucStatus = COMMAND_RET_SUCCESS;

            //
            // A simple do/while(0) control loop to make error exits
            // easier.
            //
            do
            {
                //
                // See if a full packet was received.
                //
                if(ulSize != 9)
                {
                    //
                    // Indicate that an invalid command was received.
                    //
                    g_ucStatus = COMMAND_RET_INVALID_CMD;

                    //
                    // This packet has been handled.
                    //
                    break;
                }

                //
                // Get the address and size from the command.
                //
                g_ulTransferAddress = SwapWord(g_pulDataBuffer[1]);
                g_ulTransferSize = SwapWord(g_pulDataBuffer[2]);
#ifdef BL_PROGRESS_FN_HOOK
                g_ulImageSize = g_ulTransferSize;
#endif

                //
                // This determines the size of the flash available on the
                // part in use.
                //
                ulFlashSize = BL_FLASH_SIZE_FN_HOOK();

                //
                // If we are reserving space at the top of flash then this
                // space is not available for application download but it
                // is available to be updated directly.
                //
#ifdef FLASH_RSVD_SPACE
                if((ulFlashSize - FLASH_RSVD_SPACE) != g_ulTransferAddress)
                {
                    ulFlashSize -= FLASH_RSVD_SPACE;
                }
#endif

                //
```

```
            // Check for a valid starting address and image size.
            //
            if(!BL_FLASH_AD_CHECK_FN_HOOK(g_ulTransferAddress,
                                          g_ulTransferSize))
            {
              //
              // Set the code to an error to indicate that the last
              // command failed.  This informs the updater program
              // that the download command failed.
              //
              g_ucStatus = COMMAND_RET_INVALID_ADR;

              //
              // This packet has been handled.
              //
              break;
            }


            //
            // Only erase the space that we need if we are not
            // protecting the code.
            //
#ifndef FLASH_CODE_PROTECTION
            ulFlashSize = g_ulTransferAddress + g_ulTransferSize;
#endif

            //
            // Clear the flash access interrupt.
            //
            BL_FLASH_CL_ERR_FN_HOOK();

            //
            // Leave the boot loader present until we start getting an
            // image.
            //
            for(ulTemp = g_ulTransferAddress; ulTemp < ulFlashSize;
              ulTemp += FLASH_PAGE_SIZE)
            {
              //
              // Erase this block.
              //
              BL_FLASH_ERASE_FN_HOOK(ulTemp);
            }

            //
            // Return an error if an access violation occurred.
            //
            if(BL_FLASH_ERROR_FN_HOOK())
            {
              g_ucStatus = COMMAND_RET_FLASH_FAIL;
            }
          }
          while(0);

          //
          // See if the command was successful.
          //
          if(g_ucStatus != COMMAND_RET_SUCCESS)
          {
            //
            // Setting g_ulTransferSize to zero makes COMMAND_SEND_DATA
            // fail to accept any data.
            //
```

```
            g_ulTransferSize = 0;
        }

        //
        // Acknowledge that this command was received correctly.  This
        // does not indicate success, just that the command was
        // received.
        //
        AckPacket();

        //
        // If we have a start notification hook function, call it
        // now if everything is OK.
        //
#ifdef BL_START_FN_HOOK
        if(g_ulTransferSize)
        {
            BL_START_FN_HOOK();
        }
#endif

        //
        // Go back and wait for a new command.
        //
        break;
    }

    //
    // This command indicates that control should be transferred to
    // the specified address.
    //
    case COMMAND_RUN:
    {
        //
        // Acknowledge that this command was received correctly.  This
        // does not indicate success, just that the command was
        // received.
        //
        AckPacket();

        //
        // See if a full packet was received.
        //
        if(ulSize != 5)
        {
            //
            // Indicate that an invalid command was received.
            //
            g_ucStatus = COMMAND_RET_INVALID_CMD;

            //
            // This packet has been handled.
            //
            break;
        }

        //
        // Get the address to which control should be transferred.
        //
        g_ulTransferAddress = SwapWord(g_pulDataBuffer[1]);

        //
        // This determines the size of the flash available on the
        // device in use.
```

```
            //
            ulFlashSize = BL_FLASH_SIZE_FN_HOOK();

            //
            // Test if the transfer address is valid for this device.
            //
            if(g_ulTransferAddress >= ulFlashSize)
            {
              //
              // Indicate that an invalid address was specified.
              //
              g_ucStatus = COMMAND_RET_INVALID_ADR;

              //
              // This packet has been handled.
              //
              break;
            }

            //
            // Make sure that the ACK packet has been sent.
            //
            FlushData();

            //
            // Reset and disable the peripherals used by the boot loader.
            //
#ifdef I2C_ENABLE_UPDATE
            HWREG(SYSCTL_RCGC1) &= ~SYSCTL_RCGC1_I2C0;
            HWREG(SYSCTL_SRCR1) = SYSCTL_SRCR1_I2C0;
#endif
#ifdef UART_ENABLE_UPDATE
            HWREG(SYSCTL_RCGC1) &= ~SYSCTL_RCGC1_UART0;
            HWREG(SYSCTL_SRCR1) = SYSCTL_SRCR1_UART0;
#endif
#ifdef SSI_ENABLE_UPDATE
            HWREG(SYSCTL_RCGC1) &= ~SYSCTL_RCGC1_SSI0;
            HWREG(SYSCTL_SRCR1) = SYSCTL_SRCR1_SSI0;
#endif
            HWREG(SYSCTL_SRCR1) = 0;

            //
            // Branch to the specified address.  This should never return.
            // If it does, very bad things will likely happen since it is
            // likely that the copy of the boot loader in SRAM will have
            // been overwritten.
            //
            ((void (*)(void))g_ulTransferAddress)();

            //
            // In case this ever does return and the boot loader is still
            // intact, simply reset the device.
            //
            HWREG(NVIC_APINT) = (NVIC_APINT_VECTKEY |
                                 NVIC_APINT_SYSRESETREQ);

            //
            // The microcontroller should have reset, so this should
            // never be reached.  Just in case, loop forever.
            //
            while(1)
            {
            }
        }
```

```
//
// This command just returns the status of the last command that
// was sent.
//
case COMMAND_GET_STATUS:
{
  //
  // Acknowledge that this command was received correctly.  This
  // does not indicate success, just that the command was
  // received.
  //
  AckPacket();

  //
  // Return the status to the updater.
  //
  SendPacket(&g_ucStatus, 1);

  //
  // Go back and wait for a new command.
  //
  break;
}

//
// This command is sent to transfer data to the device following
// a download command.
//
case COMMAND_SEND_DATA:
{
  //
  // Until determined otherwise, the command status is success.
  //
  g_ucStatus = COMMAND_RET_SUCCESS;

  //
  // If this is overwriting the boot loader then the application
  // has already been erased so now erase the boot loader.
  //
  if(g_ulTransferAddress == 0)
  {
    //
    // Clear the flash access interrupt.
    //
    BL_FLASH_CL_ERR_FN_HOOK();

    //
    // Erase the boot loader.
    //
    for(ulTemp = 0; ulTemp < APP_START_ADDRESS;
      ulTemp += FLASH_PAGE_SIZE)
    {
      //
      // Erase this block.
      //
      BL_FLASH_ERASE_FN_HOOK(ulTemp);
    }

    //
    // Return an error if an access violation occurred.
    //
    if(BL_FLASH_ERROR_FN_HOOK())
    {
```

```
                //
                // Setting g_ulTransferSize to zero makes
                // COMMAND_SEND_DATA fail to accept any more data.
                //
                g_ulTransferSize = 0;

                //
                // Indicate that the flash erase failed.
                //
                g_ucStatus = COMMAND_RET_FLASH_FAIL;
              }
            }

            //
            // Take one byte off for the command.
            //
            ulSize = ulSize - 1;

            //
            // Check if there are any more bytes to receive.
            //
            if(g_ulTransferSize >= ulSize)
            {
              //
              // If we have been provided with a decryption hook function
              // call it here.
              //
#ifdef BL_DECRYPT_FN_HOOK
              BL_DECRYPT_FN_HOOK(g_pucDataBuffer + 1, ulSize);
#endif

              //
              // Write this block of data to the flash
              //
              BL_FLASH_PROGRAM_FN_HOOK(g_ulTransferAddress,
                                       (unsigned char *)
                                       &g_pulDataBuffer[1],
                                       ((ulSize + 3) & ~3));

              //
              // Return an error if an access violation occurred.
              //
              if(BL_FLASH_ERROR_FN_HOOK())
              {
                //
                // Indicate that the flash programming failed.
                //
                g_ucStatus = COMMAND_RET_FLASH_FAIL;
              }
              else
              {
                //
                // Now update the address to program.
                //
                g_ulTransferSize -= ulSize;
                g_ulTransferAddress += ulSize;

                //
                // If a progress hook function has been provided, call
                // it here.
                //
#ifdef BL_PROGRESS_FN_HOOK
                BL_PROGRESS_FN_HOOK(g_ulImageSize - g_ulTransferSize,
                                    g_ulImageSize);
```

```
#endif
            }
          }
          else
          {
            //
            // This indicates that too much data is being sent to the
            // device.
            //
            g_ucStatus = COMMAND_RET_INVALID_ADR;
          }

          //
          // Acknowledge that this command was received correctly.  This
          // does not indicate success, just that the command was
          // received.
          //
          AckPacket();

          //
          // If we have an end notification hook function, and we've
          // reached the end, call it now.
          //
#ifdef BL_END_FN_HOOK
          if(g_ulTransferSize == 0)
          {
            BL_END_FN_HOOK();
          }
#endif

          //
          // Go back and wait for a new command.
          //
          break;
        }

        //
        // This command is used to reset the device.
        //
        case COMMAND_RESET:
        {
          //
          // Send out a one-byte ACK to ensure the byte goes back to the
          // host before we reset everything.
          //
          AckPacket();

          //
          // Make sure that the ACK packet has been sent.
          //
          FlushData();

          //
          // Perform a software reset request.  This will cause the
          // microcontroller to reset; no further code will be executed.
          //
          HWREG(NVIC_APINT) = (NVIC_APINT_VECTKEY |
                               NVIC_APINT_SYSRESETREQ);

          //
          // The microcontroller should have reset, so this should never
          // be reached.  Just in case, loop forever.
          //
          while(1)
```

```
            {
            }
        }

        //
        // Just acknowledge the command and set the error to indicate that
        // a bad command was sent.
        //
        default:
        {
            //
            // Acknowledge that this command was received correctly.  This
            // does not indicate success, just that the command was
            // received.
            //
            AckPacket();

            //
            // Indicate that a bad comand was sent.
            //
            g_ucStatus = COMMAND_RET_UNKNOWN_CMD;

            //
            // Go back and wait for a new command.
            //
            break;
        }
      }
    }
}

//****************************************************************************
//
// Close the Doxygen group.
//! @}
//
//****************************************************************************
#endif
```