UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO
EN INGENIERÍA

FILTRADO DIGITAL DE SEÑALES DE NAVEGACIÓN INERCIAL PARA EL SATÉLITE HUMSAT

T E S I S

QUE PARA OBTENER EL GRADO DE :

MAESTRO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – PROCESAMIENTO
DIGITAL DE SEÑALES

P R E S E N T A :

JIMÉNEZ MADRIGAL EMILIO AUGUSTO

TUTOR DE TESIS:
DR. ESAÚ VICENTE VIVAS

CO-ASESOR EXTERNO**:**
DR. HUGO RODRÍGUEZ CORTÉS

2012

JURADO ASIGNADO:

Presidente:	Dra. Lucía Medina Gómez
Secretario:	Dr. Pablo Roberto Pérez Alcázar
Vocal:	Dr. Esaú Vicente Vivas
1er Suplente:	Dr. Hugo Rodríguez Cortés
2do Suplente:	M. I. Larry Escobar Salguero

Lugares donde se realizó la tesis:

Coordinación de Eléctrica y Computación, Instituto de Ingeniería, UNAM, México
Sección de Mecatrónica, CINVESTAV – IPN, México

Tutor de tesis:

Dr. Esaú Vicente Vivas

_____

FIRMA

Agradecimientos

A la Universidad Nacional Autónoma de México y la Facultad de Ingeniería por haberme dado la oportunidad de profundizar en mis estudios y continuar con una de las actividades que más me motivan, la Ingeniería.

A CONACyT por haberme apoyado económicamente a largo de todo este tiempo permitiendo seguir con mis estudios.

A mis padres Emilio Jiménez Sánchez y Juana Leticia Madrigal por apoyarme incondicionalmente durante esta nueva etapa.

A mis hermanos por su compañía y apoyo.

A todos los integrantes de mi familia, especialmente a mi tía Flora Jiménez Sánchez (†) quien siempre creyó en mi y fue un gran pilar para mi familia.

A mis compañeros de generación y geniales ingenieros: Daniel Calderón, Luis Ángel Contreras, Armando Salomón Hernández, Jorge Armando Rodríguez y Roberto Espinosa por su amistad, y apoyo, you ROCK !!!

A los compañeros del IIMAS que tuve la oportunidad de conocer Héctor Felix, Luis López, César Gómez y Marco Becerra.

A mis compañeros del Instituto de Ingeniería Mario Alberto Mendoza, Ignacio Mendoza, Paul Domínguez, Rodrigo Cordova, Rodrigo Alva, Eduardo Vizcaino, Francisco Osorio (Enkor), Dierk Lueders, Paloma Pedrajas, Alberto Ramírez, Mario Alberto Hernández, Alejandro Cordoba, Luis Zepeda, Erika Mendoza, Genaro Islas, Alejandro Castilla, Miguel Alvarado, Juvenal Villanueva, Jesús Avilés, Esther Barrios y Jeannete Aguilar.

A mis profesores del Posgrado de la Facultad de Ingeniería por compartir sus experiencias y conocimiento.

A mis sinodales Dra. Lucía Medina, Dr. Pablo Pérez Alcázar y el M. I. Larry Escobar, que también fueron mis profesores, por todas sus enseñanzas y aceptar mi trabajo.

A mi asesor el Dr. Esaú Vicente Vivas por el apoyo y la oportunidad de participar dentro de este proyecto y al Dr. Hugo Rodríguez por todo el apoyo teórico que me facilitó y brindarme su apoyo durante mis estancias en el CINVESTAV.

"En este lugar no perdemos demasiado tiempo mirando hacia atrás. ***Camina hacia el futuro****, abriendo nuevas puertas y probando cosas nuevas, se curioso... porque nuestra curiosidad siempre nos conduce por nuevos caminos."*

Walter Elias Disney

"Existen 2 clases de hombres: Aquellos que duermen y sueñan de noche y aquellos que sueñan despiertos y de día … esos hombres son peligrosos, porque no cederán hasta ver sus ***sueños convertidos en realidad****. "*

Thomas Edward Lawrence

Resumen

El presente trabajo se enfoca en la utilización de dos estimadores de estados, Inmersión e Invarianza y el filtro de Kalman Unscented, para resolver el problema de estimación de la orientación a través de señales de sensores de navegación inercial como una primera propuesta para su utilización dentro del proyecto nanosatelital HumSAT-México, para emplear sensores de bajo costo y tamaño pequeño con buenos resultados de orientación.

El problema de estimación se plantea con el uso de sensores de acelerómetro, giróscopos y magnetómetros, así como de un sistema de desarrollo DSP (Digital Signal Processor), incorporados en una mesa suspendida en aire (MSA) que cuenta con la instrumentación fundamental para realizar estas pruebas. La MSA permite una simulación económica de las condiciones en un espacio libre de fricción para realizar pruebas de apuntamiento y de validación en los laboratorios terrestres. Esta plataforma sirve para realizar pruebas de control de orientación en Tierra, de tal modo que posteriormente permitirá extrapolar los modelos implantados en un DSP para uso espacial a bordo del satélite HumSAT en alguno de sus subsistemas de procesamiento.

Se detallan los estimadores utilizados, el primero es la técnica de Inmersión e Invarianza la cual ha cobrado importancia en la actualidad gracias a las ventajas que presenta al reducir un modelo dinámico de cierto orden a un orden menor lo que facilita su manipulación e implementación con menores recursos. El segundo es el filtro Kalman Unscented, otra técnica que recientemente empieza a ser más utilizada ya que logra sensibles mejoras sobre el filtro Kalman Extendido, debido a que trabaja directamente con los modelos no lineales de proceso y medición.

Se describe la implementación de los estimadores sobre la plataforma DSP utilizando una tarjeta de sensores que integra un acelerómetro, un giróscopo y un magnetómetro triaxiales. Mediante el análisis de las implementaciones se hace una propuesta para su aplicación en sistemas de procesamiento que consumen menos energía y de amplio uso.

Finalmente, se exponen, analizan y comparan los resultados obtenidos de los estimadores implementados en el DSP usando punto fijo y punto flotante.

Abstract

This work focuses on the use of two state estimators, the Immersion and Invariance as well as the Unscented Kalman filter to solve the problem of attitude estimation through inertial navigation sensor signals. The approach is a first proposal to use it within the HumSAT-Mexico nanosatellite project in order to employ cost effective and small sensors to reach good attitude results.

The estimation problem is addressed with the support of accelerometers, gyroscopes and magnetometers, as well as with a Digital Signal Processor (DSP) development system. All of them integrated into an air bearing system (ABS) instrument with basic equipment to perform the referred tests. The ABS allows a cost-effective simulation of free space frictionless conditions to perform attitude testing and validation in terrestrial laboratories. Afterwards, the models implemented in a successfully way on the DSP will then be extrapolated to the instrumentation of the HumSAT nanosatellite using one of its on board processing system with the best computing performance.

The first estimator depicted in this thesis uses the Immersion and Invariance technique which has become important nowadays because of its advantages of reducing a dynamic model of some order to a lower order one which therefore facilitates the handling and the implementation with lower resources. The second estimator is the Unscented Kalman filter, another technique that recently is becoming more used as it achieves significant improvements over the Extended Kalman Filter, because it works directly with nonlinear process and measurement models.

The thesis describes the implementation of the estimators on a DSP platform that is connected to a sensors electronic card that integrates three axis accelerometers, gyroscopes and magnetometers. In addition, it is performed an analysis of the implementation results to generate a proposal to implant a solution with processing systems that consume less energy and which are of wider use.

Finally, the thesis performs an analysis and comparation of results of estimators implemented in the DSP using fixed-point and floating-point.

Índice de contenido

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1 SATÉLITES ARTIFICIALES

Un satélite artificial es básicamente una nave espacial fabricada en la Tierra la cual se envía al espacio por medio de un vehículo de lanzamiento. Este vehículo es un cohete propulsado cuyo objetivo es liberar cargas útiles en el espacio exterior. Un satélite artificial es capaz de orbitar alrededor de cuerpos celestes como: lunas, planetas, estrellas o incluso galaxias. Estos tienen una vida útil finita, limitada a la capacidad y calidad de sus sistemas de potencia, una vez que la energía se termina dentro de su sistema, este queda orbitando como basura espacial.

Los satélites artificiales están clasificados de varias formas, pero esencialmente se clasifican por su misión, tipo de órbita y peso. En cuanto a los satélites por tipo de misión algunas de sus clasificaciones son satélites de telecomunicaciones (datos, voz y video), satélites de navegación (sistema GPS), satélites de reconocimiento, satélites meteorológicos y satélites astronómicos.

1.2 SATÉLITES ARTIFICIALES EN MÉXICO

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

Hasta el momento se han tenido grandes experiencias dentro del área de desarrollo de satélites experimentales, una de estas fue el microsatélite experimental SATEX desarrollado por diversas instituciones, entre ellas, el Instituto de Ingeniería de la UNAM, proyecto iniciado en 1995, figura 1.1. La finalización de las actividades encargadas

al Instituto de Ingeniería fue en 2004 cuando se dieron por terminados los sistemas de la computadora de vuelo, el subsistema de sensores, protocolos de comunicaciones, el software de vuelo y el software de estación terrena encargado de monitorear el satélite, figura 1.2. Desafortunadamente, el satélite nunca fue lanzado debido a que no fue completamente terminado.

Figura 1.1 Proyecto SATEX.

Hasta el momento, la única experiencia satelital que se ha terminado y que ha tenido un éxito parcial operativo ha sido la de los satélites Unamsats A y B, en tanto que han existido otras iniciativas que hasta el momento no han cristalizado, entre ellas, el proyecto Unamsat III, el nanosatélite Pumasat y el proyecto Cóndor, así como otras iniciativas en otras dependencias educativas Mexicanas.

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

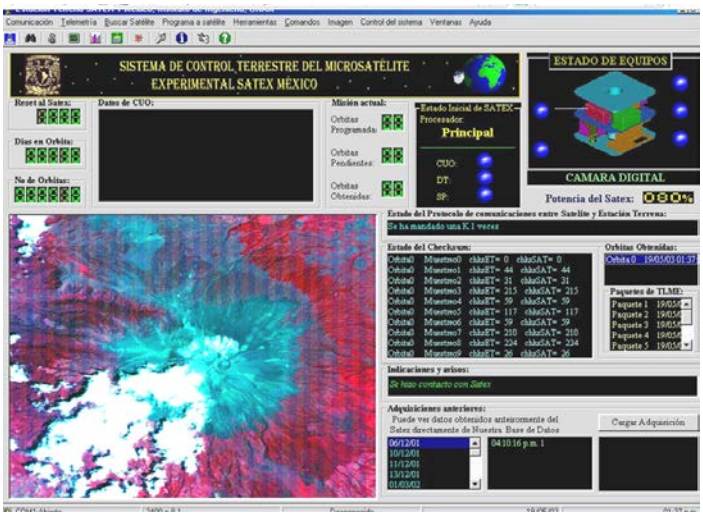


Figura 1.2
Proyecto SATEX,
Software de estación terrena.

1.3 Nano satélites

Una de clasificación de satélite es la llamada nano satélite o también llamada “nanosat” este tipo de satélite entra dentro de la categoría de los pequeños satélites que son aquellos que pesan menos de 500 kg. La clasificación de nano satélite es dada por su peso, este debe ser de entre 1 y 10 kg. La razón principal por la que se empezaron a desarrollar pequeños satélites fue para reducir su elevado costo de fabricación y lanzamiento, mientras que los pequeños pueden ser expulsados con sistemas de lanzamiento más pequeños y menos sofisticados. La utilización que se le ha dado a esta clase de satélites es servir en su mayoría como una plataforma experimental de nuevas cargas útiles, montar constelaciones de satélites para comunicaciones con baja tasa de transferencia de datos y sistemas de percepción remota de bajo costo.

1.4 Proyecto HumSAT

El proyecto HumSAT, es una iniciativa educativa internacional para construir una constelación de nano satélites cuyo principal propósito será el proveer servicios básicos de tele-salud a zonas aisladas de todo el planeta, además de conectar a grupos de usuarios con redes de sensores colocados alrededor del mundo. Estos sensores adquirirán y transmitirán datos vía radio, para monitorear una serie de parámetros de clima, humedad, temperatura del agua, contaminación, así como de datos de tipo industrial correspondientes a redes hidráulicas, de hidrocarburos y redes eléctricas, entre otras, [1].

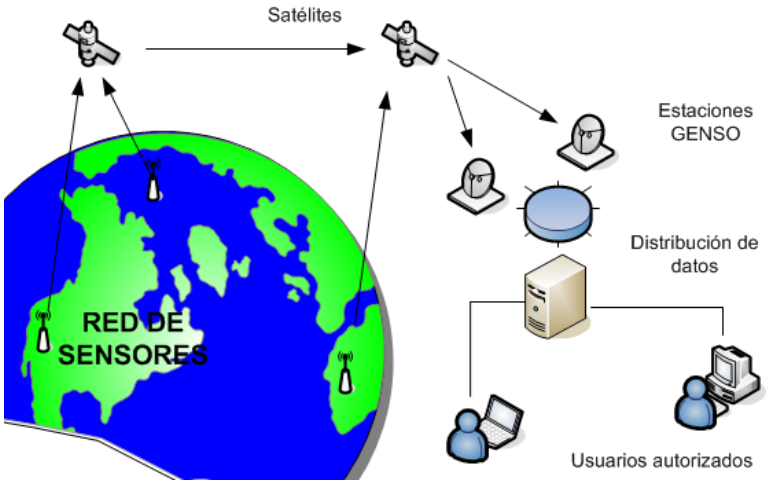


Figura 1.3 Esquema general de GENSO.

La figura 1.3 presenta el esquema general de funcionamiento de la red a montar, se utilizará el sistema GENSO [2] que es software desarrollado por un equipo internacional de estudiantes que permitirá conectarse a diferentes estaciones alrededor del mundo para obtener datos de los sensores o los satélites vía internet, el sistema GENSO es un proyecto de la Agencia Espacial Europea (ESA).

Cada satélite de la constelación HumSAT será construido por las instituciones participantes, dándoles la libertad a cada una que cuenten con las cargas útiles que necesiten o con las que deseen experimentar, solo habrán de tener en cuenta los requisitos del sistema de comunicaciones para hacerlos compatibles con el sistema GENSO.

Los principales conceptos del proyecto HumSAT son:

- Educativo, ya que ofrece que estudiantes participen en la construcción de los satélites dentro de la red.
- Cooperación internacional entre universidades para compartir tecnología espacial.
- Transferencia de tecnología espacial entre universidades participantes.

Dentro de las aplicaciones que se le darán al proyecto están principalmente:

- Telemedicina básica.
- Un sistema de monitoreo de cambio climático.
- Soporte de comunicaciones en áreas de difícil acceso con bajas tasa de transferencia de datos.
- Faro de localización de emergencia, que será de utilidad en iniciativas humanitarias o emergencias como son desastres naturales.

1.5 HumSAT México

El satélite HumSAT México está proyectado para ser un nanosatélite de no más de 3 kg de peso y con dimensiones de 10 x 10 cm en su base y 30 cm de altura, en este espacio deben de acomodadarse todos los subsistemas que conforman las tareas básicas de operación y los experimentos de abordo. En el caso de México cabe recalcar que será la primera vez que serán probados en el espacio todos los subsistemas de un sistema satelital con diseño y manufactura mexicana. En especial, será atacado el problema de la orientación del satélite que puede llegar a ser una tarea crítica para una o varias cargas útiles dentro del nanosatélite. En esta tesis se aborda parte del problema del control de la orientación o actitud del satélite que comprende la estimación de la orientación, este último proceso es muy complicado debido a diversos factores y a que es necesario integrar las mediciones de varios sensores, esto será explicado con detalles más adelante.

1.6 Objetivo de la tesis

Utilizar e implementar un método de filtrado para los sensores de orientación y navegación inercial, estos sensores consisten de dos tipos de sensores que son sensores de referencia y sensores inerciales, estos grupos de sensores serán los que ayuden a determinar la orientación del satélite HumSAT-México. Debido a que los costos de sensores de calificación espacial son demasiado elevados además de que esa clase de

sensores también son demasiado voluminosos es que se opta por la utilización de sensores de bajo costo.

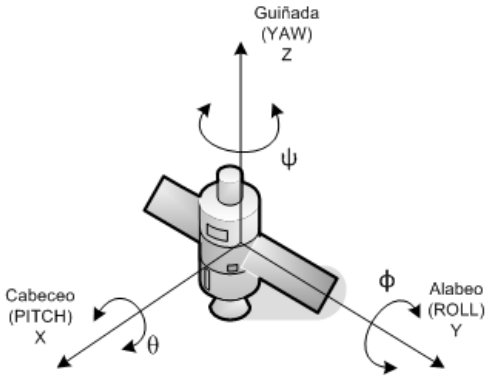


Figura 1.4
Ángulos de orientación.

Durante los últimos años se han hecho grandes adelantos en el desarrollo de las tecnologías MEMS (Micro Electro Mechanical Systems) que han permitido que sensores que antes eran difíciles de construir, voluminosos y costosos ahora se encuentren a un precio accesible, sean pequeños y su construcción sea más sencilla. En todo sensor siempre existe un factor de ruido inherente cuya naturaleza es aleatoria y se debe a fenómenos aleatorios presentes en el sensor, además de que debido a que son de fabricación MEM se vuelven muy susceptibles a la temperatura y al ruido eléctrico. Es por esto, que para tener un correcto desempeño en la medición de los ángulos de orientación del satélite (para realizar la tarea de la estimación de orientación) es necesario usar un método de filtrado que permita obtener las mediciones más adecuadas a partir de los sensores, así como para obtener mediciones que sean útiles dentro de un esquema de control, ya que los múltiples factores de ruido que se van sumando al momento de obtener una medición de la orientación hacen que esta sea demasiado inexacta y no sea adecuada para realizar un buen control.

En la implementación de un algoritmo de filtrado este debe de ejecutarse en tiempo real. El filtro que se persigue implementar en esta tesis todavía es una primera aproximación al filtro definitivo por lo que se usa una plataforma de desarrollo de gran capacidad para determinar cuáles son las demandas del filtro. Otro de los objetivos de la implementación es servir de punto de partida para determinar las características de un microcontrolador de menores dimensiones que pueda cumplir con las demandas de la estimación y el control con el menor consumo de potencia posible.

El último objetivo de esta tesis es comparar el desempeño de dos modelos de estimación de orientación usando técnicas distintas, por un lado usando la técnica de estimación de orientación de Inmersión e Invarianza propuesta en el Centro de Investigaciones y Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV) y la segunda que consistió en plantear un modelo a través del uso del filtro de Kalman Unscented que es un método de filtrado mejor validado y usado para una gran cantidad de tareas.

1.7 Definición del problema

Al poner en órbita a un satélite, este es lanzado y dejado dentro de un rango de altitud en el que debe operar. Bajo estas condiciones, el satélite queda totalmente perturbado, es decir, se le libera y deja en un estado totalmente alterado sin tener una posición estacionaria por lo que en principio debe de ejecutar una secuencia de operación a través de sus actuadores y sensores para fijar su orientación y posteriormente poder realizar tareas de estabilización y apuntamiento que requieran las cargas útiles dentro del satélite para realizar sus tareas , como pueden ser la percepción remota o comunicaciones directivas.

Para realizar tareas de apuntamiento satelital, este debe realizar un esquema de control. Dentro de este esquema de control es necesario tener un módulo que sea capaz de realizar la estimación de la orientación del satélite y realizar este cálculo para el sistema de estabilización que es muy demandante en cuanto a recursos de procesamiento. Para realizar la tarea de la estimación de orientación se requiere un grupo de sensores destinados para ello, entre mejor sea la calidad de los sensores la estimación será mejor e incluso la demanda del procesamiento se podría ver reducida, sin embargo, tener sensores de alta calidad conlleva un alto costo y espacio, algo que en el nanosatélite a desarrollar para la constelación HumSAT no es posible tener.

Actualmente, otros participantes dentro del proyecto ya han diseñado un esquema de control preliminar y también un modelo de estimación basado en el filtrado de Kalman Extendido, pero aun no se han implementado en hardware final. En la presente tesis se implementa el método de estimación de inmersión e invarianza y el filtro de Kalman Unscented como sustituto del filtro de Kalman Extendido debido a que la versión Unscented del filtro de Kalman tiene sustanciosas mejoras sobre el método Extendido sin una necesidad mayor de procesamiento.

Mediante las implementaciones a realizar del método de inmersión e invarianza y el filtro de Kalman Unscented es posible determinar las necesidades de procesamiento de ambos métodos y ver entonces cual resulta más conveniente implementar para su uso dentro del esquema de control de apuntamiento del Nanosatélite HumSAT México, y de igual forma permite determinar las características del microcontrolador más conveniente para emplear. Debido a que en este momento no se dispone de todos los sensores del Nanosatélite y a que previamente este método será utilizado dentro de una plataforma experimental de laboratorio, de momento no resultó necesario realizar 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 a que al hacer pruebas en tierra se supone que el campo magnético es fijo) debido a que por ahora se utilizará dentro de la mesa experimental de nuestro laboratorio.

Capítulo 2

Antecedentes del proyecto HumSAT-México

México es usuario de alta tecnología satelital comercial desde 1985, año en que contrató la compra del sistema de satélites Morelos, posteriormente adquirió los sistemas, Solidaridad, Satmex y ahora ha comprado tres nuevos satélites Geoestacionarios, correspondientes al sistema denominado MexSat. También como país hemos tenido experiencia en el ensamble, prueba y validación de satélites como los UNAMSAT. Adicionalmente, México ha desarrollado o participado en el desarrollo de proyectos satelitales experimentales, los cuales se describen en este capítulo, que han ayudado a obtener experiencia en el área y han motivado la participación en el proyecto HumSAT.

2.1 Proyecto SATEDU

SATEDU (Satélite Educativo) fue un proyecto que se empezó a desarrollar en el Instituto de Ingeniería de la UNAM en 2006 y en 2007 fue financiado por el Consejo Nacional de Ciencia y Tecnología (CONACYT) [3]. El proyecto consistió en diseñar, construir y validar una plataforma satelital orientada al entrenamiento de recursos humanos en el área satelital, figura 2.1.

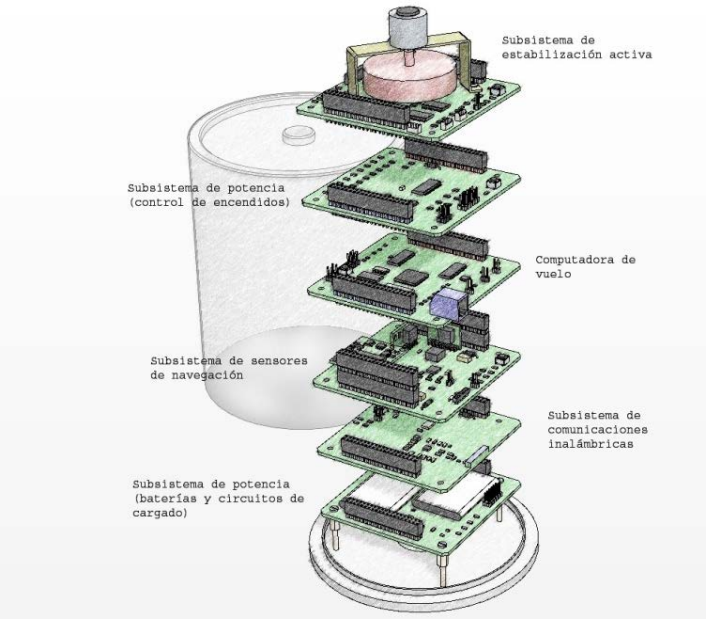


Figura 2. 1 Sistema SATEDU.

El sistema está constituido por los subsistemas que dan forma a un satélite pequeño real. De ellos, algunos tienen diferencias respecto a un satélite real, por ejemplo, el subsistema de comunicaciones que opera solo en pequeñas distancias, o en sus sensores de navegación inercial, que son muy económicos.

SATEDU se desarrolló como un sistema autómata distribuido, donde cada tarjeta del satélite tiene un microcontrolador. Tiene un tamaño de 10 X 10 X 20 cm y se comunica de manera inalámbrica con una PC a través de software desarrollado en Visual Basic. SATEDU tiene además una filosofía de funcionamiento cliente-servidor, en el que desde una laptop y software que interactúa con el satélite se encuentra el cliente o usuario que le indica al satélite las tareas que debe realizar y los subsistemas que debe monitorear.

El sistema SATEDU está constituido por los siguientes subsistemas:

- *Computadora de vuelo*: Este subsistema se encarga de coordinar a los demás subsistemas y distribuye la información que llega desde el software que funge como estación terrena.
- *Potencia*: Se encarga del encendido y apagado de todos los subsistemas de SATEDU, cuenta con varios reguladores de voltaje y baterías Li-Ion.
- *Comunicaciones*: Se encarga de recibir y enviar los datos ya sea de forma inalámbrica entre SATEDU y la PC o a través del bus de comunicaciones a la computadora de vuelo para el envío y recepción de comandos y telemetría.
- *Estabilización:* Se encarga de ejecutar las tareas de orientación a través del control de varios actuadores (rueda inercial y bobinas de torque magnético).
- *Sensores de orientación:* Tiene sensores para obtener la orientación de SATEDU y para realizar tareas de apuntamiento en conjunto con el subsistema de estabilización, entre ellos cuenta con 3 giróscopos, un acelerómetro triaxial y una brújula electrónica.

2.2 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 en el caso de satélites astronómicos o de satélites espías, los cuales pueden tener objetivos diferentes y específicos para su objeto de análisis [4].

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 la mesa suspendida en aire.

La mesa suspendida en aire, figura 2.2, es un sistema desarrollado para recrear ambientes de baja fricción y de baja gravedad, aunque es obvio que en tierra es muy difícil

recrear la gravedad baja, 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, y 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.

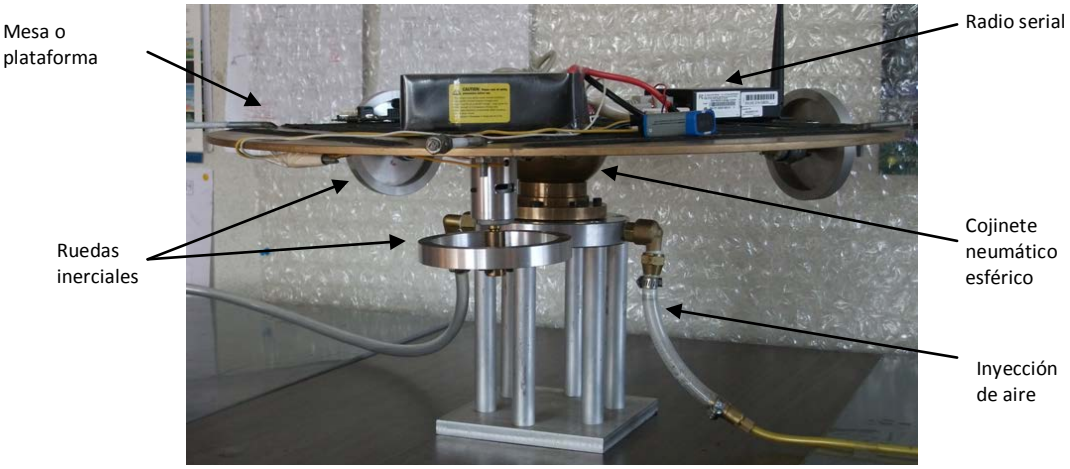


Figura 2. 2 Mesa suspendida en aire.

2.3 Subsistema de sensores de navegación inercial en SATEDU

Los sensores de estabilización en SATEDU permiten conocer la orientación del satélite. Las categorías de sensores de posición y orientación en un satélite se subdividen en 2 [4]:

- ***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 de inercia conocido hasta ahora 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 2.3.

De esa forma, se construyó un sistema que emula a un sistema de medición de posición que es similar a uno satelital. Para simularlo se construyó un sistema similar a una Unidad de Medición Inercial que es modesto y menos preciso que los equivalentes comerciales debido a la calidad de sus sensores.

Figura 2. 3 Tarjeta de sensores de navegación de SATEDU.

2.4 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.

2.5 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 HumSAT 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.

2.5.1 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 [6].

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 podrían ser usados 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 desaparece 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.

2.5.2 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[6][7].

Existen distintos tipos de giróscopos, los cuales se clasifican de acuerdo a su uso en los siguientes tipos, tabla 2.1 :

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 2. 1 Tipos de giróscopos.

Dentro de la clasificación de la tabla 2.1, cada tipo de giróscopo tiene una forma distinta de fabricación, hoy día al igual que los acelerómetros los giróscopos de fabricación MEM 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 [8].

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 2.2 [7].

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

Tabla 2. 2 Desempeño de giróscopos.

2.5.3 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[6].

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.

2.5.4 Sensor de sol

Los sensores de sol son una referencia más para la orientación de un satélite, actualmente existen 2 clases de sensores de sol, los analógicos y los digitales. Los primeros funcionan a través de parejas de sensores. Tomando al sol como una referencia puntual, la medida diferencial de la salida entre los sensores fotosensibles es una indicación de ángulo entre el sol y el elemento sobre el cual están montados los sensores [6].

Los sensores digitales miden la posición de la imagen del sol en el plano del sensor. Estos pueden usar patrones especiales de detectores térmicos en el plano del sensor o

pueden usar elementos del tipo CCD (Charge Coupled Device –dispositivo de carga acoplada), figura 2.4.

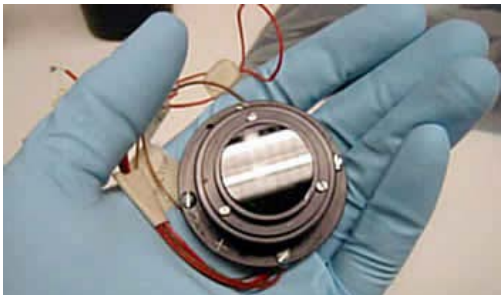


Figura 2. 4 Sensor biaxial de sol modelo 0.5 de Optical Energy Technologies.

2.5.5 Sensor de estrellas

En misiones de espacio profundo y en satélites que requieren apuntamiento muy fino, el control de apuntamiento se lleva a cabo por observación de las estrellas a través de pequeños telescopios ópticos. El apuntamiento se determina entonces por la orientación de un conjunto de ejes de referencia fijos al satélite y relativos a tres ejes mutuamente ortogonales que pueden ser definidos por el plano eclíptico y la línea del equinoccio vernal. Para este propósito, la declinación y correcta ascensión de guías de estrellas son programadas en una carta estelar contenida en la memoria de la computadora de vuelo.

Hasta hace algunos años pensar en integrar un sensor de estrellas a un nanosatélite era imposible debido a que eran instrumentos muy voluminosos y complejos, hoy día debido a la evolución tecnológica es posible construir sensores de estrellas lo suficientemente compactos para ser integrados en satélites pequeños, figura 2.5. Este instrumento es de gran utilidad en un esquema de control de apuntamiento ya que es de gran precisión[6].

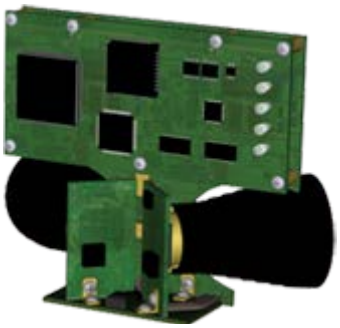


Figura 2. 5 Módulo de sensores de la compañía Onboard Systems con sensor de estrellas.

2.5.6 Receptor GPS

El sistema de posicionamiento global o GPS, es un sistema de navegación global que provee la localización de cualquier objeto en la Tierra o cercano a ella. Este sistema fue desarrollado en 1973 para superar las limitaciones de sistemas previos de navegación,

desarrollado por el departamento de defensa de los Estados Unidos de América, está completamente operativo desde 1994 y cuenta con 24 satélites en tres diferentes órbitas [9].

Un receptor GPS usado en Tierra no puede ser usado en el espacio primeramente por las protecciones militares que les establece el departamento de defensa de Estados Unidos, y secundariamente debido a las condiciones a las que estaría sometido, que van desde la radiación espacial y el medio ambiente agresivo hasta las características dinámicas que este puede soportar. Es por ello que si un GPS quiere ser utilizado en el espacio este tiene que ser un equipo autorizado internacionalmente y por tanto modificado de manera sustancial. Actualmente varios desarrolladores han modificado receptores GPS hasta volverlos útiles en el espacio como son el Centro de desarrollo espacial Alemán con un receptor llamado Phoenix[10] y Surrey Satellite Technology [11] con su receptor SGR-05U, en cada caso, ellos se aseguran de tramitar los permisos respectivos de uso orbital.

El uso de un receptor GPS, figura 2.6, es importante debido a que puede mejorar sustancialmente el desempeño de los algoritmos de estimación de orientación satelital en combinación con otros sensores.

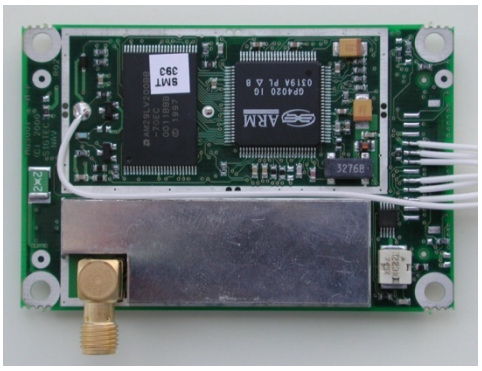
a)  b) 

Figura 2. 6 GPS para satélites pequeños, a) Phoenix b)SGR-05U.

Capítulo 3

Definiciones y notaciones de orientación

Desde que el hombre empezó a tener la necesidad de recorrer grandes distancias tuvo la necesidad de orientarse para ir de un lugar a otro, a partir de entonces empezó a idear formas e instrumentos que le permitieran realizar tal acción. Actualmente existen una gran cantidad de tecnologías y métodos sofisticados para obtener la orientación que van desde dispositivos móviles como celulares hasta sofisticados equipos de navegación que están integrados en casi toda clase de vehículos incluyendo los satélites. La tarea de la orientación de un objeto no es una tarea sencilla, y previo a disponer de una medición o control de ella fue necesario crear una serie de conceptos que ayudaran a expresarla de una manera matemática, como se describe en este capítulo.

3.1 Representación de la orientación de un satélite

Para estimar la orientación de un satélite en primer lugar se vuelve necesario saber varios fundamentos geométricos acerca de cómo es posible representar matemáticamente su orientación, a continuación se describen algunos de ellos.

3.2 Sistemas coordenados

Los sistemas coordenados son marcos de referencia que permiten ubicar un punto en el espacio, figura 3.1, en el caso de la estimación de orientación y el modelo de control de apuntamiento de un satélite es necesario el uso de varios sistemas coordenados para logar este objetivo. Esto se hace debido a que numérica y algebraicamente es más práctico trabajar con marcos de referencia locales que con un sistema de referencia universal. A continuación se muestran varios de los sistemas coordenados utilizados en el modelado de satélites [6] [12] [13].

Sistema de referencia Inercial con centro en la Tierra (S^I). En este sistema de referencia el origen del sistema coordenado está en el centro de la Tierra, en la literatura general se le conoce como ECI (Earth-Centered Inertial). Este es un marco de referencia inercial fijo que es necesario para que las leyes de Newton sean validas sobre otros sistemas de referencia móviles. Se le denota con la letra I, la Tierra gira alrededor del eje Z y el eje X apunta al equinoccio vernal (que es uno de los puntos generados por la intersección del plano

ecuatorial y el plano eclíptico del movimiento de traslación de la Tierra como se ve en la figura 3.1) el eje Y es ortogonal a esos dos ejes para completar el sistema.

Sistema de referencia fijo a la Tierra y con origen en el centro de la Tierra (S^E). En este sistema el origen está en el centro de la Tierra y comparte su origen y el eje Z con el sistema ECI, se denota con la letra *E.* El eje X de este sistema está determinado por la intersección del plano formado por el ecuador y el formado por el meridiano de Greenwich, el eje Y completa al sistema de Z y X de manera ortogonal. En la literatura se encuentra abreviado como ECEF (Earth-Centered Earth Fixed). Este sistema rota junto con la Tierra con una velocidad $\omega_E = 7.2921 \times 10^{-7}[rad/s]$ alrededor del eje Z, por lo que no es un sistema de referencia inercial.

Sistema de referencia orbital (S^O). Este sistema tiene su centro donde la nave espacial tiene su centro de masa también. El origen gira a una velocidad angular ω_O relativa al sistema de referencia ECI, los ejes están distribuidos de manera que el eje X es siempre tangencial a la curva orbital y el eje Z apunta al centro de la Tierra, finalmente el eje Y completa el sistema coordenado ortogonal. Es importante notar que esto solo aplica en una órbita circular, ya que en una órbita elíptica al cumplir con estas condiciones ya no se formaría un sistema ortogonal. Se representa con la letra *O.*

Sistema de referencia con centro en la Tierra (S^{OE}). Este sistema tiene su origen en el centro de la Tierra, el eje X apunta hacia el perigeo, el eje Y apunta hacia el eje semimenor y el eje Z completa al sistema de manera ortogonal. Este sistema se representa con las letras *OE.*

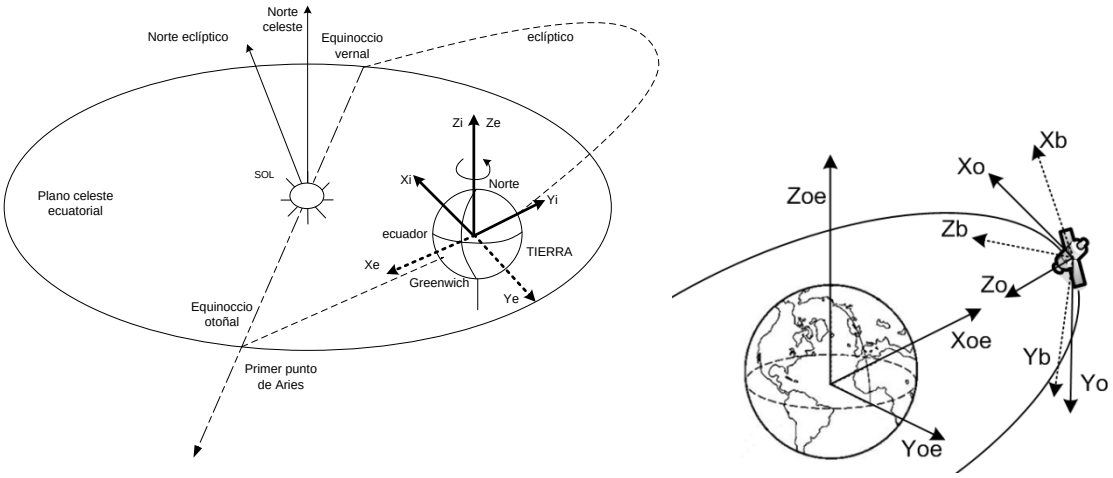


Figura 3. 1 Sistemas coordenados.

Sistema de referencia del satélite (S^B). Este sistema de referencia tiene su origen en el centro de masa del satélite, comparte su origen con el sistema de referencia orbital. Se representa con la letra *B*. La rotación entre el sistema orbital (S^O) y el sistema satelital (S^B) representa la orientación del satélite respecto a la Tierra. Los ejes dentro del satélite

son definidos localmente según convenga. Los sistemas coordenados S^B y S^O coincidirán cuando los ángulos de rotación del satélite sean 0°.

3.3 Matrices de rotación

Debido a que se utilizan varios sistemas coordenados de referencia para representar la posición y orientación relativas del satélite con respecto de varios objetos, se hace necesario el uso de la matriz de rotación también llamada matriz de cosenos directores [6] [14]. Las matrices de rotación son matrices que permiten transformar un vector de un sistema coordenado a otro diferente. De forma tal que si por ejemplo se tiene una matriz de rotación $\boldsymbol{R}_B^I$ que permite rotar un vector con coordenadas referidas al sistema de referencia del satélite $\boldsymbol{r}_B$ hacia el sistema *ECI,* se tiene que $\boldsymbol{r}_I = \boldsymbol{R}_B^I \cdot \boldsymbol{r}_B$. Algo extra que se puede realizar para evitar ambigüedades en las matrices de rotación es que al concatenar matrices de rotación se debe de llegar al mismo resultado, como es $\boldsymbol{r}_I = \boldsymbol{R}_B^I \cdot \boldsymbol{R}_I^B \cdot \boldsymbol{r}_I$ y se debe de llegar al mismo resultado por lo que se cumple que $\boldsymbol{R}_B^I \cdot \boldsymbol{R}_I^B = \boldsymbol{I}$ donde $\boldsymbol{I}$ es la matriz identidad, a su vez, esto lleva a que $\boldsymbol{R}_B^I = (\boldsymbol{R}_I^B)^{-1}$, adicionalmente hay que tener en cuenta que una matriz de rotación también es una matriz ortogonal por lo que $\boldsymbol{R}_B^I = (\boldsymbol{R}_I^B)^T$ además de que $\det(\boldsymbol{R}_B^I) = 1$.

3.4 Ángulos de Euler

Existen varias formas de representar la orientación de un satélite respecto a la Tierra, una de ellas son los ángulos de Euler, figura 3.2, los cuales son básicamente los ángulos de rotación con respecto a los ejes del sistema coordenado de referencia utilizado. La secuencia en la que se tienen que dar los ángulos de rotación en la forma que se expresa la orientación del objeto a través de este método de representación tiene que darse en un orden específico ya que el orden en que gira un objeto sobre los ejes del sistema coordenado no es conmutativo. Las formas de expresión de la orientación de un objeto consisten en combinaciones de giros sobre los ejes coordenados que pueden ser en el orden X-Y-Z, Y-X-Z, Z-X-Y o Z-Y-X. El esquema más utilizado en la aeronáutica es el X-Y-Z. Las rotaciones alrededor de cada eje están definidas de la siguiente manera[14]:

$$\boldsymbol{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \operatorname{sen}\phi \\ 0 & -\operatorname{sen}\phi & \cos\phi \end{bmatrix} \quad \boldsymbol{R}_y = \begin{bmatrix} \cos\theta & 0 & -\operatorname{sen}\theta \\ 0 & 1 & 0 \\ \operatorname{sen}\theta & 0 & \cos\theta \end{bmatrix}$$
$$\boldsymbol{R}_z = \begin{bmatrix} \cos\psi & \operatorname{sen}\psi & 0 \\ -\operatorname{sen}\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Estas son matrices de rotación, donde ϕ es el ángulo de rotación alrededor del eje x, también conocido como ángulo de alabeo (roll), θ es el ángulo de rotación alrededor del eje y, también conocido como ángulo de cabeceo (pitch) y ψ es el ángulo de rotación

alrededor del eje Z también conocido como ángulo de guiñada (yaw). Al multiplicarlas obtendremos la matriz de rotación con la que se obtiene la orientación del objeto. Esto es:

$$\boldsymbol{R} = \boldsymbol{R}_x \boldsymbol{R}_y \boldsymbol{R}_z \tag{3.2}$$

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\,\mathrm{sen}\psi & -\mathrm{sen}\theta \\ (\mathrm{sen}\phi\,\mathrm{sen}\theta\cos\psi - \cos\phi\,\mathrm{sen}\psi) & (\mathrm{sen}\phi\,\mathrm{sen}\theta\,\mathrm{sen}\psi + \cos\phi\cos\psi) & \mathrm{sen}\phi\cos\theta \\ (\cos\phi\,\mathrm{sen}\theta\cos\psi + \mathrm{sen}\phi\,\mathrm{sen}\psi) & (\cos\phi\,\mathrm{sen}\theta\,\mathrm{sen}\psi - \mathrm{sen}\phi\cos\psi) & \cos\phi\cos\theta \end{bmatrix} \tag{3.3}$$

Debido a que se guarda un orden en las rotaciones, los ángulos de Euler no son únicos para representar cierta orientación, además de esto hay algunos ángulos de Euler que no pueden determinarse del todo a través de la matriz de rotación. En tales casos, se dice que la representación de Euler se vuelve "singular" y la vuelve inútil. Un ejemplo de esto es cuando en ángulo de cabeceo (θ) es igual a ±90° y los ángulos de alabeo (ϕ) y guiñada (ψ) se vuelven indeterminados. Esta representación es muy utilizada para aeronaves, ya que es muy raro encontrar este ángulo sobre ese eje para representar su orientación. Sin embargo en casos como misiles, naves de combate o satélites, este ángulo sí se puede presentar. No obstante, es posible cambiar el orden de las rotaciones para evitar este ángulo singular sobre ese eje, pero al hacerlo se encontraría ahora sobre otra orientación. Esta deficiencia en particular lleva a buscar nuevas representaciones que no posean singularidades. Los ángulos de Euler son mutuamente independientes por lo que este conjunto representa un conjunto mínimo para la representación de la orientación, pero está restringido a aplicaciones donde las singularidades no se presentan.

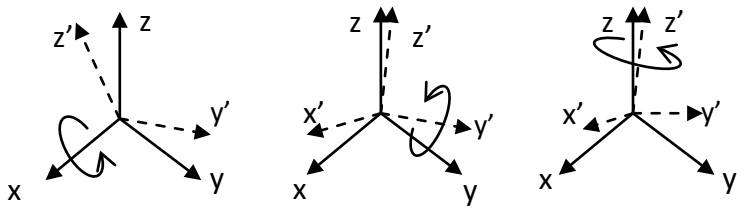


Figura 3. 2 Rotación de ángulos de Euler.

3.5 Cuaterniones

Los cuaterniones fueron descubiertos por William Rowan Hamilton en el siglo XIX [15], buscaba formas de extender los números complejos a mayores dimensiones o también conocidos como números hipercomplejos, algo que muchos matemáticos habían buscado durante años. Hasta que en 1894 obtuvo números complejos de rango 4 a los que nombró *cuaterniones*, figura 3.3. La parte crucial de este invento fue la regla fijada por Hamilton, donde:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3.4}$$

Los cuaterniones fueron un ente matemático despreciado por muchos años, no fue sino hasta 1985 cuando Ken Shoemake presentó la idea de interpolar rotaciones a través de cuaterniones para gráficos por computadora, que tiempo después impulsarían de sobremanera las industrias de videojuegos en 3D, la robótica y la aeronáutica[16]. Un cuaternión es una forma de representar un cuadro de referencia con respecto de otro, esta herramienta hace lo mismo que las matrices de rotación pero con una diferencia muy importante, solo usa 4 términos para representar esto, en lugar de los 9 términos que requiere la matriz de rotación. Esto tiene beneficios a nivel de cómputo ya que para calcular las rotaciones se puede hacer con menos operaciones que con las matrices de rotación. Otro beneficio muy importante para trabajar con esta representación es que no presenta singularidades. Un cuaternión se representa de la siguiente manera:

$$\boldsymbol{q} = \eta + i \cdot \epsilon_1 \ + j \cdot \epsilon_2 \ + k \cdot \epsilon_3 \tag{3.5}$$

Posee un elemento escalar η y 3 terminos que pueden ser vistos como un vector $\boldsymbol{\epsilon}$. Usualmente la forma en la que se expresan los cuaterniones es como una matriz columna con la siguiente forma

$$\boldsymbol{q} = \begin{bmatrix} \eta \\ \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta}{2} \\ a_1 \text{sen}\frac{\theta}{2} \\ a_2 \text{sen}\frac{\theta}{2} \\ a_3 \text{sen}\frac{\theta}{2} \end{bmatrix} \tag{3.6}$$

En esta tesis solo se utilizaron cuaterniones unitarios[17], por razones de cómputo ya que al multiplicar dos cuaterniones unitarios se obtiene otro cuaternión unitario, que para normalizarlo evita una división. Debido a que existen errores de precisión al usar cómputo de punto fijo se recomienda renormalizar los cuaterniones antes de aplicar rotaciones.

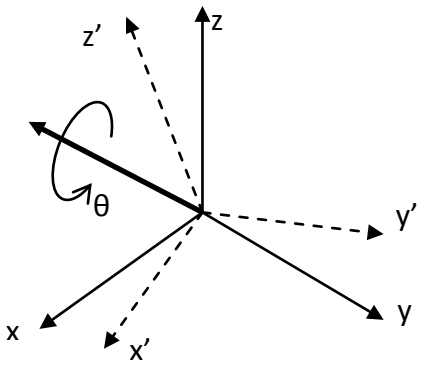


Figura 3. 3 Diagrama de cuaternión.

Algunas de las operaciones y propiedades consideradas para los cuaterniones son las siguientes

$$\begin{aligned} &\|\boldsymbol{q}\| = 1 \\ &\boldsymbol{\epsilon}^T\boldsymbol{\epsilon} + \eta^2 = 1 \\ &\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 + \eta^2 = 1 \end{aligned} \tag{3.7}$$

Al rotar vectores, debido a que estos solo poseen tres componentes, y la rotación que se le quiere dar a un vector está dada por el cuaternión $\boldsymbol{q}_1 = \begin{bmatrix}\eta_1 \\ \boldsymbol{\epsilon}_1\end{bmatrix}$ que posee cuatro componentes, es necesario agregar una cuarta componente al vector que se desea rotar a través del cuaternión por lo que a un vector v se le puede considerar como un cuaternión con $\eta = 0$, y la operación a realizar para obtener un vector u rotado a través del cuaternión q_1 es como sigue

$$\begin{bmatrix}0 \\ \boldsymbol{u}\end{bmatrix} = \boldsymbol{q}_1 \otimes \begin{bmatrix}0 \\ \boldsymbol{v}\end{bmatrix} \otimes \boldsymbol{q}_1 \tag{3.8}$$

donde $\boldsymbol{q} = \eta - i \cdot \epsilon_1 \; - j \cdot \epsilon_2 \; - k \cdot \epsilon_3$ y el producto de cuaterniones está definido como

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = \begin{bmatrix}\eta_1 \\ \boldsymbol{\epsilon}_1\end{bmatrix} \otimes \begin{bmatrix}\eta_2 \\ \boldsymbol{\epsilon}_2\end{bmatrix} = \begin{bmatrix}\eta_1\eta_2 - \boldsymbol{\epsilon}_1^T\boldsymbol{\epsilon}_2 \\ \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + S(\boldsymbol{\epsilon}_1)\boldsymbol{\epsilon}_2\end{bmatrix} \tag{3.9}$$

donde $S(\boldsymbol{\epsilon}_1) = \begin{bmatrix} 0 & -\epsilon_{13} & \epsilon_{12} \\ \epsilon_{13} & 0 & -\epsilon_{11} \\ -\epsilon_{12} & \epsilon_{11} & 0 \end{bmatrix}$ y representa una matriz antisimétrica del vector $\boldsymbol{\epsilon}_1$.

Si por otra parte solo se utilizan cuaterniones en las representaciones de la orientación, entonces al tener una orientación descrita por un cuaternión $\boldsymbol{q}_1$ y una rotación representada por el cuaternión $\boldsymbol{q}_2$, la nueva orientación de $\boldsymbol{q}_1$ estará representada por

$$\boldsymbol{q} = \boldsymbol{q}_2 \otimes \boldsymbol{q}_1 \tag{3.10}$$

Por último, dado que las matrices de rotación y los cuaterniones son herramientas encargadas de la misma tarea, expresar la orientación de un cuerpo rígido. Una expresión muy útil es la de la matriz de rotación en términos del cuaternión que es como sigue:

$$\mathbf{R} = \mathbf{I} + 2\eta S(\boldsymbol{\epsilon}) + 2S^2(\boldsymbol{\epsilon}) \tag{3.11}$$

3.6 Transformación entre diferentes sistemas coordenados

Debido a que en la representación de la orientación de cuerpos rígidos lo recomendable es utilizar más de un sistema coordenado (debido a que esto facilita la representación de la orientación de un cuerpo rígido con respecto a otro), en el caso de un satélite del cual

se desea controlar y estimar su orientación con respecto a la Tierra es necesario utilizar varios sistemas coordenados como se vio en la sección anterior.

Ahora, un punto representado con respecto a cierto sistema coordenado tiene cierta representación numérica pero al cambiar de sistema coordenado este mismo punto cambia de representación. A continuación se mencionan algunas de las transformaciones más relevantes [18].

Transformación de S^{OE} al sistema S^I y S^E: Esta permite el conocimiento de la posición del satélite respecto a los sistemas coordenados S^I y S^E.

$$\begin{aligned} \boldsymbol{R}_{OE}^{I} &= \boldsymbol{R}_Z(\Omega)\boldsymbol{R}_x(i)\boldsymbol{R}_Z(\omega) \\ \boldsymbol{R}_{OE}^{E} &= \boldsymbol{R}_Z(\Omega+\theta)\boldsymbol{R}_x(i)\boldsymbol{R}_Z(\omega) \end{aligned} \tag{3.12}$$

donde Ω es la ascensión derecha del nodo ascendente, *i* es la inclinación de la órbita respecto a la línea de nodos, ω es el argumento del perigeo y θ es la ascensión del meridiano cero.

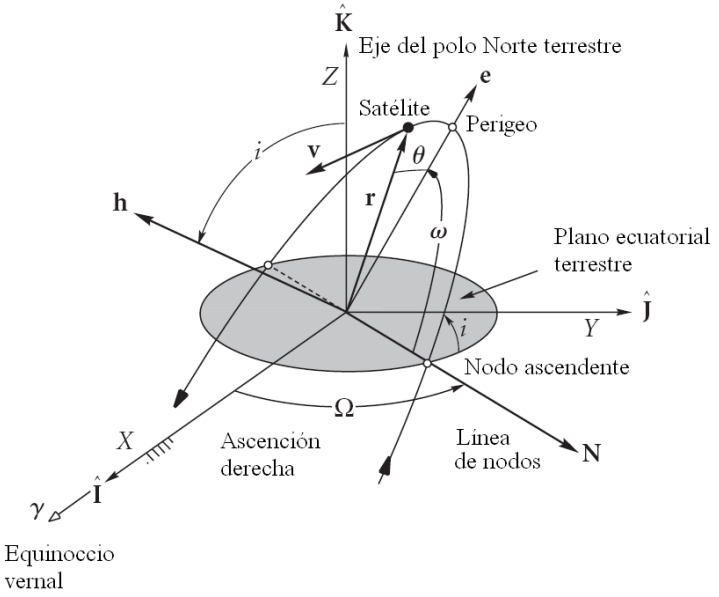


Figura 3. 4 Parámetros de los sistemas coordenados.

Transformación del sistema S^E al sistema S^I: Esta rotación define el movimiento de la rotación de la Tierra con respecto a un sistema de referencia similar, ambos tienen en común el eje Z pero el sistema S^E rota sobre ese eje con la velocidad de rotación de la Tierra y se puede expresar como $\delta = \Omega_e t$ donde Ω_e es la velocidad angular de la Tierra y t es el tiempo que ha transcurrido desde que S^E y S^I estuvieron alineados, se expresa como:

$$\boldsymbol{R}_z = \begin{bmatrix} \cos(\delta) & -\text{sen}(\delta) & 0 \\ \text{sen}(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

Transformación del sistema S^O al sistema S^{OE}: Esta rotación está definida principalmente por la traslación del satélite sobre su órbita alrededor de la Tierra:

$$\boldsymbol{R}_O^{OE} = \boldsymbol{R}_z(\chi)\boldsymbol{R}_x(\xi)\boldsymbol{R}_y(\kappa) \tag{3.14}$$

Donde $\chi = -90°, \xi = 90°$ y $\kappa = \kappa_o + \omega_o t$, κ_o corresponde al inicio del despliegue del satélite respecto de la órbita y ω_o es la velocidad angular de desplazamiento del satélite en su órbita. Con estos parámetros, Z_o del sistema S^O queda apuntando al centro de la órbita y X_o es tangente a la órbita.

Transformación del sistema S^O al sistema S^B: Esta rotación depende de la orientación del satélite respecto a la Tierra; obteniendo esta matriz de rotación se puede determinar la orientación del satélite a partir de (3.11) por medio de:

$$\boldsymbol{R}_B^O = (\boldsymbol{R}_O^B)^T = \boldsymbol{I} + 2\eta S(\boldsymbol{\epsilon}) + 2S^2(\boldsymbol{\epsilon}) \tag{3.15}$$

Transformación entre el sistema S^O y el sistema S^I: Esta rotación es una de las más importantes ya que en base a esta rotación se puede realizar la tarea del control de apuntamiento de un satélite que orbita alrededor de la Tierra. Como ya se ha visto en la sección 3.1.1 el sistema coordenado S^O es un sistema móvil con su origen en el centro de masa del satélite y el eje Z apuntando al centro de la Tierra y el sistema S^I es un sistema coordenado fijo con origen en el centro de la Tierra, al ser dos sistemas coordenados diferentes es importante realizar un modelo de la relación de rotación de un sistema con respecto a otro a través del tiempo ya que en base a esto se realiza la tarea de orientación. La tarea de orientación más sencilla es hacer que una cara del satélite apunte siempre hacia la Tierra por lo que simplemente es necesario hacer que el sistema S^O, que se encuentra con el eje Z siempre orientando hacia el centro de la Tierra, coincida con el sistema coordenado S^B.

Previo a calcular el cuaternión necesario para relacionar los sistemas S^O y S^I se necesitan varias rotaciones:

1. Transformación del sistema S^{OE} al sistema S^I

   $$\boldsymbol{R}_{OE}^I = \boldsymbol{R}_z(\Omega)\boldsymbol{R}_x(i)\boldsymbol{R}_z(\omega) \tag{3.16}$$

   donde en las matrices de rotación $\boldsymbol{R}_z$ y $\boldsymbol{R}_x$ se basan en las ecuaciones de (3.1) y los valores de los parámetros Ω, i y ω son de acuerdo a una órbita definida.

2. Transformación del sistema S^O al sistema S^{OE}

   $$\boldsymbol{R}_O^{OE} = \boldsymbol{R}_z(\chi)\boldsymbol{R}_x(\xi)\,\boldsymbol{R}_y(\alpha) \tag{3.17}$$

donde χ y ξ se definen primero, por ahora α se omite debido a que este varía con el tiempo y provocará que exista una singularidad en un ángulo, por lo que por ahora queda:

$$\boldsymbol{R}_O^{OE} = \boldsymbol{R}_z(\chi)\boldsymbol{R}_x(\xi) \tag{3.18}$$

Con las matrices de rotación anteriores se tiene que el sistema S^O puede expresarse en términos del sistema S^I con la siguiente expresión

$$\boldsymbol{R}_I^{OE} = (\boldsymbol{R}_{OE}^I)^T \qquad \boldsymbol{R}_{OE}^O = (\boldsymbol{R}_O^{OE})^T \tag{3.19}$$

$$\boldsymbol{R}_I^O = \boldsymbol{R}_{OE}^O \boldsymbol{R}_I^{OE} \tag{3.20}$$

Se transforma la matriz de rotación $\boldsymbol{R}_I^O$ a un cuaternión unitario definido como $\boldsymbol{q}_i$ y falta agregar la rotación que se omitió en (3.17) debido a una singularidad, esta rotación se daba en el eje y, de modo que el cuaternión de conversión resulta del producto de estos dos, entonces:

$$\boldsymbol{q}_d = \boldsymbol{q}_i \times \boldsymbol{q}_o = \begin{bmatrix} \eta_i\eta_o - \boldsymbol{\epsilon}_i^T\boldsymbol{\epsilon}_o \\ \eta_i\boldsymbol{\epsilon}_o + \eta_o\boldsymbol{\epsilon}_i + S(\boldsymbol{\epsilon}_i)\boldsymbol{\epsilon}_o \end{bmatrix} \tag{3.21}$$

donde $\boldsymbol{q}_d$ es el cuaternión que relaciona los sistemas coordenados S^O y S^I. Esta relación es muy importante al momento de generar resultados visuales en un simulador.

3.7 Cinemática

Los cuaterniones se eligieron como herramienta de expresión de la orientación debido a que estos no presentan singularidades, además de que reducen el procesamiento al momento de expresar rotaciones sucesivas. El cuaternión unitario a usarse se nombra $\mathbf{q}_B^I$ que expresa al cuerpo B en el sistema coordenado S^I, con base en ello se obtiene un modelo cinemático a partir de cuaterniones. Dado que el cuaternión es una forma de representación de la orientación de un cuerpo, su derivada será relacionada a la velocidad angular de un cuadro de referencia con respecto a otro, estas representaciones son[6]:

$$\begin{matrix} \mathbf{q}_{ab}, \boldsymbol{\omega}_a \ , \ \mathbf{q}_{ba}, \boldsymbol{\omega}_b \\ \mathbf{q}_{ba}, \boldsymbol{\omega}_a \ , \ \mathbf{q}_{ab}, \boldsymbol{\omega}_b \end{matrix} \tag{3.22}$$

La derivada de un cuaternión se define como

$$\dot{\mathbf{q}} = \lim_{\Delta t \to 0} \frac{\Delta \boldsymbol{q}}{\Delta t} = \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t + \Delta t) - \boldsymbol{q}(t)}{\Delta t} \tag{3.23}$$

Como se vio en (3.10) al multiplicar 2 cuaterniones se obtiene un nuevo cuaternión, en este caso la rotación puede expresarse como el cuaternión que representa la posición inicial y un cuaternión diferencial que representa el movimiento como:

$$\boldsymbol{q}(t+\Delta t) = \Delta\boldsymbol{q} \otimes \boldsymbol{q} \tag{3.24}$$

Ahora se define que $\boldsymbol{\omega} = \boldsymbol{a}\omega$ donde $\boldsymbol{a}$ es un vector unitario para la velocidad angular ω. Tomando la representación $\mathbf{q}_{ab}, \omega_a$, entonces $\Delta\boldsymbol{q}$ se convierte cuando se toma la expresión original del cuaternión:

$$\Delta\boldsymbol{q} = \begin{bmatrix} \cos\left(\frac{\omega\Delta t}{2}\right) \\ \boldsymbol{a}\,\mathrm{sen}\left(\frac{\omega\Delta t}{2}\right) \end{bmatrix} \tag{3.25}$$

Sustituyendo (3.24) en la definición de la derivada, se tiene:

$$\dot{\mathbf{q}} = \lim_{\Delta t\to 0}\frac{\Delta\boldsymbol{q}}{\Delta t} = \lim_{\Delta t\to 0}\frac{(\Delta\boldsymbol{q}-\boldsymbol{e})\boldsymbol{q}}{\Delta t} \tag{3.26}$$

Donde $\boldsymbol{e}$ es el cuaternión identidad (1,0,0,0).

Ahora expresando $\Delta\boldsymbol{q}$ como una matriz de transición de estados para un cuaternión se tiene que:

$$\Delta\boldsymbol{q} = \begin{bmatrix} \cos\left(\frac{\omega\Delta t}{2}\right) & -\boldsymbol{a}^T\,\mathrm{sen}\left(\frac{\omega\Delta t}{2}\right) \\ \boldsymbol{a}\,\mathrm{sen}\left(\frac{\omega\Delta t}{2}\right) & \mathbf{I}_{3\times 3}\cos\left(\frac{\omega\Delta t}{2}\right) - \mathrm{S}(\boldsymbol{a})\mathrm{sen}\left(\frac{\omega\Delta t}{2}\right) \end{bmatrix} \tag{3.27}$$

Substituyendo en (3.26) se tiene que:

$$\dot{\boldsymbol{q}} = \lim_{\Delta t\to 0}\frac{1}{\Delta t}\begin{bmatrix} \cos\left(\frac{\omega\Delta t}{2}\right)-1 & -\boldsymbol{a}^T\,\mathrm{sen}\left(\frac{\omega\Delta t}{2}\right) \\ \boldsymbol{a}\,\mathrm{sen}\left(\frac{\omega\Delta t}{2}\right) & \mathbf{I}_{3\times 3}\left(\cos\left(\frac{\omega\Delta t}{2}\right)-1\right) - \mathrm{S}(\boldsymbol{a})\mathrm{sen}\left(\frac{\omega\Delta t}{2}\right) \end{bmatrix}\boldsymbol{q} \tag{3.28}$$

Resolviendo el límite, y expresando ω como un vector, tenemos:

$$\dot{\boldsymbol{q}} = \left(\frac{\omega}{2}\right)\begin{bmatrix} 0 & -\boldsymbol{a}^T \\ \boldsymbol{a} & -S(\boldsymbol{a}) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -S(\boldsymbol{\omega}) \end{bmatrix}\boldsymbol{q} \tag{3.29}$$

Capítulo 4

Estimación de la orientación

Una vez analizados los conceptos sobre la orientación de un objeto en el espacio es necesario crear un esquema de medición basado en los conceptos anteriores para posteriormente también generar un esquema de control de orientación o apuntamiento. Aunque en un principio se generaron algoritmos para determinar la orientación, estos se veían influenciados por el ruido de los sensores produciendo ruido en la medición final de la orientación. Posteriormente con nuevas herramientas matemáticas como los cuaterniones, la generación de esquemas de estimación y la evolución de los sistemas de procesamiento fue posible incorporar sistemas de orientación y navegación en vehículos como satélites, robots y aeronaves. A continuación se presentan los esquemas de estimación usados en el desarrollo de la presente tesis.

4.1 Algoritmos de estimación determinísticos

Es posible decir que los algoritmos de estimación determinísticos son completamente predictivos mientras se conozcan sus entradas. En el problema de determinación de la orientación tridimensional, esta es conocida siempre y cuando sea posible tener al menos 2 vectores de observación. Al hacer una comparación entre los dos vectores de observación con sus respectivos vectores de referencia, teóricamente es posible obtener la información de la orientación tridimensional de un cuerpo.

A continuación se mencionan 2 métodos determinísticos, el método TRIAD y el método de Gauss-Newton.

4.1.1 Método TRIAD

El método TRIAD (TRIaxial Attitude Determination) es un método algebraico que sirve para encontrar la orientación de un objeto a partir de dos vectores de medición y dos vectores de referencia, para lo cual utiliza dos sistemas coordenados. Este método fue desarrollado por Harold D. Black en 1964 y fue publicado en el diario del Instituto Americano de Aeronáutica y Astronáutica. Este método fue ideado a partir de que fueron montado los primeros paneles solares en forma triaxial, de manera que a partir de la posición del sol y teniendo otro vector de referencia que era el que apuntaba a la Tierra era posible saber la orientación del satélite.

El procedimiento para obtener la orientación del objeto consiste en encontrar una serie de vectores ortogonales entre sí, $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ y $\boldsymbol{r}_3$ en el sistema coordenado donde se efectuó la medición, tales que:

$$\boldsymbol{r}_1 = \frac{\boldsymbol{u}_M}{\|\boldsymbol{u}_M\|} \qquad \boldsymbol{r}_2 = \frac{\boldsymbol{r}_1 \times \boldsymbol{v}_M}{\|\boldsymbol{r}_1 \times \boldsymbol{v}_M\|} \qquad \boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2 \qquad (4.1)$$

De igual forma busca encontrar una serie de vectores de ortogonales entre sí, $\boldsymbol{s}_1$, $\boldsymbol{s}_2$ y $\boldsymbol{s}_3$ en el sistema coordenado de referencia, tales que:

$$\boldsymbol{s}_1 = \frac{\boldsymbol{u}_R}{\|\boldsymbol{u}_R\|} \qquad \boldsymbol{s}_2 = \frac{\boldsymbol{s}_1 \times \boldsymbol{v}_R}{\|\boldsymbol{s}_1 \times \boldsymbol{v}_R\|} \qquad \boldsymbol{s}_3 = \boldsymbol{s}_1 \times \boldsymbol{s}_2 \qquad (4.2)$$

Finalmente para obtener la matriz de cosenos directores que relaciona ambos sistemas coordenados, se emplea:

$$\boldsymbol{A}_M^R = \boldsymbol{r}_1 \cdot \boldsymbol{s}_1^T + \boldsymbol{r}_2 \cdot \boldsymbol{s}_2^T + \boldsymbol{r}_3 \cdot \boldsymbol{s}_3^T \qquad (4.3)$$

Este método se eligió para trabajar en conjunto con el algoritmo del filtro de Kalman para hacer la estimación de la orientación, esto se explicará más adelante.

4.1.2 Método Gauss-Newton

Este método se basa en un criterio de error a través de mínimos cuadrados, donde el problema es típicamente representado como:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}\varepsilon(\boldsymbol{x})^T \varepsilon(\boldsymbol{x}) \qquad (4.4)$$

donde $\boldsymbol{x}$ es un vector n-dimensional, $f(\boldsymbol{x})$ es la función "costo" y $\varepsilon(\boldsymbol{x})$ es la función de error.

A diferencia del método TRIAD se pueden integrar más de dos vectores de medición y referencia.

La función del error $\boldsymbol{S}^k$ está definida por:

$$\boldsymbol{S}^k = \boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon} = \left(\boldsymbol{y}_c^i - \mathbf{M}\boldsymbol{y}_m^b\right)^T \left(\boldsymbol{y}_c^i - \mathbf{M}\boldsymbol{y}_m^b\right) \qquad (4.5)$$

donde $\boldsymbol{y}_c^i$ contiene los n-vectores de referencia en un sistema coordenado determinado, $\boldsymbol{y}_m^b$ contiene los n-vectores de mediciones expresados en el sistema coordenado donde se efectuó la medición. $\mathbf{M}$ es definido como:

$$\mathbf{M} = \begin{bmatrix} \boldsymbol{R}_b^e(\boldsymbol{q}_k) & 0 \\ 0 & \boldsymbol{R}_b^e(\boldsymbol{q}_k) \end{bmatrix} \tag{4.6}$$

donde $\boldsymbol{R}_b^e(\boldsymbol{q}_k)$ es la matriz de rotación que relaciona ambos sistemas coordenados, expresado en términos del cuaternión $\boldsymbol{q}_k$.

Para encontrar el cuaternión $\boldsymbol{q}_k$ que relaciona los vectores de medición y de referencia, se realiza un proceso iterativo de minimización de la función de error. Este proceso se describe en las siguientes ecuaciones:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k - [J^T(\boldsymbol{q}_k)\, J(\boldsymbol{q}_k)]^{-1} J^T(\boldsymbol{q}_k)\varepsilon(\boldsymbol{q}_k) \tag{4.7}$$

Donde $J(\boldsymbol{q}_k)$ es el Jacobiano de la función $\mathbf{M}$ definido por:

$$J = \left[\left(\frac{\partial M}{\partial q_1}\right) \left(\frac{\partial M}{\partial q_2}\right) \left(\frac{\partial M}{\partial q_3}\right) \left(\frac{\partial M}{\partial q_4}\right) \right] \boldsymbol{y}_m^b \tag{4.8}$$

La función de error se usa como factor de tolerancia para frenar las iteraciones necesarias para encontrar el cuaternión $\boldsymbol{q}_k$ que normalmente toma entre tres o cinco iteraciones para encontrarlo. Es recomendable definir un valor máximo de iteraciones que pudieran deberse a un valor grande de la función de error [19][20].

4.2 Algoritmos de estimación analíticos

Estos algoritmos de estimación a diferencia de los métodos determinísticos para realizar la estimación, sí toman en cuenta los factores de ruido que afectan a los procesos de medición y la dinámica del sistema que se está tratando.

A continuación se describen 2 métodos que se utilizaron en la presente tesis para implementarlos en un DSP. El primer método, llamado de Inmersión e Invarianza, fue desarrollado para esta clase de aplicación en el CINVESTAV (Centro de Investigaciones y Estudios Avanzados del Instituto Politécnico Nacional) para su aplicación en robots móviles y el segundo es una de las tantas variaciones del filtro de Kalman llamada SR-UKF (*Square Root Unscented Kalman Filter*).

Dentro de estos métodos de estimación analíticos la tarea más difícil que presentan es el ajuste de los parámetros de diseño que definen el ruido de las mediciones y el ruido presente en la dinámica del proceso a estimar; la combinación de éstos determina la rapidez de convergencia del método de estimación. Para realizar los ajustes, regularmente no existe un método bien fundamentado para lograrlo, en este caso lo que se hizo fue hacer una serie de muestreos previos para ajustar los parámetros a partir de un dato sobre ruido que ofrecen las hojas de especificaciones de los sensores.

4.2.1 Inmersión e Invarianza

Esta metodología se basa en las nociones de inmersión de sistemas e invarianza de campos (manifold invariance) (Apéndice A), las cuales son herramientas clásicas de la teoría de la regulación no lineal y control geométrico no lineal y es llamada inmersión e invarianza (immersion and invariance) [21].

El enfoque básico de la metodología de Inmersión e Invarianza (I&I), es lograr el objetivo de control por inmersión de la dinámica de la planta dentro de un sistema objetivo (posiblemente de bajo orden) que capture el comportamiento deseado del sistema.

Las principales características del método de I&I son:

- Reduce el problema de diseño del controlador del sistema a otros subproblemas, los cuales, en ciertos casos, son más fáciles de resolver.
- Difiere de la mayoría de las metodologías de diseño de controladores existentes debido a que en principio no requiere el conocimiento de la función de Lyapunov.
- Se ajusta bien a situaciones donde un controlador estabilizante para un modelo nominal de orden reducido es conocido, y se busca hacerlo más robusto con respecto a dinámicas de más alto orden. Esto se logra diseñando una ley de control que "sumerge" asintóticamente la dinámica del sistema completo en un sistema de orden reducido.
- En problemas de control adaptable el método radica en establecer esquemas que contrarrestan el efecto de parámetros desconocidos adoptando una perspectiva más robusta. Esto en contraste con algunos de los diseños adaptables existentes que tratan estos términos como perturbaciones a ser rechazadas. El método de I&I no invoca cierta equivalencia, ni requiere una parametrización lineal. Además, provee un procedimiento para agregar un término cruzado entre los parámetros estimados y los estados de la planta para la función de Lyapunov.
- Provee un marco natural para la formulación y solución de problemas asociados al diseño de observadores y la robustez en la salida de problemas de estabilización con retroalimentación, con observación de estados y estimación de parámetros tratados en conjunto.
- Permite formular y resolver el problema de diseñar controladores proporcional-integrales para una clase, en sistemas de bajo orden.

Planteamiento

Este método en un principio fue diseñado para robots móviles [22], que se desplazan a través de terrenos irregulares, por lo que la estimación primordial a realizar era la de los ángulos de cabeceo y alabeo. Para ello se considera que se tiene una Unidad de Medición Inercial capaz de entregar 3 clases de mediciones: aceleraciones lineales, velocidades

angulares y fluctuaciones del campo magnético, teniendo una señal de cada tipo por cada eje, lo que hace un total de 9, estas se obtienen con acelerómetros, giróscopos y magnetómetros respectivamente.

Es posible modelar los sensores, el sistema de giróscopos que miden la velocidad angular Ω y además un término variante en el tiempo μ_Ω, esto es:

$$\Omega_G = \Omega + \mu_\Omega \tag{4.9}$$

donde Ω_G es la salida del giróscopo.

Los acelerómetros miden la suma de las fuerzas inerciales más la gravedad, por unidad de masa, esto es:

$$\mathbf{f}_s^i = \dot{\mathbf{v}}_{CM}^i - \boldsymbol{a}_g^i \tag{4.10}$$

donde $\boldsymbol{a}_g^i = 1/m\mathbf{f}_g^i$ y la velocidad del centro de masa es $\dot{\mathbf{v}}_{CM}^i = [u\ v\ w]^T$. Este vector de fuerza también es conocido como fuerza específica.

Como los acelerómetros se encuentran fijos al cuerpo rígido su salida está dada por:

$$\boldsymbol{a}_A^b = \mathbf{R}\mathbf{f}_s^i + \boldsymbol{\mu_A} = \mathbf{R}\dot{\mathbf{v}}_{CM}^i - \mathbf{R}\boldsymbol{a}_g^i + \boldsymbol{\mu_A} \tag{4.11}$$

donde μ_A es un término desconocido de desviación variante en el tiempo.

Para la primera parte del estimador es de primordial importancia estimar los ángulos de cabeceo (θ) y alabeo (ϕ) (pitch y roll). La tercera columna de la matriz *R* como se ve en la ecuación (3.3) es independiente del ángulo de guiñada (ψ) (yaw) por lo que al estimar esa columna será posible obtener los ángulos θ y ϕ.

Para plantear el problema de la estimación de la rotación se usa la cinemática de rotación que está dada por:

$$\dot{\boldsymbol{R}} = S(\boldsymbol{\Omega})\boldsymbol{R} \tag{4.12}$$

y esta puede reescribirse en forma de columnas como:

$$\dot{\boldsymbol{r}}_i = S(\boldsymbol{\Omega})\boldsymbol{r}_i\ , \qquad i = 1,2,3 \tag{4.13}$$

y también se tiene que $\boldsymbol{R}\boldsymbol{a}_g^i = \boldsymbol{r}_3$.

En el planteamiento se obtiene el sistema:

$$\begin{aligned} \dot{\boldsymbol{r}}_3 &= S(\Omega_G)\boldsymbol{r}_3 + \boldsymbol{q} \\ \boldsymbol{a}_A^b &= -\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^b + \boldsymbol{\mu_A} \end{aligned} \tag{4.14}$$

donde *q* representa una desviación variante y lenta de la dinámica real $\boldsymbol{r}_3$.

Hay que notar que las desviaciones $\boldsymbol{q}$ y $\boldsymbol{\mu_A}$ corresponden a los ruidos del proceso y la medición respectivamente, estas deben de variar lentamente con el tiempo, y se deben considerar perfiles bajos de aceleración que no cambien bruscamente ($\dot{\mathbf{v}}_{CM}^{b} \approx 0$).

Considerando la extensión dinámica $\dot{\boldsymbol{\eta}} = -\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^{b} + \boldsymbol{\mu_A}$, entonces el sistema (4.14) puede expresarse como:

$$\begin{aligned} \dot{\boldsymbol{r}}_3 &= S(\Omega_G)\boldsymbol{r}_3 + \boldsymbol{q} \\ \dot{\boldsymbol{\eta}} &= -\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^{b} + \boldsymbol{\mu_A} \\ \boldsymbol{y} &= \boldsymbol{\eta} \end{aligned} \tag{4.15}$$

donde $\boldsymbol{\eta}$ es la nueva salida medible.

Además, se definen los errores de estimación como:

$$\begin{aligned} \boldsymbol{z}_1 &= \boldsymbol{r}_3 - \hat{\boldsymbol{r}}_3 + \beta_1(\boldsymbol{\eta}) \\ \boldsymbol{z}_2 &= \bar{q}\tanh[\hat{\boldsymbol{q}} + \beta_2(\boldsymbol{\eta})] - \boldsymbol{q} \\ \boldsymbol{z}_3 &= \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \beta_3(\boldsymbol{\eta})] - \boldsymbol{\mu_A} \end{aligned} \tag{4.16}$$

donde $\beta_i(\boldsymbol{\eta}), i = 1,2,3$ son funciones a ser definidas, $\bar{q}$ y $\bar{\mu}_A$ son las cotas superiores conocidas de $\mathbf{q}$ y $\boldsymbol{\mu_A}$ respectivamente. Hay que notar que si los errores de estimación en (4.16) convergen a cero se tiene:

$$\begin{aligned} &\lim_{t\to\infty}\left(\hat{\boldsymbol{r}}_3 - \beta_1(\boldsymbol{\eta})\right) = \boldsymbol{r}_3 \\ &\lim_{t\to\infty}(\bar{q}\tanh[\hat{\boldsymbol{q}} + \beta_2(\boldsymbol{\eta})]) = \boldsymbol{q} \\ &\lim_{t\to\infty}(\bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \beta_3(\boldsymbol{\eta})]) = \boldsymbol{\mu_A} \end{aligned}$$

Tomando la derivada de los errores de estimación a lo largo del sistema dinámico (4.15) se obtiene,

$$\begin{aligned} \dot{\boldsymbol{z}}_1 &= \dot{\boldsymbol{r}}_3 - \dot{\hat{\boldsymbol{r}}}_3 + \frac{\partial\beta_1}{\partial\eta}\left(-\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^{b} + \boldsymbol{\mu_A}\right) \\ \dot{\boldsymbol{z}}_2 &= \bar{q}\,\Phi_{\hat{\mathrm{q}}}\left[\dot{\hat{\boldsymbol{q}}} + \frac{\partial\beta_2}{\partial\eta}\left(-\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^{b} + \boldsymbol{\mu_A}\right)\right] - \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{z}}_3 &= \bar{\mu}_A\,\Phi_{\hat{\mu}_\mathrm{A}}\left[\dot{\hat{\boldsymbol{\mu}}}_{\boldsymbol{A}} + \frac{\partial\beta_3}{\partial\eta}\left(-\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^{b} + \boldsymbol{\mu_A}\right)\right] - \dot{\boldsymbol{\mu}}_{\boldsymbol{A}} \end{aligned} \tag{4.17}$$

Donde,

$$\Phi_{\hat{\mathrm{q}}}(\hat{q},\eta) = \begin{bmatrix} \varphi_1(\hat{q},\eta) & 0 & 0 \\ 0 & \varphi_2(\hat{q},\eta) & 0 \\ 0 & 0 & \varphi_3(\hat{q},\eta) \end{bmatrix},$$

y

$$\Phi_{\hat{\mu}}(\hat{\mu}_\mathrm{A},\eta) = \begin{bmatrix} \varphi_1(\hat{\mu}_\mathrm{A},\eta) & 0 & 0 \\ 0 & \varphi_2(\hat{\mu}_\mathrm{A},\eta) & 0 \\ 0 & 0 & \varphi_3(\hat{\mu}_\mathrm{A},\eta) \end{bmatrix},$$

con $\varphi_i(\hat{q},\eta) = \tanh[\hat{\boldsymbol{q}}_{\mathrm{i}} + \beta_{2i}(\eta)]^2$, $\varphi_i(\hat{\mu}_{\mathrm{A}},\eta) = \tanh[\hat{\boldsymbol{\mu}}_{\mathbf{i}} + \beta_{2i}(\eta)]^2, i = 1,2,3$. Ahora considerando que $\boldsymbol{q}$ y $\boldsymbol{\mu_A}$ son desviaciones que varían lentamente en el tiempo, se tiene que, $\dot{\boldsymbol{q}} = \dot{\boldsymbol{\mu}}_{\boldsymbol{A}} \approx 0$, como consecuencia:

$$\begin{aligned}
\dot{\boldsymbol{z}}_1 &= S(\Omega_G)\boldsymbol{r}_3 + \boldsymbol{q} - \dot{\hat{\boldsymbol{r}}}_3 + \frac{\partial \beta_1}{\partial \eta}\left(-\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^b + \boldsymbol{\mu_A}\right) \\
\dot{\boldsymbol{z}}_2 &= \bar{q}\,\Phi_{\hat{\mathrm{q}}}\left[\dot{\hat{\boldsymbol{q}}} + \frac{\partial \beta_2}{\partial \eta}\left(-\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^b + \boldsymbol{\mu_A}\right)\right] \\
\dot{\boldsymbol{z}}_3 &= \bar{\mu}_A\,\Phi_{\hat{\mu}_{\mathrm{A}}}\left[\dot{\hat{\boldsymbol{\mu}}}_{\boldsymbol{A}} + \frac{\partial \beta_3}{\partial \eta}\left(-\boldsymbol{r}_3 + \dot{\mathbf{v}}_{CM}^b + \boldsymbol{\mu_A}\right)\right]
\end{aligned} \tag{4.18}$$

Por otra parte de (4.15) se tiene que,

$$\begin{aligned}
\boldsymbol{r}_3 &= \boldsymbol{z}_1 + \hat{\boldsymbol{r}}_3 - \beta_1(\boldsymbol{\eta}) \\
\boldsymbol{q} &= \bar{q}\tanh[\hat{\boldsymbol{q}} + \beta_2(\boldsymbol{\eta})] - \boldsymbol{z}_2 \\
\boldsymbol{\mu_A} &= \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \beta_3(\boldsymbol{\eta})] - \boldsymbol{z}_3
\end{aligned} \tag{4.19}$$

Así reemplazando (4.17) en (4.18) se tiene,

$$\begin{aligned}
\dot{\boldsymbol{z}}_1 &= S(\Omega_G)(\boldsymbol{z}_1 + \hat{\boldsymbol{r}}_3 - \boldsymbol{\beta}_1) - \boldsymbol{z}_2 + \bar{q}\tanh[\hat{\boldsymbol{q}} + \boldsymbol{\beta}_2] - \dot{\hat{\boldsymbol{r}}}_3 \\
&\quad + \frac{\partial \beta_1}{\partial \eta}\left(-\boldsymbol{z}_1 - \hat{\boldsymbol{r}}_3 + \boldsymbol{\beta}_1 + \dot{\mathbf{v}}_{CM}^b - \boldsymbol{z}_3 + \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \boldsymbol{\beta}_3]\right) \\
\dot{\boldsymbol{z}}_2 &= \bar{q}\,\Phi_{\hat{\mathrm{q}}}\left[\dot{\hat{\boldsymbol{q}}} + \frac{\partial \beta_2}{\partial \eta}\left(-\boldsymbol{z}_1 - \hat{\boldsymbol{r}}_3 + \boldsymbol{\beta}_1 + \dot{\mathbf{v}}_{CM}^b - \boldsymbol{z}_3 + \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \boldsymbol{\beta}_3]\right)\right] \\
\dot{\boldsymbol{z}}_3 &= \bar{\mu}_A\,\Phi_{\hat{\mu}_{\mathrm{A}}}\left[\dot{\hat{\boldsymbol{\mu}}}_{\boldsymbol{A}} + \frac{\partial \beta_3}{\partial \eta}\left(-\boldsymbol{z}_1 - \hat{\boldsymbol{r}}_3 + \boldsymbol{\beta}_1 + \dot{\mathbf{v}}_{CM}^b - \boldsymbol{z}_3 + \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \boldsymbol{\beta}_3]\right)\right]
\end{aligned} \tag{4.20}$$

Por último definiendo $\dot{\hat{r}}_3$ en términos de las variables medibles se tiene,

$$\begin{aligned}
\dot{\hat{\boldsymbol{r}}}_3 &= S(\Omega_G)(\hat{\boldsymbol{r}}_3 - \boldsymbol{\beta}_1) + \bar{q}\tanh[\hat{\boldsymbol{q}} + \boldsymbol{\beta}_2] \\
&\quad + \frac{\partial \beta_1}{\partial \eta}(-\hat{\boldsymbol{r}}_3 + \boldsymbol{\beta}_1 + \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \boldsymbol{\beta}_3]) \\
\dot{\hat{\boldsymbol{q}}} &= -\frac{\partial \beta_2}{\partial \eta}(-\hat{\boldsymbol{r}}_3 + \boldsymbol{\beta}_1 + \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \boldsymbol{\beta}_3]) \\
\dot{\hat{\boldsymbol{\mu}}}_{\boldsymbol{A}} &= -\frac{\partial \beta_3}{\partial \eta}(-\hat{\boldsymbol{r}}_3 + \boldsymbol{\beta}_1 + \bar{\mu}_A \tanh[\hat{\boldsymbol{\mu}}_{\boldsymbol{A}} + \boldsymbol{\beta}_3])
\end{aligned} \tag{4.21}$$

donde $\bar{\mu}_A$ y $\bar{q}$ son constantes, $\boldsymbol{\beta}_1$, $\boldsymbol{\beta}_2$ y $\boldsymbol{\beta}_3$ están en función de las mediciones tomadas del acelerómetro y las constantes γ_1, γ_2 y γ_3 como

$$\begin{aligned}
\boldsymbol{\beta}_1 &= \gamma_1 \boldsymbol{\eta}, \\
\boldsymbol{\beta}_2 &= \gamma_2 \boldsymbol{\eta}, \\
\boldsymbol{\beta}_3 &= \gamma_3 \boldsymbol{\eta}
\end{aligned}$$

donde $\boldsymbol{\eta}$, se obtiene mediante la integración de las medidas de los acelerómetros, es decir $\boldsymbol{\eta} = \int \dot{\boldsymbol{\eta}} \cdot \mathrm{dt}$, donde $\dot{\boldsymbol{\eta}} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$. Finalmente las derivadas parciales $\frac{\partial \beta_1}{\partial \eta}, \frac{\partial \beta_2}{\partial \eta}$ y $\frac{\partial \beta_3}{\partial \eta}$ se obtienen mediante la forma $\frac{\partial \beta_i}{\partial \eta} = diag\{\gamma_i\},\ \gamma_i > 0,\ i = 1,2,3.$

La obtención de los ángulos se hace a través de la primera ecuación (4.16) y las ecuaciones (4.21) al hacerlo de manera recursiva e integrando $\hat{\dot{\boldsymbol{r}}}_3$. Al obtener el vector $\boldsymbol{r}_3$ a partir de la forma genérica de la matriz de rotación $\mathbf{R}$ se obtienen los ángulos θ y ϕ.

Para estimar el ángulo ψ (yaw) se utiliza nuevamente la técnica de I&I pero se propone un estimador que utiliza los ángulos θ y ϕ previamente calculados y las mediciones del magnetómetro.

Para obtener el ángulo de guiñada (ψ) se propone:

$$\psi_m = \arctan\left[\frac{m_z \mathrm{sen}\phi - m_y \cos\phi}{m_x \cos\theta + m_y \mathrm{sen}\theta \mathrm{sen}\phi + m_z \cos\phi \mathrm{sen}\theta}\right] \tag{4.22}$$

donde m_x, m_y y m_z son las componentes de los magnetómetros ubicados en los ejes XYZ o bien de las mediciones tomadas por un dispositivo triaxial; θ y ϕ fueron obtenidos en el proceso de estimación anterior. Si bien es posible obtener el último ángulo de orientación de manera directa a través de (4.22), también estaría afectado por los ruidos en cada eje del propio magnetómetro, entregando así una señal ruidosa en el ángulo de guiñada. Para evitar el ruido de los magnetómetros y siguiendo los pasos del anterior proceso de estimación se llega a:

$$\begin{aligned}
\dot{\hat{\psi}} &= \frac{\mathrm{sen}\phi}{\cos\theta} q + \frac{\cos\phi}{\cos\theta} r - \bar{q}_\psi \tanh(\hat{q}_b + \beta_2) + \frac{\partial \beta_1}{\partial \eta}\left(\hat{\psi} - \beta_1 - \bar{v}_b \tanh(\hat{v} + \beta_3)\right) \\
\dot{\hat{q}}_b &= -\frac{\partial \beta_2}{\partial \eta}\left(\hat{\psi} - \beta_1 - \bar{v}_b \tanh(\hat{v} + \beta_3)\right) \\
\dot{\hat{v}} &= -\frac{\partial \beta_3}{\partial \eta}\left(\hat{\psi} - \beta_1 - \bar{v}_b \tanh(\hat{v} + \beta_3)\right)
\end{aligned} \tag{4.23}$$

donde $\bar{q}_\psi$ y $\bar{v}_b$ son constantes; β_1, β_2 y β_3 dependen del cálculo de ψ_m hecho a través de la ecuación (4.22) a partir de los ángulos θ y ϕ calculados anteriormente, y también dependen de las constantes a_1, a_2 y a_3. La obtención de β_1, β_2 y β_3 se dan como:

$$\begin{aligned}
\beta_1 &= a_1 \eta_1, \\
\beta_2 &= a_2 \eta_2, \\
\beta_3 &= a_3 \eta_3
\end{aligned}$$

donde $\eta = \int \psi_m \cdot \mathrm{dt}$, y finalmente las derivadas parciales $\frac{\partial \beta_1}{\partial \eta}, \frac{\partial \beta_2}{\partial \eta}$ y $\frac{\partial \beta_3}{\partial \eta}$ son de la forma $\frac{\partial \beta_i}{\partial \eta} = \boldsymbol{a}_i,\ i = 1,2,3$. La determinación del ángulo de guiñada se realiza igualmente a través de una de las funciones de error que se definen para llegar a (4.23), [22]:

$$\psi = z_1 + \hat{\psi} - \beta_1$$

donde $z_1 = 0$; y $\hat{\psi}$ y β_1 se obtienen a partir de las ecuaciones en (4.23).

4.2.2 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 donde presentaba un método nuevo para separar señales aleatorias y definidas de ruido aleatorio. Este método vino a resolver varios problemas que anteriormente eran intratables usando los métodos de Norbert Wiener, ya que no se adecuaban bien a formas discretas ni eran fácilmente extensibles a problemas que comprendían múltiples entradas y múltiples salidas.

El filtro de Kalman es una técnica muy usada para calcular los estados óptimos estimados de un sistema dinámico. Estas estimaciones son óptimas de tal forma que los errores de estimación se minimizan en el sentido de mínimos cuadrados. Este filtro fue muy bien recibido entre los ingenieros para resolver problemas en el área de navegación, entre otros problemas como: radar, visión computacional o en economía para la estimación de modelos macroeconómicos. Los enormes avances que se empezaban a dar en el área de la computación contribuyeron enormemente a la implementación de este método para varias aplicaciones en tiempo real[23].

En principio R. E. Kalman describió esta nueva aproximación del problema de filtrado y predicción de manera discreta, proponiendo una solución recursiva. En el filtro de Kalman es necesario recurrir a dos modelos, el modelo del proceso, que describe como cambia un sistema 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 agrega un modelo de los ruidos del proceso y el ruido de la medición. 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.

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{F}\boldsymbol{x} + \boldsymbol{w} \\ \boldsymbol{z} &= \mathbf{H}\boldsymbol{x} + \boldsymbol{v} \end{aligned} \tag{4.24}$$

Donde $\boldsymbol{x}$ es el vector de estado, cada variable que compone este vector describe un estado del sistema, $\mathbf{F}$ es la matriz del sistema y describe la forma en que el sistema varía

con el tiempo y otros parámetros, $\boldsymbol{w}$ 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 $\boldsymbol{z}$ es el vector de medición, cada variable representa una cantidad medible por los sensores, $\mathbf{H}$ es la matriz de medición que representa la relación entre las cantidades medibles y los estados del sistema, y $\boldsymbol{v}$ 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:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{\Phi}_k \boldsymbol{x}_k + \boldsymbol{w}_k \\ \boldsymbol{z}_k &= \mathbf{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \end{aligned} \tag{4.25}$$

donde $\boldsymbol{x}_k$ y $\boldsymbol{z}_k$ son los vectores de estado y medición en el tiempo t_k, $\boldsymbol{\Phi}_k$ es la matriz de transición que describe como el sistema cambia en el tiempo t_{k+1} de los anteriores estados del sistema, $\mathbf{H}_k$ es la matriz de medición y por último, $\boldsymbol{w}_k$ y $\boldsymbol{v}_k$ son los procesos de ruido, estos últimos deben de ser descritos como procesos estocásticos a través de dos matrices de covarianza $\boldsymbol{Q}_k$ y $\boldsymbol{R}_k$ indicando el grado de aleatoriedad del ruido[24][25].

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

1. Se obtienen los modelos del proceso del sistema y el proceso de medición como en (4.25).
2. Previo a empezar el proceso de iteraciones, se debe definir un vector de estado inicial $\boldsymbol{x}_0$ y una matriz de error de covarianza inicial $\mathbf{P}_0$.
3. Los estados estimados $\hat{\boldsymbol{x}}_k$ (^ denota que corresponde a una estimación) y la matriz de error de covarianzas $\mathbf{P}_k$ junto con $\boldsymbol{\Phi}_k$ y $\mathbf{Q}_k$ que son usados para realizar las predicciones del vector de estado del sistema $\hat{\boldsymbol{x}}_{k+1}^-$ para el siguiente escalón de tiempo y también para la predicción de la matriz de covarianzas $\mathbf{P}_{k+1}^-$, (¯ 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.

$$\begin{aligned} \hat{\boldsymbol{x}}_{k+1}^- &= \boldsymbol{\Phi}_k \hat{\boldsymbol{x}}_k \\ \mathbf{P}_{k+1}^- &= \boldsymbol{\Phi}_k \mathbf{P}_k \boldsymbol{\Phi}_k^{\mathrm{T}} + \mathbf{Q}_k \end{aligned} \tag{4.26}$$

4. Se usa $\mathbf{P}_k^-$, $\mathbf{H}_k$ y $\mathbf{R}_k$ para calcular la ganancia del filtro $\mathbf{K}_k$.

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k)^{-1} \tag{4.27}$$

5. Se obtiene un nuevo conjunto de mediciones y junto con $\mathbf{K}_k$, $\hat{\boldsymbol{x}}_k^-$ y $\mathbf{H}_k$ se calcula el vector $\hat{\boldsymbol{x}}_k$.

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \mathbf{K}_k (\boldsymbol{z}_k - \mathbf{H}_k \hat{\boldsymbol{x}}_k^-) \tag{4.28}$$

6. De las matrices $\mathbf{K}_k$, $\mathbf{I}$ y $\mathbf{H}_k$ se calcula la matriz de estimación de covarianza del error $\mathbf{P}_k$, y se repite este proceso desde el punto 3, todo el proceso se ilustra en la figura 4.1 en forma de diagrama de flujo.

Cabe recalcar que este proceso del filtro de Kalman es aplicable solo sobre sistemas lineales. 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 utilizado ampliamente por muchas décadas en una gran

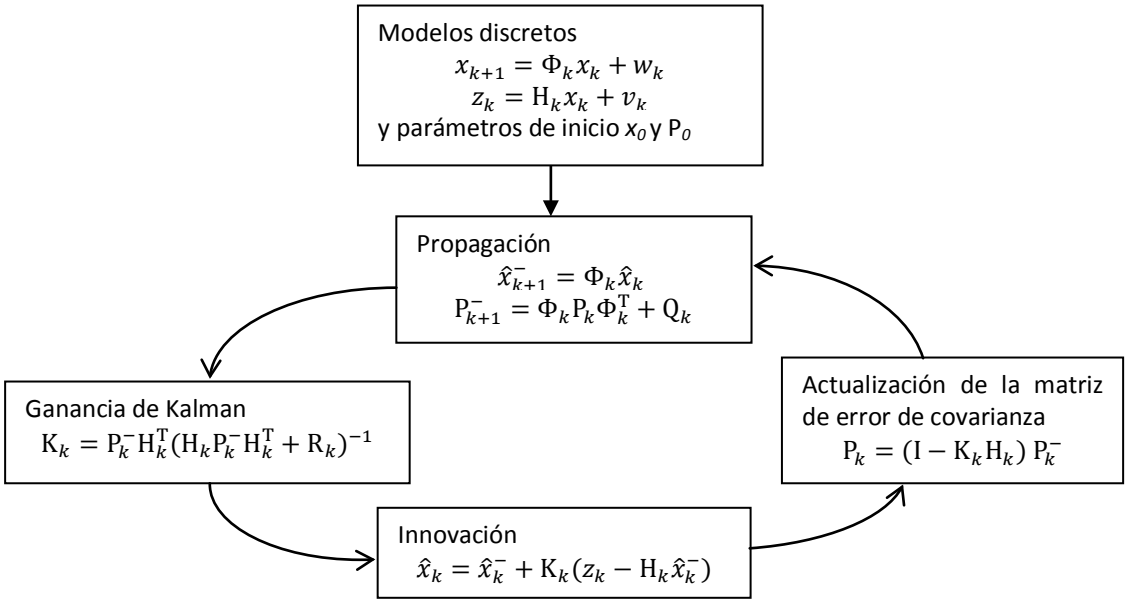


Figura 4. 1 Filtro de Kalman discreto.

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 las transformaciones lineales que se presentan en el filtro de Kalman por matrices Jacobianas.

EKF tiene varias ventajas y por mucho tiempo ha sido un método con un buen desempeño en diversas aplicaciones, sin embargo dentro de sus limitaciones están:

1. 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.

2. La linealización del proceso es posible solo si existe la matriz Jacobiana. Y no suele ser siempre el caso porque aunque exista la matriz el sistema puede tener

descontinuidades, puede haber un cambio abrupto de parámetros y otros factores que afecten de manera importante al filtro.

3. 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.

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

4.3 Filtro Kalman Unscented

El filtro de Kalman Unscented (UKF) fue propuesto por Simon J. Julier y Jeffrey K. Uhlmann en 1997, [25][26], quienes proponen 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 linealización en el EKF, ofreciendo un mecanismo más directo y explícito para transformar la información de la media y la covarianza, figura 4.2, 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.

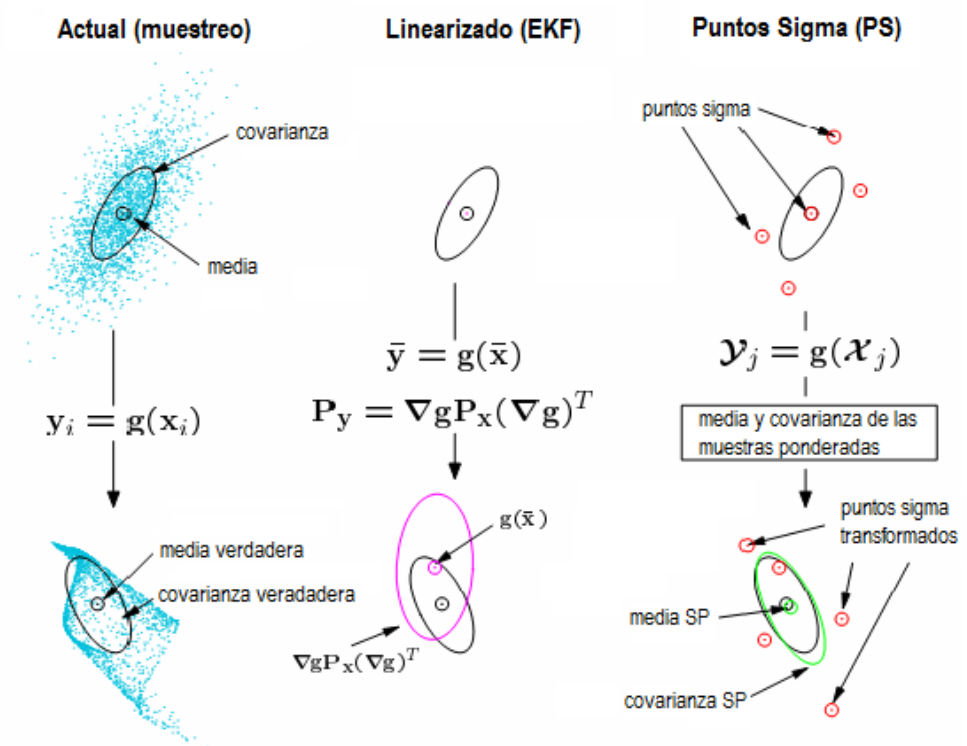


Figura 4. 2 Comparativa de la propagación de la media y la covarianza usando distintos métodos.

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 se calcula 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 se eligen 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.

1. 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 utilizarse para calcular las cantidades predichas como necesarias para cualquier transformación.
2. El costo computacional del algoritmo es del mismo orden de magnitud que el del EKF.
3. 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.
4. 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 resulta necesario obtener los puntos sigma, así como utilizar de manera directa las ecuaciones no lineales del sistema para la predicción de los estados y las mediciones así como de las matrices de covarianzas asociadas. Se siguen los siguientes pasos:

1. Se inicializa el UKF con $\boldsymbol{x}_0$ y $\mathbf{P}_0$, como ya se hacia en el método anterior.
2. Se calcula la predicción de los estados $\boldsymbol{x}^-$.
   a. Se definen los *2L* puntos sigma $\boldsymbol{\mathcal{X}}_{k-1}$

$$\boldsymbol{\mathcal{X}}_{k-1} = \left[\hat{\boldsymbol{x}}_{k-1} \quad \hat{\boldsymbol{x}}_{k-1} + \gamma\sqrt{\mathbf{P}_{k-1}} \quad \hat{\boldsymbol{x}}_{k-1} - \gamma\sqrt{\mathbf{P}_{k-1}}\right] \tag{4.29}$$

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

   b. Se usan los puntos sigma $\boldsymbol{\mathcal{X}}_{k-1}$ y la ecuación no lineal del proceso para transformar a los *2L* puntos obtenidos

$$\boldsymbol{\mathcal{X}}^*_{k|k-1} = \mathbf{F}[\boldsymbol{\mathcal{X}}_{k-1}, \mathbf{u}_{k-1}] \tag{4.30}$$

c. Se calcula el vector de predicciones del estado

$$\hat{\boldsymbol{x}}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \boldsymbol{\mathcal{X}}_{i,k|k-1}^* \tag{4.31}$$

donde W_i es un conjunto de pesos escalares definidos como $W_0^{(m)} = \frac{\lambda}{L+\lambda}$, $W_0^{(c)} = \frac{\lambda}{(L+\lambda)+(1-\alpha^2+\beta)}$, $W_i^{(m)} = W_i^{(c)} = \frac{1}{2(L+\lambda)}, i = 1,\ldots 2L$, $\lambda = \alpha^2(L+\kappa) - L$ y $\gamma = \sqrt{(L+\lambda)}$, son parámetros escalares, L corresponde a la dimensión de las variables de estado y α determina la dispersión de los puntos sigma alrededor de $\hat{x}$ y está ubicado entre $10^{-4} \leq \alpha \leq 1$. β se usa para incorporar el conocimiento previo de la distribución de $\boldsymbol{x}$ (para el caso de distribuciones gaussianas $\beta = 2$ es óptimo).

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

$$\mathbf{P}_k^- = \sum_{i=0}^{2L} W_i^{(c)} [\boldsymbol{\mathcal{X}}_{i,k|k-1}^* - \hat{\boldsymbol{x}}_k^-][\boldsymbol{\mathcal{X}}_{i,k|k-1}^* - \hat{\boldsymbol{x}}_k^-]^T + \mathbf{R}^{\mathrm{v}} \tag{4.32}$$

donde $\mathbf{R}^{\mathrm{v}}$ es la matriz de covarianza del ruido en el proceso.

3. Se calcula la predicción de las mediciones y las matrices de covarianza asociadas por medio de:

a. Usando $\boldsymbol{\mathcal{X}}_{k|k-1}$ para encontrar los 2L puntos sigma para las mediciones como:

$$\begin{gathered} \boldsymbol{\mathcal{X}}_{k|k-1} = \left[\hat{\boldsymbol{x}}_k^- \quad \hat{\boldsymbol{x}}_k^- + \gamma\sqrt{\mathbf{P}_k^-} \quad \hat{\boldsymbol{x}}_k^- - \gamma\sqrt{\mathbf{P}_k^-}\right] \\ \boldsymbol{\mathcal{Y}}_{k|k-1} = \mathbf{H}[\boldsymbol{\mathcal{X}}_{k|k-1}] \end{gathered} \tag{4.33}$$

b. Se usa $\boldsymbol{\mathcal{Y}}_{k|k-1}$ para calcular predicciones de las mediciones $\hat{\boldsymbol{y}}_k^-$:

$$\hat{\boldsymbol{y}}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \boldsymbol{\mathcal{Y}}_{i,k|k-1} \tag{4.34}$$

c. Se usa $\boldsymbol{\mathcal{Y}}_{k|k-1}$ y $\hat{\boldsymbol{y}}_k^-$ para calcular la matriz de predicción de la covarianza del error del modelo de medición como:

$$\mathbf{P}_{\tilde{y}_k\tilde{y}_k} = \sum_{i=0}^{2L} W_i^{(c)} [\boldsymbol{\mathcal{Y}}_{i,k|k-1} - \hat{\boldsymbol{y}}_k^-][\boldsymbol{\mathcal{Y}}_{i,k|k-1} - \hat{\boldsymbol{y}}_k^-]^T + \mathbf{R}^{\mathrm{n}} \tag{4.35}$$

donde $\mathbf{R}^{\mathrm{n}}$ es la matriz de covarianza del ruido de medición.

d. Se usan $\boldsymbol{\mathcal{Y}}_{k|k-1}, \hat{\boldsymbol{y}}_k^-, \boldsymbol{\mathcal{X}}_{k|k-1}$ y $\hat{\boldsymbol{x}}_k^-$, para calcular la matriz de predicción de la covarianza del error cruzado como:

$$\mathbf{P}_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} [\boldsymbol{\mathcal{X}}_{i,k|k-1} - \hat{\boldsymbol{x}}_k^-][\boldsymbol{\mathcal{Y}}_{i,k|k-1} - \hat{\boldsymbol{y}}_k^-]^T \tag{4.36}$$

4. Cuando se obtiene un nuevo conjunto de mediciones $\boldsymbol{y}_k$, ya es posible obtener la estimación de los estados $\hat{\boldsymbol{x}}_k$ mediante:

a. $\mathbf{P}_{\tilde{y}_k\tilde{y}_k}$ y $\mathbf{P}_{x_k y_k}$ para calcular la matriz de ganancias:

$$\mathbf{K}_k = \mathbf{P}_{x_k y_k} \mathbf{P}_{\tilde{y}_k\tilde{y}_k}^{-1} \tag{4.37}$$

b. $\hat{\boldsymbol{x}}_k^-$, $\mathbf{K}_k$, $\boldsymbol{y}_k$ y $\hat{\boldsymbol{y}}_k^-$ como sigue:

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \mathbf{K}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k^-) \tag{4.38}$$

c. $\mathbf{P}_k^-$, $\mathbf{P}_{\tilde{y}_k\tilde{y}_k}$ y $\mathbf{K}_k$ para calcular la estimación de la matriz de covarianzas del error $\mathbf{P}_k$ como sigue:

$$\mathbf{P}_k = \mathbf{P}_k^- - \mathbf{K}_k\mathbf{P}_{\tilde{y}_k\tilde{y}_k}\mathbf{K}_k{}^T \tag{4.39}$$

5. Este proceso se repite desde el paso 2 hasta el 4.

Debe señalarse que dentro del proceso de estimación se agrega un nuevo tipo de operación que consiste en sumar un vector columna a una matriz ($\mathbf{A} \pm \boldsymbol{u}$), 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 implementarlo presenta el problema de cómo realizar una raíz cuadrada de una matriz cuadrada, la cual 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. Es por ello que se creo una variante de este método de filtrado llamado Square Root Unscented Kalman Filter (SR-UKF), el cual se decidió implementar finalmente en esta tesis y se trata a continuación.

4.3.1 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 [27]. 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 $\mathbf{P}$ está definida como $\mathbf{S}\mathbf{S}^T = \mathbf{P}$, lo que hace es trabajar directamente con la matriz S sin la necesidad de refactorizar para obtener la matriz $\boldsymbol{P}$. Esto lo logra usando tres herramientas del algebra lineal, que son la descomposición QR, la actualización del factor de Cholesky (Apéndice A) 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:

1. Se inicializa el SR-UKF con $\boldsymbol{x}_0$ y $\mathbf{S}_0$.
2. Se calcula la predicción de los estados $\boldsymbol{x}^-$.
   a. Se definen los *2L* puntos sigma $\boldsymbol{\mathcal{X}}_{k-1}$:

$$\boldsymbol{\mathcal{X}}_{k-1} = [\hat{\boldsymbol{x}}_{k-1} \quad \hat{\boldsymbol{x}}_{k-1} + \gamma\mathbf{S}_k \quad \hat{\boldsymbol{x}}_{k-1} - \gamma\mathbf{S}_k] \tag{4.40}$$

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

   b. Se usan los puntos sigma $\boldsymbol{\mathcal{X}}_{k-1}$ y la ecuación no lineal del proceso para transformar a los *2L* puntos obtenidos:

$$\boldsymbol{\mathcal{X}}_{k|k-1}^* = \mathbf{F}[\boldsymbol{\mathcal{X}}_{k-1}, \mathbf{u}_{k-1}] \tag{4.41}$$

c. Se calcula el vector de predicciones del estado

$$\hat{\boldsymbol{x}}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \boldsymbol{\mathcal{X}}_{i,k|k-1}^* \tag{4.42}$$

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

$$\begin{aligned} \mathbf{S}_k^- &= \mathrm{qr}\left\{\left[\sqrt{W_1^{(c)}}\left(\boldsymbol{\mathcal{X}}_{1:2L,k|k-1}^* - \hat{\boldsymbol{x}}_k^-\right) \quad \sqrt{\mathbf{R}^{\mathrm{v}}}\right]\right\} \\ \mathbf{S}_k^- &= \mathrm{cholupdate}\left\{\mathbf{S}_k^- \;,\; \boldsymbol{\mathcal{X}}_{0,k}^* - \hat{\boldsymbol{x}}_k^- \;,\; W_0^{(c)}\right\} \end{aligned} \tag{4.43}$$

donde $\mathbf{R}^{\mathrm{v}}$ es la matriz de covarianza del ruido en el proceso, y su raíz se obtiene a partir de la descomposición de Cholesky.

3. Se calcula la predicción de las mediciones y las matrices de covarianza asociadas por medio de:

a. $\boldsymbol{\mathcal{X}}_{k|k-1}$ para encontrar los 2L puntos sigma para las mediciones como:

$$\begin{aligned} \boldsymbol{\mathcal{X}}_{k|k-1} &= [\hat{\boldsymbol{x}}_k^- \quad \hat{\boldsymbol{x}}_k^- + \gamma\mathbf{S}_k^- \quad \hat{\boldsymbol{x}}_k^- - \gamma\mathbf{S}_k^-] \\ \boldsymbol{\mathcal{Y}}_{k|k-1} &= \mathbf{H}[\boldsymbol{\mathcal{X}}_{k|k-1}] \end{aligned} \tag{4.44}$$

b. $\boldsymbol{\mathcal{Y}}_{k|k-1}$ para calcular predicciones de las mediciones $\hat{\boldsymbol{y}}_k^-$.

$$\hat{\boldsymbol{y}}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \boldsymbol{\mathcal{Y}}_{i,k|k-1} \tag{4.45}$$

c. $\boldsymbol{\mathcal{Y}}_{k|k-1}$ y $\hat{\boldsymbol{y}}_k^-$ para calcular la matriz de predicción de la covarianza del error del modelo de medición como:

$$\begin{aligned} \mathbf{S}_{\tilde{y}_k} &= \mathrm{qr}\left\{\left[\sqrt{W_1^{(c)}}\left(\boldsymbol{\mathcal{Y}}_{1:2L,k|k-1} - \hat{\boldsymbol{y}}_k^-\right) \quad \sqrt{\mathbf{R}^{\mathrm{n}}}\right]\right\} \\ \mathbf{S}_{\tilde{y}_k} &= \mathrm{cholupdate}\left\{\mathbf{S}_{\tilde{y}_k} \;,\; \boldsymbol{\mathcal{Y}}_{0,k} - \hat{\boldsymbol{y}}_k^- \;,\; W_0^{(c)}\right\} \end{aligned} \tag{4.46}$$

donde $\mathbf{R}^{\mathrm{n}}$ es la matriz de covarianza del ruido de medición y su raíz se calcula previamente a través de la descomposición de Cholesky.

d. Se usan $\boldsymbol{\mathcal{Y}}_{k|k-1}, \hat{\boldsymbol{y}}_k^-, \boldsymbol{\mathcal{X}}_{k|k-1}$ y $\hat{\boldsymbol{x}}_k^-$, para calcular la matriz de predicción de la covarianza del error cruzado como:

$$\mathbf{P}_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} [\boldsymbol{\mathcal{X}}_{i,k|k-1} - \hat{\boldsymbol{x}}_k^-][\boldsymbol{\mathcal{Y}}_{i,k|k-1} - \hat{\boldsymbol{y}}_k^-]^T \tag{4.47}$$

4. Cuando se obtiene un nuevo conjunto de mediciones $\boldsymbol{y}_k$, es posible obtener la estimación de los estados $\hat{\boldsymbol{x}}_k$ mediante:

a. $\mathbf{S}_{\tilde{y}_k}$ y $\mathbf{P}_{x_k y_k}$ para calcular la matriz de ganancias.

$$\mathbf{K}_k = (\mathbf{P}_{x_k y_k} / \mathbf{S}_{\tilde{y}_k}^T) / \mathbf{S}_{\tilde{y}_k} \tag{4.48}$$

b. $\hat{\boldsymbol{x}}_k^-, \mathbf{K}_k, \boldsymbol{y}_k$ y $\hat{\boldsymbol{y}}_k^-$ como sigue:

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \mathbf{K}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k^-) \tag{4.49}$$

c. $\mathbf{S}_k^-$, $\mathbf{S}_{\tilde{y}_k}$ y $\mathbf{K}_k$ para calcular la estimación de la matriz de covarianzas del error $\mathbf{S}_k$ como sigue:

$$\mathbf{U} = \mathbf{K}_k \mathbf{S}_{\tilde{y}_k}$$
$$\mathbf{S}_k = \text{cholupdate}\{\mathbf{S}_k^-, \ \mathbf{U} \ , -1\} \tag{4.50}$$

5. Este proceso se repite desde el paso 2 hasta el 4.

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

Para un satélite real, como ya se ha visto en el capítulo 2, 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 obtenida de los sensores. El SR-UKF como ya se ha visto es un método analítico que una vez debidamente ajustado, fusiona los datos de sensores de distinta clase para obtener la orientación sin los procesos de ruido que aquejan a los sensores.

Debido a 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 ([18] y [28]). 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:

$$\boldsymbol{x} = [\boldsymbol{\omega} \quad \mathbf{q}]^T = \left[\omega_x \ \ \omega_y \ \ \omega_z \ \ \eta \ \ \epsilon_1 \ \ \epsilon_2 \ \ \epsilon_3\right]^T \tag{4.51}$$

donde $\boldsymbol{\omega}$ corresponde a las velocidades angulares del satélite sobre sus diferentes ejes y $\mathbf{q}$ corresponde al cuaternión que representa su orientación o apuntamiento, por lo que se tienen 7 estados (3 componentes de la velocidad angular y 4 componentes del cuaternión). Para iniciar el 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 $\boldsymbol{x}_0$.

La matriz $\mathbf{S}_0$ corresponde a la matriz de covarianzas que corresponden al ruido del proceso, este valor inicial es propuesto y puede definirse 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 $\mathbf{F}$ (como se ve en la ecuación (4.24)) puede expresarse como:

$$\mathbf{F}_k = e^{F\Delta k} \approx \mathbf{I} + \mathbf{F}\Delta t \tag{4.52}$$

donde Δt corresponde al tiempo de muestreo.

Para la tarea de la estimación de la orientación del satélite solo es necesario estudiar su cinemática como se vio en el capítulo anterior, por lo que la matriz $\mathbf{F}$ se determinó a través del modelo cinemático que corresponde a la rotación de un cuerpo rígido, por lo que el modelo del proceso queda como:

$$\mathbf{F}_k = \mathbf{I}_{7\times7} + \frac{\tau}{2}\begin{bmatrix} 0 & \cdots & & & 0 & & \\ \vdots & \ddots & & & \vdots & & \\ & & 0 & -x_1 & -x_2 & -x_3 \\ 0 & \cdots & x_1 & 0 & x_3 & -x_2 \\ & & x_2 & -x_3 & 0 & x_1 \\ & & x_3 & x_2 & -x_1 & 0 \end{bmatrix}_{7\times7} \tag{4.53}$$

donde τ corresponde al tiempo de muestreo y $\boldsymbol{x}$ al vector de estados.

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 TRIAD para obtener un cuaternión como medición, por lo que la matriz que describe al proceso de medición es una matriz identidad, es decir:

$$\mathbf{H}_k = \mathbf{I}_{7\times7} \tag{4.54}$$

Por último, como ya se ha mencionado ya con anterioridad este modelo servirá para ser probado sobre una mesa suspendida en aire que servirá como simulador para la prueba de los algoritmos de apuntamiento del satélite real a este modelo, así como al de inmersión e invarianza, hay que agregar el cambio de sistemas coordenados. Basándose en los cambios entre sistemas coordenados del capítulo 3.6 se tiene que obtener la matriz de rotación $\boldsymbol{R}_E^B$ de forma tal que se obtengan las mediciones del cuerpo pero desde el punto de vista de la Tierra, esta matriz se obtiene de la siguiente multiplicación de matrices de rotación:

$$\boldsymbol{R}_E^B = \boldsymbol{R}_E^I \boldsymbol{R}_I^O (\boldsymbol{R}_B^O)^T = \boldsymbol{R}_E^I \boldsymbol{R}_I^O \boldsymbol{R}_O^B \tag{4.55}$$

Esto se aplicaría a las mediciones obtenidas desde el acelerómetro y el magnetómetro, ya que son sensores que poseen un vector de referencia fijo a través del cual se mide la orientación, de manera que las transformaciones de las medidas de los sensores quedan como:

$$\boldsymbol{a}_B = \boldsymbol{R}_E^B[\boldsymbol{a}_E + v_a(t)]$$
$$\boldsymbol{m}_B = \boldsymbol{R}_E^B[\boldsymbol{m}_E + v_m(t)]$$

donde $v_a(t)$ y $v_m(t)$ son procesos de ruido aleatorios que están presentes en los sensores. Los giróscopos no se ven influenciados por estos cambios de sistemas de referencia, ya que sus mediciones están directamente relacionadas al satélite y no a la Tierra. Estos cambios del sistema de referencia se aplicarían en cada iteración del filtro UKF, después de que se muestrea a los sensores antes de empezar el filtrado. Por ultimo cabe hacer notar que el acelerómetro no es un sensor que se use dentro de un satélite pequeño por lo que tendría que ser reemplazado por otra clase se sensor, como es el sensor de Sol.

Capítulo 5

Implementación

Hacer procesamiento de datos en tiempo real para muchos procesos era algo imposible hace algunas décadas debido a que no existía un dispositivo que tuviera la suficiente capacidad y velocidad para procesar la información. A finales de los años 70 y en años subsecuentes compañías como Intel, NEC, AT&T y varias más empezaron a desarrollar dispositivos de procesamiento muy poderosos llamados DSP. Hoy día, la capacidad de procesamiento de muchos dispositivos es muy alta y la cantidad de aplicaciones que se han desarrollado mediante ellos ha hecho crecer mucho el avance de la tecnología. Debido a ello es posible integrar procesos complicados de control y procesamiento en un dispositivo muy pequeño. En este capítulo se describe como se usó una plataforma de desarrollo DSP para implementar el estimador de Inmersión e Invarianza así como la primera propuesta del filtro SR-UKF para estimar la orientación del satélite HumSAT.

5.1 Plataforma DSP

Actualmente, *Texas Instruments* es de los principales desarrolladores de DSP (Digital Signal Processor) en el mundo siendo uno de los grandes pioneros en el mercado desde hace más de 20 años. Hoy día existen una serie de dispositivos digitales, entre ellos los microcontroladores, a quienes se puede considerar con cierta equivalencia a un DSP, sin embargo, aunque los microcontroladores pueden hacer procesamiento de señales no lo harían ni con la velocidad ni con la eficiencia con la que un DSP podría hacerlo. Un DSP es un dispositivo que aunque es parecido a un microcontrolador, su principal diferencia radica en que tiene una arquitectura diseñada para manejar información a grandes velocidades y para su procesamiento en aplicaciones de tiempo real [29].

En el caso del problema de la orientación, como ya se ha visto en el capítulo anterior, se necesitan efectuar una serie de cálculos a gran velocidad para realizar la estimación de la orientación en tiempo real. Debido a que el Instituto de Ingeniería UNAM persigue ejecutar de forma auto-sostenida tales algoritmos en una mesa experimental suspendida en aire, se optó por emplear una tarjeta de desarrollo. El DSP elegido fue el TMS320F28335 de *Texas Instruments* debido a que ya se tenían algunas experiencias con dispositivos similares y a que la familia C2000 de DSP de Texas Instruments tiene los periféricos necesarios para hacer la adquisición de señales y realizar el control de actuadores a través de sus módulos PWM.

5.1.1 Tarjeta de desarrollo eZdspF28335

Esta fue la tarjeta utilizada para el desarrollo de los estimadores de orientación, fabricada por la compañía *Spectrum Digital,* la cual permite evaluar las capacidades del DSP TMS320F28335, figura 5.1. Esta tarjeta fue seleccionada con base en la cantidad de periféricos que tiene, pensando en que aparte de desarrollar los actuales estimadores serviría para realizar la parte de control de los actuadores integrados en la mesa suspendida en aire [30].

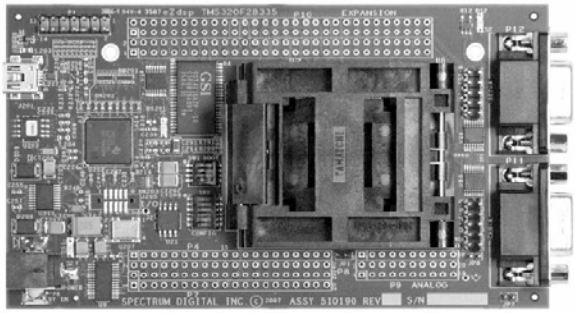


Figura 5. 1 Tarjeta de desarrollo eZdspF28335.

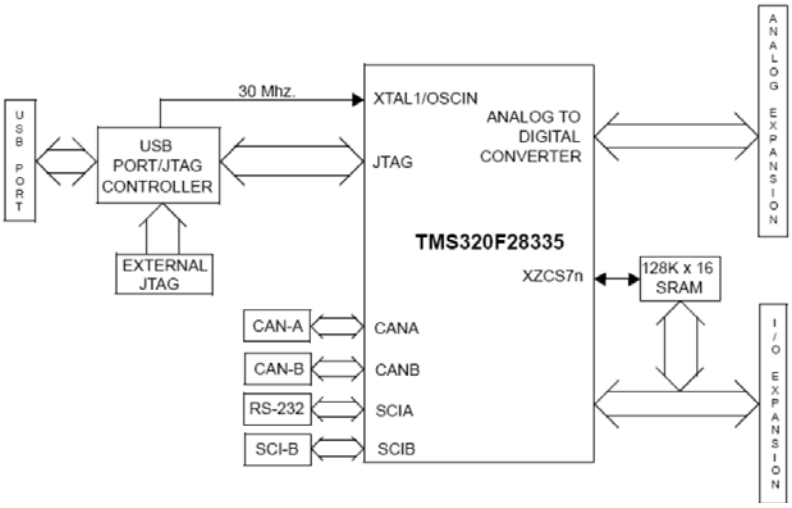


Figura 5. 2 Diagrama de bloques de la tarjeta de desarrollo.

El diagrama de bloques de la tarjeta se muestra en la figura 5.2, en el cual se tiene el siguiente hardware:

- DSP TMS320F28335 con unidad de punto flotante.
- Oscilador de 30 MHz.
- Memoria SRAM externa de 256kB.
- Memoria EEPROM 24WC256 de 256kB.
- Conector hacia los puertos de comunicaciones RS-232 y CAN del DSP.
- Conector hacia las entradas del convertidor analógico-digital del DSP.
- Conectores de expansión, para tener acceso a las terminales del DSP.
- Interruptores de configuración.
- Conector de emulación del protocolo JTAG IEEE 1149.1.

5.1.2 DSP TMS320F28335

Este DSP pertenece a la familia C2000 de Texas Instruments, integra una arquitectura de 32 bits, gran cantidad de periféricos, convertidores analógico - digitales y empaquetado de 176 pines, esta familia en especial ofrece gran capacidad para resolver problemas de control en tiempo real para una gran variedad de aplicaciones [31].

Características:

- Tecnología CMOS de alto desempeño
  - Hasta 150 MHz (6.67 ns por ciclo de reloj).
  - Núcleo de 1.9V / 1.8V, E/S de 3.3V.
- CPU de 32 bits de alto desempeño
  - Cuenta con una unidad de punto flotante (FPU) basada en el protocolo IEE-754.
  - Operaciones MAC de 16x16-bit y 32x32-bit.
  - Operación MAC dual de 16x16.
  - Arquitectura Harvard.
  - Alta velocidad de respuesta en el manejo de interrupciones.
  - Modelo de programación de memoria unificado.
  - Eficiencia en código (C/C++ y ensamblador).
  - 8 niveles de pipeline con protección.
- Interface externa (XINTF) de 16 o 32 bit
  - Alcance de 2M x 16 direcciones.
- Memoria en chip
  - 256K x 16 en memoria Flash y 34K x 16 en memoria SARAM.
  - 1K x 16 OTP ROM.
- Memoria Boot ROM (8K x 16), esta memoria tiene los modos de inicio vía puertos SCI, SPI, CAN, I2C, McBSP y XINTF, además de tablas de senos y cosenos.

Periféricos:

- 6 Canales DMA (para el ADC, McBSP, ePWM, XINTF y la SRAM).
- 3 CPU-timers de 32-bit.
- Hasta 6 módulos mejorados de PWM (ePWM).
- Hasta 6 módulos de captura mejorados (eCAP).
- Hasta 2 módulos de enconder en cuadratura (eQEP).
- Módulo mejorado de convertidores analógico-digitales.
- Hasta 2 módulos mejorados de red de control de área (eCAN).
- Hasta 3 módulos de comunicación serial (SCI).
- Un modulo de interfaz de periféricos serial (SPI).
- Un modulo de comunicaciones inter – circuitos integrados (I2C).
- Hasta 2 puertos seriales multicanal con buffer (McBSP).
- Entradas/salidas digitales compartidas.

- Interfaz externa (XINTF).

En la figura 5.3 se presenta el diagrama de bloques del dispositivo DSP.

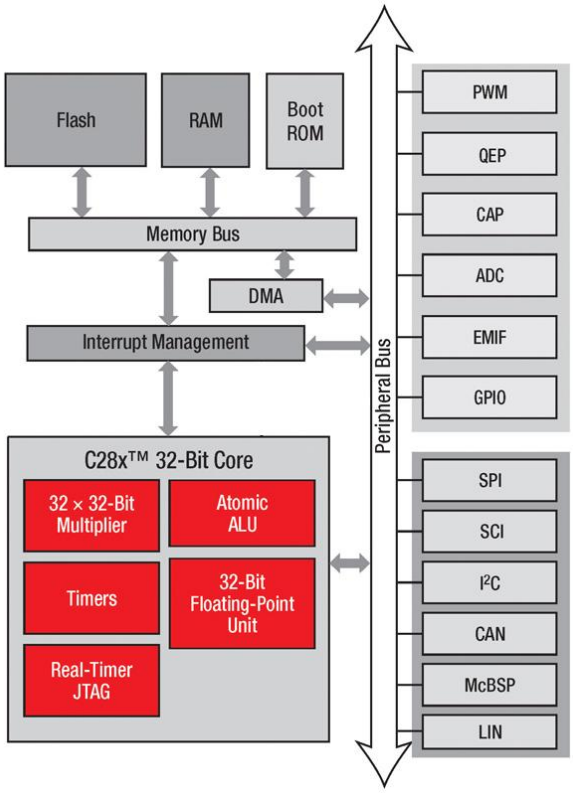


Figura 5. 3 Diagrama de bloques del DSP.

5.2 Plataforma de software

El software desarrollado en esta tesis para el DSP fue programado sobre la interfaz de desarrollo que también proporciona Texas Instruments para la programación de sus dispositivos, llamada *Code Composer Studio* (CCS), figura 5.4. 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.

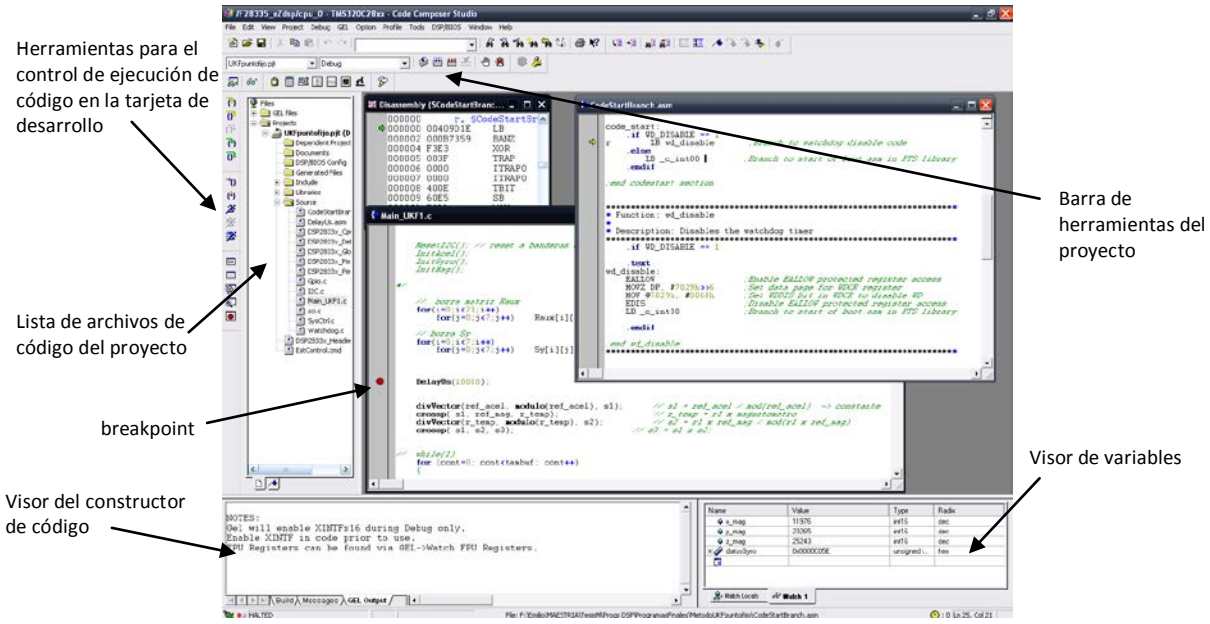


Figura 5. 4 Interfaz del CCS.

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 3.3 que incluye el mismo fabricante de la tarjeta de desarrollo eZdspF28335, 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 [29].

5.3 Tarjeta de sensores

La tarjeta de sensores que se utilizó fue de la compañía *Sparkfun,* figura 5.5, la cual contiene las tres clases de sensores requeridas para experimentar con métodos de estimación de orientación. La tarjeta integra 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 [32].

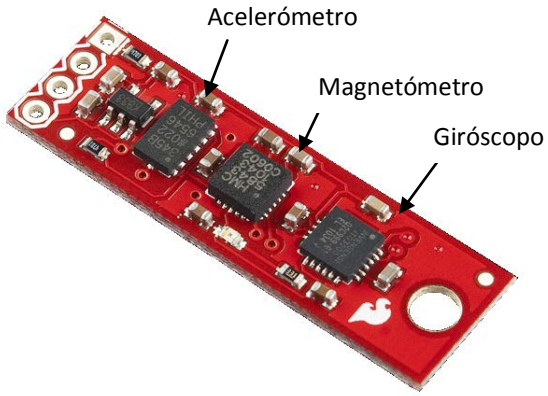


Figura 5. 5 Tarjeta de sensores de *Sparkfun*.

- ***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˚ [33].

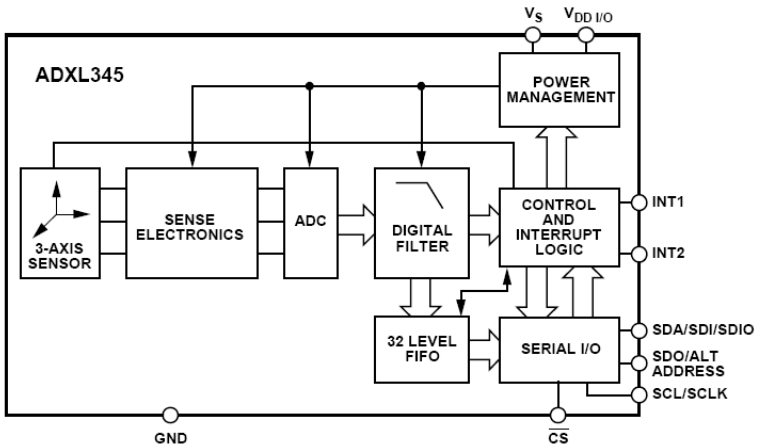


Figura 5. 6 Diagrama de bloques del acelerómetro ADXL345.

- ***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], [34].

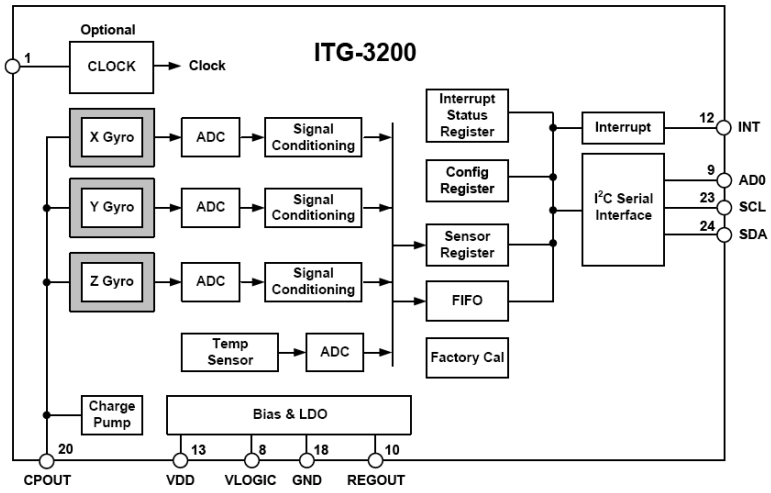


Figura 5. 7 Diagrama de bloques del giróscopo ITG-3200.

Resumiendo en la siguiente tabla, las características de los sensores son:

	Magnetómetro	**Acelerómetro**	**Giróscopo**
Rango	±4 *gauss*	±2 *g*, ±4 *g*, ±8 *g*, ±16 *g*	±2000 [°/s]
Longitud de palabra	12 bits	10 bits	16 bits
Error de alineación		±0.1	
Sensibilidad		3.9 m*g*/LSB	14.375 LSB / (°/s)
Sensibilidad cruzada	± 0.2%	± 1%	± 2%
Velocidad de medición	0.5 – 50 Hz	6.25-3200 Hz	1-8 kHz (interno)
Noise performance		<1 LSB rms (xy), <1.5 LSB rms (z)	0.03 °/s/√Hz

Tabla 5. 1 Resumen de propiedades de la tarjeta de sensores de *Sparkfun.*

5.4 Esquemas de programación

A continuación se describen los detalles generales de los programas realizados en la tarjeta de desarrollo para implementar los algoritmos del método de Inmersión e Invarianza (I&I) y el filtro de Kalman Unscented (UKF) encaminados a calcular la orientación del satélite con respecto a la Tierra. Estos algoritmos se ilustran en diagramas de flujo resumidos que se irán detallando. Cabe resaltar que, en vista de que los

algoritmos serán empleados en la mesa suspendida en aire para realizar pruebas de validación, junto con los algoritmos de control correspondientes, se omite la transformación de coordenadas y algunas otras consideraciones que fueron recalcadas en el capítulo 4.

5.4.1 Inmersión e Invarianza

En esta sección se describe el método de I&I y se hacen algunas anotaciones con respecto a su implementación en la mesa suspendida en aire. En la figura 5.8 se ve el diagrama de flujo que se siguió durante su programación y posteriormente se describen las funciones que en él se indican.

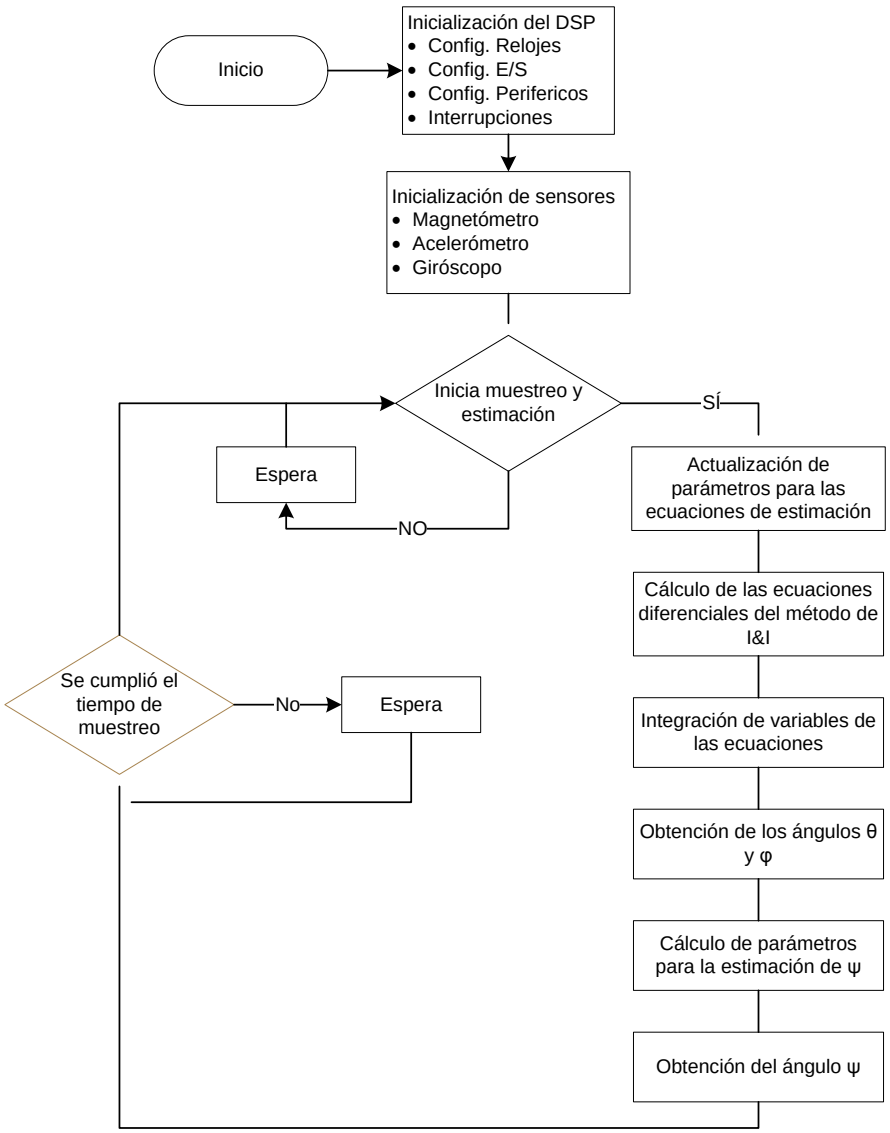


Figura 5. 8 Diagrama de flujo del método de Inmersión e Invarianza.

Inicialización del DSP: En esta parte se configura al DSP con sus registros de configuración de forma tal que se especifican las entradas y salidas utilizadas al momento de la implementación. Se configuran los pines destinados al control del bus I2C y los pines del puerto serie se configuran y se dejan como opción para que más adelante al emplear la mesa suspendida en aire (MSA) se realice envío de mensajes a una PC sobre el estado de los sistemas por medio de un radioenlace. En el DSP es necesario encender relojes para el funcionamiento de algunos periféricos como el puerto I2C y el puerto serie. El puerto I2C se configuró como maestro a una velocidad de 100 kHz en su bus, mientras el puerto serie fue configurado como asíncrono a una tasa de transmisión de 9600 bauds, 8 bits, sin paridad y un bit de parada.

Inicialización de sensores: Se tienen que inicializar los 3 sensores de manera independiente a través del puerto de comunicaciones I2C (Apéndice B) previo al inicio del algoritmo de estimación. La configuración que tienen es la siguiente:

- *Magnetómetro*:
  - Dirección de dispositivo: 0x1E (Hexadecimal).
  - Velocidad de muestreo de 50 Hz (máxima).
  - Rango de medición de ±1 Gauss, 1300 [lectura/mGauss], rango de lectura en los registros de -2048 a 2047 (12 bits con complemento a 2).
  - Modo de conversión continua, en este modo toma lecturas continuas y las deposita en los registros de lectura correspondientes.
- *Acelerómetro*:
  - Dirección del dispositivo: 0x53 (hexadecimal).
  - Se definen los umbrales de actividad e inactividad.
  - Se define el tiempo de inactividad.
  - Se define un modo de operación normal y una velocidad de muestreo de 100 Hz.
- *Giróscopo*:
  - Dirección del dispositivo: 0x68 (hexadecimal).
  - Se configura una velocidad de muestreo de 100 Hz.
  - Rango de medición de ±2000 [°/s]
  - Se configura el filtro paso bajas interno con una frecuencia de corte de 42 Hz.
  - Se activa la interrupción para cuando haya datos nuevos disponibles.

Actualización de parámetros: en este módulo se actualiza la matriz anti-simétrica relacionada con las velocidades angulares $S(\Omega_G)$, y los parámetros β_1, β_2 y β_3.

Como se observa en la figura 5.9 la secuencia de actualización de parámetros β se da a partir de las mediciones de los acelerómetros en matrices de 3x1. El proceso de integración para obtener la matriz η se realiza utilizando el método de Euler de la siguiente forma:

$$\eta_{i+1} = \eta_i + \tau \cdot \dot{\eta}$$

donde τ es el período de muestreo utilizado, este procedimiento se realiza para cada componente de la matriz.

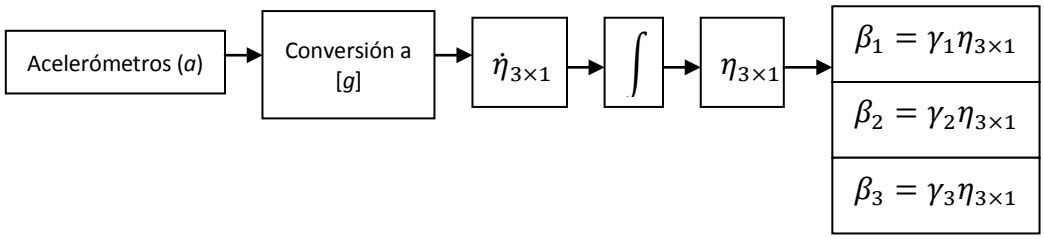


Figura 5. 9 Secuencia de actualización de parámetros β.

La actualización de la matriz anti-simétrica $S(\Omega_G)$ se hace acomodando las lecturas de los giróscopos como se muestra en la figura 5.10.

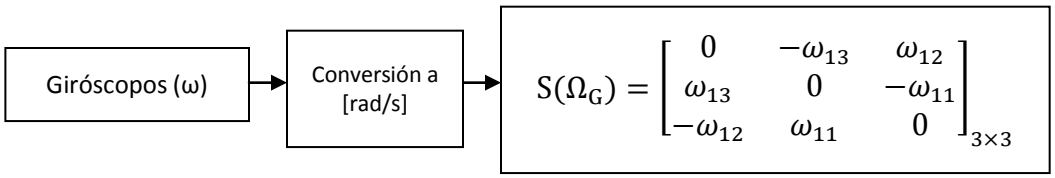


Figura 5. 10 Matriz anti-simétrica de datos de los giróscopos.

Cálculo de ecuaciones e integración: a partir de los parámetros actualizados se aplican las ecuaciones que se ven en (4.21) siguiendo el diagrama que se presenta en la figura 5.11.

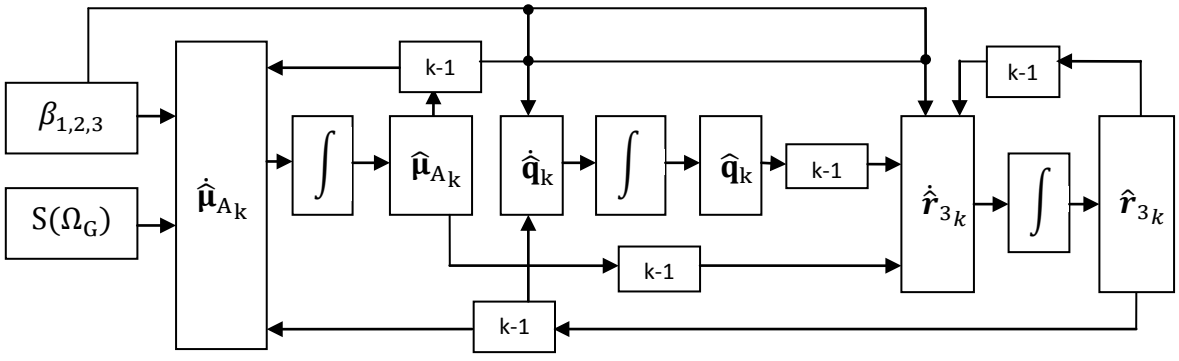


Figura 5. 11 Diagrama de flujo de datos en el cálculo de ecuaciones del método I&I en los ángulos θ y ϕ.

De igual forma se utiliza el método de Euler para las rutinas de integración.

Obtención de los ángulos θ y ϕ: para obtener los ángulos, a partir del vector $\hat{r}_3$ se utiliza la primera ecuación vista en (4.19) considerando el error $z_1 = 0$. De esto se obtendrá el

vector r_3, este vector se normaliza previamente. Una vez normalizado este vector, como ya se había indicado en el capítulo 4.2.1, corresponde a la tercera columna de la matriz de cosenos directores que se ve en la ecuación (3.3), a partir de entonces solo se despejan los ángulos, figura 5.12.

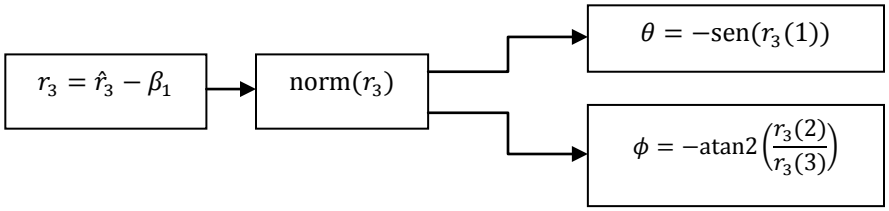


Figura 5. 12 Obtención de ángulos θ y ϕ.

Cálculo de parámetros de estimación y obtención del ángulo ψ: para obtener el último ángulo se recurre a los ángulos θ y ϕ como ya se había visto en el capítulo (4.2.1) y expresado en la ecuación (4.22), pero debido a que las lecturas de los magnetómetros también tienen ruido, es necesario realizar un proceso de estimación. Para realizar el proceso de estimación de manera similar que en el caso anterior se obtienen los parámetros β además de que se proponen los parámetros ***a,*** solo que en este caso ya no se trata con matrices de 3x1 sino de datos escalares y el procedimiento de solución sigue el diagrama de flujo indicado en la figura 5.13.

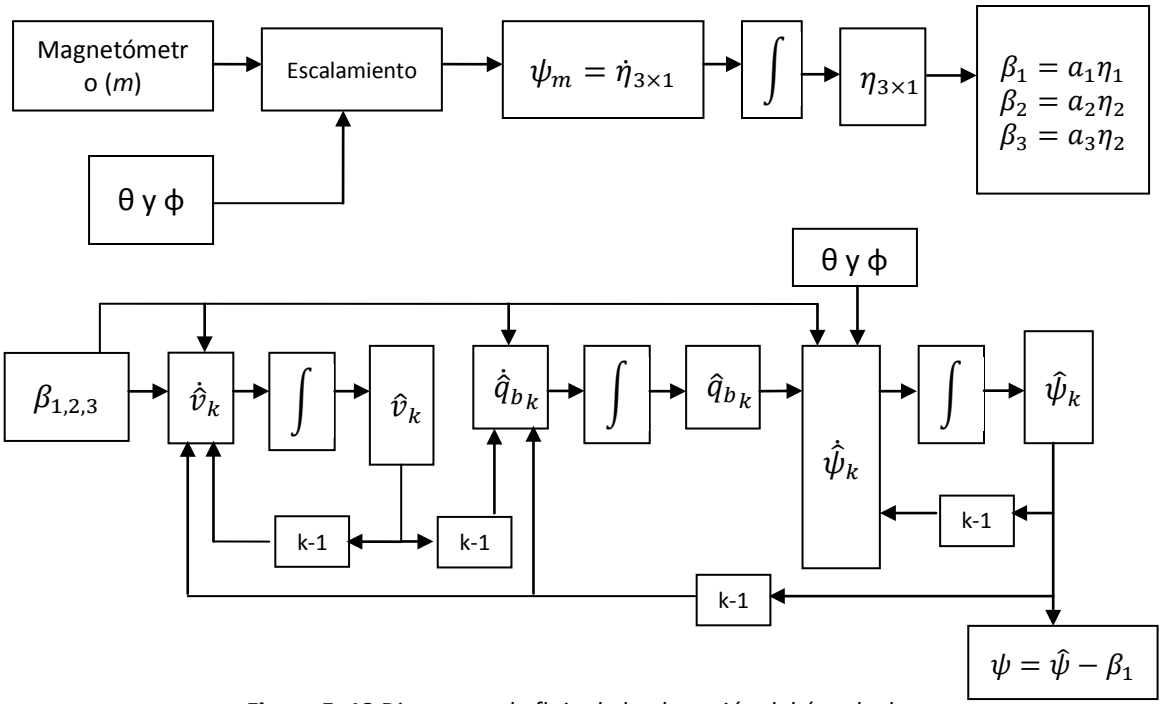


Figura 5. 13 Diagramas de flujo de la obtención del ángulo ψ.

5.4.2 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 DSP, este diagrama se muestra de manera general en la figura 5.14. La inicialización del DSP y la tarjeta de sensores son exactamente iguales que en el método de Inmersión e Invarianza anteriormente descrito.

En el método SR-UKF implementado en el DSP 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 iniciales para el algoritmo y también se calculan de manera previa algunas constantes que no es necesario calcular con cada iteración. Las condiciones iniciales que se requieren en el algoritmo son un vector de estado inicial $\hat{x}_0$ y una matriz de covarianzas inicial.

Las matrices de covarianza del ruido tanto para el modelo de proceso como para el modelo de medición $\sqrt{\mathbf{R}^{\mathrm{v}}}$ y $\sqrt{\mathbf{R}^{\mathrm{n}}}$, también se pueden calcular de manera previa para no generarlas en cada iteración.

Otras de las constantes que se calculan son los pesos $W_i^{(m)}$ y $W_i^{(c)}$, que se calculan a partir de los parámetros escalares L, γ, β, α y κ, el cálculo de estos pesos se explicó en el capítulo 4.3. El ajuste de estos parámetros se tiene que realizar de manera previa a través de una serie de pruebas ya que no hay una metodología para elegir algunos de estos. Para este caso también se puede realizar un cálculo previo de una parte del método TRIAD que corresponde a los vectores de referencia.

Método TRIAD: El método TRIAD es un método determinístico empleado en la estimación de orientación como ya se había mencionado en el capítulo (4.1.1), 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 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. 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.

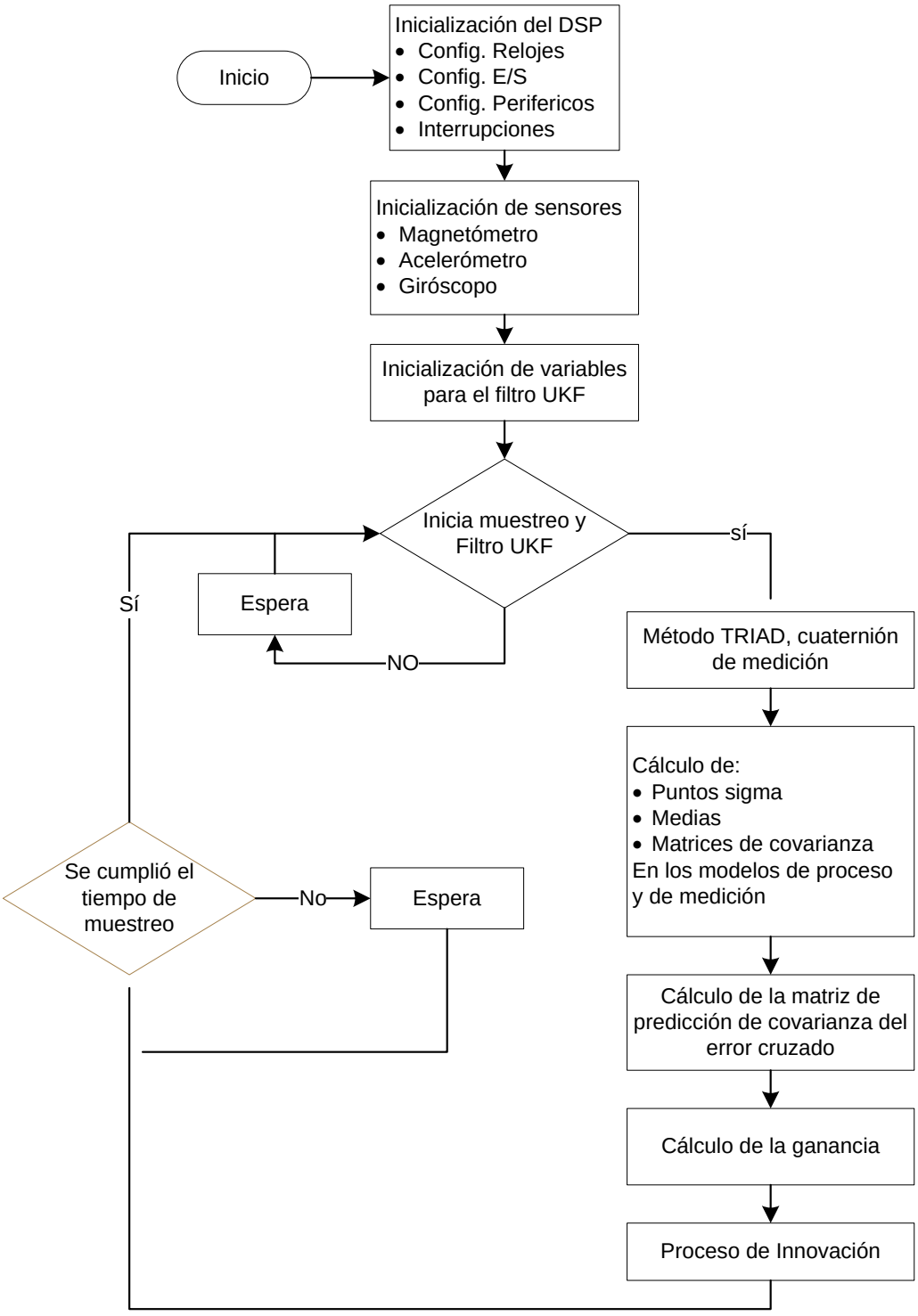


Figura 5. 14 Diagrama de flujo del filtro de Kalman Unscented.

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. Pero el método solo permite obtener la matriz de cosenos directores, por lo que hay que obtener el cuaternión a través de la siguiente forma:

$$d_1 = a_{23} - a_{32} \qquad d_2 = a_{31} - a_{13} \qquad d_3 = a_{12} - a_{21}$$
$$\lambda = \text{norm}(d)$$
$$\gamma = a_{11} + a_{22} + a_{33}$$
$$\theta = \text{acos}(\gamma)$$

$\eta = \cos\left(\frac{\theta}{2}\right)$ ... parte escalar del cuaternión

$\boldsymbol{e} = \boldsymbol{\lambda} sen\left(\frac{\theta}{2}\right)$ ... parte vectorial del cuaternión (imaginaria)

donde η es la parte escalar del cuaternión y $\boldsymbol{e}$ es la parte vectorial. Aunque una de las ventajas principales del filtro UKF a diferencia del método de I&I es evitar el uso de funciones trascendentes, se tiene que hacer uso de ellas en este paso para la correcta representación del cuaternión en forma normalizada.

Puntos sigma: Los puntos sigma se obtienen tanto para el modelo del proceso como para el modelo de medición, esto se hace a partir del vector de estimación del estado anterior $\hat{\boldsymbol{x}}_{k-1}$ y el vector $\hat{\boldsymbol{x}}_k^-$ para el caso del modelo de medición, las matrices de covarianza del error $\mathbf{S}_k$ y $\mathbf{S}_k^-$ para el caso del modelo de medición, y el factor escalar γ. Los puntos sigma quedan expresados en dos matrices de 7x15, una para el modelo del proceso y la otra para el modelo de medición. En este paso cabe resaltar que se define una operación donde un vector columna se suma a cada columna de una matriz.

$$\boldsymbol{\mathcal{X}}_{k-1} = [[\hat{\boldsymbol{x}}_{k-1}]_{7\times1} \quad \overbrace{[\hat{\boldsymbol{x}}_{k-1} + \gamma\mathbf{S}_k]_{7\times7}}^{\begin{bmatrix} (\hat{x}_{k-1})_{11} + \gamma(\mathrm{S}_k)_{11} & \cdots & (\hat{x}_{k-1})_{11} + \gamma(\mathrm{S}_k)_{17} \\ \vdots & \ddots & \vdots \\ (\hat{x}_{k-1})_{71} + \gamma(\mathrm{S}_k)_{71} & \cdots & (\hat{x}_{k-1})_{71} + \gamma(\mathrm{S}_k)_{77} \end{bmatrix}_{7\times7}} \quad [\hat{\boldsymbol{x}}_{k-1} - \gamma\mathbf{S}_k]_{7\times7}]$$

Medias: las medias se obtienen a partir de una transformación de los puntos sigma a través de los modelos del proceso y de medición, de esto se obtiene una nueva matriz de 7x15. Finalmente se multiplica cada columna de la matriz por cada peso de ajuste $W_i^{(m)}$ al tiempo que se suman todas las columnas para obtener el vector de la media, este proceso se realiza tanto en el modelo de proceso como en el modelo de medición.

$$\boldsymbol{\mathcal{X}}_{k|k-1}^* = \mathbf{F}[\mathcal{X}_{k-1}, \mathbf{u}_{k-1}] = \begin{bmatrix} \mathrm{F}_{11} & \cdots & \mathrm{F}_{1,15} \\ \vdots & \ddots & \vdots \\ \mathrm{F}_{17} & \cdots & \mathrm{F}_{7,15} \end{bmatrix} \text{ ... (puntos transformados)}$$

$$\hat{\boldsymbol{x}}_k^- = W_1^{(m)} \begin{bmatrix} \mathrm{F}_{11} \\ \vdots \\ \mathrm{F}_{71} \end{bmatrix} + \cdots + W_{15}^{(m)} \begin{bmatrix} \mathrm{F}_{1,15} \\ \vdots \\ \mathrm{F}_{7,15} \end{bmatrix} \text{ .... (media)}$$

Raíces cuadradas de las matrices de covarianza del error: estas raíces se obtienen a partir de las matrices de covarianza del ruido $\sqrt{\mathbf{R}^{\mathrm{v}}}$ y $\sqrt{\mathbf{R}^{\mathrm{n}}}$, las medias $\hat{\boldsymbol{x}}_k^-$ y $\hat{\boldsymbol{y}}_k^-$, y las matrices de puntos sigma transformados a través del modelo de proceso ($\boldsymbol{\mathcal{X}}_{i,k|k-1}^*$) o de medición ($\boldsymbol{\mathcal{Y}}_{k|k-1}$) obtenidas, como se ve en las ecuaciones (4.43) y (4.46) respectivamente. En el proceso para obtener estas matrices es necesario recurrir a la factorización QR y la descomposición de Cholesky (Apéndice A), para ello es que en el DSP se implementaron rutinas generalizadas para resolver ambos métodos en matrices de cualquier tamaño. En el proceso de obtención de la factorización QR hay que notar que solo es necesario obtener la matriz **R**, es necesario obtener una matriz de 7x7 por lo que previamente se transpone la matriz a factorizar, de esta manera se obtienen, una matriz **Q** de 21x21 y la matriz **R** de 21x7, pero la matriz obtenida está compuesta principalmente por ceros como se muestra a continuación:

$$\mathbf{S}_k^- = \mathrm{qr}\left\{\left(\left[\left[\sqrt{W_1^{(c)}}\left(\boldsymbol{\mathcal{X}}_{1:2L,k|k-1}^* - \hat{x}_k^-\right)\right]_{7\times 14} \quad \left[\sqrt{\mathbf{R}^{\mathrm{v}}}\right]_{7\times 7}\right]_{7\times 21}\right)^{\mathrm{T}}\right\} \begin{cases} \boldsymbol{Q}_{21\times 21} \\ \boldsymbol{R}_{21\times 7} \end{cases}$$

Pero

$$\mathbf{R}_{21\times 7} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{17} \\ r_{21} & \cdots & r_{26} & 0 \\ \vdots & \iddots & 0 & 0 \\ r_{71} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{17} \\ r_{21} & \cdots & r_{26} & 0 \\ \vdots & \iddots & 0 & 0 \\ r_{71} & 0 & \cdots & 0 \end{bmatrix} = \mathbf{R}_{7\times 7}$$

Y después se aplica la descomposición de Cholesky

$$\mathbf{S}_k^- = \mathrm{cholupdate}\left\{\mathbf{R}_{7\times 7}\ , \left(\boldsymbol{\mathcal{X}}_{0,k}^* - \hat{\boldsymbol{x}}_k^-\right)_{7\times 1}\ ,\ W_0^{(c)}\right\}$$

que equivale a $$\mathbf{S}_k^- = \mathrm{chol}\left\{\mathbf{R}^{\mathrm{T}}\mathbf{R} + \sqrt{W_0^{(c)}}(\boldsymbol{\mathcal{X}}_{0,k}^* - \hat{\boldsymbol{x}}_k^-)(\boldsymbol{\mathcal{X}}_{0,k}^* - \hat{\boldsymbol{x}}_k^-)^{\mathrm{T}}\right\}$$

que fue como quedó implementado en el DSP. Por último, cabe resaltar que durante la obtención de la matriz **R** y su empleo en la descomposición de Cholesky hay una gran cantidad de operadores que son cero, a los que no es necesario operarlos, como sucede durante la obtención de la matriz **R** y las multiplicaciones de la matriz **R** durante la descomposición de Cholesky. Por ello es que en la implementación de los algoritmos se hicieron estas consideraciones adaptando los métodos para evitar realizar las multiplicaciones por cero.

Matriz de covarianza del error cruzada ($\mathbf{P}_{x_k y_k}$): esta se obtiene a partir de los pesos $W_i^{(c)}$ calculados en la inicialización de variables, los puntos sigma usados en el modelo de medición ($\boldsymbol{\mathcal{X}}_{k|k-1}$) y los puntos sigma transformados en el modelo de medición ($\boldsymbol{\mathcal{Y}}_{k|k-1}$), así como las medias del modelo de proceso ($\widehat{\boldsymbol{x}}_k^-$) y el modelo de medición ($\widehat{\boldsymbol{y}}_k^-$), como se ve en la ecuación (4.47) y entonces se obtiene una matriz de 7x7.

Ganancia ($\mathbf{K}_k$): para el cálculo de la ganancia es necesario recurrir a los recursos previamente calculados, como la matriz de covarianza del error cruzada ($\mathbf{P}_{x_k y_k}$) y la matriz de covarianza del modelo de medición ($\mathbf{S}_{\tilde{y}_k}$), como se ve en la ecuación (4.48). En la ecuación se advierte que se recurre a una división, pero estas variables son matrices por lo que tendría que recurrirse al uso de matrices inversas para obtener la matriz de ganancias de la siguiente forma:

$$\underbrace{\mathbf{K}_k}_{\text{X}}\underbrace{(\mathbf{S}_{\tilde{y}_k}\mathbf{S}_{\tilde{y}_k}^T)}_{\text{A}} = \underbrace{\mathbf{P}_{x_k y_k}}_{\text{B}}$$

Se transponen las matrices,

$$\underbrace{(\mathbf{S}_{\tilde{y}_k}\mathbf{S}_{\tilde{y}_k}^T)^T}_{\text{A}}\underbrace{\mathbf{K}_k{}^T}_{\text{X}} = \underbrace{\mathbf{P}_{x_k y_k}{}^T}_{\text{B}}$$

Se aplica la factorización QR a la matriz $\boldsymbol{A} = \mathbf{S}_{\tilde{y}_k}\mathbf{S}_{\tilde{y}_k}^T$, y se obtiene que $[\boldsymbol{Q}, \boldsymbol{R}] = qr(\boldsymbol{A})$ y finalmente,

$$\underbrace{\boldsymbol{R}}_{\text{A}}\underbrace{\mathbf{K}_k{}^T}_{\text{X}} = \underbrace{\mathbf{Q}^{\mathrm{T}}\mathbf{P}_{x_k y_k}{}^T}_{\text{B}}$$

donde, $\boldsymbol{A}$ es una matriz triangular superior, y X y B son matrices cuadradas, pero según el teorema de mínimos cuadrados X y B son vectores por lo que para obtener la matriz de ganancias $\boldsymbol{K}_k$ hay que repetir el proceso según el número de columnas, y entonces para la primera columna se tendría que:

$$\begin{bmatrix} R_{11} & R_{12} & \cdots & R_{17} \\ 0 & R_{22} & \cdots & R_{27} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{77} \end{bmatrix}\begin{bmatrix} K_{11} \\ K_{21} \\ \vdots \\ K_{71} \end{bmatrix} = \begin{bmatrix} \mathrm{Q}^{\mathrm{T}}\mathrm{P}_{x_k y_k}{}^T{}_{11} \\ \mathrm{Q}^{\mathrm{T}}\mathrm{P}_{x_k y_k}{}^T{}_{21} \\ \vdots \\ \mathrm{Q}^{\mathrm{T}}\mathrm{P}_{x_k y_k}{}^T{}_{71} \end{bmatrix}$$

Y así consecutivamente hasta terminar el número de columnas, para obtener la matriz de ganancias $\boldsymbol{K}_k$.

Proceso de innovación: finalmente, para completar el proceso de iteración, se utiliza la matriz de ganancias ($\mathbf{K}_k$) para obtener el nuevo vector de estados ($\hat{\boldsymbol{x}}_k$) y la nueva matriz de covarianzas del error ($\mathbf{S}_k$), esto según las ecuaciones (4.49) y (4.50).

5.5 Propuesta de implementación en MCUs de menores dimensiones

Las implementaciones hechas a lo largo de esta tesis han sido a través de una tarjeta de desarrollo de DSP para probar los estimadores, si bien estas implementaciones han sido posibles, es necesario tener en cuenta que estos algoritmos tienen que ser aplicados en un nanosatélite. En el cual hay que usar componentes que tengan el menor consumo posible para acoplarse a la disponibilidad energética del vehículo espacial. Sin embargo, en el caso del dispositivo que realice tareas fundamentales de procesamiento resulta necesario hacer un balance entre su capacidad de procesamiento y su consumo. En el caso del DSP utilizado en la tarjeta de desarrollo se utilizan algunos periféricos y su hoja de especificaciones brinda el consumo estimado de varios periféricos, Tabla 5.2.

DSP TMS320F28335 a 150 MHz y 3.3 V **Rango de consumo: 20mA – 315mA (con todos los periféricos en uso)**	
Periférico	**Consumo [mA]**
I2C	2.5
ePWM (3 módulos)	15
SCI	5
CPU-TIMER	2
FPU (opcional)	15
Lectura de memoria FLASH	35 – 40
Pines de Entrada/Salida	30 – 50
TOTAL	89.5 - 129.5

Tabla 5. 2 Consumo de los periféricos utilizados en el DSP TMS320F28335.

Previo a elegir un nuevo microcontrolador para ejecutar los algoritmos es necesario analizar las necesidades de procesamiento de los mismos, basándonos en las cantidades y los tipos de operaciones que se deben de manejar en los mismos. Adicionalmente, también hay que analizar la cantidad de memoria de datos que se requiere. Para el caso del algoritmo de estimación de Inmersión e Invarianza, sus necesidades de memoria y de procesamiento se muestran en las Tablas 5.2 y 5.3 respectivamente.

	Cantidad	**Bytes**
Matrices [3x3]	4	144
Vectores [3x1]	19	228
Variables	15	60
	TOTAL	**432**

Tabla 5. 2 Recursos de memoria del método de Inmersión e Invarianza sin alojamiento dinámico.

	sumas	multiplicaciones	divisiones	Raíz cuadrada	F. trascendentes
Obtención $\boldsymbol{\beta}$'s		9			
Integral	3	3			
Cálculo $\hat{\dot{\boldsymbol{r}}}_{\boldsymbol{3}}$	24	18			2
Cálculo $\hat{\dot{\boldsymbol{q}}}$	9	6			1
Cálculo $\hat{\dot{\boldsymbol{\mu}}}_{\boldsymbol{A}}$	9	6			1
Integrales	9	9			
$\boldsymbol{r}_{\boldsymbol{3}}$ normalizado	2	3		1	
Cálculo $\boldsymbol{\varphi}_{\boldsymbol{m}}$	3	7			8
Integral	3	3			
Obtención $\boldsymbol{\beta}$'s		3			
Cálculo $\hat{\dot{\boldsymbol{\varphi}}}$	7	5	2		6
Cálculo $\hat{\dot{\boldsymbol{q}}}_{\boldsymbol{b}}$	3	2			1
Cálculo $\hat{\dot{\boldsymbol{v}}}$	3	2			1
integrales	3	3			
$\boldsymbol{\varphi}$	1				
Conversión ángulos	6	7			3
TOTAL	**83**	**83**	**5**	**1**	**20**

Tabla 5. 3 Recursos operativos del método de Inmersión e Invarianza.

Para el algoritmo SR-UKF sus necesidades operativas y de memoria utilizadas para la dimensión del filtro programado, se muestran en las tablas 5.4 y 5.5, respectivamente.

	sumas	multiplicaciones	divisiones	Raíz cuadrada	F. trascendentes
TRIAD	43	53	18	5	3
Puntos sigma (proceso)	98	49			
Transformación de puntos sigma y media $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$	285	345			
Obtener $\mathbf{S}_{\boldsymbol{k}}^{-}$	3899	3955	21	14	
Puntos sigma (medición)	98	49			
Transformación de puntos sigma y media $\widehat{\boldsymbol{y}}_{\boldsymbol{k}}^{-}$	105	105			
Obtener $\mathbf{S}_{\tilde{\boldsymbol{y}}_{\boldsymbol{k}}}$	3899	3955	21	14	
Matriz $\mathbf{P}_{\boldsymbol{x}_{\boldsymbol{k}}\boldsymbol{y}_{\boldsymbol{k}}}$	1918	1183			
Ganancia, $\mathbf{K}_{\boldsymbol{k}}$	973	945	119	7	
Innovación, $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}$ y $\mathbf{S}_{\boldsymbol{k}}$	2491	2429	4	50	
TOTAL	**13809**	**13068**	**183**	**90**	**3**

Tabla 5. 3 Recursos operativos del filtro SR-UKF.

		Cantidad	Bytes
Sensores	Variables 16 bit	27	54
TRIAD	Cuaternión [4x1]	1	16
	Matrices [3x3]	4	144
	Variables (32 bit)	41	164
UKF	Matrices [7x7]	8	1568
	Matrices [7x15]	4	1680
	Matrices [21x7]	1	588
	Vectores[7x1]	8	224
	Vectores[15x1]	2	120
	Variables (32 bit)	6	24
		TOTAL	**4582**

Tabla 5. 4 Recursos de memoria del método SR-UKF sin alojamiento dinámico.

Con base en las tablas de necesidades de los algoritmos presentadas anteriormente y considerando su consumo de energía, se seleccionaron las siguientes propuestas de microcontroladores [35][36]:

- Atmel AT91SAM3S4A, flash 256 kB, SRAM 48kB
- Atmel AT91SAM4S8A, flash 512kB, SRAM 128kB
- Texas Instruments Stellaris LM4F111E5QR, flash 128kB, SRAM 32kB
- Texas Instruments Stellaris LM3S1367, flash 128kB, SRAM 32kB

Aunque si bien estos microcontroladores cumplen con las características de consumo y capacidad de memoria para sostener estos algoritmos, falta evaluar el desempeño de procesamiento. Todos los microcontroladores fueron elegidos porque poseen núcleos de procesamiento que son muy utilizados actualmente en una gran cantidad de dispositivos móviles por su capacidad de procesamiento y bajo consumo, particularmente los núcleos ARM en sus versiones Cortex M3 y Cortex M4. Respecto a este último, ha sido precisamente el que se ha empleado para desarrollar la nueva computadora de vuelo para nuestro satélite HumSAT (en otra tesis de maestría en Electrónica de nuestro grupo que se defenderá en Enero de 2012), por lo cual será factible probar la implementación de los algoritmos cuando se fabrique esta computadora, lo cual sucederá antes de Marzo de 2012.

Por otro lado, a primera vista y debido a que el microcontrolador Stellaris LM4F111E5QR de *Texas Instruments* posee una unidad de punto flotante, pareciera que su empleo sería más sencillo para trasladarle el código debido a que son del mismo fabricante. Por otro lado, el microcontrolador AT91SAM4S8A posee el nuevo núcleo Cortex M4 que incorpora una unidad de punto flotante y posee capacidades de un DSP con bajo consumo. Los microcontroladores AT91SAM3S4A y LM3S1367 poseen igualmente una buena capacidad de procesamiento a bajo consumo pero no tienen la unidad de punto flotante lo que hace necesario utilizar aritmética de punto fijo, 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. Lo único que faltaría revisar para estos microcontroladores es su manejo de funciones trascendentes ya que suelen ser procesos que consumen una gran cantidad de operaciones.

Capítulo 6

Resultados experimentales

En el presente capítulo se presentan los resultados experimentales obtenidos a partir de los dos estimadores programados, el método de Inmersión e Invarianza y el filtro SR-UKF, tanto en MATLAB como en la plataforma DSP mencionada en el capítulo anterior. Primero se muestran las simulaciones realizadas en MATLAB a partir de muestreos directamente obtenidos de los sensores, un muestreo corresponde a una IMU comercial y otros muestreos al grupo de sensores que se mostró en el capítulo anterior.

Finalmente se muestran los resultados experimentales obtenidos sobre la plataforma DSP logrados a partir de la programación del método de Inmersión e Invarianza y el filtro SR-UKF usando punto fijo y punto flotante.

6.1 Simulaciones realizadas en MATLAB

Previo a la ejecución de las simulaciones se utiliza un muestreo generado con los sensores que componen a la IMU. Se obtuvieron muestras tanto de una IMU comercial como de la tarjeta de sensores que se describió en el capítulo anterior. La IMU comercial que se utilizó fue la MTi-G de la compañía Xsens (figura 6.1), dentro de la cual existe un grupo de sensores junto con un microcontrolador que tiene programado un filtro Extendido de Kalman (EKF) para realizar la estimación de la orientación.

Figura 6. 1 IMU de la compañía Xsens.

Esta IMU entrega los datos procesados en forma de cuaternión, ángulos o bien los datos directos de sus sensores. En la tabla 6.1 se muestran las características de los sensores de la IMU, [36].

		Velocidad de giro	Aceleración	Campo magnético	Temperatura
	Unidades	**[°/s]**	**[m/s^2]**	**[mGauss]**	**[°C]**
Dimensiones		3 ejes	3 ejes	3 ejes	-
Escala Total	**[unidades]**	+/- 300	+/- 50	+/- 750	-55 a 125
Lienalidad	**[% de ET]**	0.1	0.2	0.2	<1
Estabilidad del Bias	**[unidades 1σ]**	1	0.02	0.1	0.5
Estabilidad del factor de escala	**[% 1σ]**	-	0.03	0.5	-
Densidad del ruido	**[unidades/√Hz]**	0.05	0.002	0.5	-
Error de alineación	**[grados]**	0.1	0.1	0.1	-
Ancho de banda	**[Hz]**	40	30	10	-
Resolución A/D	**[bits]**	16	16	16	12

Tabla 6. 1 Características de los sensores de la IMU MTi-G de Xsens.

En las figura 6.2 se presentan datos de muestreo realizado por la IMU en sus tres sensores, cada uno de tres ejes.

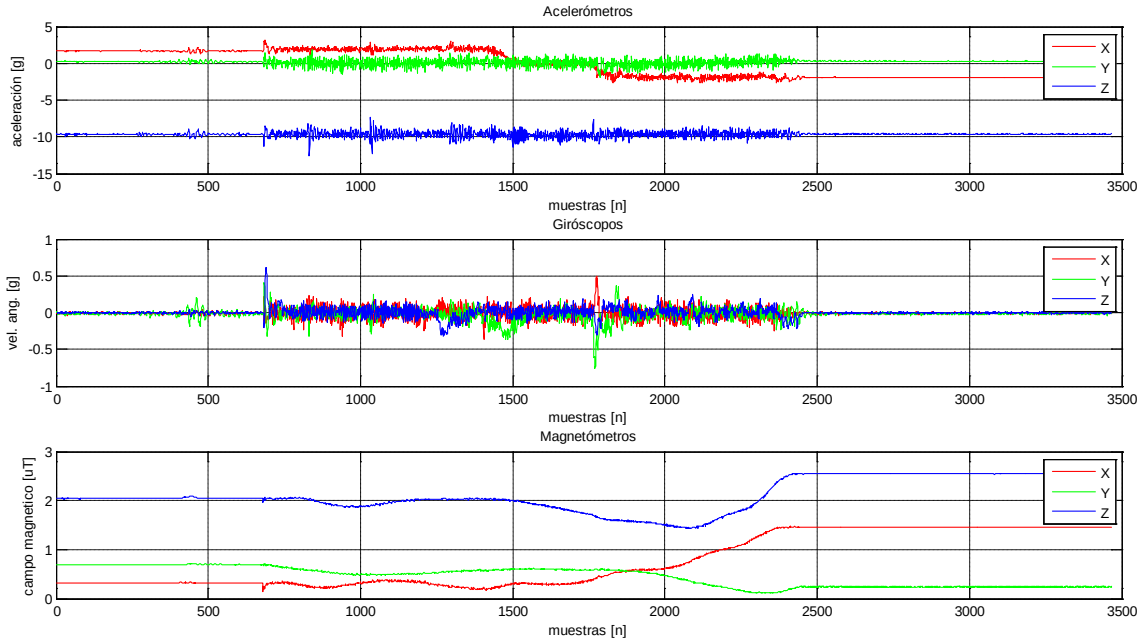


Figura 6. 2 Muestreo de sensores (X, Y, Z) de la IMU MTi-G de Xsens .

En las siguientes figuras se muestran las estimaciones de los ángulos de orientación y se comparan las salidas del método TRIAD contra el método de estimación de la IMU

MTi-G, figura 6.3; contra el método de filtrado SR-UKF simulado en MATLAB, figura 6.4; y contra el método de I&I simulado en MATLAB, figura 6.5.

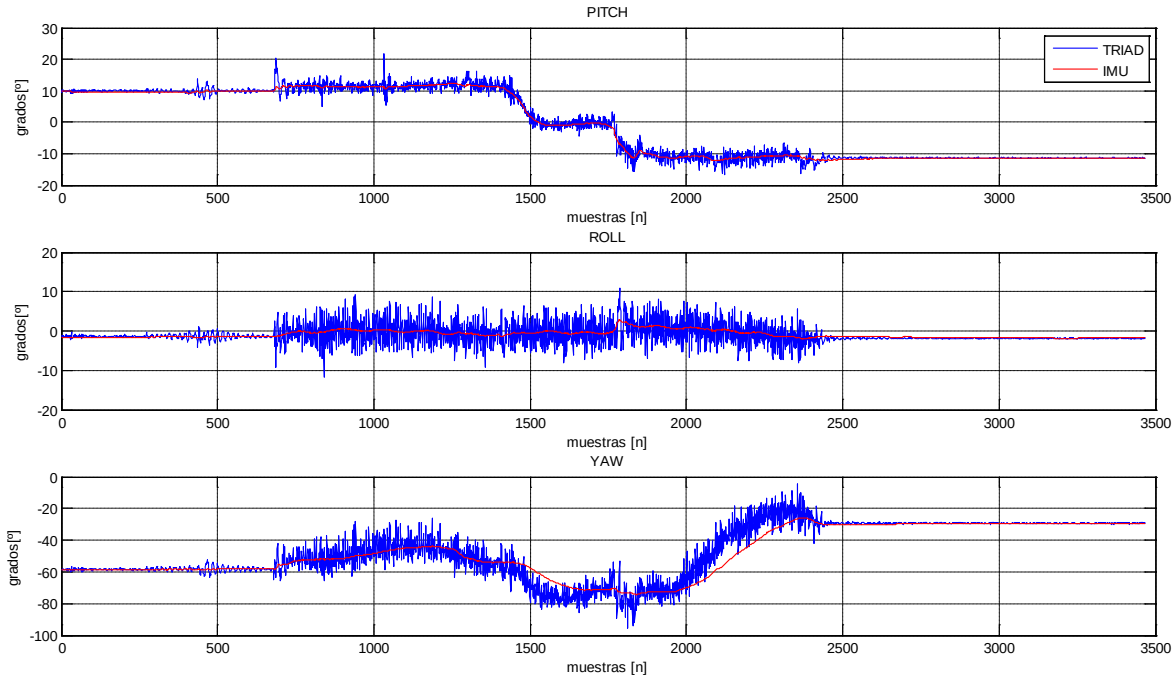


Figura 6. 3 Comparación entre el Método TRIAD y el método de estimación de la IMU MTi-G.

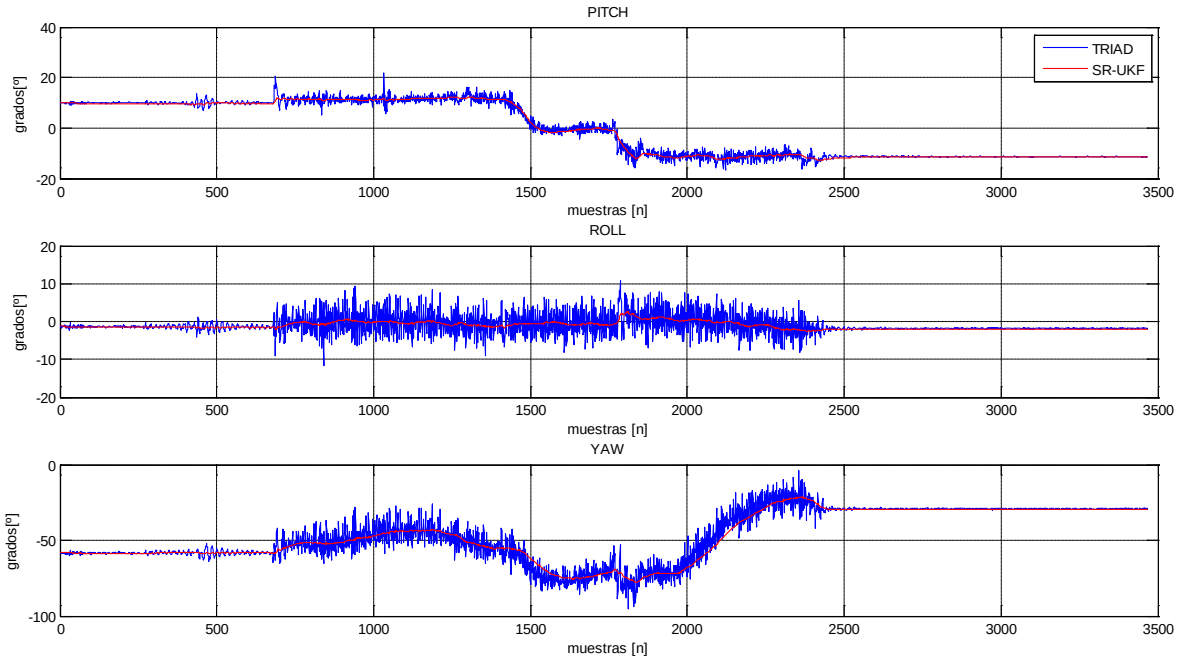


Figura 6. 4 Comparación entre el método TRIAD y el método de filtrado SR-UKF simulado en MATLAB.

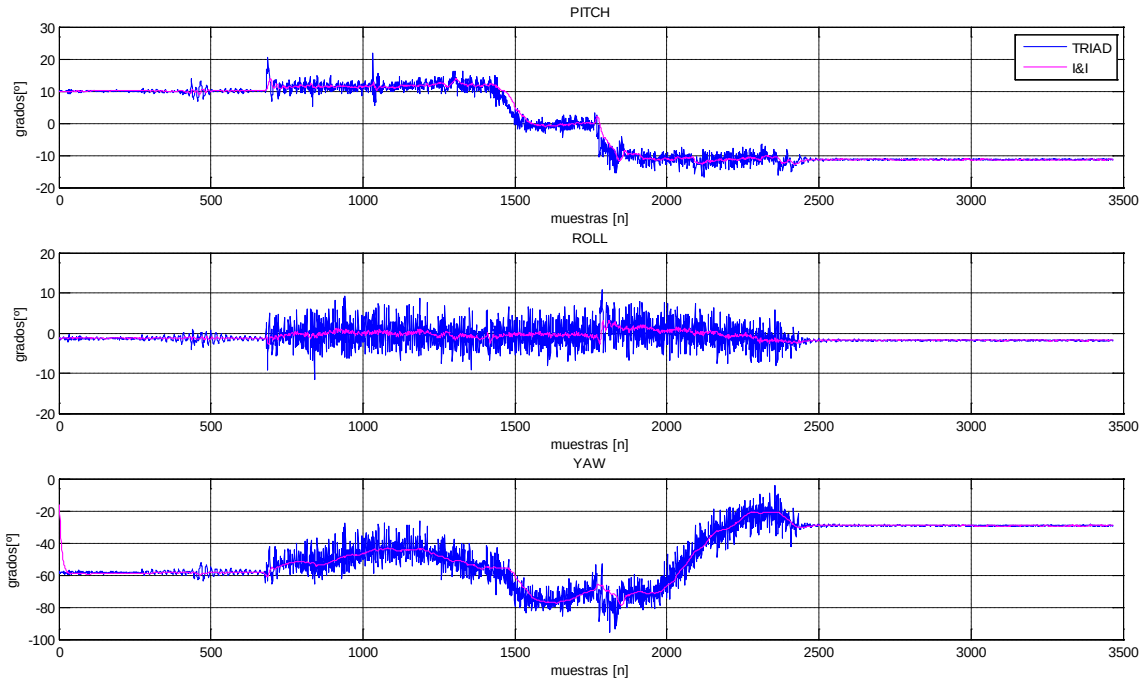


Figura 6. 5 Comparación entre el método TRIAD y el método de I&I simulado en MATLAB.

En cada una de las figuras 6.6, 6.7 y 6.8 se muestra la comparación de resultados de los métodos: I&I, filtro SR-UKF y filtrado EKF de la IMU MTi-G, en los angulos de cabeceo, alabeo y guiñada. Estos resultados fueron obtenidos en MATLAB para cada ángulo de inclinación, y en la tabla 6.2 se muestran los parámetros utilizados en los estimadores para lograr tales resultados.

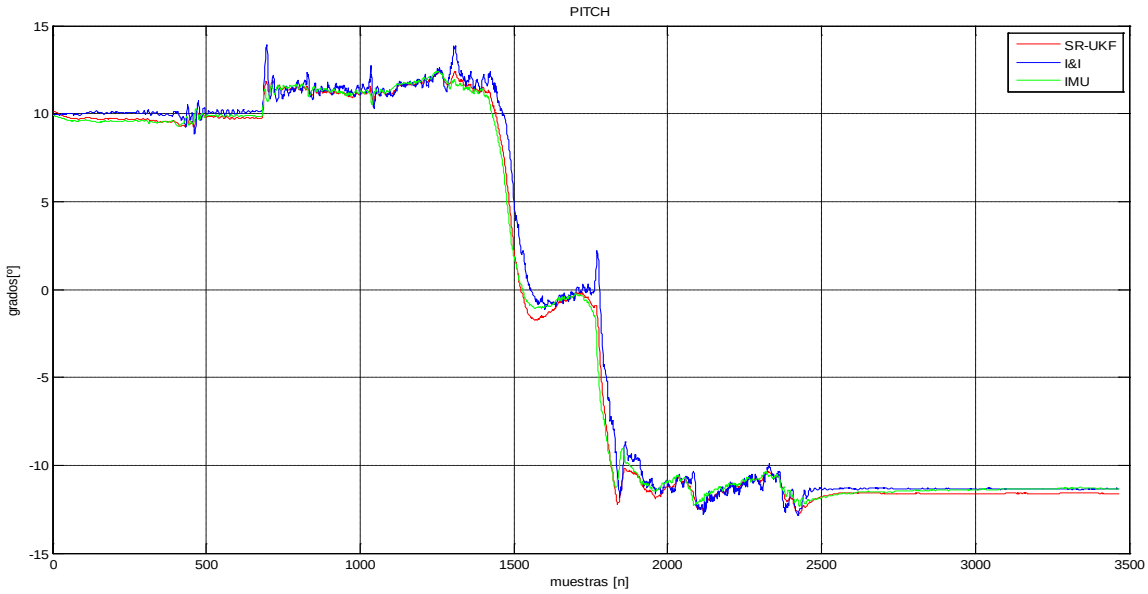


Figura 6. 6 Comparación de los tres métodos de estimación SR-UKF, I&I e IMU EKF, en el ángulo de cabeceo.

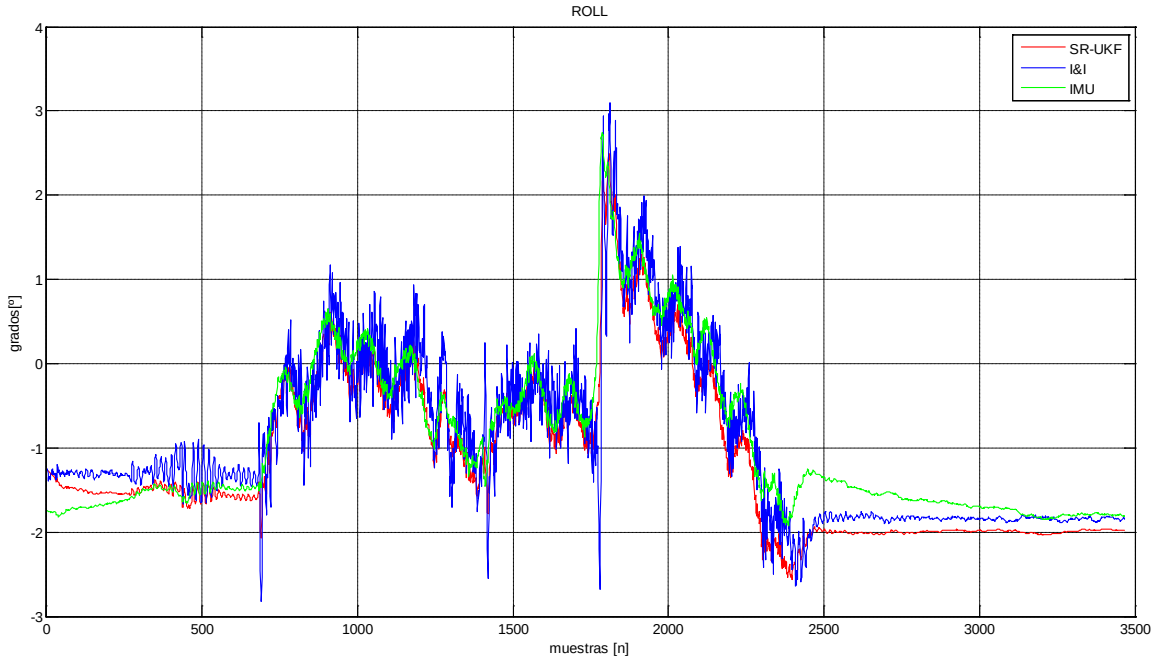


Figura 6. 7 Comparación de los tres métodos de estimación SR-UKF, I&I e IMU EKF en el ángulo de alabeo.

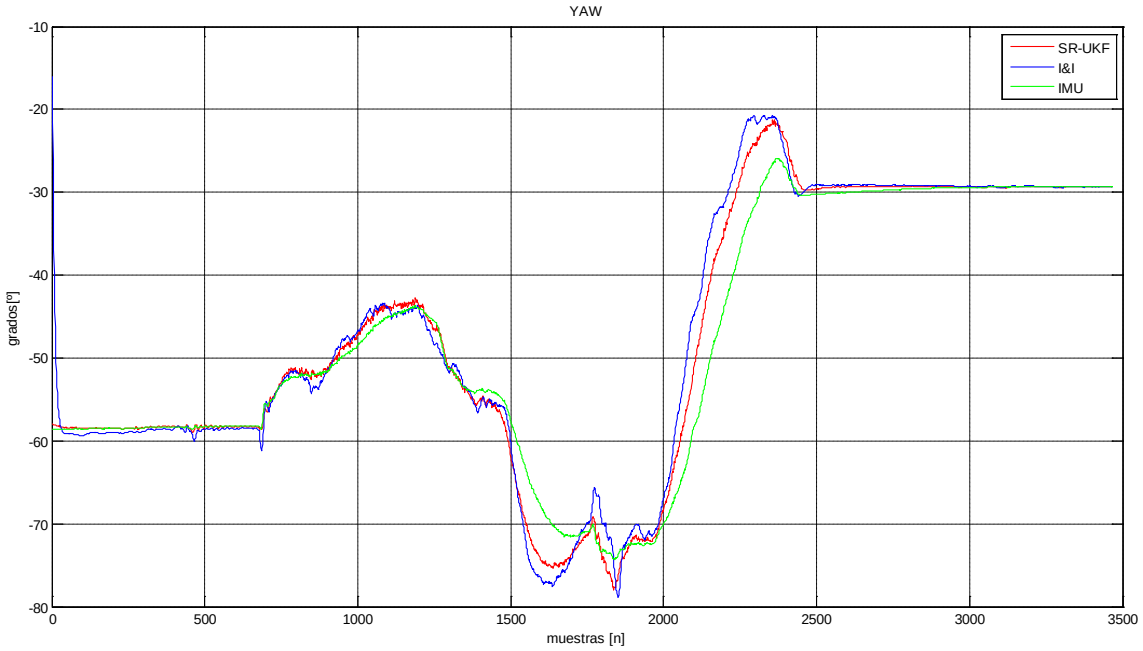
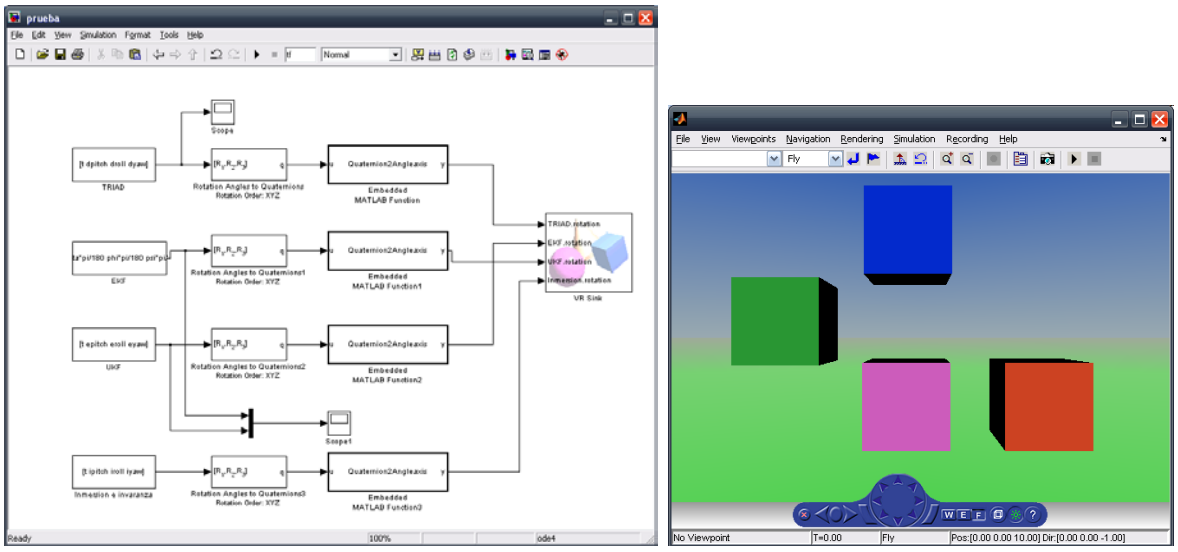


Figura 6. 8 Comparación de los tres métodos de estimación SR-UKF, I&I e IMU EKF, en el ángulo de guiñada.

Inmersión e Invarianza		Filtro Kalman Unscented	
γ_1	8	L	7
γ_2	0.1	β	2
γ_3	0.001	α	0.45
$\bar{q}$	-2	γ	1.190588089979066
$\bar{\mu}_A$	-2	$W_0^{(m)}$	0.7975
a_1	-12.5	$W_0^{(c)}$	3.595
a_2	-1	$W_{1-14}^{(m)} = W_{1-14}^{(c)}$	0.014464285714286
a_3	-80	$\mathbf{S}_0$	$diag\{0.5\}_{7\times7}$
$\bar{q}_\psi$	2.5	$\sqrt{\mathbf{R}^{\mathrm{v}}}$	$diag\{[1.2\ 1.2\ 1.2\ 0.4\ 0.4\ 0.4\ 0.4]\}_{7\times7}$
$\bar{v}_b$	0.001	$\sqrt{\mathbf{R}^{\mathrm{n}}}$	$diag\{[1\ 1\ 1\ 0.5\ 0.5\ 0.5\ 0.5]\}_{7\times7}$
Tiempo de muestreo: 10 [ms]			

Tabla 6. 2 Parámetros de sintonización utilizados para el muestreo en la IMU MTi-G de Xsens.

Para observar los efectos de los métodos de estimación se realizó también una simulación mediante el toolbox de animación 3D de MATLAB donde se presentan 4 cubos representando la orientación obtenida con los 3 muestreos filtrados de los sensores y la medición de la orientación por el método TRIAD, figura 6.9.



(a) (b)

Figura 6. 9 (a) Diagrama de bloques de Simulink para la animación en 3D y (b)Animación 3D de MATLAB representando la orientación obtenida con los muestreos filtrados de los sensores .

También se efectuaron simulaciones a partir de los muestreos que se realizaron de la tarjeta de sensores de *Sparkfun*, con el fin de obtener los parámetros que serían utilizados en el método de I&I y el filtro SR-UKF sobre la plataforma DSP. En la figura 6.10 se muestra el histórico del muestreo de los sensores, en las figuras 6.11, 6.12 y 6.13 se

muestran los comparativos entre el método TRIAD y el método I&I, el método TRIAD y el filtro SR-UKF y la comparativa entre el método I&I y el filtro SR-UKF, respectivamente.

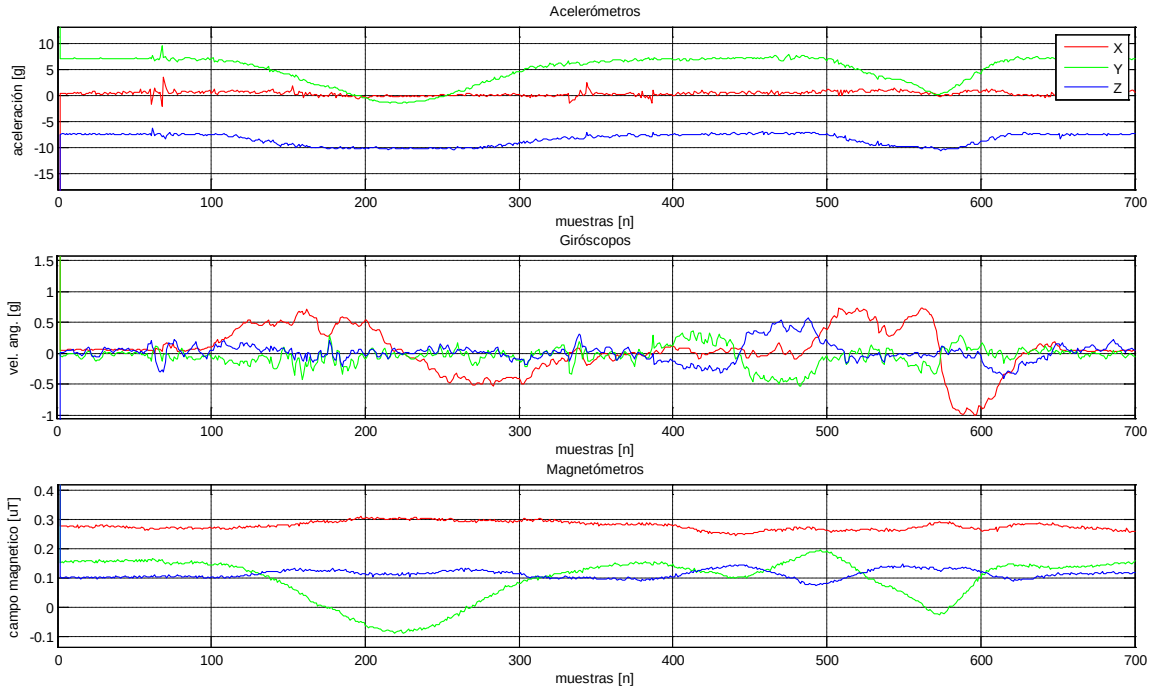


Figura 6. 10 Datos de muestreo de sensores de la tarjeta *Sparkfun*.

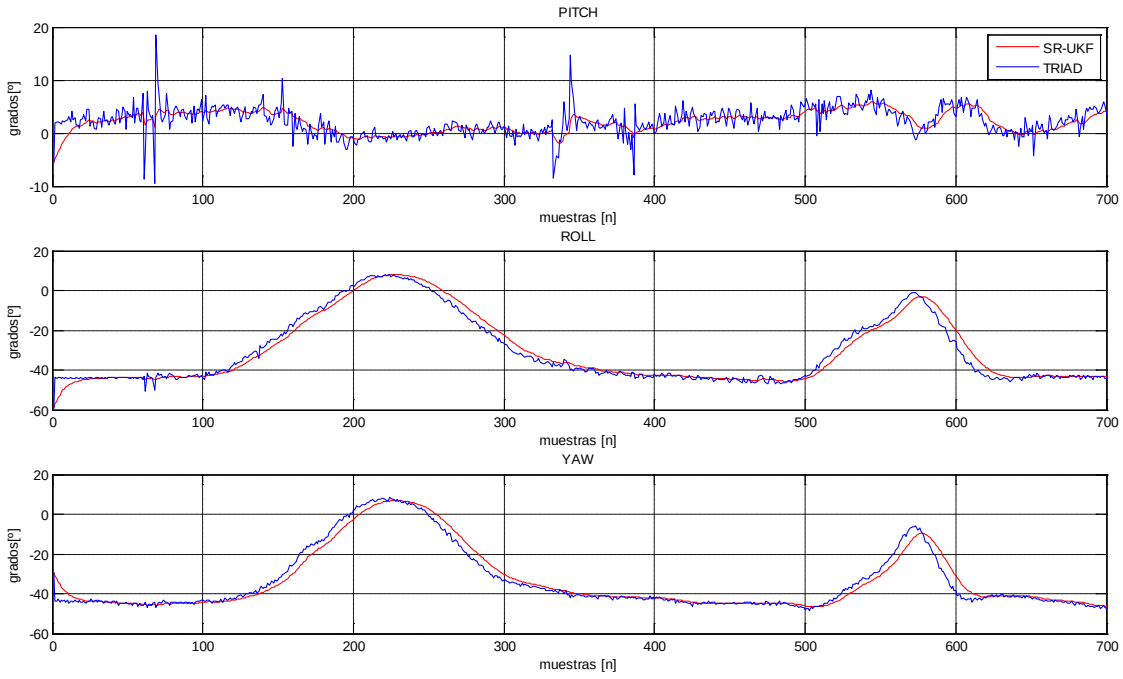


Figura 6. 11 Comparación entre método TRIAD y el filtrado SR-UKF con datos de la tarjeta *Sparkfun*.

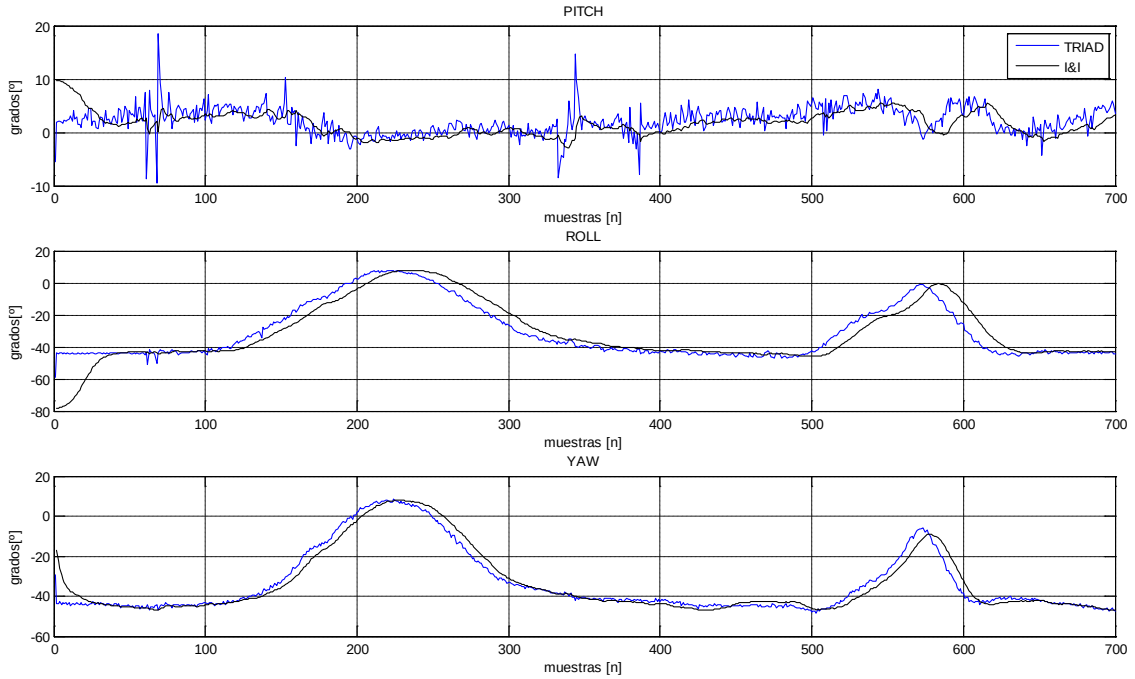


Figura 6. 12 Comparación entre el método TRIAD y el método de I&I con datos de la tarjeta *Sparkfun*.

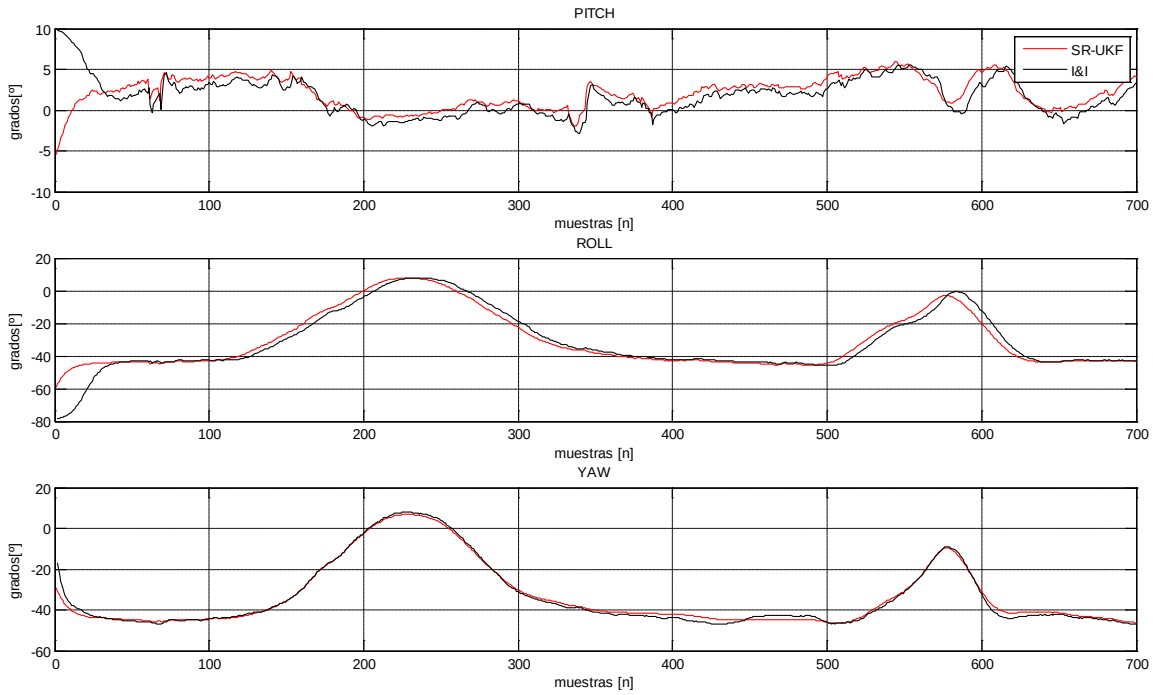


Figura 6. 13 Comparativa entre el método I&I y el filtrado SR-UKF con datos de la tarjeta *Sparkfun* simulados en MATLAB.

Todas las pruebas en MATLAB se realizaron para tener una idea previa y detallada del comportamiento del algoritmo que sería trasladado al DSP en lenguaje C, para sintonizar así los métodos de estimación de una manera más rápida. Los parámetros de cada método se señalan en la tabla 6.3.

Inmersión e Invarianza		Filtro Kalman Unscented	
γ_1	8	L	7
γ_2	0.1	β	2
γ_3	0.001	α	0.6
$\bar{q}$	-2	γ	1.587450786638754
$\bar{\mu}_A$	-2	$W_0^{(m)}$	0.64
a_1	-12.5	$W_0^{(c)}$	3.28
a_2	-1	$W_{1-14}^{(m)} = W_{1-14}^{(c)}$	0.025714285714286
a_3	-80	$\mathbf{S}_0$	$diag\{0.2\}_{7\times7}$
$\bar{q}_\psi$	2.5	$\sqrt{\mathbf{R}^{\mathrm{v}}}$	$diag\{[0.3\ 0.3\ 0.3\ 0.5\ 0.5\ 0.5\ 0.5]\}_{7\times7}$
$\bar{v}_b$	0.001	$\sqrt{\mathbf{R}^{\mathrm{n}}}$	$diag\{[2.5\ 2.5\ 2.5\ 0.5\ 0.5\ 0.5\ 0.5]\}_{7\times7}$
Tiempo de muestreo: 20 [ms]			

Tabla 6. 3 Parámetros de sintonización para la tarjeta de sensores de *Sparkfun*.

6.2 Experimento sobre la plataforma DSP

Para realizar las pruebas de los algoritmos referidos en el DSP se hicieron muestreos previos de la tarjeta de sensores y se guardaron en archivos en formato .dat mediante la herramienta de desarrollo *Code Composer Studio*. Esto con el objeto de realizar los procesos de estimación descritos en esta tesis (I&I y SR-UKF) sobre la tarjeta de desarrollo empleando el mismo muestreo, para después realizar una comparativa más clara entre ambos métodos. La conexión hecha para realizar los experimentos se ilustra en la figura 6.14. Es importante señalar que ambos métodos trabajaron en tiempo real con una frecuencia de muestreo de 20ms, la cual fue determinada por la frecuencia de muestreo más rápida del sensor más lento, que en este caso fue el magnetómetro.

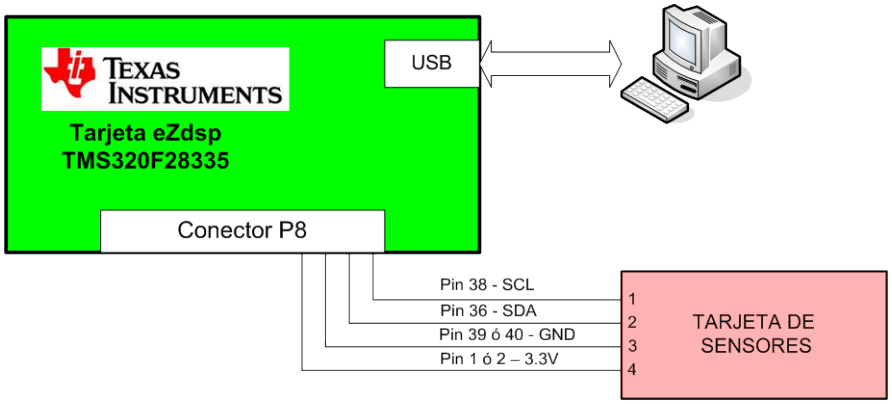


Figura 6. 14 Diagrama de conexión.

Los algoritmos de estimación fueron probados usando tanto la unidad de punto flotante del DSP con el protocolo IEEE 754, y como punto fijo utilizando bibliotecas de

Code Composer Studio, en este caso se utilizó el formato Q20 que tiene una precisión de 0.000000954 y un rango de -2048 a 2047.999999046. En ambos casos se utiliza una longitud de palabra de 32 bits.

En las figuras 6.15 y 6.16 se compara respectivamente el método TRIAD de Matlab contra los métodos de I&I y el filtrado SR-UKF usando punto flotante.

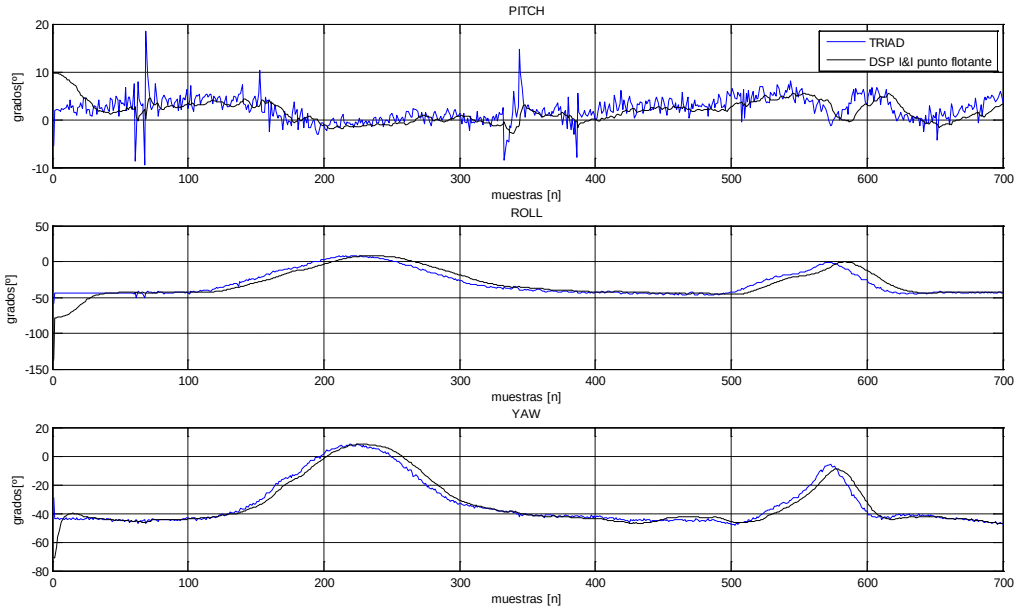


Figura 6. 15 Comparación entre método TRIAD (MATLAB) y el método I&I, en DSP usando punto flotante.

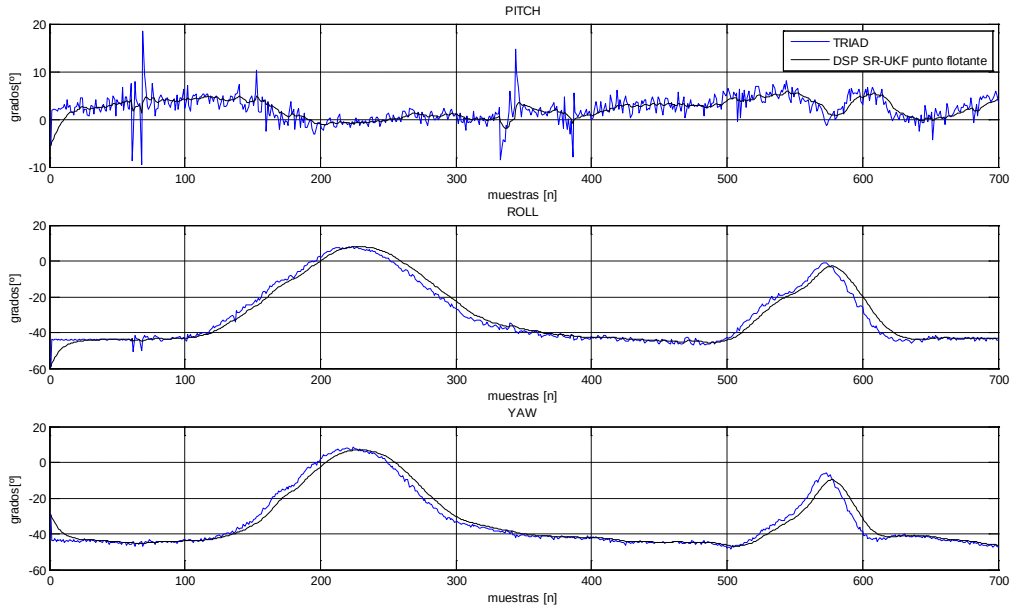


Figura 6. 16 Comparación entre el método TRIAD (MATLAB) y el filtro SR-UKF, en DSP usando punto flotante.

En tanto que en la figura 6.17 se comparan ambos métodos de estimación (I&I y filtro SR-UKF).

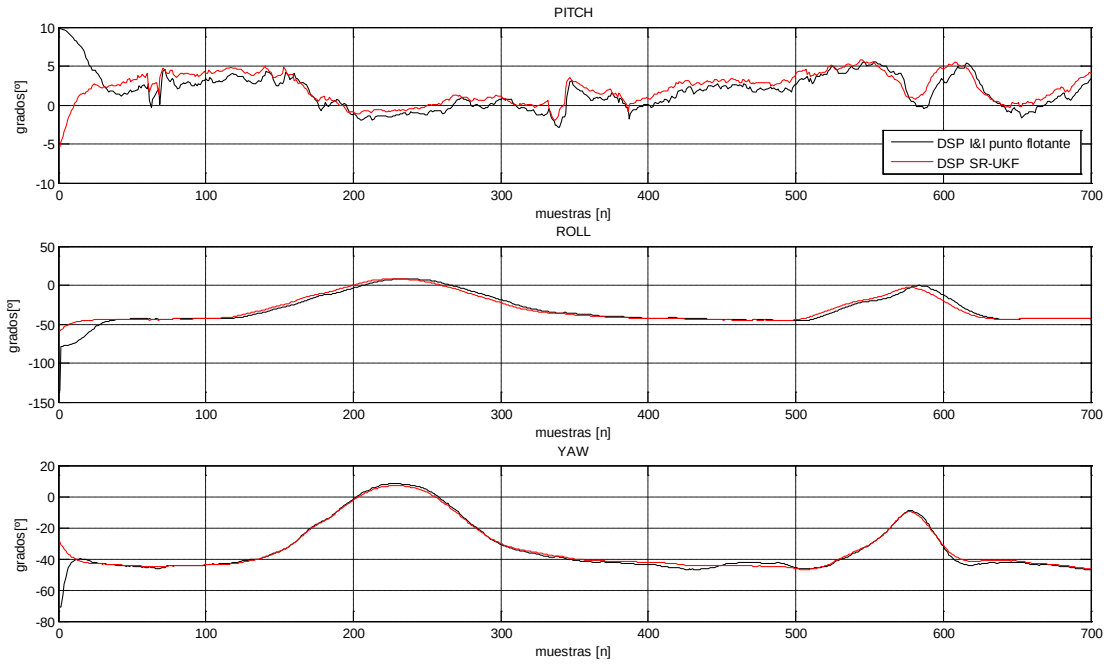


Figura 6. 17 Comparación entre el método de I&I y el filtro SR-UKF, en DSP usando punto flotante.

En las figuras 6.18, 6.19 y 6.20 se realiza lo mismo pero para punto fijo y en las figuras 6.21 y 6.22 se realizan las comparativas de los resultados obtenidos en MATLAB y en la plataforma DSP usando punto flotante y punto fijo respectivamente.

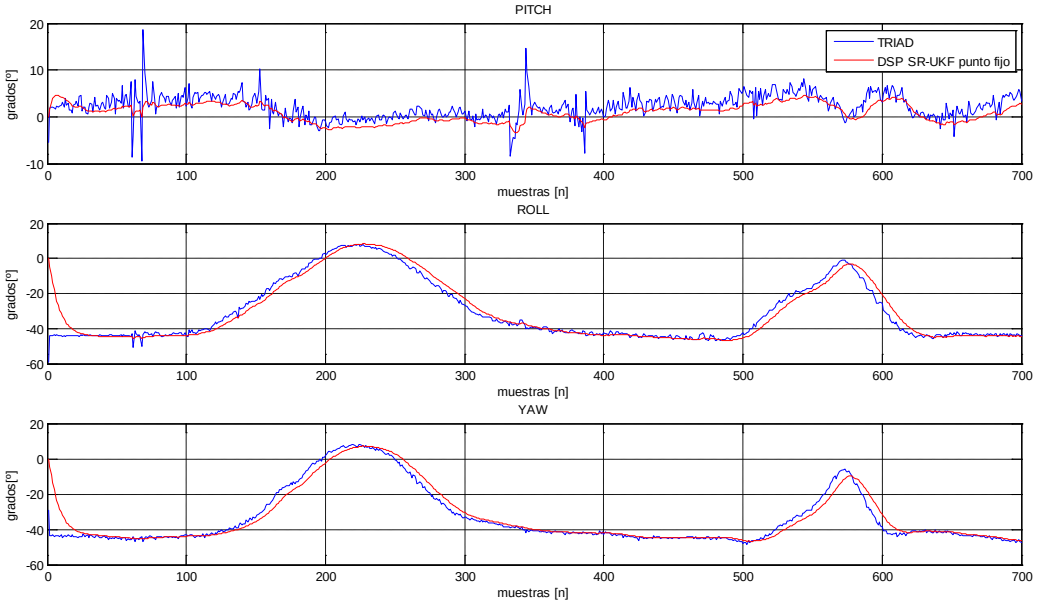


Figura 6. 18 Comparación entre el método TRIAD (MATLAB) y el método I&I, en DSP usando punto fijo (Q20).

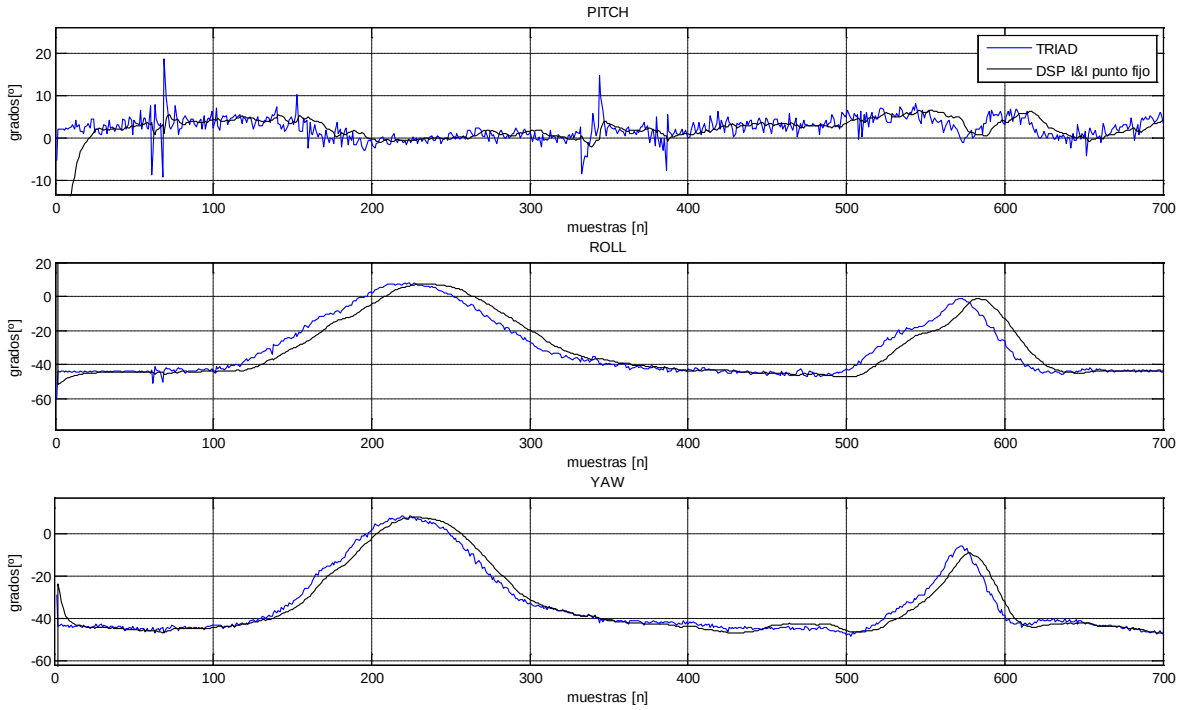


Figura 6. 19 Comparación entre el método TRIAD (MATLAB) y el filtro SR-UKF, en DSP usando punto fijo (Q20).

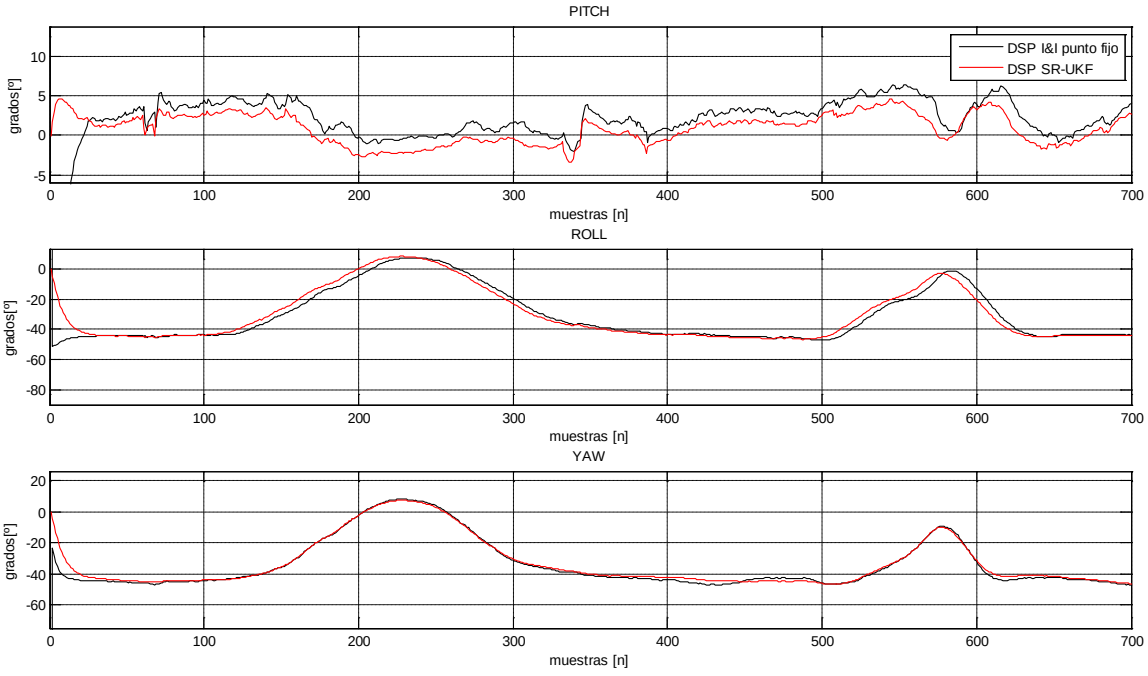


Figura 6. 20 Comparación entre el método de I&I y el filtro SR-UKF, en DSP usando punto fijo (Q20).

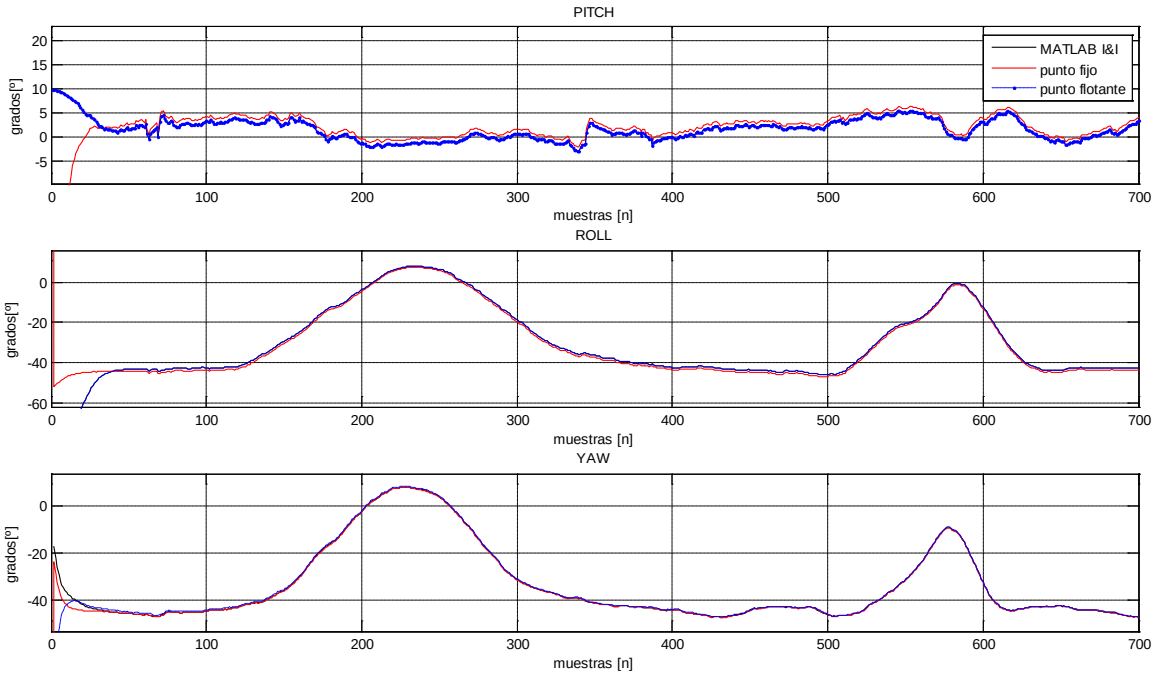


Figura 6. 21 Comparación entre el método de I&I en MATLAB y con DSP tanto en punto flotante como en punto fijo.

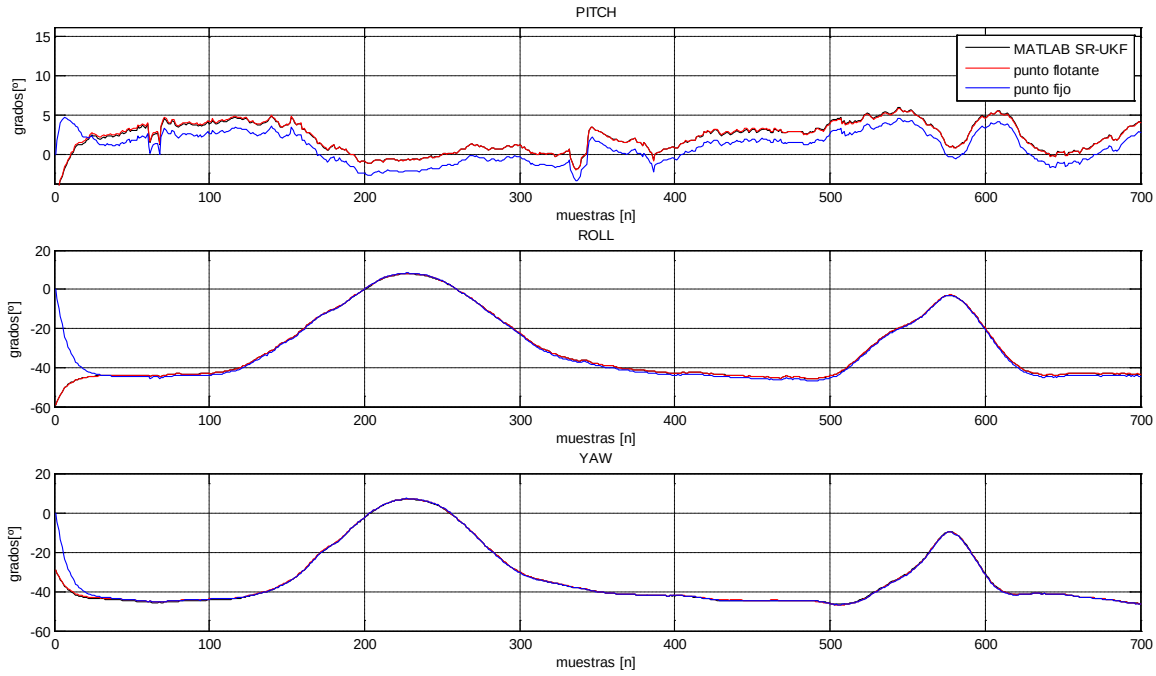


Figura 6. 22 Comparación entre el Filtro SR-UKF en MATLAB y con DSP tanto en punto flotante como en punto fijo.

Para verificar que los tiempos de ejecución de los estimadores sobre el DSP sean adecuados y estén dentro del tiempo de muestreo especificado, a través de *Code Composer Studio* se activó una opción para medir el número de ciclos de reloj que emplea el programa en cada iteración, lo que llevo a los resultados mostrados en la siguiente tabla.

Método	Ciclos de reloj	Tiempo [ms]
Inmersión e Invarianza en punto fijo (Q20)	21,631 – 22,167	0.1442 - 0.1478
Inmersión e Invarianza en punto flotante	12,656 – 13,625	0.0844 - 0.0908
SR-UKF en punto fijo (Q20)	547,728 – 547,825	3.6515 - 3.6522
SR-UKF en punto flotante	491,061 – 519,800	3.2737 - 3.4653

Tabla 6. 4 Tabla de tiempo de ejecución de los algoritmos de estimación.

Finalmente, se muestra también en la siguiente tabla el error cuadrático medio (ECM) de los métodos de filtrado en el DSP contra de los resultados obtenidos en MATLAB, ya que por ahora no fue posible tener acceso a un equipo que permitiera obtener la curva real del movimiento seguido durante el muestreo de los sensores.

Método	Ángulo	ECM [rads]	ECM [°]
Inmersión e Invarianza en punto fijo (Q20)	Cabeceo	0.07546729	4.323957277
	Alabeo	0.08098266	4.639964902
	Guiñada	0.02781173	1.593494927
Inmersión e Invarianza en punto flotante	Cabeceo	0.00646275	0.370288747
	Alabeo	0.08942578	5.123720326
	Guiñada	0.06734253	3.858443178
SR-UKF en punto fijo (Q20)	Cabeceo	0.02764058	1.583688655
	Alabeo	0.08095550	4.638408827
	Guiñada	0.04150120	2.377844108
SR-UKF en punto flotante	Cabeceo	0.00176207	0.100959343
	Alabeo	0.00113182	0.064848811
	Guiñada	0.00424271	0.243089712

Tabla 6. 5 Tabla de error cuadrático medio de los resultados en el DSP comparados con MATLAB.

Capítulo 7

Conclusiones y trabajo futuro

7.1 Conclusiones

En el presente trabajo de tesis se utilizaron e implementaron con éxito en DSP el estimador de Inmersión e Invarianza y también se utilizó e implementó el filtro SR-UKF, ambos métodos orientados a resolver el problema de estimación de la orientación en el satélite HumSAT-México como parte del problema del control de orientación en su primera aproximación para ser validado en una mesa suspendida en aire. A lo largo de la realización de la tesis fue posible darse cuenta de la gran cantidad de herramientas que áreas como la aviación, la robótica móvil y la computación gráfica han generado para el desarrollo espacial. A través de los resultados obtenidos en la implementación se definieron las necesidades de procesamiento y memoria de ambos métodos de estimación, también se compararon los métodos y con base en ello se determinó el desempeño, las ventajas y desventajas de cada método en el tratamiento del problema de la estimación de la orientación.

De este modo, del presente desarrollo de tesis se desprenden las siguientes ventajas y desventajas del método de Inmersión e Invarianza en cuanto al tema de estimación de la orientación:

Ventajas:

- Su elaboración en software es más sencilla en comparación con SR-UKF.
- No requiere de mucha memoria para operar.
- No es necesario disponer de un vector de referencia magnético al momento de realizar la estimación.

Desventajas:

- Emplea un número importante de funciones trascendentes (20 contra solo 3 del SR-UKF), que no son solo circulares sino también hiperbólicas.
- Solo funciona bien ante cambios suaves de aceleración.

En cuanto a las ventajas y desventajas del filtro SR-UKF se tienen las siguientes:

Ventajas:

- Se logran cambios más suaves en los resultados de estimación ante cambios bruscos en los movimientos de orientación.
- La mayoría de sus operaciones se reduce a sumas y multiplicaciones.
- Puede replantearse el filtro para un mayor número de estados sin tener cambios demasiado significativos en el algoritmo de filtrado.

Desventajas:

- El algoritmo es muy extenso y necesita una gran cantidad de memoria de datos.

A partir del análisis de las operaciones de los algoritmos implementados en punto fijo en la plataforma DSP fue posible proponer microcontroladores de menores dimensiones, para que sean empleados en trabajos futuros. Sin embargo, cuando se usa un DSP para realizar operaciones que involucran funciones trascendentes, éstas ejecutan rápidamente debido a que su arquitectura está optimizada para ese propósito, lo cual no sucede con otros microcontroladores.

Por ello, al momento de realizar la implementación en microcontroladores se requerirán de muchas instrucciones para la ejecución de una sola función trascendente lo que a su vez extenderá los tiempos de ejecución y de no obtener el tiempo de ejecución mínimo requerido también requiera ya sea elevar la frecuencia de reloj del microcontrolador (que se traducirá en mayor consumo energético), o bien cambiar de microcontrolador. Por ello, ante el número importante de funciones trascendentes requeridas en el método I&I, resulta que el método SR-UKF hasta ahora presenta la mejor opción para ser implementado incluso en microcontroladores.

Finalmente, a través de la comparación de salidas de señales de sensores realizadas con una IMU comercial fue posible validar los estimadores propuestos y fue posible observar que su desempeño es muy similar. Por lo tanto, una conclusión muy importante de esta tesis es que con la incorporación de estos métodos se pueden emplear sensores de bajo costo en satélites experimentales con buenos resultados.

Una vez agotada la optimización de los métodos de estimación la precisión de apuntamiento satelital dependerá únicamente de la calidad de los sensores utilizados, esto es lo que distingue a un satélite que es capaz de tomar fotografías con precisión de arcos de segundo (1/3600 de grado) de otro que solo requiere apuntar antenas directivas con cierto umbral de precisión, a fin de cuentas es la aplicación, los recursos financieros y

el espacio que se tenga disponible en un satélite lo que determinará la calidad de los sensores a utilizar y sus características de apuntamiento.

7.2 Trabajo futuro

El trabajo desarrollado en esta tesis sirve de punto de partida para empezar a realizar las pruebas de orientación sobre una plataforma de pruebas suspendida en aire para validar los primeros algoritmos de orientación en tres ejes, debido a que ya dispondrá de mediciones confiables de orientación. El trabajo encaminado a realizar estas pruebas experimentales de control de orientación en 3 ejes se está realizando actualmente en el Instituto de Ingeniería, UNAM y se está preparando la mesa suspendida en aire para estas pruebas, figura 7.1.

Como trabajo futuro queda la implementación de los algoritmos de estimación de orientación en microcontroladores de menores dimensiones para determinar cuál es el más balanceado entre capacidad de procesamiento y consumo energético para brindar los mejores resultados en el satélite HumSAT-México.

Figura 7. 1 Mesa suspendida en aire instrumentada con DSP en proceso de integración.

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.

Cabe destacar que los algoritmos presentados en esta tesis aun son susceptibles de depuración mediante el uso de alojamiento dinámico de la memoria de datos, que no se realizó en esta tesis, alguna mejora dentro de las funciones matemáticas programadas como la factorización QR o la descomposición de Cholesky, será necesario el reemplazo del acelerómetro por un sensor de Sol, anexar un GPS y considerarlo dentro del modelo de estimación.

De igual forma, se pueden realizar procesos de normalización dentro de los algoritmos de estimación para aumentar el factor de precisión de los algoritmos haciendo uso del punto fijo ya que en esta tesis solo fue posible lograr un factor de precisión de Q20.

También se puede obtener el rango de precisión máximo factible de obtener mediante los sensores utilizados, debido a que ello determinará el éxito de los experimentos satelitales que demandan orientación fina. Adicionalmente, también se puede realizar la planeación del algoritmo para que opere en órbita con funciones que agreguen un diagnóstico de operación de los sensores utilizados, así como un sistema que permita el ahorro de energía cuando los algoritmos de apuntamiento no sean requeridos (Figura 7.2).

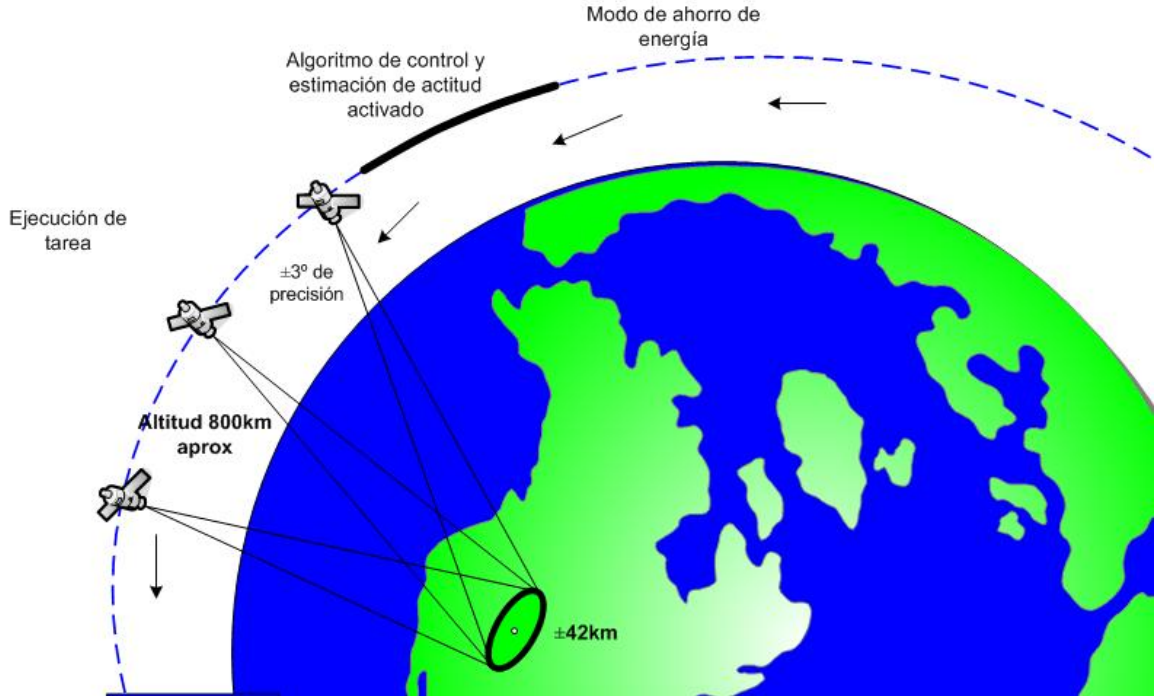


Figura 7. 2 Propuesta de funcionamiento del algoritmo de estimación y control de apuntamiento al instalarlo a bordo de un satélite para ahorrar energía.

APÉNDICE A

A.1 Campos invariantes e Inmersión de sistemas

En esta sección se da la definición de campos invariantes e inmersión de sistemas. Considere el sistema autónomo:

$$\dot{x} = f(x), \qquad y = h(x) \tag{A.1}$$

con el estado $x \in \mathbb{R}^n$ y salida $y \in \mathbb{R}^m$.

Definición A.1 El campo $\mathcal{M} = \{x \in \mathbb{R}^n | s(x) = 0\}$, con s(x) lisa, se dice que es (positivamente) invariante para $\dot{x} = f(x)$ si $s(x(0)) = 0$ e implica que $s(x(t)) = 0$, para toda $t \geq 0$.

Considere ahora el (objetivo) sistema

$$\dot{\xi} = \alpha(\xi), \quad \zeta = \beta(\xi), \tag{A.2}$$

con estado $\xi \in \mathbb{R}^p$ $(p < n)$ y salida $\zeta \in \mathbb{R}^m$.

Definición A.2 El sistema B.2 se dice que esta inmerso en el sistema B.1 si ahí existe un mapeo liso $\pi: \mathbb{R}^p \longmapsto \mathbb{R}^n$ satisfaciendo $x(0) = \pi(\xi(0))$ y $\beta(\xi_1) \neq \beta(\xi_2) \Longrightarrow h(\pi(\xi_1)) \neq h(\pi(\xi_2))$ tal que

$$f\big(\pi(\xi)\big) = \frac{\partial \pi}{\partial \xi}\alpha(\xi) \tag{A.3}$$

y

$$h\big(\pi(\xi)\big) = \beta(\xi) \tag{A.4}$$

para toda $\xi \in \mathbb{R}^p$.

Así, en términos generales, se dice que un sistema Σ_1 está inmerso en un sistema Σ_2 si el mapeo de entrada-salida de Σ_2 sea una restricción del mapeo de entrada-salida de Σ_1, p.e., cualquier respuesta de salida generada por Σ_2 es también una respuesta de salida de Σ_1 para un conjunto restringido de condiciones iniciales.

A.2 Factorización QR

Teorema Sea $A \in \mathcal{M}_{n\times m}(\mathbb{R})$ de rango m, entonces existe una matriz $\mathrm{Q} \in \mathcal{M}_{n\times m}(\mathbb{R})$ que verifica $\mathrm{Q}^{\mathrm{t}}\mathrm{Q} = \mathrm{I}$ (I matriz identidad de $n \times n$) y una matriz triangular superior $R \in \mathcal{M}_{n\times m}(\mathbb{R})$ tal que $\boldsymbol{A} = QR$.

Para realizar la factorización QR se divide a las columnas de la matriz **A** en vectores ***v***

$$\boldsymbol{A} = (\boldsymbol{v}_1|\boldsymbol{v}_2|\dots|\boldsymbol{v}_m)$$

Se utiliza el método de Gram-Schmidt sobre los vectores ***v*** para obtener un conjunto de vectores ortonormales ***u***, de donde mediante inducción matemática se obtendrá la matriz R que corresponde a una matriz de cambio de base dada como

$$\begin{aligned}
\boldsymbol{u}_1 = \boldsymbol{v}_1 &= \|\boldsymbol{u}_1\|\boldsymbol{y}_1 \\
\boldsymbol{v}_2 = \boldsymbol{u}_2 + c_{21}\boldsymbol{u}_1 &= \|\boldsymbol{u}_2\|\boldsymbol{y}_2 + c_{21}\|\boldsymbol{u}_1\|\boldsymbol{y}_1 \\
&\vdots \\
\boldsymbol{v}_k = \boldsymbol{u}_k - c_{kk-1}\boldsymbol{u}_{k-1} - \cdots - c_{k2}\boldsymbol{u}_2 - c_{k1}\boldsymbol{u}_1 & \\
= \|\boldsymbol{u}_k\|\boldsymbol{y}_k + c_{kk-1}\|\boldsymbol{u}_{k-1}\|\boldsymbol{y}_{k-1} - \cdots - \|\boldsymbol{u}_2\|\boldsymbol{y}_2 - c_{k1}\|\boldsymbol{u}_1\|\boldsymbol{y}_1 &
\end{aligned}$$

donde $c_{kj} = \frac{\langle \boldsymbol{v}_k, \boldsymbol{u}_j \rangle}{\langle \boldsymbol{u}_j, \boldsymbol{u}_j \rangle}$ para $j = 1, \dots, m-1$ y $k = 1, \dots, m$.

Y

$$\mathbf{R} = \begin{pmatrix} \|\boldsymbol{u}_1\| & c_{21}\|\boldsymbol{u}_1\| & \dots & c_{m1}\|\boldsymbol{u}_1\| \\ 0 & \|\boldsymbol{u}_2\| & \cdots & c_{m2}\|\boldsymbol{u}_2\| \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|\boldsymbol{u}_m\| \end{pmatrix}$$

Por otro lado se define la matriz **Q** cuyas columnas son vectores *y*

$$\mathbf{Q} = (\boldsymbol{y}_1|\boldsymbol{y}_2|\dots|\boldsymbol{y}_m)$$

donde $\boldsymbol{y}_k = \frac{\boldsymbol{u}_k}{\|\boldsymbol{u}_k\|}, \ \forall k = 1, \dots, m$

Finalmente,

$$\mathbf{QR} = (\boldsymbol{y}_1|\boldsymbol{y}_2|\dots|\boldsymbol{y}_m)\begin{pmatrix} \|\boldsymbol{u}_1\| & c_{21}\|\boldsymbol{u}_1\| & \dots & c_{m1}\|\boldsymbol{u}_1\| \\ 0 & \|\boldsymbol{u}_2\| & \cdots & c_{m2}\|\boldsymbol{u}_2\| \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|\boldsymbol{u}_m\| \end{pmatrix} = \mathbf{A}$$

A.3 Descomposición de Cholesky

Una matriz A simétrica y positiva definida puede ser factorizada de manera eficiente por medio de una matriz triangular inferior y una matriz triangular superior. Para una matriz

no singular la descomposición LU nos lleva a considerar una descomposición de tal tipo $\boldsymbol{A} = LU$; dadas las condiciones de $\boldsymbol{A}$, simétrica y definida positiva, no es necesario hacer pivoteo, por lo que ésta factorización se hace eficientemente y en un número de operaciones la mitad de LU tomando la forma $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$, donde $\boldsymbol{L}$ (la cual podemos "verla" como la raíz cuadrada de $\boldsymbol{A}$) es una matriz triangular inferior donde los elementos de la diagonal son positivos.

Esto se puede ver ilustrado de la siguiente manera

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{m1} & l_{m2} & \cdots & l_{mn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{m1} \\ 0 & l_{22} & \cdots & l_{m2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & l_{mn} \end{bmatrix}$$

donde

$$l_{mm} = + \sqrt{a_{mm} - \sum_{k=1}^{m-1} l_{km}^2}$$

y

$$l_{mn} = \frac{a_{mn} - \sum_{k=1}^{m-1} l_{km} l_{kn}}{l_{mm}}$$

para m<n.

A.4 Problema de mínimos cuadrados

Teorema Sea un sistema inconsistente de ecuaciones, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, se desea encontrar un vector, $\boldsymbol{x}$, que pertenezca a $\mathbb{R}^m$ tal que el error $\|\boldsymbol{\varepsilon}\| = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|$ sea lo menor posible. El vector $\boldsymbol{x}$ es llamado *solución de mínimos cuadrados*.

El problema de mínimos cuadrados puede ser resuelto igualmente a través de factorización QR, como enuncia el siguiente teorema.

Teorema Sea un sistema $\boldsymbol{A}$ que tiene columnas linealmente independientes. Entonces el sistema normal asociado con $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ puede ser escrito como, $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{Q}^T\boldsymbol{b}$.

APÉNDICE B

BUS I2C

Para simplificar la interconexión de dispositivos al microprocesador, Philips desarrolló un sencillo bus bidireccional basado en dos hilos por el que se trasmiten los datos vía serie.

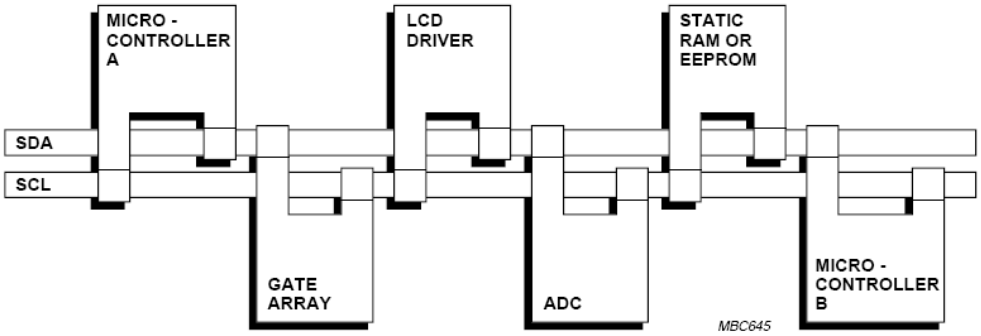


Figura B. 1 Esquema general del bus I2C.

Las líneas SDA y SCL son bidireccionales y están polarizadas a positivo mediante resistencia de "pull-up" de forma que en reposo están a nivel alto.

En el bus exciten maestros (que generan la señal de SCL y controlan la comunicación) y esclavos que responden a peticiones del maestro.

El dato en SDA debe estar estable durante el periodo ALTO de reloj. SDA sólo puede cambiar mientras SCL se encuentre a nivel BAJO.

La excepción a esta regla son condiciones de INICIO y PARO.

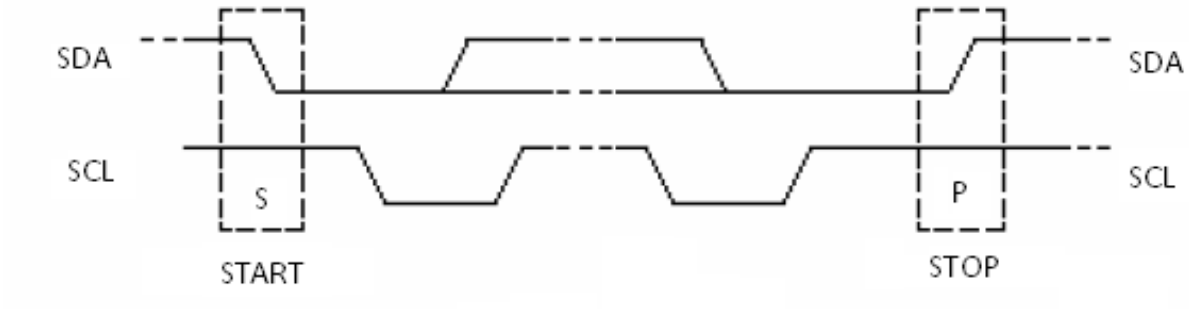


Figura B. 2 Diagrama de condición de INICIO y PARO en el bus I2C.

Trasferencia de datos:

Cada dato que se envía por SDA está formado por 8 bits.

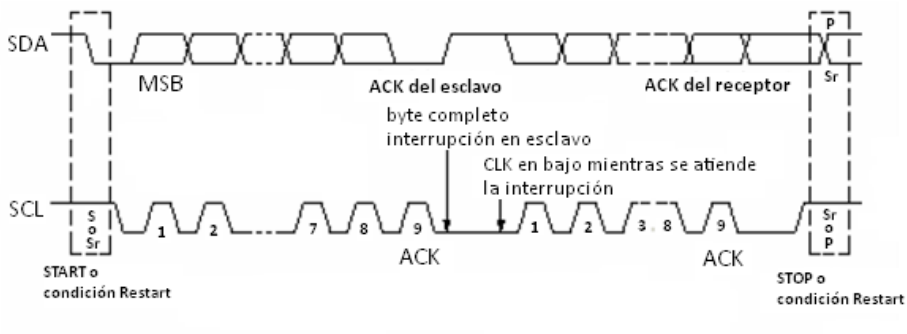


Figura B. 3 Transferencia de datos en el bus I2C.

Tras cada bloque debe recibirse una señal de reconocimiento.

Reconocimiento:

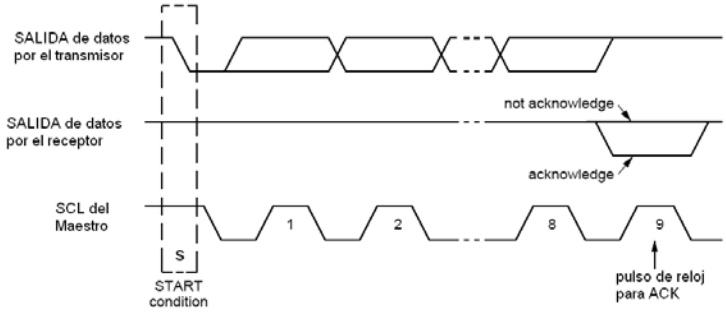


Figura B. 4 Reconocimiento de dispositivos por el bus I2C.

Trasferencia completa:

Tras el envío de inicio, en los siguientes 7 bits se codifica la dirección del dispositivo. El siguiente bit(R/W) indica lectura(1) o escritura(0). Cada 8 bits, el maestro debe esperar una señal de reconocimiento por parte del esclavo. A continuación sigue trasmitiendo la secuencia correspondiente, hasta que finalmente se envía la condición de parada.

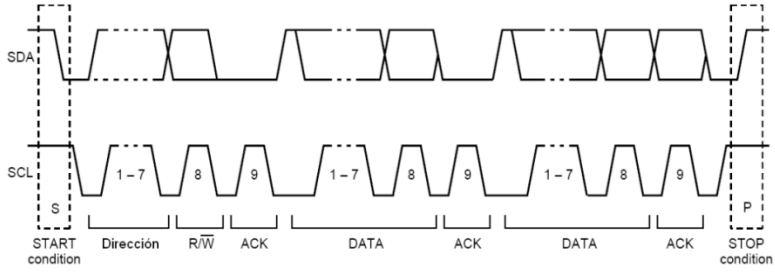


Figura B. 5 Completado de transferencia de datos del bus I2C.

Formatos de envío:

Solo transmitiendo: Donde el dispositivo Maestro transmite a un Esclavo.

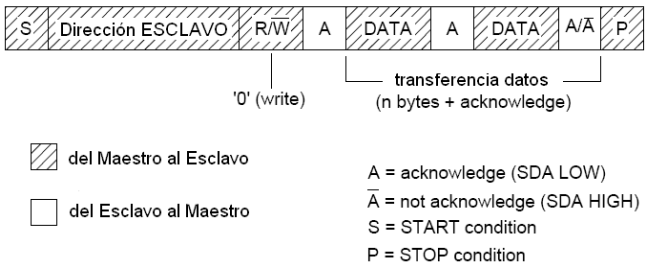


Figura B. 6 Solo transferencia de datos del bus I2C maestro-esclavo.

Solo recibiendo: El dispositivo Maestro lee datos de un esclavo.

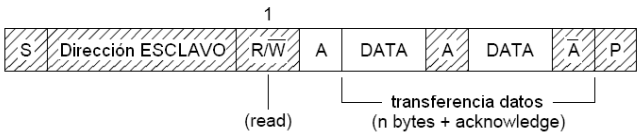


Figura B. 7 Lectura de datos maestro-esclavo.

Combinado: Es posible transmitir y recibir sin fijar la condición de parada, solo de inicio.

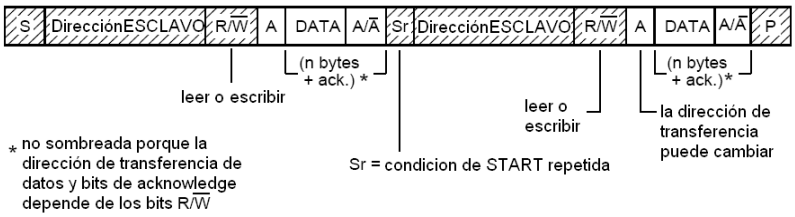


Figura B. 8 Combinación de Lectura y escritura de datos maestro-esclavo.

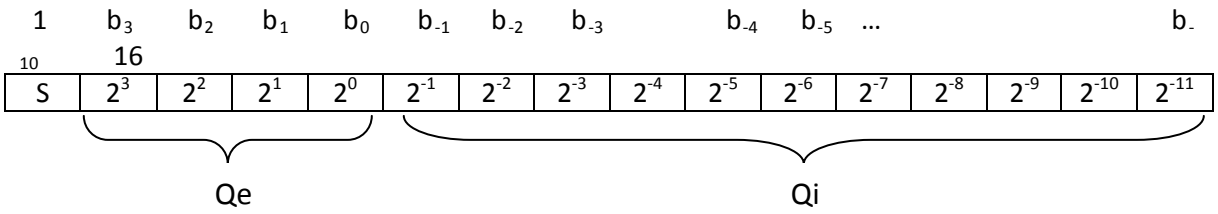
APÉNDICE C

C.1 Punto Fijo

Conversión a formato Qi: Todas las máquinas digitales son capaces de representar en su interior la información basándose en 1's y 0's, básicamente es un sistema binario que posteriormente puede ser interpretado en otra base como la decimal o la hexadecimal, y se pensaría que solo es posible representar números enteros positivos en estas maquinas pero para poder hacer la representación de números reales dentro de ellas es necesario crear arreglos para poder interpretarlos.

Para poder representar un número real se toma la convención de punto fijo llamada formato Qi que es la siguiente:

Supongamos que se tiene una longitud de palabra de 16 bits, de modo que como se ilustra en el recuadro de abajo el 1er bit será usado como bit de signo, 0 para representar positivo y 1 para representar a un número negativo. Los bits restantes serán distribuidos de manera tal que sea posible representar todos los números de un conjunto dado en su parte entera y los espacios restantes se usan para representar al numero decimal considerando que en lugar de potencia de dos positivas se tienen potencias negativas, para que mediante combinaciones de ellas sea posible obtener el numero deseado lo mas próximo posible.



Podemos representar a esta forma mediante la siguiente expresión

$$L = S + Qe + Qi$$

Donde:
L = longitud de palabra
S = bit de signo
Qe = bits para representar a números enteros
Qi = bits para representar a números decimales

Y el número real que se quiere representar está definido como:

$$N = \sum_{k=0}^{Q_e-1} b_k \cdot 2^k + \sum_{k=1}^{Q_i} \frac{b_{-k}}{2^k} - S \cdot 2^{Q_e}$$

C.2 Punto Flotante

El formato de punto flotante es otra forma de representar números reales pero aquí la longitud de palabra se vuelve de 32 bits y está compuesta por tres partes: el bit de signo, el espacio para representar el exponente y el espacio para representar a la mantisa ya que ahora los números se representarán usando la siguiente forma:

$$\text{Número} = \text{Mantisa} \times 10^{\text{exponente}}$$

En el proyecto se adoptó la convención seguida por la IEEE (IEEE 754), es el estándar más extendido para el computo en punto flotante, y es seguido por muchas de las mejoras de CPU y FPU. El estándar define formatos para la representación de números en punto flotante (incluyendo el cero) y valores desnormalizados, así como valores especiales como infinito y NaN, con un conjunto de operaciones en punto flotante que trabaja sobre estos valores. También especifica cuatro modos de redondeo y cinco excepciones.

IEEE 754 especifica cuatro formatos para la representación de valores en punto flotante: precisión simple (32 bits), precisión doble (64 bits), precisión simple extendida (≥ 43 bits, no usada normalmente) y precisión doble extendida (≥ 79 bits, usualmente implementada con 80 bits). Sólo los valores de 32 bits son requeridos por el estándar, los otros son opcionales. Muchos lenguajes especifican qué formatos y aritmética de la IEEE implementan, a pesar de que a veces son opcionales. Por ejemplo, el lenguaje de programación C ahora permite pero no requiere la aritmética de la IEEE (el tipo *float* del lenguaje C de programación es típicamente usado para la precisión simple de la IEEE y el tipo double usa la precisión doble del la IEEE).

Las características más importantes del formato IEEE 754 en su formato simple se citan a continuación:

- Los números en punto fijo normalizados se representan con exponentes desde 1 hasta 254 (el cero y el 255 se utilizan para casos especiales). El exponente está sesgado, siendo el rango de exponentes de (1-127 =) –126 a (254-127 =) 127.
- Un número normalizado debe contener un bit 1 a la izquierda del punto binario correspondiente a la mantisa; este se encuentra implícito, de manera a tener una mantisa efectiva de 24 bits: 1, x x x x x x x x
- Un exponente cero junto con una parte fraccionaria cero representa el cero positivo o negativo, dependiendo del bit de signo.
- Un exponente todo unos junto con una mantisa cero representa, dependiendo también del bit de signo, infinito positivo o negativo. Como en el caso del cero, el infinito se utiliza para redondeo de números muy pequeños o muy grandes, que no se pueden representar con este formato.
- Un exponente cero junto con una parte fraccionaria distinta de cero representa lo que se conoce como un número no normalizado.
- A un exponente de todo unos junto con una parte fraccionaria distinta de cero se le da el nombre de NaN, que significa “no representa un número” (Not a Number), que se emplean para señalar varias condiciones de excepción.

APÉNDICE D

Código en MATLAB

```
% SR-UKF vs Inmersion
clc;
clear;

% ARCHIVOS DE DATOS CRUDOS
xa = load('xa.dat');
xg = load('xg.dat');
xm = load('xm.dat');

ya = load('ya.dat');
yg = load('yg.dat');
ym = load('ym.dat');

za = load('za.dat');
zg = load('zg.dat');
zm = load('zm.dat');

xa_ajustado = xa*9.81/256;
ya_ajustado = ya*9.81/256;
za_ajustado = za*9.81/256;

xg_ajustado = xg*pi/(180*14.375);
yg_ajustado = yg*pi/(180*14.375);
zg_ajustado = zg*pi/(180*14.375);

t=0:0.02:.02*length(xa)-.02;    % tiempo
tf= .02*length(xa)-.02;

%% INMERSION E INVARIANZA

dt = 0.020;
% integrando
n_dif= [0; 0; 0];
n = [0; 0; 0];
n_ant = [0; 0; 0];

g1 = 8;   %gamas
g2 = 0.1;
g3 = 0.001;

a1=-12.5;
a2=-1;
a3=-80;
qb=2.5;
vb=0.001;

% eta_dif=psi_mx;
eta=0;
eta_ant=0;

dB1n = g1*diag(ones(1,3));
dB2n = g2*diag(ones(1,3));
dB3n = g3*diag(ones(1,3));

dB1=a1; dB2=a2; dB3=a3;

uAm = -2;
qm = -2;

%valores iniciales
uA_est_ant = zeros(3,1);
q_est_ant = zeros(3,1);
```

```
r3_est_ant = zeros(3,1);

Sw_ant = zeros(3,3);

psi_est_ant = 0;
qb_est_ant = 0;
v_est_ant = 0;

for i=1:length(xa)

    B1 = n_ant*g1;  %betas
    B1_int(i,:,:)=B1;
    B2 = n_ant*g2;
    B3 = n_ant*g3;

    n_dif = [xa_ajustado(i); ya_ajustado(i); za_ajustado(i)];
    n = n_ant + dt*n_dif;
    n_int(i,:,:)=n;
    n_ant = n;      % integrando por Euler

    Sw = [     0          -zg_ajustado(i)    yg_ajustado(i);        %matriz antisimetrica
          zg_ajustado(i)          0         -xg_ajustado(i);
         -yg_ajustado(i)   xg_ajustado(i)         0          ];

    uA_est = uA_est_ant + dt*(-dB3n*(-r3_est_ant + B1 + uAm*tanh(uA_est_ant + B3)));
    q_est  = q_est_ant  + dt*(-dB2n*(-r3_est_ant + B1 + uAm*tanh(uA_est_ant + B3)));
    r3_est = r3_est_ant + dt*(Sw_ant*(r3_est_ant - B1) + qm*tanh(q_est_ant + B2) + dB1n*(-
r3_est_ant + B1 + uAm*tanh(uA_est_ant + B3)));


    %actualizacion de estados
    uA_est_ant = uA_est;
    q_est_ant = q_est;
    r3_est_ant = r3_est;
    Sw_ant = Sw;

    r3 = r3_est - B1;

    % normaliza
    r3_norm = r3/norm(r3);

    % angulos (rads)
    ipitch(i) = -asin(r3_norm(1));
    iroll(i) = atan2(r3_norm(2),r3_norm(3));

    %calculo directo del YAW mediante la formula
    psi_mx(i) = atan2(-(-zm(i)*sin(iroll(i)) + ym(i)*cos(iroll(i))),...
        (xm(i)*cos(ipitch(i)) + ym(i)*sin(iroll(i))*sin(ipitch(i)) +
zm(i)*cos(iroll(i))*sin(ipitch(i))));

    if i==2
        eta = 0; eta_ant =0;
        psi_est_ant = 0;
        qb_est_ant=0;
        v_est_ant=0;
    end

    eta = eta_ant + dt*psi_mx(i)*1.35;

    B1y = eta*a1;
    B2y = eta*a2;
    B3y = eta*a3;

    % estimacion YAW
    psi_est(i) = psi_est_ant + dt*(sin(iroll(i))/cos(ipitch(i))*yg_ajustado(i) +
cos(iroll(i))/cos(ipitch(i))*zg_ajustado(i) - qb*tanh(qb_est_ant + B2y) ...
        + dB1*(psi_est_ant - B1y - vb*tanh(v_est_ant + B3y)));

    qb_est(i) = qb_est_ant + dt*(- dB2*(psi_est_ant - B1y - vb*tanh(v_est_ant + B3y)));
    v_est(i) = v_est_ant + dt*(- dB3*(psi_est_ant - B1y - vb*tanh(v_est_ant + B3y)));
```

```
    % estimacion YAW
    psi_est_ant = psi_est(i);
    qb_est_ant = qb_est(i);
    v_est_ant = v_est(i);
    eta_ant = eta;

    %estimacion del YAW
    iyaw(i) = (psi_est(i) - B1y);

end

%% FILTRO KALMAN UNSCENTED
% SR-UKF

% variables
Ts = .02;   % 20ms de muestreo
t = Ts:Ts:length(xa)*Ts;

stateDim = 7;
measDim = 7;
x=[0;0;0;0;0;0;0];

S = chol(0.2^2*eye(stateDim))';  %Covarianza Inicial
Q=0.5^2*eye(stateDim);          %Covarianza Proceso
covQ = 0.3;
Q(1,1) = covQ^2;
Q(2,2) = covQ^2;
Q(3,3) = covQ^2;

R=2.5^2*eye(measDim);           %Covarianza de medicion
covR= 0.5;
R(4,4) = covR^2;
R(5,5) = covR^2;
R(6,6) = covR^2;
R(7,7) = covR^2;

sqQ = chol(Q);
sqR = chol(R);
alpha=.6;
beta=2;
Wm=[1-alpha^(2) (alpha^(2))/(2*stateDim)+zeros(1,2*stateDim)];
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);
gama = sqrt(stateDim*alpha^2);


% vectores de referencia de gravedad y magnetico
ref_mag=[0.26923076923, -0.04769230769, 0.238461538461];
ref_acel=[0 0 -9.81];

salida=zeros(7,length(xa));

for i=1:length(xa)

    % datos de los sensores muestreados a 20ms
    data(i,1) = xg_ajustado(i);
    data(i,2) = yg_ajustado(i);
    data(i,3) = zg_ajustado(i);
    data(i,4:7)=TRIAD([xm(i)/1300 ym(i)/1300 zm(i)/1300]',ref_mag',[xa_ajustado(i)
ya_ajustado(i) za_ajustado(i)]',ref_acel');

    % Calculo de los puntos sigma
    A = gama*S;
    xSigmaPts = [x x(:,ones(1,stateDim))+A x(:,ones(1,stateDim))-A];

    % Realiza la actualizacion del estado usando la transformada Unscented
    % (TU)
    numpts=2*stateDim+1;
    xMediaUT=zeros(stateDim,1);
    xSigmaPtsUT=zeros(stateDim,numpts);
    for k=1:numpts
```

```
        xSigmaPtsUT(:,k)=proceso(xSigmaPts(:,k),Ts);
        xMediaUT=xMediaUT+Wm(k)*xSigmaPtsUT(:,k);
    end
    [Temp S] = qr([sqrt(Wc(2))*(xSigmaPtsUT(:,2:numpts)-xMediaUT(:,ones(1,numpts-1)))
sqQ]',0);
    S = chol(S'*S+(Wc(1)^(1/2))*((xSigmaPtsUT(:,1) - xMediaUT(:))*(xSigmaPtsUT(:,1)-
xMediaUT(:))'));

    % Predice las mediciones actuales usando la TU
    ySigmaPts=[xMediaUT xMediaUT(:,ones(1,stateDim))+(gama*S) xMediaUT(:,ones(1,stateDim))-
(gama*S)];
    for k=1:numpts
        ySigmaPtsUT(:,k)=medicion(ySigmaPts(:,k),ref_acel,ref_mag);
        yMediaUT=yMediaUT+Wm(k)*ySigmaPtsUT(:,k);
    end

    for k=1:numpts
        ySigmaPtsUT(:,k)=medicion(xSigmaPtsUT(:,k),ref_acel,ref_mag);
        yMediaUT=yMediaUT+Wm(k)*ySigmaPtsUT(:,k);
    end
    [Temp Sy] = qr([sqrt(Wc(2))*(ySigmaPtsUT(:,2:numpts)-yMediaUT(:,ones(1,numpts-1)))
sqR]',0);
    Sy = chol(Sy'*Sy + (Wc(1)^(1/2))*((ySigmaPtsUT(:,1)-yMediaUT(:))*(ySigmaPtsUT(:,1)-
yMediaUT(:))'));

    Sy = Sy'; % Cambia a matriz triangular inferior
    % Calcula la matriz de ganancias de Kalman
    Pxy = (xSigmaPtsUT(:,1:numpts)-
xMediaUT(:,ones(1,numpts)))*diag(Wc)*(ySigmaPtsUT(:,1:numpts)-yMediaUT(:,ones(1,numpts)))';
    K = (Pxy/(Sy)')/Sy;

    % Actualiza la estimación del estado
    x = xMediaUT + K*(data(i,:)' - yMediaUT);
    x(4:7)=quatnormalize(x(4:7)')';     % normaliza el cuaternion
    salida(:,i) = x;

    % Actualiza el estado de covarianza
    upMat = K*Sy;
    for k=1:measDim
    S = chol(S'*S - upMat(:,k)*upMat(:,k)');
    end
    S = S'; % Cambia a matriz triangular inferior
end


function y = TRIAD(mag, ref_m, acel, ref_a)
s1 = ref_a/norm(ref_a);
s2 = cross(s1,ref_m)/ norm(cross(s1,ref_m));
s3 = cross(s1,s2);
r1 = acel/norm(acel);
r2 = cross(r1,mag)/ norm(cross(r1,mag));
r3 = cross(r1,r2);
Arot = r1*s1' + r2*s2' + r3*s3';  % matriz de rotación
R = Arot;
v1=R(2,3)-R(3,2);
v2=R(3,1)-R(1,3);
v3=R(1,2)-R(2,1);
V = [v1;v2;v3];
cb= (trace(R)-1)/2;
theta=acos(cb);
lambda =V /((v1^2+v2^2+v3^2)^0.5);
eta=cos(theta/2);
epsilon=sin(theta/2)*lambda;
q_ref1=[epsilon;eta];
y=q_ref1;
```

APÉNDICE E

Código en C

E.1 Algoritmo de Inmersión e Invarianza

```
/*     ESTIMADOR INMERSION E INVARIANZA */

#include "DSP2833x_Device.h"
#include "stdlib.h"
#include "stdio.h"
#include "math.h"
//#include "IQmathLib.h"

// GAMAS
#define        g1             8
//#define      g2             0.5
#define        g2             0.1
#define        g3             0.001

// muestreo
#define        dt             0.020

// medias
#define        uAm            -2
#define        qm             -2

#define        PI             3.141592653589793

// angulo YAW
/*#define      dB1            -12.5          // dB1 = a1
#define        dB2            -2             // dB2 = a2
#define        dB3            -20            // dB3 = a3

#define        a1             -12.5          // dB1 = a1
#define        a2             -2             // dB2 = a2
#define        a3             -20            // dB3 = a3
*/
#define        dB1            -12.5          // dB1 = a1
#define        dB2            -1             // dB2 = a2
#define        dB3            -80            // dB3 = a3

#define        a1             -12.5          // dB1 = a1
#define        a2             -1             // dB2 = a2
#define        a3             -80            // dB3 = a3


//#define      qb             0.5
//#define      vb             0.01
#define        qb             2.5
#define        vb             0.001

#define        ajuste 1.35

// variables de tamBuf
#define        tamBuf                700

int16 x_mag[tamBuf],y_mag[tamBuf],z_mag[tamBuf];           // lecturas del Magnetometro
int16 x_acel[tamBuf],y_acel[tamBuf],z_acel[tamBuf]; // acelerometro
int16 x_gyro[tamBuf],y_gyro[tamBuf],z_gyro[tamBuf]; // gyro
```

```
/* funciones aritmeticas */
void tanh2(float a[3], float b[3]);
void escXmatrix(float a, float b[3], float c[3]);
void matrix3Mult(float a[3][3], float b[3], float c[3]);
void sumMatrix(float a[3], float b[3], float c[3]);
void restMatrix(float a[3], float b[3], float c[3]);

void corrimiento4(float a[3], float b[3], float c[3], float d[3]);
void integra(float v_new[3], float v_ant[3], float v_ini[3]);
void copia(float new[3], float old[3]);     // copia de matriz new -> old

void matrizAntisim(float wx, float wy, float wz);
void normaliza (float a[3], float norm[3]);

void angulos(float a[3], float b, float t);

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

float convGrados(float rads);

/* variables */
float  arreglo[20];
float  mult1;
float  mult2;
float  resultado;
int i;   // contador

float n_dif[3], n[3], n_ant[3]={0,0,0};             // eta derivada, eta, eta
anterior(integracion)
float B1[3],B2[3],B3[3];             // betas

float r3[3]={0,0,0}, r3_estdif[3]={0,0,0}, r3_est[3]={0,0,0}, r3_est_ant[3]={0,0,0};
float q_estdif[3]={0,0,0}, q_est[3]={0,0,0}, q_est_ant[3]={0,0,0};
float uA_estdif[3]={0,0,0}, uA_est[3]={0,0,0}, uA_est_ant[3]={0,0,0};

float phi, theta;

/* yaw */
float eta=0, eta_ant=0;
float psi_est=0, psi_est_ant = 0;
float qb_est=0, qb_est_ant = 0;
float v_est=0, v_est_ant = 0;
float psi_mx=0;
float B1y=0, B2y=0, B3y=0;
float psie=0;
float magnet[3]={0,0,0};


/*auxiliares*/
float  aux1[3], aux2[3], aux3[3], aux4[3], aux5[3], gyros[3];


/* Matrices */
float dB1n[3][3] ={{g1,0,0},{0,g1,0},{0,0,g1}};
float dB2n[3][3] ={{g2,0,0},{0,g2,0},{0,0,g2}};
float dB3n[3][3] ={{g3,0,0},{0,g3,0},{0,0,g3}};

float Sw[3][3] = {{0,0,0},{0,0,0},{0,0,0}}, Sw_ant[3][3] = {{0,0,0},{0,0,0},{0,0,0}} ;   //
matriz antisimetrica


/* resultados temporales */
float roll[tamBuf];
float pitch[tamBuf];
float yaw[tamBuf];
```

```
void main (void)
{
	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

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

	i = 0;
	bandera_lectura = 0;		// bandera de lectura
	convAcel(x_acel[i], y_acel[i], z_acel[i], n_dif);
	n_ant[0] = dt*n_dif[0]; n_ant[1]=dt*n_dif[1]; n_ant[2]=dt*n_dif[2];

	for (i=1; i< tamBuf; i++){

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

	convAcel(x_acel[i], y_acel[i], z_acel[i], n_dif);
	convGyro(x_gyro[i-1], y_gyro[i-1], z_gyro[i-1], gyros);

	matrizAntisim(gyros[0], gyros[1], gyros[2]);// obtiene la matriz antisimetrica y la
deposita en Sw(var global)

	// betas
	escXmatrix(g1,n_ant,B1);  // B = g * n  (escalar X matriz)
	escXmatrix(g2,n_ant,B2);
```

```
        escXmatrix(g3,n_ant,B3);

        // n = n_ant + dt*n_dif;   integral
        integra(n, n_ant, n_dif);
        copia(n,n_ant);              // ya esta n (etha)

        // calculo de uA estimada
        sumMatrix(uA_est, B3, aux1);         // uA_m * tanh(uA_est + B3)
        tanh2(aux1, aux2);
        escXmatrix(uAm, aux2, aux1);

        restMatrix(r3_est, aux1, aux2);             // r3_est - uA_m * tanh(uA_est + B3)

        restMatrix(aux2, B1, aux1);          // r3_est - uA_m * tanh(uA_est + B3) - B1

        matrix3Mult(dB3n, aux1, uA_estdif);  // dB3n * ( r3_est - uA_m * tanh(uA_est + B3) -
B1 ) = uA_estdif

        // calculo de q estimada
        sumMatrix(uA_est, B3, aux1);
        tanh2(aux1, aux2);
        escXmatrix(uAm, aux2, aux1);

        restMatrix(r3_est, aux1, aux2);

        restMatrix(aux2, B1, aux1);

        matrix3Mult(dB2n, aux1, q_estdif);


        // calculo de r3 estimada
        sumMatrix(uA_est, B3, aux1);
        tanh2(aux1, aux2);
        escXmatrix(uAm, aux2, aux1);

        sumMatrix(B1, aux1, aux2);

        restMatrix(aux2, r3_est, aux1);

        matrix3Mult(dB1n, aux1, aux3);


        sumMatrix(q_est, B2, aux1);
        tanh2(aux1, aux2);
        escXmatrix(qm, aux2, aux4);

        restMatrix(r3_est, B1, aux1);
        matrix3Mult(Sw, aux1, aux5);

        sumMatrix(aux4, aux3, aux2);
        sumMatrix(aux5, aux2, r3_estdif);

        integra(r3_est, r3_est_ant, r3_estdif);             // integra
        copia(r3_est, r3_est_ant);

        integra(uA_est, uA_est_ant, uA_estdif);             // integra
        copia(uA_est, uA_est_ant);

        integra(q_est, q_est_ant, q_estdif);        // integra
        copia(q_est, q_est_ant);

        ////////////////////
        /// obtiene r3 ////
        restMatrix(r3_est, B1, r3);          // r3 = r3_est - B1;

        normaliza(r3, aux1);

        theta = -(float)asin((double)aux1[0]);
        phi = (float)atan2((double)aux1[1], (double)aux1[2]);

        pitch[i]=theta;
```

```
        roll[i] = phi;

        convMag(x_mag[i], y_mag[i], z_mag[i], aux1);
        psi_mx = atan2(-(-aux1[2]*sin(phi) + aux1[1]*cos(phi)),(aux1[0]*cos(theta) +
aux1[1]*sin(phi)*sin(theta) + aux1[2]*cos(phi)*sin(theta)));

    eta = eta_ant + dt*psi_mx*ajuste;

    B1y = eta*a1;
    B2y = eta*a2;
    B3y = eta*a3;

    psi_est = psi_est_ant + dt*(sin(phi)/cos(theta)*gyros[1] + cos(phi)/cos(theta)*gyros[2]
- qb*tanh(qb_est_ant + B2y) + dB1*(psi_est_ant - B1y - vb*tanh(v_est_ant + B3y)));
    qb_est = qb_est_ant + dt*(- dB2*(psi_est_ant - B1y - vb*tanh(v_est_ant + B3y)));
    v_est = v_est_ant + dt*(- dB3*(psi_est_ant - B1y - vb*tanh(v_est_ant + B3y)));

    // estimacion YAW
    psi_est_ant = psi_est;
    qb_est_ant = qb_est;
    v_est_ant = v_est;
    eta_ant = eta;

    psie = (psi_est - B1y); //*180/PI;   // psi estimada en grados.

        yaw[i] = psie;
        }

        while(1);

}


/* MATRICES */
void escXmatrix(float a, float b[3], float c[3])
{
        c[0] = a*b[0];
        c[1] = a*b[1];
        c[2] = a*b[2];
}

void matrix3Mult(float a[3][3], float b[3], float c[3])
{
        c[0] = a[0][0]*b[0] + a[0][1]*b[1] + a[0][2]*b[2];
        c[1] = a[1][0]*b[0] + a[1][1]*b[1] + a[1][2]*b[2];
        c[2] = a[2][0]*b[0] + a[2][1]*b[1] + a[2][2]*b[2];
}

void sumMatrix(float a[3], float b[3], float c[3])  // a+b=c
{
        c[0] = a[0] + b[0];
        c[1] = a[1] + b[1];
        c[2] = a[2] + b[2];
}

void restMatrix(float a[3], float b[3], float c[3])
{
        c[0] = a[0] - b[0];
        c[1] = a[1] - b[1];
        c[2] = a[2] - b[2];
}


// tangente hiperbolica de matriz 3x1
void tanh2(float a[3], float b[3])
{
        b[0] = (float)tanh((float)a[0]);
        b[1] = (float)tanh((float)a[1]);
        b[2] = (float)tanh((float)a[2]);
}
```

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

void corrimiento4(float a[3], float b[3], float c[3], float d[3])
{
	int i;
	for (i=0; i<3; i++)
	{
		d[i] = c[i];
		c[i] = b[i];
		b[i] = a[i];
	}
}

void integra(float v_new[3], float v_ant[3], float v_ini[3])
{
	v_new[0] = v_ant[0] + dt*v_ini[0];
	v_new[1] = v_ant[1] + dt*v_ini[1];
	v_new[2] = v_ant[2] + dt*v_ini[2];
}

void copia(float new[3], float old[3])      // copia de matriz new -> old
{
	old[0] = new[0];
	old[1] = new[1];
	old[2] = new[2];
}


void matrizAntisim(float wx, float wy, float wz)
{
	/*Sw[0][0] = 0;*/   Sw[0][1] = -wz; Sw[0][2] = wy;
	Sw[1][0] = wz;  /* Sw[1][1] = 0; */   Sw[1][2] = -wx;
	Sw[2][0] = -wy; Sw[2][1] = wx;  /* Sw[2][2] = 0; */
}


void normaliza (float a[3], float norm[3])
{
	float norma;

	norma = sqrt( a[0]*a[0] + a[1]*a[1] + a[2]*a[2] );

	norm[0] = a[0] / norma;
	norm[1] = a[1] / norma;
	norm[2] = a[2] / norma;
}


void angulos(float a[3], float b, float t)
{
	t = (float)asin((double)a[0]);
	b = (float)atan2((double)a[1], (double)a[2]);
}

float convGrados(float rads)
{
	float temp;
	temp = rads*180/PI;
	return(temp);
}


// SENSORES
// convierte y ajusta de los datos del acelerometro a m/s2
void convAcel(float ax, float ay, float az, float gs[3])
{
	float temp;
	temp = ax - 6; // se le resta el zero-bias
	temp = ax;
	gs[0] = temp * 0.0383203125;  // 9.81 / 256
```

```
        temp = ay - 1;
        temp = ay;
        gs[1] = temp * 0.0383203125;

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

// convierte datos del gyro
void convGyro(float gx, float gy, float gz, float ws[3])
{
        float temp;
        temp = gx - 39;
        temp = gx;
        ws[0] = temp * 0.00121414208834;

        temp = gy + 10;
        temp = gy;
        ws[1] = temp * 0.00121414208834;

        temp = gz;
        ws[2] = temp * 0.00121414208834;
}

void convMag(int16 mx, int16 my, int16 mz, float mags[3])
{
        mags[0] = mx;
        mags[1] = my;
        mags[2] = mz;
}


interrupt void Int_timer0(void)
{

        // lectura de sensores

                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);

                x_mag = (datosMag[0] << 8) | datosMag[1];
                y_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
}
```

E.2 Algoritmo del filtro SR-UKF

```
/* FILTRO KALMAN UNSCENTED */

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

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

       - 2 algoritmos de: DCM2quat (con y sin funciones trascendentes)

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

#include "DSP2833x_Device.h"
#include "DSP2833x_I2C_defines.h"             // Macros used for I2C examples.
#include "stdlib.h"
#include "stdio.h"
#include "math.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                700           // tamaño del buffer de datos
#define       PI            3.141592653589793

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


/*** INicializacion del DSP ***/
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 del DSP */


/*** 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 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 alt_dcm2quat(float Mat[3][3], float qc[4]);    // convierte de Matriz de rot a
quaternion usando funciones trascendentes
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 ***************/

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];

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
#define       timer          20000


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

/*****************************************************************************/
/*** PARAMETROS DEL FILTRO DE KALMAN UNSCENTED *******************************/
/*****************************************************************************/

float S[7][7] = { {(0.2),0,0,0,0,0,0},              // MATLAB bFUNC
                            {0,(0.2),0,0,0,0,0},
                            {0,0, (0.2),0,0,0,0},
                            {0,0,0,(0.2),0,0,0},
                            {0,0,0,0,(0.2),0,0},
                            {0,0,0,0,0,(0.2),0},
                            {0,0,0,0,0,0,(0.2)} };

float sqR[7]= {(2.5),(2.5),(2.5),(0.5),(0.5),(0.5),(0.5)};        // MATLAB bFUNC
float sqQ[7]= {(0.3),(0.3),(0.3),(0.5),(0.5),(0.5),(0.5)};        //MATLAB (bFUNC)


float alfa = (0.6);   // factores de ajuste (MATLAB bfunc) (anterior 0.5)
float beta = (2);
```

```
// vectores de peso

float Wm[15] =
{(0.64),(0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),

       (0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),

       (0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286)};

float Wc[15] =
{(3.28),(0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),

       (0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),

       (0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286),(0.025714285714286)};

float sqrtWc1 = (1.811077027627484);        // MATLAB bFUNC
float sqrtWc2 = (0.160356745147455);

float eta = (1.587450786638754);            // factor de escala, MATLAB bFUNC

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

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;

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

int16 cont=0;



void main(void)
{

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

       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, timer);

        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;

        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;


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

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

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

                // 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);


                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];

                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);


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++)             // 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-
pMat(:,k)*upMat(:,k)'); (MATLAB)
}
```

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


        }       // end 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 u[7][7];        // puede quitarse

        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<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 u[21][7];		// puede quitarse

	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);
                }
        }
}

// Conversion alternativa de DCM a cuaternion
// se usan funciones trascendentes

void alt_dcm2quat(float Mat[3][3], float qc[4])
{
        float dif[3];
        float lambda[3];
        float aux1, aux2;

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

        aux1 = modulo(dif);

        divVector(dif, aux1, lambda);

        aux2 = (Mat[0][0] + Mat[1][1] + Mat[2][2] - 1) / 2;
        aux1 = acos(aux2);
        qc[0] = cos(aux1/2);  // escalar cuaternion
        aux2 = sin(aux1/2);

        qc[1] = aux2*lambda[0];       // parte imaginaria
        qc[2] = aux2*lambda[1];
        qc[3] = aux2*lambda[2];

}



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

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

	temp = (az);
```

```
        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);
//      ws[0] = temp*PI/14.375/180;   // g * (PI / 180 / 14.375) = 0.00121414208834
        ws[0] = (temp * (0.00121414208834));

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

        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++;

        // 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
}
```

Bibliografía

[1] http://www.humsat.org/

[2] http://www.genso.org/

[3] **Jiménez, E.** *Subsistemas de estabilización activa y sensores para un simulador satelital.* Tesis de licenciatura UNAM, México, 2009.

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

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

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

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

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

[9] **National Research Council (U.S.).** *The global positioning system: a shared national asset : recommendations for technical improvements and enhancements*. Ed. National Academies Press. EEUU, 1997.

[10] Phoenix Miniature GPS Receiver. Sitio Web, [http://www.weblab.dlr.de/rbrt/GpsNav/Phoenix/Phoenix.html]

[11] SGR-05U - Space GPS Receiver. Sitio Web, [http://www.sst-us.com/shop/subsystems/gps/sgr-05u---space-gps-receiver]

[12] **Meyer, R.** *Elements of Space Technology for aerospace engineers*. Ed. Academic Press, EEUU, 1999.

[13] **Curtis, H.** *Orbital Mechanics for Engineering Students*. Ed. Elsevier. EEUU, 2005.

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

[15] **Kuipers, J.** *Quaternions and rotation sequences.* Ed. Princeton University Press. EEUU, 1999.

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

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

[18] **Córdova, J.** *Control de estabilización para picosatélite universitario*. Tesis de licenciatura UNAM, México, 2008.

[19] **Marins, J. et al.** *An extended Kalman Filter for Quaternion-Based Orientation Estimation Using MARG Sensors*. En *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, Hawaii, EEUU, Noviembre, 2001.

[20] **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.

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

[22] **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.

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

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

[25] **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.

[26] **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.

[27] **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.

[28] **Córdova, J.** *Estimación y control de orientación para el nanosatelite HumSAT-México.* Tesis de maestría UNAM, México, 2011.

[29] **Escobar, L., Psenicka, B. y Molero, M.** *Arquitecturas de DSPs, familias TMS320C54x y TMS320C54XX y aplicaciones.* Facultad de Ingeniería. UNAM, México, 2005.

[30] **Spectrum Digital Incorporated.** *eZdsp F28335 Technical Reference*. Manual técnico, 2007.

[31] **Texas Instruments.** *TMS320F28335/28334/28332, TMS320F28235/28234/28232, Digital Signal Controllers (DSCs) Data Manual*. Documento SPRS439E, Junio 2007.

[32] **Honeywell International Inc.** *Hoja de especificaciones. 3-Axis Digital Compass IC HMC5843*. Febrero 2009.

[33] **Analog Devices, Inc.** *Hoja de especificaciones. Digital Accelerometer: ADXL345*. 2009.

[34] **InvenSense Inc.** *ITG-3200 Product Specification Revision 1.4*. Marzo 2010. PS-ITG-3200A-00-01.4.

[35] **Texas Instruments.** *ARM Processors, ARM processor-based solutions.* Sitio Web [http://www.atmel.com/products/AT91/default.asp?category_id=163&family_id=605].

[36] **Atmel.** *Atmel ARM-based solutions*. Sitio Web [http://www.atmel.com/products/at91/default.asp?source=cms&category_id=163&family_id=605&source=global_nav].

[37] **Xsens Technologies B.V.** *MTi and MTx User Manual and Technical documentation*, August 2008.

[38] **Dawkins, P.** Linear Algebra. [http:// tutorial.math.lamar.edu/terms.aspx].

[39] **Philips Semiconductors.** *The I2C-Bus Specification*. Version 2.1. Enero 2000.