CÁLCULO DE LA ORIENTACIÓN DEL SISTEMA ArduIMU+ V3 MEDIANTE LA IMPLEMENTACIÓN DE UNA MATRIZ DE DIRECCIONES COSENO (DCM)

RAUL HUMBERTO RODRIGUEZ VASQUEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELAS DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA
2014

CÁLCULO DE LA ORIENTACIÓN DEL SISTEMA ArduIMU+ V3 MEDIANTE LA IMPLEMENTACIÓN DE UNA MATRIZ DE DIRECCIONES COSENO (DCM)

RAUL HUMBERTO RODRIGUEZ VASQUEZ

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
M.Sc. ALFREDO RAFAEL ACEVEDO PICÓN

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICO-MECÁNICAS
ESCUELAS DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2014

DEDICATORIA

A mi abuela ROSALBA, la cual siempre quiso lo mejor para mí y me apoyó con todo lo que necesité para llevar a cabo una carrera profesional.

A mi madre VICTORIA, por sus consejos y el gran amor que me ha demostrado durante toda mi vida.

A mi padrastro EFREN por su apoyo incondicional, lo cual hizo más fáciles las cosas para toda mi familia, y por lo tanto para mí.

Al director de proyecto, ALFREDO ACEVEDO, por ser un excelente profesor además del apoyo brindado, sin el cual no hubiese podido llevar a cabo la tesis necesaria para graduarme.

Raúl

TABLA DE CONTENIDO

	Pág.
INTRODUCCIÓN	13
1. DESCRIPCIÓN DEL PROYECTO	15
1.1 FORMULACIÓN DEL PROBLEMA	15
1.2 OBJETIVOS	16
1.2.1 Objetivo General	16
1.2.2 Objetivos Específicos	16
1.3 JUSTIFICACIÓN	16
2. SISTEMA ArduIMU+ V3	18
2.1 HARDWARE DE MEDICIÓN	18
2.1.1 Unidad de Procesamiento de Movimiento MPU-6000.	18
2.1.2 Módulo para la medición de bajos campos magnéticos HMC5883L.	19
2.2 MICROPROCESADOR Atmega328	19
2.2.1 Entorno de desarrollo arduino.	20
2.2.2 Plataforma de creación de prototipos electrónicos Open Source.	20
3. CALIBRACIÓN DEL SISTEMA ArduIMU+ V3	22
3.1 UNIDAD DE PROCESAMIENTO DE MOVIMIENTO MPU-6000	23
3.1.1 Acelerómetro	23
3.1.2 Giroscopio.	25
3.1.3 Sensor de temperatura	27
3.2 MÓDULO PARA LA MEDICIÓN DE BAJOS CAMPOS MAGNÉTICOS HMC5883L	27

4. LA MATRIZ DCM 31
4.1 MODELADO MATEMÁTICO DE LA DCM 31
4.2 LA DCM EN EL SISTEMA ArduIMU+ V3 34

5. EL FILTRO KALMAN 37

6. RESULTADOS 39

7. CONCLUSIONES 45

8. OBSERVACIONES 47

9. DIAGRAMAS DE FLUJO 49
9.1 DIAGRAMA DE FLUJO - ALGORITMO PARA LA ADQUISICIÓN DE LAS SEÑALES 49
9.2 DIAGRAMA DE FLUJO – CONSTRUCCIÓN DE LA MATRIZ DCM 50
9.3 DIAGRAMA DE FLUJO – FILTRADO KALMAN 51

REFERENCIAS BIBLIOGRAFICAS 52

BIBLIOGRAFÍA 54

ANEXOS 56

LISTA DE FIGURAS

	Pág.
Figura 1. Disposición de los ejes de los ICs de medición al interior del ArduIMU+ V3.	22
Figura 2. Marcos referenciales Oxyz y OXYZ.	31
Figura 3. Matriz de Direcciones Coseno (DCM^B).	32
Figura 4. Rotaciones en el espacio euclidiano tridimensional.	33
Figura 5. DCM^B compuesta de múltiples rotaciones.	33
Figura 6. Ángulos Tait-Bryan	34
Figura 7. Regla del triple producto	36
Figura 8. Desplazamiento angular	37
Figura 9. Compilación del algoritmo ArduIMU+ V3 en el software Arduino	39
Figura 10. Resultados de los ángulos Tait-Bryan obtenidos mediante el algoritmo propuesto.	40
Figura 11. Verificación del ángulo de alabeo arrojado por el sistema ArduIMU+ V3	41
Figura 12. Verificación del ángulo de cabeceo arrojado por el sistema ArduIMU+ V3.	42
Figura 13. Verificación del ángulo de guiñada arrojado por el sistema ArduIMU+ V3.	42
Figura 14. Errores medidos en los ángulos de alabeo y guiñada cuando el cabeceo supera su ángulo crítico. El alabeo y guiñada deberían ser cero.	43

LISTA DE TABLAS

Pág.

Tabla 1. Datos de la sensibilidad en el acelerómetro embebido en el ArduIMU+ V3. 24

Tabla 2. Promedio de los resultados de la toma de datos para el acelerómetro. 25

Tabla 3. Datos de la sensibilidad en el giroscopio embebido en el ArduIMU+ V3. 26

Tabla 4. Datos de sensibilidad en el sensor de temperatura embebido en el ArduIMU+ V3. 27

Tabla 5. Datos de la sensibilidad en el magnetómetro embebido en el ArduIMU+V3 28

Tabla 6. Promedio de los resultados de la toma de datos para el magnetómetro. 29

LISTA DE ANEXOS

	Pág.
ANEXO A. MPU-6000.h	56
ANEXO B. HMC5883L. h	59
ANEXO C. MPU6000.cpp	61
ANEXO D. MPU6000.cpp	66
ANEXO E. HMC5883L.cpp	68
ANEXO F. DCM.cpp	72
ANEXO G. ARDUIMU.cpp	78
ANEXO H. Matrix_Algebra.cpp	82
ANEXO I. Output.cpp	84

RESUMEN

TÍTULO: CÁLCULO DE LA ORIENTACIÓN DEL SISTEMA ArduIMU+ V3 MEDIANTE LA IMPLEMENTACIÓN DE UNA MATRIZ DE DIRECCIONES COSENO (DCM)*

AUTOR: RAUL HUMBERTO RODRIGUEZ VASQUEZ**

PALABRAS CLAVES: UAV, ArduIMU+ V3, MEMs, DCM, Ángulos de Navegación Tait-Bryan, Matrices de Rotación, Filtro Kalman.

Descripción: Con el presente trabajo de investigación se desea evaluar la funcionalidad de la tarjeta ArduIMU+ V3 como herramienta para el cálculo de los ángulos Tait-Bryan mediante los cuales se puede conocer la orientación de una aeronave no tripulada con respecto a un marco de referencia global. Para comenzar fue necesario consultar el manual de operación de la misma, adquirir la tarjeta, estudiar el software base de su configuración y realizar las pruebas básicas para lograr la calibración y respectiva compensación de cada uno de los sensores inmersos en el sistema, tales como el magnetómetro, el acelerómetro, el giroscopio y el sensor de temperatura. También se hizo necesario estudiar los procedimientos matemáticos para la obtención de los ángulos de navegación y el filtrado Kalman, además del desarrollo e implementación de las rutinas necesarias para la obtención de la orientación del dispositivo mediante los ángulos Tait-Bryan y su posterior filtrado con el cual se logra brindarle precisión a los ángulos obtenidos de la matriz. Para dar sustento a los resultados obtenidos, se hicieron diferentes pruebas individuales para cada uno de los ángulos arrojados y se dispusieron los respectivos diagramas de flujo, los resultados obtenidos, las conclusiones inferidas y las observaciones para futuros trabajos al final del documento.

---

* Proyecto de grado

** Facultad de Ingenierías Fisico-Mecánicas Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones Director M.Sc. Alfredo Rafael Acevedo Picón

ABSTRACT

TITLE: CALCULATION OF THE ArduIMU SYSTEM'S ORIENTATION THROUGH THE IMPLEMENTATION OF A DIRECTION COSINE MATRIX (DCM)*

AUTHOR: RAUL HUMBERTO RODRIGUEZ VASQUEZ**

KEYWORDS: UAV, ArduIMU+ V3, MEMs, DCM, Tait-Bryan Angles, Rotation Matrix, Kalman Filter.

Description: This research work is to assess the functionality of the ArduIMU+ V3 system as a tool for the calculation of the Tait-Bryan angles of navigation through which we can know the orientation of an Unmanned Aerial Vehicle on a global framework. To begin, it was necessary to refer to the instructions manual of the system, purchase the ArduIMU+ V3 system card, study the basic software of the system's configuration and perform basic tests for calibration and respective compensation for each of the sensors in the system, such as magnetometer, accelerometer, gyroscope and the temperature sensor. It was also necessary to study mathematical procedures to obtain the Tait-Bryan angles of navigation and Kalman filtering, as well as the development and implementation of the necessary routines to obtain the orientation of the device using the Tait-Bryan angles (roll, pitch and yaw) and its subsequent filtering which helps to give precision at angles obtained from the matrix. To give sustenance to the obtained results, were made different individual tests for each thrown angles and will set out the respective flow diagrams, the results obtained, the inferred conclusions, the observations for future works and the complete used bibliography at the end of the document.

---

* Project of grade

** Faculty of Physical Engineering-Mechanical School of Electrical Engineering, Electronics and Telecommunications Director M.Sc. Alfredo Rafael Acevedo Picón

INTRODUCCIÓN

Una Unidad de Medición Inercial o IMU (del inglés, *Inertial Measurement Unit*) es en general un dispositivo electrónico usado para detectar de manera autónoma **la posición** y **la orientación** de un objeto. Es normalmente empleado en los sistemas de navegación de naves espaciales, satélites, misiles dirigidos, transbordadores, barcos, aviones, helicópteros, vehículos no tripulados o cualquier otro móvil en que sea necesario estimar estas mediciones.

Una IMU emplea básicamente una combinación de acelerómetros y giroscopios para medir aceleración y velocidad angular. Posteriormente integra estas variables para obtener el cambio total sobre los valores iniciales lo que permite seguir la posición y orientación del objeto empleando un método conocido como **navegación por estima**. Sin embargo, los dispositivos electrónicos y la integración numérica introducen errores numéricos que no permiten obtener el mismo resultado que el obtenido de la integración simbólica.

Existen algunos sistemas que pueden acoplarse a una IMU para eliminar los errores que limitan su precisión. Típicamente el GPS es usado para corregir el término de deriva en la posición, los sistemas barométricos para la corrección de la altitud, los sensores de gravedad para la vertical local, los sensores de velocidad externos para compensar la deriva por velocidad y un compás magnético para la corrección de la orientación. Hay que notar que la mayoría de estos sistemas tienen sus propios errores, los cuales pueden ser compensados entre ellos.

Implementar una IMU con una DCM como herramienta de cálculo de orientación fue idea de dos consagrados aeromodelistas que encontraron en la DCM la

oportunidad de brindar a los aeromodelos la estabilidad y el control necesarios para realizar acrobacias, William Premerlani y Paul Bizard [1]. El primer borrador de su documento llamado "*Direction Cosine Matrix IMU: Theory*" comprende la teoría relacionada a la implementación de una IMU basada en una DCM para aplicaciones en aeromodelos.

El sistema desarrollado representa la rotación del objeto mediante una matriz de direcciones coseno, emplea la DCM para controlar y navegar el aeromodelo y calcula el valor de los elementos de la matriz de la información medida por un juego de acelerómetros y giroscopios básicamente, además de emplear un GPS como vector de referencia para corregir la deriva en **la posición** y cancelar el error en la **orientación** causado por la deriva de los giroscopios.

Pero ¿qué hay de los sistemas que no cuentan con un GPS?, ¿Podrían igualmente emplear un magnetómetro para obtener la información necesaria? En las secciones posteriores se tratará todo lo relacionado a la implementación de una DCM en un sistema ArduIMU+ V3 empleando un conjunto de magnetómetro, acelerómetro y giroscopio.

1. DESCRIPCIÓN DEL PROYECTO

1.1 FORMULACIÓN DEL PROBLEMA

Existen diferentes sistemas que no cuentan con un GPS o simplemente no pueden obtener información por radiofrecuencia, por lo que resulta conveniente encontrar otro dispositivo que funcione como alternativa para el vector provisto por el GPS. Uno que no presente deriva como principal condición y además trabaje de manera autónoma, que no dependa de datos externos sino que por el contrario infiera la información por sí mismo.

Los magnetómetros se han desempeñado muy bien en el control de **guiñado** para aplicaciones de planeado, por lo que resultan ser una buena opción para reemplazar el GPS. No tienen deriva, su respuesta transitoria es mucho más rápida que la del GPS y no necesitan moverse constantemente para poder inferir el vector de referencia de la información medida, como sucede con el GPS. Y aunque particularmente se necesita de un magnetómetro de 3 ejes para poder medir el norte magnético, actualmente existen varios comercialmente disponibles y a muy bajos precios.

Llevar a cabo este sistema nos permitirá brindar otra opción a la variedad de sistemas de medición inercial existentes, permitiéndonos conocer la orientación de un objeto de manera autónoma y precisa. Sustituir el GPS por un magnetómetro de 3 ejes, brinda a la IMU basada en una DCM las características necesarias para perfilar este sistema como una nueva alternativa para los sistemas de navegación a escala, entre otros.

1.2 OBJETIVOS

1.2.1 Objetivo General

- Proponer una herramienta software que permita calcular la orientación del sistema ArduIMU+ V3 utilizando la matriz de direcciones coseno (DCM).

1.2.2 Objetivos Específicos

- Desarrollar un algoritmo que permita la adquisición de las señales de aceleración lineal, velocidad angular y campo magnético utilizando los sensores de la unidad de medición inercial (MPU6000A) y el magnetómetro (HMC5883L) presentes en el sistema.
- Implementar un segundo algoritmo que procese las mediciones tomadas y las organice convenientemente para obtener la matriz de direcciones coseno (DCM).
- Mejorar el algoritmo ya existente al adicionarle un código que permita realizar un filtrado Kalman sobre los valores de la matriz para dar precisión a los ángulos Tait-Bryan obtenidos (alabeo, cabeceo, guiñada).

1.3 JUSTIFICACIÓN

Los seres humanos siempre están muy pendientes de los recursos que se puedan utilizar, para minimizar situaciones en donde se coloquen en peligro vidas humanas, bien sea por desastres naturales, seguridad de un lugar, reconocimiento de espacios o vigilancia de elementos de gran valor comercial, entre otros.

En determinados casos la presencia de un ser humano es imposible, ya sea por el difícil acceso al terreno o por las condiciones inseguras del lugar, en estos casos

el UAV juega un papel primordial, puesto que solo necesita una condición inicial y un algoritmo con un control óptimo que conlleve al éxito de la misión, esto con el fin de proteger las vidas humanas.

La base de todos los proyectos es el conocer los elementos con los que se realiza su ejecución. Por lo tanto, con este trabajo de investigación se pretende obtener con precisión los ángulos que brindan la orientación de un vehículo no tripulado, además de conocer la usabilidad y flexibilidad de la tarjeta ArduIMU+ V3. (en forma cualitativa) para implementar diferentes algoritmos que permitan obtener la posición en un trabajo posterior.

2. SISTEMA ArduIMU+ V3

El sistema ArduIMU+ V3 pertenece a la familia de dispositivos que pueden funcionar como pilotos automáticos, empleados para el control de navegación de vehículos aéreos no tripulados o UAVs (del inglés, Unmanned Aerial Vehicles). Estos sistemas pueden hacer funcionar estos vehículos aéreos de manera autónoma partiendo de una condición inicial y sin la necesidad de que un operador los controle desde tierra mediante un radiocontrol, ya que cada sistema emplea una plataforma de programación abierta basada en software y hardware flexibles y fáciles de usar.

Esta reciente versión de una IMU se lleva a cabo sobre una plataforma de hardware de medición establecida en un circuito impreso junto a un microprocesador Arduino-compatible Atmega328 y un entorno de desarrollo de código abierto. El hardware de medición consiste en una unidad de procesamiento de movimiento MPU-6000 y un módulo para la medición de bajos campos magnéticos HMC5883L, mientras el entorno de desarrollo implementa el lenguaje de programación *Processing/Wiring* y el cargador de arranque (boot loader) que corre en el circuito impreso.

2.1 HARDWARE DE MEDICIÓN

2.1.1 Unidad de Procesamiento de Movimiento MPU-6000. La unidad de procesamiento de movimiento MPU-6000 de Invensense consta principalmente de un procesador digital de movimiento (DMP) que toma los datos de dos sistemas micro electromecánicos (MEMS) de tres ejes embebidos en la unidad. El primero, un acelerómetro que arroja resultados relacionados con la aceleración lineal a lo

largo de cualquiera de los 3 ejes de desplazamiento y el segundo, un giroscopio que brinda información de la velocidad angular alrededor de cualquiera de los 3 ejes.

2.1.2 Módulo para la medición de bajos campos magnéticos HMC5883L. El módulo HMC5883L de *Honeywell* es un módulo multi-chip que cuenta con un sensor magneto-resistivo de la serie de alta resolución HMC118X para medir campos magnéticos muy bajos y un circuito integrado para aplicaciones específicas, en este caso, un ASIC de amplificación. La tecnología magneto-resistiva anisótropa (AMR) con la que cuenta este módulo, provee ciertas ventajas en comparación con otros sensores magnéticos como sensibilidad y linealidad en los ejes de precisión, además de la capacidad de medir dirección y magnitud de campos magnéticos desde el orden de algunos mili-gauss hasta los 8 gauss.

2.2 MICROPROCESADOR Atmega328

El microprocesador Atmega328 a 16MHz de *Atmel* es un microcontrolador CMOS de 8-bits diseñado con características de bajo consumo, permitiendo al diseñador del sistema optimizar la velocidad de procesamiento a cambio del consumo de energía. Además de contar con un entorno de desarrollo libre que se puede descargar gratuitamente, este IC puede ser programado a través de cualquiera de los puertos de comunicación serial que tiene integrado el sistema para su programación.

Al ser Arduino-compatible, el microprocesador puede ser programado mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing), comúnmente empleado para desarrollar objetos interactivos autónomos o ser conectado a diferentes tipos de software instalados en un computador (Flash, Processing, MaxMSP).

2.2.1 Entorno de desarrollo arduino. El entorno de desarrollo Arduino consta de un editor de texto para escribir código, un área de mensajes, una consola de texto, una barra de herramientas con botones para funciones comunes y una serie de menús que permiten conectarse al hardware Arduino a velocidades de 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200 baudios.

Este software se encuentra fácilmente en internet, de donde puede ser descargado e instalado sin ningún costo para desarrollar programas o cargar algunos ya existentes en las diferentes plataformas de creación de prototipos electrónicos que se basen en código abierto.

2.2.2 Plataforma de creación de prototipos electrónicos Open Source. La plataforma de creación de prototipos electrónicos es un proyecto de la comunidad de DIYers a nivel mundial que trabajan sobre código abierto en drones basados en el modelo de Matriz de Direcciones Coseno de Bill Premerlani [1]. Esta plataforma brinda el código para el ArduIMU+ V3 y otros sistemas arduino basados en Unidades de Medición Inercial (IMU) o Sistemas de Referencia de Inclinación y Rumbo o AHRS (del inglés, Attitude and Heading Reference Systems).

Algunos manuales para el manejo del sistema ArduIMU+ V3, grupos de discusión y lógicamente el código para este proyecto se pueden encontrar en el sitio web oficial de la plataforma [2]. El código es descargable y cuenta con alrededor de una decena de revisiones y mejoras, de las cuales extraeremos las librerías para el MPU-6000 y el HMC5883L, y los comandos básicos para la comunicación del MPU-6000 con el entorno de desarrollo arduino.

Las secciones de código extraído de la plataforma de creación de prototipos se encuentran dispuestos en los anexos 1-3 de la siguiente forma:

- Anexo 1. MPU6000.h – Librería
- Anexo 2. HMC5883L.h – Librería

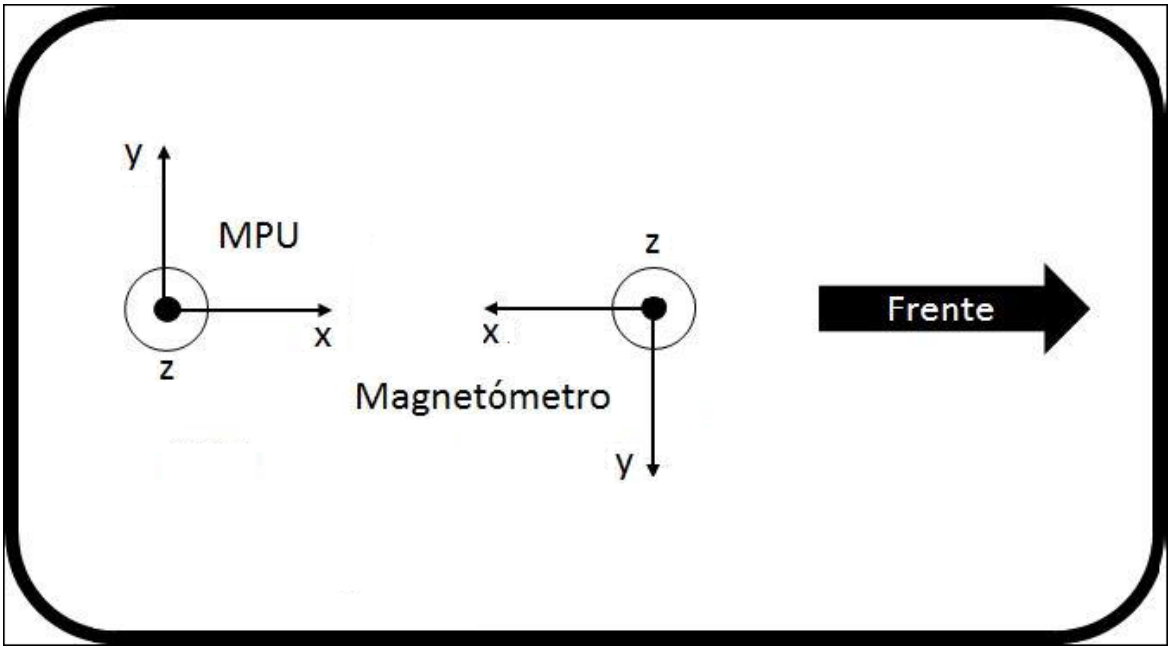
- Anexo 3. MPU6000.cpp – Parte 1.

Además, se pueden encontrar decenas de aeromodelos desarrollados por la misma comunidad de DIYers o Do It Yourselfers para su análisis y posterior estudio en el sitio web de la comunidad, llamado DIY DRONES [3].

3. CALIBRACIÓN DEL SISTEMA ArduIMU+ V3

El hardware de medición del sistema ArduIMU+ V3, el cual está conformado por la unidad de procesamiento de movimiento MPU-6000 y el módulo para la medición de bajos campos magnéticos HMC5883L, tiene la siguiente disposición de ejes para cada circuito integrado.

Figura 1. Disposición de los ejes de los ICs de medición al interior del ArduIMU+ V3.



Fuente: Autores.

El primer paso al emplear dos o más módulos de medición es alinear sus ejes de coordenadas. Para esto se debe tomar la disposición de los ejes del acelerómetro como el sistema de coordenadas de referencia, posteriormente identificar la posición relativa de los sistemas de coordenadas del magnetómetro y el giroscopio con respecto a la posición del acelerómetro y finalmente mediante la variable

SENSOR_SIGN[*eje],* invertir los ejes de los sistemas que lo necesiten, de tal forma que todos coincidan entre sí.

Ya que contamos con un sistema de 9 grados de libertad, debemos compensar el offset de cada una de las 9 salidas arrojadas por el sistema considerando de manera independiente la sensibilidad de cada uno de las unidades de medición que conforman el sistema, y así lograr mayor precisión en el valor medido en cada una de las salidas del sistema en estado estable.

De entre los diferentes factores que se encuentran en la hoja de datos de cada IC y que definen y afectan la sensibilidad de cada unidad de medición, los que emplearemos por ser los más importantes son el Rango de Escala Completa (*Full-Scale Range)* y el Factor de Escala (*Sensitive Scale Factor)* para el acelerómetro y el giroscopio, así como la Ganancia (Sensitive Gain) y la Resolución Digital (Digital Resolution) para el magnetómetro.

3.1 UNIDAD DE PROCESAMIENTO DE MOVIMIENTO MPU-6000

El MPU-6000 está compuesto por un acelerómetro de 3 ejes, un giroscopio de 3 ejes y un sensor de temperatura. Además de esto, este IC se alimenta de 3.3 [V], se comunica mediante la interfaz SPI y cada valor de salida consta de 16 bits dispuestos en 2 bytes.

3.1.1 Acelerómetro. Ya que el acelerómetro del sistema ArduIMU+ V3 se fundamenta en tecnología MEMS, toma sus datos por intermedio de un transductor que convierte tres señales mecánicas en tres señales eléctricas ortogonales, dicho de otra forma, este componente propone un valor de aceleración para cada uno de los 3 ejes del dispositivo a partir de los cambios

internos en la trasferencia de calor causados por aceleraciones sobre las moléculas de gas en el IC embebido.

Una vez tomados los ejes del acelerómetro como ejes de referencia y dispuestas las librerías y los comandos básicos para el MPU-6000 en el entorno de desarrollo arduino, se debe revisar la sensibilidad en la hoja de datos del MPU-6000 [4] para encontrar el valor del Rango de Escala Completa y el Factor de Escala que más convengan al acelerómetro en el proyecto, y de esta forma poder calcular su offset y compensarlo posteriormente.

Tabla 1. Datos de la sensibilidad en el acelerómetro embebido en el ArduIMU+ V3.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		*g*	
	AFS_SEL=1		±4		*g*	
	AFS_SEL=2		±8		*g*	
	AFS_SEL=3		±16		*g*	
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*	
	AFS_SEL=1		8,192		LSB/*g*	
	AFS_SEL=2		4,096		LSB/*g*	
	AFS_SEL=3		2,048		LSB/*g*	

Fuente: Hoja de datos del MPU-6000 [4].

Debido a que el acelerómetro mide la fuerza inercial, y tal fuerza puede ser causada por la gravitación y la aceleración del movimiento del dispositivo, para este proyecto se emplea el Rango de Escala Completa de ±8 g lo que conlleva a un Factor de Escala de 4096 LSB/g. Esto quiere decir que en estado estable y con el eje Z perpendicular a la superficie de la tierra, la salida del acelerómetro en el eje Z debe arrojar un valor de 4096 LSB debido a la gravedad, y 0 LSB para los ejes X y Y respectivamente.

Una vez reemplazado el valor del Rango de Escala en el código tomado de la plataforma de creación de prototipos, se procede a la toma de datos para hallar el

valor del offset para cada uno de los ejes del acelerómetro, de los cuales se obtiene el siguiente promedio de valores medidos en estado estable para cada uno de los ejes del acelerómetro.

Tabla 2. Promedio de los resultados de la toma de datos para el acelerómetro.

Promedio de LSB/g	Eje Negativo	Eje Positivo	Offset	Factor de Escala experimental
Eje X	-3972 LSB	4205 LSB	118 LSB	4090 LSB/g
Eje Y	-4087 LSB	4057 LSB	-15 LSB	4072 LSB/g
Eje Z	-4709 LSB	3633 LSB	-538 LSB	4171 LSB/g

Una vez calculado manualmente el Offset y el Factor de Escala Experimental para cada eje del acelerómetro, se define a continuación una ecuación que además de compensar el offset nos permite obtener un valor muy aproximado a los 4096 LSB cuando cualquiera de los 3 ejes del dispositivo es sometido únicamente a la fuerza de gravedad. Esta ecuación se emplea en el código llevado a cabo para la compensación del dispositivo como se puede observar en la segunda parte del ANEXO 3, encargada de la compensación.

$$accel\,[eje] = (accel.eje\ -\ offset) * \frac{Factor\ de\ Escala\ Teórico}{Factor\ de\ Escala\ Experimental}\ [LSB] \quad (1)$$

3.1.2 Giroscopio. Como el acelerómetro del sistema ArduIMU+ V3, el giroscopio de este sistema también se fundamenta en tecnología MEMS. En este caso, el giroscopio toma los datos por intermedio de un sensor que basa su funcionamiento en la conservación del momento angular y el efecto Coriolis, con los cuales mide la velocidad angular a la que rota el dispositivo sobre sus ejes cartesianos en un determinado intervalo de tiempo y por consiguiente, al giro al cual se ve sometido el cuerpo.

Tabla 3. Datos de la sensibilidad en el giroscopio embebido en el ArduIMU+ V3.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		º/s	
	FS_SEL=1		±500		º/s	
	FS_SEL=2		±1000		º/s	
	FS_SEL=3		±2000		º/s	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(º/s)	
	FS_SEL=1		65.5		LSB/(º/s)	
	FS_SEL=2		32.8		LSB/(º/s)	
	FS_SEL=3		16.4		LSB/(º/s)	

Fuente: Hoja de datos del MPU-6000 [4].

Debido a que el giroscopio es menos sensible a los movimientos mecánicos lineales, se decidió emplear el mayor Factor de Escala en la hoja de datos del Giroscopio, 131 LSB/(º/s), el cual brinda una mayor precisión del giro al cual se ve sometido el dispositivo. Con esto se busca suavizar los errores y obtener un mejor estimado de la orientación del dispositivo del que podríamos obtener usando únicamente el acelerómetro y el magnetómetro.

El offset de este dispositivo depende de la lectura inicial, y según la hoja de datos del MPU-6000 debe ser cero en estado estable, pero para obtener una lectura más acertada se creó una rutina de inicialización que genera un promedio de los datos leídos en estado estable para cada uno de los ejes de coordenadas del giroscopio. Esta rutina se encuentra detallada en la segunda parte del ANEXO 3, junto con la ecuación que representa el mejor estimado del valor de giro del dispositivo en cada eje de coordenadas.

$$gyro\ [eje] = SENSOR_SIGN[eje] * (gyro.eje\ -\ offset)\ \ [LSB] \quad (2)$$

3.1.3 Sensor de temperatura

Tabla 4. Datos de sensibilidad en el sensor de temperatura embebido en el ArduIMU+ V3.

PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
TEMPERATURE SENSOR						
Sensitivity	Untrimmed		340		LSB/ºC	
Temperature Offset	35ºC		-521		LSB	

Fuente: Hoja de datos en el MPU-6000 [4]

Al igual que los valores de salida en el acelerómetro y el giroscopio, el valor de salida de este sensor consta de 16 bits dispuestos en 2 bytes. Además de esto, trabajando en base a las especificaciones de sensibilidad de este sensor podemos conocer la temperatura que experimenta el sistema al construir la siguiente ecuación que podrá verse implementada en la segunda parte del ANEXO 3, junto a los otros valores compensados del MPU-6000.

$$tempt = \frac{temp}{340} + 36.53\ [°C] \quad (3)$$

3.2 MÓDULO PARA LA MEDICIÓN DE BAJOS CAMPOS MAGNÉTICOS HMC5883L

Es importante notar que a pesar de que el código que se encuentra en la plataforma de creación de prototipos tiene los comandos básicos para la comunicación del HMC5883L con el entorno de desarrollo arduino, este código tiene el problema de que cada intervalo de tiempo se repiten los 3 valores de las salidas arrojadas por el módulo. Esto se debe a que este código emplea un modo de operación de medición continua, acumulando datos en el registro de datos de salida del HMC5883 más rápido de lo que el microprocesador los puede leer.

Para solucionar este impase, se decide hacer algunas variaciones en el código encontrado en la plataforma y emplear el modo de operación de medición simple o Single-Measurement Mode, en el cual el magnetómetro hace una nueva medición de datos únicamente hasta que los anteriores son leídos del registro de datos de salida por el microprocesador. La información relacionada a los modos de operación del magnetómetro pueden ser encontrados en la hoja de datos del HMC5883L [5].

Tabla 5. Datos de la sensibilidad en el magnetómetro embebido en el ArduIMU+V3

GN2	GN1	GN0	Recommended Sensor Field Range	Gain (LSb/ Gauss)	Digital Resolution (mG/LSb)	Output Range
0	0	0	± 0.88 Ga	1370	0.73	0xF800–0x07FF (-2048–2047)
0	0	1	± 1.3 Ga	1090 (default)	0.92	0xF800–0x07FF (-2048–2047)
0	1	0	± 1.9 Ga	820	1.22	0xF800–0x07FF (-2048–2047)
0	1	1	± 2.5 Ga	660	1.52	0xF800–0x07FF (-2048–2047)
1	0	0	± 4.0 Ga	440	2.27	0xF800–0x07FF (-2048–2047)
1	0	1	± 4.7 Ga	390	2.56	0xF800–0x07FF (-2048–2047)
1	1	0	± 5.6 Ga	330	3.03	0xF800–0x07FF (-2048–2047)
1	1	1	± 8.1 Ga	230	4.35	0xF800–0x07FF (-2048–2047)

Fuente: Hoja de datos del HMC5883L [5].

Es bien sabido que este sensor le permite al sistema conocer el campo magnético de la tierra expresado en los 3 ejes cartesianos del dispositivo, y es de vital importancia para la estabilización y navegación de cualquier aeromodelo que no posea un GPS ya que permite encontrar el ángulo entre el dispositivo y el vector definido como norte magnético. No obstante para que los valores arrojados por

este dispositivo sea fiables, cada una de las 3 salidas deben ser igualmente compensadas.

Ahora bien, para que el nuevo código de comunicación del módulo con el entorno de desarrollo arduino sea viable y bien compensado, es menester elegir convenientemente la sensibilidad del magnetómetro de la hoja de datos del HMC5883 [5], la cual debe ser primeramente analizada de manera experimental debido a que el campo magnético de la tierra no es el mismo en todas las locaciones. En este caso, se ha considerado mantener la ganancia establecida por defecto en el dispositivo para no incurrir en su saturación.

Aprovechando la función de autotest para corrientes positiva y negativa que posee este módulo, se llevó a cabo una rutina que permite calcular el valor promedio del campo magnético positivo y negativo. Además, una vez reemplazado el valor de ganancia en el código generado, se procede a la toma de datos para hallar el valor del offset para cada uno de los ejes del magnetómetro, de los cuales se obtiene el siguiente promedio de valores medidos en estado estable para cada uno de sus ejes.

Tabla 6. Promedio de los resultados de la toma de datos para el magnetómetro.

Promedio de LSB/Gauss	Eje Negativo	Eje Positivo	Offset	Ganancia experimental
Eje X	-1316.5 LSB	1239.5 LSB	-38.5 LSB	1278.05 LSB/Gauss
Eje Y	-1313.55 LSB	1073.05 LSB	-120.25 LSB	1193.3 LSB/Gauss
Eje Z	-1284.58 LSB	1141.58 LSB	-71.5 LSB	1213.08 LSB/Gauss

Una vez calculado manualmente el offset y la ganancia para cada eje del magnetómetro, se define a continuación una ecuación que nos permite compensar el offset. Esta ecuación se emplea en el código llevado a cabo para la compensación del dispositivo como se puede observar en el ANEXO 4, encargado de la compensación.

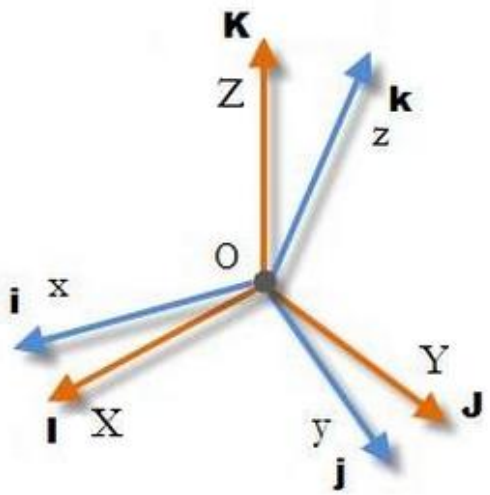
$$mag[eje] = SENSOR_SIGN[eje] * \left(\frac{Ganancia\ Experimental}{Ganancia\ por\ autotest} * mag.eje - offset\right) [LSB] \quad (4)$$

4. LA MATRIZ DCM

4.1 MODELADO MATEMÁTICO DE LA DCM

En términos generales, la cinemática de orientación trata de calcular la orientación relativa del sistema de coordenadas Oxyz en relación con el sistema de coordenadas global OXYZ, donde los vectores unitarios **i**, **j** y **k**, co-direccionales al sistema ArduIMU+ V3, tienen el mismo origen fijo O que los vectores unitarios **I**, **J** y **K,** co-direccionales al marco de referencia global, tal y como se muestra en la Figura 2.

Figura 2. **Marcos referenciales Oxyz y OXYZ.**



En base a la relación existente entre estos dos sistemas de coordenadas, se puede concluir que cada uno de los vectores unitarios globales **I**, **J** y **K** pueden ser expresados empleando los vectores unitarios **i**, **j** y **k**, como el conjunto de las proyecciones de cada vector unitario global sobre los ejes del marco de referencia del cuerpo.

$$\mathbf{I}^{B} = \{\, \mathbf{I.i},\ \mathbf{I.j},\ \mathbf{I.k} \,\}^{T},\ \mathbf{J}^{B} = \{\, \mathbf{J.i},\ \mathbf{J.j},\ \mathbf{J.k} \,\}^{T},\ \mathbf{K}^{B} = \{\, \mathbf{K.i},\ \mathbf{K.j},\ \mathbf{K.k} \,\}^{T}$$

Ya que por definición, el producto punto entre dos vectores unitarios se puede representar como el coseno del ángulo entre los dos vectores, **a.b** = |**a**||**b**| cos(**a**,**b**) = cos(**a**,**b**), se pueden organizar convenientemente el conjunto de vectores unitarios globales para obtener la matriz de rotación DCM^B o Matriz de Direcciones Coseno, que como su nombre lo indica, consiste en todas las posibles combinaciones de los cosenos de los ángulos entre el cuerpo y el marco de referencia global.

Figura 3. **Matriz de Direcciones Coseno (DCM^B).**

$$[\mathbf{I}^B, \mathbf{J}^B, \mathbf{K}^B] = \begin{bmatrix} \mathbf{I.i} & \mathbf{J.i} & \mathbf{K.i} \\ \mathbf{I.j} & \mathbf{J.j} & \mathbf{K.j} \\ \mathbf{I.k} & \mathbf{J.k} & \mathbf{K.k} \end{bmatrix} = \begin{bmatrix} \cos(\mathbf{I},\mathbf{i}) & \cos(\mathbf{J},\mathbf{i}) & \cos(\mathbf{K},\mathbf{i}) \\ \cos(\mathbf{I},\mathbf{j}) & \cos(\mathbf{J},\mathbf{j}) & \cos(\mathbf{K},\mathbf{j}) \\ \cos(\mathbf{I},\mathbf{k}), & \cos(\mathbf{J},\mathbf{k}) & \cos(\mathbf{K},\mathbf{k}) \end{bmatrix} = \mathrm{DCM}^{\mathrm{B}}$$

Una propiedad muy útil de esta matriz de rotación es que puede componer múltiples rotaciones juntas, donde la matriz de rotación inicial es equivalente a aplicar un conjunto de rotaciones en sucesión. Esta secuencia de rotaciones debe ser realizada con especial cuidado ya que la multiplicación de matrices no es conmutativa, por lo que se puede inferir que las rotaciones tampoco lo son.

En lo concerniente a este trabajo de grado, cada una de estas rotaciones es representada por tres ángulos llamados Tait-Bryan, empleados en la aeronáutica para describir el alabeo (*roll*), el cabeceo (*pitch*) y la guiñada (*yaw o heading*) de una aeronave, correspondientes a las tres rotaciones consecutivas en el espacio euclidiano tridimensional alrededor de los ejes *x*, *y*, *z* del sistema ArduIMU+ V3 respectivamente .

Figura 4. **Rotaciones en el espacio euclidiano tridimensional.**

$$\text{Roll: } R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$\text{Pitch: } R_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$\text{Yaw: } R_z = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figura 5. **DCM^B compuesta de múltiples rotaciones.**

$$\mathrm{DCM}^{\mathrm{B}} = R_x R_y R_z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathrm{DCM}^{\mathrm{B}} = \begin{pmatrix} \cos\theta\cos\psi & -\cos\theta\sin\psi & \sin\theta \\ \cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi - \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\theta \\ \sin\phi\sin\psi - \cos\phi\sin\theta\cos\psi & \sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \cos\phi\cos\theta \end{pmatrix}$$

Los ángulos Tait-Bryan, empelados en este trabajo de grado para obtener la orientación del sistema ArduIMU+ V3, pueden ser calculados a partir de los elementos de la Matriz de Direcciones Coseno DCM^B de la Figura 5. El primer método para lograr derivar estos resultados de la ecuación de la Figura 5, emplea los cinco elementos de la fila 1 y la columna 3 como se muestra a continuación en la figura 6. El segundo, halla el ángulo de cabeceo de igual forma, pero emplea los más grandes 4 elementos de la matriz DCM para hallar el alabeo y la guiñada haciendo algunas sustituciones matemáticas.

Figura 6. Ángulos Tait-Bryan

$$\theta = \sin^{-1}\left(\text{DCM}[1,3]\right)$$

$$\varphi = \tan^{-1}\left(-\frac{\text{DCM}[2,3]}{\text{DCM}[3,3]}\right)$$

$$\psi = \tan^{-1}\left(-\frac{\text{DCM}[1,2]}{\text{DCM}[1,1]}\right)$$

Ya que el ángulo de cabeceo se encuentra entre [-90°, 90°], los ángulos de alabeo y guiñada deben abarcar [-180°, 180°] para poder representar todas las posibles combinaciones de los ángulos entre el cuerpo y el marco de referencia global. Desafortunadamente la función tangente inversa tiene una imagen que abarca únicamente [-90°, 90°], razón por la que se hizo necesario emplear la función *atan2(y,x).*

En una variedad de lenguajes de programación, esta función de una tangente inversa con dos argumentos recoge información de los signos de las dos entradas con el fin de devolver el cuadrante apropiado del ángulo computado. Por esta razón, en el desarrollo del código para la obtención de los ángulos Tait-Bryan, se decidió emplear esta función en vez de la función tangente inversa de argumento simple.

4.2 LA DCM EN EL SISTEMA ArduIMU+ V3

Al emplear la Matriz de Direcciones Coseno para determinar la orientación del sistema ArduIMU+ V3, lo primero que se debe hacer es fijar el sistema de coordenadas globales al marco de referencia de la tierra donde el vector unitario **I** tendrá la misma dirección del vector que apunta al Norte terrestre, el vector unitario **K** tendrá la misma dirección del vector que apunta al Cenit y así, con estos

dos vectores fijos y de acuerdo a la regla de la mano derecha, el vector unitario **J** debería apuntar al Oeste.

Se puede notar que los ejes de coordenadas del acelerómetro coinciden convenientemente con el sistema de referencia global y además, mediante las pruebas de calibración del dispositivo, se logró constatar que la salida arrojada por el acelerómetro cuando el dispositivo se encuentra en reposo es un vector que apunta hacia el cenit, razón por la cual se decide únicamente normalizar el vector de salida del acelerómetro para obtener el vector unitario **K**.

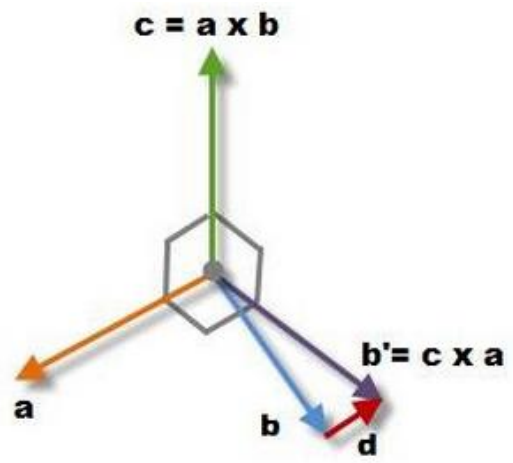
Es menester recordar que la disposición de los ejes del magnetómetro y el acelerómetro no son las mismas, por lo que se debe hacer uso de la variable *SENSOR_SIGN*[*eje*] presente en la ecuación (2) para invertir el sentido de los ejes cartesianos del magnetómetro que son necesarios para que los sistemas de coordenadas cartesianas de ambas unidades sean co-direccionales entre sí. Para este trabajo y en base a la Figura 1, se deben invertir sólo las direcciones de los ejes *x* y *y* z del magnetómetro.

Por otro lado, debido a que el módulo de medición de bajos campos magnéticos nos provee un vector con dirección del norte magnético local, debemos realizar una compensación adicional a este vector incluyendo la declinación magnética o ángulo comprendido entre el norte magnético y el norte geográfico. Por suerte en internet se pueden encontrar varios estimadores del valor de la declinación magnética para lograr obtener un vector unitario **I** compensado, que apunte al norte geográfico [6].

El vector unitario **J** se puede obtener del producto cruz entre los vectores unitarios **K** e **I**, pero debido a que el campo magnético no es paralelo a la superficie de la tierra, antes se debe lograr que el ángulo entre estos dos vectores sea 90°

haciendo uso de *la regla del triple producto*, donde **b'** = **c** x **a** = (**a** x **b**) x **a** = -**a** (**a**.**b**) + b(**a**.**a**) = **b** – **a** (**a.b**).

Figura 7. **Regla del triple producto**



Ya con los 3 vectores normalizados y ortogonales entre ellos, se puede proceder a reemplazar los valores de los vectores unitarios **I**, **J** y **K** por los elementos de la matriz DCM^B de la Figura 3, y posteriormente hallar los ángulos Tait-Bryan iniciales para el sistema ArduIMU+ V3 como es mostrado en la figura 6.

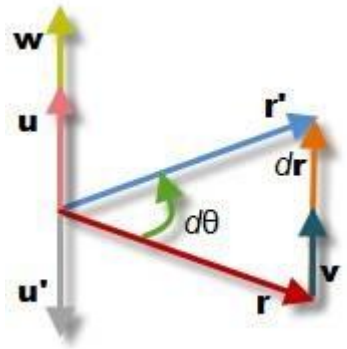
En caso de contar con el sistema ArduPilot Mega 2.5, el cual también hace parte de los dispositivos Arduino, se recomienda recurrir al trabajo de grado llevado a cabo por Jaimes y Sepulveda [7], en el cual se logra obtener los ángulos de orientación Tait-Bryan al implementar la matriz de direcciones coseno (DCM) empleando únicamente un conjunto de acelerómetro y magnetómetro.

5. EL FILTRO KALMAN

Este capítulo se lleva a cabo con la intención de introducir un algoritmo inspirado en el Filtrado Kalman empleado en la guía de introducción a la cinemática de orientación de starlino [8]. En esta guía se analiza la rotación como una función del tiempo, brindando las pautas para establecer la orientación del dispositivo en el tiempo t y nos permite expresar la matriz de rotación como la matriz DCM(t) basada en la velocidad angular.

Como se estudió en el capítulo 4, con la primera medición tomada del acelerómetro y el magnetómetro se calculó la matriz de direcciones coseno DCM_0 en el tiempo t_0. Ahora definimos la salida del giroscopio como $\mathbf{w} = \{ w_x , w_y , w_z \}$, de donde notamos a $\mathbf{w_g}$ como la velocidad angular medida por el giroscopio y se define al desplazamiento angular durante un periodo de tiempo dt como se muestra en la figura 8:

Figura 8. **Desplazamiento angular**



$$d\boldsymbol{\theta}_{\mathbf{g}} = dt\, \mathbf{w_g} \quad (5)$$

Ahora bien, haciendo una nueva medición sobre el acelerómetro y el magnetómetro, podemos calcular el desplazamiento angular experimentado por cada uno de estos dispositivos de la siguiente forma:

$$d\boldsymbol{\theta_a} = dt\ \mathbf{w_a} = \mathbf{K}^B{}_0 \times (\mathbf{K}^B{}_{1A} - \mathbf{K}^B{}_0) \quad (6)$$
$$d\boldsymbol{\theta_m} = dt\ \mathbf{w_m} = \mathbf{I}^B{}_0 \times (\mathbf{I}^B{}_{1M} - \mathbf{I}^B{}_0) \quad (7)$$

La idea de contar con 3 valores diferentes para el desplazamiento angular es promediar estos valores mediante las variables s_a, s_g y s_m y obtener un nuevo y mejor estimado para el desplazamiento $d\boldsymbol{\theta}$ que combine los resultados de los 3 dispositivos de medición existentes en el sistema mediante la ecuación (8):

$$d\boldsymbol{\theta} = (s_a\ d\boldsymbol{\theta_a} + s_g\ d\boldsymbol{\theta_g} + s_m\ d\boldsymbol{\theta_m}) / (s_a + s_g + s_m) \quad (8)$$

Teniendo el valor del desplazamiento angular podemos calcular los vectores **I**, **J**, **K** en el tiempo t_1, con los cuales se construye la matriz de direcciones coseno DCM_1 y de la cual se obtienen los nuevos ángulos Tait-Bryan para dicha matriz. Cabe resaltar que en la práctica el vector $\mathbf{J}^B{}_1$ se calcula nuevamente del producto cruz entre los vectores unitarios $\mathbf{I}^B{}_1$ y $\mathbf{K}^B{}_1$ después de ortonormalizados, enfatizando en que ésta lógica es aproximada y su validez depende de lo pequeño que sea el dt.

$$\mathbf{K}^B{}_1 \approx \mathbf{K}^B{}_0 + (\ d\boldsymbol{\theta} \times \mathbf{K}^B{}_0\ ) \quad (9)$$
$$\mathbf{I}^B{}_1 \approx \mathbf{I}^B{}_0 + (\ d\boldsymbol{\theta} \times \mathbf{I}^B{}_0) \quad (10)$$
$$\mathbf{J}^B{}_1 \approx \mathbf{J}^B{}_0 + (\ d\boldsymbol{\theta} \times \mathbf{J}^B{}_0) \quad (11)$$

Finalmente, podemos emplear reiteradamente este algoritmo para obtener la matriz de direcciones coseno DCM_2, DCM_3 y así sucesivamente, de las cuales podremos extraer los valores de los ángulos Tait-Bryan en cualquier tiempo t_n.

6. RESULTADOS

Considerando que la memoria del sistema ArduIMU+ V3 tiene un máximo de 30.720 bytes, éste algoritmo fue optimizado de tal forma que sólo se hicieron necesarios 16.150 bytes incluyendo el debug o depuración por puerto serial, es decir, alrededor del 52.57% de la memoria Flash total que posee el dispositivo para almacenar programas.

Figura 9. **Compilación del algoritmo ArduIMU+ V3 en el software Arduino**

Una vez el sistema ArduIMU+ V3 se encuentra estable y sin perturbaciones, se puede poner en marcha el código para analizar los resultados arrojados inicialmente:

Figura 10. Resultados de los ángulos Tait-Bryan obtenidos mediante el algoritmo propuesto.

```
Inicio del programa
Inicializacion del MPU6000:
    3-Axis Magnetometer: ok!
    3-Axis Gyroscope: ok!
Inicializacion del HMC5883L:
    3-Axis Magnetometer: ok!

|0.7862371444   0.6177111625    -0.0162551736|
|-0.6177437782  0.7850973606    -0.0448871517|
|-0.0149655365  0.0453334760    0.9988597869|
Formula 1:      [2.57           -0.93           -46.37  ]
Formula 2:      [2.57           -0.93           -46.37  ]

|0.7863158226   0.6176071166    -0.0163994646|
|-0.6176497936  0.7851842880    -0.0446594762|
|-0.0147055315  0.0452455806    0.9988676071|
Formula 1:      [2.56           -0.94           -46.36  ]
Formula 2:      [2.56           -0.94           -46.36  ]

|0.7865803241   0.6172704219    -0.0163902902|
|-0.6173155784  0.7854565620    -0.0444908428|
|-0.0145890736  0.0451135969    0.9988754272|
Formula 1:      [2.55           -0.94           -46.33  ]
Formula 2:      [2.55           -0.94           -46.33  ]
```

Fuente: Autores.

Se puede apreciar en la figura 10 que lo primero que aparece al correr el software propuesto es un encabezado producto de la inicialización y calibración de los sensores en posición horizontal y sin ningún tipo de alteración. Esta calibración permite garantizar que cada sensor funciona y es compensado adecuadamente, de lo contrario el programa arrojaría un error visible en el encabezado y no funcionaría.

Seguidamente del encabezado, se pueden observar los ángulos Tait-Bryan arrojados por la DCM inicial o DCM_0, la cual es resultado únicamente de las mediciones realizadas por el magnetómetro y el acelerómetro. Considerando las

condiciones iniciales, estos resultados son coherentes con los ángulos de alabeo, cabeceo y guiñada reales medidos desde el sistema ArduIMU+ V3 hasta el marco de referencia global, ya que son aproximadamente cero para el alabeo y cabeceo, y diferente de cero para la guiñada.

Los dos siguientes resultados de los ángulos Tait-Bryan en la figura 10 son los resultados de las matrices DCM_1 y DCM_2, las cuales responden a la matriz DCM_n que incorpora el filtrado Kalman para realizar correcciones sobre los ángulos medidos inicialmente, correcciones que corresponden mayormente a las mediciones del giroscopio, y en menor medida a las mediciones del acelerómetro y el magnetómetro.

Figura 11. **Verificación del ángulo de alabeo arrojado por el sistema ArduIMU+ V3**

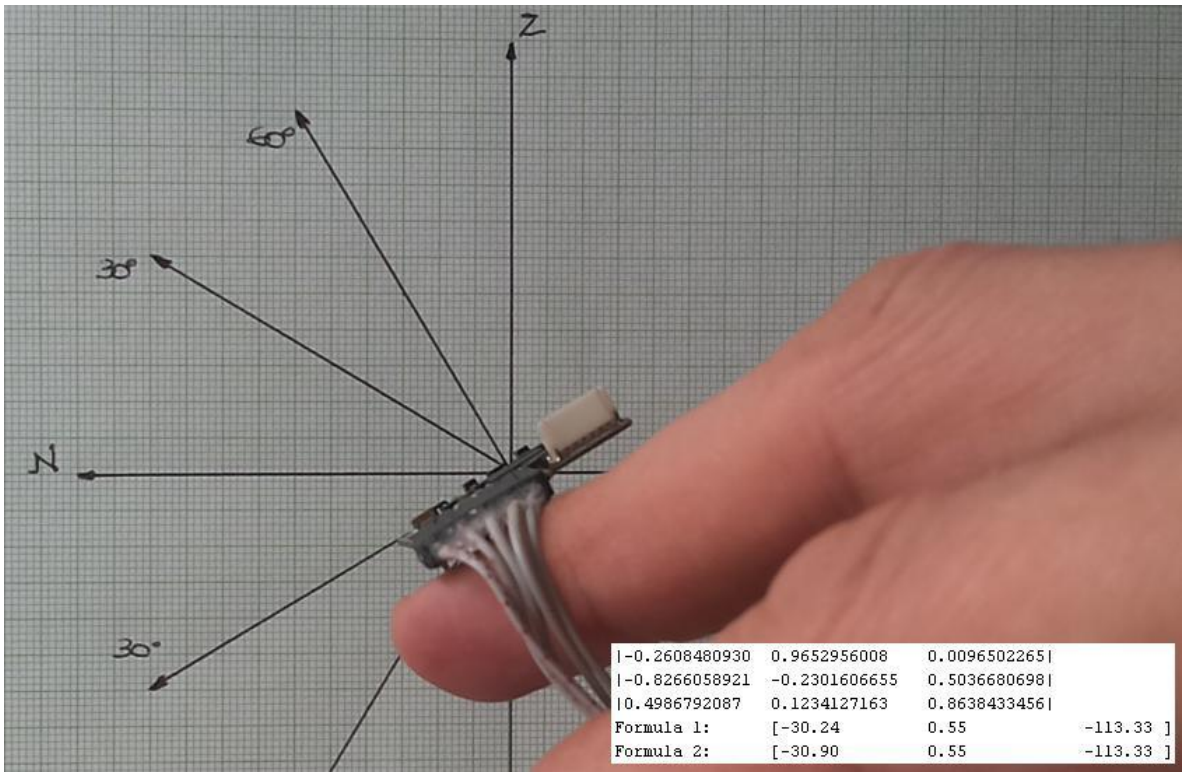


Figura 12. **Verificación del ángulo de cabeceo arrojado por el sistema ArduIMU+ V3.**

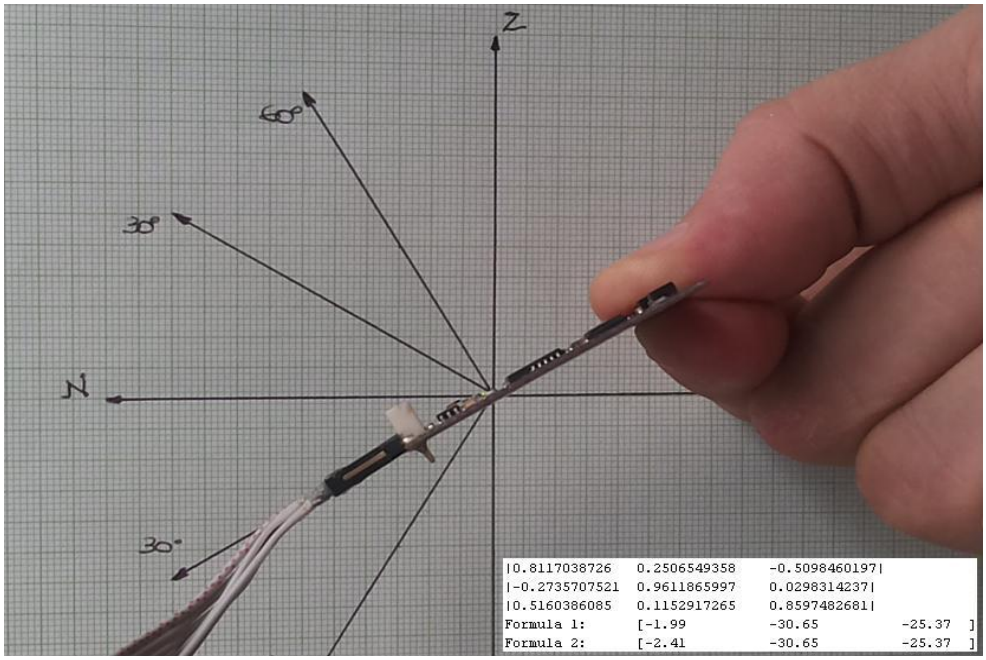
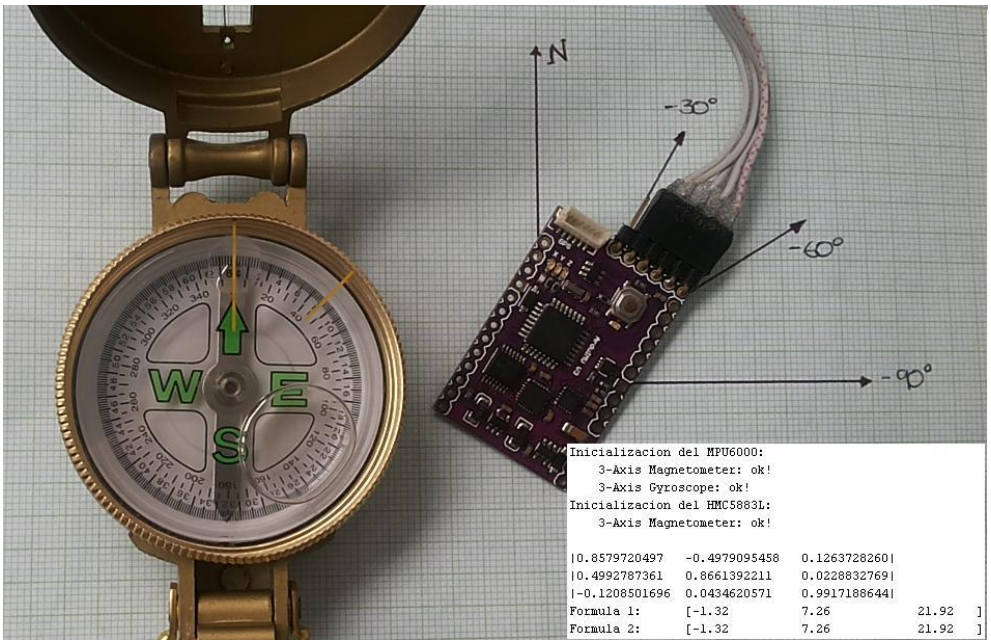


Figura 13. Verificación del ángulo de guiñada arrojado por el sistema ArduIMU+ V3.



Las figuras 11, 12 y 13 muestran respectivamente los ángulos de alabeo, cabeceo y guiñada arrojados mediante la verificación individual de cada ángulo empleando los datos de la matriz DCM_n mediante dos métodos diferentes. Es apreciable que los ángulos son similares inicialmente con cualquiera de las dos fórmulas, resultado de gran interés pues implica que cualquiera de los dos métodos empleados son viables.

En general, las pruebas realizadas para calcular los ángulos Tait-Bryan responden bien a pesar del problema que poseen cuando el sistema tiende a llegar a su posición vertical, es decir, cuando el cabeceo supera los $\pm$70° grados como se alcanza a apreciar en la figura 14. En este caso, los ángulos de alabeo y guiñada comienzan a aumentar considerablemente su valor para perder toda coherencia. Es necesario resaltar que cuando el sistema vuelve a bajar de los $\pm$70° grados, el error desaparece.

Figura 14. Errores medidos en los ángulos de alabeo y guiñada cuando el cabeceo supera su ángulo crítico. El alabeo y guiñada deberían ser cero.

```
|0.0688705682   0.0967577838    0.9915061950|
|0.2629369258   0.9583339691    -0.1163868188|
|-0.9623518943  0.2687635898    0.0580491495|
Formula 1:      [63.49          82.53           -62.77  ]
Formula 2:      [63.03          82.58           -62.77  ]
```

Este último error se debe a que el ángulo entre el vector campo magnético y el vector gravedad no son completamente ortogonales, ya que antes de comenzar el vuelo tomamos al vector gravedad como un vector fijo pues este no experimenta otra fuerza diferente a la fuerza de gravedad. Con esta suposición podemos calcular la matriz de rotación inicial obligando al vector campo magnético a ubicarse perpendicular a la superficie de la tierra y obteniendo una primera matriz fiablemente.

Ahora bien, cuando el vuelo ya ha comenzado, este vector aceleración ya no está apuntando fiablemente hacia el cenit, sino que se ve alterado por las aceleraciones y desaceleraciones del aeroplano, así como cuando el sistema intenta ascender o descender, lo cual altera la dirección de este vector y por lo tanto de la matriz de rotación durante ascensos y descensos muy pronunciados.

7. CONCLUSIONES

Se logró caracterizar adecuadamente cada una de las unidades de medición del sistema e implementar un algoritmo que interprete adecuadamente sus datos en base al código abierto para el ArduIMU+ V3 dispuesto en la plataforma de creación de prototipos electrónicos, código que fue alterado con el propósito de obtener un mejor desempeño en la toma de datos del magnetómetro principalmente y evitar errores debido a la duplicación de sus datos.

Se identificaron los requerimientos básicos para el desarrollo e implementación de un segundo algoritmo para el procesamiento de las mediciones tomadas y su conveniente organización para obtener la matriz de direcciones coseno DCM^B, mediante la cual es posible obtener los ángulos Tait-Bryan entre el marco de referencia global y el sistema ArduIMU+ V3 en cualquier tiempo t_n.

Hasta este punto se lograron emular los resultados alcanzados por Jaimes y Sepulveda [7] en su trabajo de grado al implementar la matriz DCM sobre el sistema Ardupilot Mega 2.5. Ahora bien, las mejoras emuladas con este nuevo código comprenden la compensación de cada eje del acelerómetro, el empleo de las función autotest en el magnetómetro junto con el cambio del modo continuo de medición de datos al modo simple, con lo que se logró evitar la duplicación de datos, aumentar la tasa de transferencia de 9600 baudios a 115200 baudios y consecuentemente reducir el error y obtener una mejor aproximación al lograr un dt mucho menor.

Como última medida se logró establecer la orientación del sistema ArduIMU+ V3 mediante la matriz DCM(t), llevando a cabo un filtrado Kalman simple mediante el cual se logra hacer una predicción más acertada de los vectores unitarios $\mathbf{I_B}$, $\mathbf{J_B}$ y

$\mathbf{K_B}$ al incorporar las mediciones del giroscopio a las del acelerómetro y las del magnetómetro. Hay que resaltar que esta medida se realiza ya que los datos arrojados por el acelerómetro no son 100% fiables una vez que la aeronave ha comenzado a desplazarse.

Con todo esto se puede inferir que el sistema desarrollado es viable y puede ser tomado como referencia para posteriores trabajos de grado que necesiten emplear un modelo opcional para sistemas que no usen o simplemente no puedan hacer uso del GPS.

8. OBSERVACIONES

Es preciso recordar que este algoritmo parte de la premisa de que para hallar la primera matriz o DCM_0, es necesario garantizar que el sistema ArduIMU+ V3 se encuentre horizontal a la superficie de la tierra y que además no esté sometido a aceleraciones diferentes a la aceleración causada por la fuerza de gravedad.

En este trabajo se empleó un giroscopio para suavizar el error en el acelerómetro, ya que éste es menos sensible a los movimientos mecánicos lineales que afectan al acelerómetro. Sin embargo, los giroscopios tienen otro tipo de problemas como por ejemplo la deriva, lo que causa que su valor no vuelva a cero cuando se detiene el movimiento.

Este tipo de errores en cada dispositivo de medición son la razón por la cual se emplean *pesos* para cada valor del desplazamiento angular estimado por cada dispositivo, los cuales se recomienda calcular y ajustar experimentalmente para lograr minimizar el ruido y obtener la velocidad de respuesta deseada para el sistema en general.

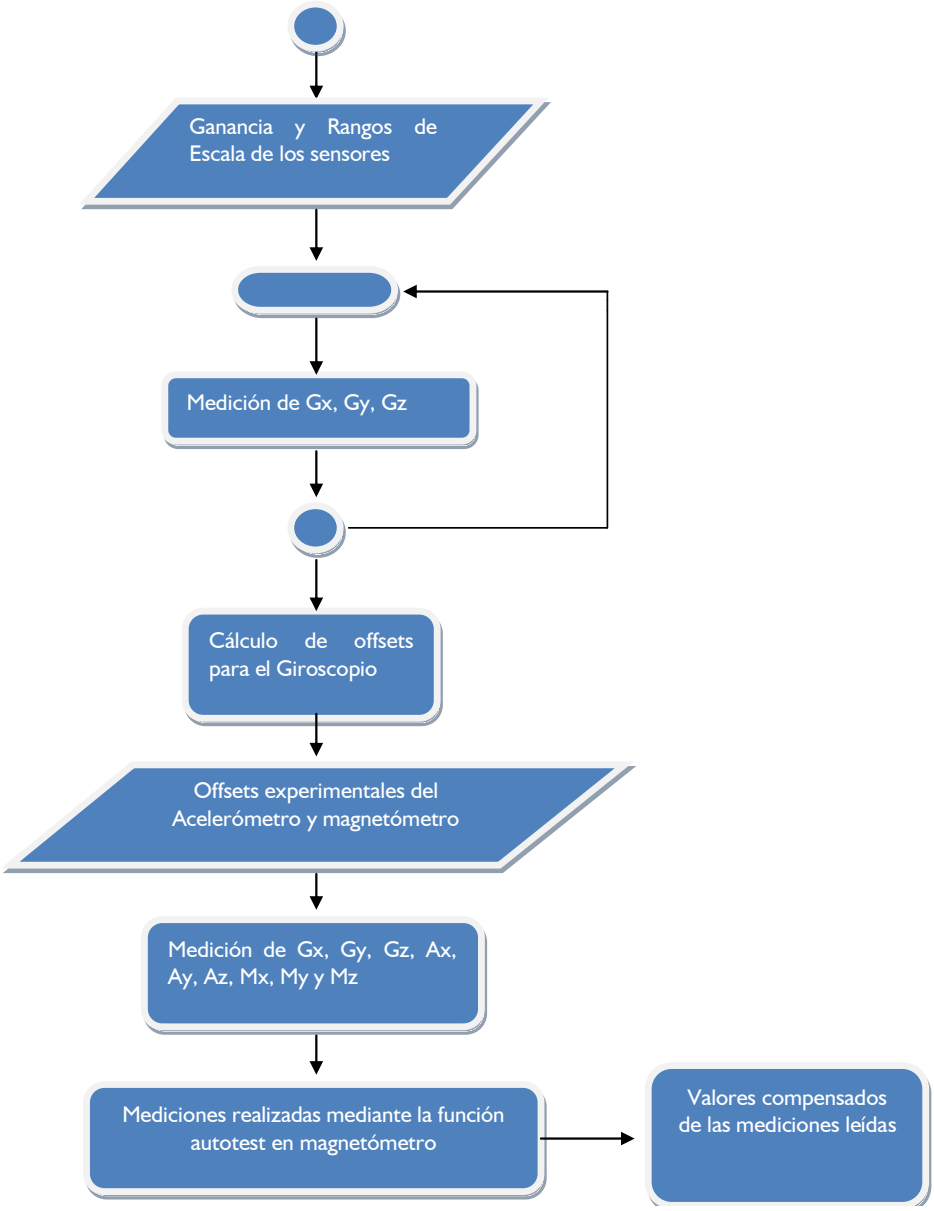
Muy a pesar del problema que tenemos con ángulos de cabeceo mayores a $\pm 70°$, lo que no nos permite trabajar con aeronaves que deban hacer acrobacias que les exijan volar en posición vertical, con este trabajo de grado nos acercamos un poco más a la idea de llevar a cabo una estrategia de control digital sobre el sistema ArduIMU+ V3 que le permita a una aeronave seguir rutas predefinidas, controlar su orientación y además su dirección.

Para un futuro trabajo de grado, se propone elaborar el código para una rutina de calibración de todos los ejes del acelerómetro y buscar la forma de optimizar el

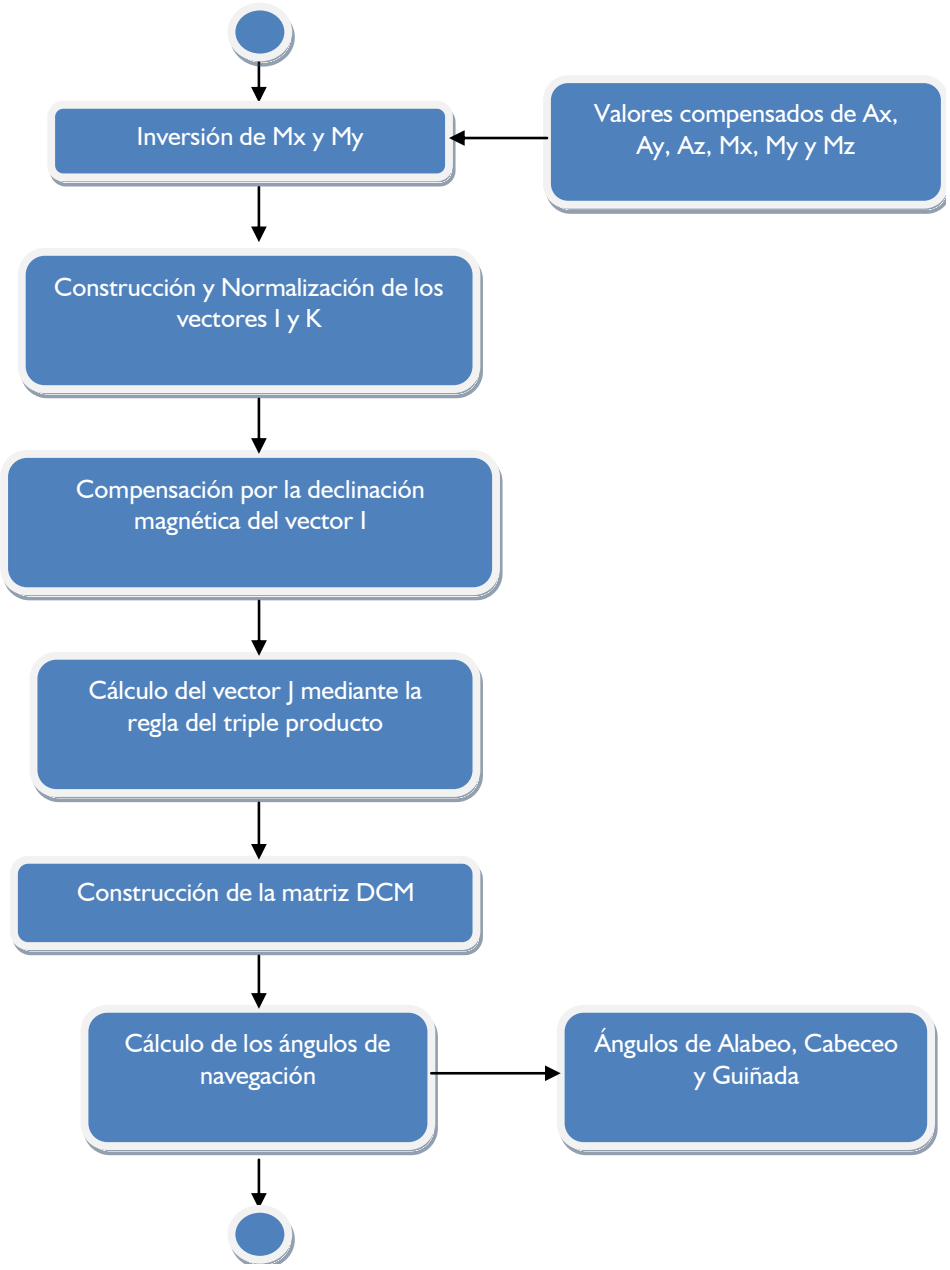
código empleado para ortogonalizar los vectores unitarios $\mathbf{I}^{\mathbf{B}}$ y $\mathbf{K}^{\mathbf{B}}$ una vez el sistema ha dejado el reposo y ha comenzado a moverse. Para ésto se podría intentar cambiar la rutina establecida dejando fijo el vector campo magnético una vez se ha dispuesto horizontal a la superficie de la tierra y corregir el valor para el vector aceleración con respecto al cenit.

9. DIAGRAMAS DE FLUJO

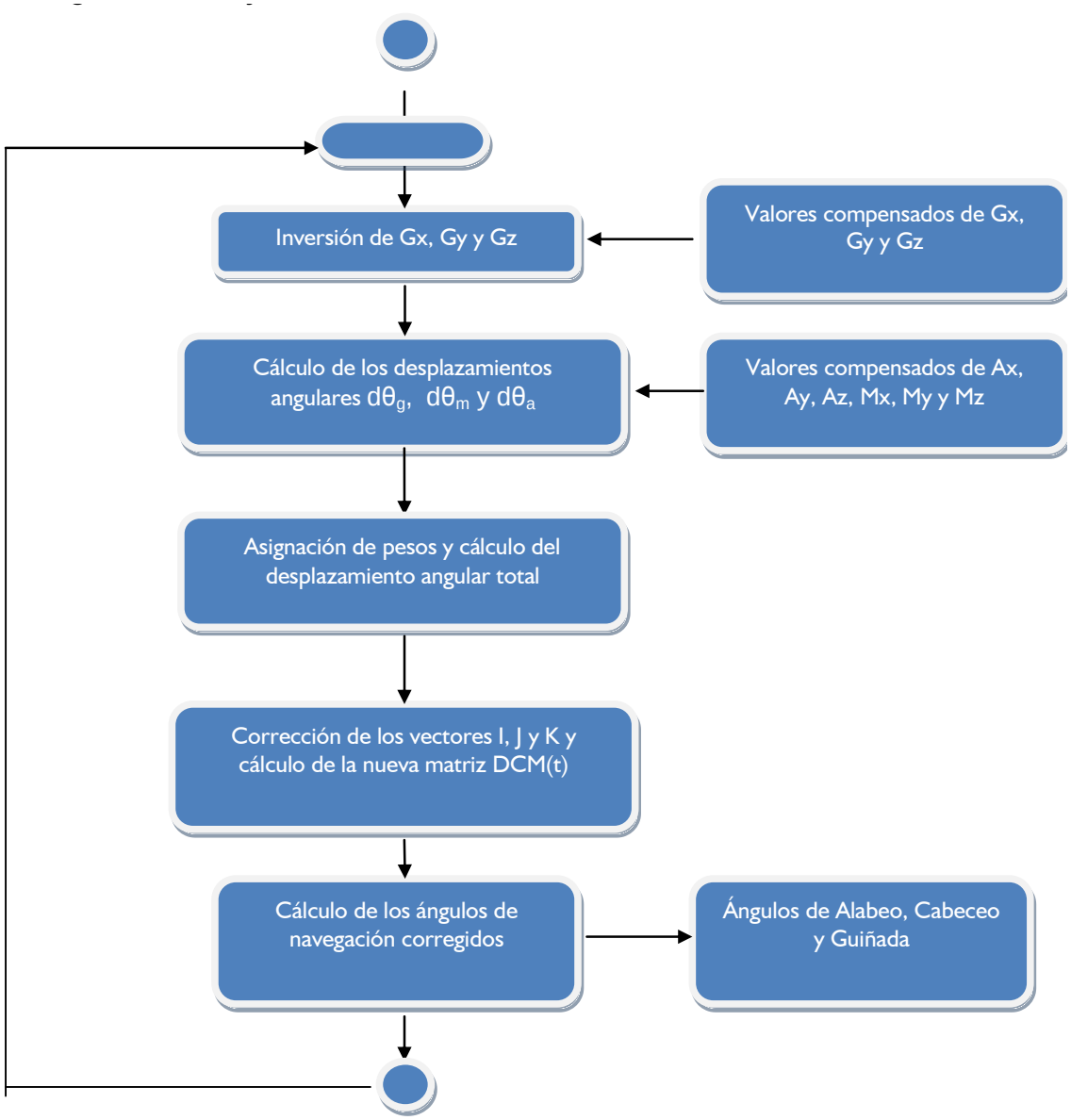
9.1 DIAGRAMA DE FLUJO - ALGORITMO PARA LA ADQUISICIÓN DE LAS SEÑALES



9.2 DIAGRAMA DE FLUJO – CONSTRUCCIÓN DE LA MATRIZ DCM



9.3 DIAGRAMA DE FLUJO – FILTRADO KALMAN



REFERENCIAS BIBLIOGRAFICAS

[1] Direction Cosine Matrix IMU: Theory, Borrador, William Premerlani and Paul Bizard, 5/17/2009. Disponible en línea en: http://dc261.4shared.com/doc/Af232ta1/preview.html. Consultado el 24 de Enero de 2.014.

[2] Plataforma de Creación de Prototipos Electrónicos Open Source. 8/13/2013. Disponible en línea en: https://code.google.com/p/ardu-imu/. Consultado el 26 de Enero de 2.014.

[3] Sitio oficial de la comunidad DIY DRONES. 1/26/2014. Disponible en línea en: http://diydrones.com/. Consultado el 26 de Enero de 2.014.

[4] MPU-6000 and MPU-6050, Product Specification, Revision 3.4, InvenSense, 8/19/2013. Disponible en línea en: http://invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf. Consultado el 24 de Enero de 2.014.

[5] 3-Axis Digital Compass IC, HMC5883L, Honeywell, 2/21/2013. Disponible en línea en: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf. Consultado el 24 de Febrero de 2014.

[6] Magnetic Field Calculators, National Geophysical Data Center, NOAA. Disponible en línea en: http://www.ngdc.noaa.gov/geomag-web/. Consultado el 29 de Febrero de 2014.

[7] Evaluación de la solución ArduPilot Mega 2.5 para la implementación de algoritmos de control en aeroplanos a escala no tripulados (UAV). Leonardo Jaimes y Cesar Sepúlveda. 2/21/2013. UIS, sede Bucaramanga.

[8] DCM Tutorial - An Introduction to Orientation Kinematics. 5/17/2011. Disponible en línea en http://www.starlino.com/dcm_tutorial.html. Consultado el 30 de Febrero de 2.014.

BIBLIOGRAFÍA

3-Axis Digital Compass IC, HMC5883L, Honeywell, 2/21/2013. Disponible en línea en: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf. Consultado el 24 de Febrero de 2014.

DCM Tutorial - An Introduction to Orientation Kinematics. 5/17/2011. Disponible en línea en http://www.starlino.com/dcm_tutorial.html. Consultado el 30 de Febrero de 2.014.

Direction Cosine Matrix IMU: Theory, Borrador, William Premerlani and Paul Bizard, 5/17/2009. Disponible en línea en: http://dc261.4shared.com/doc/Af232ta1/preview.html. Consultado el 24 de Enero de 2.014.

Evaluación de la solución ArduPilot Mega 2.5 para la implementación de algoritmos de control en aeroplanos a escala no tripulados (UAV). Leonardo Jaimes y Cesar Sepúlveda. 2/21/2013. UIS, sede Bucaramanga.

Magnetic Field Calculators, National Geophysical Data Center, NOAA. Disponible en línea en: http://www.ngdc.noaa.gov/geomag-web/. Consultado el 29 de Febrero de 2014.

MPU-6000 and MPU-6050, Product Specification, Revision 3.4, InvenSense, 8/19/2013. Disponible en línea en: http://invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf. Consultado el 24 de Enero de 2.014.

Plataforma de Creación de Prototipos Electrónicos Open Source. 8/13/2013. Disponible en línea en: https://code.google.com/p/ardu-imu/. Consultado el 26 de Enero de 2.014.

Sitio oficial de la comunidad DIY DRONES. 1/26/2014. Disponible en línea en: http://diydrones.com/. Consultado el 26 de Enero de 2.014.

ANEXOS

ANEXO A. MPU-6000.h

```
// Librería tomada de la Plataforma de Creación de Prototipos Electrónicos [2]
// MPU6000 support for ArduIMU V3
#define MPU6000_CHIP_SELECT_PIN        4        // MPU6000 CHIP SELECT

// MPU 6000 registers
#define MPUREG_WHOAMI                           0x75
#define MPUREG_SMPLRT_DIV                       0x19
#define MPUREG_CONFIG                           0x1A
#define MPUREG_GYRO_CONFIG                      0x1B
#define MPUREG_ACCEL_CONFIG                     0x1C
#define MPUREG_INT_PIN_CFG                      0x37
#define MPUREG_INT_ENABLE                       0x38
#define MPUREG_ACCEL_XOUT_H                     0x3B
#define MPUREG_ACCEL_XOUT_L                     0x3C
#define MPUREG_ACCEL_YOUT_H                     0x3D
#define MPUREG_ACCEL_YOUT_L                     0x3E
#define MPUREG_ACCEL_ZOUT_H                     0x3F
#define MPUREG_ACCEL_ZOUT_L                     0x40
#define MPUREG_TEMP_OUT_H                       0x41
#define MPUREG_TEMP_OUT_L                       0x42
#define MPUREG_GYRO_XOUT_H                      0x43
#define MPUREG_GYRO_XOUT_L                      0x44
#define MPUREG_GYRO_YOUT_H                      0x45
#define MPUREG_GYRO_YOUT_L                      0x46
```

```
#define MPUREG_GYRO_ZOUT_H                    0x47
#define MPUREG_GYRO_ZOUT_L                    0x48
#define MPUREG_USER_CTRL                      0x6A
#define MPUREG_PWR_MGMT_1                     0x6B
#define MPUREG_PWR_MGMT_2                     0x6C

// Configuration bits  MPU6000
#define BIT_SLEEP                             0x40
#define BIT_H_RESET                           0x80
#define BITS_CLKSEL                           0x07
#define MPU_CLK_SEL_PLLGYROX                  0x01
#define MPU_CLK_SEL_PLLGYROZ                  0x03
#define MPU_EXT_SYNC_GYROX                    0x02
#define BITS_FS_250DPS                        0x00
#define BITS_FS_500DPS                        0x08
#define BITS_FS_1000DPS                       0x10
#define BITS_FS_2000DPS                       0x18
#define BITS_FS_MASK                          0x18
#define BITS_DLPF_CFG_256HZ_NOLPF2            0x00
#define BITS_DLPF_CFG_188HZ                   0x01
#define BITS_DLPF_CFG_98HZ                    0x02
#define BITS_DLPF_CFG_42HZ                    0x03
#define BITS_DLPF_CFG_20HZ                    0x04
#define BITS_DLPF_CFG_10HZ                    0x05
#define BITS_DLPF_CFG_5HZ                     0x06
#define BITS_DLPF_CFG_2100HZ_NOLPF            0x07
#define BITS_DLPF_CFG_MASK                    0x07
#define BIT_INT_ANYRD_2CLEAR                  0x10
#define BIT_RAW_RDY_EN                        0x01
#define BIT_I2C_IF_DIS                        0x10
```

```
// global variables
volatile uint8_t MPU6000_newdata;
// Sensor variables
int accelX;
int accelY;
int accelZ;
int gyroX;
int gyroY;
int gyroZ;

#define BITS_FS_2G          0x00   // range of ± 2g(16384LSB/g)
#define BITS_FS_4G          0x08   // range of ±  4g (8192LSB/g)
#define BITS_FS_8G          0x10   // range of ±  8g (4096LSB/g)
#define BITS_FS_16G         0x18   // range of ±16g (2048LSB/g)
```

ANEXO B. HMC5883L. h

```
// Librería tomada de la Plataforma de Creación de Prototipos Electrónicos [2]
// HMC5883L support for ArduIMU V3
#define COMPASS_ADDRESS         0x1E
#define ConfigRegA              0x00
#define ConfigRegB              0x01
#define ModeRegister            0x02
#define DataOutputXMSB          0x03
#define DataOutputXLSB          0x04
#define DataOutputZMSB          0x05
#define DataOutputZLSB          0x06
#define DataOutputYMSB          0x07
#define DataOutputYLSB          0x08
#define StatusRegister          0x09
#define IDRegisterA             0x0A
#define IDRegisterB             0x0B
#define IDRegisterC             0x0C

// ConfigRegB valid gain configuration bits
#define DeviceGain_0_1370       0x00
#define DeviceGain_1_1090       0x20
#define DeviceGain_2_820        0x40
#define DeviceGain_3_660        0x60
#define DeviceGain_4_440        0x80
#define DeviceGain_5_390        0xA0
#define DeviceGain_6_330        0xC0
#define DeviceGain_7_230        0xE0
```

```
// ModeRegister valid modes
#define ContinuousConversion    0x00
#define SingleConversion        0x01

// ConfigRegA valid sample averaging
#define SampleAveraging_1       0x00
#define SampleAveraging_2       0x01
#define SampleAveraging_4       0x02
#define SampleAveraging_8       0x03

// ConfigRegA valid data output rates
#define DataOutputRate_0_75HZ   0x00
#define DataOutputRate_1_5HZ    0x01
#define DataOutputRate_3HZ      0x02
#define DataOutputRate_7_5HZ    0x03
#define DataOutputRate_15HZ     0x04
#define DataOutputRate_30HZ     0x05
#define DataOutputRate_75HZ     0x06

// ConfigRegA valid measurement configuration bits
#define NormalOperation         0x00
#define PositiveBiasConfig      0x01
#define NegativeBiasConfig      0x02

// Config 3-Axis Magnetometer Limits
#define Min_Lim          679.15
#define Max_Lim         1607.05
```

ANEXO C. MPU6000.cpp

```
// Código tomado de la Plataforma de Creación de Prototipos Electrónicos [2]
// MPU6000 support for ArduIMU
#include "MPU6000.h"
#include <SPI.h>
// MPU6000 SPI functions
byte MPU6000_SPI_read(byte reg)
{
  byte dump;
  byte return_value;
  byte addr = reg | 0x80;                                    // Set most significant
bit
  digitalWrite(MPU6000_CHIP_SELECT_PIN, LOW);
  dump = SPI.transfer(addr);
  return_value = SPI.transfer(0);
  digitalWrite(MPU6000_CHIP_SELECT_PIN, HIGH);
  return(return_value);
}

void MPU6000_SPI_write(byte reg, byte data)
{
  byte dump;
  digitalWrite(MPU6000_CHIP_SELECT_PIN, LOW);
  dump = SPI.transfer(reg);
  dump = SPI.transfer(data);
  digitalWrite(MPU6000_CHIP_SELECT_PIN, HIGH);
}
```

```
// MPU6000 INTERRUPT ON INT0
void MPU6000_data_int()
{
  MPU6000_newdata++;
}

// MPU6000 Initialization and configuration
void MPU6000_Init(void)
{
  // MPU6000 chip select setup
  pinMode(MPU6000_CHIP_SELECT_PIN, OUTPUT);
  digitalWrite(MPU6000_CHIP_SELECT_PIN, HIGH);

  // SPI initialization
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV16);          // SPI at 1Mhz (on 16Mhz
clock)
  delay(10);

  // Chip reset
  MPU6000_SPI_write(MPUREG_PWR_MGMT_1, BIT_H_RESET);
  delay(100);

  // Wake up device and select GyroZ clock (better performance)
  MPU6000_SPI_write(MPUREG_PWR_MGMT_1, MPU_CLK_SEL_PLLGYROZ);
  delay(1);

  // Disable I2C bus (recommended on datasheet)
  MPU6000_SPI_write(MPUREG_USER_CTRL, BIT_I2C_IF_DIS);
  delay(1);
```

```
// Sample rate = 200Hz Fsample= 1Khz/(4+1) = 200Hz
// MPU6000_SPI_write(MPUREG_SMPLRT_DIV,0x04);
// Sample rate = 50Hz    Fsample= 1Khz/(19+1) = 50Hz
MPU6000_SPI_write(MPUREG_SMPLRT_DIV,19);
delay(1);
// FS & DLPF   FS=2000º/s, DLPF = 20Hz (low pass filter)
MPU6000_SPI_write(MPUREG_CONFIG, BITS_DLPF_CFG_20HZ);
delay(1);

// Gyro scale 250º/s
MPU6000_SPI_write(MPUREG_GYRO_CONFIG,BITS_FS_250DPS);
delay(1);

// Accel range 8g (4096LSB/g)
MPU6000_SPI_write(MPUREG_ACCEL_CONFIG,BITS_FS_8G);
delay(1);

// INT CFG => Interrupt on Data Ready
MPU6000_SPI_write(MPUREG_INT_ENABLE,BIT_RAW_RDY_EN);
delay(1);
MPU6000_SPI_write(MPUREG_INT_PIN_CFG,BIT_INT_ANYRD_2CLEAR);
delay(1);
// Oscillator set
//
MPU6000_SPI_write(MPUREG_PWR_MGMT_1,MPU_CLK_SEL_PLLGYROZ);
delay(1);

// MPU_INT is connected to INT 0. Enable interrupt on INT0
attachInterrupt(0,MPU6000_data_int,RISING);
```

```
  MPU6000_set_offset();
  Serial.print("    3-Axis Gyroscope: ok!\n");
}

// Read gyros and accel sensors on MPU6000
void MPU6000_read()
{
int byte_H;
int byte_L;

// Read AccelX
byte_H = MPU6000_SPI_read(MPUREG_ACCEL_XOUT_H);
byte_L = MPU6000_SPI_read(MPUREG_ACCEL_XOUT_L);
accelX = (byte_H<<8)| byte_L;
// Read AccelY
byte_H = MPU6000_SPI_read(MPUREG_ACCEL_YOUT_H);
byte_L = MPU6000_SPI_read(MPUREG_ACCEL_YOUT_L);
accelY = (byte_H<<8)| byte_L;

// Read AccelZ
byte_H = MPU6000_SPI_read(MPUREG_ACCEL_ZOUT_H);
byte_L = MPU6000_SPI_read(MPUREG_ACCEL_ZOUT_L);
accelZ = (byte_H<<8)| byte_L;

// Read Temp
byte_H = MPU6000_SPI_read(MPUREG_TEMP_OUT_H);
byte_L = MPU6000_SPI_read(MPUREG_TEMP_OUT_L);
temp = (byte_H<<8)| byte_L;
```

```
// Read GyroX
byte_H = MPU6000_SPI_read(MPUREG_GYRO_XOUT_H);
byte_L = MPU6000_SPI_read(MPUREG_GYRO_XOUT_L);
gyroX = (byte_H<<8)| byte_L;
// Read GyroY
byte_H = MPU6000_SPI_read(MPUREG_GYRO_YOUT_H);
byte_L = MPU6000_SPI_read(MPUREG_GYRO_YOUT_L);
gyroY = (byte_H<<8)| byte_L;
// Read GyroZ
byte_H = MPU6000_SPI_read(MPUREG_GYRO_ZOUT_H);
byte_L = MPU6000_SPI_read(MPUREG_GYRO_ZOUT_L);
gyroZ = (byte_H<<8)| byte_L;
}
```

ANEXO D. MPU6000.cpp

```
// Código realizado para este Trabajo de Investigación por el autor
// Compensación de las variables arrojadas por el MPU6000.
void MPU6000_Read()
{
  MPU6000_read();

  gyro[0]=SENSOR_SIGN[0]*(gyroX-mag_offset[0]);
  gyro[1]=SENSOR_SIGN[1]*(gyroY-mag_offset[1]);
  gyro[2]=SENSOR_SIGN[2]*(gyroZ-mag_offset[2]);

  tempt=temp/((double)340.0)+36.53;

  accel[0]=((float)accelX-mag_offset[3])*4096/4090;
  accel[1]=((float)accelY-mag_offset[4])*4096/4072;
  accel[2]=((float)accelZ-mag_offset[5])*4096/4171;
}

void MPU6000_set_offset()
{
  int i=0;

  delay(73);              // Time to get real values
  while(i<200)             // 200 samples each 10 ms to get an average of the value at
initial conditions.
  {
    MPU6000_read();
```

```
    mag_offset[0]+=gyroX;
    mag_offset[1]+=gyroY;
    mag_offset[2]+=gyroZ;

    delay(10);                                          // Time between samples
    i++;
  }

  mag_offset[0]=mag_offset[0]/(i+1);
  mag_offset[1]=mag_offset[1]/(i+1);
  mag_offset[2]=mag_offset[2]/(i+1);

  mag_offset[3]=118;                                    // Valor tomado
experimentalmente
  mag_offset[4]=-15;                                    // Valor tomado
experimentalmente
  mag_offset[5]=-538;                                   // Valor tomado
experimentalmente

  Serial.println("    3-Axis Magnetometer: ok!");
}
```

ANEXO E. HMC5883L.cpp

```
// Código realizado para este Trabajo de Investigación por el autor
// HMC5883L support for ArduIMU V3

#include "HMC5883L.h"
#include <Wire.h>

bool HMC5883_init()
{
  Wire.begin();

  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.write((uint8_t)ConfigRegB);
  Wire.write(DeviceGain_1_1090);                                    // 1.3 Gauss
  Wire.endTransmission();

  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.write(DataOutputXMSB);                                       // Sends
address to read from
  Wire.endTransmission();

  HMC5883_set_offset();
  HMC5883_Read();
  HMC5883_Read();                          // From here we take the data for
the axis test

  if(bias_x>Min_Lim && bias_x<Max_Lim)
```

```
  {
    if (bias_y>Min_Lim && bias_y<Max_Lim)
    {
      if (bias_z>Min_Lim && bias_z<Max_Lim)
      {
        Serial.print("    3-Axis Magnetometer: ok!\n");
      }
    }
  }
  else
  {
    Serial.print("    3-Axis Magnetometer: Error");
  }

  return(1);
}

void HMC5883_set_offset()                                    // set mag offsets
{
  mag_offset[6] = -38.5;
  mag_offset[7] = -120.25;
  mag_offset[8] = -71.5;
}

void HMC5883_read()                                          //  Read  Sensor
data in chip axis
{
  int i = 0;
  byte buff[6];
```

```
  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.write((uint8_t)ModeRegister);
  Wire.write((uint8_t)SingleConversion);                    // Set single mode
(default to 10Hz)
  Wire.endTransmission();                                   // End
transmission

  Wire.requestFrom(COMPASS_ADDRESS, 6);                     // request 6 bytes from
device
  while(Wire.available())
  {
    buff[i] = Wire.read();                                  // receive one byte
    i++;
  }
  delay(6);

  if (i==6)
  {
    mag_x = ((((int)buff[0]) << 8) | buff[1]);              // X axis
    mag_y = ((((int)buff[4]) << 8) | buff[5]);              // Y axis
    mag_z = ((((int)buff[2]) << 8) | buff[3]);              // Z axis
  }
}

void HMC5883__read_bias()
{
  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.write((uint8_t)ConfigRegA);
  Wire.write(SampleAveraging_8<<5 | DataOutputRate_75HZ<<2 |
NormalOperation);
```

```
  Wire.endTransmission();

  HMC5883_read();

  bias_x=mag_x;
  bias_y=mag_y;
  bias_z=mag_z;
}

void HMC5883__read_normal()
{
  Wire.beginTransmission(COMPASS_ADDRESS);
  Wire.write((uint8_t)ConfigRegA);
  Wire.write(SampleAveraging_8<<5 | DataOutputRate_75HZ<<2 |
PositiveBiasConfig);
  Wire.endTransmission();

  HMC5883_read();
}

void HMC5883_Read()
{
  HMC5883__read_bias();
  HMC5883__read_normal();

  mag[0]=(1278.05*mag_x/bias_x-mag_offset[6])*SENSOR_SIGN[6];
  mag[1]=(1193.30*mag_y/bias_y-mag_offset[7])*SENSOR_SIGN[7];
  mag[2]=(1213.08*mag_z/bias_z-mag_offset[8])*SENSOR_SIGN[8]*1.16/1.08;
}
```

ANEXO F. DCM.cpp

```
// Código realizado para este Trabajo de Investigación por el autor
// DCM support for ArduIMU V3

void get_initial_values()
{
  int i;
  float T[3][3]={0,0,0,0,0,0,0,0,0};

  for(i=0;i<20;i++)
  {
    MPU6000_Read();
    HMC5883_Read();

    Vector_Add(&accel[0], T[0], T[0]);
    Vector_Add(&mag[0], T[1], T[1]);
  }
  tiempo[0]=millis();

  Vector_Normalize(T[0], Kb[0]);
  Vector_Normalize(T[1], T[1]);

  ERR=Vector_Dot_Product(Kb[0], T[1]);
  Vector_Scale(-1.0*ERR, Kb[0], T[0]);
  Vector_Add(T[0], T[1], T[2]);
  Vector_Normalize(T[2], Ib[0]);
```

```
//  T[2][0]=T[1][0];
//                          T[1][0]=T[2][0]*cos(-MAGNETIC_DECLINATION)-T[1][1]*sin(-
MAGNETIC_DECLINATION);
//                          T[1][1]=T[2][0]*sin(-MAGNETIC_DECLINATION)+T[1][1]*cos(-
MAGNETIC_DECLINATION);

  Vector_Cross_Product(Kb[0], Ib[0], Jb[0]);
  Vector_Normalize(Jb[0], Jb[0]);
}

void get_DCM(float* Ib, float* Jb, float* Kb)
{
  byte i;

  for(i=0;i<3;i++)
  {
    DCM[i][0]=*Ib;
    Ib++;
  }
  for(i=0;i<3;i++)
  {
    DCM[i][1]=*Jb;
    Jb++;
  }
  for(i=0;i<3;i++)
  {
    DCM[i][2]=*Kb;
    Kb++;
  }
}
```

```
void get_Tait_Bryan()
{
  float roll, pitch, yaw;

  pitch=asin(DCM[0][2]);
  T_B[1]=pitch*180.0/M_PI;

  roll=atan2(-DCM[1][2],DCM[2][2]);
  T_B[0]=roll*180.0/M_PI;

  yaw=atan2(-DCM[0][1],DCM[0][0]);
  T_B[2]=yaw*180.0/M_PI;
}

void get_Tait_Bryan2()
{
  float lambda, sigma, roll, pitch, yaw;

  lambda=atan2((DCM[2][1]-DCM[1][0]),(DCM[1][1]+DCM[2][0]));
  sigma=atan2((DCM[2][1]+DCM[1][0]),(DCM[1][1]-DCM[2][0]));

  roll=(sigma+lambda)/2;
  T_B2[0]=roll*180.0/M_PI;

  pitch=atan2(DCM[0][2],(cos(roll)*DCM[2][2]-sin(roll)*DCM[1][2]));
  T_B2[1]=pitch*180.0/M_PI;

  yaw=(sigma-lambda)/2;
  T_B2[2]=yaw*180.0/M_PI;
}
```

```
void get_dt()
{
  tiempo[1]=millis();
  dt=(tiempo[1]-tiempo[0])/1000;
  tiempo[0]=tiempo[1];
}

void get_new_values()
{
  get_dt();
  get_dTg();

  MPU6000_Read();
  HMC5883_Read();
  float S[3][3];

  Vector_Normalize(&accel[0], Kb[1]);
  Vector_Normalize(&mag[0], S[0]);
  ERR=Vector_Dot_Product(Kb[1], S[0]);
  Vector_Scale(-1.0*ERR, Kb[1], S[1]);
  Vector_Add(S[0], S[1], S[2]);
  Vector_Normalize(S[2], Ib[1]);
  Vector_Cross_Product(Kb[1], Ib[1], Jb[1]);
  Vector_Normalize(Jb[1], Jb[1]);
  get_dTa();
  get_dTm();
  get_dT();
  Vector_Scale(M_PI/180.0, &dT[0], &dT[0]);

  float L[3][3];
```

```
  Vector_Cross_Product(&dT[0], Kb[0], L[2]);
  Vector_Add(Kb[0], L[2], Kb[2]);
  Vector_Normalize(Kb[2], Kb[2]);
  Vector_Cross_Product(&dT[0], Ib[0], L[0]);
  Vector_Add(Ib[0], L[0], Ib[2]);
  Vector_Normalize(Ib[2], Ib[2]);

  Vector_Cross_Product(Kb[2], Ib[2], L[1]);
  Vector_Normalize(L[1], Jb[2]);
}

void get_dTg()
{
  Vector_Scale(1.00/131, &gyro[0], &W[0]);                    //   get   W
[º/s]
  Vector_Scale(dt, &W[0], &dTg[0]);                           //
dTg=dt(W) [º]
}

void get_dTa()
{
  float A[3][3];
  Vector_Scale(-1.00, Kb[0], A[0]);                     // T[0]=-Kb[0]
  Vector_Add(Kb[1], A[0], A[1]);                  // T[1]=Kb[1]-Kb[0]
  Vector_Cross_Product(Kb[0], A[1], A[2]);              //
T[2]=Kb[0]x(Kb[1]-Kb[0])
  Vector_Scale(180.0/M_PI, A[2], &dTa[0]);        // dTa [º]
}
```

```
void get_dTm()
{
  float M[3][3];
  Vector_Scale(-1.00, Ib[0], M[0]);                                    // T[0]=-Ib[0]
  Vector_Add(Ib[1], M[0], M[1]);                                  // T[1]=Ib[1]-Ib[0]
  Vector_Cross_Product(Ib[0], M[1], M[2]);                             //
T[2]=Ib[0]x(Ib[1]-Ib[0])
  Vector_Scale(180.0/M_PI, M[2], &dTm[0]);                             // dTm [º]
}

void get_dT()
{
  float T[4][3];
  Vector_Scale(Sg/(Sg+Sa+Sm), &dTg[0], T[0]);
  Vector_Scale(Sa/(Sg+Sa+Sm), &dTa[0], T[1]);
  Vector_Scale(Sm/(Sg+Sa+Sm), &dTm[0], T[2]);
  Vector_Add(T[0], T[1], T[3]);
  Vector_Add(T[2], T[3], &dT[0]);
}

void reset_initial_values()
{
    Vector_Scale(1.00, Kb[2], Kb[0]);                             // Kb[0]=Kb[1]
    Vector_Scale(1.00, Ib[2], Ib[0]);                                    // Ib[0]=Ib[1]
    Vector_Scale(1.00, Jb[2], Jb[0]);                             // Ib[0]=Ib[1]
}
```

ANEXO G. ARDUIMU.cpp

```
// Código realizado para este Trabajo de Investigación por el autor
// Local magnetic declination (in radians)
// I used this web : http://www.ngdc.noaa.gov/geomag-web/
// corrects magnetic bearing to true north
//#define MAGNETIC_DECLINATION 0.143310924                    //Cúcuta
(8º12'40")
#define MAGNETIC_DECLINATION 0.134283693                    //Bmanga
(7º41'38")

#define Sg 0.86
#define Sa 0.07
#define Sm 0.07

//Mix
int SENSOR_SIGN[]= {-1,-1,-1,1,1,1,-1,-1,1};          // GX, GY, GZ, AX, AY, AZ,
MX, MY, MZ
float mag_offset[9];
unsigned long time;

// Matrix
float Ib[4][3], Jb[4][3], Kb[4][3], DCM[3][3], T_B[3], T_B2[3], W[3];   // GYRO,
ACCEL, MAG
float tiempo[2], ERR, dt, dTg[3], dTa[3], dTm[3], dT[3];

// HMC5883L
int mag_x, mag_y, mag_z;
```

```
int bias_x, bias_y, bias_z;
float heading = 0;
float mag[3];

// MPU6000
float accel[3], gyro[3];
double gravedad=0, tempt = 0;
int temp = 0;

void setup()
{
  // Para comunicarse con el computador, utilice una de estas velocidades: 4800,
9600, 14400,
     19200, 28800, 38400, 57600 o 115200.
  Serial.begin(115200);
  Serial.println("Inicio del programa");
  Serial.println("Inicializacion del MPU6000:");
  MPU6000_Init();
  Serial.println("Inicializacion del HMC5883L:");
  HMC5883_init();
  get_initial_values();

  get_DCM(Ib[0], Jb[0], Kb[0]);
  Serial.println("\t");
  for(int i=0;i<3;i++)
  {
    Serial.print("|");
    for(int j=0;j<3;j++)
    {
      if(j!=0)
```

```
      {
        Serial.print("\t");
      }
      Serial.print(DCM[i][j], 10);
    }
    Serial.println("|");
  }

  get_Tait_Bryan();
  Serial.print("Formula 1:\t[");
  Serial.print(T_B[0]);
  Serial.print("\t\t");
  Serial.print(T_B[1]);
  Serial.print("\t\t");
  Serial.print(T_B[2]-MAGNETIC_DECLINATION*180/M_PI);
  Serial.println("\t]");

  get_Tait_Bryan2();
  Serial.print("Formula 2:\t[");
  Serial.print(T_B2[0]);
  Serial.print("\t\t");
  Serial.print(T_B2[1]);
  Serial.print("\t\t");
  Serial.print(T_B2[2]-MAGNETIC_DECLINATION*180/M_PI);
  Serial.println("\t]");
}

void loop()
{
  get_new_values();                          //  get Kb[1], Ib[1] and Jb[1]
```

```
  get_DCM(Ib[2], Jb[2], Kb[2]);
  get_Tait_Bryan();
  get_Tait_Bryan2();
  reset_initial_values();
  printdata();
}
```

ANEXO H. Matrix_Algebra.cpp

```
// Código realizado para este Trabajo de Investigación por el autor
// Matrix Algebra support for ArduIMU V3//
//Computes the dot product of two vectors
float Vector_Dot_Product(float v1[3],float v2[3])
{
  float op=0;
    for(int c=0; c<3; c++)
  {
    op+=v1[c]*v2[c];
  }
  return op;
}

//Computes the cross product of two vectors
void Vector_Cross_Product(float v1[3], float v2[3], float vOut[3])
{
  vOut[0]= (v1[1]*v2[2])-(v1[2]*v2[1]);
  vOut[1]= (v1[2]*v2[0])-(v1[0]*v2[2]);
  vOut[2]= (v1[0]*v2[1])-(v1[1]*v2[0]);
}

//Multiply the vector by a scalar.
void Vector_Scale(float scale2, float vIn[3], float vOut[3])
{
  for(int c=0; c<3; c++)
  {
```

```
    vOut[c]=scale2*vIn[c];
  }
}
void Vector_Add(float vIn1[3], float vIn2[3], float vOut[3])
{
  for(int c=0; c<3; c++)
  {
    vOut[c]=vIn1[c]+vIn2[c];
  }
}
float Vector_Magnitude(float* vector)
{
  byte i;
  float magnitude=0;
  magnitude=Vector_Dot_Product(&vector[0],&vector[0]);
  magnitude=sqrt(magnitude);
  return magnitude;
}

void Vector_Normalize(float* vIn, float* vOut)
{
  signed int i;
  float magnitude;
  magnitude=Vector_Magnitude(&vIn[0]);
  magnitude=1.0/magnitude;
  Vector_Scale(magnitude, &vIn[0], &vOut[0]);
}
```

ANEXO I. Output.cpp

```
// Código realizado para este Trabajo de Investigación por el autor

void printdata(void)
{
  Serial.println("\t");
  for(int i=0;i<3;i++)
  {
    Serial.print("|");
    for(int j=0;j<3;j++)
    {
      if(j!=0)
      {
        Serial.print("\t");
      }
      Serial.print(DCM[i][j], 10);
    }
    Serial.println("|");
  }
  Serial.print("Formula 1:\t[");
  Serial.print(T_B[0]);
  Serial.print("\t\t");
  Serial.print(T_B[1]);
  Serial.print("\t\t");
  Serial.print(T_B[2]-MAGNETIC_DECLINATION*180/M_PI);
  Serial.println("\t]");
  Serial.print("Formula 2:\t[");
```

```
  Serial.print(T_B2[0]);
  Serial.print("\t\t");
  Serial.print(T_B2[1]);
  Serial.print("\t\t");
  Serial.print(T_B2[2]-MAGNETIC_DECLINATION*180/M_PI);
  Serial.println("\t]");
}
```