IMPLEMENTACION DE UNA UNIDAD DE MEDICION INERCIAL (*IMU*) EN ARDUPILOT 2.5

GABRIELA BERMUDEZ MARQUEZ

CARLOS AUGUSTO NIÑO DIAZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERIAS FÍSICO-MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2014

IMPLEMENTACION DE UNA UNIDAD DE MEDICION INERCIAL (*IMU*) EN ARDUPILOT 2.5

GABRIELA BERMUDEZ MARQUEZ

CARLOS AUGUSTO NIÑO DIAZ

Trabajo de Grado para optar al título de

Ingeniero Electrónico

Director

M. Sc ALFREDO RAFAEL ACEVEDO PICÓN

Ingeniero Electrónico

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERIAS FÍSICO-MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

BUCARAMANGA

2014

DEDICATORIAS

Primero, gracias a Dios por permitirme cumplir esta meta y darme fuerza en mis momentos de flaqueza.

Gracias a mis padres, quienes me motivaban a superar cada obstáculo que se me presentaba y a esforzarme cada vez más.

A quienes en Bucaramanga me acogieron como su hija: mis tíos Cesar y Carolina y mis primos Cesar y Sulvey, gracias por el apoyo y la compañía continua.

A mis hermanas y toda mi familia quienes siempre estuvieron pendiente de mi progreso y bienestar, enviándome en todo momento buena energía para seguir adelante.

Finalmente, pero no menos importante, a mi mamá Olivia, esa abuela alcahueta, comprensiva y cariñosa que estaba ahí siempre que quería una palabra de aliento o un buen consejo, además de preocupada siempre por mi bienestar.

Gabriela Bermúdez Márquez.

A mis padres Maria Cristina y Carlos Vidal, por entregarme lo más valioso que un padre puede darle a sus hijos: Tiempo

Carlos A. Niño Díaz.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMEN

TITULO: Implementación de una unidad de medición inercial (*IMU*) en Ardupilot Mega 2.5.*

AUTORES: Gabriela Bermúdez Márquez. **
Carlos Augusto Niño Díaz. **

PALABRAS CLAVES: Ardupilot, *DCM*, Unidad de medición inercial, Matriz de rotación, Controlador PI.

Descripción:

En el presente proyecto se busca implementar una unidad de medición inercial (*IMU*) en el *Ardupilot* Mega 2.5, con información proveniente de los sensores de dicha tarjeta (giróscopo, magnetómetro, acelerómetro) además de un GPS, mediante el método de la matriz de cosenos directores (*DCM*).

Se construirá un algoritmo el cual involucra un proceso de creación de una matriz que se elabora en base a los datos provenientes de los sensores en tiempo real. A partir de ésta matriz, llamada matriz DCM inicial, se realizarán una serie de correcciones y ortonormalizaciones en cada iteración. Éste procedimiento busca mantener la integridad de la matriz *DCM*, la cual es fuente primaria de información para conocer la posición de un objeto y núcleo principal de la teoría propuesta por *Premerlani*/*Bizard.* La fuente principal de información para construir la *IMU* es el giróscopo, es por esto que se acopla un controlador PI para corregir los datos provenientes del mismo. Además, empleando la información que se adquiere del acelerómetro se corrigen los elementos de la matriz que permiten hallar los ángulos *roll-pitch*. Del mismo modo el magnetómetro/GPS se usa para corregir los elementos de la matriz que permiten hallar el ángulo *yaw*. Por último se realizarán una serie de pruebas implementando un protocolo que verifique la confiabilidad de los datos obtenidos de la IMU.

---

* Proyecto de Grado
** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Alfredo Rafael Acevedo Picón, MSc.

ABSTRACT

TITLE: Implementation of inertial measurement unit (*IMU*) in Ardupilot 2.5. *

AUTHORS: Gabriela Bermúdez Márquez. **
Carlos Augusto Niño Díaz. **

KEYWORDS: Ardupilot, DCM, Inertial Measurement Unit, Rotation Matrix, Controller PI.

Description:

This project is about to elaborate an inertial measurement unit (IMU) in the Ardupilot Mega 2.5 using information from gyroscope, magnetometer, GPS and accelerometer, through the method of the direction cosine matrix (DCM).

First of all the DCM matrix is calculated primarily from gyroscope sensor, then a correction and an updating of the DCM is done using the information acquired from the other sensors (accelerometer and magnetometer). The accelerometer is used to correct the information of the DCM related to the roll-pitch angles. The magnetometer/GPS is used to correct the information of the DCM related to the yaw angle. The process of creating and updating the matrix is made based on real-time data that is provide by sensors, in conjugation with a system that couples a PI controller to correct the data from the gyro. The gyro is the main source of the DCM; this is the reason why it is necessary to correct it. DCM matrix is the main core of the Premerlani/Bizard theory. Using the information of the elements of the matrix it is possible to known the 3 angles (Roll, Pitch and Yaw) that defines a body in the three-dimensional world. Additionally, a test protocol to verify the reliability of the data obtained from the IMU will be performed.

---

* Degree Work
** Physico-mechanical Engineering Faculty. School of Electrical Engineering. Supervisor: Alfredo Rafael Acevedo Picón, MSc.

INTRODUCCION

La convergencia de sensores ha contribuido al desarrollo de diversos dispositivos electrónicos que usan la información que proviene de ellos para determinar características del sistema, por ejemplo, para el caso específico del presente proyecto, la posición de un objeto en el espacio; dichos dispositivos disponen de la ayuda de sensores como acelerómetro, giróscopo, magnetómetro, GPS, entre otros para llevar a cabo dicho fin.

La sensibilidad de los sensores es un factor importante al momento de elegir la aplicación, considerando objetos con variaciones a gran escala (proyectos de exploración) o a pequeña escala (robótica). Para el caso específico en el cual el objetivo es obtener la posición de un objeto, se requiere de una intervención específica de diversas variables provenientes de los sensores nombrados anteriormente (para cada aplicación, existen formas diferentes de usar la información proveniente de los sensores). Realizando un correcto procesamiento de estas, se puede determinar la posición en tiempo real del objeto deseado con sus respectivas variaciones en el espacio respecto a un marco de referencia establecido.

El Ardupilot Mega 2.5 es un sistema de desarrollo que cuenta con diversos sensores, como los mencionados anteriormente, y es a partir de estos (en suma con un GPS) que es posible construir una *IMU* para ubicar un objeto en el espacio por medio de los ángulos de Euler utilizando la metodología de la *Matriz de Cosenos Directores*. Por lo tanto, este sistema permite realizar navegación y control de vehículos no tripulados. El *APM* 2.5 posee un microprocesador ATMEGA 2560 capaz de realizar rápidamente las operaciones lógicas necesarias para implementar el algoritmo que genera la *DCM*. Teniendo en cuenta lo expuesto anteriormente, el *APM* 2.5 es un dispositivo que cuenta con el hardware necesario para realizar construir una *IMU*.

La meta del proyecto es implementar una unidad de medición inercial (*IMU*), que brinde información sobre los 3 ángulos que definen la posición de un cuerpo en el espacio tridimensional (*Roll, Pitch y Yaw*), actualizando la información de los

mismos en tiempo real; una aplicación casi inmediata donde la *IMU* cobra un valor primordial es el control de vehículos aéreos no tripulados (UAV), debido a que sin esta información es imposible realizar control de la posición del mismo.

De los datos provenientes de los sensores relacionados anteriormente (y con un correcto procesamiento de las variables obtenidas de estos) se puede obtener la información necesaria para determinar las condiciones del dispositivo que se requieran.

El procesamiento de las variables se basa en la teoría de la matriz de cosenos directores (*DCM*) en la cual se usa principalmente la información proveniente del giróscopo (acelerómetro y GPS son correctores en la teoría) en suma con un controlador PI; la función de este sistema es mantener las condiciones de estructura (ortogonalidad) y forma de la *DCM*. Por lo tanto en medio del proceso para conseguir una *DCM* fiable, es necesario compilar un código que permita corregir la matriz en tiempo real, éste proceso de creación y corrección de la matriz permiten finalmente mantener la integridad de la misma en cada iteración. Matemáticamente hablando la *DCM* fue diseñada para calcular los ángulos de Euler, por lo tanto, una vez asegurada la integridad de la *DCM*, se pueden obtener los ángulos por medio de una operación trigonométrica entre algunos de los elementos de la matriz.

1. DESCRIPCION DEL PROYECTO

1.1. FORMULACION DEL PROBLEMA

Los sistemas de desarrollo como el Ardupilot, manejan prestaciones de hardware abiertas, pero las empresas que desarrollan el software no permiten modificar el código fuente de forma abierta, completa y/o parcial. Este tipo de algoritmos se conocen como código privativo. En este proyecto, se pretende implementar un código abierto (*open source*) que permita acceder a prestaciones tales como obtener la posición de un cuerpo en el espacio euclidiano.

El problema específico objeto de la investigación encuentra su fundamento en que el código fuente no se encuentra disponible para su modificación total o parcial. La idea de este proyecto es crear uno que permita al usuario aprovechar los recursos electrónicos del *APM* 2.5 (configurado como *IMU*) de tal forma que no haya necesidad de hacer uso de software privativo. Dicho código deberá poder entregar información actualizada sobre la posición actual del dispositivo, es decir datos referentes a los ángulos que en el espacio tridimensional, definen la posición de un objeto.

1.2. OBJETIVOS

1.2.1. Objetivo general

Desarrollar el código para Ardupilot 2.5 que implemente una unidad de medición inercial (*IMU*) basada en la técnica de la matriz de cosenos directores (*DCM*).

1.2.2. Objetivos especificos

- Construir e implementar un algoritmo en Ardupilot Mega 2.5 que permita crear una unidad de medición inercial a partir del método de la matriz de cosenos directores propuesta por Premerlani y Bizard.
- Comparar las mediciones tomadas por el algoritmo de posición realizado, con respecto a la ubicación real del vehículo, evaluando la fidelidad del proceso.
- Realizar un protocolo de pruebas de posicionamiento del vehículo para verificar el funcionamiento del algoritmo.
- Implementar un algoritmo de un controlador digital en el sistema, que permita corregir la *DCM*, comparando con la posición actual deseada.

1.3. JUSTIFICACION

La implementación de la unidad de medición inercial (*IMU*) en el *APM* 2.5 se realiza con el motivo de conocer la posición de un objeto hipotético. Dado que la implementación de una *IMU* necesariamente involucra hardware y software, se hace necesario no solo el dispositivo que acople todos los sensores que permiten encontrar la posición del objeto anteriormente mencionado de prueba, sino también el algoritmo que permita utilizar la información proveniente de los sensores para calcular los ángulos de Euler. El hardware está disponible en el entorno de desarrollo Ardupilot, pero dada la carencia de un software abierto, no es posible generar cambios que se acomoden a las necesidades puntuales de cada usuario.

Supone una gran ventaja en el área académica disponer de diversos recursos *open source* debido a la filosofía del código abierto: libre y completamente modificable. Por lo anteriormente mencionado, uno de los entregables de este proyecto es un código libre para construir una IMU a partir del hardware que está contenido en el Ardupilot Mega utilizando el método de la *DCM* e implementando un esquema que involucre un controlador digital.

En concordancia con lo anterior, debido a que existe una carencia de software abierto al usuario, no es posible generar cambios que se acomoden a las necesidades puntuales de un proyecto dado. Por lo tanto, se hace necesario desarrollar un código abierto que permita al usuario personalizarlo de acuerdo a los requerimientos del proyecto.

2. MARCO TEORICO

2.1. ARDUPILOT MEGA 2.5

El Ardupilot Mega 2.5 (*APM* 2.5) es uno de los sistemas de control y navegación de vehículos no tripulados más avanzados y de libre distribución que existen. Basado en *IMU* (*Inertial Measurement Unit*) (o en español UMI: Unidad de Medición Inercial*)*, esta tarjeta de desarrollo posee diferentes sensores que le permiten conocer la posición relativa del vehículo al cual se encuentra acoplado.

Entre las características más notables del *APM* 2.5 está el “piloto automático”, un estado en el cuál hace uso de todos los sensores que posee para enviar señales directamente a los actuadores de un hipotético vehículo no tripulado, los cuales le permiten desplazarse en el espacio, mientras recibe información en tiempo real de los sensores (información que permite calcular posición, dirección, etc.). Es decir, el *APM* 2.5 tiene la capacidad de controlar un vehículo no tripulado para que se mueva en el espacio tridimensional de forma autónoma.

Los sensores con los que cuenta el *APM* 2.5 son:

- (1) MPU-6000 de 6 ejes que consisten en acelerómetro de 3 ejes, giroscopio de 3 ejes, y sensor de temperatura.
- (1) HMC-5883 que contiene un compás digital también de 3 ejes,
- (1) sensor de presión barométrica

Además posee una memoria interna para el registro de datos, distribuidos de forma precisa para sacar un máximo provecho del procesador Atmel presente en el entorno de desarrollo.

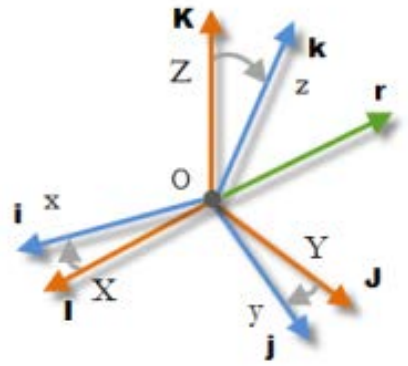
Adicionalmente a los sensores incorporados en la tarjeta, al procesador y a la memoria, cuenta con puertos de comunicación SPI, I2C, UART, entre otros, que permite conectar periféricos tales como una antena de GPS, un kit de telemetría y otras diferentes interfaces.

2.2. MATRIZ DE COSENOS DIRECTORES

La matriz de cosenos directores es una matriz de dimensiones $3x3$ en la que se guarda la información proveniente de los sensores, principalmente del giróscopo. La *DCM* se usa para convertir la información de un marco de referencia a otro, es decir, el marco de referencia relativo al *APM* 2.5 según su ubicación en el espacio y el sistema de coordenadas global, v.g. en una UAV (*unmanned aerial vehicle)* se hablaría del sistema coordenado de la tierra y del aeroplano.

Para el sistema coordenado mostrado en la Figura 1 si se requiere pasar un vector del sistema "O-x-y-z" al marco de referencia global (O-X-Y-Z) se debe multiplicar dicho vector por la *DCM*. Tanto el marco de referencia global como el marco de referencia del cuerpo tienen el mismo origen O fijo.

Figura 1. Marco de referencia y sistema de coordinadas global.



Fuente [2]

La matriz de cosenos directores (*DCM*), es el resultado del cálculo de las proyecciones de un marco de referencia dado por el sistema de coordinadas global, sobre el marco de referencia del dispositivo que está asociado a la *IMU*, por ello la *DCM* también recibe el nombre de matriz de rotación entre sistemas.

Con el fin de profundizar más en la matemática que existe detrás de la *DCM*, se muestran a continuación las ecuaciones para transformar un vector de información que se encuentra en el marco de referencia de la *IMU* (v.g. Velocidad, Aceleración, etc):

$$Q_I \rightarrow Vector\ de\ informacion\ según\ el\ marco\ de\ referencia\ de\ la\ IMU.$$

$$\boldsymbol{Q_G \rightarrow Vector\ de\ informacion\ según\ el\ marco\ de\ referencia\ global.} \quad (1$$

$$Q_G = R\ Q_I \quad (2$$

Donde R es la matriz de rotación expresada por:

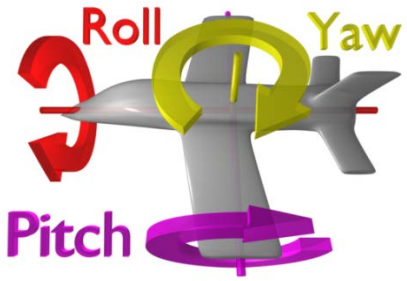
$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r_{xx}} & \boldsymbol{r_{xy}} & \boldsymbol{r_{xz}} \\ \boldsymbol{r_{yx}} & \boldsymbol{r_{yy}} & \boldsymbol{r_{yz}} \\ \boldsymbol{r_{zx}} & \boldsymbol{r_{zy}} & \boldsymbol{r_{zz}} \end{bmatrix} \quad (3$$

Como el objetivo de este proyecto es conseguir la posición del *APM* 2.5, se hace necesario introducir la relación de los ángulos de Euler con la *DCM*. Los ángulos de Euler definen la posición de un objeto en el espacio tridimensional. Esta matriz se relaciona con dichos ángulos por medio de la siguiente ecuación:

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{\cos\theta\cos\Psi} & \boldsymbol{\sin\phi\sin\theta\cos\Psi - \cos\phi\sin\Psi} & \boldsymbol{\cos\phi\sin\theta\cos\Psi + \sin\phi\sin\Psi} \\ \boldsymbol{\cos\theta\sin\Psi} & \boldsymbol{\sin\phi\sin\theta\sin\Psi + \cos\phi\cos\Psi} & \boldsymbol{\cos\phi\sin\theta\sin\Psi - \sin\phi\cos\Psi} \\ \boldsymbol{-\sin\theta} & \boldsymbol{\sin\phi\cos\theta} & \boldsymbol{\cos\phi\cos\theta} \end{bmatrix} \quad (4$$

En términos generales, las ecuaciones (2) (3) y (4) expresan como realizar el cambio de un vector medido según el marco de referencia de la *IMU*, con el marco de referencia global, en términos de los cosenos directores o de los ángulos Euler y viceversa.

Figura 2. Ángulos Euler.



Fuente Wikipedia.

Se ha dicho que el *APM* 2.5 está basado en la Unidad de Medición Inercial y en el párrafo anterior se habló de cómo convertir un vector que está en el marco de

referencial global al marco de referencia de la *IMU*. A continuación se profundiza más sobre la relación que existe entre el *APM* 2.5 y la *IMU* explicando esta relación por medio de dos tópicos: '¿Qué es una *IMU*?', '¿Por qué podemos construir una *IMU* a partir del hardware que posee el *APM* 2.5?'.

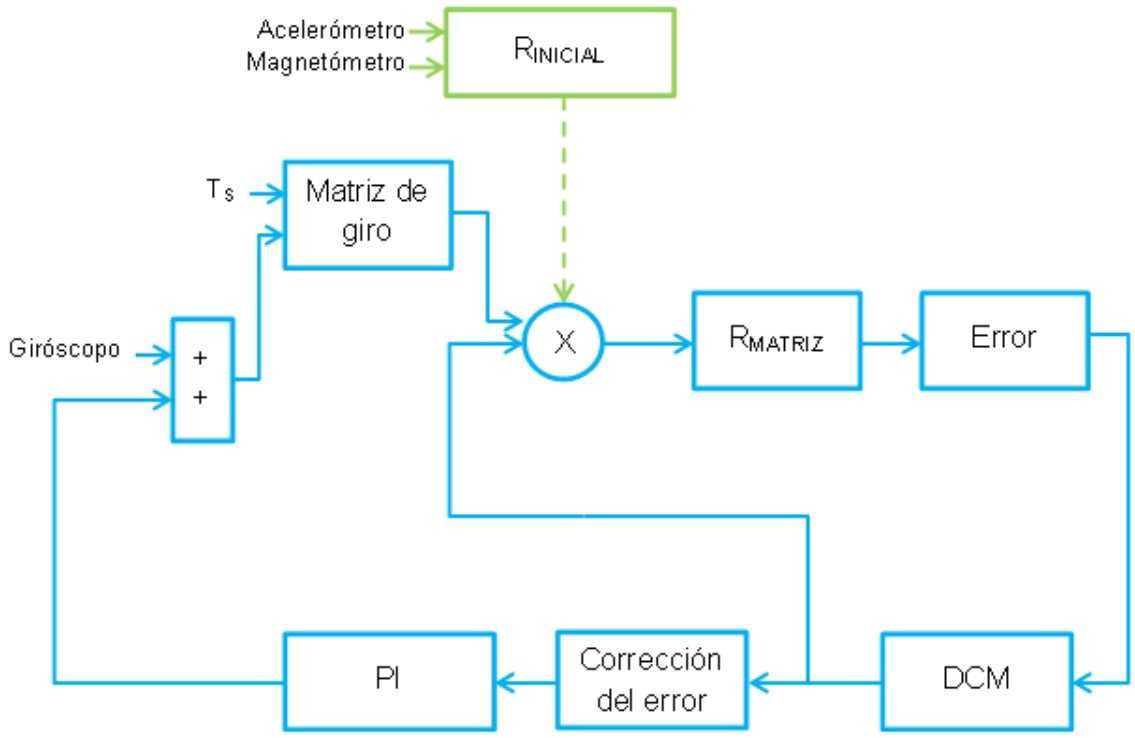
Una unidad de medición inercial (UMI) en inglés *IMU,* es un dispositivo que mide su propia posición en el espacio tridimensional (ángulos de Euler). Por lo tanto si se acopla a cualquier objeto, es posible también conocer la posición de dicho objeto. La precisión de la información que entrega la *IMU* es directamente proporcional a la cantidad de sensores que posee. Cuando se habla de una *IMU6DOF* o una *IMU9DOF*, el termino DOF hace referencia a *Degress Of Freedom* o en español, grados de libertad. Este "calificativo" que se le da a la IMU hace referencia a la cantidad de sensores que alimentan la información que de ella se obtiene. Las *IMU* 6DOF son dispositivos que usan un acelerómetro y un giróscopo de 3 ejes cada uno. Las *IMU 9DOF* son dispositivos que además de usar los mismos sensores que las *6DOF* usan un magnetómetro de 3 ejes. Por lo tanto ésta última suele ser más precisa que la primera. El *APM* 2.5 es una plataforma de desarrollo, que como se ha dicho antes, posee los 3 sensores (acelerómetro, magnetómetro y giróscopo) de 3 ejes cada uno, por lo tanto, es un circuito que potencialmente puede servir para implementar una *IMU9DOF.* Adicionalmente, en este trabajo de grado se implementa una *IMU* basada en la teoría de las matrices de cosenos directores, en la cual, además de usar el acelerómetro, magnetómetro y el giróscopo también se implementa un GPS con motivo de corregir y afinar aún más los datos que son impresos por la *IMU* (ángulos de Euler).

En definitiva, el *APM* 2.5 es una plataforma que permite implementar una *IMU* debido a su estructura interna (procesador) y a sus características de prestación (sensores). En este caso en particular, la *IMU* que se puede implementar es una *IMU9DOF*.

3. DESARROLLO DE RUTINAS

En el método propuesto por William Premerlani y Paul Bizard [11] para calcular la *DCM* se debe seguir un procedimiento en donde se inicia con el cálculo de una versión inicial de la matriz, la cual, es normalizada, corregida y ortogonalizada. En el diagrama de bloques de la Figura 3, se puede observar el proceso de forma general.

Figura 3. Diagrama de bloques.



Fuente: Autores.

La "*Matriz de giro",* construida de la información proveniente del giróscopo, es la fuente primaria de la DCM, es por esto, que el giróscopo es el sensor que provee la mayor parte de la información necesaria para construir la DCM. Solo en la primera iteración se calcula la matriz inicial, la cual es construida usando información proveniente del magnetómetro y del acelerómetro tal y como se describe en las ecuaciones (5), (6), (7), (8), (9) y (10). La matriz R debe ser

corregida con el fin de eliminar las irregularidades en la estructura de la matriz, es decir, los vectores que componen la matriz (columnas) deben ser ortogonales. El "error" que se calcula, está directamente relacionado a la falta de ortogonalidad de la matriz R. Cuando se corrige el error de ortogonalidad en la matriz R, ésta toma el nombre de matriz DCM. Este proceso inicia un bucle, el cual, como se observa en la Figura 3 es complementado por el controlador PI. La función del controlador, es básicamente corregir los datos provenientes del giróscopo utilizando datos provenientes del acelerómetro y del magnetómetro/GPS.

A continuación se desglosa de forma particular todo el procedimiento que se describió en los párrafos anteriores. Primero se calcula la versión inicial de la matriz DCM:

$$\overrightarrow{\boldsymbol{I_\iota}} = \frac{\vec{\boldsymbol{M}}}{|\vec{\boldsymbol{M}}|} \quad \therefore \quad \vec{\boldsymbol{M}} = [\boldsymbol{M_x} \quad \boldsymbol{M_y} \quad \boldsymbol{M_z}] \tag{5}$$

$$\overrightarrow{\boldsymbol{K_\iota}} = -\frac{\vec{\boldsymbol{A}}}{|\vec{\boldsymbol{A}}|} \quad \therefore \quad \overrightarrow{\boldsymbol{A}} = [\boldsymbol{A_x} \quad \boldsymbol{A_y} \quad \boldsymbol{A_z}] \tag{6}$$

$$\overrightarrow{\boldsymbol{I_{n\iota}}} = \frac{\overrightarrow{\boldsymbol{I_{np}}}}{|\overrightarrow{\boldsymbol{I_{np}}}|} \quad \therefore \quad \boldsymbol{I_{np}} = \overrightarrow{\boldsymbol{K_\iota}} \boldsymbol{x} \overrightarrow{\boldsymbol{I_\iota}} \tag{7}$$

$$\overrightarrow{\boldsymbol{K_{n\iota}}} = -\frac{\overrightarrow{\boldsymbol{K_\iota}}}{|\overrightarrow{\boldsymbol{K_\iota}}|} \tag{8}$$

$$\overrightarrow{\boldsymbol{J_{n\iota}}} = \overrightarrow{\boldsymbol{K_{n\iota}}} \, \boldsymbol{x} \, \overrightarrow{\boldsymbol{I_{n\iota}}} \tag{9}$$

En donde $\vec{A}$ y $\vec{M}$ son un vector de 3 dimensiones ('X', 'Y' y 'Z') proveniente del acelerómetro y del magnetómetro respectivamente, y las versiones normalizadas de los vectores $\vec{I},\ \vec{J},\ \vec{K}$ son las filas de la matriz DCM inicial:

$$\boldsymbol{R_{INICIAL}} = \begin{bmatrix} \overrightarrow{\boldsymbol{I_{n\iota}}} \\ \overrightarrow{\boldsymbol{J_{n\iota}}} \\ \overrightarrow{\boldsymbol{K_{n\iota}}} \end{bmatrix} \tag{10}$$

Cabe resaltar que los vectores $\overrightarrow{I_{n\iota}}, \overrightarrow{J_{n\iota}}\ y\ \overrightarrow{K_{n\iota}}$ son 1x3, formando la matriz $R_{INICIAL}$ como una matriz de 3x3, que se usa solo para la primera iteración del programa general. Después de estas operaciones se calcula la "matriz de giro", en la que intervienen los valores del giróscopo (W) y el periodo de muestreo de los sensores

(T_S=0.02 s). Este periodo de muestreo fue seleccionado teniendo en cuenta dos criterios:

- La frecuencia máxima de operación de los sensores.
- A mayor velocidad de operación, mayor precisión en los datos obtenidos.

La frecuencia máxima recomendada de trabajo para el MPU6000 es de 50 [Hz], es decir, en cada segundo, el acelerómetro, el giróscopo y el sensor de temperatura calculan 50 datos nuevos. Por lo tanto, para obtener la mayor sensibilidad posible en las mediciones, se selecciona un periodo de muestreo correspondiente a

$$Ts = 1/50 = 0.02\ [s]$$

La matriz de giro se calcula como se aprecia en la expresión 11:

$$\boldsymbol{R_G} = \begin{bmatrix} \mathbf{1} & -\boldsymbol{W_z} * \boldsymbol{T_s} & \boldsymbol{W_y} * \boldsymbol{T_s} \\ \boldsymbol{W_z} * \boldsymbol{T_s} & \mathbf{1} & -\boldsymbol{W_x} * \boldsymbol{T_s} \\ -\boldsymbol{W_y} * \boldsymbol{T_s} & \boldsymbol{W_x} * \boldsymbol{T_s} & \mathbf{1} \end{bmatrix} \quad (11$$

Al multiplicar esta matriz R_G(matriz de giro) por la matriz R que se calcula en cada iteración, se actualiza la matriz *DCM* con las señales del giróscopo según como esté variando la *IMU* en el espacio, pero después de esto es necesario renormalizar los vectores y verificar la ortogonalidad entre ellos.

$$\boldsymbol{R(t+dt)} = \boldsymbol{R(t)} \begin{bmatrix} \mathbf{1} & -\boldsymbol{W_z} * \boldsymbol{T_s} & \boldsymbol{W_y} * \boldsymbol{T_s} \\ \boldsymbol{W_z} * \boldsymbol{T_s} & \mathbf{1} & -\boldsymbol{W_x} * \boldsymbol{T_s} \\ -\boldsymbol{W_y} * \boldsymbol{T_s} & \boldsymbol{W_x} * \boldsymbol{T_s} & \mathbf{1} \end{bmatrix} \quad (12$$

El proceso de Re-normalización se hace para evitar pequeños errores acumulados que reduzcan las condiciones de ortogonalidad, es decir, este proceso obliga a los vectores que componen la matriz *DCM* a ser lo más perpendiculares posible para evitar inexactitud en los resultados de la *DCM*. El error debido a la desviación de la ortogonalización se presenta mediante el producto punto del vector r_x y el vector r_y transpuesto; si los vectores son ortogonales entre sí, el producto punto entre ellos debería ser cero, y por esto es considerado el error de ortogonalidad.

$$\boldsymbol{error} = \boldsymbol{X} \cdot \boldsymbol{Y^T} = [\boldsymbol{r_{xx}} \quad \boldsymbol{r_{xy}} \quad \boldsymbol{r_{xz}}] \begin{bmatrix} \boldsymbol{r_{yx}} \\ \boldsymbol{r_{yy}} \\ \boldsymbol{r_{yz}} \end{bmatrix} \quad (13$$

Se considera que el error es aportado por mitades de cada vector, es decir, la no ortogonalidad se debe mitad al vector X y mitad al Y, y es por esto que se divide en dos, como se puede apreciar en la expresión 14 y 15:

$$\begin{bmatrix} \boldsymbol{r_{xx}} \\ \boldsymbol{r_{xy}} \\ \boldsymbol{r_{xz}} \end{bmatrix}_{\boldsymbol{ortogonal}} = \boldsymbol{X_{ortogonal}} = \boldsymbol{X} - \frac{\boldsymbol{error}}{\boldsymbol{2}}\boldsymbol{Y} \quad (14$$

$$\begin{bmatrix} \boldsymbol{r_{yx}} \\ \boldsymbol{r_{yy}} \\ \boldsymbol{r_{yz}} \end{bmatrix}_{\boldsymbol{ortogonal}} = \boldsymbol{Y_{ortogonal}} = \boldsymbol{Y} - \frac{\boldsymbol{error}}{\boldsymbol{2}}\boldsymbol{X} \quad (15$$

Con este paso se puede verificar el error de ortogonalidad fue reducido y llevado a cero después de algunas iteraciones, manteniendo las magnitudes de cada fila y columna de la matriz es aproximadamente igual a 1. Después de reducir el error en el vector X y Y de la *DCM*, se hace necesario ajustar la fila Z mediante el producto cruz de las dos anteriores, asegurando así la ortogonalidad entre todas las filas de la *DCM*.

$$\begin{bmatrix} \boldsymbol{r_{zx}} \\ \boldsymbol{r_{zy}} \\ \boldsymbol{r_{zz}} \end{bmatrix}_{\boldsymbol{ortogonal}} = \boldsymbol{Z_{ortogonal}} = \boldsymbol{X_{ortogonal}} \times \boldsymbol{Y_{ortogonal}} \quad (16$$

El último paso en el proceso de re-normalización es escalar las filas de la matriz *DCM* para asegurar que cada una tenga magnitud igual a uno. Aunque las magnitudes casi nunca serán mayores a uno, pero para verificar se realiza la expansión de Taylor como se aprecia en la expresión No. 17.

$$X_{normalizada} = \frac{1}{2}\left(3 - X_{ortogonal} \cdot X_{ortogonal}\right)X_{ortogonal}$$

$$\boldsymbol{Y_{normalizada}} = \frac{\boldsymbol{1}}{\boldsymbol{2}}\left(\boldsymbol{3} - \boldsymbol{Y_{ortogonal}} \cdot \boldsymbol{Y_{ortogonal}}\right)\boldsymbol{Y_{ortogonal}} \quad (17$$

$$Z_{normalizada} = \frac{1}{2}\left(3 - Z_{ortogonal} \cdot Z_{ortogonal}\right)Z_{ortogonal}$$

Con los pasos mencionados anteriormente se consigue una matriz de cosenos directores normalizada y ortogonal entre sí. Al finalizar el proceso de normalización de la matriz *DCM*, se realiza una corrección del desvío de la *IMU* a partir de los datos provenientes del GPS (Curso sobre la tierra – *Course over ground*(*COG*) y Velocidad)

El *COG* es un ángulo que indica la trayectoria de navegación del vehículo respecto al norte verdadero y es a partir de este que se puede indicar la cantidad de desviación del vehículo según los datos de la matriz R. Primero, es necesario establecer un vector de referencia, para eso se debe obtener el seno y coseno del *COG* en el marco de referencia de la tierra, y así obtener la corrección del ángulo *Yaw* respecto a esta.

$$\boldsymbol{COG_X = \cos cog} \qquad \boldsymbol{COG_Y = \sin cog} \tag{18}$$

$$\boldsymbol{Corrección YawTierra = r_{xx} COG_Y - r_{yx} COG_X} \tag{19}$$

Teniendo la corrección del ángulo *Yaw* respecto a tierra, se cambia al marco de referencia del aire con ayuda de la *DCM* con el fin de ajustar la desviación del giróscopo.

$$\boldsymbol{Corrección YawAire = Corrección YawTierra} \begin{bmatrix} \boldsymbol{r_{zx}} \\ \boldsymbol{r_{zy}} \\ \boldsymbol{r_{zz}} \end{bmatrix} \tag{20}$$

Adicional a la corrección del ángulo de guiñada (*Yaw*), es necesaria la corrección de los ángulos de cabeceo y alabeo (*Roll y Pitch*), al igual que la corrección del ángulo *Yaw*, se debe calcular la corrección para aire y esto se hace mediante la tercera fila de la *DCM*, así:

$$\boldsymbol{Corrección RollPitchAire} = \begin{bmatrix} \boldsymbol{r_{zx}} \\ \boldsymbol{r_{zy}} \\ \boldsymbol{r_{zz}} \end{bmatrix} \times \boldsymbol{g_{reference}} \tag{21}$$

Realizando movimientos continuos en el dispositivo, el acelerómetro puede saturarse, es decir, la aceleración excede el valor límite del acelerómetro y es por esto que se introduce un error en los cálculos de los ángulos de cabeceo y alabeo. El vector $g_{reference}$ se calcula con el acelerómetro y el GPS así:

$$\boldsymbol{g_{reference}} = \begin{bmatrix} \boldsymbol{Acelerómetro_x} \\ \boldsymbol{Acelerómetro_y} \\ \boldsymbol{Acelerómetro_z} \end{bmatrix} + \boldsymbol{\omega_{giro}} \times \begin{bmatrix} \boldsymbol{velocidad} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad (22$$

Finalmente, se tiene el controlador PI para realizar la realimentación del sistema como se plantea en la Figura 3, cada corrección de desviación de los ángulos (*Yaw* y *Roll-Pitch*) es multiplicada por unos pesos y adicionados al controlador PI para, a su vez, ser añadido a la señal leída por el giróscopo y así producir un vector de giro corregido.

Primero se realiza el promedio de la corrección total, dándole pesos a la corrección de *yaw* y la de *roll-pitch*.

$$\boldsymbol{CorrecciónTotal} = \boldsymbol{W_{RP}CorrecciónRollPitchAire} + \boldsymbol{W_Y\ CorrecciónYawAire} \quad (23$$

Después, se pasa esta corrección total por el controlador PI así:

$$\omega_{PCorrección} = K_P CorrecciónTotal \quad (24$$

$$\omega_{ICorrección} = \omega_{ICorrección} + \int K_I dt CorrecciónTotal \quad (25$$

$$\omega_{Corrección} = \omega_{PCorrección} + \omega_{ICorrección} \quad (26$$

Como se mencionó anteriormente, esta corrección se adiciona a la lectura de la señal del giroscopio, y esta suma alimenta la matriz de giro que permite la actualización de la *DCM* con sus respectivos cambios en tiempo. Esta relación se presenta en la Figura 3.

$$\boldsymbol{\omega} = \boldsymbol{\omega_{giro}} + \boldsymbol{\omega_{Corrección}} \quad (27$$

El controlador PI se implementó con el fin de corregir el *offset* y el *drift* del giróscopo. La forma como lo hace es utilizando información proveniente de otros sensores para calcular el error que existe entre lo que está leyendo el giróscopo en comparación con lo que debería estar leyendo. La razón por la cual no se utiliza la información proveniente de la corrección directamente como fuente principal de información para la *DCM* es porque esta información tiene una "frecuencia de entrega" mucho menor a la que entrega el giróscopo, y la *IMU* debe calcularse a la misma frecuencia a la cual el giróscopo tiene información

nueva disponible, logrando así, detectar la mayor cantidad de movimientos en el espacio de la *IMU*.

4. PRUEBAS EXPERIMENTALES

Con el fin de corroborar la fidelidad y veracidad de los resultados del código planteado para el proyecto se realizaron dos pruebas, la primera en la que se verificaban los ángulos individualmente comparando la posición real del vehículo con la posición deseada (ángulo conocido); y la segunda prueba en la que se siguió un circuito en el que habían variaciones en *roll*, *pitch* y *yaw* también conocidas.

4.1. CALCULO DE LAS CONSTANTES

4.1.1. **Selección de pesos para la corrección *roll-pitch, yaw*:** Cada sensor posee problemas de error en sus mediciones, este error es inherente a todo dispositivo electrónico, y se vislumbra aún más en los sensores, no solo porque se esta convirtiendo una señal analógica a una señal digital, sino también porque el proceso de medir en sí, trae consigo un error innato en la medida. Por lo tanto, para corregir dichos errores (numéricos, *offset*, medición) se requiere cuantificar dichos errores para después corregir la información que se está calculando. El resultado del proceso anteriormente descrito, se ve reflejado en la teoría descrita por Premerlani y Bizard en la ecuación No. 23, donde se tiene una corrección para el *Roll-Pitch* y el *Yaw.*

W_{RP} y W_Y son pesos inherentes al proceso matemático involucrado en el cálculo de dichas correcciones y deben hallarse de forma empírica asegurando la mayor confiabilidad de los datos que se están obteniendo al final: *roll, pitch, yaw*.

Luego de realizar pruebas, se comprendió que a medida que aumentan los pesos, la velocidad con la cual responden a perturbaciones, también aumenta. Pero si se aumentan demasiado los pesos, los datos que se imprimen dejan de ser correctos. Finalmente, luego de realizar varias pruebas, se llegó a los siguientes valores:

$$W_{RP} = 2$$

$$W_Y = 3$$

Es importante entender que estos valores tienen un impacto menor en comparación con las constantes del controlador PI, en el proceso de corrección los datos provenientes giróscopo. Por lo tanto es una buena estrategia, comenzar calibrando estos dos pesos antes de realizar la sintonización empírica del controlador PI.

4.1.2. Selección de constantes del controlador pi: Para el controlador PI, se debe realizar un proceso de sintonización empírica con el fin de calcular la constante proporcional e integral $\boldsymbol{K_P}$ y $\boldsymbol{K_I}$ respectivamente. La ecuación No. 24 está relacionada con la implementación de la constante proporcional y la ecuación No. 25 con la implementación de la constante integral.

Para implementar esta última en un algoritmo, se hace necesario realizar una integración discreta, lo cual se traduce a construir un código que permita calcular la integral de forma que el error en la computación de los datos sea mínimo y la velocidad de cálculo sea aceptable. El método escogido fue la suma de *Riemann* hacia adelante.

La selección de las ganancias termina siendo una decisión entre precisión y velocidad de recuperación a las perturbaciones. Esto se ve reflejado en el compromiso que existe entre corregir el error en estado estacionario y corregir el error en estado transitorio. Al aumentar la constante proporcional, se ayuda a mejorar la velocidad de corrección del error en estado transitorio pero se debe tener la precaución de no aumentarla excesivamente ya que puede aparecer un pico que sobrepase el valor deseado. Al aumentar la constante integral, se ayuda a corregir el error en estado estacionario, teniendo la precaución de no desestabilizar el sistema.

Luego de realizar varias pruebas en donde se ubicaba el APM 2.5 en una posición conocida y teniendo en cuenta lo descrito anteriormente, las constantes obtenidas son las siguientes:

$$K_P = 13.9$$

$$K_I = 0.5$$

4.1.3. Selección de pesos de los ángulos: Para obtener los ángulos $\boldsymbol{\theta}$ (*Roll*), $\boldsymbol{\psi}$*(Pitch*), $\boldsymbol{\phi}$(*Yaw*), a partir de los elementos de la matriz se utilizan las ecuaciones 28, 29 y 30:

$$\theta = K_{P1} * \sin^{-1}(R_{31}) \quad (28$$

$$\psi = \mathrm{K_{P2}} * \sin^{-1}\left(\frac{R_{21}}{\cos\theta}\right) \quad (29$$

$$\phi = K_{P3} * \sin^{-1}\left(\frac{R_{32}}{\cos\theta}\right) \quad (30$$

Los elementos R_{31}, R_{21} y R_{32} son calculados en cada iteración y se extraen de la matriz *DCM*. Es importante entender que las unidades de los ángulos $'\theta'$, $'\psi'$ y $'\phi'$ son radianes, por lo tanto R_{31}, R_{21} y R_{32} son números que varían entre – 1 y 1. K_{P1}, $K_{P\mathbf{2}}$ y K_{P3} son un factores inherentes al ángulo y se calculan por medio de sintonización empírica, en conjunción con los pesos de las correcciones *Roll-Pitch*, *Yaw* y las constantes del controlador PI.

Para este ejercicio, se realizaron pruebas donde se hacían variar físicamente los ángulos que describían la posición de la IMU hacia valores de ángulos conocidos. Luego iteración por iteración se calculaban los ángulos a partir de la *DCM* y se corregían los factores hasta que finalmente se obtuvo un resultado satisfactorio, con un error no mayor al 2% relativo:

$$K_{P1} = -2$$

$$K_{P\mathbf{2}} = 0.85$$

$$K_{P3} = 2$$

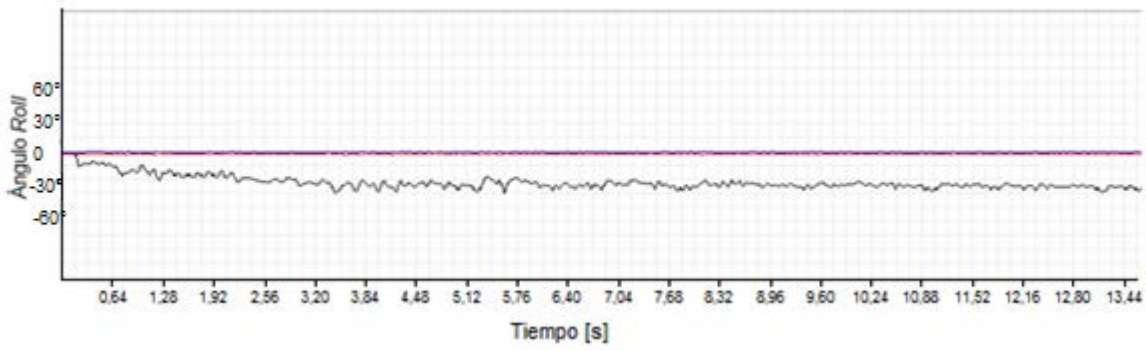
Es importante comprender que en este proceso de sintonización empírica, se deben sintonizar primero los pesos de las correcciones *Roll-Pitch* y *Yaw*. Luego de esto se sintoniza el controlador PI que corrige la información proveniente del giróscopo. Finalmente se sintonizan los pesos de los ángulos para asegurar los datos que se van a imprimir en la *IMU*.

4.2. PRUEBA DE ANGULOS (INDIVIDUAL)

En esta prueba se construyeron tres plataformas que permitían realizar variaciones de los ángulos de modo individual, desplazándose a una elevación conocida del ángulo que se deseaba comprobar y manteniendo los otros dos estables.

Para las pruebas del ángulo *roll* se giró el dispositivo sobre el eje longitudinal del vehículo (va desde el frente a la cola del dispositivo); se realizaron pruebas a 30°, 45° y 60°, los resultados se muestran en la Figura 4, Figura 5 y Figura 6.

Figura 4. Prueba *Roll* a -30°.

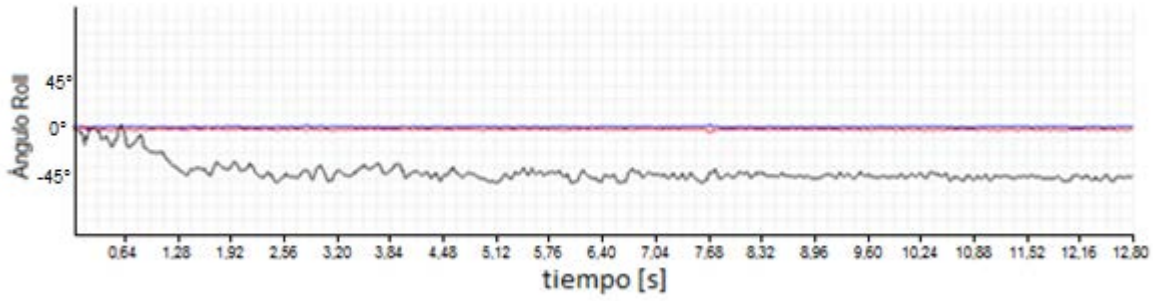


Fuente: Autores

La Figura 4 muestra la variación realizada al ángulo de alabeo, se realizó la prueba durante 13.6 [s] aproximadamente girando el vehículo gradualmente, esto lleva a un tiempo de subida de 3.5 [s]. Las rotaciones del vehículo se hacen de modo paulatino ya que los sensores tienen límites de cambio y al pasar estos las lecturas son valores indeterminados. Los valores de los ángulos al estabilizarse el vehículo se consideran admisibles ya que la diferencia con el ángulo teórico real es de 1° equivalente a 3.45% de error promedio relativo.

En la Figura 5 se grafica la variación del ángulo *roll* al realizar una inclinación del vehículo a -45° de modo gradual, de estos valores en estado estable se calculó la diferencia a partir del dato teórico medido (-45°) y su respectivo error. Para esta prueba el error promedio relativo es de 2.87%, se representa con variación de 1.3° aproximadamente.

Figura 5. Prueba *Roll* a -45°.

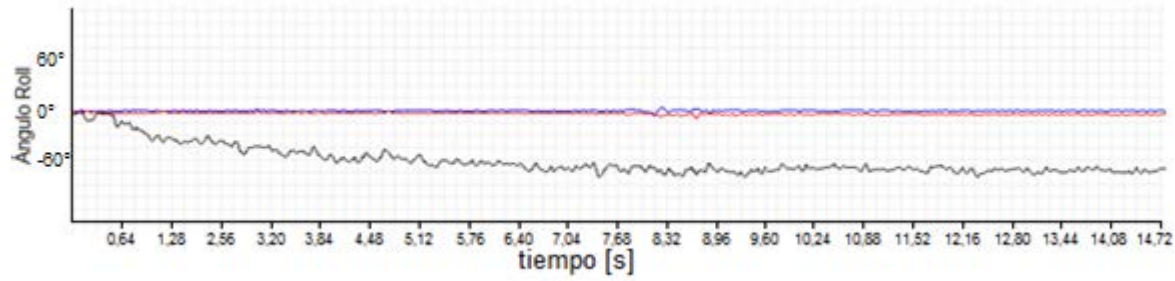


Fuente: Autores.

Al igual que en la prueba anterior, la inclinación se debe realizar gradualmente teniendo un tiempo de subida de 2.56 [s] reflejados en la gráfica, y después se mantiene estable por el tiempo necesario.

En la Figura 6 se muestra la variación del ángulo de alabeo al realizar la inclinación a -60°, de los valores obtenidos en estado estable se obtiene un valor de error promedio relativo de 3.81%, este valor se representa con una diferencia de 2.2° aproximadamente. Se tiene un tiempo de subida de 10.24 [s], mucho mayor a las pruebas anteriores ya que el recorrido para llegar a este valor es más largo y hay que girarlo lentamente para que los sensores respondan adecuadamente.

Figura 6. Prueba *Roll* a -60°.



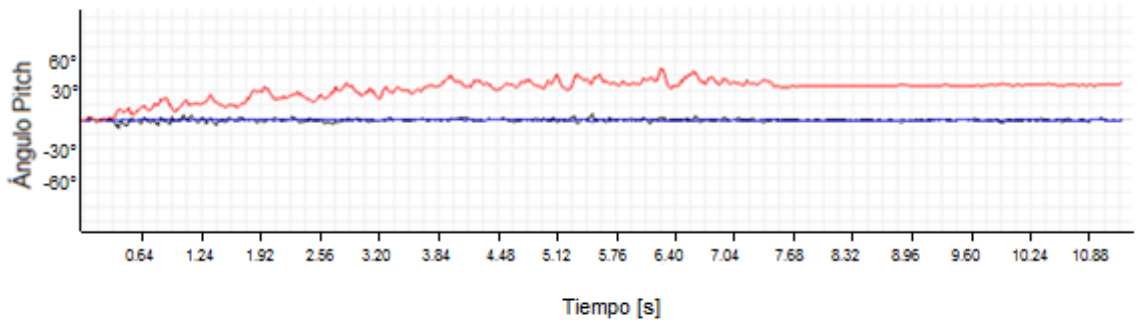
Fuente: Autores.

Como se ve en las figuras 4, 5 y 6 los datos son aproximadamente el valor que se midió al inclinar el dispositivo, y por tanto se considera una medida admisible con un error promedio de 3.37%, este error es relativo a los errores de

ortogonalización de la DCM, errores numéricos en la computación y errores debidos a la cuantificación de las señales analógicas que son convertidas a señales digitales por los sensores, entre otros.

Para la prueba del ángulo de cabeceo (*Pitch*) se realizaron movimientos a través del eje lateral del vehículo, llevándolo a un valor conocido igual que las pruebas del ángulo de alabeo. En la Figura 7, Figura 8 y Figura 9 se muestran las pruebas para 30°, 45° y 60° respectivamente.

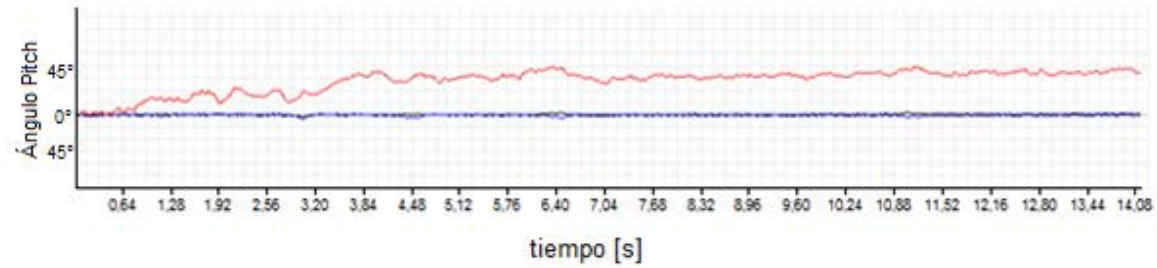
Figura 7. Prueba *Pitch* a 30°.



Fuente: Autores.

Al realizar la variación del ángulo *Pitch* y llevarlo hasta 30°, se obtienen la gráfica de la Figura 7, de la que los datos en estado estable se encuentran un error promedio relativo de 3.92%, representado en una diferencia de 1.18° aproximadamente. Para esta prueba se tomó aproximadamente 4 [s] de subida del ángulo, pero después de este tiempo logró establecerse sin mayores alteraciones.

Figura 8. Prueba *Pitch* a 45°.

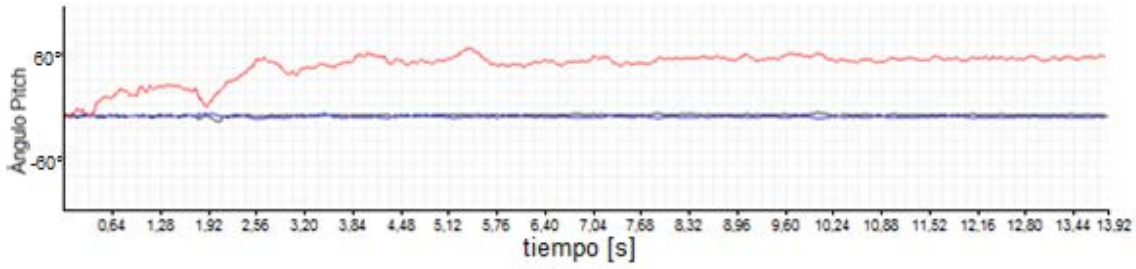


Fuente: Autores.

Para las pruebas de 45° del ángulo *Pitch* se obtiene la gráfica de la Figura 8, en este caso se presentó un error promedio relativo de 2.59%, equivalente a 1.16° aproximadamente, el tiempo de subida de esta prueba fue 3.84 [s] aproximadamente, y a partir de este momento se mantuvo en un valor relativamente estable.

Finalmente para las pruebas del ángulo *Pitch* a 60° (Figura 9) se obtiene un error promedio relativo de 2.44%, representado en una diferencia de ángulo de 1.46° aproximadamente.

Figura 9. Prueba *Pitch* a 60°.



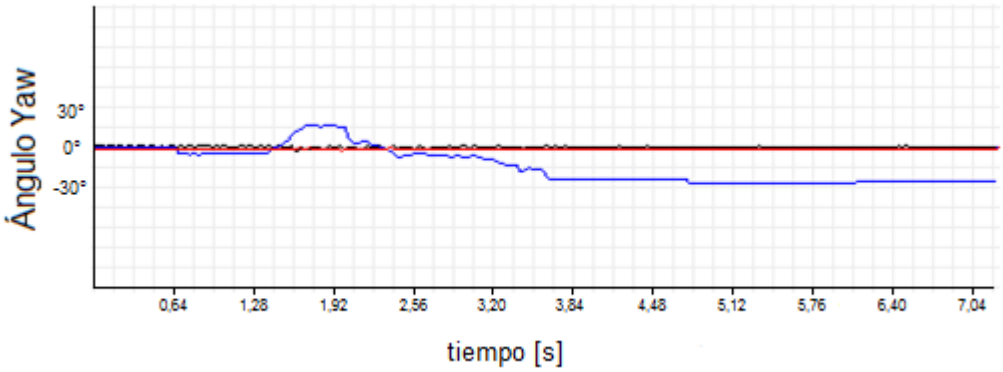
Fuente: Autores.

Con los resultados obtenidos se observa que los datos de las pruebas realizadas para el ángulo *Pitch* corresponden a las respectivas pruebas y se mantiene un error promedio de 3%, este error es relativo según la velocidad en la que se gire el vehículo y la estabilidad del mismo al momento de tomar los valores de estado estacionario.

La prueba del ángulo de *yaw* se realizó al girar el vehículo a 30°, 45° y 60° con ayuda de la plataforma diseñada para tal fin a través del eje vertical del dispositivo. Los resultados de estas pruebas se relacionan en la Figura 10, Figura 11 y Figura 12.

En la Figura 10 se grafica la respuesta del código a variaciones del ángulo de guiñada, se presenta error promedio relativo de 1.22%, equivalente a 0.36° de diferencia aproximadamente. El pico que se observa en la gráfica es debido a la conexión del GPS, ya que al no tener buena señal se envían datos que no corresponden al valor real.

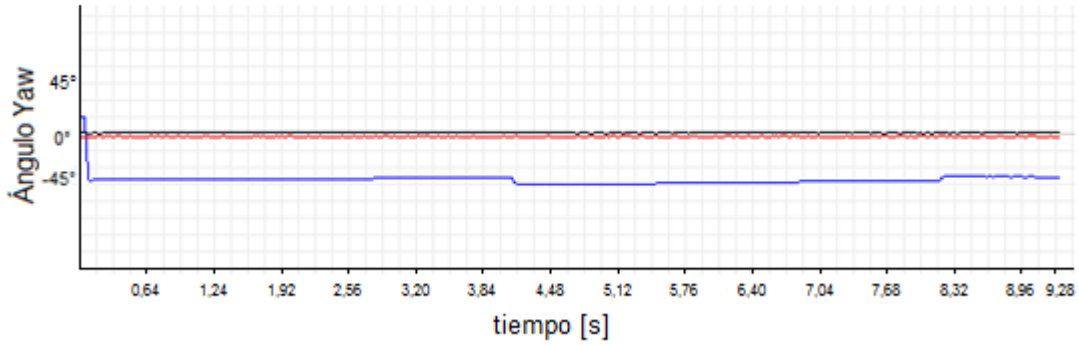
Figura 10. Prueba *Yaw* a -30°.



Fuente: Autores.

Similar a las pruebas realizadas para los otros ángulos, al girar el vehículo a través del eje vertical hasta -45° se obtienen que el error promedio de la prueba es 0.49%, reflejándose en una diferencia angular de 0.21° aproximadamente.

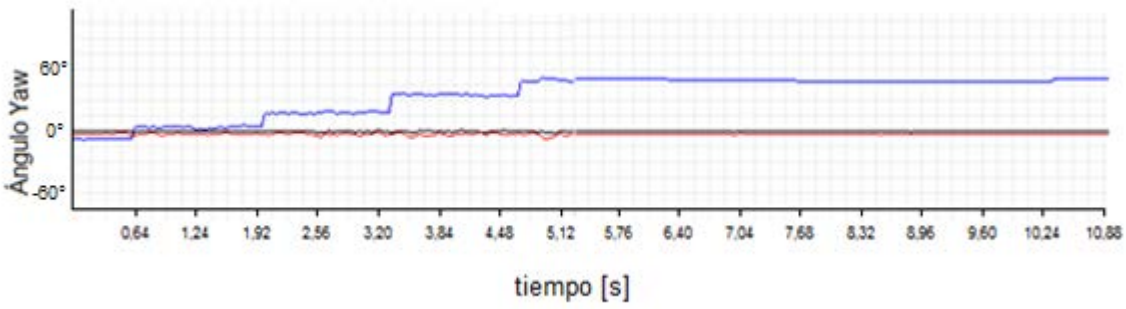
Figura 11. Prueba *Yaw* a -45°.



Fuente: Autores.

Finalmente al repetir la prueba para 60° (Figura 12), se obtiene error promedio relativo de 2.06%, equivalente a 1.23° de diferencia con el ángulo teórico. El tiempo de estabilización del ángulo es 3 [s] aproximadamente, ya que influye la velocidad con la que se giró el vehículo

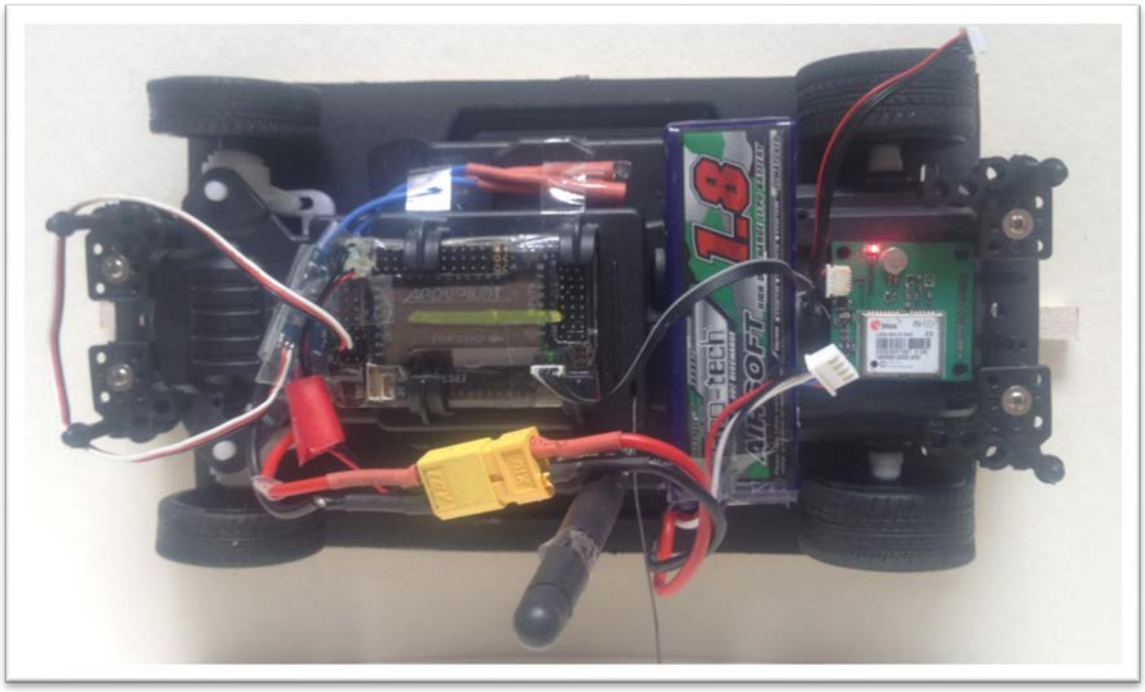
Figura 12. Prueba *Yaw* a 60°.



Fuente: Autores.

4.3. PRUEBA EN UN VEHICULO NO TRIPULADO

Figura 13. Vehículo usado para las pruebas. Toma Superior sin protector.



Fuente: Autores.

Para las pruebas en el vehículo terrestre no tripulado fue necesario un montaje remoto en un dispositivo en movimiento el cual se comunicaba por telemetría y

enviaba los datos a un centro de control fijo. El montaje incluía la batería en conjugación con el regulador de voltaje como fuente de alimentación, la antena de telemetría para enviar y recibir datos de forma inalámbrica y el GPS (ver Anexo 11). En la Figura 13 se muestra la conexión realizada en el vehículo; en el montaje local solo se necesitaba la antena de telemetría para la recepción de datos (Anexo 12) ya sea con el cable FTDI (utilizando el firmware de *Mission Planner*) o conectándola al Arduino Mega (para pruebas con el código que fue construido en el presente trabajo de grado).

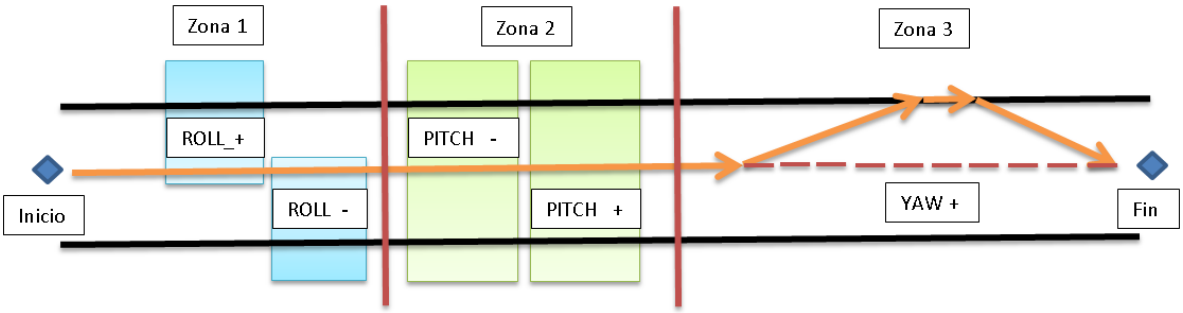
El procedimiento que se siguió es el siguiente:

1. Dibujar un circuito sobre un terreno seleccionado donde las variaciones de los ángulos (*Roll, Pitch, Yaw)* sean conocidas *(*Figura 14*).*
2. Montar los dispositivos necesarios para realizar la prueba en el vehículo terrestre. (Figura 13)
3. Cargar el firmware privativo del *Mission Planner*[1] en el *APM* 2.5
4. Realizar el recorrido del circuito mientras se toman los datos por medio de la antena local por medio del cable FTDI.
5. Cargar el algoritmo realizado al *APM* 2.5 (montaje remoto)
6. Cargar el algoritmo realizado para la antena local (conectada vía Arduino Mega al computador). Ver Anexo 12
7. Realizar el recorrido del circuito y guardar los datos impresos en *Serial Chart*[2]. (ver observación No. 3 para mayor información sobre *Serial Chart*)
8. Usar la macro que se desarrolló (Anexo 9) para coger los datos en bruto que imprime la *IMU* y convertirlos en gráficas por medio de Excel. También se puede graficar en línea directamente con el *Serial Chart*, se puede escoger cuál de las dos opciones usar dependiendo de si se necesita hacer análisis o no de los datos impresos por la *IMU*.
9. Comparar las gráficas y calcular el error promedio relativo.

[1] *Mission Planner*: Es una herramienta empleada por el *APM* 2.5 que brinda soporte en navegación y medición de variables de vuelo.

[2] *Serial Chart*: Es el software que se usó para tomar los datos en tiempo real, desarrollado por Sergio Baluta en el proyecto *Starlino*.

Figura 14. Circuito seguido por el vehículo para la prueba.



Fuente: Autores.

En la Figura 14 se muestra un plano por zonas del circuito que siguió el vehículo terrestre. En la primera zona se probó el ángulo *Roll* (+, -). En la segunda zona se probó el ángulo de *Pitch* (-, +) y en la tercera zona se probó el ángulo *Yaw* (+).Dado que el experimento trataba de comparar *Mission Planner* con el código elaborado por los autores las pruebas tuvieron que ser divididas por zonas a causa de una limitante que tiene el software privativo *Mission Planner* para guardar los datos por más de un tiempo finito. Cabe resaltar que el código expuesto en el presente trabajo de grado no tiene esta limitante.

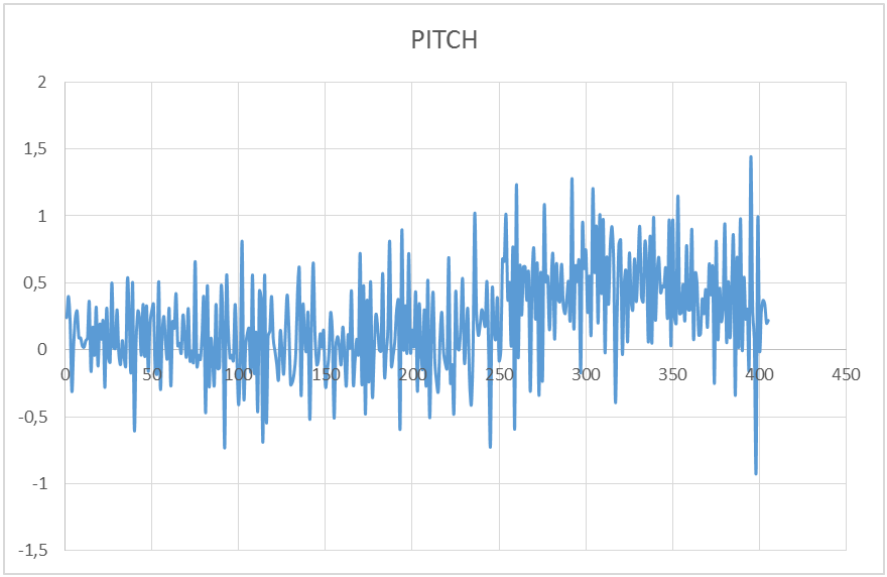
Figura 15. Vehículo usado para las pruebas. Toma Superior con protector.

Fuente: Autores.

Figura 16. Vehículo usado para las pruebas. Toma Lateral con protector.

Fuente: Autores.

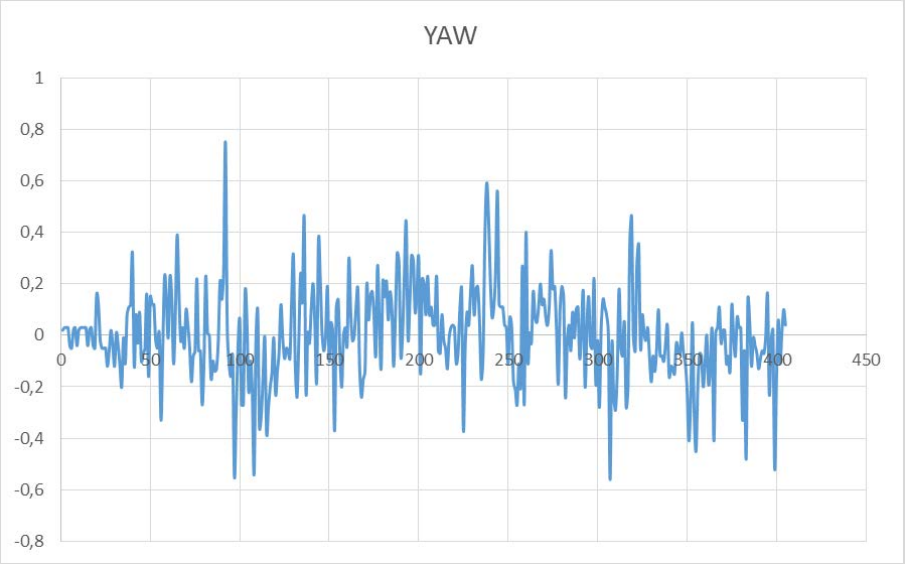
Figura 17. Ángulo de cabeceo en prueba de alabeo en circuito.



Fuente: Autores.

En el circuito seguido por el vehículo (Figura 14), el ángulo de alabeo que debería inclinarse en ambos sentidos era de +22° y -22° aproximadamente. En la Figura 17, Figura 18 y Figura 19 se muestran los resultados del circuito con el código planteado para los ángulos *Pitch*, *Yaw* y *Roll* respectivamente, y en la Figura 20 se muestran los resultados empleando la herramienta de *Mission Planner*.

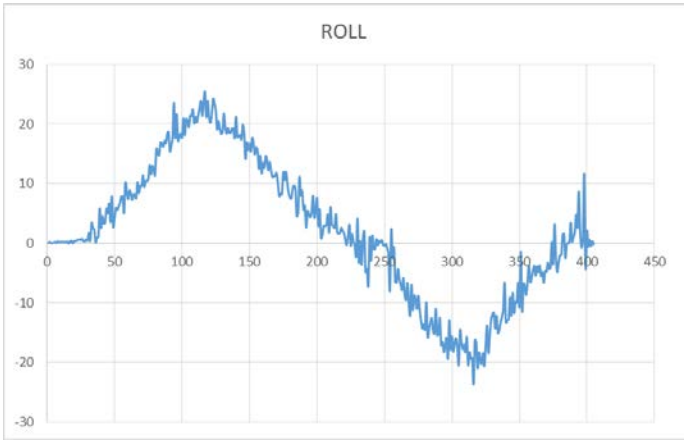
Figura 18. Ángulo de guiñada en prueba de alabeo en circuito.



Fuente: Autores.

En la Figura 17 y Figura 18 se observan variaciones de los ángulos de cabeceo y guiñada mientras se probaba el ángulo de alabeo, pero cabe resaltar que las variaciones de éstos son inferiores a 1.5° para el ángulo de cabeceo y 0.8° para el ángulo de guiñada, por ende se consideran estas variaciones como irregularidades del circuito de prueba.

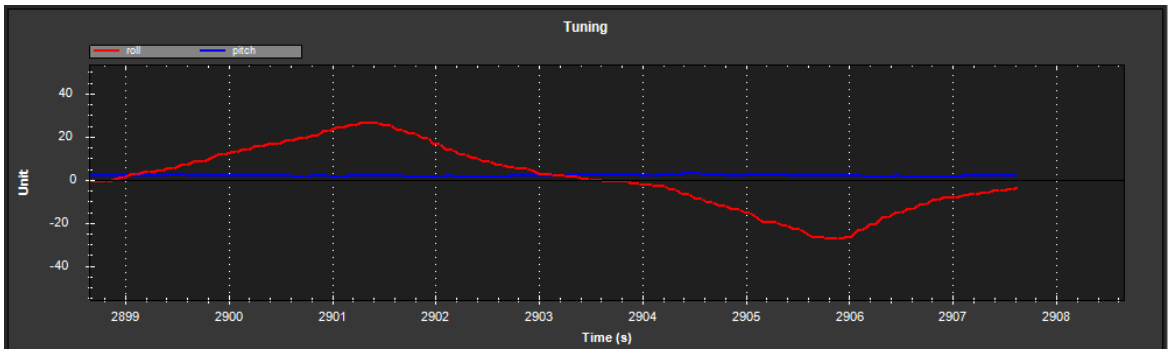
Figura 19. Prueba *Roll* en el circuito.



Fuente: Autores.

Como se observa en las Figuras 19 y 20, de la prueba realizada con el código implementado en el proyecto, se obtiene un ángulo de elevación de 21° aproximadamente, y en *Mission Planner* se obtiene una inclinación de 22°. Por lo tanto el error del código realizado es de 1° en el ángulo de alabeo que equivale al 4% de error promedio relativo en comparación con el dato del ángulo conocido y graficado por *Mission Planner* de 22°.

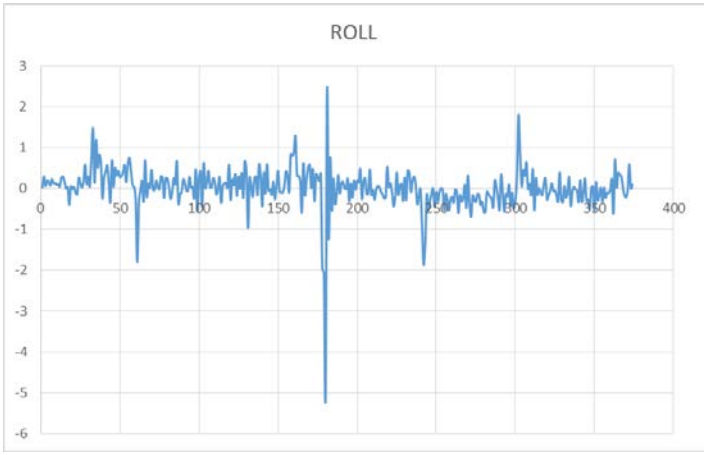
Figura 20. Prueba *Roll* en circuito con *Mission Planner*.



Fuente: Autores.

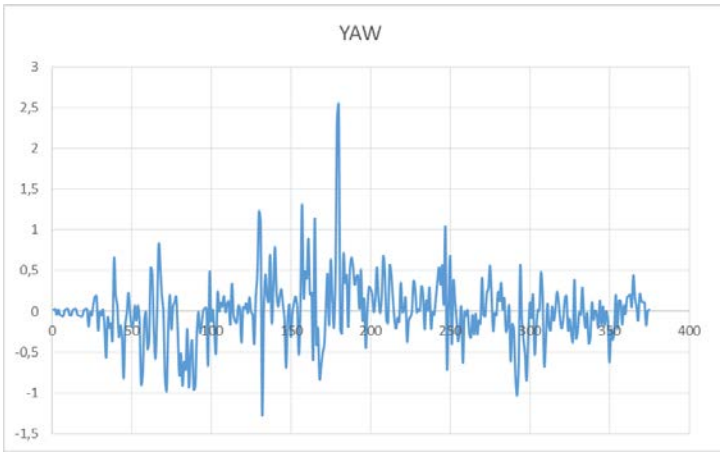
Para la prueba del ángulo de cabeceo (*Pitch*) también se registraron inclinaciones en ambos sentidos del ángulo, comparando los datos del código planteado (Figura 21, Figura 22 y Figura 23) y los datos obtenidos con el firmware de *Mission Planner* (Figura 24).

Figura 21. Ángulo de alabeo en prueba de cabeceo en circuito.



Fuente: Autores.

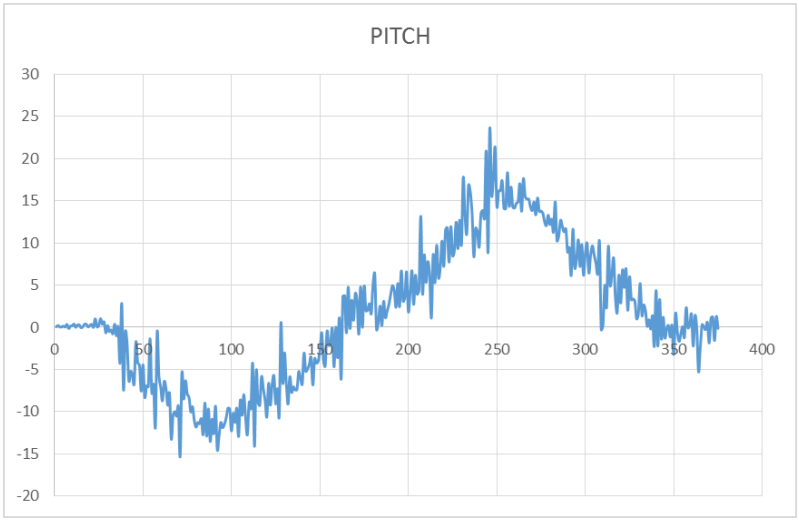
Figura 22. Ángulo de guiñada en prueba de cabeceo en circuito.



Fuente: Autores.

En la Figura 21 y Figura 22 se puede observar la variación de los ángulos de alabeo y guiñada mientras se realiza la prueba en el circuito para el ángulo de cabeceo; aunque se presentan variaciones en ambos ángulos en ambos sentidos, es de gran importancia observar que las variaciones son mínimas respecto a los cambios del ángulo de cabeceo.

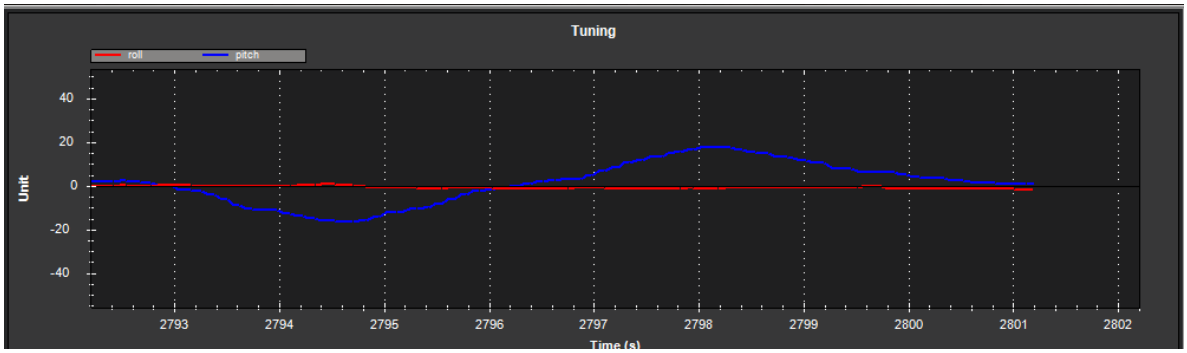
Figura 23. Prueba *Pitch* en el circuito.



Fuente: Autores.

En estas pruebas la inclinación del circuito seguido por el vehículo era aproximadamente -16° y 17°, esto se comprueba con los resultados obtenidos por *Mission Planner* en los que se obtienen los mismos datos como se muestran en la Figura 24; al comparar estos resultados con los del código implementado se puede observar que el valor pico es de -15° aproximadamente. En este caso el error es de aproximadamente 2°, lo que equivale a un error del 4% promedio relativo.

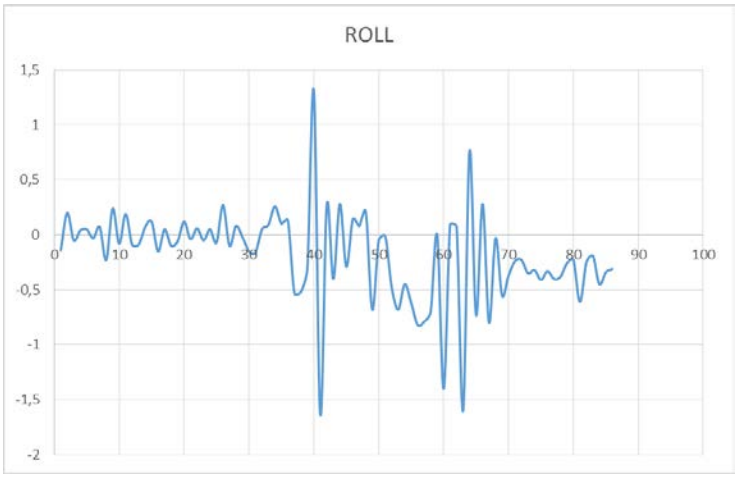
Figura 24. Prueba *Pitch* en circuito con *Mission Planner*.



Fuente: Autores.

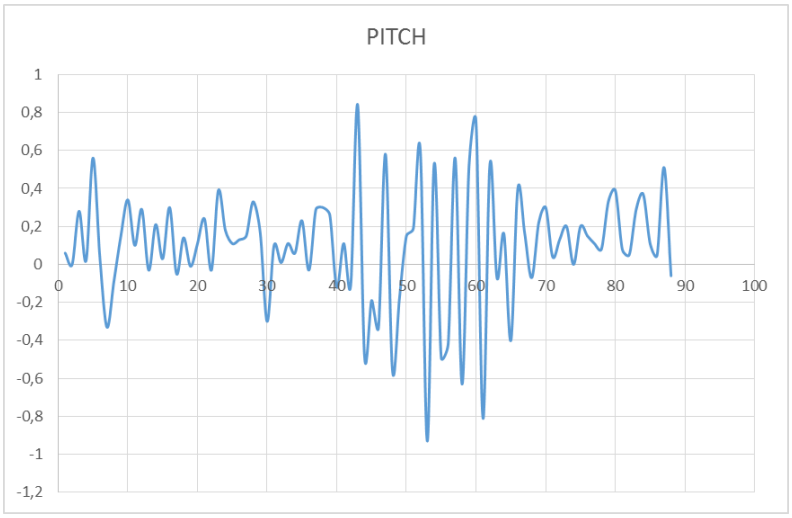
Finalmente para la prueba del ángulo de guiñada en el circuito construido, se realizó el mismo giro para la prueba de *Mission Planner* y la prueba del código realizado pero a través del eje vertical del vehículo, los resultados se grafican en la Figura 25, Figura 26 y Figura 27 para los ángulos cabeceo, alabeo y guiñada respectivamente.

Figura 25. Ángulo de alabeo en prueba de guiñada en circuito.



Fuente: Autores.

Figura 26. Ángulo de cabeceo en prueba de guiñada en circuito.

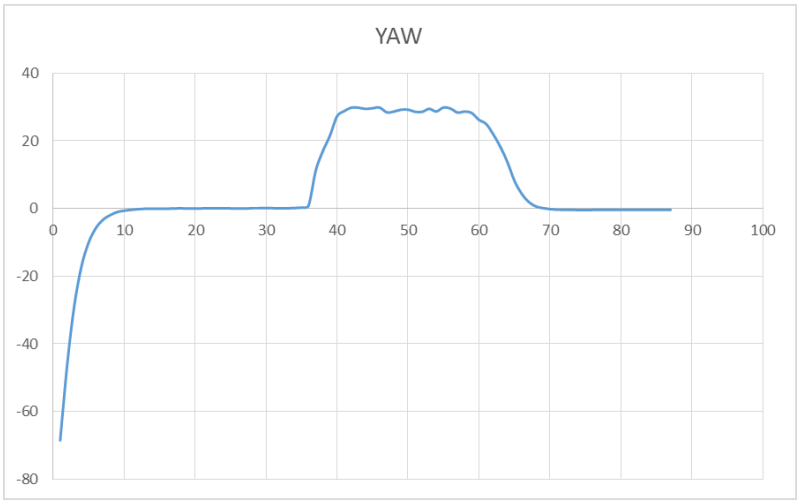


Fuente: Autores.

Al igual que en las pruebas anteriores, existe una diferencia entre el ángulo impreso por la *IMU* elaborada en el presente proyecto en comparación con el ángulo conocido que debería imprimir y el ángulo que imprime el *APM* 2.5 usando

el firmware privativo. Sin embargo estas variaciones son inferiores a 1.5° promedio relativo y como se ha mencionado anteriormente este error es relativo a los errores de ortogonalización de la DCM, errores numéricos en la computación y errores debidos a la cuantificación de las señales analógicas que son convertidas a señales digitales por los sensores, entre otros.

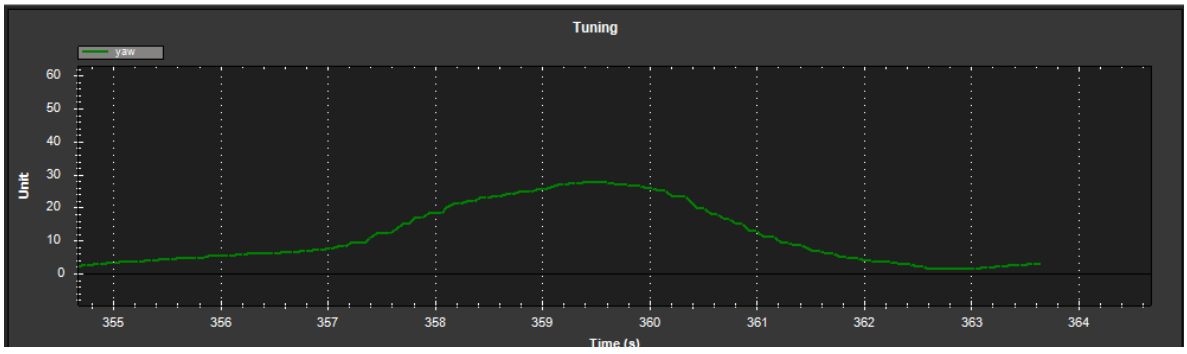
Figura 27. Prueba *Yaw* con código realizado.



Fuente: Autores.

Para esta prueba el vehículo realizó un giro (variando el ángulo de *Yaw*) de +27.5° como lo indica la Figura 28, al compararlo con los datos obtenidos por el código realizado se obtuvo 28.7°, obteniendo una diferencia relativa de 1° aproximadamente que equivale al 4% de error promedio relativo

Figura 28. Prueba *Yaw* con *Mission Planner.*



Fuente: Autores.

Al realizar las pruebas en los tres ángulos siguiendo un circuito, se observaron las siguientes particularidades de gran relevancia:

1. El error dependía de la velocidad del vehículo al momento de realizar las pruebas. Esto está relacionado a la capacidad que tiene la IMU para detectar cambios bruscos de posición. Entre más brusco sea el cambio de posición, mayor es el error promedio relativo.

2. Los sensores tienen un límite para detectar cambios, este dato se cuantifica en la hoja de datos del dispositivo. Por ejemplo, el giróscopo tiene una sensibilidad de 2000°/s.

3. Si el clima esta nublado es probable que el GPS no adquiera *fix.* En este caso es posible reemplazar el GPS por un magnetómetro. Sin embargo, este cambio solo se puede llevar a cabo cuando el vehículo es terrestre, esto es debido a que el dato que se obtiene del magnetómetro se ve influenciado por el viento, y en el caso de los vehículos terrestres, el viento no provoca variaciones perpendiculares a la velocidad del mismo.

5. PROTOCOLO DE PRUEBAS

Para realizar las pruebas con el código implementado se debe tener en cuenta una serie de consideraciones:

1. Se debe cargar el código realizado sin tener alimentado el *APM* 2.5 ya que el dispositivo también es alimentado vía puerto serial. De no hacerlo se puede producir un corto en la tarjeta.

2. La tarjeta tiene 3 modos de alimentación: por medio del puerto Serial (5[V]), por los puertos de entrada y por los puertos de salida, usando el regulador de 5[V].

3. El puerto serial que usa el *APM* 2.5 para comunicarse con el computador es el mismo que usa para comunicarse con la antena *3DR-radio*, por lo tanto no se pueden realizar ambas comunicaciones de modo simultáneo.

4. Para asegurar una buena señal de recepción en el GPS se debe trabajar en campo abierto, preferiblemente con clima despejado.

5. Para realizar las pruebas se debe conocer las variaciones en los tres ejes (lateral, longitudinal y vertical) que son provocadas por el cambio de posición del dispositivo.

6. Para instalar la *IMU* en un dispositivo no tripulado se debe considerar que la tarjeta tiene por convención (en el algoritmo) unos ejes ya pre-establecidos, en donde el norte de la tarjeta está ubicado en los puertos de salida y las variaciones en los ángulos vienen como sigue:

    i. Para las pruebas del ángulo de Alabeo (R*oll)* se gira el dispositivo sobre el eje longitudinal.
    ii. Para la prueba del ángulo de Cabeceo (*Pitch*) se gira el dispositivo sobre el eje lateral.
    iii. Para la prueba del ángulo de Guiñada (Yaw) se gira el dispositivo sobre el eje vertical.

6. CONCLUSIONES

- ✓ Errores numéricos, en conjugación con las desviaciones y el *offset* del giróscopo gradualmente en cada iteración acumulan un error en los elementos de la *DCM*. Teniendo en cuenta la cantidad de iteraciones que se realizan por segundo (hasta 10 iteraciones en promedio), en poco tiempo, el error será demasiado grande (superior al 10%). Por lo tanto no es viable construir una unidad de medición inercial sin implementar un controlador PI que ayude a corregir el error en el giróscopo por medio de información proveniente de otros sensores.

- ✓ En la Figura 10 se puede observar que al principio de la prueba, el ángulo de *Yaw* tuvo un pico en la mitad superior de la gráfica. Luego de analizar el código, se concluyó que el ángulo de *Yaw* es muy sensible a la "salud" de la señal del magnetómetro y/o GPS, debido a que éstos últimos son fuente principal de corrección para el ángulo de *Yaw*. En otras palabras, si se pierde el *fix* del GPS o el magnetómetro envía señales erróneas debidas a perturbaciones provocadas por algún campo magnético cercano (por ejemplo), entonces la corrección del Yaw seria errónea y la *IMU* imprimiría datos equívocos de dicho ángulo.

- ✓ De las ecuaciones 29 y 30, donde el denominador de los cocientes que calculan los ángulos $'\psi'$,$'\phi$' es $\cos\theta$ se puede concluir que los ángulos críticos son los que se encuentran en intervalos de $n * \pi/2$ (donde n =1, 3, 5, 7, etc.) debido a que estos ángulos provocarían una división por cero, y el dato del ángulo $'\psi'$ y $'\phi'$ tendería a infinito.

- ✓ Para la prueba en el ángulo *Roll,* guardando los últimos 22 datos tomados por el software en el puerto serial se calcula el error promedio. Para las pruebas en 30°, el error promedio es de 3.45%. Para las pruebas en 45°, el error promedio es de 2.87%. Para las pruebas en 60°, el error promedio es de 3.81%. Se concluye que el error promedio en el ángulo de R*oll* es mayor a medida que los datos llegan al ángulo crítico de 90°.

- ✓ Para la prueba en el ángulo *Pitch,* guardando los últimos 22 datos tomados por el software en el puerto serial se calcula el error promedio. Para las pruebas en 30°, el error promedio es de 3.92%. Para las pruebas en 45°, el error promedio es de 2.59%. Para las pruebas en 60°, el error promedio es de

2.43%. Se concluye que el error promedio en el ángulo de *Pitch* es mayor a medida que los datos llegan al ángulo crítico de 0°.

- ✓ Para la prueba en el ángulo *Yaw,* guardando los últimos 22 datos tomados por el software en el puerto serial se calcula el error promedio Para las pruebas en 30°, el error promedio es de 1.22%. Para las pruebas en 45°, el error promedio es de 0.49%. Para las pruebas en 60°, el error promedio es de 2.06%. Se concluye que el error promedio en el ángulo de *Yaw* es mayor a medida que los datos llegan al ángulo crítico de 90°.

- ✓ De los resultados obtenidos en las pruebas descritas en los numerales 4.2 y 4.3 se concluye que la *IMU* implementada en el presente trabajo de grado utilizando la teoría de la Matriz de Cosenos Directores descrita por Premerlani y Bizard tiene resultados aceptables para captar variaciones en la posición causadas por pequeñas perturbaciones en la *IMU* con errores de precisión no mayores al 5% promedio relativo.

- ✓ En la ecuación 11, se asume que el tiempo de muestreo debe ser lo suficientemente pequeño, para que las variaciones en los elementos de la matriz R no sean grandes. Esto permite concluir que si existen cambios en la posición del *APM* 2.5 que provoquen variaciones lo suficientemente altas, los datos relevantes a los ángulos *Roll*, *Pitch* y *Yaw* serán completamente inexactos. Vale la pena aclarar la diferencia que existe entre los “límites de operación” de la teoría propuesta por Premerlani/Bizard y los límites de operación de los sensores que posee el *APM* 2.5.

- ✓ El error obtenido en cada prueba era relativo a la velocidad del vehículo al momento de realizar las inclinaciones en los respectivos ángulos. Esto está relacionado a la capacidad que tiene la IMU para detectar cambios bruscos de posición, es decir, entre más brusco sea el cambio, mayor es el error promedio relativo.

7. OBSERVACIONES

- La corrección de la matriz DCM se realiza debido a la existencia de errores numéricos en las operaciones que se realizan en cada iteración, las cuales provocan una pérdida de ortogonalidad que debe ser rectificada.

- El tiempo de muestreo (Ts) es igual a 0.02 [s]. Este periodo de muestreo fue seleccionado teniendo en cuenta que la frecuencia máxima recomendada de trabajo para el MPU6000 es de 50 [Hz], es decir, en cada segundo, el acelerómetro, el giróscopo y el sensor de temperatura calculan 50 datos nuevos. Por lo tanto, para obtener la mayor sensibilidad posible en las mediciones, se selecciona un periodo de muestreo correspondiente a $Ts = \frac{1}{50} = 0.02\ [s]$. Es importante entender que las variaciones angulares que ocurran en intervalos de tiempo inferiores no serán detectadas, y que en cierta manera, el código que se implementó, realiza una suposición de lo que ocurrió en ese instante. Dicha suposición, también es fuente de error.

- El software que se usó para tomar los datos en tiempo real, es de código abierto, su nombre es *Serial Chart* y fue desarrollado por Sergio Baluta en el proyecto *Starlino*. Es un programa muy versátil que además de capturar los datos también los gráfica. Como es un software completamente abierto, desde el código hasta el entorno gráfico permite ser modificado. En la referencia No. 23, el autor entrega toda la información y los archivos necesarios para poder hacer uso de este software a discreción.

- En el presente proyecto existen dos formas de visualizar los ángulos calculados por la *IMU*, la primera es utilizando *Serial Chart*, la segunda es utilizando la *macro* de *Excel (*adjunta en el anexo 9*)*. En el primer caso, se puede graficar en vivo, en el segundo caso, la gráfica obtenida es posterior a la prueba. Las ventajas de usar *Serial Chart* son las de observar las señales *Roll, Pitch* y *Yaw* de forma instantánea. Las ventajas de usar la *macro* de *Excel* son básicamente la posibilidad de hacer análisis de datos, ya que ésta última, conserva los valores en un archivo, mientras que *Serial Chart* no.

BIBLIOGRAFÍA

[1] A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV: Mark Euston, Paul Coote, Robert Mahony, Jonghyuk Kim and Tarek Hamel. Disponibles en: https://gentlenav.googlecode.com/files/MahonyPapers.zip Consultados el 15 de marzo de 2013.

[2] A guide to using IMU (Accelerometer and gyroscope Devices), in embedded Applications, Sergio Baluta, 2009, Disponible en: http://www.starlino.com/imu_guide.html, Consultado el 12 de abril de 2013.

[3] APM Planner Software, Disponible en: http://ardupilot.com/downloads/?did=91. Consultado el 9 de Marzo de 2014.

[4] Arduino Cookbook, Michael Margolis, "O'Reilly Media, Inc.", Mar 4, 2011 - 662 páginas Disponible para su previsualización en: http://books.google.com.co/books?id=raHyKejOBF4C&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false. Consultado el 22 de Noviembre de 2014.

[5] Arduipilot Config Tool, Disponible en: https://code.google.com/p/ardupilot/wiki/ConfigTool. Consultado el 19 de marzo de 2014.

[6] ArduPilot MAVLink Parameters, disponible en: https://code.google.com/p/ardupilot-mega/wiki/APM_Parameters. Consultado el 22 de febrero de 2014.

[7] Ardupilot Mega 2.5; Disponible en: http://plane.ardupilot.com/wiki/common-apm25-and-26-overview/. Consultado el 2 de Octubre de 2013.

[8] Computing Euler Angles from Direction Cosines, William Premerlani, 2010, Disponible en: https://gentlenav.googlecode.com/files/EulerAngles.pdf. Consultado el 7 de diciembre de 2013.

[9] DCM Tutorial - An introduction to orientation Kinematics, Sergio Baluta, 2009, Disponible en: http://www.starlino.com/dcm_tutorial.html, Consultado el 2 de noviembre de 2013.

[10] Declinación magnética de Bucaramanga. Disponible en: http://magnetic-declination.com/. Consultado el 2 de Enero de 2014.

[11] Direction Cosine Matrix IMU: Theory, William Premerlani and Paul Bizard, 2009, disponible en: http://gentlenav.googlecode.com/files/DCMDraft2.pdf. Consultado el 2 de octubre de 2013.

[12] Diy drones - William Premerlani; Disponible en: http://diydrones.com/profile/WilliamPremerlani. Consultado el 2 de Enero de 2014.

[13] Evaluación de la solución Ardupilot mega 2.5, para la implementación de algoritmos de control en aeroplanos a escala no tripulados (UAV): Leonardo Miguel Jaimes Sánchez y Cesar Javier Sepúlveda Peña, 2013, disponible en: http://tangara.uis.edu.co/biblioweb/tesis/2013/148550.pdf. Consultado el 27 de Octubre de 2013.

[14] Evaluating the 3DR uBlox LEA-6H GPS, Disponible en: https://sites.google.com/site/wayneholder/self-driving-car---part/evaluating-the-3dr-ublox-lea-6-gps. Consultado el 13 de Febrero de 2014.

[15] How to Build Ardupilot with Arduino, Disponible en: http://dev.ardupilot.com/wiki/building-ardupilot-with-arduino-windows/. Consultado el 25 de Enero de 2014.

[16] Lea 6 Datasheet (Gps.g6 Hw 09004), 2011, Disponible en: http://es.scribd.com/doc/62261367/Lea-6-Datasheet-Gps-g6-Hw-09004. Consultado el 2 de Enero de 2014.

[17] LEA-6 / NEO-6 / MAX-6 u-blox 6 GPS Modules Hardware Integration Manual, Disponible en: http://wenku.baidu.com/view/180dc59971fe910ef12df884.html. Consultado el 2 de Enero de 2014.

[18] Nonlinear Complementary Filters on the Special Orthogonal Group: Robert Mahony, 2008. Disponibles en: https://gentlenav.googlecode.com/files/MahonyPapers.zip Consultados el 15 de marzo de 2013.

[19] Serial Terminal Basics, Disponible en: https://learn.sparkfun.com/tutorials/terminal-basics/tera-term-windows. Consultado el 10 de febrero de 2014.

[20] SerialChart; Disponible en: https://code.google.com/p/serialchart/. Consultado el 15 de Febrero de 2014.

[21] TinyGPS, Arduiniana, Disponible en: http://arduiniana.org/libraries/tinygps/. Consultado el 27 de Enero de 2014.

[22] u- blox 6 Receiver Description Including Protocol Specification. Disponible en: http://ebookbrowsee.net/u-blox6-receiverdescriptionprotocolspec-gps-g6-sw-10018-pdf-d138244651. Consultado el 2 de Enero de 2014.

[23] Using the 3DR Radio for telemetry with APM 2, 2013, Disponible en: https://code.google.com/p/ardupilot-mega/wiki/3DRadio. Consultado el 17 de febrero de 2014.

[24] Visual Basic 2010, disponible en: http://www.vbtutor.net/vb2010/. Consultado el 2 de febrero de 2014.

[25] Writing and Reading Serial Data in Matlab, Disponible en: http://www.mathworks.com/help/matlab/matlab_external/writing-and-reading-data.html. Consultado el 5 de febrero de 2014.

ANEXOS

Anexo A. Algoritmo de construcción de la DCM.

```
/****************************** 21/04/2014 ******************************/
/*-----------------------------------------------------------------------|
PROGRAMA: Datos que imprime la IMU
Descripción: En este código se ejecutan las secuencias principales, se obtiene la DCM y se calculan los
ángulos de Euler.

Hecho por: Gabriela Bermudez y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
CEMOS.
|***********************************************************************/

/*Declaración de variables*/

double Ts=0.02;                    // Tiempo mínimo de muestreo.
int bandera=1;                     // Bandera matriz Inicial (1 vez)
const int longitud=3;              // Tamaño de los vectores.
double Ibi[3], Kbi[3], Jbi[3];                  // Vectores I,K,J.
double Kbni[3], Jbni[3], Ibni[3], Ini[3], Inp[3];   // Vectores I,J,K normalizados.
double R[3][3],RFa[3][3];                       // Matriz R y Matriz R refrescada.
double Ru[3],Rc[3],error;                       // Error y variables auxiliares
double M[3],A[3],W[3];                          // Vector Magnetómetro, Acelerómetro y Giróscopo
double cry[3], crx[3];                          // Variables auxiliares, ortogolazicación de la DCM
double xo[3],yo[3],zo[3];                       // Vectores ortogonalizados
double xnp, ynp, znp;                           // Vectores auxiliares para re-normalización
double xn[3], yn[3], zn[3];                     // Vectores re-normalizados, que componen la DCM
double cog, heady;                              // COG= Course Over Ground | Heading = COG
double COGX, COGY, YCG;                         // Componentes en X y Y del COG. CorreccionYaw.
double Rd[3],YCP[3];                            // Yawcorrectionplane y variables auxiliares
double V[3],gr[3], Ac[3];                  // Course Velocity GPS(UBX) | Gref Acent
double RPCP[3];                                 // rollpitchcorrectionplane
float velocity;                          // Velocity = SOG = Speed Over Ground

// MPU6000 HMC5883
int datoViejo=0;
byte calibracion=0;
double gyro[3], acel[3], magnet[3], tempt;
double offset[9];
```

```
// Variables del PI
double WRP=2, WY=3;                          // Pesos corrección de Roll-Pitch y Yaw
double KP=13.9, KI=0.5;                      // Constante proporcional e integral
double PT1=-2,PT2=0.85,PT3=2;                // Peso para cada ángulo
double WPC[3], WIC[3], WC[3], WI[3], WIA[3]; // Varibles auxiliares del controlador PI
double TC1[3], TC2[3], TCA[3], TCI[3],TC[3];     // Varibles auxiliares del controlador PI
float yaw,roll,pitch;                        // Angulos de Euler

#include "MPU6000.h";
#include "HMC5883.h";
#include "ublox6.h";
#include <Wire.h>

void setup(){

 Serial.begin(57600);
 pinMode(40, OUTPUT);                        // Se detiene el barómetro, porque ocupa el bus SPI

 digitalWrite(40, HIGH);
 pinMode(31, INPUT);                         // Se establece pin de lectura nueva magnetómetro

 calibracion=0;
 MPU6000_newdata=0;
 MPU6000_Init();
 HMC5883_init();

 MPU6000_calibrar();                         // Se calibran acelerómetro y giroscopo
 HMC5883_calibrar();                         // Se calibra magnetómetro
 GPS_init();                                 // Se Inicializa el GPS
}

void loop(){
 MPU6000_ReadWithOffset();                   // Se obtienen datos nuevos (Giro.,Acc.)
 HMC5883_readCalibrated();                   // Se obtienen datos nuevos (Mag.

 /* Se Inicializan los Vectores W,A,M */
 W[0]=gyro[0]+WC[0];
 W[1]=gyro[1]+WC[1];
 W[2]=gyro[2]+WC[2];

 A[0]=acel[0];
 A[1]=acel[1];
 A[2]=acel[2];
```

```
 M[0]=magnet[0];
 M[1]=magnet[1];
 M[2]=magnet[2];

/* Lectura del GPS */
 GPS_loop();

/*****************************************************************/
/** Esto ocurre solo 1 vez cada vez que se corre el código **/
/** Se calcula Ibi y Kbi **/
/*****************************************************************/
if(bandera==1)
{
 normalizarVector(M,Ibi,longitud);            // Ibi=M/|M|
 normalizarVector(A,Kbi,longitud);            // Kbi=-A/|A|
 productoPorEscalar(-1,Kbi,Kbi,longitud);     // Kbi=-1*Kbi

/** Normalización (Ibni, Kbni, Jbni) **/
 normalizarVector(Kbi,Kbni,longitud);         // Kbni=kbi/|kbi|
 productoVectorial(Kbi,Ibi,Inp);              // Inp=(Kbi)x(Ibi)
 productoVectorial(Inp,Kbi,Ini);              // Ini=(Inp)x(Kbi)
 normalizarVector(Ini,Ibni,longitud);         // Ibni=Ini/|Ini|
 productoVectorial(Kbni,Ibni,Jbni);           // I,J,K Normalizados

// DCM Inicial (Matriz R)
 R[0][0]=Ibni[0];
 R[0][1]=Ibni[1];
 R[0][2]=Ibni[2];
 R[1][0]=Jbni[0];
 R[1][1]=Jbni[1];
 R[1][2]=Jbni[2];
 R[2][0]=Kbni[0];
 R[2][1]=Kbni[1];
 R[2][2]=Kbni[2];

 bandera=0;
}
/*****************************************************************/
/*****************************************************************/

/* Calculo DCM Actualizada (Matriz RFa) */
RFa[0][0]=1;
RFa[0][1]=-Ts*W[2];
RFa[0][2]=W[1]*Ts;
```

```
RFa[1][0]=W[2]*Ts;
RFa[1][1]=1;
RFa[1][2]=-W[0]*Ts;
RFa[2][0]=-W[1]*Ts;
RFa[2][1]=W[0]*Ts;
RFa[2][2]=1;
ProductoMatricial(R,RFa,R);

// Calculo del Error
Rc[0]=R[0][0];                          // Vector X
Rc[1]=R[0][1];
Rc[2]=R[0][2];
Ru[0]=R[1][0];                          // Vector Y
Ru[1]=R[1][1];
Ru[2]=R[1][2];
error=(Rc[0]*Ru[0]+Rc[1]*Ru[1]+Rc[2]*Ru[2]);
productoPorEscalar(error/2,Ru,cry,longitud);   // Variables auxiliares: cry, crx
productoPorEscalar(error/2,Rc,crx,longitud);

/* Calculo de lo valores necesarios para renormalizar R */
restarVectores(Rc,cry,xo,longitud);             // Variables auxiliares
restarVectores(Ru,crx,yo,longitud);             // Variables auxiliares
productoVectorial(xo,yo,zo);                   //Zo=(Xo)x(Yo)

xnp=(xo[0]*xo[0])+(xo[1]*xo[1])+(xo[2]*xo[2]);
xnp=0.5*(3-xnp);
productoPorEscalar(xnp,xo,xn,longitud);

ynp=(yo[0]*yo[0])+(yo[1]*yo[1])+(yo[2]*yo[2]);
ynp=0.5*(3-ynp);
productoPorEscalar(ynp,yo,yn,longitud);

znp=(zo[0]*zo[0])+(zo[1]*zo[1])+(zo[2]*zo[2]);
znp=0.5*(3-znp);
productoPorEscalar(znp,zo,zn,longitud);

/* Se construye la DCM Re-Normalizada */
R[0][0]=xn[0];
R[0][1]=xn[1];
R[0][2]=xn[2];
R[1][0]=yn[0];
R[1][1]=yn[1];
R[1][2]=yn[2];
R[2][0]=zn[0];
```

```
R[2][1]=zn[1];
R[2][2]=zn[2];

/* Cálculo de Corrección Yaw */
cog=(calcularInclinacion(0));                    //Se puede usar el heady (GPS) o el heading (MAG)
COGX=cos(cog);
COGY=sin(cog);

YCG=(R[0][0]*COGY)-(R[1][0]*COGX);
Rd[0]=R[2][0];
Rd[1]=R[2][1];
Rd[2]=R[2][2];
productoPorEscalar(YCG, Rd, YCP, longitud);


/* Speed Over Ground SOG = Velocity */
V[0]=velocity;
V[1]=0;
V[2]=0;

/* Cálculo de Corrección Roll-Pitch */
productoVectorial(gyro,V,Ac);
int i;
for(i=0;i<longitud;i++)
{
  gr[i]=A[i]+Ac[i];
}
productoVectorial(Rd,gr,RPCP);
//********************** Controlador PI **********************//
/* Se declaran los TotalCorrectionanteriores para no declararlo como matriz */

TCA[0]=TC[0];
TCA[1]=TC[1];
TCA[2]=TC[2];

productoPorEscalar(WRP, RPCP, TC1, longitud);   // TC1=WRP*RPCP
productoPorEscalar(WY, YCP, TC2, longitud);    // TC2=WY*YCP

for(i=0;i<longitud;i++)
{
  TC[i]=TC1[i]+TC2[i];
}

/* Se declaran los WI correctios anteriores para no declararlo
```

```
como matriz se usan en el WIC al sumar el anterior */

WIA[0]=WIC[0];
WIA[1]=WIC[1];
WIA[2]=WIC[2];

/*Correción proporcional */
productoPorEscalar(KP, TC, WPC, longitud);

/*Correccion integral */
TCI[0]=Ts*(TCA[0]+TC[0])*0.5;
TCI[1]=Ts*(TCA[1]+TC[1])*0.5;
TCI[2]=Ts*(TCA[2]+TC[2])*0.5;

productoPorEscalar(KI, TCI, WI, longitud);
WIC[0]=WIA[0]+WI[0];
WIC[1]=WIA[1]+WI[1];
WIC[2]=WIA[2]+WI[2];

/*Correcion total  */
WC[0]=WPC[0]+WIC[0];
WC[1]=WPC[1]+WIC[1];
WC[2]=WPC[2]+WIC[2];

pitch=PT1*asin(R[2][0]);          //Thetha
yaw=PT2*asin(R[1][0]/cos(pitch));   //Ye
roll=PT3*asin(R[2][1]/cos(pitch));  //Phi

/** Angulos Euler, resultado de la IMU **/
Serial.print(pitch*180/PI);
Serial.print(",");
Serial.print((yaw*180/PI));
Serial.print(",");
Serial.println(roll*180/PI);
delay(10);
}
```

Anexo B. Librería de operaciones.

```
/****************************** 21/04/2014 ******************************/
/*-------------------------------------------------------------------------|
PROGRAMA: Operaciones Básicas con matrices
```

```
Descripción: Conglomerado con las Operaciones

Hecho por: Gabriela Bermudez y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
CEMOS.
|*************************************************************************/

// Resta de Vectores
double restarVectores(double restando1[], double restando2[], double resta[], int longitud)
{
int i;
for(i=0;i<longitud;i++)
{
resta[i]=restando1[i]-restando2[i];
}
}

//Magnitud de un Vector
double magnitudVector(double vector[], int longitud)
{
int i;
double magnitud=0;
for(i=0;i<longitud;i++)
{
magnitud=magnitud+(vector[i])*(vector[i]);
}
magnitud=sqrt(magnitud);
return magnitud;
}

//Producto Escalar de un vector
double productoPorEscalar(double escalar, double vector[], double resultado[], int longitud)
{
int i;

for(i=0;i<longitud;i++)
{
resultado[i]=escalar*(vector[i]);
}
}

//Normalizando un  Vector
double normalizarVector(double vector[], double resultado[], int longitud)
{
```

```
double magnitud;
magnitud=magnitudVector(vector, longitud);
magnitud=1/magnitud;

productoPorEscalar(magnitud, vector, resultado, longitud);
}

//Producto Vectorial de un Vector
double productoVectorial(double vector1[], double vector2[],double resultado[])
{
//elemento I
resultado[0]=(vector1[1]*vector2[2])-(vector1[2]*vector2[1]);
//elemento J
resultado[1]=(vector1[2]*vector2[0])-(vector1[0]*vector2[2]);
//elemento K
resultado[2]=(vector1[0]*vector2[1])-(vector1[1]*vector2[0]);
}

//Producto Matricial de un Vector
double ProductoMatricial(double a[3][3], double b[3][3], double resultado[3][3])
{
        resultado[0][0]=(a[0][0]*b[0][0])+(a[0][1]*b[1][0])+(a[0][2]*b[2][0]);
        resultado[0][1]=(a[0][0]*b[0][1])+(a[0][1]*b[1][1])+(a[0][2]*b[2][1]);
        resultado[0][2]=(a[0][0]*b[0][2])+(a[0][1]*b[1][2])+(a[0][2]*b[2][2]);
        resultado[1][0]=(a[1][0]*b[0][0])+(a[1][1]*b[1][0])+(a[1][2]*b[2][0]);
        resultado[1][1]=(a[1][0]*b[0][1])+(a[1][1]*b[1][1])+(a[1][2]*b[2][1]);
        resultado[1][2]=(a[1][0]*b[0][2])+(a[1][1]*b[1][2])+(a[1][2]*b[2][2]);
        resultado[2][0]=(a[2][0]*b[0][0])+(a[2][1]*b[1][0])+(a[2][2]*b[2][0]);
        resultado[2][1]=(a[2][0]*b[0][1])+(a[2][1]*b[1][1])+(a[2][2]*b[2][1]);
        resultado[2][2]=(a[2][0]*b[0][2])+(a[2][1]*b[1][2])+(a[2][2]*b[2][2]);
}

//Matriz Transpuesta
void MatrixTranspuesta(double A[3][3], double C[3][3])
{
C[0][0]=A[0][0];
C[1][0]=A[0][1];
C[2][0]=A[0][2];
C[0][1]=A[1][0];
C[1][1]=A[1][1];
C[2][1]=A[2][1];
C[0][2]=A[2][0];
C[1][2]=A[2][1];
C[2][2]=A[2][2];
```

```
}
```

Anexo C. GPS.h

```
/****************************** 21/04/2014 ******************************/
/*-------------------------------------------------------------------------|
PROGRAMA: GPS
Descripción: Archivo secundario

Modificado por: Gabriela Bermudez y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
CEMOS.
|*************************************************************************/

#include <PString.h>

#define MAX_LENGTH 512

#define  POSLLH_MSG  0x02
#define  SBAS_MSG    0x32
#define  VELNED_MSG  0x12
#define  STATUS_MSG  0x03
#define  SOL_MSG     0x06
#define  DOP_MSG     0x04
#define  DGPS_MSG    0x31

#define LONG(X)    *(long*)(&data[X])
#define ULONG(X)   *(unsigned long*)(&data[X])
#define INT(X)     *(int*)(&data[X])
#define UINT(X)    *(unsigned int*)(&data[X])

unsigned char data[MAX_LENGTH];

long lastTime = 0;
```

Anexo D. Librería GPS.

```
/****************************** 21/04/2014 ******************************/
/*-------------------------------------------------------------------------|
PROGRAMA: GPS
Descripción: Código principal para obtener COG y SOG
```

```
Modificado por: Gabriela Bermudez y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
CEMOS.
|*************************************************************************/

#include <PString.h>

unsigned char  state, lstate, code, id, chk1, chk2, ck1, ck2;
unsigned int  length, idx, cnt;

void enableMsg (unsigned char id, boolean enable) {
 //            MSG   NAV   < longitud >  NAV
 byte cmdBuf[] = {0x06, 0x01, 0x03, 0x00, 0x01, id, enable ? 1 : 0};
 sendCmd(sizeof(cmdBuf), cmdBuf);
}

void GPS_init() {
 Serial1.begin(38400);

 // Se habilita la información que interesa (COG, SOG -> VELNED)
 enableMsg(POSLLH_MSG, false);
 enableMsg(SBAS_MSG, false);
 enableMsg(VELNED_MSG, true);
 enableMsg(STATUS_MSG, false);
 enableMsg(SOL_MSG, false);
 enableMsg(DOP_MSG, false);
 enableMsg(DGPS_MSG, false);
}

void GPS_loop() {
 if (Serial1.available()) {
   unsigned char cc = Serial1.read();

 //Asegurando las lecturas//
   switch (state) {
     case 0:
       ck1 = ck2 = 0;
       if (cc == 0xB5)
         state++;
       break;
     case 1:
       if (cc == 0x62)
         state++;
```

```
  else
    state = 0;
  break;
case 2:
  code = cc;
  ck1 += cc;
  ck2 += ck1;
  state++;
  break;
case 3:
  id = cc;
  ck1 += cc;
  ck2 += ck1;
  state++;
  break;
case 4:
  length = cc;
  ck1 += cc;
  ck2 += ck1;
  state++;
  break;
case 5:
  length |= (unsigned int) cc << 8;
  ck1 += cc;
  ck2 += ck1;
  idx = 0;
  state++;
  if (length > MAX_LENGTH)
    state= 0;
  break;
case 6:
  data[idx++] = cc;
  ck1 += cc;
  ck2 += ck1;
  if (idx >= length) {
    state++;
  }
  break;
case 7:
  chk1 = cc;
  state++;
  break;
case 8:
  chk2 = cc;
```

```
      boolean checkOk = ck1 == chk1  &&  ck2 == chk2;

//Decodificando y asignado la lectura: Velocity = SOG | Heady = COG
      if (checkOk) {
         velocity=ULONG(20);
         heady=(LONG(24) / 100000);
        switch (code) {
          case 0x01:
           if (lastTime != ULONG(0)) {
            lastTime = ULONG(0);
           }
           break;
        }
      }
      state = 0;
      break;
   }
  }
}

void sendCmd (unsigned char len, byte data[]) {
  Serial1.write(0xB5);
  Serial1.write(0x62);
  unsigned char chk1 = 0, chk2 = 0;
  for (unsigned char ii = 0; ii < len; ii++) {
    unsigned char cc = data[ii];
    Serial1.write(cc);
    chk1 += cc;
    chk2 += chk1;
  }
  Serial1.write(chk1);
  Serial1.write(chk2);
}
```

Anexo E. MPU6000.h

```
/******************************** 21/04/2014 ********************************/
/*--------------------------------------------------------------------------|
PROGRAMA: Acelerómetro y Giroscopo
Descripción: Archivo secundario

Modificado por: Gabriela Bermudez y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
```

```
CEMOS.
|*************************************************************************/
#include <SPI.h>

#define MPU6000_CHIP_SELECT_PIN 53

// Registros
#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
#define MPUREG_PRODUCT_ID 0x0C

// Revisiones
#define MPU6000ES_REV_C4          0x14    // 0001          0100
#define MPU6000ES_REV_C5          0x15    // 0001          0101
#define MPU6000ES_REV_D6          0x16    // 0001          0110
#define MPU6000ES_REV_D7          0x17    // 0001          0111
#define MPU6000ES_REV_D8          0x18    // 0001          1000
#define MPU6000_REV_C4            0x54    // 0101          0100
#define MPU6000_REV_C5            0x55    // 0101          0101
#define MPU6000_REV_D6            0x56    // 0101          0110
#define MPU6000_REV_D7            0x57    // 0101          0111
#define MPU6000_REV_D8            0x58    // 0101          1000
```

```
#define MPU6000_REV_D9                  0x59    // 0101                         1001


// Configuracion de bits  MPU 6000
#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

// Variables Globales
volatile uint8_t MPU6000_newdata;

// Variables para cada sensor
int accelX;
int accelY;
int accelZ;

int gyroX;
int gyroY;
int gyroZ;

int temp;
```

Anexo F. Librería MPU6000.

```
/******************************** 21/04/2014 ********************************/
/*-----------------------------------------------------------------------|
PROGRAMA: Acelerómetro y Giróscopo
Descripción: Código principal para obtener datos del Acelerómetro y Giróscopo

Modificado por: Gabriela Bermudez y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
CEMOS.
|***********************************************************************/

#include <SPI.h>

byte MPU6000_SPI_read(byte reg)
{
 byte dump;
 byte return_value;
 byte addr = reg | 0x80;                    // Selecciona el bit mas significativo
 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);
}

void MPU6000_data_int()
{
 MPU6000_newdata++;
}

// MPU6000 inicializacion, configuracion.
void MPU6000_Init(void)
{
 byte idProducto;
   pinMode(MPU6000_CHIP_SELECT_PIN, OUTPUT);
   digitalWrite(MPU6000_CHIP_SELECT_PIN, HIGH);
```

```
  // SPI
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV16);      // SPI: 1Mhz  (Reloj: 16Mhz)
  delay(10);

  // Reset
  MPU6000_SPI_write(MPUREG_PWR_MGMT_1, BIT_H_RESET);
  delay(100);
  MPU6000_SPI_write(MPUREG_PWR_MGMT_1, MPU_CLK_SEL_PLLGYROZ);
  delay(1);
  // I2C bus desabilitado
  MPU6000_SPI_write(MPUREG_USER_CTRL, BIT_I2C_IF_DIS);
  delay(1);
  MPU6000_SPI_write(MPUREG_SMPLRT_DIV,19);     // Taza de ejemplo = 50Hz |1Khz/(19+1) = 50Hz
  delay(1);
  MPU6000_SPI_write(MPUREG_CONFIG, BITS_DLPF_CFG_20HZ);
  delay(1);
  MPU6000_SPI_write(MPUREG_GYRO_CONFIG,BITS_FS_2000DPS);  // Escala del giro 2000º/s
  delay(1);
  MPU6000_SPI_write(MPUREG_ACCEL_CONFIG,0x08);          // Escada del accel. 4g (4096LSB/g)
  delay(1);
  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);
  attachInterrupt(6,MPU6000_data_int,RISING);
}

// Giro, accel. MPU6000
void MPU6000_Read()
{
 int byte_H;
 int byte_L;

 // 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;

  // 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;
```

```
  // 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;

  // 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;

  // 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;
  // 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;
  // 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;
}

void MPU6000_calibrar(){
 // Función para determinar el offset de los sensores
 int oldData=0, conteo=0;
 signed int offsetGX=0, offsetGY=0, offsetGZ=0, offsetAX=0, offsetAY=0, offsetAZ=0;
 // Espera un tiempo prudencial para la estabilidad de los sensores, el mismo tiempo que durará la
calibración
 delay(100);
 while(conteo<5){
  // 5 lecturas durante 50 ms para obtener un promedio del valor leído
  while(oldData==MPU6000_newdata){
  }
  oldData=MPU6000_newdata;
  MPU6000_Read();
  //Valores del offset
  offsetGX+=gyroX;
  offsetGY+=gyroY;
  offsetGZ+=gyroZ;
  offsetAX+=accelX;
  offsetAY+=accelY;
```

```
   offsetAZ+=accelZ;
   //Siguiente lectura
   delay(10);
   conteo++;
  }
  //Promedio de los datos leídos
  offset[0]=offsetAX/5;
  offset[1]=offsetAY/5;
  offset[2]=offsetAZ/5-8192;          //este debe marcar, en reposo, 1.0 g
  offset[3]=offsetGX/5;
  offset[4]=offsetGY/5;
  offset[5]=offsetGZ/5;
}



void MPU6000_ReadWithOffset(){
  if(datoViejo!=MPU6000_newdata){

    datoViejo=MPU6000_newdata;
    MPU6000_Read();

    //el valor dividido depende de la configuración seleccionada
    acel[0]=(accelX-offset[0])/8192.0;
    acel[1]=(accelY-offset[1])/8192.0;
    acel[2]=(accelZ-offset[2])/8192.0;
    gyro[0]=(gyroX-offset[3])/16.4;
    gyro[1]=(gyroY-offset[4])/16.4;
    gyro[2]=(gyroZ-offset[5])/16.4;
    tempt=temp/((double)340.0)+36.53;
  }
}
```

Anexo G. HMC5883.h

```
/******************************** 21/04/2014 ********************************/
/*-----------------------------------------------------------------------------|
PROGRAMA: Magnetómetro
Descripción: Inclinacion de Bucaramanga

Modificado por: Gabriela Bermudez y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
CEMOS.
```

```
|*************************************************************************/

// configuración del sensor HMC5883 para el Ardupilot Mega 2.5

#define inclinacionBucaramanga 0.133

byte c;

byte magnetTempCalibrado=0;

double magnetTempCal[3]={1.0, 1.0, 1.0}, magnetAjusteInterno[3]={0.0, 0.0, 0.0};
```

Anexo H. Librería HMC5883

```
/******************************** 21/04/2014 ********************************/
/*-------------------------------------------------------------------------|
PROGRAMA: Magnetómetro
Descripción: Código principal para obtener datos del magnetómetro

Hecho por: Leonardo Miguel Jaimes y Cesar Javier Sepulveda
Modificado por: Gabriela Bermúdez M. y Carlos A. Niño
UNIVERSIDAD INDUSTRIAL DE SANTANDER
CEMOS.
|*************************************************************************/

// Configuración del sensor HMC5883 para el Ardupilot Mega 2.5

#define escala 2.56

void HMC5883_init(){

 Wire.begin();
 Wire.beginTransmission(0x1E);    //dirección del dispositivo
 Wire.write(0x00);            //registro A
 Wire.write(0x78);            //8 muestras en promedio; 75 hertz;
 Wire.write(0x01);            //registro B
 Wire.write(0xA0);            //4.7 gauss
 Wire.write(0x02);            //registro mode
 Wire.write(0x00);            //modo continuo, i2c en low speed

 c=Wire.endTransmission();
}
```

```
void HMC5883_readUnscaled(){
  byte c, datos[6], i, j;
  double dato[3];

  magnet[0]=-1000000;
  magnet[1]=-1000000;
  magnet[2]=-1000000;

  dato[0]=-1000000;
  dato[1]=-1000000;
  dato[2]=-1000000;

  if(digitalRead(31)==HIGH){          //Pin de lectura magnetómetro
    Wire.beginTransmission(0x1E);
    Wire.write(0x03);                 //Primer registro de datos
    c=Wire.endTransmission();

    Wire.beginTransmission(0x1E);
    c=Wire.requestFrom(0x1E,6);
/* Vector de datos desde el magnetometro */
    if(Wire.available() == 6){
      for(i=0;i<6;i++){
        datos[i]=Wire.read();
      }
    }
    c=Wire.endTransmission();
/* Ver datos en un solo byte */
    j=0;
    for(i=0;i<3;i++){
      dato[i]=double((datos[j]<<8) | datos[j+1]);
      j+=2;
    }
    magnet[0]=dato[0];
    magnet[1]=dato[2];
    magnet[2]=dato[1];
  }
}

void HMC5883_read(){
  byte i;

  HMC5883_readUnscaled();

  magnet[0]=magnet[0]*escala;
```

```
 magnet[1]=magnet[1]*escala;
 magnet[2]=magnet[2]*escala;
}

void HMC5883_readCalibrated(){
 byte i;

 HMC5883_readUnscaled();

 magnet[0]=magnet[0]*escala*offset[6]*magnetTempCal[0];
 magnet[1]=magnet[1]*escala*offset[7]*magnetTempCal[1];
 magnet[2]=magnet[2]*escala*offset[8]*magnetTempCal[2];
}

float calcularInclinacion(byte compensacion){
 float inclinacion;

/* el valor de la variable compensancion es para en un futuro lograr compensar si la tarjeta no se encuentra
paralela al plano XY */
 inclinacion=atan2(magnet[0], -magnet[1])+inclinacionBucaramanga;
 inclinacion+=2.0*M_PI;
 if(inclinacion<0){
   inclinacion+=(2.0*M_PI);
 }else if(inclinacion>2.0*M_PI){
   inclinacion-=(2.0*M_PI);
 }
 inclinacion=inclinacion*(180.0/(M_PI));
 return inclinacion;
}

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

void calibrarTemperatura(){

 //los datos de 'esperado' son basados en mediciones experimentales
 byte i, conta=0, contador=0;
 double cal[3],calibrador[3],esperado[3]={1580.0, 1500.0, 1500.0};

 //configuro el dispositivo
 Wire.begin();
 Wire.beginTransmission(0x1E);   //dirección del dispositivo
 Wire.write(0x00);             //registro A
 Wire.write(0x79);             //8 muestras en promedio; 75 hertz; self_test positivo;
```

```
Wire.write(0x01);            //registro B
Wire.write(0xA0);            //4.7 gauss, 390 LSb/gauss
Wire.write(0x02);            //registro mode
Wire.write(0x00);            //modo continuo, i2c en low speed
i=Wire.endTransmission();

//Se espera que el dispositivo se establezca
delay(50);

//lee valores, un máximo de 30 intentos, 5 valores únicamente.
while(conta<30){
 HMC5883_readUnscaled();
 for(i=0;i<3;i++){
  cal[i]=double(esperado[i])/magnet[i];
  cal[i]=abs(cal[i]);
 }
 if(cal[0]>0.7 && cal[0]<1.3 && cal[1]>0.7 && cal[1]<1.3 && cal[2]>0.7 && cal[2]<1.3){
  for(i=0;i<3;i++){
   calibrador[i]+=cal[i];
  }
  contador++;
 }
 //incrementam el contador de ciclos infinitos
 conta++;
 //espera un tiempo prudencial para el siguiente dato, máximo 3 segundos calibrando
 delay(100);
}

if(magnetTempCalibrado==0){
 //lee valores para actualizar el registro temporal de valores leídos, deja la calibración intacta.
 //pregunta si pudo calibrar, o si no.
 if(contador>=5){
  for(i=0;i<3;i++){
   //promedio de la calibracion
   magnetAjusteInterno[i]=calibrador[i]/contador;
  }
  magnetTempCalibrado=1;
 }
}else{
 //actualiza los valores leidos, recalcula la calibracion
 if(contador>=5){
  for(i=0;i<3;i++){
   //promedio de la calibracion, guarda el ajuste
   calibrador[i]=calibrador[i]/contador;
```

```
      magnetTempCal[i]=magnetAjusteInterno[i]/calibrador[i];
      magnetAjusteInterno[i]=calibrador[i];
     }
    }
   }
  //re inicializa el dispositivo
  HMC5883_init();
}

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

//los datos de 'esperado' son basados en mediciones experimentales
void HMC5883_calibrar(){
  byte i, conta=0, contador=0;
  double cal[3],esperado[3]={1580, 1500, 1500};

  //Se calibra temperatura
  if(magnetTempCalibrado==0){
    calibrarTemperatura();
  }
  //configuro el dispositivo
  Wire.begin();
  Wire.beginTransmission(0x1E);    //dirección del dispositivo
  Wire.write(0x00);              //registro A
  Wire.write(0x79);              //8 muestras en promedio; 75 hertz; self_test positivo;
  Wire.write(0x01);              //registro B
  Wire.write(0xA0);              //4.7 gauss, 390 LSb/gauss
  Wire.write(0x02);              //registro mode
  Wire.write(0x00);              //modo single, i2c en low speed
  i=Wire.endTransmission();

  //establece a cero el vector de calibración
  for(i=6;i<9;i++){
    offset[i]=0;
  }
  //espera que el dispositivo se establezca
  delay(50);

  //lee el vector de valores, un máximo de 30 intentos, 5 valores únicamente
  while(conta<30){
    HMC5883_readUnscaled();
    for(i=0;i<3;i++){
      cal[i]=double(esperado[i])/magnet[i];
```

```
    cal[i]=abs(cal[i]);
   }
   if(cal[0]>0.7 && cal[0]<1.3 && cal[1]>0.7 && cal[1]<1.3 && cal[2]>0.7 && cal[2]<1.3){
    for(i=6;i<9;i++){
     offset[i]+=cal[i-6];
    }
    contador++;
   }
   //incrementa el contador de ciclos infinitos
   conta++;
   //espera un tiempo prudencial para el siguiente dato, máximo pasaran 3 segundos calibrando
   delay(100);
  }
  //pregunta si pudo calibrar, o si no
  if(contador>=5){
   for(i=0;i<3;i++){
    offset[i+6]=offset[i+6]/contador*390.0/390.0;
   }
  }else{
   for(i=6;i<9;i++){
    offset[i]=1.0;
   }
  }
  //re calibra a la condición inicial deseada
  HMC5883_init();
}
```

Anexo I. Código Macro Excel.

```
Sub MC_PY()

' MC_PY Macro
' MACRO DEL PROYECTO

    Columns("A:A").Select
    Selection.TextToColumns Destination:=Range("A1"), DataType:=xlDelimited, _
    TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=False, Tab:=False, _
    Semicolon:=False, Comma:=True, Space:=False, Other:=False, FieldInfo _
    :=Array(Array(1, 1), Array(2, 1)), TrailingMinusNumbers:=True
    Range("A1").Select
    Range(Selection, Selection.End(xlDown)).Select
    Range(Selection, Selection.End(xlDown)).Select
    Selection.Cut
```

```
Range("A2").Select
ActiveSheet.Paste
Range("B1").Select
Range(Selection, Selection.End(xlDown)).Select
Range(Selection, Selection.End(xlDown)).Select
Selection.Cut
Range("B2").Select
ActiveSheet.Paste
Range("A1").Select
ActiveCell.FormulaR1C1 = "TIPO"
Range("B1").Select
ActiveCell.FormulaR1C1 = "DATO"
Range("A1").Select
Selection.Font.Bold = True
Range("B1").Select
Selection.Font.Bold = True
Cells.Select
With Selection
   .HorizontalAlignment = xlCenter
   .VerticalAlignment = xlBottom
   .WrapText = False
   .Orientation = 0
   .AddIndent = False
   .IndentLevel = 0
   .ShrinkToFit = False
   .ReadingOrder = xlContext
   .MergeCells = False
End With
With Selection
   .HorizontalAlignment = xlCenter
   .VerticalAlignment = xlCenter
   .WrapText = False
   .Orientation = 0
   .AddIndent = False
   .IndentLevel = 0
   .ShrinkToFit = False
   .ReadingOrder = xlContext
   .MergeCells = False
End With
Range("A1").Select
Selection.AutoFilter
ActiveSheet.Range("$A$1:$B$1198").AutoFilter Field:=1, Criteria1:="P"
Range("A2").Select
Range(Selection, Selection.End(xlToRight)).Select
```

```
Range(Selection, Selection.End(xlDown)).Select
Range(Selection, Selection.End(xlDown)).Select
Range(Selection, Selection.End(xlDown)).Select
Selection.Copy
Sheets.Add After:=ActiveSheet
ActiveSheet.Paste
Sheets("Hoja1").Select
Range("A1048576").Select
Selection.End(xlUp).Select
Selection.End(xlUp).Select
Selection.End(xlUp).Select
Range("A2").Select
ActiveSheet.Range("$A$1:$B$1198").AutoFilter Field:=1, Criteria1:="R"
Range("A4:B4").Select
Range(Selection, Selection.End(xlDown)).Select
Range(Selection, Selection.End(xlDown)).Select
Range(Selection, Selection.End(xlDown)).Select
Application.CutCopyMode = False
Selection.Copy
Sheets.Add After:=ActiveSheet
ActiveSheet.Paste
Sheets("Hoja1").Select
Range("A1048576").Select
Selection.End(xlUp).Select
Selection.End(xlUp).Select
ActiveSheet.Range("$A$1:$B$1198").AutoFilter Field:=1, Criteria1:="Y"
Range("A3:B3").Select
Range(Selection, Selection.End(xlDown)).Select
Range(Selection, Selection.End(xlDown)).Select
Range(Selection, Selection.End(xlDown)).Select
Application.CutCopyMode = False
Selection.Copy
Sheets.Add After:=ActiveSheet
ActiveSheet.Paste
Columns("B:B").Select
Selection.Font.Bold = True
Sheets("Hoja4").Select
Sheets("Hoja4").Name = "YAW"
Sheets("Hoja3").Select
Columns("B:B").Select
Selection.Font.Bold = True
Sheets("Hoja3").Select
Sheets("Hoja3").Name = "ROLL"
Sheets("Hoja2").Select
```

```
Columns("B:B").Select
Selection.Font.Bold = True
Sheets("Hoja2").Select
Sheets("Hoja2").Name = "PITCH"
ActiveSheet.Shapes.AddChart2(240, xlXYScatterSmoothNoMarkers).Select
ActiveChart.SetSourceData Source:=Range("PITCH!$B:$B")
ActiveChart.ChartTitle.Select
Sheets.Add After:=ActiveSheet
Sheets("Hoja5").Select
Sheets("Hoja5").Name = "PLOT"
Sheets("PITCH").Select
ActiveChart.ChartTitle.Text = "PITCH"
Selection.Format.TextFrame2.TextRange.Characters.Text = "PITCH"
With Selection.Format.TextFrame2.TextRange.Characters(1, 5).ParagraphFormat
   .TextDirection = msoTextDirectionLeftToRight
   .Alignment = msoAlignCenter
End With
With Selection.Format.TextFrame2.TextRange.Characters(1, 5).Font
   .BaselineOffset = 0
   .Bold = msoFalse
   .NameComplexScript = "+mn-cs"
   .NameFarEast = "+mn-ea"
   .Fill.Visible = msoTrue
   .Fill.ForeColor.RGB = RGB(89, 89, 89)
   .Fill.Transparency = 0
   .Fill.Solid
   .Size = 14
   .Italic = msoFalse
   .Kerning = 12
   .Name = "+mn-lt"
   .UnderlineStyle = msoNoUnderline
   .Spacing = 0
   .Strike = msoNoStrike
End With
ActiveChart.ChartArea.Select
ActiveChart.ChartArea.Copy
Sheets("PLOT").Select
ActiveSheet.Paste
Sheets("ROLL").Select
ActiveSheet.Shapes.AddChart2(240, xlXYScatterSmoothNoMarkers).Select
ActiveChart.SetSourceData Source:=Range("ROLL!$B:$B")
ActiveChart.ChartTitle.Select
ActiveChart.ChartTitle.Text = "ROLL"
Selection.Format.TextFrame2.TextRange.Characters.Text = "ROLL"
```

```
With Selection.Format.TextFrame2.TextRange.Characters(1, 4).ParagraphFormat
   .TextDirection = msoTextDirectionLeftToRight
   .Alignment = msoAlignCenter
End With
With Selection.Format.TextFrame2.TextRange.Characters(1, 4).Font
   .BaselineOffset = 0
   .Bold = msoFalse
   .NameComplexScript = "+mn-cs"
   .NameFarEast = "+mn-ea"
   .Fill.Visible = msoTrue
   .Fill.ForeColor.RGB = RGB(89, 89, 89)
   .Fill.Transparency = 0
   .Fill.Solid
   .Size = 14
   .Italic = msoFalse
   .Kerning = 12
   .Name = "+mn-lt"
   .UnderlineStyle = msoNoUnderline
   .Spacing = 0
   .Strike = msoNoStrike
End With
ActiveChart.ChartArea.Select
ActiveChart.ChartArea.Copy
Sheets("PLOT").Select
Range("H1").Select
ActiveSheet.Paste
Sheets("YAW").Select
ActiveSheet.Shapes.AddChart2(240, xlXYScatterSmoothNoMarkers).Select
ActiveChart.SetSourceData Source:=Range("YAW!$B:$B")
ActiveChart.ChartTitle.Select
ActiveChart.ChartTitle.Text = "YAW"
Selection.Format.TextFrame2.TextRange.Characters.Text = "YAW"
With Selection.Format.TextFrame2.TextRange.Characters(1, 3).ParagraphFormat
   .TextDirection = msoTextDirectionLeftToRight
   .Alignment = msoAlignCenter
End With
With Selection.Format.TextFrame2.TextRange.Characters(1, 3).Font
   .BaselineOffset = 0
   .Bold = msoFalse
   .NameComplexScript = "+mn-cs"
   .NameFarEast = "+mn-ea"
   .Fill.Visible = msoTrue
   .Fill.ForeColor.RGB = RGB(89, 89, 89)
   .Fill.Transparency = 0
```

```
        .Fill.Solid
        .Size = 14
        .Italic = msoFalse
        .Kerning = 12
        .Name = "+mn-lt"
        .UnderlineStyle = msoNoUnderline
        .Spacing = 0
        .Strike = msoNoStrike
    End With
    ActiveChart.ChartArea.Select
    ActiveChart.ChartArea.Copy
    Sheets("PLOT").Select
    Range("A16").Select
    ActiveSheet.Paste
    ActiveWindow.Zoom = 85
    ActiveWindow.Zoom = 70
    ActiveWindow.Zoom = 85
    ActiveWindow.Zoom = 70
    ActiveSheet.ChartObjects("Gráfico 3").Activate
    ActiveSheet.Shapes("Gráfico 3").IncrementLeft 842.1428346457
    ActiveSheet.Shapes("Gráfico 3").IncrementTop -225
    ActiveSheet.ChartObjects("Gráfico 1").Activate
    ActiveSheet.ChartObjects("Gráfico 1").Activate
    ActiveSheet.Shapes("Gráfico 1").ScaleWidth 1.1636905074, msoFalse, _
        msoScaleFromTopLeft
    ActiveSheet.Shapes("Gráfico 1").ScaleHeight 1.2549602654, msoFalse, _
        msoScaleFromTopLeft
    ActiveSheet.ChartObjects("Gráfico 2").Activate
    ActiveSheet.ChartObjects("Gráfico 2").Activate
    ActiveSheet.Shapes("Gráfico 2").ScaleWidth 1.1666666667, msoFalse, _
        msoScaleFromTopLeft
    ActiveSheet.Shapes("Gráfico 2").ScaleHeight 1.2450397346, msoFalse, _
        msoScaleFromTopLeft
    ActiveSheet.ChartObjects("Gráfico 3").Activate
    ActiveSheet.ChartObjects("Gráfico 3").Activate
    ActiveSheet.Shapes("Gráfico 3").ScaleWidth 1.1964286964, msoFalse, _
        msoScaleFromTopLeft
    ActiveSheet.Shapes("Gráfico 3").ScaleHeight 1.2450397346, msoFalse, _
        msoScaleFromTopLeft
    Range("A1").Select

End Sub
```

Anexo J. Configuración de *Serial Chart*.

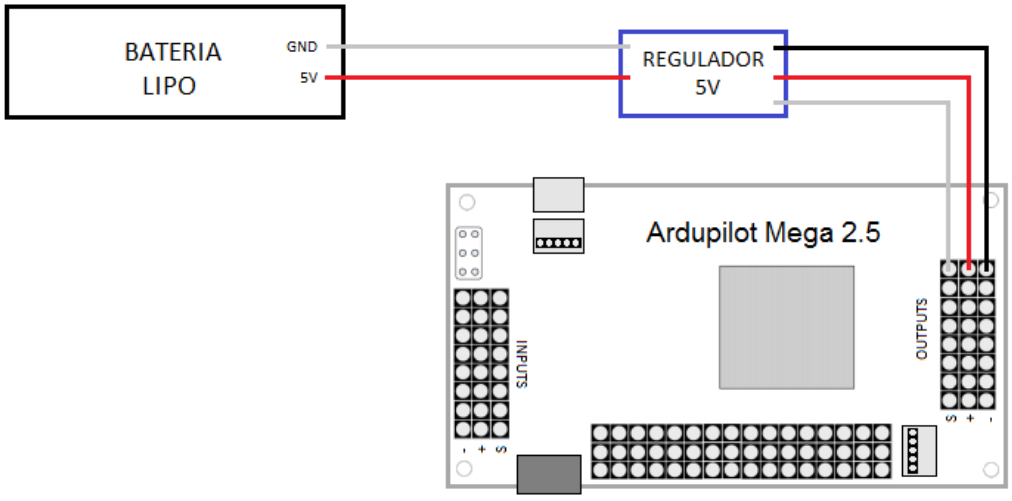
```
[_setup_]
port=COM5
baudrate=57600
width=2000
height=201
background_color = white
grid_h_origin = 100
grid_h_step = 10
grid_h_color = #EEE
grid_h_origin_color = #CCC
grid_v_origin = 0
grid_v_step = 10
grid_v_color = #EEE
grid_v_origin_color = transparent
[_default_]
min=-90
max=90
[Field1]
color=black|
min=-90
max=90
[Field2]
color=blue
min=-90
max=90
[Field3]
color=red
min=-90
max=90
```

Anexo K. Montaje remoto

Para que el montaje remoto funcionara mientras que el vehículo se desplazaba por el circuito diseñado, fue necesario una fuente de alimentación para el *APM* 2.5 que consistía en una batería LiPo[3] de 3 celdas (11.4 V) y un regulador que disminuye este valor de tensión a 5V, necesarios para alimentar la tarjeta de desarrollo, el esquema de alimentación se muestra en la Figura 29.

[3] Batería de polímero de litio, similar a la batería de iones de litio (Li-ion), pero con mayor densidad de energía y mayor tasa de descarga.

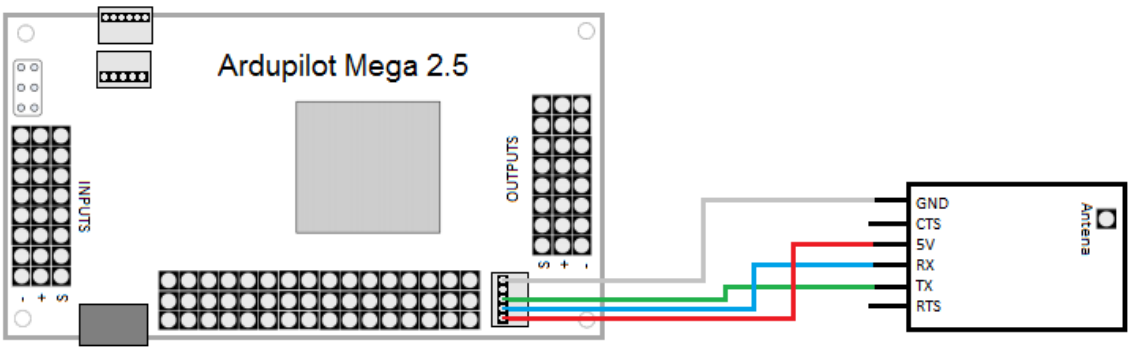
Figura 29. Esquema alimentación *APM* 2.5.



Fuente: Autores.

Adicional a la alimentación, el montaje remoto también necesitó la antena de telemetría para la transmisión de datos al lugar local, ésta se alimentó a su vez con la tensión del *APM*, el envío de datos se hace mediante los puertos de transmisión y recepción seriales (3DR-RADIO) disponibles en la tarjeta, esta conexión se muestra en la Figura 30.

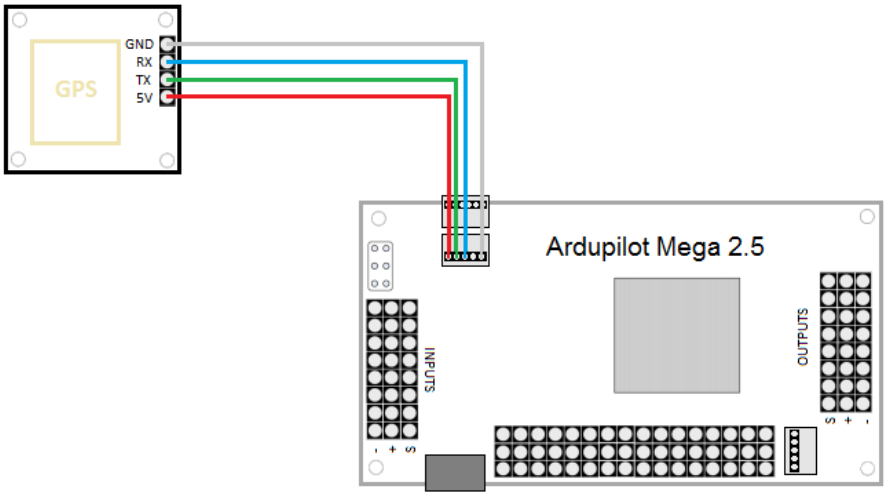
Figura 30. Esquemático antena de telemetría.



Fuente: Autores.

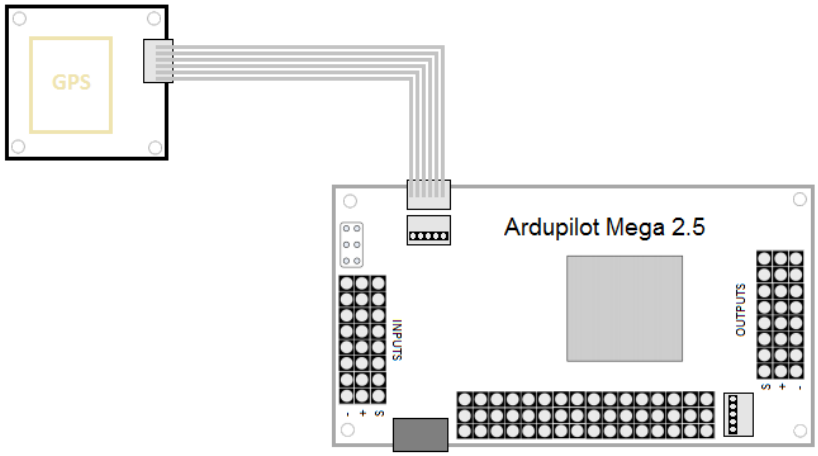
Finalmente se necesita la conexión del GPS para el funcionamiento del código, para este se realiza una conexión serial por el puerto UART como se muestra en la Figura 31, mientras que para el funcionamiento con el firmware del *Mission Planner* se realiza conexión serial por el puerto SPI como muestra la Figura 32.

Figura 31. Conexión GPS para prueba con el código.



Fuente: Autores.

Figura 32. Conexión GPS para prueba con firmware del *Mission Planner*.



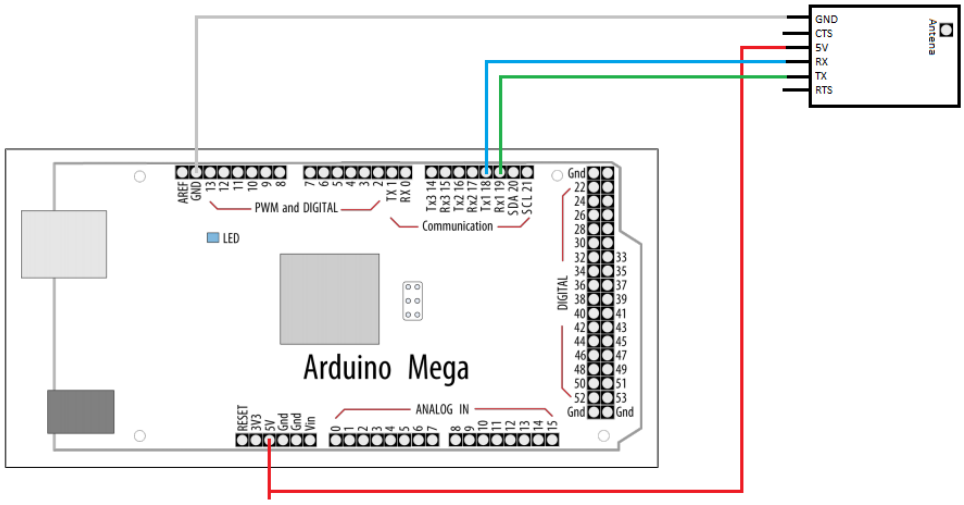
Fuente: Autores.

En el momento de realizar la prueba con el código planteado en el proyecto, éste se debe cargar a la tarjeta y después alimentarla para poner en movimiento el vehículo; si por el contrario se van a comprobar los datos, se debe cargar el firmware de *Mission Planner*.

Anexo L. Montaje local

El montaje local incluye la antena de telemetría necesaria para la recepción de información; si la prueba se va a realizar con el código implementado en el proyecto, se necesita una tarjeta de desarrollo adicional como el Arduino Mega para conectarlo por el puerto de transmisión y recepción como se muestra en la Figura 33.

Figura 33. Conexión telemetría en Arduino Mega.



Fuente: Autores.

Al Arduino Mega es necesario cargarle un código que empiece la transmisión de datos de modo serial y sincronizándolos para recibirlo como una cadena de caracteres (Anexo 13).

Si la prueba se hace con el firmware del *Mission Planner* se debe conectar la antena por cable FTDI al puerto USB del computador y en el programa se reciben los datos enviados por telemetría del montaje remoto.

Anexo M. Código para telemetría.

```
int i = 0;

char cadena[8];

byte contador=0;

int valor=0;


void setup() {

 Serial1.begin(57600);

 Serial.begin(57600);

}


void loop() {

 if(Serial1.available()){

   while (Serial1.available()){

    delayMicroseconds(1500);

    cadena[contador]=Serial1.read();

    contador++;

   }

//    valor=atoi(cadena);

   contador=0;

   Serial.println(cadena);


   i++;

 }
```

```
}
```