Sistema de medición actitudinal de un cuerpo para uso en túneles de viento

Presentado por:

Juan Diego Monroy Rojas

Fundación universitaria los libertadores

Facultad de Ingeniería y Ciencias Básicas

Programa de Ingeniería Aeronáutica

Bogotá D.C.

2019

Sistema de medición actitudinal de un cuerpo para uso en túneles de viento

Presentado por:

Juan Diego Monroy Rojas

Propuesta de trabajo de grado en cumplimiento parcial de los requerimientos para optar por el título de

Ingeniero Aeronáutico

Dirigido por

Iván Felipe Rodríguez Barón M.Sc.

Codirector

Jorge Luis Nisperuza Toledo P.hD

Presentada a

Fundación Universitaria los Libertadores

Facultad de Ingeniería y Ciencias Básicas

Programa de Ingeniería Aeronáutica

Bogotá D.C.

2019

Notas de aceptación

Firma del presidente del jurado

Firma del jurado

Firma del jurado

Bogotá D.C., Marzo de 2019

Las Directivas de la Fundación Universitaria los Libertadores, los jurados calificadores y el cuerpo docente no son responsables por los criterios e ideas expuestas en el siguiente documento. Estos corresponden únicamente a los autores y los resultados de su trabajo

Dedicatoria

Dedico este proyecto de grado a mis padres, quienes siempre me apoyaron económicamente, emocionalmente y en todas las decisiones que he tomado junto a mi familia.

A mis maestros que siempre tuvieron la paciencia para enseñarme y me motivaron a ser mejor profesional sin dejar al lado ser excelente persona.

Agradecimientos

Un sincero agradecimiento a:

Msc. Iván Felipe Rodríguez Barón

Gracias a su apoyo pude concluir este trabajo de grado, sus ayudas fueron importantes para la realización de la mejor forma posible y evitaron que no lo llegara a concluir.

Contenido

Índice de figuras

Índice de tablas

Abreviaturas

Cl: Coeficiente de sustentación

S: Superficie de contacto

L: Sustentación

q: Presión dinámica

ρ: Densidad

Ø: Angulo

ω: Velocidad angular

Resumen

Dentro de la industria aeronáutica existen diversos tipos de enfoques, como el diseño, el desarrollo, el mantenimiento y puesta a punto de aeronaves y sus componentes, entre otros. Los cuales deben ser llevados a cabos por el conjunto del sector privado (industria), el sector público (estado) y el sector académico (centros de estudio), este último, encargado de generar nuevas conocimientos, investigaciones y desarrollos, que sean regulados por el estado e industrializado por el sector privado. En este trabajo de grado se realizará un sistema el cual en tiempo real nos mostrara el comportamiento actitudinal de un perfil aerodinámico, reconociendo su ángulo de ataque en el cual se encuentra mediante un acelerómetro y giróscopo el cual es el MPU-6050 y para que todos estos datos adquiridos sean visibles en una interfaz amigable con el usuario, se utilizó una placa de Arduino UNO que al conectarla al computador nos muestra todos valores en tiempo real de una forma más ordenada. Teniendo en cuenta el sistema realizado, se aplicó en el túnel de viento de la Fundación Universitaria Los Libertadores para comparar la toma de datos del sistema que se realizó con la forma que se utiliza actualmente en los laboratorios que se desarrolla la cual es por diferencial de presión con los tubos pitot, esto se realizó para mejorar los datos adquiridos. Al realizar las diferentes pruebas y comparar las dos formas de adquisición de datos se concluyó que con el sistema de medición actitudinal los valores son más cercanos al valor teórico el cual siempre se trata de llegar.

Palabras clave

Acelerómetro, Túnel de viento, Arduino, Actitud de una aeronave, comportamiento aerodinámico.

Abstract

Within the aeronautical industry there are several types of approaches, such as design, development, maintenance and tuning of aircraft and their components, among others. Which must be carried out by the whole of the private sector (industry), the public sector (state) and the academic sector (study centers), the latter, responsible for generating new knowledge, research and developments, which are regulated by the state and industrialized by the private sector. In this work of degree will be a system which in real time will show us the attitudinal behavior of an aerodynamic profile, recognizing its angle of attack in which it is found by means of an accelerometer and gyroscope which is the MPU-6050 and so that all This acquired data is visible in a friendly interface with the user, an Arduino UNO board was used, which when connected to the computer shows all values in real time in a more ordered way. Taking into account the system carried out, it was applied in the wind tunnel of the Los Libertadores University Foundation to compare the data collection of the system that was made with the form that is currently used in the laboratories that is developed which is by differential of pressure with the pitot tubes, this was done to improve the acquired data. When performing the different tests and comparing the two forms of data acquisition, it was concluded that with the attitudinal measurement system the values are closer to the theoretical value which is always to arrive

Keywords

Accelerometer, Wind tunnel, Arduino, Attitude of an aircraft, aerodynamic behavior.

Capítulo 1

1.1 Introducción

La Fundación Universitaria Los Libertadores cuenta con un túnel aerodinámico experimental de circuito abierto y de flujo subsónico de baja velocidad, este túnel es de succión, *TELSAT AEROSPACE*, con una estructura hecha en fibra de vidrio, y la sección de prueba está hecha en acrílico, en el cual se utilizan perfiles aerodinámicos y para medir el ángulo de ataque en el cual se encuentra se realiza de forma visual y puede generar errores en la toma de datos. Francis Herbert Wenham (1824-1908), un Miembro del Consejo de la Sociedad Aeronáutica de Gran Bretaña, diseñó y operó el primer túnel aerodinámico en 1871. Un túnel de viento conocido como «tubo aerodinámico» fue diseñado y construido por Tsiolkovski en 1897, desde entonces se han ido mejorando y así la toma de datos de ellos son más precisas, por ende, cuando se vaya a realizar un estudio en el túnel de viento el cual posee la Fundación Universitaria Los Libertadores los datos obtenidos tengan una mayor precisión [8].

La presente investigación se refiere al sistema de adquisición de datos aplicado en el túnel de viento de la Fundación Universitaria Los Libertadores para ver el comportamiento aerodinámico de los perfiles aerodinámicos para observar la actitud en la cual se encuentra y con los datos adquiridos poder hallar la sustentación del perfil aerodinámico el cual se está poniendo a prueba. La característica principal de este sistema es que nos da los datos en tiempo real y cualquier alteración en el perfil aerodinámico se podrá visualizar inmediatamente y poder encontrar valores más cercanos a los valores teóricos. Para analizar el porqué del desarrollo de un sistema de adquisición de datos en tiempo real es necesario mencionar sus causas. Una de ellas es que muchos de los perfiles aerodinámicos que se elaboran para utilizarlos junto a los tubos pitot no están elaborados con las medidas exactas para que las mangueras que van conectadas a los tubos pitot nos muestren el verdadero diferencial de presión por lo tanto puede llegar a alterar los datos que los estudiantes están midiendo.

El trabajo de grado se realizó por el interés de ver si existía una forma más precisa de hallar los valores de sustentación de los perfiles aerodinámicos que se ponen a prueba en el túnel de viento. Por otra parte, el realizar un sistema el cual sea una opción por si en algún caso los tubos pitot llegan a fallar existe otra forma confiable el cual pueden aplicar para desarrollar los laboratorios que se estén realizando en el momento.

El trabajo de grado se realizó por medio de una serie de pruebas con un perfil aerodinámico (NACA-2412) en diferentes posiciones las cuales fueron a -10°, 0°,

10° para poder confirmar que con el sistema de adquisición de datos los valores al hallar la sustentación del perfil aerodinámico son más exactos, durante las pruebas realizadas, uno de los obstáculos en la aplicación del montaje del sistema de adquisición de datos era escoger el lugar del perfil en el cual los datos no fueran afectados por el ambiente en el cual estaba operando por lo tanto se aplicó en el centro de gravedad del perfil y dentro del mismo para que no hubieran interferencias y llegara q generar datos erróneos.

El objetivo de este trabajo de grado es implementar un sistema de toma de datos basado en un acelerómetro y giróscopo el cual será puesto dentro del perfil aerodinámico puesto a prueba el cual va conectado a una plataforma de lectura de datos que permite en tiempo real ver el ángulo de ataque en el cual se encuentra el perfil aerodinámico, pero este trabajo tiene una limitación la cual es que si se llega a superar dos veces la fuerza de gravedad en el cual está expuesto el perfil aerodinámico el acelerómetro junto al giróscopo dejan de transmitir información a la interfaz de datos hasta que se vuelta a encontrar en los rangos permitidos de operación, pero para estas pruebas no hubo ningún inconveniente ya que el perfil estaba fijo al túnel de viento.

1.2 Planteamiento del problema

Un túnel de viento es una herramienta experimental para estudiar los efectos del flujo de aire sobre objetos o cuerpos sólidos.

La Fundación Universitaria Los Libertadores cuenta con un túnel de viento sub sónico el cual en la actualidad es importante para ver el comportamiento de los perfiles aerodinámicos y con ellos obtener datos para el desarrollo de los laboratorios propuestos por la universidad.

Sin embargo, en el desarrollo de estos laboratorios se requieren datos básicos como la presión, temperatura, densidad, velocidad del aire que se genera dentro del túnel de viento y el ángulo de ataque en el cual se encuentra el perfil aerodinámico para poder llegar a encontrar el coeficiente de sustentación del perfil aerodinámico y por último la sustentación del mismo, pero los problemas comienzan en la adquisición de datos ya sea porque los estudiantes no saben manejar correctamente los materiales que brinda la universidad, que los elementos estén mal calibrados, que estén malas condiciones o que el perfil que los estudiantes hayan construido no sea acoplado correctamente lo cual estas causas pueden llegar a generar efectos negativos el cual el más importante es la mala toma adquisición de datos.

Con los inconvenientes que se pueden generar en los laboratorios debería haber un sistema alterno el cual sea de una forma de utilizar más sencilla, más amigable con los estudiantes y que los datos sean claros y precisos en tiempo real pero que al utilizarlo se garantice que los datos adquiridos por el sistema son correctos cuando es puesto aprueba en el túnel de viento y que la diferencia se acerque más a los valores deseados.

¿Cómo garantizar si los datos adquiridos de la actitud del perfil son correctos en la sección de pruebas del túnel de viento y cuál es la diferencia entre el cálculo actual y el del proyecto el cual se está realizando?

1.3 Justificación

En la Fundación Universitaria Los Libertadores se realizan pruebas en el túnel de viento sub sónico de una forma, la cual en la adquisición de datos no es del todo confiable ya que la posición de los perfiles aerodinámicos que se ponen a prueba son puestos en diferentes posiciones la cuales varíen su ángulo de ataque, por lo tanto, si llega haber algún error en este procedimiento los datos adquiridos después de la posición del perfil aerodinámico pueden llegar a ser erróneos, por lo tanto al realizar el montaje del sistema de medición actitudinal en los perfiles aerodinámicos de pruebas tendrán beneficios acorto plazo como la adquisición de datos más confiable y a largo plazo que los laboratorios se podría desarrollar de una formas más rápida y eficiente con este sistema que se propone, además la utilidad de este montaje es que se puede aplicar para cualquier perfil aerodinámico y no es costoso realizarlo por lo que se puede llegar a aplicar en todos los perfiles por separado.

1.4 Estructura del trabajo

En este primer capítulo, será desarrollado el marco conceptual e introducción, es planteada la pregunta de investigación, planteamiento del problema y la justificación del proyecto, con el fin de contextualizar la investigación, seguido de capítulo dos dónde es planteada la metodología a seguir para el desarrollo del mismo.

Los capítulos tres y cuatro contienen los marcos conceptual y teórico, respectivamente, planteando los conocimientos básicos necesarios para el desarrollo de la investigación.

En el capítulo cuatro se realiza la explicación del montaje y desarrollo del sistema de medición actitudinal de un cuerpo, para obtener los resultados que serán analizados en el capítulo seis.

Finalmente serán presentadas las conclusiones del proyecto y en anexo el código computacional utilizado.

2 Objetivos

Objetivo general

Desarrollar un sistema de medición actitudinal para adquisición de datos de posición y temperatura de un perfil aerodinámico en el túnel de viento.

Objetivos específicos.

Realizar el sistema de adquisición de datos en tiempo real para calcular el valor del cabeceo, alabeo y la guiñada de un perfil aerodinámico.

Implementar el sistema de adquisición de datos en el túnel de viento de la Fundación Universitaria Los Libertadores, para hacer pruebas experimentales.

Realizar pruebas aerodinámicas en un perfil NACA 2412 con tres diferentes ángulos de ataque (10°, 0°, -10°) para poder calcular el valor de la sustentación experimental y compararlo con los valores teóricos.

3 Metodología

La metodología utilizada en el trabajo de grado desde el significado de las diferentes metodologías es la metodología cuantitativa, como lo menciona supone un planteamiento, un acercamiento a la realidad del estudio y a la teoría, como en primer lugar, el objeto de análisis es una realidad observable, medible y que se puede percibir de manera precisa [13] y la metodología experimental es un tipo de método de investigación en el que el investigador controla deliberadamente las variables para delimitar relaciones entre ellas, está basado en la metodología científica. En este método se recopilan datos para comparar las mediciones de comportamiento de un grupo control, con las mediciones de un grupo experimental. Las variables que se utilizan pueden ser variables dependientes (las que queremos medir o el objeto de estudio del investigador) y las variables independientes (las que el investigador manipula para ver la relación con la dependiente). Además, debemos controlar todas las demás variables que puedan influir en el estudio (variables extrañas) [14].

a) Fundamentación teórica de los conceptos de acelerómetros, giróscopos, Arduino y túnel de viento.

b) Desarrollo del sistema de adquisición de datos con la aplicación del código a la placa de Arduino para que se visualicen los datos tomados por el acelerómetro y el giróscopo y se puedan visualizar de una forma más clara en la interfaz desarrollada en computador.

c) Aplicación en el túnel de viento de la Fundación Universitaria Los Libertadores junto al perfil aerodinámico que se utilizó para las pruebas.

3.1 Estado del arte

Diseño y construcción de un guante de datos mediante sensores de flexibilidad y acelerómetro

En este artículo se muestra un guante de datos diseñado a partir de sensores de flexibilidad construidos ad hoc (es apropiado) para este prototipo y un acelerómetro. Este guante de datos se ha construido con un sistema de comunicación inalámbrico que facilita en gran modo su ergonomía y facilidad de uso. Se ha realizado una optimización en el tamaño de los datos enviados para poder mantener una tasa de comunicaciones elevada, para lo cual, se ha reducido el número de bits del acelerómetro que hay que enviar, con una reducción en la precisión muy baja. Se presentan los resultados de una investigación en sensores aplicables a los movimientos de los dedos y la mano, y la posibilidad de adquisición y envió inalámbrico a un sistema de procesamiento.

En este proyecto se mejoró la calidad de los valores de los datos transmitidos inalámbricamente a la interfaz del computador por la velocidad en el cual se transmiten sin necesidad de un cable de datos, por lo que al prototipo le da mayor libertad de movimiento y confiabilidad en la adquisición de datos. Todo esto se pudo realizar por los elementos de mayor calidad lo cual también generan un mayor costo en su desarrollo como se puede observar en la siguiente figura (Ver Figura 1).

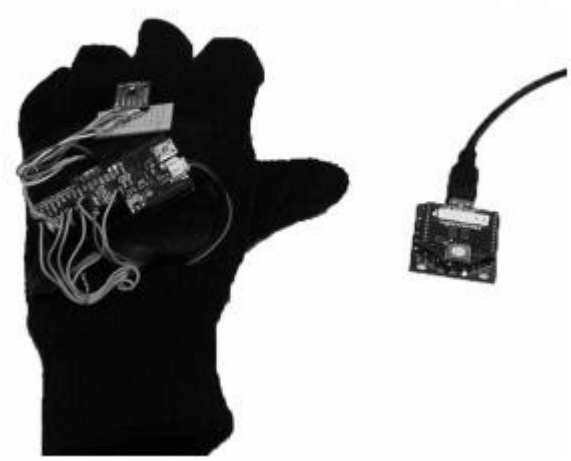

Figura 1 Prototipo guante de datos y terminal de recepción
Fuente: [14]

Control de un Quadrotor mediante la plataforma Arduino

El objetivo de este trabajo es diseñar un sistema UAV quadrotor. Con él se desea tener una plataforma abierta que permita en el futuro implementar y probar nuevos sistemas de control y navegación. Para conseguir este objetivo se diseñó el hardware y software necesario para tener una arquitectura completa, así como un sencillo sistema de control que permita su estabilidad. Para garantizar que haya estabilidad en el UAV hallaron la velocidad angular en cada motor ya que cada motor tiene que volar a una velocidad distinta para que genere sustentación, para eso no se utilizó un MPU como en el proyecto del guante de datos ni en el sistema de adquisición de datos, ya que para la comunicación se necesitó un Xbee ya que genera una comunicación de punto a punto sin necesidad de un cable de datos y además de eso tiene un rango de operación bastante amplio.

Lo que logro este proyecto fue que todo el sistema de sensores y circuitos los montaron en una placa de topos como se observa en la Figura 2 (Ver Figura 2). Esta placa es totalmente adaptable la placa Arduino y no ayudo a ocupar menos espacio y el número de cables y así ensamblar todas las placas una encima de otra, además utilizo diferentes materiales como el Xbee y aplico diferentes materiales como por ejemplo el acelerómetro y giroscopio por componentes separados.

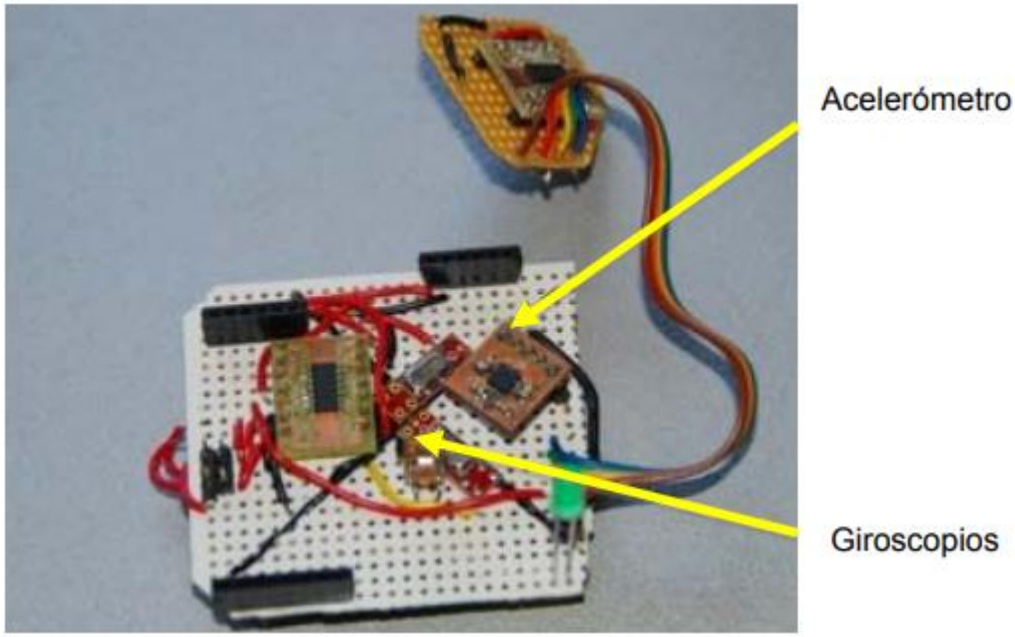

Figura 2 Placa sensores
Autor: [15]

También genero un diseño para encontrar la operación de cada motor como su velocidad e inclinación para mejorar el diseño y esto se dio gracias al número de sensores que se aplicaron al montaje, y el espacio que ocupa el montaje de los sensores es reducido como lo podemos ver en la siguiente figura (Ver Figura 3).

Figura 3 Fotografía del Quadrotor FLISI1
Autor: [15]

4 Marco conceptual

4.1 Sensor acelerómetro

El sensor acelerómetro APU6050 nos permite medir la aceleración en los tres ejes espaciales X, Y, Z. Es un sensor compacto y de tamaño muy reducido. Funciona a 5v y normalmente lo encontramos en forma de módulo con los componentes necesarios para su conexión directa a la placa Arduino [10].

El rango capaz de medir este sensor es de -2G a +2G (G=aceleración de la gravedad, 1G=9.8m/s2 [10].

El sensor se conecta a tres entradas analógicas, una por cada eje de medición, el conector VCC se conecta a 5v [10].

Con los valores de aceleración podemos medir la velocidad, detectar "sacudidas ", detectar *"*gestos*"* o analizar movimientos como andar o correr (*podómetro*). Un ejemplo del uso de acelerómetros es el famoso mando de la video consola Wii (ver Figura 1) (aparte de acelerómetros incluye otros sensores como giroscopios) [5]

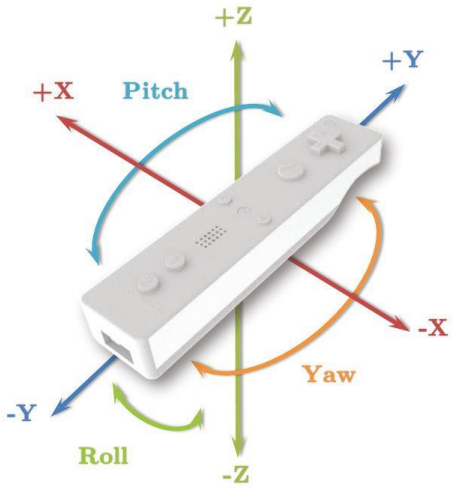

Figura 4 Implementación de un acelerómetro con giróscopo en un control para consola Wii
Fuente: [5]

Con el sensor APU6050 además de obtener los valores de aceleración, mediante algunos cálculos trigonométricos (que *ArduinoBlocks* realiza automáticamente) podemos obtener los valores de rotación (ver Figura 2): alabeo (ángulo de rotación en eje X) y cabeceo (ángulo de rotación en eje Y) [5].

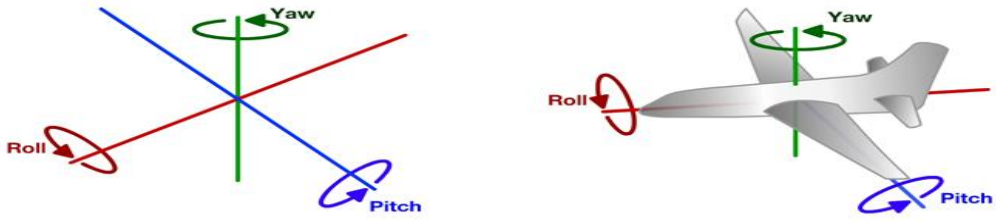

Figura 5 Actitud de la aeronave en sus 3 ejes
Fuente: [5]

```
Alabeo = RAD_TO_DEG * (atan2(-yAng, -zAng) + PI);
Guiñada = RAD_TO_DEG * (atan2(-xAng, -zAng) + PI);
```

El valor de guiñada (ángulo de rotación en eje Z) no podemos calcularlo sólo con un acelerómetro pues en este sentido de giro la aceleración por efecto de la gravedad no varía, por lo que sensor APU6050 combina acelerómetro con giroscopio para calcular el valor de guiñada o sensores magnéticos (IMU) en su defecto [5].

4.2 ¿Qué es un Arduino?

Arduino es una plataforma de prototipos electrónica de código abierto (*open-source*) basada en *hardware y software* flexibles y fáciles de usar. Arduino puede sentir el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros artefactos. El microcontrolador de la placa se programa usando el (*Arduino Programming Language*) (basado en *Wiring*) y el *Arduino Development Environment* (basado en Processing). Los proyectos de Arduino pueden ser autónomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo, con Flash, Processing, MaxMSP, etc.). Las placas se pueden ensamblar a mano o encargarlas pre ensambladas; el software se puede descargar gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia *open-source*, por lo que eres libre de adaptarlas a tus necesidades [3].

4.3 ¿Por qué Arduino?

Hay muchos otros microcontroladores y plataformas microcontroladoras disponibles para computación física. *Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard*, y muchas otras ofertas de funcionalidad similar. Todas estas herramientas toman los desordenados detalles de la programación de microcontrolador y la encierran en un paquete fácil de usar. Arduino también simplifica el proceso de trabajo con microcontroladores, pero ofrece algunas ventajas para profesores, estudiantes y aficionados interesados sobre otros sistemas [3].

Capítulo 5

5 Marco teórico

5.1 Inclinación usando giroscopio y acelerómetro

Tomando como referencia que se necesita un acelerómetro y un giroscopio se utilizan de manera individual, se tienen varios inconvenientes que producen que los resultados deseados (actitud) tienda a tener errores en su magnitud.

Debido a estos inconvenientes, surge la necesidad de trabajar con ambos sensores simultáneamente, con el fin de poder tomar las mejores características de cada uno y que a la vez se contrarresten en cierto grado los inconvenientes.

Existe un método capaz de lograr esta fusión de características positivas y de atenuación de los problemas individuales, este método es llamado filtro de complemento acelerómetro más giroscopio [10].

5.2 Filtro de complemento Acelerómetro + Giroscopio

El filtro de complemento o en inglés "*Complementary Filter*" es uno de los más usados por su fácil implementación, combina el ángulo calculado por el giroscopio y el ángulo calculado por el acelerómetro.

La necesidad de combinar ambas lecturas es que, si solo trabajamos con el acelerómetro, este es susceptible a las aceleraciones producto del movimiento del MPU o a fuerzas externas, pero en tiempos largos el ángulo no acumula errores. A diferencia que si trabajamos solo con el giroscopio si bien este no es susceptible a fuerzas externas, con el tiempo el desvío es muy grande y nos sirve solo para mediciones de tiempos cortos [10].

5.3 Filtro de Kalman

El filtro de Kalman es un algoritmo que fue desarrollado por Rudolf E. Kalman en 1960 y describe una solución recursiva para problemas de filtrado de datos discreto. Desde su publicación, es aplicado especialmente en sistemas de navegación autónomos o asistidos.

El filtro de Klaman es un estimador óptimo que puede implementarse de manera sencilla en sistemas de carácter tanto lineal como no lineal, y cuyo procesamiento de datos es de carácter recursivo. Se denomina óptimo ya que recibe y procesa todas las mediciones disponibles y en base a estas, estima el valor actual de las variables de interés. Para llevar a cabo este algoritmo es necesario [6].

Uno de los mejores filtros para eliminar el desvio es el filtro Kalman, pero se necesita una buena capacidad de procesamiento computacional, haciéndolo difícil implementar en Arduino. Por eso utilizamos el filtro de complemento [10].

En la toma de datos al saber la actitud en la cual se encuentra el perfil en el túnel de viento el solo nos dará el ángulo de ataque en el cual se encuentra, teniendo ese dato a continuación podremos encontrar la sustentación que nos genera el perfil con las tablas de introducción to flight [4] que se encuentran al final del libro nos da unas gráficas para cada uno de los perfiles NACA [10].

Para desarrollar las ecuaciones siguientes primero debemos conocer las variables las cuales manejaremos en las ecuaciones y así encontrar la sustentación en las gráficas [4].

$$\text{Presión dinámica (q)} = \frac{1}{2}\rho * V^2 \qquad \textit{(1)}$$

Donde ρ es la densidad en la cual se encuentra el perfil aprueba y V es la velocidad del perfil o en su defecto en el túnel de viento la velocidad del aire que es generada el túnel.

Para el coeficiente de sustentación (Cl) tenemos la ecuación:

$$\mathrm{Cl} = \frac{\mathrm{L}}{\mathrm{q} * \mathrm{S}} \qquad \textbf{(2)}$$

Donde S es el área de contacto del perfil la cual está generando sustentación y la sustentación (L)

Con la siguiente ecuación podemos hallar la sustentación (L) la cual es:

$$\mathrm{L} = \mathrm{q} * \mathrm{S} * \mathrm{Cl} \qquad \textbf{(3)}$$

Capítulo 6

Montaje y desarrollo del sistema

6.1.1 Montaje

El montaje se desarrolló con un acelerómetro MPU-6050 con el que funciona con un voltaje de 5v, que va conectado a una placa Arduino ADC que obtiene el voltaje de salida del acelerómetro por el modulo del micro controlador en el cual es donde se pone la formula la cual nos indica la actitud del perfil, ya obtenido el ángulo dado en grados. La corrección de los ángulos que arroja el acelerómetro para que haya menor error se explica en el diagrama de bloques (ver Figura 3).

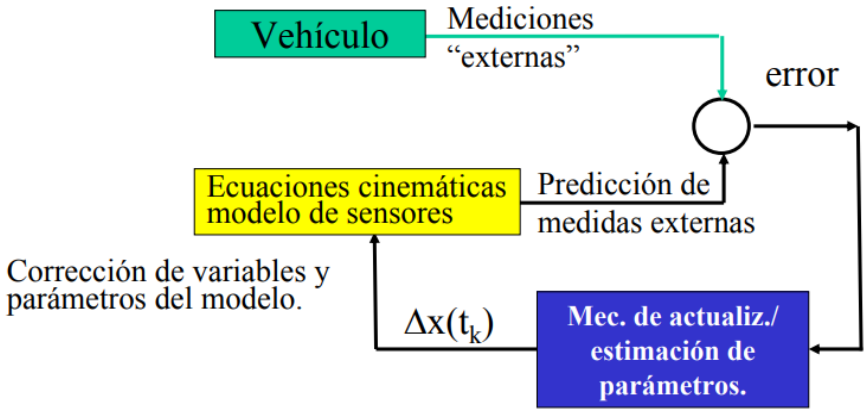

Figura 6 Navegación integrada
Fuente: [2]

6.1.2 Elementos del montaje experimental

En el montaje se utilizó un acelerómetro MPU-6050 y una placa de Arduino UNO las cuales poseen las siguientes características.

6.1.3 Sensor MPU-6050

El MPU-6050 como se muestra (ver Figura 4) es una unidad de medición inercial o IMU (*Inertial Measurment Units*) de seis grados de libertad pues combina un acelerómetro de tres ejes y un giroscopio de tres ejes. Este sensor es muy utilizado en navegación, goniometría, estabilización, etc.

Figura 7 Datasheet MPU6050
Fuente: [1]

El módulo Acelerómetro MPU tiene un giroscopio de tres ejes con el que podemos medir velocidad angular y un acelerómetro también de tres ejes como se muestra (ver Figura 5) con el que medimos los componentes X, Y y Z de la aceleración. La dirección de los ejes está indicada en el módulo el cual hay que tener en cuenta para no equivocarnos en el signo de las aceleraciones.

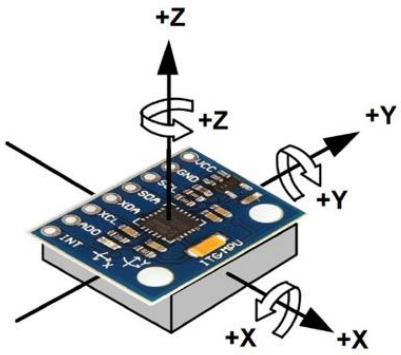

Figura 8 Signo de las aceleraciones
Fuente: [3]

6.1.4 Calculando el ángulo de inclinación con el acelerómetro del MPU6050

Se tiene en cuenta que la única fuerza que actúa sobre el sensor es la fuerza de la gravedad. Entonces los valores que obtenemos en las componentes del acelerómetro corresponden a la gravedad y los ángulos de la resultante serán la inclinación del plano del sensor, puesto que la gravedad siempre es vertical (ver Figura 6).

Para entenderlo mejor, asumimos que se encuentra en un plano X-Z y se inclina el MPU6050 un ángulo θ, dicho ángulo se calcula de la siguiente forma [3].

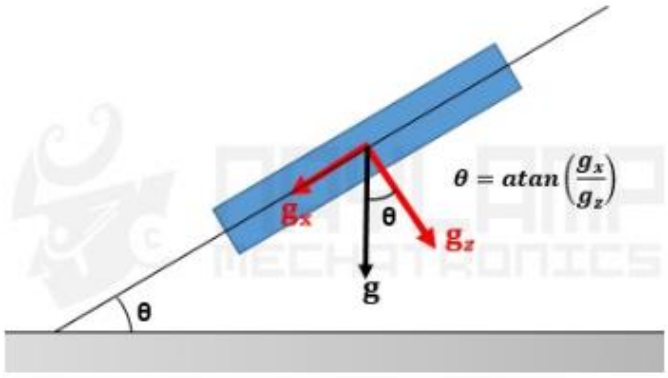

Figura 9 Ángulo de inclinación
Fuente: [3]

Lo anterior sirve para calcular el ángulo en un plano 2D, pero para calcular los ángulos de inclinación en un espacio 3D tanto en X como en Y se utiliza las siguientes formulas [3]:

$$\emptyset_x = tan^{-1}\left(\frac{a_x}{\sqrt[2]{a_y^2+a_z^2}}\right) \quad \textit{(4)}$$

$$\emptyset_y = tan^{-1}\left(\frac{a_y}{\sqrt[2]{a_x^2+a_z^2}}\right) \quad \textit{(5)}$$

Esto funciona solo si la única aceleración presente es la gravedad, pero si se mueve rápidamente el MPU y sin realizar ninguna inclinación el ángulo que se obtiene con el programa anterior varía, generando errores para estos casos.

6.1.5 Calculando el ángulo de rotación usando el giroscopio del MPU5060

Como se explicó al inicio, el giroscopio nos entrega la velocidad angular, y para calcular el ángulo actual se necesita integrar la velocidad y conocer el ángulo inicial. Esto se realiza usando la siguiente formula [3]:

$$\emptyset_x = \emptyset_{x0} + \omega_x \, \Delta t \quad \textit{(6)} \qquad \emptyset_y = \emptyset_{y0} + \omega_y \, \Delta t \quad \textit{(7)}$$

Donde la velocidad angular es igual:

$$\omega = \left(\frac{d\emptyset}{dt}\right) \quad \textit{(8)}$$

Con un giroscopio se puede medir la velocidad angular, y si se integra la velocidad angular con respecto al tiempo se obtiene el desplazamiento angular (posición angular si se sabe dónde se inició el giro)

Tener en cuenta que cuando nos referimos a θx nos referimos al ángulo que gira el eje X sobre su propio eje. En la siguiente imagen (ver Figura 7), se observa que la velocidad angular es perpendicular al plano de rotación. [3]

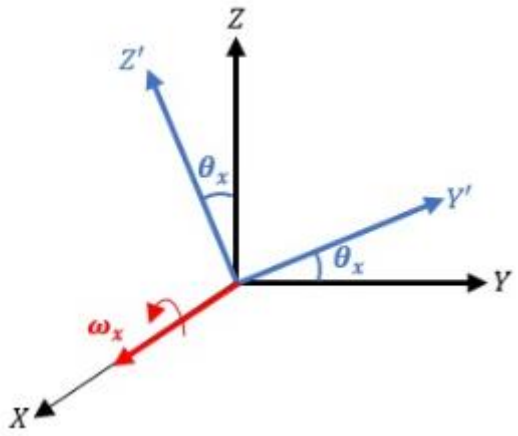

Figura 10 Ángulos de rotación en el plano
Fuente: [3]

6.1.6 Acelerómetro + giroscopio

La ecuación para calcular el ángulo usando el filtro de complemento es:

$$Angulo\ = 0.98\left(\text{angulo} + \omega_{giroscopio}\right) + 0.02(angulo_{acelerometro}) \textbf{ (9)}$$

De esta forma el ángulo del acelerómetro está pasando por un filtro pasa bajos, amortiguando las variaciones bruscas de aceleración; y el ángulo calculado por el giroscopio tiene un filtro pasa altos teniendo gran influencia cuando hay rotaciones rápidas. [3]

6.1.7 Arduino UNO

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso [3].
Como se muestra (ver Figura 8), podemos observar todos sus componentes.

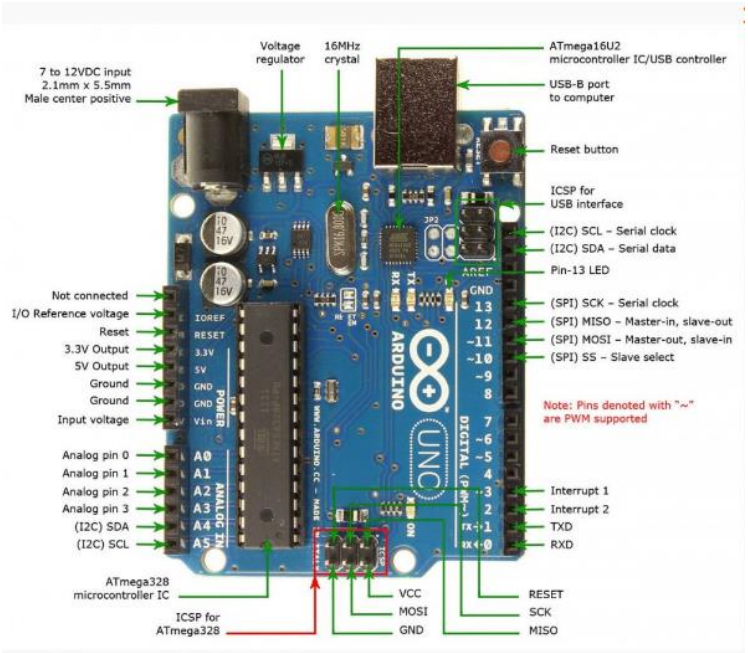

Figura 11 Datasheet Arduino UNO
Fuente: [3]

6.2 Aplicación Práctica

La aplicación experimental se realizó con el fin de demostrar la capacidad de un sistema para el cálculo de los ángulos de los objetos que se ponen aprueba en el túnel de viento, comprobándolo con las mediciones que anteriormente se hacían la cual es bastante inexacta (visualmente y diferencial de presiones) donde en el túnel de viento se tiene un perfil NACA 2412 con el cual se realizaron las pruebas.

Para hallar el valor del ángulo de ataque el cabeceo (*Pitch*), con el MPU-6050 conectado al Arduino el sistema recolecta los datos y los muestra en el computador. Se tomaron tres medidas en diferentes posiciones en el cual se encontraba el NACA 2412 (perfil aerodinámico) para saber si los valores al modificarlo tienen coherencia con la medida visual y así poder deducir que los valores que se están hallando son valores de plena confianza.

Capítulo 7

7.1 Análisis de resultados

El MPU-6050 posee múltiples librerías las cuales facilitan su uso y las cuales se pueden encontrar por internet ya creadas, la librería que se utilizó realiza una comunicación I2C con el modulo extrayendo en tiempo real los datos censados y así se realizó el proyecto de manera modular.

Lo pasos que realiza el código son:

1. Extraer datos del giróscopo y convertirlo en grados.

2. Extraer los datos del acelerómetro y convertirlos en grados.

3. Unir las dos lecturas de giróscopo y acelerómetro para tener medidas más estables utilizando el filtro del complemento el cual fue especificado anteriormente.

4. El programa realiza una calibración periódica para disminuir los errores que

se puedan llegar a generar al iniciar el código.

Todo lo anterior se visualiza por serial.

A continuación, podremos observar los valores que nos arroja el código el cual es mostrado en el programa llamado *processing* (ver figura 9), en el cual se utiliza un proceso para dibujar las figuras, otro para adquisición de los datos de la comunicación serial y basados en estos se rota la figura previamente dibujada y se imprimen los ángulos obtenidos, lo cual se logra enviando los datos del código ya programado en Arduino en paquetes serial para realizar la visualización.

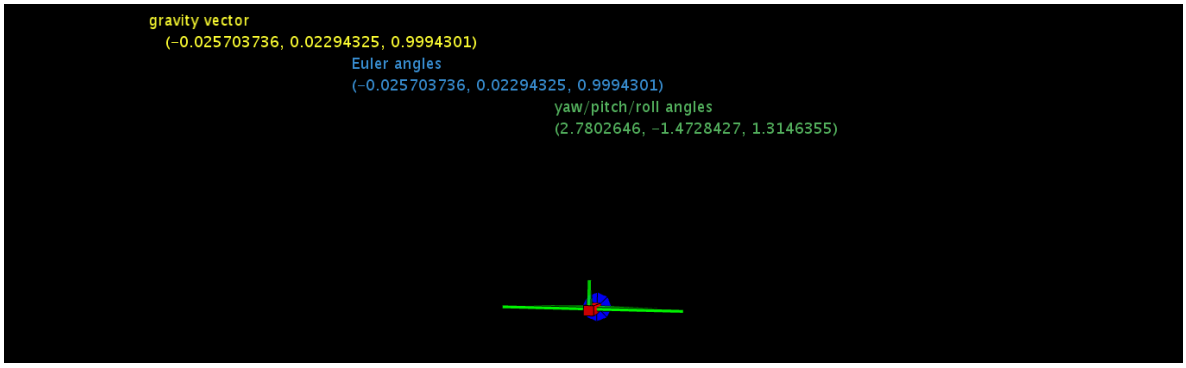

Figura 12 Datos en la interfaz de Processing
Fuente: (Autor)

También si desea ver los valores en tiempo real en la interfaz que nos ofrece Arduino (ver Figura 10), se puede observar tanto los valores de cabeceo, alabeo y guiñada más la temperatura en grados Celsius, para visualizar los datos sin especificar los valores y sin que salga la gráfica de la aeronave debe correr el código por la plataforma de Arduino.

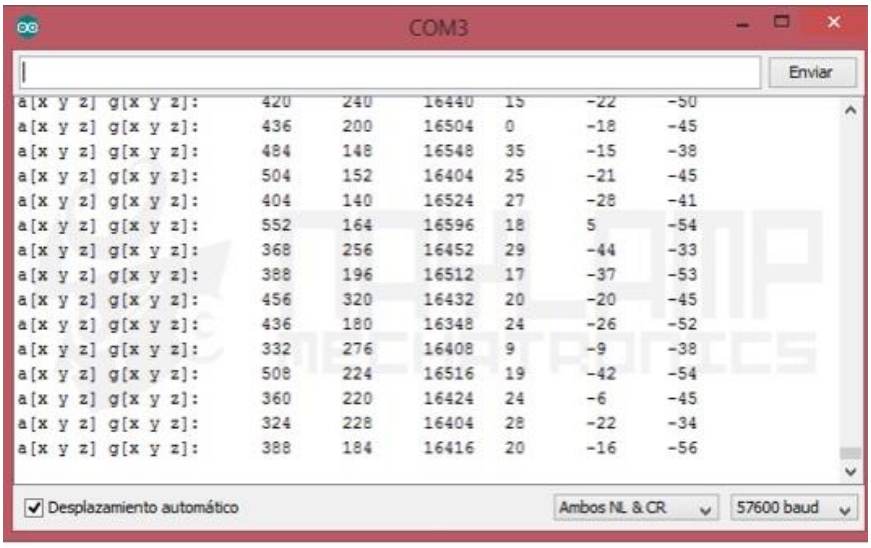

Figura 13 Datos del MPU-6050 en la interfaz de Arduino
Fuente: (Autor)

Para realizar una toma de datos de forma correcta en cualquier prueba que se vaya a realizar se deben de tener tres factores en cuenta.

1. El acelerómetro donde quiera que se encuentre debe de estar fijado al cuerpo (en este caso e coloco en el centro de gravedad del perfil) para que al variar el ángulo de ataque del perfil el ángulo mostrado en la interfaz sea el más preciso.

2. El perfil debe estar completamente estático ya que los valores que está mostrando son valores en tiempo real lo cual al variar el ángulo de ataque al instante los valores que se están visualizando cambian y podrían afectar el laboratorio que se esté realizando.

3. No importa en qué posición se ponga el perfil ya que el acelerómetro MPU-6050 mostrara el ángulo en que se encuentre, pero si se necesita que el perfil se encuentre fijo para que no haya interferencias en la toma de datos.

Se puede observar en las figuras anteriores que existe una mayor exactitud en la toma de datos generado por el acelerómetro y giróscopo MPU-6050 ya que muestra valores en tiempo real, la otra forma en el cual se realizaba era utilizando unos marcadores de color negro y marcar en la parte exterior del túnel junto a un

transportador para acercarse lo más posible al ángulo en el que el perfil aerodinámico se encuentra (ver Figura 11).

Figura 14 Ángulos marcados en el túnel de viento
Fuente: (Autor)

Este es el montaje final en el túnel de viento del perfil NACA 2412 y en el centro de gravedad dentro del perfil se encuentra el acelerómetro y giróscopo por Arduino conectado al computador en el cual se puede visualizar los valores que se están realizando en tiempo real (ver Figura 12).

Figura 15 Montaje final
Fuente: (Autor)

Valores de lift en diferentes coeficientes de lift: $L = C_l * \frac{1}{2}\rho * V^2 * S$

Hallamos Re (Reynolds) para utilizarlo en la tabla del perfil NACA-2412 que más adelante se encuentra:

$Re = \frac{\rho.c.v}{\mu}$ Donde de ρ es densidad del aire (kg/m^3), C longitud característica de la geometría en m, v velocidad del fluido en (m/s) y μ viscosidad absoluta del fluido (m^2/s). El fluido en este caso es el aire

$Re = \frac{0.895.0.175.20}{1.789*10^{-5}}$ = 175097.820011

Valor de coeficiente de sustentación para -10°= -0.8 (valor teórico)

Valor de coeficiente de sustentación para 0°= 0.2 (valor teórico)

Valor de coeficiente de sustentación para 10°= 1.2 (valor teórico)

ρ En Bogotá= 0.895$\frac{Kg}{m^3}$ [9]

V= se utilizó una velocidad de 0.02km/s = 20m/s

Wing spam=60cm, cuerda tip=18 y una cuerda de raíz de 18

S= 600cm^2 = 0.06m^2

Valor de coeficiente de sustentación para -10°= -0.85 (valor medido con el MPU6050)

Valor de coeficiente de sustentación para 0°= 0.27 (valor medido con el MPU6050)

Valor de coeficiente de sustentación para 10°= 1.3 (valor medido con el MPU6050)

Para la posición más cercana a los grados propuestos se realizó con el MPU-6050 y luego sin mirar la interfaz se posiciona frontalmente para poner en donde se encuentra el transportador y se utilizó los ángulos que en perspectiva se visualizaron los cuales fueron:

Valor de coeficiente de sustentación para -10°= -0.6 (valor medido de forma visual)

Valor de coeficiente de sustentación para 0°= 0.5 (valor medido de forma visual)

Valor de coeficiente de sustentación para 10°= 1.1 (valor medido de forma visual)

Estos ángulos pueden variar drásticamente dependiendo de la perspectiva en la cual e observa el perfil aerodinámico.

La diferencia de los valores se puede observar en la siguiente tabla (ver Tabla 1)

Calculo de sustentación teórico	Calculo de sustentación con medición de ángulo de ataque visual	Calculo de sustentación con medición de ángulo de ataque por mpu-6050	Ángulos
-8.592 N	-7.44 N	-9.12 N	-10°
2.14 N	2.21 N	2.19 N	0°
12.88 N	11.81 N	13.96 N	10°

Tabla 1 Tabla de comparación con diferentes tipos de medición de ángulo de ataque
Fuente: (Autor)

En la siguiente tabla puede observar la diferencia de error porcentual en los tres datos adquiridos en los tres ángulos ya propuestos (ver Tabla 2)

Calculo de error porcentual de sustentación de medición de ángulo de ataque visual	Calculo de erro porcentual de sustentación con medición de ángulo de ataque mpu-6050	Ángulos
13.4%	5.15%	-10°
3.27%	2.34%	0°
8.31%	8.39%	10°

Tabla 2 Tabla de comparación de error porcentual con diferentes tipos de medición de ángulo de ataque
Fuente: (Autor)

Como se puede observar los valores medidos con el MPU-6050 son muy cercanos, claro está que los valores pueden mejorar si se utilizan materiales los cuales atenúen el ruido generado en el montaje para que los valores traten cada vez más a estabilizarse y sean más confiables, los valores que se tomaron fueron los que se observó que se repetían más veces y se hizo una aproximación para la facilidad de utilizar las tablas del perfil NACA-2412 que está a continuación (ver Tabla 3).

La siguiente tabla muestra los valores de sustentación en los tres diferentes ángulos ya planteados anteriormente:

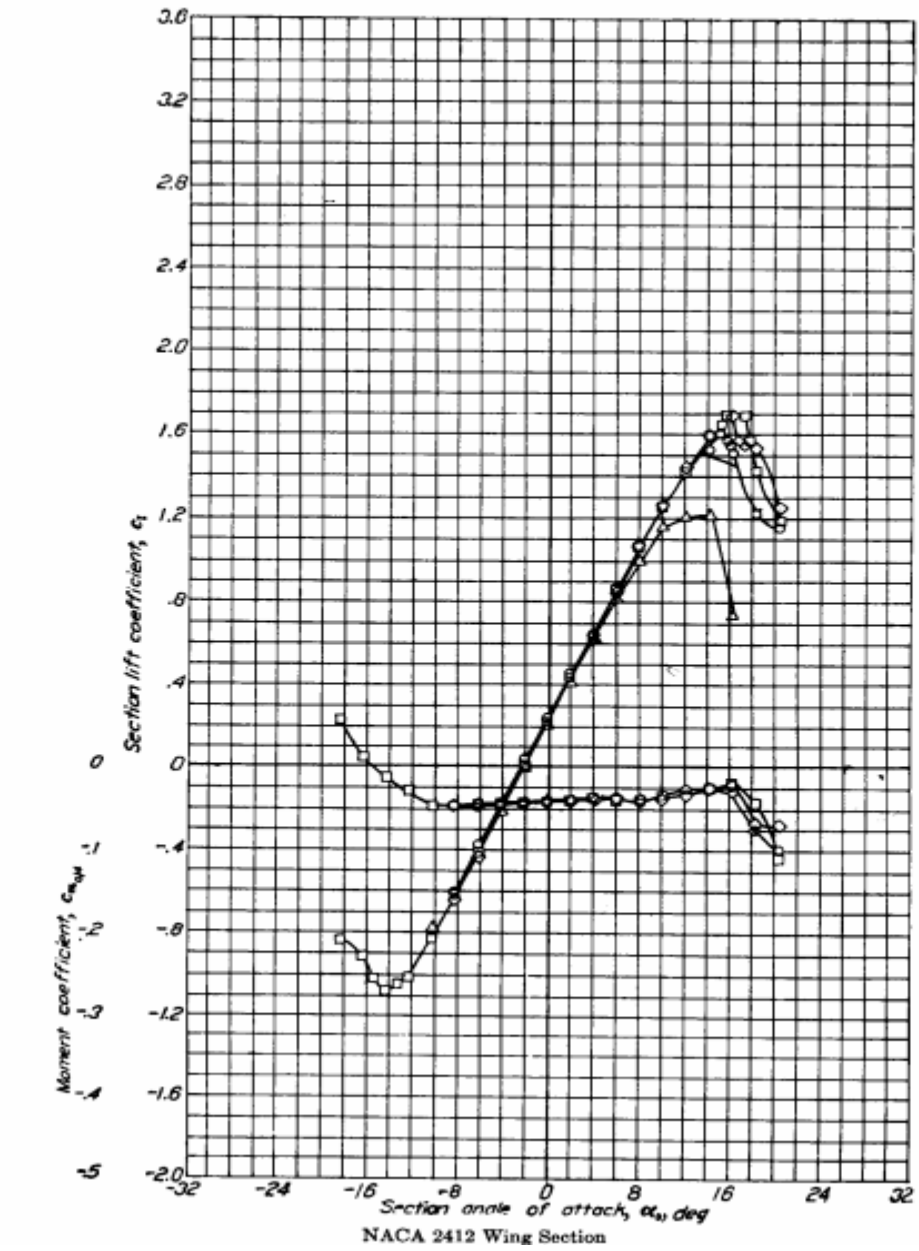

Tabla 3 Tabla de CL y Cd NACA-2412
Fuente: [4]

Capítulo 8

8.1 Discusión

En las mediciones del proyecto se puede observar que los valores obtenidos en los ángulos cabeceo, alabeo y guiñada son afectados por el tipo de perfil aerodinámico ya que no son posiblemente rectos en su totalidad, también cabe resaltar que una prueba anterior con un perfil realizado en fomi no fue lo suficientemente rígido y los valores eran afectados por la deflexión del perfil.

Los resultados obtenidos en este proyecto, muestra que con el prototipo planteado del sistema de adquisición de datos por sensores de acelerómetro y giroscopio, efectivamente hubo una mejora en los datos obtenidos ya que se acercan más al valor teórico al que siempre se trata de llegar.

Por lo anterior, se acepta que es más confiable de utilizar y con el bajo costo de su montaje para cada perfil aerodinámico es viable aplicarlo en los futuros laboratorios, ya que se obtendrá valores más exactos.

Por otro lado, se comprueba que en la variación en la posición del perfil aerodinámico no importa sus condiciones de operación, solo se necesita no sobrepasar la fuerza de dos veces la gravedad para que sea constante la toma de datos, de igual forma si se sobrepasa al volver a las condiciones de operación el muestra en ese preciso momento la actitud del perfil en el cual se encuentra, por lo que no afectaría gravemente los datos finales.

Por lo mencionado en los párrafos anteriores y comparándolo con los dos proyectos ya mencionados anteriormente en el estado del arte, los proyectos tienen condiciones de operaciones más altas por sus materiales y la misión de cada uno, por lo que al utilizar comunicación inalámbrica tiene mayor libertad y sus datos son más constantes por el contrario de un cable de datos, si uno quiere aproximarse cada vez más se necesita materiales de alta calidad por lo que el diseño y la construcción del montaje elevaría su costo, en eso el proyecto realizado del sistema de adquisición de datos por sensores es mucho más económico y su aplicabilidad en el túnel de viento de la Fundación Universitario Los Libertadores es alta por su bajo costo y los datos más cercanos que por presiones.

Cabe mencionar que el montaje que se realizó en este proyecto puede ser aplicado en todos los perfiles aerodinámicos por el contrario de los anteriores por sus múltiples sensores que tienen pueden llegar a alterar los datos que se obtienen en el túnel de viento por qué se da en el ejemplo del proyecto de control de quadrotor mediante la plataforma Arduino el cual muestra que necesita la actitud de cuatro partes del montaje final y el perfil aerodinámico consta de una placa fija por lo que

si se desea aplicar más sensores al perfil aerodinámico deberíamos aplicar más superficies móviles para que no sea un costo más generado de generar y haga la misma función que con menos sensores.

Capítulo 9

Conclusiones

Se desarrolló un sistema de medición actitudinal para adquisición de datos de posición y temperatura de un perfil aerodinámico en el túnel de viento el cual fue implementado en un perfil aerodinámico con éxito.

Se realizó un sistema de adquisición de datos en tiempo real para calcular el valor del cabeceo, alabeo y la guiñada de un perfil aerodinámico el cual responde perfectamente, para mayor exactitud en la adquisición de datos seguir las recomendaciones mencionadas anteriormente.

Se implementó el sistema de adquisición de datos en el túnel de viento de la Fundación Universitaria Los Libertadores, con el cual se realizaron las pruebas experimentales que nos muestran que con el sistema desarrollado por acelerómetro y giróscopo se acerca más a los valores teóricos.

Se realizó pruebas aerodinámicas en el perfil NACA-2412 y con las pruebas realizadas en el túnel de viento puedo concluir que los ángulos los cuales muestra el sistema de adquisición de datos son los ángulos reales en el cual se encuentra el cuerpo, el cual se encuentra a prueba un perfil (NACA 2412) en los diferentes ángulos (10°,0°,-10°).

Bibliografía

[1] 101, C. (s.f.). *MPU6050 - Accelerometer and Gyroscope Module*. Recuperado el 12 de Diciembre de 2019, de https://components101.com/sensors/mpu6050-module

[2] Aires, F. i. (Noviembre de 2007). *Navegacion integrada.* Recuperado el 01 de Diciembre de 2018, de http://laboratorios.fi.uba.ar/lscm/espana/apuntes/Introduccion.pdf

[3] Electronics, M. (s.f.). *Arduino*. (Arduino.CL) Recuperado el 26 de Octubre de 2018, de https://arduino.cl/que-es-arduino/

[4] Jr, J. D. (1989). *INTRODUCTION TO FLIGHT.* Washington: McGraw-Hill Science Engineering.

[5] Llamas, L. (s.f.). *Ingeniería, informatica y diseño*. Recuperado el 20 de Octubre de 2018, de Determinar la orientación con Arduino y el IMU MPU-6050: https://www.luisllamas.es/arduino-orientacion-imu-mpu-6050/

[6] Maybeck, P. S. (1979). *Stochastic Models, Estimation and Control: Volume 1.* Recuperado el 19 de Diciembre de 2018

[7] Mechtronics, N. (s.f.). *Tutorial MPU6050, Acelerómetro y Giroscopio*. Recuperado el 15 de Diciembre de 2018, de https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Acelerómetro-y-Giroscopio.html

[8] Mejías, A. M. (Octubre de 2012). *DISEÑO Y ANÁLISIS COMPUTACIONAL PARA TUENEL DE VIENTO DE BAJA VELOCIDAD .* Recuperado el 20 de Noviembre de 2018, de https://e-archivo.uc3m.es/bitstream/handle/10016/15901/pfc_alberto_munoz_mejias_2012.pdf

[9] Mónica Patricia Burgos Gutiérrez, S. A. (15 de Diciembre de 2018). *Análisis del recurso energético eólico para la ciudad de Bogotá DC.* Obtenido de Análisis del recurso energético eólico para la ciudad de Bogotá DC: http://www.unilibre.edu.co/revistaavances/14/analisis-del-recurso-energetico-eolico-para-la-ciudad-de-bogota.pdf

[10] Pozo Espín, D. F. (22 de Febrero de 2010). *Diseño y construcción de una plataforma didáctica para medir ángulos de inclinación usando sensores inerciales como acelerómetro y giroscopio*. Recuperado el 05 de Enero de 2019, de http://bibdigital.epn.edu.ec/handle/15000/1794

[11] wholesale, F. p. (s.f.). *Arduino UNO R3*. Recuperado el 12 de Diciembre de 2018, de http://full-parts.com/arduino-uno-r3.html

[12] Bisquerra, R. (. (2019, Enero 20). *Métodos de investigación educativa. Guía práctica. Barcelona: CEAC*. Retrieved from Métodos de investigación educativa. Guía práctica. Barcelona: CEAC:

https://cvc.cervantes.es/ensenanza/biblioteca_ele/diccio_ele/diccionario/metodologiacuantitativa.htm

[13] Françoise Parot, R. D. (2019, Enero 20). *Metodología de la Investigación.* . Retrieved from Metodología de la Investigación. : https://bloglosariopsa.wordpress.com/2008/11/12/metodo-experimental/

[14] Miguel A. Arenas, J. M. (15 de Febrero de 2019). *Dise˜no y Construcci´on de un Guante de Datos.* Obtenido de Dise˜no y Construcci´on de un Guante de Datos: https://www.researchgate.net/profile/Jose_Palomares/publication/263353011_DISENO_Y_CONSTRUCCION_DE_UN_GUANTE_DE_DATOS_MEDIANTE_SENSORES_DE_FLEXIBILIDAD_Y_ACELEROMETRO/links/55fa4c6e08aeafc8ac35d634/DISENO-Y-CONSTRUCCION-DE-UN-GUANTE-DE-DATOS-MEDIANTE-SENS

[15] Real, C. N. (20 de Marzo de 2019). *Control de un Quadrotor mediante la plataforma Arduino .* Obtenido de https://upcommons.upc.edu/bitstream/handle/2099.1/8047/memoria.pdf

Recomendaciones

Para que los datos que se obtienen del montaje en la interfaz del computador sean más estables se recomienda soltar todo el Arduino UNO en una placa universal y que los conectores sean con unos pines de espada para disminuir el ruido y como lo ya dicho sean más estables.

Para la posición adecuada del montaje en el perfil se recomienda que se ubique lo más cercano posible en el centro de gravedad de una forma fija y que no sea afectado por las condiciones que se presentan fuera del perfil en especial el acelerómetro MPU-6050.

Los datos cada vez mejoraran dependiendo de qué tan bueno sean los materiales lo cuales utilizamos en el montaje como por ejemplo si no se llegase a querer utilizar un cable de datos se podría utilizar otro tipo de Arduino UNO el cual haga transferencia de datos vía Bluetooth.

ESPECIFICACIONES DEL CODIGO

Aquí se realiza la lectura del acelerómetro y del giroscopio.

ANEXO A

En esta primera parte del código muestra un resumen de lo que se va a realizar junto a las referencias de donde se obtuvo el código y las políticas que maneja.

```
#include <I2Cdev.h>


// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     2016-04-18 - Eliminated a potential infinite loop
//     2013-05-08 - added seamless Fastwire support
//                - added note about gyro calibration
//     2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//     2012-06-20 - improved FIFO overflow handling and simplified read process
//     2012-06-19 - completely rearranged DMP initialization code and simplification
//     2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//     2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//     2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                - add 3D math helper file to DMP6 example sketch
//                - add Euler output and Yaw/Pitch/Roll output formats
```

```
//     2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//     2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//     2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
```

```
#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* =========================================================================
  NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
  depends on the MPU-6050's INT pin being connected to the Arduino's
  external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
  digital I/O pin 2.
 * ========================================================================= */

/* =========================================================================
  NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
  when using Serial.write(buf, len). The Teapot output uses this method.
  The solution requires a modification to the Arduino USBAPI.h file, which
```

```
  is fortunately simple, but annoying. This will be fixed in the next IDE
  release. For more info, see these links:
  http://arduino.cc/forum/index.php/topic,109987.0.html
  http://code.google.com/p/arduino/issues/detail?id=958
 * ======================================================================== */



// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
//#define
OUTPUT_READABLE_YAWPITCHROLL////////////////////////////////////////////////////////
quitar comentario en caso de querer ver en arduino.
```

```
// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT//Este es el que toca comentar para para poderlo ver en arduino



#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
```

```
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]        quaternion container
VectorInt16 aa;         // [x, y, z]           accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]           gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]           world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]           gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };
```

Rutina de detección de interferencias

En esta parte del código indica si el pin de interrupción del MPU se ha elevado.

```
// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
   mpuInterrupt = true;
}
```

Configuración inicial

En esta parte del código se realiza la comunicación serial el cual en la nota de los primeros renglones especifica las condiciones de operación para un correcto funcionamiento como el voltaje en el cual está operando, la velocidad en el cual se van a mostrar los datos en la interfaz, se verifica la conexión, se configure el giróscopo y el comando inicial en Arduino que sería cualquier Tecla para que se ejecute el programa y por último la configuración de la salida en la ilustración de los datos.

```
// ================================================================

// ===                  INITIAL SETUP                  ===

// ================================================================


void setup() {

   // join I2C bus (I2Cdev library doesn't do this automatically)

   #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

      Wire.begin();

      Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation
difficulties

   #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE

      Fastwire::setup(400, true);

   #endif


   // initialize serial communication

   // (115200 chosen because it is required for Teapot Demo output, but it's

   // really up to you depending on your project)

   Serial.begin(115200);

   while (!Serial); // wait for Leonardo enumeration, others continue immediately


   // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or Arduino
```

```
    // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
```

```
mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

// make sure it worked (returns 0 if so)
if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
    Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
    Serial.println(F(")..."));
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
} else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
```

```
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}
```

Loop del programa principal

En la parte final de código se generan la calibración de datos los cuales están siendo tomados por el giróscopo y el acelerómetro en donde el código corre varias veces la toma de datos hasta que si los valores son constantes (lo cual es casi imposible por el ruido de los componentes lo cuales se utilizaron en el montaje, los valores solo varían en cuestiones de milésimas) se auto calibra para que no se generen errores a la hora de la toma de datos, también cuando se genera una fuerza G mayor de lo que soporta el MPU-6050 cuando vuelve a operar entre sus límites el vuelve a mostrar los valores sin que haya algún tipo de cambio en la operación.

```
// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        if (mpuInterrupt && fifoCount < packetSize) {
```

```
    // try to get out of the infinite loop
    fifoCount = mpu.getFIFOCount();
  }
  // other program behavior stuff here
  // .
  // .
  // .
  // if you are really paranoid you can frequently test in between other
  // stuff to see if mpuInterrupt is true, and if so, "break;" from the
  // while() loop to immediately process the MPU data
  // .
  // .
  // .
}

// reset interrupt flag and get INT_STATUS byte
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >= 1024) {
  // reset so we can continue cleanly
  mpu.resetFIFO();
  fifoCount = mpu.getFIFOCount();
  Serial.println(F("FIFO overflow!"));
```

```
// otherwise, check for DMP data ready interrupt (this should happen frequently)
} else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;

    #ifdef OUTPUT_READABLE_QUATERNION
        // display quaternion values in easy matrix form: w x y z
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        Serial.print("quat\t");
        Serial.print(q.w);
        Serial.print("\t");
        Serial.print(q.x);
        Serial.print("\t");
        Serial.print(q.y);
        Serial.print("\t");
        Serial.println(q.z);
    #endif

    #ifdef OUTPUT_READABLE_EULER
        // display Euler angles in degrees
```

```
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(ypr[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(ypr[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
```

```
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    Serial.print("aworld\t");
    Serial.print(aaWorld.x);
    Serial.print("\t");
    Serial.print(aaWorld.y);
    Serial.print("\t");
    Serial.println(aaWorld.z);
#endif

#ifdef OUTPUT_TEAPOT
    // display quaternion values in InvenSense Teapot demo format:
    teapotPacket[2] = fifoBuffer[0];
```

```
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

ANEXO B

// Demostración de la clase de dispositivo I2C (I2Cdev) Procesando el boceto para la salida MPU6050 DMP

// 6/20/2012 por Jeff Rowberg <jeff@rowberg.net>

// Las actualizaciones deberían (con suerte) estar siempre disponibles en https://github.com/jrowberg/i2cdevlib

//

// Registro de cambios:

// 2012-06-20 - lanzamiento inicial

/* ==

El código de la biblioteca del dispositivo I2Cdev se coloca bajo la licencia MIT

Derechos de autor (c) 2012 Jeff Rowberg

Por la presente se otorga el permiso, sin cargo, a cualquier persona que obtenga una copia.

de este software y los archivos de documentación asociados (el "Software"), para

en el Software sin restricción, incluyendo sin limitación los derechos

para usar, copiar, modificar, fusionar, publicar, distribuir, sublicenciar y / o vender

copias del Software, y para permitir que las personas a las que el Software está

Amueblado para ello, sujeto a las siguientes condiciones:

El aviso de copyright anterior y este aviso de permiso se incluirán en

Todas las copias o partes sustanciales del Software.

EL SOFTWARE SE PROPORCIONA "TAL CUAL", SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O

IMPLÍCITOS, INCLUIDOS, PERO NO LIMITADOS A LAS GARANTÍAS DE COMERCIABILIDAD,

APTITUD PARA UN PROPÓSITO PARTICULAR Y NO INCUMPLIMIENTO. EN NINGÚN CASO DEBE

LOS AUTORES O LOS TITULARES DEL DERECHO DE AUTOR SERÁN RESPONSABLES POR CUALQUIER RECLAMACIÓN, DAÑO O OTRO

LA RESPONSABILIDAD, YA SEA EN UNA ACCIÓN DE CONTRATO, POR CORTE O DE OTRA MANERA, DERIVADA DE,

```
FUERA DE O EN CONEXIÓN CON EL SOFTWARE O EL USO O OTRAS REPARACIONES EN
EL SOFTWARE.
===============================================
*/

import processing.serial.*;
import processing.opengl.*;
import toxi.geom.*;
import toxi.processing.*;



ToxiclibsSupport gfx;

Serial port;                  // El puerto serie
char[] teapotPacket = new char[14];  // InvenSense Teapot packet
int serialCount = 0;              // posición actual del byte del paquete
int synced = 0;
int interval = 0;

float[] q = new float[4];
Quaternion quat = new Quaternion(1, 0, 0, 0);

float[] gravity = new float[3];
float[] euler = new float[3];
float[] ypr = new float[3];

void setup() {
```

```
  // ventana gráfica de 300px usando la representación de OpenGL
  size(700, 700, OPENGL);
  gfx = new ToxiclibsSupport(this);

  // configurar luces y antialiasing
  lights();
  smooth();

  // display serial port list for debugging/clarity
  println(Serial.list());

  // obtener el primer puerto disponible (utilice este O el código de puerto específico a continuación)
  String portName = Serial.list()[0];

  // obtener un puerto serie específico (use este O el primer código disponible arriba)
  // String portName = "COM4";

  // abrir el puerto serie
  port = new Serial(this, "COM3", 115200);
  // escribir ();
  // enviar un solo carácter para activar DMP init / start
  // (esperado por MPU6050_DMP6 ejemplo del bosquejo de Arduino)
  port.write('r');

}

void draw() {
```

```
 if (millis() - interval > 1000) {
     // escribir ();
   // enviar un solo carácter para activar DMP init / start
   // (esperado por MPU6050_DMP6 ejemplo del bosquejo de Arduino)
// reenviar un solo carácter para activar DMP init / start
     // en caso de que la MPU se detenga / reinicie mientras el applet se está ejecutando
     port.write('r');
     interval = millis();
   }

   // black background
   background(0);

   // translate everything to the middle of the viewport
   pushMatrix();
   translate(width / 2, height / 2);



   // Rotación de 3 pasos desde ángulos de giro / inclinación / giro (bloqueo de cardán)
   // ... y otras rarezas que aún no he descubierto
   // rotateY (-ypr [0]);
   // rotateZ (-ypr [1]);
   // rotateX (-ypr [2]);
```

```
  // Toxiciclibs dirigen la rotación del ángulo / eje desde el cuaternión (¡NO hay bloqueo de cardán!)
  // (el orden de los ejes [1, 3, 2] y la inversión [-1, +1, +1] es una consecuencia de
  // diferentes supuestos de orientación del sistema de coordenadas entre Procesamiento
  // y InvenSense DMP)
  float[] axis = quat.toAxisAngle();
  rotate(axis[0], -axis[1], axis[3], axis[2]);

  // dibujar cuerpo principal en rojo
  fill(255, 0, 0, 200);
  box(10, 10, 200);

/// relleno (249, 250, 50);
// textSize (24);
// texto ("Giroscopio", (int) ancho / 6.0 - 300, -200);
// pushMatrix ();

  // dibujar la punta frontal en azul
  fill(0, 0, 255, 200);
  pushMatrix();
  translate(0, 0, -120);
  rotateX(PI/2);
  drawCylinder(0, 20, 20, 8);
  popMatrix();

  // dibujar alas y aleta de cola en verde
  fill(0, 255, 0, 200);
  beginShape(TRIANGLES);
```

```
  vertex(-100,  2, 30); vertex(0,  2, -80); vertex(100,  2, 30);  // wing top layer
  vertex(-100, -2, 30); vertex(0, -2, -80); vertex(100, -2, 30);  // wing bottom layer
  vertex(-2, 0, 98); vertex(-2, -30, 98); vertex(-2, 0, 70);  // tail left layer
  vertex( 2, 0, 98); vertex( 2, -30, 98); vertex( 2, 0, 70);  // tail right layer
  endShape();
  beginShape(QUADS);
  vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -80);
  vertex( 100, 2, 30); vertex( 100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -80);
  vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(100, -2,  30); vertex(100, 2,  30);
  vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2, -30, 98); vertex(-2, -30, 98);
  vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
  vertex(-2, -30, 98); vertex(2, -30, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
  endShape();

  popMatrix();

  textSize(18);
 String accStr = "(" + (float) gravity[0] + ", " + (float) gravity[1] + ", " + (float) gravity[2] + ")";
 String gyrStr = "(" + (float) euler[0]*180.0f/PI + ", " + (float) euler[1]*180.0f/PI + ", " + (float)
euler[2]*180.0f/PI + ")";
 String filStr = "(" + (float) ypr[0]*180.0f/PI + ", " + (float) ypr[1]*180.0f/PI + ", " + (float)
ypr[2]*180.0f/PI + ")";

 fill(249, 250, 50);
 text("gravity vector", (int) width/6.0 - 60, 25);
 text(accStr, (int) (width/6.0) - 40, 50);
```

```
  fill(56, 140, 206);
  text("Euler angles", (int) width/2.0 - 280, 75);
  text(accStr, (int) (width/2.0) - 280, 100);

  fill(83, 175, 93);
  text("yaw/pitch/roll angles", (int) (5.0*width/6.0) - 500, 125);
  text(filStr, (int) (5.0*width/6.0) - 500, 150);
delay(100);
}

void serialEvent(Serial port) {
   interval = millis();
   while (port.available() > 0) {
      int ch = port.read();

      if (synced == 0 && ch != '$') return;   // sincronización inicial: también se usa para volver a sincronizar / realinear si es necesario
      synced = 1;
      print ((char)ch);

      if ((serialCount == 1 && ch != 2)
         || (serialCount == 12 && ch != '\r')
         || (serialCount == 13 && ch != '\n'))  {
         serialCount = 0;
         synced = 0;
         return;
      }
```

```
if (serialCount > 0 || ch == '$') {
   teapotPacket[serialCount++] = (char)ch;
   if (serialCount == 14) {
      serialCount = 0; // reiniciar la posición del byte del paquete

      // obtener cuaternión del paquete de datos
      q[0] = ((teapotPacket[2] << 8) | teapotPacket[3]) / 16384.0f;
      q[1] = ((teapotPacket[4] << 8) | teapotPacket[5]) / 16384.0f;
      q[2] = ((teapotPacket[6] << 8) | teapotPacket[7]) / 16384.0f;
      q[3] = ((teapotPacket[8] << 8) | teapotPacket[9]) / 16384.0f;
      for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4 + q[i];

      // establecer nuestro quilion de toxilibs a nuevos datos
      quat.set(q[0], q[1], q[2], q[3]);

      //
      // los cálculos a continuación son innecesarios para la orientación utilizando solo toxilibs

      // calcular el vector de gravedad
      gravity[0] = 2 * (q[1]*q[3] - q[0]*q[2]);
      gravity[1] = 2 * (q[0]*q[1] + q[2]*q[3]);
      gravity[2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] + q[3]*q[3];

      // calcula los ángulos de Euler
      euler[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
      euler[1] = -asin(2*q[1]*q[3] + 2*q[0]*q[2]);
      euler[2] = atan2(2*q[2]*q[3] - 2*q[0]*q[1], 2*q[0]*q[0] + 2*q[3]*q[3] - 1);
```

```
            // calculate yaw/pitch/roll angles // calcular ángulos de guiñada / cabeceo / balanceo

            ypr[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);

            ypr[1] = atan(gravity[0] / sqrt(gravity[1]*gravity[1] + gravity[2]*gravity[2]));

            ypr[2] = atan(gravity[1] / sqrt(gravity[0]*gravity[0] + gravity[2]*gravity[2]));


             //output various components for debugging

            //println("q:\t" + round(q[0]*100.0f)/100.0f + "\t" + round(q[1]*100.0f)/100.0f + "\t" +
round(q[2]*100.0f)/100.0f + "\t" + round(q[3]*100.0f)/100.0f);

            //println("euler:\t" + euler[0]*180.0f/PI + "\t" + euler[1]*180.0f/PI + "\t" +
euler[2]*180.0f/PI);

            //println("ypr:\t" + ypr[0]*180.0f/PI + "\t" + ypr[1]*180.0f/PI + "\t" + ypr[2]*180.0f/PI);

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


 ////////////////////////////////////////////////////////////////////////////////////////////////
            //
          }
       }
    }
}


void drawCylinder(float topRadius, float bottomRadius, float tall, int sides) {

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

 //fill(249, 250, 50);

 //text("Gyroscope", (int) width/6.0 - 60, 25);

 //text(gyrStr, (int) (width/6.0) - 40, 50);

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
```

```
float angle = 0;
 float angleIncrement = TWO_PI / sides;
 beginShape(QUAD_STRIP);
 for (int i = 0; i < sides + 1; ++i) {
    vertex(topRadius*cos(angle), 0, topRadius*sin(angle));
    vertex(bottomRadius*cos(angle), tall, bottomRadius*sin(angle));
    angle += angleIncrement;
 }
 endShape();

 // Si no es un cono, dibuje la tapa circular
 if (topRadius != 0) {
    angle = 0;
    beginShape(TRIANGLE_FAN);

   // Punto central
    vertex(0, 0, 0);
    for (int i = 0; i < sides + 1; i++) {
       vertex(topRadius * cos(angle), 0, topRadius * sin(angle));
       angle += angleIncrement;
    }
    endShape();
 }

 // Si no es un cono, dibuje la tapa circular de fondo
 if (bottomRadius != 0) {
    angle = 0;
```

```
      beginShape(TRIANGLE_FAN);

      // Punto central
      vertex(0, tall, 0);
      for (int i = 0; i < sides + 1; i++) {
         vertex(bottomRadius * cos(angle), tall, bottomRadius * sin(angle));
         angle += angleIncrement;
      }
      endShape();
   }
}
```