Registro de puntos en el espacio a partir del seguimiento de la línea de vista de un individuo.

Ing. Gabriel Andrés Martín Velandia.
Mayo 2016.

Pontificia Universidad Javeriana.
Facultad de Ingeniería.
Maestría en Ingeniería Electrónica

Dedicatoria

Dedico este trabajo a mí esposa Gisela, mi bebé Juan Nicolás, su apoyo y compañía incondicional en todo momento me dieron la fuerza necesaria para culminar esta investigación que requirió de conocimiento, perseverancia y sacrificio.

Agradecimientos

Gracias a la Fuerza Aérea Colombiana por su apoyo económico y a mis comandantes por darme el tiempo justo para cumplir todas las responsabilidades que permitieron finalizar este trabajo.

De igual forma a mi Director Carlos y Codirector Iván por su orientación y paciencia que guiaron el camino de esta Investigación.

Resumen

Este trabajo describe el proceso de diseño e implementación del prototipo de un sistema que utiliza una plataforma con dos grados de libertad, para indicar la dirección de la línea de vista de un individuo en Tiempo Real, a través del sensado de movimiento de su casco, teniendo como referencia un sistema móvil. Para esto utiliza dos Unidades de Medición Inercial (IMU) de bajo costo con 9 grados de libertad, a las cuales se aplica fusión sensorial de los valores obtenidos de los acelerómetros, magnetómetros y giróscopos, a través de un filtro de Kalman Angular para cada IMU, el cual permite determinar la inclinación y orientación del casco y el sistema de referencia, para luego calcular su valor diferencial y con base a este, ajustar los ángulos de PITCH y YAW de una plataforma que sigue la línea de vista del usuario del casco en tiempo real. En el primer capítulo de este documento, se realiza una introducción, mostrando la importancia del prototipo a implementar, el estado del arte observando trabajos similares, y se determinan los objetivos. El siguiente capítulo explica los fundamentos teóricos requeridos para su implementación. Posterior a esto, en el capítulo 3 se determinan los principales parámetros de diseño y especificaciones generales del prototipo. Luego en el capítulo 4 se explica su implementación, y finalmente, en los capítulos 5 y 6, se describen las pruebas efectuadas y resultados obtenidos, concluyendo cuales fueron los logros alcanzados.

Tabla de Contenidos

Lista de tablas

Lista de figuras

Capítulo 1

Introducción e información general

Importancia y aplicaciones de la Línea de Vista de un Individuo (Line of Sight LoS)

El sistema visual humano tiene un campo visual máximo de 124º en la horizontal y de 120º en la vertical como se describe en la Figura 1. Dentro de este campo, existe un área denominada "campo binocular", en la cual los dos ojos pueden ver simultáneamente un mismo objeto, lo que permite obtener imágenes nítidas, con percepción de profundidad. Esta zona, es la que utiliza un individuo promedio, para ubicar los objetivos o blancos con los que desea interactuar. De igual forma, dentro de esta zona, podemos definir un vector perpendicular a los ojos, que inicia en medio de los mismos, al que se denomina línea de vista (Line of Sight - LoS), el cual determina exactamente el centro de atención de un individuo al observar un blanco, como se observa en color rojo en la Figura 1.

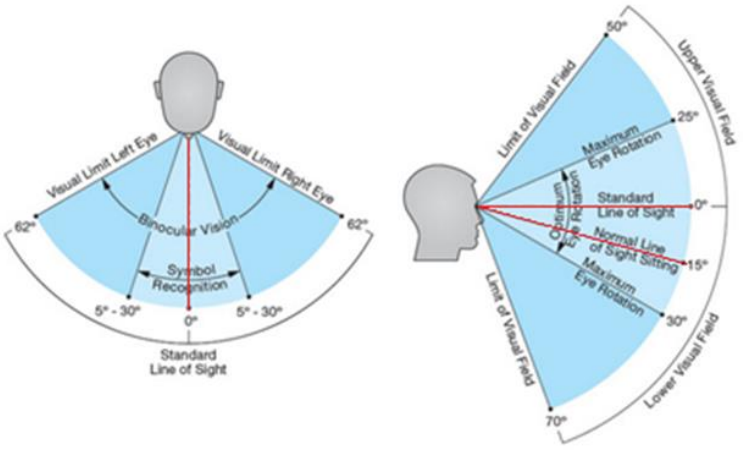


Figura 1. Características del campo visual humano [1].

Teniendo en cuenta lo anterior, la línea de vista de un individuo es una variable de alto interés, para diferentes aplicaciones civiles y militares, como lo pueden ser:

- La detección de la línea de vista de un piloto en una aeronave con respecto a la misma, para con base en ella, controlar el movimiento de un dispositivo electro-óptico de cualquier tipo, o de un arma; que fije su línea de vista o dirección de disparo, sobre el mismo punto en el espacio que está observando el piloto. Esto facilita al piloto la adquisición, seguimiento y análisis de blancos o la marcación de los mismos sobre un mapa de referencia [2].
- La detección de la línea de vista de un individuo que utilice un sistema de realidad virtual fijo a sus ojos, en el cual se cree un mundo virtual que cambie la imagen del campo visual del individuo a medida que cambia la dirección de su línea de vista [3].
- La detección de la línea de vista de un individuo que utilice algún dispositivo de imagen como lo puede ser un HMD (Helmet mounted display) o visor específico, para observar el mundo real y marcar sobre la imagen del mismo, lugares de interés que siempre aparezcan sobre el punto marcado como referencia, sin importar el cambio de la línea de vista del dispositivo óptico [4].

Las aplicaciones anteriormente mencionadas, tienen en común que la detección de la línea de vista de un individuo, se efectúa a través del seguimiento de la posición, ya sea de un casco o un visor, fijo a la cabeza del observador; en la literatura, se pueden encontrar sistemas similares a este, utilizados en el área aeronáutica, denominados como H.T.S.- Helmet Tracking System [5].

La mayoría de sistemas H.T.S, utilizan los ángulos de Euler, para determinar la dirección de la línea de vista del individuo que se monitorea, como se observa en la Figura 2. De esta forma, las variables de interés son los ángulos de: alabeo o Roll (ρ), Azimut o Yaw

(γ) y Elevación o Pitch (ϕ). Estos sistemas, limitan su operación teniendo en cuenta los movimientos típicos de un individuo en cada dirección, como lo son ±45º en elevación o pitch, ±180º en azimut o yaw, y ±30º en alabeo o Roll [6] y operan con una tasa promedio de muestreo de 100 Hz [7].

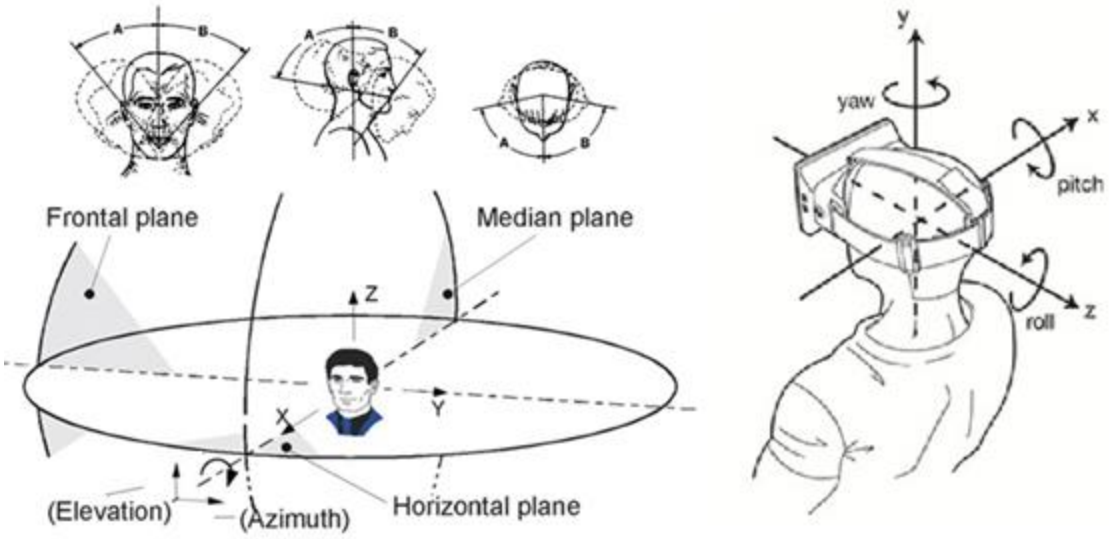


Figura 2. Modelado de movimientos de la cabeza humana.

Principales tecnologías de seguimiento de línea de vista de la cabeza humana.

A continuación se realizará una breve descripción de las tecnologías utilizadas para realizar el seguimiento de la línea de vista de un individuo, teniendo como referencia los sistemas H.T.S. que monitorean el movimiento de la cabeza de un piloto dentro de la cabina de una aeronave.

Los seguidores magnéticos fueron los primeros utilizados con este objetivo, por lo que podemos considerarlos, la tecnología más madura en este campo después de años de evolución; mostrando ventajas como el reducido tamaño de los sensores requeridos en el casco del individuo y el NO requerir una línea de vista constante entre el transmisor y el

receptor. Pero algunas de sus grandes limitaciones fueron; el reducido rango de los campos magnéticos, y su alta sensibilidad a la interferencia por objetos metálicos en la parte interna de la cabina, lo que obligaba a efectuar extensos y continuos procedimientos de mapeo para compensar estas distorsiones, siempre que se efectuaba cualquier cambio en la cabina [5].

De igual forma, se han utilizado sistemas con sensores ópticos, donde se utilizan cámaras o dispositivos de sensado electro ópticos de posición, montados fijos con respecto al sistema de referencia, observando un número determinado de LEDs infrarrojos ubicados en el casco del individuo a monitorear o viceversa. Para de esta forma, obtener un sistema de posicionamiento preciso, con mejores prestaciones que el obtenido con seguidores magnéticos, que no requiere un mapeo de cabina; pero con la limitación de requerir una línea de vista constante entre el transmisor y el receptor, con condiciones de iluminación limitadas para evitar interferencias, lo que es difícil de garantizar dentro del ambiente de una cabina, como los trabajos desarrollados por Bin Luo [6],William A. Hoff [8] y Kiyohide Satoh [9].

Una de las últimas tecnologías desarrolladas, ha implementado sistemas seguidores de movimiento, basados en sensores inerciales implementados en MEMS (Sistemas Microelectromecánicos) [10], este tipo de sensores no tiene limitaciones de rango, requerimientos de línea de vista y no tiene riesgos por interferencia magnética, acústica, óptica o de radio frecuencia. Como ventaja adicional se puede determinar la velocidad de muestreo de acuerdo a los requerimientos de la aplicación, con un amplio ancho de banda de movimiento y baja latencia. Pero uno de sus principales inconvenientes es que

cualquier sesgo o ruido leve del sensor inercial, cuando se integra en el tiempo, hace que las salidas de orientación y posición se desvíen poco a poco de forma acumulativa, por lo tanto este tipo de sensores, deben ser ocasionalmente corregidos por otros dispositivos, como se plantea en algunas investigaciones que utilizan sistemas híbridos para eliminar este inconveniente [5], [11], [12].

De igual forma, en muchas aplicaciones, es necesario conocer la línea de vista de un individuo, pero teniendo como referencia un mundo móvil con otros ángulos y movimientos posibles, como lo pueden ser un vehículo aéreo o terrestre como se presenta en la Figura 3, para los sistemas de referencia a y b. Para este caso es necesario detectar el movimiento de ambos sistemas de referencia, y relacionarlos de la manera adecuada.

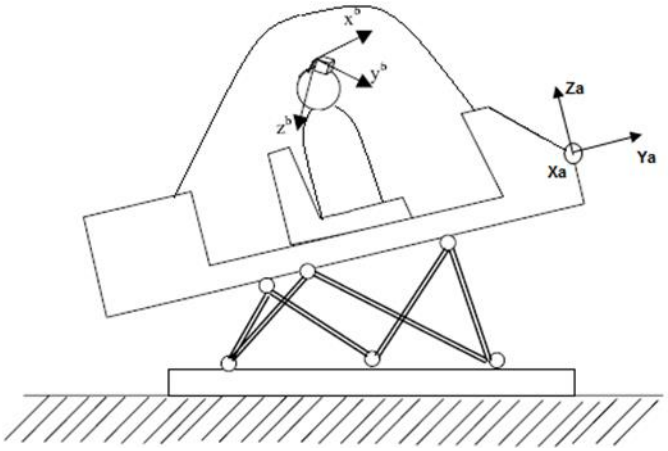


Figura 3. Sistema de Coordenadas teniendo Como Referencia un Mundo Móvil.

Cada aplicación tiene requerimientos y características específicas, que deben tener en cuenta, las características del movimiento de la plataforma móvil y del individuo monitoreado, para con base en ello, determinar los requerimientos de resolución del movimiento, velocidad de muestreo requerida para cada sistema de referencia y el tipo de sensores a utilizar.

Sistemas de este tipo, son fabricados por pocas empresas reconocidas alrededor del mundo, e integrados con otros sistemas altamente complejos, en las áreas de la aeronáutica y realidad virtual principalmente, lo que ha ocasionado, elevados costos en su comercialización y poca o ninguna aplicación en los desarrollos locales. Esto justifica la necesidad de iniciar procesos de investigación como el desarrollado en este documento, con el fin de obtener el conocimiento, e iniciar el diseño e implementación de prototipos funcionales de sistemas de esta clase, que puedan ser integrados en aeronaves o sistemas de seguridad principalmente, desarrollados a nivel local.

Teniendo en cuenta lo anterior, con el desarrollo de esta investigación, se pretende diseñar e implementar el prototipo de un sistema, que determine la dirección de la línea de vista de un individuo que solo efectúa movimientos de rotación de su cabeza, a través del sensado de movimiento de su casco, teniendo como referencia un sistema móvil; y evaluar su operación, utilizando sistemas ópticos para comparar los resultados. El primero de ellos estará fija al casco, indicando la línea de vista del individuo; y el otro dispositivo óptico con dos grados de libertad (yaw y picth) estará fija a la referencia, efectuando los mismos movimientos del casco del individuo, buscando mantener la línea de vista de ambos sistemas, sobre el mismo punto en el espacio.

Para comprobar la correcta operación del sistema, se seguirá un protocolo de pruebas específico, dentro de un entorno real, que simule los movimientos posibles de un individuo dentro de una cabina de una aeronave. Para esto, se realizará un montaje determinado, el cual utilizará un sistema VICON como referencia, para garantizar la precisión de los ángulos a los cuales es sometido el sistema.

Objetivo General.

Diseñar e implementar el prototipo de un sistema que utilice una cámara de video con dos grados de libertad, para indicar la dirección de la línea de vista de un individuo en Tiempo Real, a través del sensado de movimiento de su casco, teniendo como referencia un sistema móvil.

Objetivos Específicos.

- Diseñar e implementar el sistema de sensado, que permita determinar la línea de vista de un individuo que solo efectúa movimientos de rotación de su cabeza, a través del sensado de movimiento de su casco, teniendo como referencia un sistema móvil.
- Desarrollar un algoritmo que procese los datos tomados de los sensores en tiempo real, generando como resultado los ángulos de inclinación en pitch, roll y yaw, de la línea de vista del individuo tomando como referencia un sistema móvil.
- Controlar el desplazamiento de una cámara con dos grados de libertad (pitch y yaw) fija a la referencia, para que registre el mismo punto en el espacio que indica la línea de vista del individuo que se está monitoreando.
- Verificar el rendimiento del sistema diseñado, siguiendo un protocolo de pruebas determinado, que utilice un brazo manipulador o sistema de referencia fiable, para simular los movimientos probables, y comparar las diferencias obtenidas entre la línea de vista del usuario del casco y el replicado por la sistema fijo al sistema de referencia.

Capítulo 2

Fundamentos Teóricos y Conceptuales del Prototipo

Unidades de Medición Inercial IMU

En términos generales una Unidad de Medición Inercial IMU es un dispositivo que determina la posición y orientación en el espacio de la plataforma que la transporta. Nació como respuesta a una necesidad militar, utilizándola para aplicaciones como el guiado de misiles, la orientación de submarinos, UAV's, aviones de combate entre otras.

Las dimensiones de las IMU varían de acuerdo al principio físico y técnica de construcción de los sensores que la componen. Los primeros prototipos de este tipo de dispositivos, utilizaron una cantidad limitada de sensores que principalmente eran giróscopos de momento angular, mostrando alta confiabilidad, gran precisión y un gran intervalo de medición, incluso sin requerir alimentación eléctrica, pero su gran tamaño y peso limitaba su aplicación por problemas de portabilidad.

Durante la última década, gracias a los avances principalmente de la micro y nanotecnología, se han desarrollado nuevos sensores y dispositivos que pueden ubicarse de forma simultánea dentro de un chip, los cuales se denominan Sistemas Micro Electro Mecánicos. Estos tienen grandes ventajas de costo y portabilidad, permitiendo implementar IMU's de alta precisión, que integran en un solo dispositivos acelerómetros, giróscopos y magnetómetros de tres ejes, los cuales gracias al desarrollo de los sistemas de micro procesamiento de datos, permiten realizar estimaciones precisas de posición, orientación, velocidad lineal y angular utilizando procesos complejos de filtrado y fusión sensorial de forma rápida y eficiente.

A continuación se explicarán todos los sistemas, sensores, procedimientos y conceptos utilizados y requeridos para solucionar el problema propuesto.

Sistemas MEMS

Son Sistemas Micro Electro Mecánicos, que pueden definirse de forma general como elementos mecánicos y electromecánicos miniaturizados que se realizan con técnicas de micro-fabricación, con dimensiones que pueden variar desde fracciones de una micra, hasta varios milímetros.

Estos sistemas pueden contener desde estructuras simples sin elementos móviles, hasta sistemas electromecánicos altamente complejos con múltiples elementos móviles controlados por microelectrónica integrada. Para su producción se utilizan las mismas técnicas de fabricación por lotes que se utiliza en la industria de los circuitos integrados obteniendo bajos costos de producción.

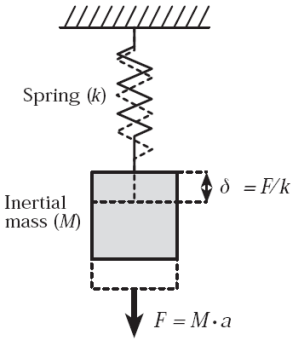
Dentro de los MEMS, los elementos más notables que han tenido un acelerado desarrollo en los últimos años, son los micro-sensores y micro-actuadores que tiene como función convertir energía de una forma a otra, estos son aplicables en diferentes ramas para detectar temperaturas, fuerzas de inercia, presión, variaciones químicas, radiación, campos magnéticos entre otros.

Pero es necesario precisar que el verdadero potencial de los MEMS está en la combinación de estos micro-sensores y micro-actuadores en un sustrato de silicio, con los circuitos integrados, permitiendo implementar sistemas de control automatizados altamente complejos con la fiabilidad y confiabilidad adecuada.

Para el desarrollo de este trabajo se utilizan sistemas MEMS del tipo acelerómetros, giróscopos y magnetómetros integrados en una Unidad de Medición Inercial (IMU) que hace uso de técnicas de filtrado y fusión sensorial para determinar la inclinación y orientación de un objeto con respecto a una referencia.

Acelerómetros MEMS

Dentro de los sistemas MEMS podemos encontrar un sistema específicamente diseñado para medir aceleraciones, el cual principalmente utiliza sistemas masa – resorte, que generan cambios piezoeléctricos, de capacitancia o en la convección térmica natural del aire, para de esta forma generar una señal eléctrica dependiente de la aceleración a la cual es sometida la masa, como se observa en la Figura 4 parte superior. Este mismo sistema puede utilizar diferentes tipos de estructuras, las cuales son implementadas dentro de un MEMS como se observa en la Figura 4 parte inferior.



Resonant frequency:

$$f_r = \frac{1}{2\pi}\sqrt{\frac{k}{M}}$$

Noise equivalent acceleration:

$$a_{noise} = \sqrt{\frac{8\pi K_B T f_r B}{Q M}} \; ; \; B < f_r$$

K_B = Boltzmann constant
T = Temperature
B = Bandwidth
Q = Quality factor

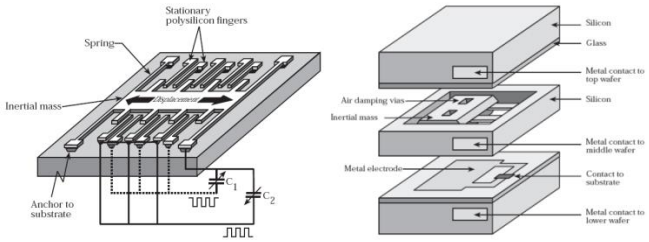


Figura 4. Parte superior. Acelerómetro masa - resorte, frecuencia de resonancia y ruido conocidos. Parte inferior. Estructuras típicas de acelerómetros MEMS capacitivos y pizoresistivos [13].

Pero dentro de este tipo de estructuras, siempre se genera una cantidad determinada de ruido, por lo general causada por las fluctuaciones térmicas y mecánicas al interior del sensor, siendo este nivel de ruido diferente para cada sensor; en la literatura estos tipos de ruido se conocen como ruido blanco y caminata aleatoria [14].

Teniendo en cuenta lo anterior, para tener alta precisión, es necesario utilizar técnicas de filtrado o fusión sensorial que utilicen un determinado número de mediciones para estimar un valor de aceleración, la precisión de cada tipo de acelerómetro se incrementa al igual que su costo, hasta llegar a acelerómetros de gama alta, que son principalmente utilizados en aplicaciones aeroespaciales.

Entre las principales aplicaciones de este tipo de sensores, tenemos la detección de presencia de movimiento, la navegación inercial a través de la estimación de la velocidad y posición de la aceleración, la intensidad de la vibración de un equipo o de la tierra y la medición de inclinación de un objeto.

En la mayoría de aplicaciones, los acelerómetros se combinan para crear acelerómetros de múltiples ejes; para este trabajo se utiliza un acelerómetro de tres ejes perpendiculares (X, Y, Z), tomando la aceleración encontrada sobre cada eje, para determinar la inclinación en pitch y en roll del objeto monitoreado.

Medición de Inclinación con un Acelerómetro de Tres Ejes

Para explicar el proceso de medición de inclinación utilizando únicamente los acelerómetros, vamos a definir dos sistemas de referencia de tres dimensiones cada uno. El primero lo llamaremos el sistema de referencia móvil (X_m, Y_m, Z_m) y el segundo el sistema de referencia base (X_b, Y_b, Z_b). Las rotaciones alrededor del eje X_b se denominarán rotaciones en Roll y su medida será la cantidad de grados ρ entre el eje base Y_b y el eje móvil Y_m, de igual forma definimos que las rotaciones alrededor del eje Y_b, las cuales se denominan rotaciones en Pitch y su medida es la cantidad de grados ϕ entre el eje base X_b y el eje móvil X_m como se observa en la Figura 5.

Si observamos la Figura 5, utilizando la trigonometría y las propiedades del acelerómetro de tres ejes perpendiculares del cual obtenemos las mediciones $A_{x,}, A_y, A_z$, podemos definir los ángulos de Pitch *(ϕ)* y Roll *(ρ)* como se expresa en las Ecuaciones (1) y (2).

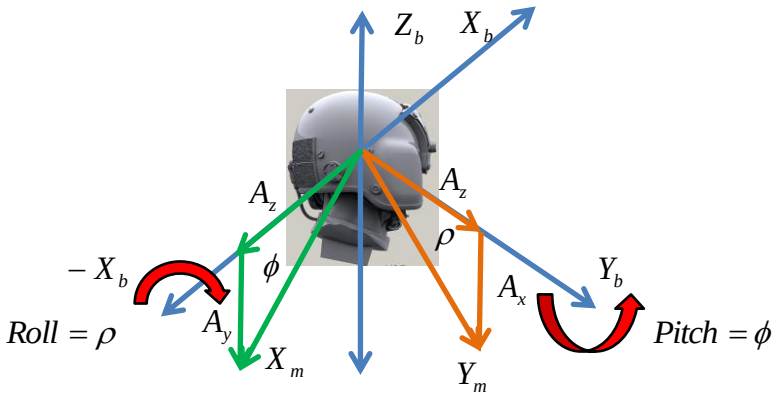


Figura 5. Sistemas de Coordenadas para calcular pitch (ϕ) y rol l (ρ) utilizando acelerómetros.

$$\emptyset = \boldsymbol{asin}\left(\frac{\boldsymbol{A_y}}{\sqrt{\boldsymbol{A_y^2 + A_z^2}}}\right) \qquad (1)$$

$$\rho = \boldsymbol{asin}\left(\frac{\boldsymbol{A_x}}{\sqrt{\boldsymbol{A_x^2 + A_z^2}}}\right) \quad (2)$$

Pero es necesario aclarar que esta técnica, solo se puede utilizar cuando el sistema está en estado estacionario, siendo sometido únicamente a la aceleración de la gravedad. Si el sistema está sometido a aceleraciones externas, es necesario utilizar otro tipo de sensor adicional que permita realizar fusión sensorial para reducir el error.

Giróscopos MEMS

De forma general un giróscopo es un sensor que mide la tasa de rotación de un objeto. Existen de diferentes tipos, como lo pueden ser los giróscopos de momento rotatorio, los giróscopos de anillo laser; o los implementados en sistemas MEMS, que utilizan fundamentalmente el efecto Coriolis para determinar las velocidades angulares, el efecto Coriolis es una consecuencia directa del movimiento de un cuerpo en un marco de referencia rotatorio, la aceleración de Coriolis es la que debe ser aplicada a un móvil que se encuentra en una superficie rotatoria para que este mantenga su orientación [13].

Existen diferentes arquitecturas para crear una señal eléctrica dependiente de los efectos producidos por las Fuerzas de Coriolis, En la Figura 6 podemos ver algunas estructuras que reciben en nombre de la empresa donde fueron creadas [13].

Este tipo de sensores es utilizado para determinar la orientación de un objeto, teniendo como ventaja, que a diferencia de los acelerómetros, que el objeto puede ser sometido a la aceleración de la gravedad y aceleraciones externas de forma simultáneas, generando una señal confiable de velocidad de rotación, la cual puede ser fácilmente integrada en el tiempo para encontrar la inclinación.

Pero en el proceso de integración de cualquier tipo de giróscopo, es inevitable la generación de error derivativo o Drift, el cual es un error acumulativo no constante que se incrementa con el tiempo, las técnicas de corrección de este tipo de error pueden utilizar la re calibración constante del sistema al pasar por cero en aplicaciones que puedan utilizar esta característica, o la combinación con otro tipo de sensores como lo pueden ser los acelerómetros para utilizar las potencialidades de cada sistema y obtener una señal con menor error.

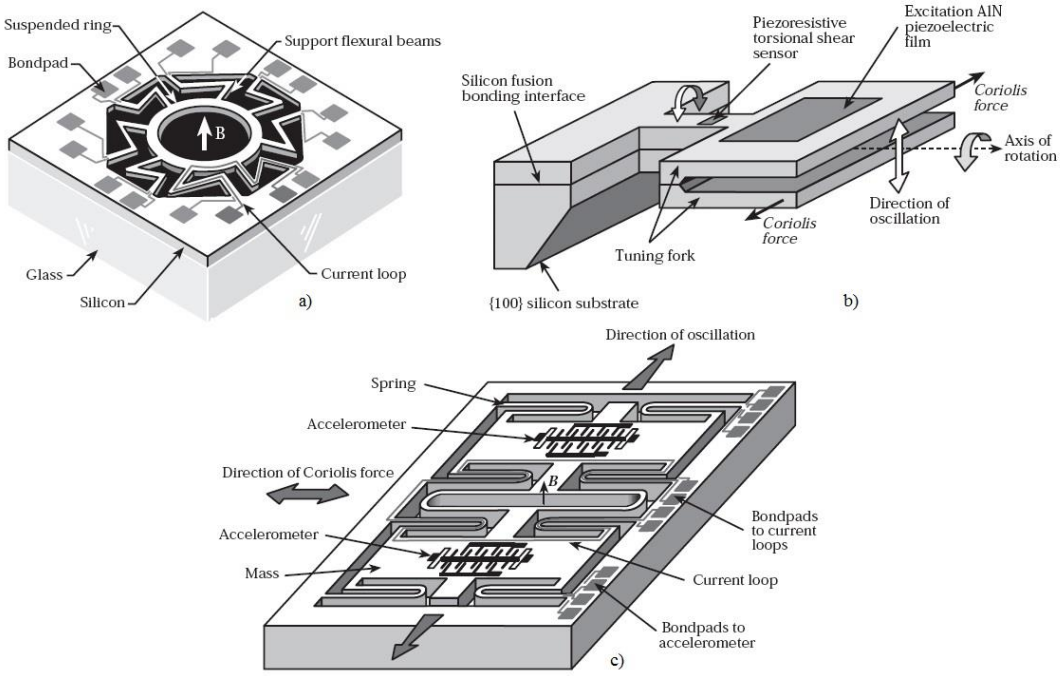


Figura 6. MEMS de Giróscopos: a) Silicon Sensing System, b) Daimler Benz, c) Robert Bosch [13].

Otro de los inconvenientes de este tipo de sensores, como en cualquier dispositivo, es el ruido (ruido blanco y caminata aleatoria [14]). Sin embargo, la mayoría las aplicaciones de los giróscopos, implican la detección de movimientos grandes y como resultado el ruido es fácil de distinguir o ignorar.

En la mayoría de aplicaciones, los giróscopos al igual que los acelerómetros, se combinan para crear giróscopos de múltiples ejes; para este trabajo se utiliza un giróscopo de tres ejes perpendiculares (X, Y, Z), tomando la velocidad de rotación angular encontrada sobre cada eje, para determinar la inclinación en pitch, roll y yaw del objeto monitoreado.

Medición de Inclinación y Orientación con un Giróscopo de Tres Ejes

Si conocemos la posición inicial de una plataforma $(\rho_x|_{t=0}, \emptyset_y|_{t=0}, \gamma_z|_{t=0})$ y se conoce la velocidad de cambio de los ángulos de Euler $(\dot{\rho}_x, \dot{\emptyset}_y, \dot{\gamma}_z)$, se puede determinar la inclinación y la orientación a través de la integración en el tiempo de estas señales, como se muestra en las ecuaciones (3), (4) y (5).

$$\rho_x(t) = \int_0^t \dot{\rho}_x \, d\tau + \rho_x|_{t=0} \qquad (3)$$

$$\emptyset_y(t) = \int_0^t \dot{\emptyset}_y \, d\tau + \emptyset_y|_{t=0} \qquad (4)$$

$$\gamma_z(t) = \int_0^t \dot{\gamma}_z \, d\tau + \gamma_z|_{t=0} \qquad (5)$$

Como se mencionó anteriormente la señal resultante de estas 3 ecuaciones, tiene menor ruido que las obtenidas de los acelerómetros, pero el drift, genera un error acumulativo que crece con el tiempo que debe ser corregido para reducir el error final de la estimación.

Magnetómetros MEMS

Los magnetómetros son sensores de campo magnético, que normalmente utilizan el campo magnético terrestre para determinar la orientación o heading de un objeto, la cual para esta aplicación en específico, corresponde al ángulo de Yaw (γ). Los sistemas MEMS, principalmente utilizan el efecto Hall, el cual determina que una

lámina metálica que sea sometida a un campo magnético perpendicular lo suficientemente fuerte, genera un voltaje en los extremos de la misma como se observa en la Figura 7.

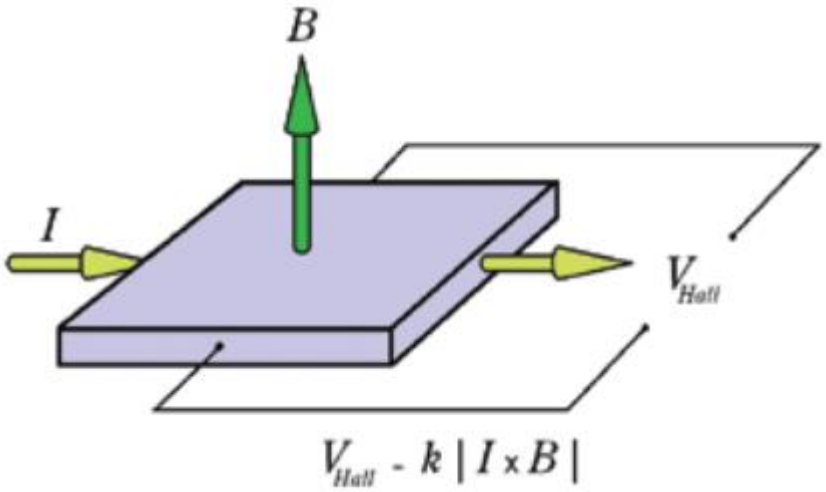


Figura 7. Principio del Efecto Hall.

Para aplicaciones prácticas, este tipo de sensores debe ser utilizado en configuraciones de dos o tres ejes perpendiculares (X/Y y X/Y/Z), acompañado de un sistema de medición de inclinación, el cual estime la orientación (γ) dependiendo de la inclinación (ϕ y ρ) del dispositivo.

Los magnetómetros utilizados como brújula, requieren efectuar una calibración, en la cual se indique las características del campo magnético en una región particular del mundo y la afectación de diferentes tipos de perturbaciones, las cuales pueden ser causadas por materiales circundantes *(Soft Iron)* o campos magnéticos localizados *(Hard Iron)*, generados por fuentes externas como lo pueden ser imanes, materiales ferrosos, bobinas etc.

Por último, como en cualquier tipo de sensor, es necesario tener en cuenta que en la señal resultante siempre genera una cantidad determinada de ruido, por lo general causada por las fluctuaciones térmicas y mecánicas al interior del sensor, siendo este nivel de ruido diferente para cada sensor o eje. En la literatura estos tipos de ruido se conocen como ruido blanco y caminata aleatoria [13].

Medición de Orientación o Heading con Magnetómetro y Acelerómetro de Tres Ejes

Para realizar este cálculo se utiliza el algoritmo de la brújula electrónica compensada por inclinación [15], para esta aplicación se utiliza el estándar industrial "NED" (North, East, Down) El eje X del sistema es la dirección de indicación de la brújula (Cero grados), el eje Y apunta a la derecha y el eje Z hacia abajo.

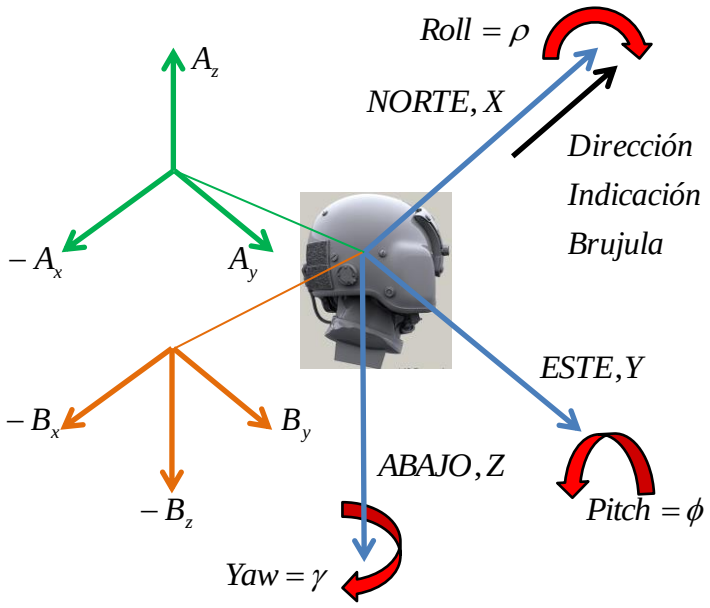


Figura 8. Sistemas de Coordenadas Para Calculo de Orientación o Heading.

Teniendo en cuenta lo anterior, cualquier orientación puede ser representada como el resultado de rotaciones en Pitch *(ϕ)*, Roll *(ρ)* y Yaw *(γ)* aplicadas a la posición inicial en la dirección de indicación de la brújula (NORTE, X), de esta forma como se observa en la

Figura 9, el acelerómetro A_T y el magnetómetro B_T inician sus lecturas con los valores observados en las ecuaciones *(6)* y *(7)*.

La aceleración de la gravedad es 9.81 m/s². B es el campo magnético terrestre, el cual varía sobre la superficie de la tierra de un mínimo de 22μT sobre Suramérica hasta un máximo de 67μT en el sur de Australia. δ Es el ángulo de inclinación del campo magnético terrestre medido hacia abajo de la horizontal, el cual varía sobre la superficie de la tierra de -90º en el polo sur magnético, pasando por 0º cerca de la línea ecuatorial, hasta 90º en el polo norte magnético.

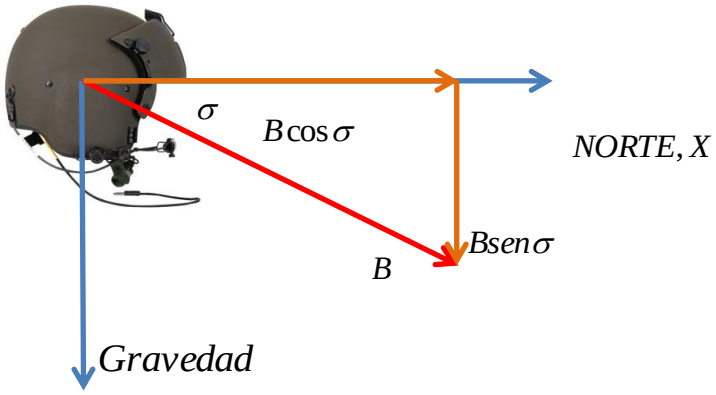


Figura 9. Vector Gravitacional y de Campo Magnético.

$$A_T = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \qquad (6)$$

$$B_T = B \begin{bmatrix} cos\delta \\ 0 \\ sen\delta \end{bmatrix} \qquad (7)$$

Las aceleraciones del objeto observado A_T y las mediciones del magnetómetro B_T, medidas después de efectuar tres rotaciones $R_Z(\gamma)$, luego en $R_Y(\emptyset)$ y finalmente en $R_x(\rho)$, se describen con las ecuaciones *(8)* y *(9)* asumiendo que el objeto observado está en estado estacionario únicamente sometido a la aceleración de la gravedad e

ignorando las perturbaciones por *Hard iron* y *Soft iron* en el campo magnético terrestre, las cuales se eliminan después de calibrar la brújula.

$$A_T = R_X(\rho)R_Y(\emptyset)R_Z(\gamma)A_T = R_X(\rho)R_Y(\emptyset)R_Z(\gamma)\begin{bmatrix}0\\0\\g\end{bmatrix} \quad (8)$$

$$B_T = R_X(\rho)R_Y(\emptyset)R_Z(\gamma)B_T = R_X(\rho)R_Y(\emptyset)R_Z(\gamma)B\begin{bmatrix}cos\sigma\\0\\sen\sigma\end{bmatrix} \quad (9)$$

Las tres matrices de rotación mencionadas en las ecuaciones *(8)* y *(9)* son las matrices de rotación que se observan en las ecuaciones *(10)*, *(11)* y *(12)*.

$$R_X(\rho) = \begin{bmatrix}1 & 0 & 0\\0 & cos\rho & sen\rho\\0 & -sen\rho & cos\rho\rho\end{bmatrix} \quad (10)$$

$$R_Y(\emptyset) = \begin{bmatrix}cos\emptyset & 0 & -sen\emptyset\\0 & 1 & 0\\sen\emptyset & 0 & cos\emptyset\end{bmatrix} \quad (11)$$

$$R_Z(\gamma) = \begin{bmatrix}cos\gamma & sen\gamma & 0\\-sen\gamma & cos\gamma & 0\\0 & 0 & 1\end{bmatrix} \quad (12)$$

Ahora para modelar los efectos del *Hard-iron* debemos adicionar un vector magnético *V*, el cual rota con el objeto observado sin importar la orientación complementando la ecuación *(9)*, obteniendo la ecuación *(13)*, donde V_X, V_Y y V_Z son los componentes del vector de *Hard-Iron*.

$$B_T = R_X(\rho)R_Y(\emptyset)R_Z(\gamma)B\begin{bmatrix}cos\sigma\\0\\sen\sigma\end{bmatrix} + V = R_X(\rho)R_Y(\emptyset)R_Z(\gamma)B\begin{bmatrix}cos\sigma\\0\\sen\sigma\end{bmatrix} + \begin{bmatrix}V_X\\V_Y\\V_Z\end{bmatrix} \quad (13)$$

Teniendo en cuenta lo anterior, el algoritmo de la brújula electrónica compensada por inclinación [15], primero calcula los ángulos de Pitch *(ϕ)* y Roll *(ρ)* de las lecturas de los acelerómetros pre multiplicando la ecuación *(8)* por las matrices inversas de rotaciones en Roll $R_X(\rho)$ y Pitch $R_Y(\emptyset)$, obteniendo la ecuación *(14)*.

$$R_Y(-\emptyset)R_X(-\rho)A_T = R_Y(-\emptyset)R_X(-\rho)\begin{bmatrix}A_X\\A_Y\\A_Z\end{bmatrix} = R_Z(\gamma)\begin{bmatrix}0\\0\\g\end{bmatrix} = \begin{bmatrix}0\\0\\g\end{bmatrix} \quad (14)$$

Expandiendo la ecuación *(14)*, obtenemos las ecuaciones *(15)* y *(16)*.

$$\begin{bmatrix} cos\emptyset & 0 & sen\emptyset \\ 0 & 1 & 0 \\ -sen\emptyset & 0 & cos\emptyset \end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & cos\rho & -sen\rho\\0 & sen\rho & cos\rho\end{bmatrix}\begin{bmatrix}A_X\\A_Y\\A_Z\end{bmatrix} = \begin{bmatrix}0\\0\\g\end{bmatrix} \quad (15)$$

$$\begin{bmatrix} cos\emptyset & sen\emptyset.sen\rho & sen\emptyset.cos\rho \\ 0 & cos\rho & -sen\rho \\ -sen\emptyset & cos\emptyset.sen\rho & cos\emptyset.cos\rho \end{bmatrix}\begin{bmatrix}A_X\\A_Y\\A_Z\end{bmatrix} = \begin{bmatrix}0\\0\\g\end{bmatrix} \quad (16)$$

Ahora, tomando la componente en Y de la ecuación *(16)*, podemos definir el ángulo de Roll *(ρ)* obteniendo la ecuación ***(18)***.

$$A_Y cos\rho - A_Z sen\rho = 0 \quad (17)$$

$$\boldsymbol{\rho = tan^{-1}\left(\frac{A_Y}{A_Z}\right)} \quad \boldsymbol{(18)}$$

De igual forma, tomando la componente en X de la ecuación *(16)*, podemos definir el ángulo de Pitch *(ϕ)* obteniendo la ecuación ***(20)***.

$$A_X cos\emptyset + A_Y sen\emptyset.sen\rho + A_Z sen\emptyset.cos\rho = 0 \quad (19)$$

$$\boldsymbol{\emptyset = tan^{-1}\left(\frac{-A_X}{A_Y sin\rho + A_Z cos\rho}\right)} \quad \boldsymbol{(20)}$$

De esta forma, conociendo los ángulos de Pitch *(ϕ)* y Roll *(ρ)* del acelerómetro, las lecturas del magnetómetro, pueden ser anti rotadas para la orientación del objeto utilizando la ecuación *(13)*, como se observa en las ecuaciones *(21)* a *(24)*.

$$R_Z(\gamma)\begin{bmatrix}Bcos\sigma\\0\\Bsen\sigma\end{bmatrix} = \begin{bmatrix} cos\gamma & sen\gamma & 0 \\ -sen\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix}Bcos\sigma\\0\\Bsen\sigma\end{bmatrix} = R_Y(-\emptyset)R_X(-\rho)(B_T - V) \quad (21)$$

$$=> \begin{bmatrix} cos\gamma.Bcos\sigma \\ -sen\gamma.Bcos\sigma \\ Bsen\sigma \end{bmatrix} = \begin{bmatrix} cos\emptyset & 0 & sen\emptyset \\ 0 & 1 & 0 \\ -sen\emptyset & 0 & cos\emptyset \end{bmatrix}\begin{bmatrix}1 & 0 & 0\\0 & cos\rho & -sen\rho\\0 & sen\rho & cos\rho\end{bmatrix}\begin{bmatrix}B_X - V_X\\B_Y - V_Y\\B_Z - V_Z\end{bmatrix} \quad (22)$$

$$= \begin{bmatrix} cos\emptyset & sen\emptyset.sen\rho & sen\emptyset.cos\rho \\ 0 & cos\rho & -sen\rho \\ -sen\emptyset & cos\emptyset.sen\rho & cos\emptyset.cos\rho \end{bmatrix}\begin{bmatrix}B_X - V_X\\B_Y - V_Y\\B_Z - V_Z\end{bmatrix} \quad (23)$$

$$= \begin{bmatrix} (B_X - V_X)cos\emptyset + (B_Y - V_Y)sen\emptyset.sen\rho + (B_Z - V_Z)sen\emptyset.cos\rho \\ (B_Y - V_Y)cos\rho - (B_Z - V_Z)sin\rho \\ -(B_X - V_X)sen\emptyset + (B_Y - V_Y)cos\emptyset.sen\rho + (B_Z - V_Z)cos\emptyset.cos\rho \end{bmatrix} = \begin{bmatrix} B_{fx} \\ B_{fy} \\ B_{fz} \end{bmatrix} \quad (24)$$

Los elementos B_{fx}, B_{fy} y B_{fz}del vector observado en la ecuación *(24)*, representan los componentes del magnetómetro después de corregir la interferencia de *Hard-Iron* y después de anti rotarlos a un plano nivelado cuando $\emptyset = \rho = 0$.

Tomando los componentes en X y en Y de la ecuación *(24)*, podemos encontrar el ángulo de orientación en Yaw *(γ)* como se observa en las ecuaciones *(25)* a ***(28)***.

$$cos\gamma.Bcos\sigma = B_{fx} \quad (25)$$

$$sen\gamma.Bcos\sigma = -B_{fy} \quad (26)$$

$$\gamma = tan^{-1}\left(\frac{-B_{fy}}{B_{fx}}\right) \quad (27)$$

$$\boldsymbol{\gamma = \tan^{-1}\left(\frac{(B_Z - V_Z)sin\rho - (B_Y - V_Y)cos\rho}{(B_X - V_X)cos\emptyset + (B_Y - V_Y)sen\emptyset.sen\rho + (B_Z - V_Z)sen\emptyset.cos\rho}\right)} \quad \boldsymbol{(28)}$$

Fusión Sensorial de una I.M.U.

La fusión sensorial podemos definirla como un proceso de negociación que utiliza la asociación, correlación y combinación de datos e información de múltiples fuentes, con el fin de reducir el error en la estimación de variables. Este proceso se caracteriza por mejorar de forma continua sus estimaciones, evaluando la necesidad de fuentes adicionales o modificaciones del proceso en sí, para lograr mejores resultados [16].

En el caso específico de este trabajo, se realiza una fusión sensorial entre los acelerómetros y giróscopos para determinar la inclinación de un casco en Pitch *(ϕ)* y Roll *(ρ);* y entre los giróscopos y el magnetómetros para determinar la orientación o heading en Yaw *(γ)*.

Para esto, empezaremos definiendo las variables de interés, como se observa en la Figura 10, las variables conocidas de forma directa utilizando los acelerómetros de la IMU, son las aceleraciones $(A_x,\ A_y\ A_z)$, de igual forma utilizando los giróscopos obtenemos las velocidades de rotación $(w_x,\ w_y\ w_z)$. Teniendo en cuenta estas variables conocidas, se puede estimar la inclinación en Pitch ϕ_y y Roll ρ_x, la orientación en Yaw γ_x, la posición *(x, y, z)* y las velocidades lineales $(v_x,\ v_y\ v_z)$.

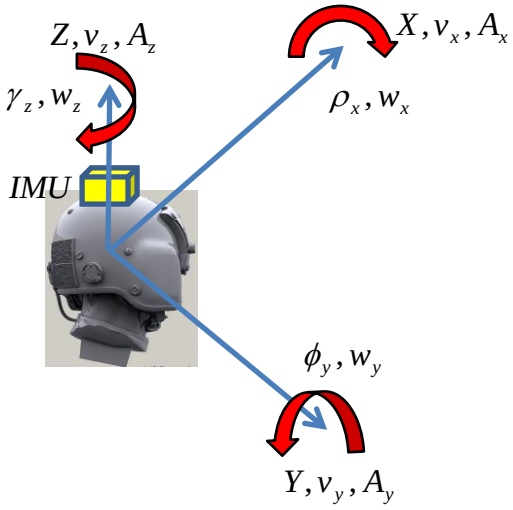


Figura 10. Variables de Interés.

Teniendo en cuenta lo anterior, para estimar las velocidades lineales $(v_x,\ v_y\ v_z)$ y la posición de la plataforma *(x, y, z);* se deben realizar los procesos de integración que se observan en las ecuaciones *(29)(30)* y *(31)*, las cuales generan un error acumulativo que se incrementa con el tiempo; por este motivo en la aplicaciones reales que no son objeto de este trabajo, esta medición es fusionada con información de posición suministrada por un DGPS *(Differential Global Positioning System)* que genera información más precisa pero a una frecuencia más baja, para de esta forma dar estimaciones de velocidad y posición con menor error.

$$v_x = \int A_x\, dt\,, x = \int v_x\, dt \qquad (29)$$

$$v_y = \int A_y\, dt\,, y = \int v_y\, dt \qquad (30)$$

$$v_z = \int A_z\, dt\,, z = \int v_z\, dt \qquad (31)$$

Las variables restantes, inclinación en Pitch ϕ_y, Roll ρ_x la orientación en Yaw γ_x, y las velocidades angulares $\dot{\rho}_x, \dot{\phi}_y, \dot{\gamma}_x$, las cuales son las de interés en este trabajo, pueden estimarse utilizando las técnicas anteriormente mencionadas para los acelerómetros, giróscopos y magnetómetros para luego ser fusionadas aplicando la estructura de la Figura 11, utilizando un filtro de Kalman angular, como se explica en las siguientes secciones.

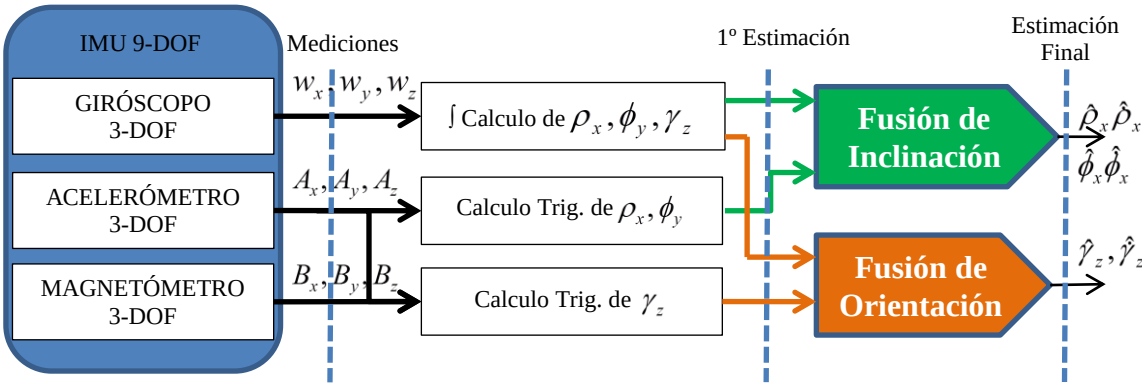


Figura 11. Estructura de Fusión Sensorial en una IMU para calcular $\hat{\rho}_x, \hat{\rho}_x, \hat{\phi}_y, \hat{\phi}_y, \hat{\gamma}_z, \hat{\gamma}_z$.

Filtro de Kalman

El filtro Kalman se comporta como un estimador lineal y óptimo del estado de un proceso, en los casos en que el proceso a ser estimado es regido por una dinámica lineal y el ruido que lo perturba es blanco gaussiano. Su propósito es utilizar mediciones que son adquiridas a lo largo del tiempo y afectadas por variaciones aleatorias (ruido), junto con el conocimiento del comportamiento del sistema, para producir estimaciones que tiendan

a estar más cerca del valor real del proceso en cuestión. Puede ser implementado como se presenta en el algoritmo recursivo de la Figura 12, que se ejecuta en tiempo real y realiza un promedio ponderado entre la predicción de un estado y las mediciones del mismo, utilizando la certidumbre de cada estimación como factor de ponderación. Con este procedimiento, el filtro Kalman logra que la varianza de la estimación final sea menor que la varianza de cada estimación por separado [17].

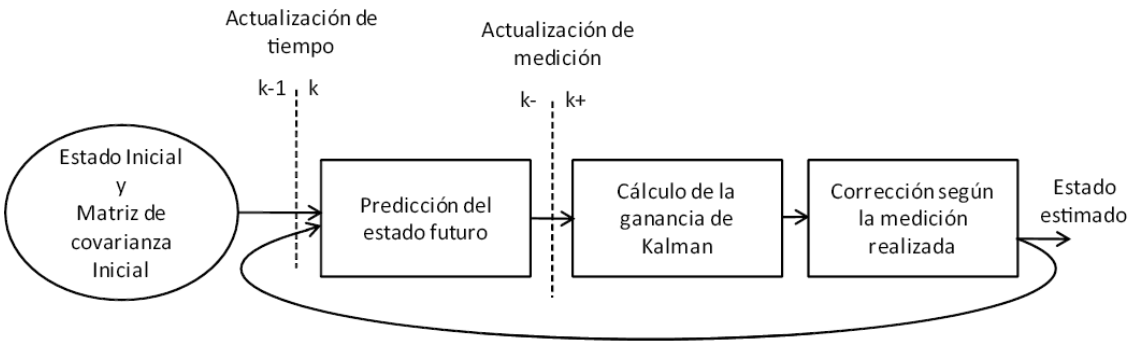


Figura 12. Secuencia de Operación del Filtro de Kalman.

Otra característica de este filtro es su eficiencia en el uso de recursos computacionales dado que aunque las ecuaciones que lo describen son expresadas en términos matriciales, luego de disponer del modelo sobre el cual se va a aplicar el filtro, se puede reducir las expresiones hasta obtener ecuaciones lineales con operaciones escalares definidas en el dominio de los reales. No requiere amplio volúmenes de almacenamiento de datos porque la salida en un instante de tiempo cualquiera solo depende de los resultados obtenidos en el instante previo y de la incertidumbre del mismo.

Filtro de Kalman Discreto

Para modelar el proceso a ser estimado, las variables de interés del sistema son definidas como estados y agrupadas en un vector de estado (x). El modelo contiene un

componente dinámico expresado en la ecuación *(32)*, que incluye la información determinística que define el comportamiento del sistema $(\phi x_{k-1} + \Gamma u_{k-1})$ y un término aleatorio representando el ruido del proceso (w_{k-1}). También se modela la forma en la que se puede acceder a la información de los estados del sistema $(z_k = Hx_k)$, tomando en cuenta las variaciones aleatorias asociadas a la medición (v_k), como lo muestra la ecuación *(33)*.

$$x_K = \phi x_{k-1} + \Gamma u_{k-1} + w_{k-1} \qquad (32)$$

$$z_k = Hx_k + v_k \qquad (33)$$

Cada iteración del filtro puede ser subdividida en dos etapas, una de actualización de tiempo y otra de actualización de medición. En la primera se realiza una estimación a priori del estado actual del proceso $\hat{x}_k(-)$, en donde se proyectan los estados en un instante de tiempo anterior $\hat{x}_{k-1}$ utilizando la matriz de transición ϕ y las señales de control u_{k-1} como describe la ecuación *(34)*, La ecuación *(35)* describe como la incertidumbre de la estimación previa P_{k-1} se propaga al estado actual $P_k(-)$ utilizando el modelo del sistema y la varianza del proceso Q.

$$\hat{x}_k(-) = \phi \hat{x}_{k-1} + \Gamma u_{k-1} \qquad (34)$$

$$\hat{P}_k(-) = \phi P_{k-1} \phi^T + Q \qquad (35)$$

En la segunda etapa del filtro, la actualización de medición es la encargada de incorporar las nuevas mediciones del proceso a la predicción realizada en la etapa anterior para producir un estimado con menor varianza. Primero se calcula la ganancia de Kalman K_k a partir de la certidumbre de la predicción de los estados y de la varianza de las mediciones R como lo describe la ecuación *(36)*. Luego se obtiene un estimado a

posteriori del estado del sistema como se observa en las ecuaciones *(37)* y *(38)* en donde se corrige la estimación a priori usando la nueva medición z_k con la ganancia de Kalman, y se actualiza la incertidumbre asociada a la estimación a posteriori $P_k(+)$, respectivamente.

$$K_k = P_k(-)H^T\left[HP_k(-)H^T + R\right]^{-1} \qquad (36)$$

$$\hat{x}_k = \hat{x}_k(-) + K_k\left[z_k - H\hat{x}_k(-)\right] \qquad (37)$$

$$P_k(+) = \left[I - K_k H\right]P_k(-) \qquad (38)$$

Fusión Sensorial en una IMU, Utilizando el Filtro de Kalman Discreto

Como se había explicado en la Figura 10, las variables de interés del presente trabajo son: las inclinaciones en Pitch ϕ_y y Roll ρ_x, la orientación en Yaw γ_x, y las velocidades angulares $\dot{\rho}_x, \dot{\phi}_y, \dot{\gamma}_x$, para lo cual se puede diseñar un filtro de Kalman Angular e implementarlo de forma paralela para realizar la fusión sensorial en cada ángulo de interés, como se observa en la Figura 13. Para esto, de acuerdo al modelo dinámico del sistema, se determinan los estados de interés, la matriz de transición de estados, la matriz de sensitividad y las entradas de control existentes. Posterior a esto se realiza el proceso de puesta a punto del filtro, fijando las condiciones iniciales, la varianza del proceso y la varianza de las mediciones.

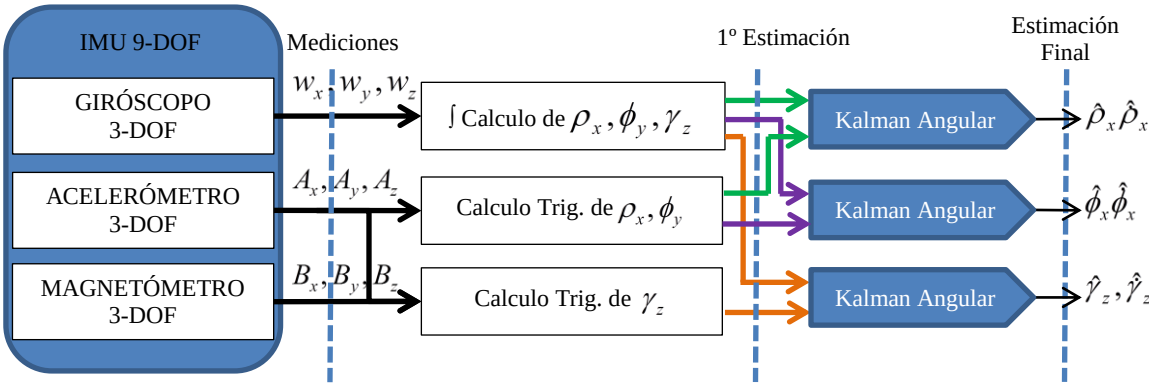


Figura 13. Aplicación de Tres Filtros de Kalman Angulares a la IMU.

Teniendo en cuenta que con este filtro se desea estimar un ángulo y su tasa de cambio, podemos definirlos como estados del sistema. Para esto se utiliza un modelo que expresa la relación matemática entre el ángulo y su tasa de cambio en un instante de tiempo dado, con dichas cantidades en un instante previo, como se observa en la Figura 14, donde se presenta la evolución temporal del estado de Roll ρ_x para el filtro angular, los otros ángulos de interés, se realizan de manera análoga.

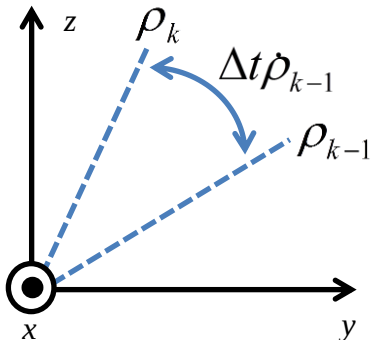


Figura 14. Modelo de Transición de Estados Para el Filtro de Kalman Angular.

Utilizando la Figura 14 como referencia, se puede deducir la ecuación *(40)*, que representa el modelo dinámico del sistema angular para cualquiera de los ejes de rotación. En este modelo se pueden observar los estados y la matriz de transición de

estados, de igual forma cabe anotar la ausencia de señales de control. Para generalizar la notación, llamaremos cualquier ángulo de interés como θ.

$$\hat{x}_k(-) = \phi\hat{x}_{k-1} + \Gamma u_{k-1} \quad (39)$$

$$\begin{bmatrix} \hat{\theta}_k \\ \hat{\dot{\theta}}_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\theta}_{k-1} \\ \hat{\dot{\theta}}_{k-1} \end{bmatrix} + 0(u_{k-1}) \quad (40)$$

Utilizando las técnicas y procedimientos mencionados anteriormente, con los acelerómetros, giróscopos y magnetómetros podemos obtener los ángulos de inclinación y la velocidad de rotación en cada eje, por lo tanto el modelo de salida, es el que se señala en la ecuación *(42)*, de donde se obtiene la matriz de sensitividad *H*.

$$z_k = Hx_k + v_k \quad (41)$$

$$\begin{bmatrix} \hat{\theta}_k \\ \hat{\dot{\theta}}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_k \\ \hat{\dot{\theta}}_k \end{bmatrix} + v_k \quad (42)$$

Posterior a esto, como se observa en las ecuaciones *(32)* y *(33)*, teniendo en cuenta que el proceso a ser estimado y las mediciones del mismo están sujetos a variaciones aleatorias de distribución gaussiana, las cuales se representan con las variables aleatorias w_k y v_k de varianzas σ_w^2 y σ_v^2, podemos utilizar la publicación del filtro de Kalman en 1960 [18], donde demostró que este tipo de filtro, presenta un desempeño óptimo cuando los parámetros de ajuste Q y R son igual a la covarianza del proceso σ_w^2 y la covarianza de las mediciones σ_v^2, respectivamente, como se señala en las ecuaciones *(43)* y *(44)*.

$$Q = \sigma_w^2, w_k \rightarrow N(0, Q) \quad (43)$$

$$R = \sigma_v^2, v_k \rightarrow N(0, R) \quad (44)$$

Teniendo en cuenta lo anterior, la covarianza de las mediciones se determina directamente de las muestras tomadas, pero para determinar la covarianza del proceso no existe un procedimiento establecido, ya que no se tiene acceso directo al proceso sino a

sus mediciones. Por lo general se seleccionan valores pequeños cuando existe mucha confianza en el modelo desarrollado, y valores mayores a medida que se tengan mayores dudas sobre él. Una buena forma de conseguir un buen desempeño, es seguir el procedimiento de prueba y error realizado por Welch y Bishop [19].

Por último aunque según Kalman [18], el comportamiento del filtro en estado estacionario es independiente de las condiciones iniciales seleccionadas, es importante escoger el estado inicial del sistema y la matriz de covarianza de los estados iniciales, lo más cercano a sus valores reales para reducir el tiempo de estabilización y el error en la etapa transitoria, una forma de estimar la matriz de covarianza inicial puede ser efectuando una ejecución de prueba del filtro y observar el valor en el que se estabiliza para luego utilizarlo como valor inicial.

En la Tabla 1, se muestra un resumen de las matrices y vectores utilizados en el filtro de Kalman Angular implementado en este trabajo.

Tabla 1. Resumen de Matrices Utilizadas en Filtro de Kalman Angular

NOMBRE	MATRIZ
Estados del Sistema	$x_k = \begin{bmatrix} \hat{\theta}_k \\ \hat{\dot{\theta}}_k \end{bmatrix}$
Matriz de Transición	$\phi = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$
Vector de Mediciones	$z_k = \begin{bmatrix} \theta_k + v_{\theta_k} \\ \dot{\theta}_k + v_{\dot{\theta}_k} \end{bmatrix}$
Matriz de Sensitividad	$H = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
Matriz de Covarianza del Proceso	$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}$

Matriz de Covarianza de las Mediciones	$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$

Principios de Funcionamiento de Servomotores

El control de un motor, requiere tres elementos básicos, un motor, un controlador o drive y uno o más dispositivos de realimentación. Los drive normalmente controlan la corriente para producir torque, velocidad o posición. Y los elementos de realimentación pueden ser encoders, resolvers o potenciómetros que indican al sistema de control la variable a controlar [20].

En este trabajo se utilizan específicamente servomotores, los cuales son dispositivos que tienen la capacidad de situar su eje de rotación en cualquier posición dentro que un rango de operación que normalmente está entre 0º y 180º de acuerdo a un tren de pulsos de control que varía su ciclo útil de forma proporcional al ángulo deseado, de acuerdo a las especificaciones dadas por el fabricante.

La composición básica de estos dispositivos se observa en la Figura 15. En ésta, se pueden distinguir como elementos básicos, el motor de corriente continua que le da movilidad, los engranajes reductores que convierten la velocidad de giro del motor en torque y el circuito de control que incluye un potenciómetro que sirve como realimentación, el cual le indica la posición actual del eje.

Control de Servomotores

El control de un servomotor, se reduce a indicar su posición mediante una señal modulada de pulsos PWM (Pulse width modulation), la cual varía el ancho del pulso de forma proporcional al ángulo en el que se desea posicionar, Figura 16, para esto, cada

fabricante establece estándares diferentes donde indica el mínimo y máximo ancho del pulso, como los que se observan en la Tabla 2.

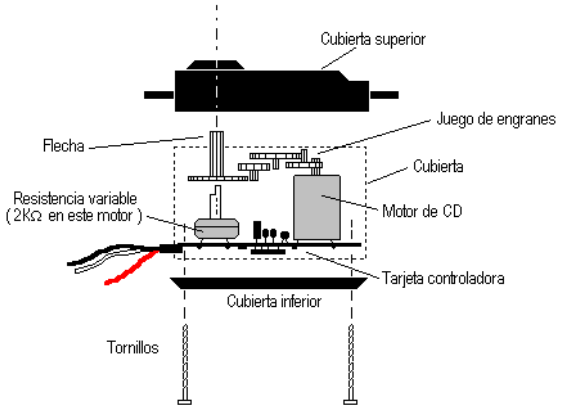


Figura 15. Partes de un Servomotor.

Tabla 2. Comparación de Tiempos de Operación de Algunos Servomotores Comerciales.

FABRICANTE	DURACIÓN DEL PULSO EN MILISEGUNDOS			FRECUENCIA EN HERTZ
	MIN 0º	NEU 90 º	MÁX 180 º	
FUTABA	0.9	1.5	2.1	50
HITEC	0.9	1.5	2.1	50
GRAUPNER/JR	0.8	1.5	2.2	50
MULTIPLEX	1.05	1.6	2.15	40
ROBBE	0.65	1.3	1.95	50
SIMPROP	1.2	1.7	2.2	50

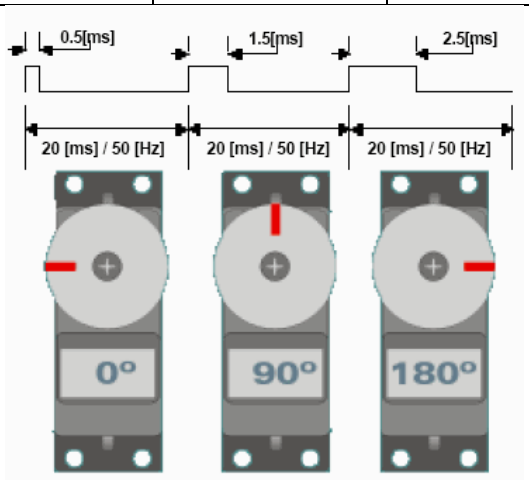


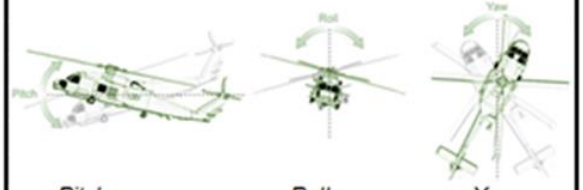
Figura 16. Señal de Control de Posicionamiento de un Servo.

Capítulo 3

Especificaciones del Prototipo.

Para establecer las especificaciones del prototipo a desarrollar, se toman como base las características y límites del movimiento de la cabeza humana en cada dirección, información técnica de manuales de diseño de H.M.D. (Helmet Mounted Dysplay) [7], [21], e información técnica de los límites de vuelo de un helicóptero UH-60L Black Hawk [22], logrando determinar las características de los movimientos a registrar como se observa en la Tabla 3, a una frecuencia mínima de muestreo de 100 Hz para detectar los movimientos con la resolución y velocidad requerida por la aplicación.

Tabla 3. Características de movimientos a registrar

CARACTERISTICAS DE MOVIMIENTO DE CABEZA			CARACTERISTICAS DE MOVIMIENTO DE PLATAFORMA		
ANGULO	LIMITES EN GRADOS	VELOCIDAD MAXIMA	ANGULO	LIMITES EN GRADOS	VELOCIDAD MAXIMA
PITCH	±60°	±280°/seg	PITCH	±30°	±200°/seg
ROLL	±30°	±280°/seg	ROLL	±60°	±200°/seg
YAW	± 90	±280°/seg	YAW	±360°	±200°/seg
 Pitch Roll Yaw			 Pitch Roll Yaw		

De igual forma, teniendo en cuenta que la aplicación propuesta, requiere la detección, análisis y observación de un objetivo por parte del usuario, se establecer un tiempo de retardo máximo de 0,5 segundos para que la plataforma se posicione siguiendo la L.o.S. del individuo que utiliza el casco.

Adicional a estas especificaciones, se debe tener en cuenta algunos requerimientos generales de operación del prototipo desarrollado, como lo son un

ambiente de operación que permita cambios bruscos de luminosidad, temperatura y polución y no requiera una línea de vista continua entre un sistema emisor y receptor para determinar el LoS del usuario del casco; de igual forma los sensores utilizados y sistemas de conexión, no deben afectar el confort en la operación normal de la plataforma que conduce el operario del casco.

El sistema de procesamiento utilizado debe procesar en tiempo real la información generada por los sensores, y permitir ser escalado fácilmente a las diferentes plataformas embebidas comerciales sin realizar modificaciones mayores.

La plataforma de seguimiento de la línea de vista del usuario del casco, debe realizar movimientos con grados de libertad en Pitch y Yaw con la misma amplitud de los movimientos registrados, con una resolución menor a un grado, con capacidad de mover una carga útil de máximo 150 gramos.

Por último, teniendo en cuenta que el prototipo solo pretende demostrar el principio de funcionamiento y la viabilidad de su aplicación en diferentes plataformas móviles, debe utilizar sensores y dispositivos de costo razonable que optimicen su operación a través de algoritmos o técnicas de fusión y filtrado.

Capítulo 4

Desarrollo del Prototipo.

Esquema General del Prototipo

Teniendo como referencia los parámetros de diseño establecidos, y las características de las tecnologías utilizadas para la detección de movimiento, mencionadas anteriormente; se determinó utilizar en la etapa de sensado de movimiento dos IMU´s con 9 grados de libertad, cada una compuesta por arreglos ortogonales de tres acelerómetros, giróscopos y magnetómetros, conectadas a través de dos puertos USB a un sistema de procesamiento que ejecuta una aplicación en C, la cual calcula la inclinación y orientación diferencial de las dos IMU y utiliza el controlador de servos conectado a través de otro puerto USB, para posicionar los servomotores de la plataforma seguidora de L.o.S., replicando de esta forma la línea de vista del usuario del casco, como se observa en la estructura de la Figura 17.

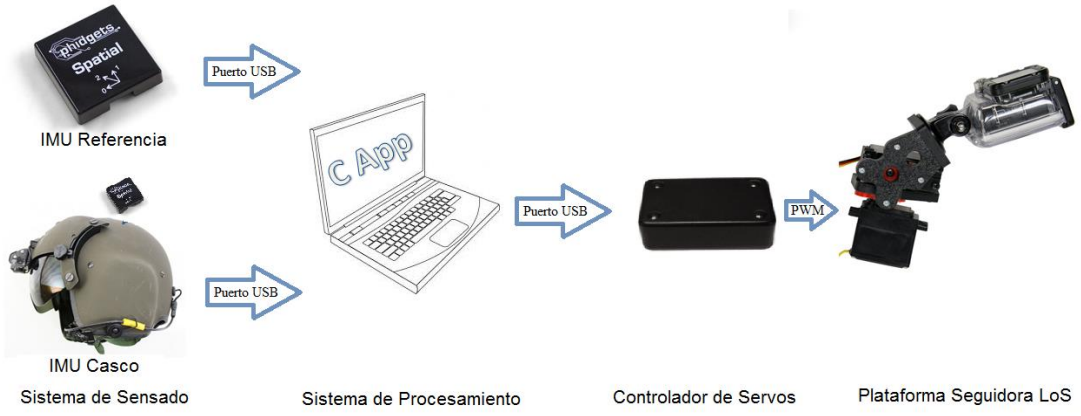


Figura 17. Esquema general del prototipo implementado.

A continuación se explican detenidamente los componentes y procesos realizados en cada etapa para cumplir con los requerimientos de diseño establecidos para el prototipo.

Sistema de Sensado de Movimiento del Casco con Respecto a una Referencia

Teniendo en cuenta que para esta etapa se requiere encontrar el valor diferencial de orientación e inclinación de dos IMU's, se determinó utilizar elementos de la misma referencia que tengan parámetros similares, seleccionando específicamente la IMU P/N 1042 del fabricante canadiense Phidgets, teniendo en cuenta sus ventajas de bajo costo, facilidad de conexión (USB) e Interfaz de programación de aplicaciones, la cual cumple los requerimientos de diseño y tiene los parámetros observados en la Tabla 4.

Tabla 4. Especificaciones Técnicas de IMU P/N 1042 de Phidgets.

Accelerometer		**Board**	
Acceleration Measurement Max	± 8 g	API Object Name	Spatial
Acceleration Measurement Resolution	976.7 µg	Current Consumption Max	40 mA
Accelerometer White Noise σ	2.8 mg	Sampling Speed Min	1 s/sample
Accelerometer Minimum Drift σ	1.9 mg	Sampling Speed Max	4 ms/sample
Accelerometer Optimal Averaging Period	286 s	Sampling Speed Min (Webservice)	1 s/sample
Gyroscope		Sampling Speed Max (Webservice)	12 ms/sample
Gyroscope Speed Max	± 2000°/s	USB Voltage Min	4.4 V DC
Gyroscope Resolution	0.07°/s	USB Voltage Max	5.3 V DC
Gyroscope White Noise σ	0.59°/s	USB Speed	Full Speed
Gryoscope Minimum Drift σ	0.0019°/s	Operating Temperature Min	-40 °C
Gyroscope Optimal Averaging Period	8628 s	Operating Temperature Max	85 °C
Compass			
Magnetic Field Max	5.5 G		
Compass Resolution	3 mG		
Compass White Noise σ	1.2 mG		
Compass Minimum Drift σ	87 µG		
Compass Optimal Averaging Period	52 s		

La primera IMU (IMU CASCO), se ubica fija al casco (HGU-56P) como el de la Figura 17, teniendo la función de entregar los datos de aceleración, velocidad de rotación e intensidad de campo magnético en los ejes X, Y y Z al sistema de procesamiento, para con base en ellos calcular la orientación e inclinación del casco con respecto a la gravedad y campo magnético terrestre.

La segunda IMU, se ubica fija al sistema de referencia móvil, que puede ser un vehículo terrestre, aéreo, etc. Teniendo la función de entregar los datos de aceleración, velocidad de rotación e intensidad de campo magnético en los ejes X, Y y Z al sistema de procesamiento para calcular la orientación e inclinación del sistema de referencia con respecto a la gravedad y campo magnético terrestre.

Con los datos de las dos IMU anteriores, al encontrar la diferencia entre sus orientaciones e inclinaciones, es posible encontrar los verdaderos ángulos de interés para esta investigación, los cuales corresponden a la orientación e inclinación del casco, pero con respecto a la IMU de referencia.

Para cada una de estas IMU, es necesario realizar un proceso de calibración que elimine el sesgo de los acelerómetros y giróscopos, y las afectaciones de Hard Iron y Soft Iron de los magnetómetros, para lo cual se utilizó la aplicación e información del fabricante PHIDGETS [23] [24] [25] [26], comprobando su resultado, obteniendo gráficas como las que se observan en la Figura 18 donde se registra el comportamiento gaussiano de la señal de los acelerómetros centrado en cero y en la Figura 19, donde se comprueba cómo se eliminan la interferencia por de Hard Iron que desplaza el centro de la esfera de intensidades de campo magnético del origen y de Soft Iron que deforma la esfera de intensidades de campo magnético, en esta gráfica se observa en color rojo la esfera de intensidades de campo magnético antes de la calibración y en color verde la obtenida después de ingresar los parámetros de calibración obtenidos.

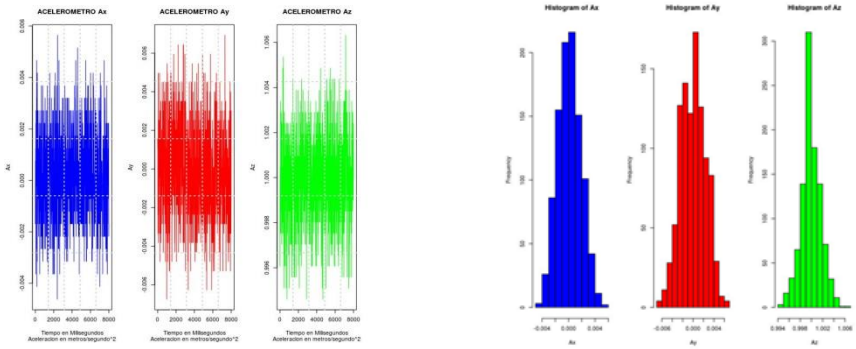


Figura 18. Señales Obtenidas de los Acelerómetros de la IMU CASCO en reposo, centrados en cero en ***x*** *y* ***y****, y uno (1) en* ***z****, después del procedimiento de calibración.*

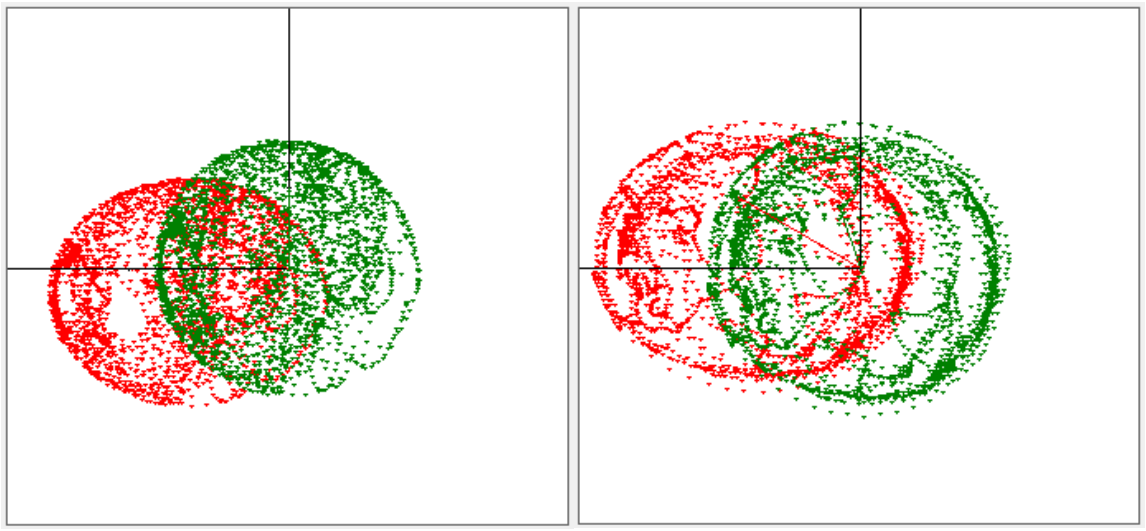


Figura 19. Calibración de Magnetómetros corrigiendo Soft Iron y Hard Iron en IMU (Izquierda IMU Plataforma, Derecha IMU casco). Esfera roja obtenida antes de la calibración, Esfera Verde obtenida después de calibrados.

Sistema de Procesamiento, Control y Visualización

Para implementar el sistema de procesamiento, control y visualización cumpliendo los requisitos de diseño establecidos y con el fin del optimizar las librerías disponibles para las IMU seleccionadas y controlador de servos de la empresa PHIDGETS, se determinó realizar una aplicación en lenguaje C que puede ser ejecutada

en cualquier plataforma con sistema operativo LINUX que tenga instaladas las librerías del fabricante "*libusb-1.0 development libraries*" [24].

Para la implementación del algoritmo en el sistema de procesamiento, con el fin de obtener un programa de rápida ejecución que permita realizar un procesamiento en tiempo real, se determinó realizar programación creando un hilo o proceso para cada sensor o dispositivo, es decir, un hilo principal que controla la posición de los servos de la plataforma, y dos hilos secundarios en los que se realiza el procesamiento de los datos de las IMU, aplicando un filtro de Kalman discreto independiente para cada ángulo de interés en cada hilo para fusionar los datos de las IMU (IMU casco e IMU referencia), estos hilos utilizan las ecuaciones encontradas en el segundo capítulo de este documento para calcular las variables de interés de este trabajo.

Hilo Principal y de Control de Servos de PITCH y YAW.

El hilo principal tiene como funciones primordiales definir los procesos generales comunes a los dos hilos secundarios, como lo son crear los archivos planos *.csv que solo son utilizados para evaluar el comportamiento del sistema después de finalizar su ejecución, definir los controladores y configuraciones de los dispositivos, iniciar y finalizar los hilos secundarios y controlar la posición de los servos de PITCH y ROLL de acuerdo a la orientación e inclinación calculada por los hilos secundarios para el casco y la referencia tomando la cantidad de datos indicada, en la Figura 20, se observa detalladamente el diagrama de flujo de este proceso o hilo.

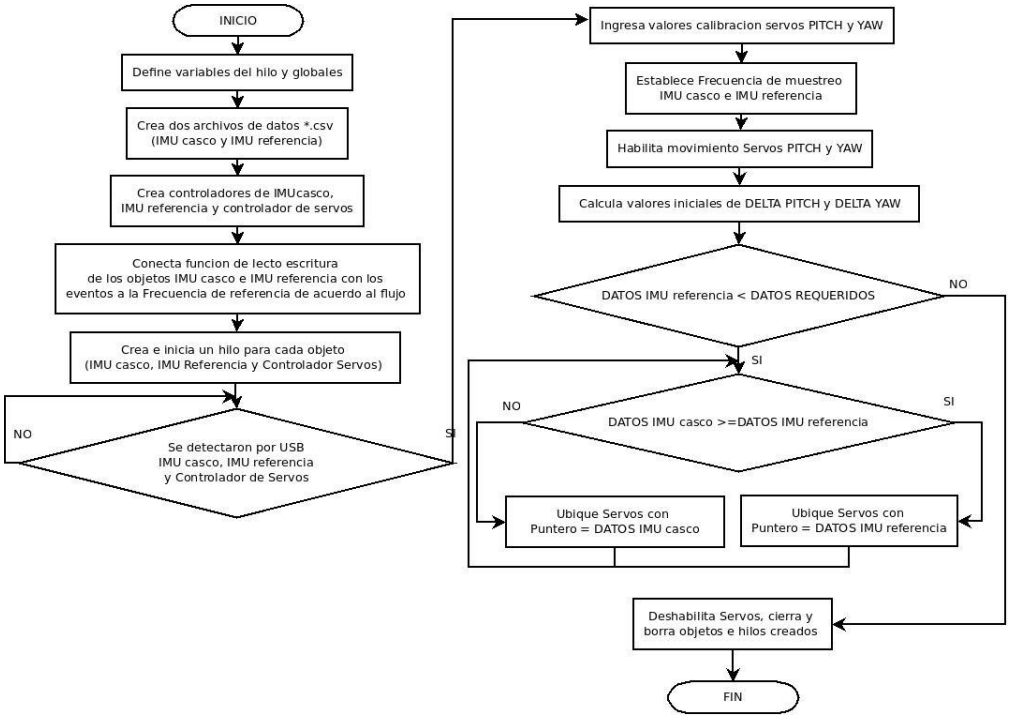


Figura 20. Diagrama de flujo de hilo principal.

Hilo Secundario y de Control y Procesamiento de IMU casco.

Este hilo secundario se encarga de visualizar las variables de interés en pantalla, leer a la frecuencia definida *(Frecuencia de muestreo = 8 Milisegundos)* los datos de los acelerómetros, giróscopos y magnetómetros fijos al casco, almacenándolos en el archivo plano definido, calcular la orientación e inclinación del casco, con referencia a la gravedad y campo magnético terrestres, realizando fusión sensorial con un filtro de Kalman y por último determinar la inclinación y orientación que debe tener la plataforma para seguir la línea de vista del casco. Es necesario aclarar que como cada IMU tiene un hilo independiente y totalmente asíncrono del otro, en este hilo para calcular la posición de la plataforma, se utiliza el puntero de menor valor entre DATOS IMU casco y DATOS IMU referencia, para asegurar que existe una lectura para ambas IMU. En la Figura 21, se observa detalladamente el diagrama de flujo de este proceso o hilo.

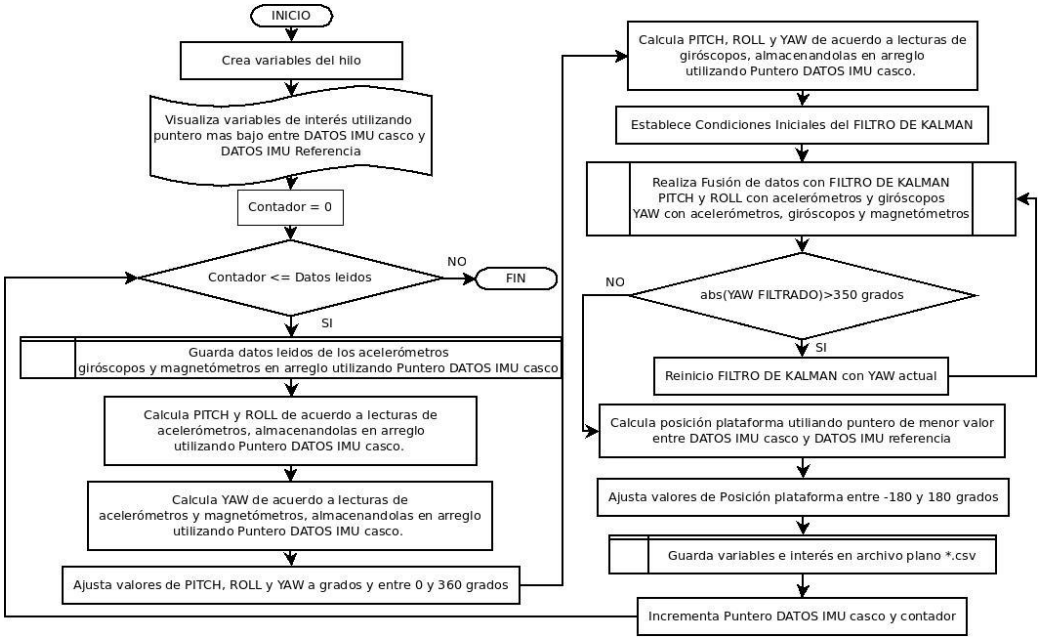


Figura 21. Diagrama de flujo del hilo secundario y de control y procesamiento de IMU casco.

Hilo Secundario y de Control y Procesamiento de IMU referencia.

Este hilo secundario se encarga de leer a la frecuencia definida *(Frecuencia de muestreo = 8 Milisegundos)* los datos de los acelerómetros, giróscopos y magnetómetros fijos a la referencia, almacenándolos en el archivo plano predeterminado, calcular la orientación e inclinación del sistema de referencia con respecto a la gravedad y campo magnético terrestres, y finamente realizar fusión sensorial con un filtro de Kalman para determinar la inclinación y orientación del sistema de referencia con respecto a la gravedad y campo magnético terrestre. En la Figura 22, se observa detalladamente el diagrama de flujo de este proceso o hilo.

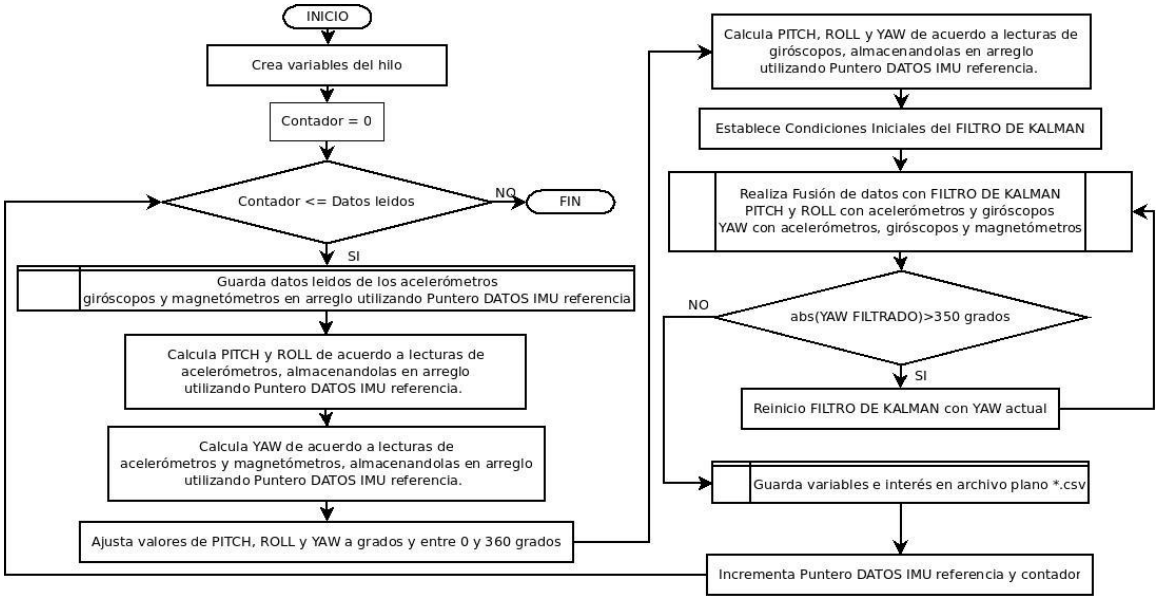


Figura 22. Diagrama de flujo del hilo secundario y de control y procesamiento de IMU casco.

Implementación de Fusión de Datos de Cada IMU con filtro de Kalman.

Para realizar la implementación del filtro de Kalman angular para cada ángulo de interés en las IMU, se utilizan las ecuaciones *(34),(35),(36),(37)* y *(38)*, explicadas en el capítulo 2 de este documento, desplegando las matrices hasta obtener de cada ecuación matricial un conjunto de ecuaciones lineales equivalentes, logrando de esta manera obtener ecuaciones lineales de fácil implementación en lenguajes de programación como "C" utilizado en este trabajo, o cualquier tipo de microcontrolador, estas ecuaciones se resumen en la Tabla 5.

Tabla 5. Conjunto de ecuaciones lineales del filtro discreto de Kalman.

$$\hat{x}_k(-) = \phi\hat{x}_{k-1} + \Gamma u_{k-1}$$
`titahatminus=titahat[k-1]+dt*titapointhat[k-1];`
`titapointhatminus = titapointhat[k-1];`
$$\hat{P}_k(-) = \phi P_{k-1}\phi^T + Q$$
`P11minus = P11[k-1]+dt*P21[k-1]+Q11;`
`P12minus = P12[k-1]+dt*P22[k-1]-dt*(P11[k-1]+dt*P21[k-1])+Q12;`
`P21minus = P21[k-1]+Q2;`

```
P22minus = P22[k-1]-P21[k-1]*dt+Q2;
```

$$K_k = P_k(-)H^T\left[HP_k(-)H^T + R\right]^{-1}$$

```
k11num=P11minus*R22-P12minus*R21+P11minus*P22minus-
        P12minus*P21minus;
k11den1=P11minus*R22-P12minus*R21-P21minus*R12+P22minus*R11;
k11den2=R11*R22-R12*R21+P11minus*P22minus-P12minus*P21minus;
k11=k11num/(k11den1+k11den2);

k12num=-P11minus*R12+P12minus*R11;
k12den1=P11minus*R22-P12minus*R21-P21minus*R12
        +P22minus*R11+R11*R22;
k12den2=-R12*R21+P11minus*P22minus-P12minus*P21minus;
k12=k12num/(k12den1 + k12den2);

k21numr=(P21minus*R22-P22minus*R21);
k21den1=P11minus*R22-P12minus*R21-P21minus*R12
        +P22minus*R11+R11*R22;
k21den2=-R12*R21+P11minus*P22minus-P12minus*P21minus;
k21=k21num/(k21den1+k21den2);

k22num=-(P21minus*R12-P22minus*R11-P11minus*P22minus
       +P12minus*P21minus);
k22den1=P11minus*R22-P12minus*R21-P21minus*R12
       +P22minus*R11+R11*R22;
k22den2=-R12*R21+P11minus*P22minus-P12minus*P21minus;
k22 = k22num/(k22den1+k22den2);
```

$$\hat{x}_k = \hat{x}_k(-) + K_k\left[z_k - H\hat{x}_k(-)\right]$$

```
titahat[k]=titahatminus-k11*(titahatminus-pitch[k]);
titahat[k]=titahat[k]-k12*(titapointhatminus-wx[k]);
titapointhat[k]=titapointhatminus-k21*(titahatminus-pitch[k]);
titapointhat[k]=titapointhat[k]-k22*(titapointhatminus-wx[k]);
```

$$P_k(+) = \left[I - K_k H\right]P_k(-)$$

```
P11[k]=-P21minus*k12-P11minus*(k11-1);
P12[k]=-P22minus*k12-P12minus*(k11-1);
P21[k]=-P11minus*k21-P21minus*(k22-1);
P22[k]=-P12minus*k21-P22minus*(k22-1);
```

Para complementar y entender mejor la implementación realizada en lenguaje “C”; en la Tabla 6, se especifica en notación matricial los valores utilizados para esta implementación, como lo son la frecuencia de muestreo determinada para cada una de las IMU (8 milisegundos), los valores de la varianza obtenidos de las mediciones para los acelerómetros y giróscopos y los valores encontrados utilizando el método de prueba y error para la varianza del proceso realizado por Welch y Bishop [19].

Tabla 6. Valores de Implementación del filtro de Kalman

NOMBRE	MATRIZ
Estados del Sistema	$\hat{x}_k = \begin{bmatrix} \hat{\theta}_k \\ \hat{\dot{\theta}}_k \end{bmatrix}$ $\hat{\theta}_k$ = Angulo Estimado. $\hat{\dot{\theta}}_k$ =Velocidad de Rotación Estimada.
Matriz de Transición	$\phi = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$ Δt = 1 / Frecuencia de muestreo = 8 ms
Vector de Mediciones	$z_k = \begin{bmatrix} \theta_k \\ \dot{\theta}_k \end{bmatrix}$ θ_k = Angulo calculado con acelerómetros. $\dot{\theta}_k$ = Velocidad de rotación calculada de gyro.
Matriz de Sensitividad	$H = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
Matriz de Covarianza del Proceso	$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} = \begin{bmatrix} 0.00000102 & 0 \\ 0 & 10 \end{bmatrix}$ Q_{11} Corresponde a la varianza del proceso respecto a los acelerómetros. Q_{22} Corresponde a la varianza del proceso respecto a los gyros.
Matriz de Covarianza de las Mediciones	$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} = \begin{bmatrix} 0.009364 & 0 \\ 0 & 0.04 \end{bmatrix}$ R_{11} Corresponde a la varianza obtenida de las muestras respecto a los acelerómetros. R_{22} Corresponde a la varianza obtenida de las muestras respecto a los gyros.

Interfaz de Control de Servos de PITCH y YAW.

Como ya se había mencionado anteriormente, el control de estos servos se realiza en el hilo principal y su posición es actualizada en tiempo real con el valor del ángulo calculado, a la frecuencia de muestreo seleccionada (8 milisegundos), con base en la

diferencia entre los ángulos de orientación e inclinación de la IMU del casco y la IMU de referencia.

Teniendo en cuenta que el controlador de servos seleccionado para esta aplicación, el cual se explica más adelante, es otro dispositivo PHIDGET con librerías de desarrollo que facilitan la implementación; el algoritmo de control de los servos, utiliza estas librerías para indicar una posición entre 0 y 180 grados para posicionar el servo como se observa en la Figura 24 en color rojo.

Por lo anterior, y tomando como referencia el sistema de coordenadas del casco, como se observa en la Figura 24 en color azul, es necesario igualar el sistema de coordenadas del controlador de los servos con el del casco, para lo cual basta con sumar al sistema de coordenadas de los servos, 90 grados a cualquier posición entre -90 y 90 Grados del casco.

De igual forma para calcular el ángulo de YAW *(β en la* Figura 23 *)* que debe tener la plataforma para replicar la línea de vista del usuario del casco a una distancia determinada *(L)*, se utiliza la ecuación *(45)*, la cual es calculada solucionando el problema geométrico observado en la Figura 23.

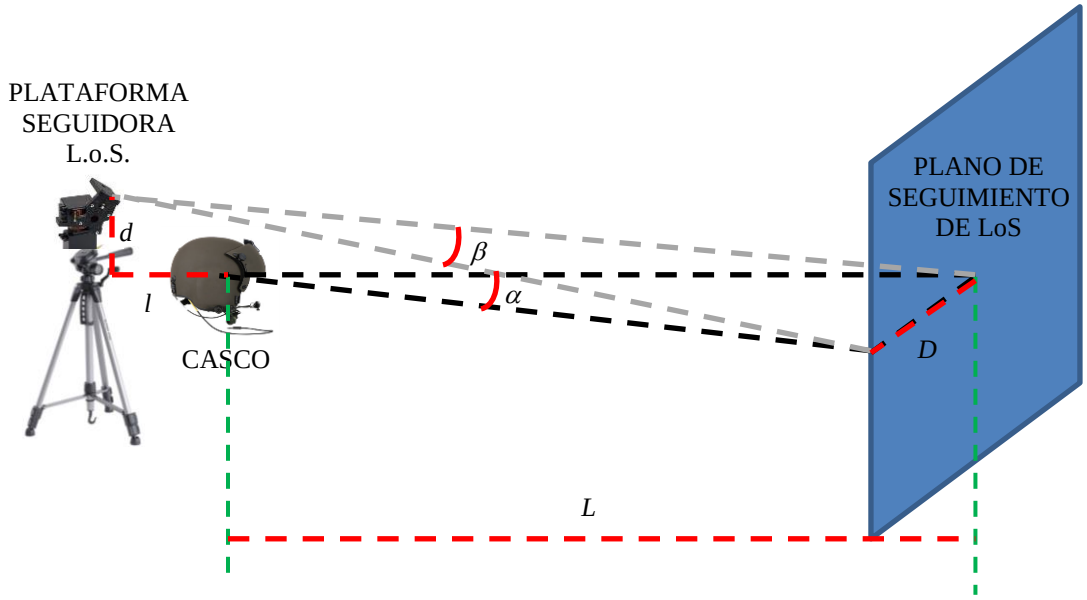


Figura 23. Cálculo del angulo de YAW (β) de la Plataforma

$$Yaw = \beta = \tan^{-1}\left(\frac{L\tan\alpha}{\sqrt{(l+L)^2 + d^2}}\right) \qquad (45)$$

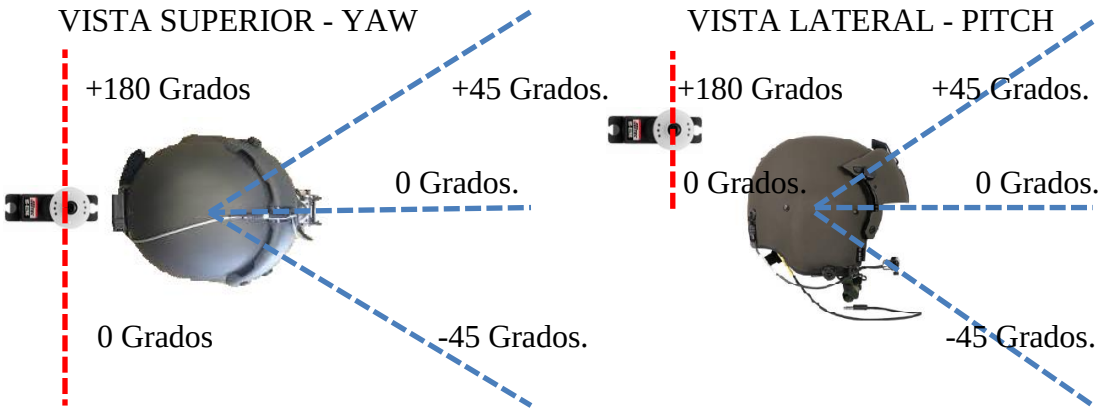


Figura 24. Sistema de coordenadas de control de los servos (Rojo) y casco (Azul).

Teniendo en cuenta lo anterior, si utilizamos la ecuación *(45)* y consideramos las condiciones de la aplicación propuesta, en la cual la distancia *(L)* siempre va a ser mucho

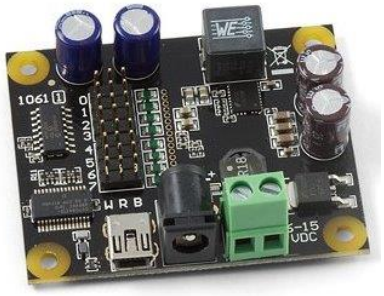
mayor que *(l)* y *(d)* se puede considerar que para replicar la L.o.S, del usuario del casco, los ángulo de inclinación y orientación del casco son iguales a los de la plataforma.

Controlador de Servos de PITCH y YAW.

Para efectuar la conexión entre el sistema de procesamiento, control y visualización, y la plataforma seguidora de LoS, se seleccionó el controlador de servos P/N 1061_1 de la empresa PHIDGETS, el cual tiene la capacidad de controlar la posición, velocidad y aceleración hasta de 8 servomotores, con una resolución de 125 pasos por grado, utilizando una conexión USB con el sistema de control y procesamiento. Las especificaciones de este dispositivo se encuentran en la Tabla 7.

Este controlador requiere un proceso de calibración inicial para mejorar su precisión, teniendo en cuenta que cada servomotor posee características específicas determinadas en su fabricación [25]; este proceso consiste en indicar el ancho del pulso de control PWM para posicionarse en 0 Grados y 180 Grados. Después de realizada su calibración, puede utilizarse directamente con las librerías de control indicando el ángulo en el cual debe posicionarse entre 0 y 180 grados.

Tabla 7. Especificaciones técnicas del controlador de servos 1061_1 PHIDGETS.

Servo Controller		**Physical Properties**	
API Object Name	AdvancedServo	Power Jack Hole Diameter	5.5 mm
Number of Motor Ports	8	Power Jack Pin Diameter	2.1 mm
Pulse Width Min	83.3 ns	Power Jack Polarity	Center Positive
Pulse Width Max	2.7 ms	Recommended Wire Size	12 - 24 AWG
Pulse Width Resolution	83.3 ns	Object Temperature Min	0 °C
Pulse Code Period Max	25 ms	Object Temperature Max	70 °C
Electrical Properties			
Supply Voltage Min	6 V DC		
Supply Voltage Max	15 V DC		
Current Consumption Max	26 mA		
Continuous Motor Current Max	(per motor) 1.6 A		
Overcurrent Trigger	(combined) 12 A		
Surge Current Max	(per motor) 3 A		
Output Impedance (Motor)	600 Ω		
Output Motor Voltage	5 V DC		
USB Speed	Full Speed		

Plataforma Seguidora de L.o.S.

Para implementar esta plataforma, en primer lugar se determinó utilizar la plataforma comercial de pan & tilt P/N SPT100 del fabricante SERVOCITY, la cual es un sistema recomendado para cámaras o sensores de hasta 10 onzas (280 Gramos), con servomotores P/N HS-485HB que pueden desplazar cargas hasta de 6 onzas (150 gramos).

Para la aplicación propuesta en este documento, esta plataforma se fijó a un trípode junto con la tarjeta de control de servos, y la IMU de referencia. De igual forma se adicionó un LED láser de baja potencia en el plato de fijación, como referencia para determinar la dirección hacia donde está orientada la plataforma y de esta manera poder comparar de forma directa la L.o.S. del casco y la replicada por la plataforma.

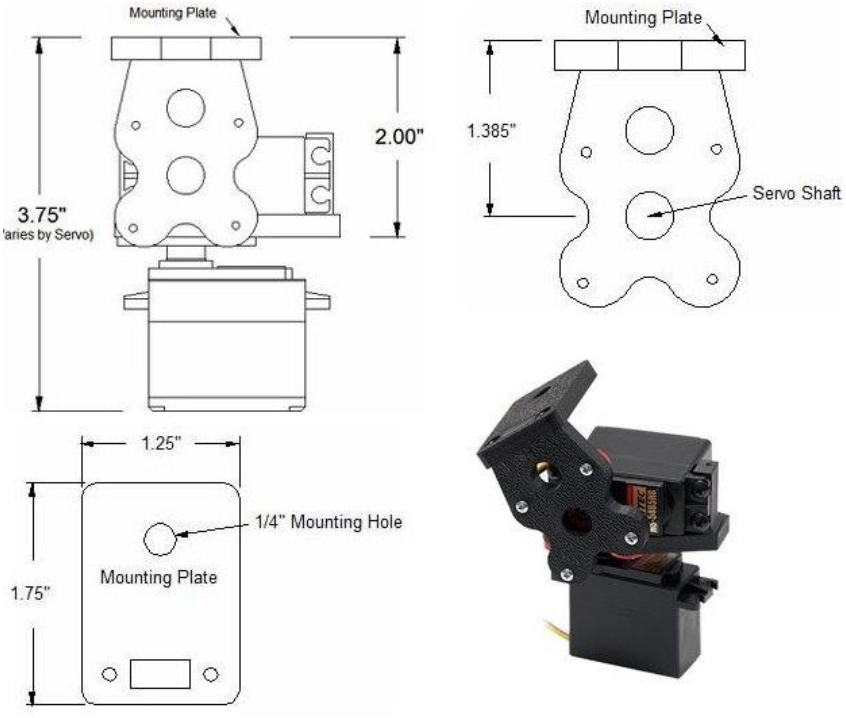


Figura 25. Plataforma de control de pan & tilt.

Capítulo 5

Análisis de Resultados.

Para comprobar el funcionamiento del prototipo implementado y analizar su comportamiento, se realizaron pruebas parciales de sus componentes principales, y posterior a esto, se integraron todos los elementos del prototipo hasta realizar una prueba final.

Pruebas de Funcionamiento del Filtro de Kalman Angular.

En esta prueba, se pudo comprobar la reducción de la varianza del ángulo calculado, después de aplicar el filtro de Kalman Angular implementado, comparándolo con el ángulo calculado exclusivamente con los acelerómetros y magnetómetros, obteniendo los resultados observados en la Tabla 8 y Figura 26.

En la Figura 26, se puede observar cómo aunque la IMU se encontraba en reposo, los valores de ángulos calculados únicamente con los acelerómetros y magnetómetros (azul) presentan alta varianza; de igual forma se observa que los valores de ángulos calculados con los giróscopos (verde) presentan menor varianza, pero con el tiempo se alejan del valor real, debido al error acumulativo de integración. Por último, se observa cómo, el filtro Kalman angular implementado, reduce la varianza del ángulo calculado, más del 99% para los ángulos de inclinación y el 95% para el ángulo de orientación.

Tabla 8. Reducción de varianza con filtro de Kalman implementado

VARIABLE	VAR ACEL	VAR KALMAN	% REDUCCIÓN VAR
PICTH	1.4518	0.003	99.7%
ROLL	1.7287	0.002	99.8%
YAW	0.6	0.03	95.0%

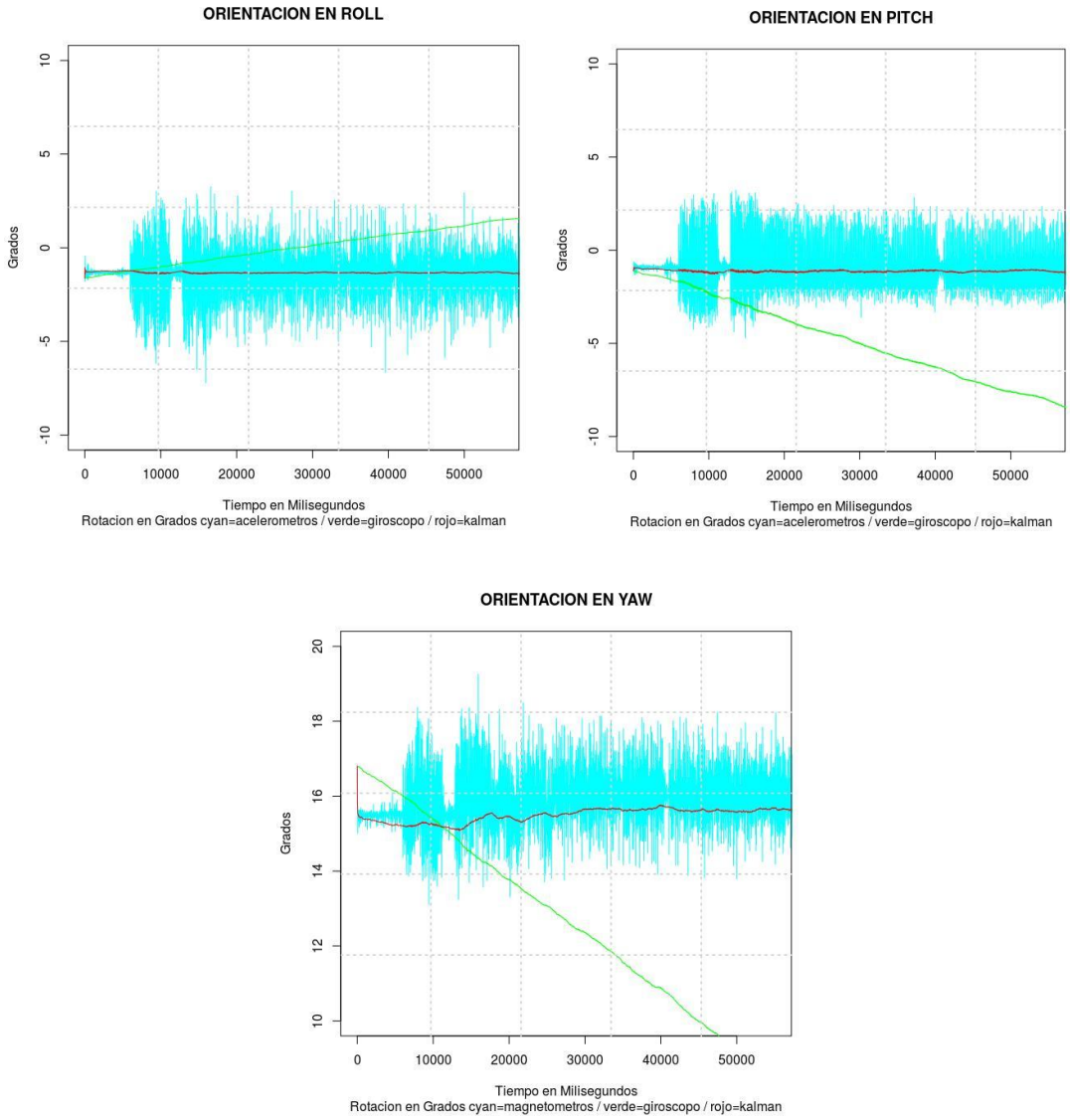


Figura 26. Comparación de ángulo calculado con acelerómetros (azul), giróscopos (verde) y su resultado después de aplicar el Filtro de Kalman (rojo), IMU en reposo.

Posterior a esta prueba, se desarrollaron pruebas adicionales, sometiendo la IMU a diferentes inclinaciones y orientaciones, realizando movimientos rápidos y secos que incrementen la varianza de los ángulos calculados con los acelerómetros y magnetómetros, obteniendo resultados satisfactorios similares, los cuales se pueden observar detalladamente en la Figura 27.

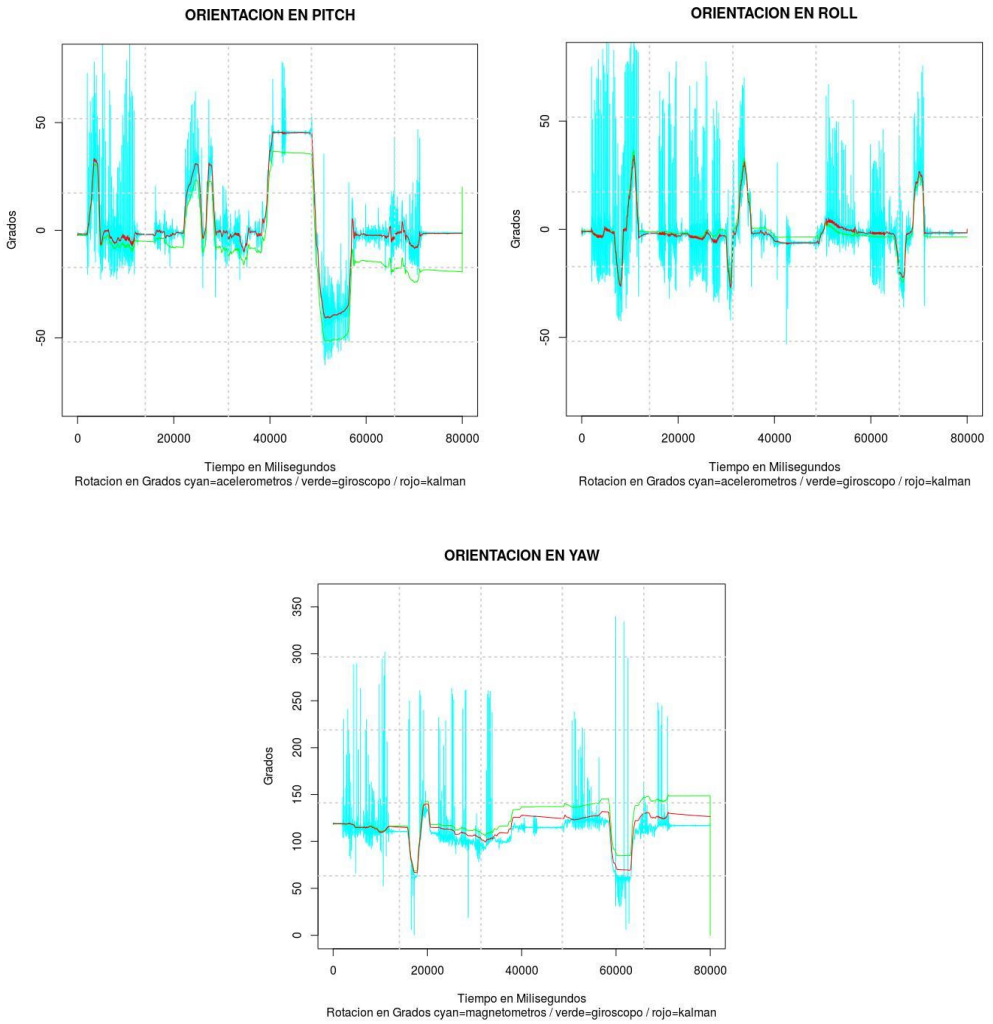


Figura 27. Comparación de ángulos calculados con acelerómetros (azul), giróscopos (verde) y Kalman (rojo), IMU en movimiento.

Pruebas Comparativas de Posición estimada del Casco (IMU casco vs VICON).

Después de comprobar la correcta operación del filtro de Kalman angular implementado, se determinó utilizar el sistema VICON disponible en la Universidad como ground truth, para comparar la inclinación y orientación calculada en tiempo real por el sistema de procesamiento implementado utilizando los sensores de la IMU, con la

posición real del casco medida por el sistema VICON por medios ópticos totalmente independientes.

Para esto, fue necesario realizar un montaje específico y una aplicación en C, que tratara de sincronizar de la mejor forma, el cálculo de inclinación y orientación del sistema de procesamiento implementado y la toma de datos del sistema VICON, para de esta forma determinar el error del ángulo de orientación e inclinación calculado por el prototipo. En la Figura 28, se observan el montaje y las gráficas de algunos de los resultados obtenidos en las pruebas realizadas.

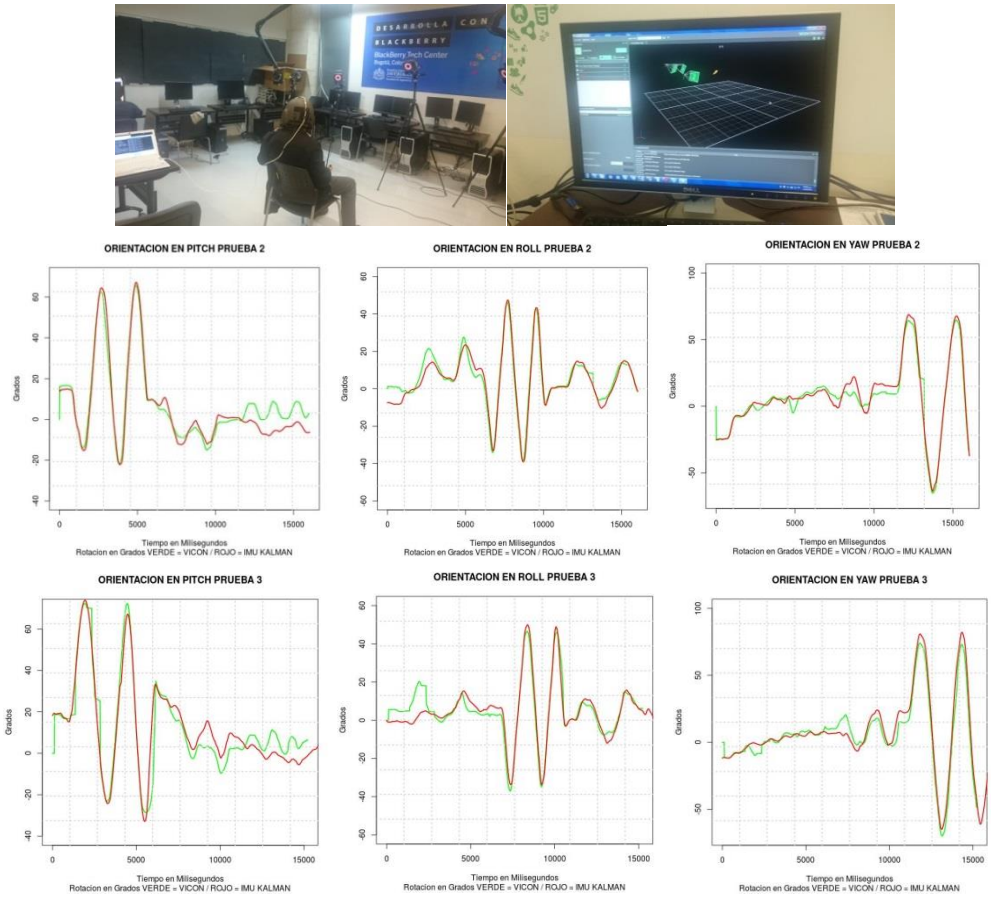


Figura 28. Comparación del ángulo calculado por el sistema de procesamiento utilizando los sensores de la IMU y los datos medidos por el sistema VICON en tiempo real

En la Tabla 9, se cuantifican los resultados de estas pruebas, calculando los valores de error medio cuadrático (MSE) y raíz del error medio cuadrático (RMSE) para cada ángulo estimado.

Tabla 9. Error entre valor estimado por el sistema de procesamiento y valor medido por sistema VICON.

	CARACTERÍSTICA	TEST 1	TEST 2	TEST 3	TEST 4	PROM
	CANTIDAD DE DATOS COMPARADOS	929	878	818	883	877
PITCH	**ERROR CUADRÁTICO MEDIO EN GRADOS ^2 (MSE)**	20.153	24.566	48.073	33.833	31.656
	RAÍZ DEL ERROR CUADRÁTICO MEDIO EN GRADOS (RMSE)	4.489	4.956	6.933	5.8166	5.548
ROLL	**ERROR CUADRÁTICO MEDIO EN GRADOS ^2 (MSE)**	13.347	15.071	19.426	23.544	17.847
	RAÍZ DEL ERROR CUADRÁTICO MEDIO EN GRADOS (RMSE)	3.6533	3.882	4.407	4.8522	4.198
YAW	**ERROR CUADRÁTICO MEDIO EN GRADOS ^2 (MSE)**	65.319	18.008	34.678	137.49	63.873
	RAÍZ DEL ERROR CUADRÁTICO MEDIO EN GRADOS (RMSE)	8.08	4.243	5.888	11.725	7.484

En la Figura 28 y la Tabla 9, se puede observar que el ángulo de orientación e inclinación estimado por el sistema de procesamiento tiene un error promedio (RMSE) entre 4.1º y 7.4º lo cual es aceptable teniendo en cuenta las características de la IMU y la calibración inicial del filtro de kalman; de igual forma, los errores de más de 20º observados en las Figura 28, principalmente se deben a la pérdida de sincronía entre la toma de datos de la IMU y el sistema VICON por problemas en la estimación realizada por el sistema VICON, este problema se superó más adelante, al realizar la comparación directa de las líneas de vista del casco y de la plataforma únicamente con las estimaciones realizadas por el sistema VICON.

Pruebas de Comparación de Lineas de Vista del Casco y Plataforma Seguidora.

Esta prueba se realiza con el fin de comprobar que la línea de vista del usuario del casco y la línea de vista de la plataforma seguidora de L.o.S, se cruzan en un punto en el espacio a una distancia determinada, encontrando de esta forma, cual es el error de la línea de vista replicada por la plataforma en 9 puntos de control específicos.

Para realizar esta prueba, se utilizaron los elementos mencionados en la Tabla 10 y se siguió el protocolo de pruebas explicado a continuación, el cual inicia la operación del prototipo sobre un punto central al cual están orientadas ambas líneas de vista, y posterior a esto, toma medidas de distancia radial de forma manual en 9 puntos de control determinados sobre un tablero de objetivos.

Tabla 10. Elementos utilizados para prueba de comparación de líneas de vista L.o.S. Casco Vs Plataforma Seguidora

ÍTEM	***DESCRIPCIÓN***	***GRAFICA***
1	***Plataforma seguidora*** de línea de vista del casco con movimientos automáticos en pitch y yaw controlada por el sistema de procesamiento, soportada sobre un trípode que permite realizar ajustes manuales en pitch, roll y yaw, con láser rojo que proyecta su orientación replicando la del casco del usuario.	
2	***Casco*** utilizado por el usuario, el cual tiene fijo a su estructura una Unidad de Medida Inercial IMU, y 01 laser rojo que indica la línea de vista del usuario.	

3	***Tablero de objetivos*** con 9 puntos de referencia, sobre los cuales se determinará el error existente entre la línea de vista indicada por la plataforma seguidora y la del casco utilizado por el usuario de forma manual, cada punto de referencia tiene dos círculos de 5 y 10 centímetros de radio.	

Protocolo de pruebas para comparar las L.o.S del casco y de la plataforma seguidora.

1. Se realiza la calibración del sistema, ubicando el punto central del tablero a una distancia determinada, que para este caso son 5 metros, al mismo nivel de la línea de vista del usuario del casco (Pitch = cero grados), para esto se utiliza el láser fijo al casco y la medición de Pitch obtenida con la IMU.
2. Se termina la calibración del sistema, orientando la plataforma seguidora en posición neutral de los motores (pitch y yaw = 0 Grados) haciendo coincidir el láser que proyecta su orientación con el punto central del tablero de objetivos.
3. Garantizando las condiciones anteriores en las cuales se hace coincidir el punto central del tablero de objetivos con la línea de vista de la plataforma seguidora en posición neutral y la línea de vista del usuario del casco, se inicia la ejecución del programa que controla la plataforma para seguir la línea de vista del usuario del casco.
4. El usuario del casco, realiza el movimiento de su línea de vista por cada uno de los 9 puntos de control del tablero de objetivos utilizando el láser fijo al casco para centrarlo en cada punto y se realiza una medición manual para determinar la distancia radial a la cual se orienta el láser de la plataforma seguidora para cada punto, encontrando de esta forma 9 datos de error por cada prueba.

Este protocolo de pruebas fue repetido en 8 ocasiones obteniendo los resultados observados en la Tabla 11, y su ejecución puede ser observada en los siguientes videos (https://www.youtube.com/watch?v=ThyC0K9Nm8E,https://www.youtube.com/watch?v=vfhu5LJbi7A).

Tabla 11. Error radial entre L.o.S del casco y la plataforma seguidora a una distancia de 5 metros.

PUNTO	ERROR RADIAL MEDIDO EN CENTÍMETROS								
	TEST 1	TEST 2	TEST 3	TEST 4	TEST 5	TEST 6	TEST 7	TEST 8	PROMEDIO
1	5.5	8	8.5	8	4.9	6.3	5.5	8.3	6.88
2	2.3	4	3.5	3.9	5.1	5.1	3	4.3	3.90
3	8	7.5	8	7.5	7	7.3	8.1	7	7.55
4	1	1.5	1.7	3	6.2	3.5	4.8	5.2	3.36
5	1.2	1.5	1	1.4	2	0.5	4.1	1.5	1.65
6	1.3	2.1	4.2	5	4.3	1.5	2	1	2.68
7	1.5	6.3	4.5	4	5.3	4.3	2.5	3.2	3.95
8	3.8	5.9	5.5	5.4	5.1	5.9	1	1.5	4.26
9	4	3.5	5.5	4	4.5	6.3	5.4	4.9	4.76
ERROR PROMEDIO EN TODOS LOS PUNTOS DE CONTROL									**4.33**

Las mediciones observadas en la Tabla 11, muestran un error promedio en la línea de vista adoptada por la plataforma seguidora de 4.3 centímetros a una distancia de 5 metros entre el casco y el tablero de objetivos, lo cual transformado a grados teniendo en cuenta las características del tablero de objetivos, es un error menor a un grado, logrando de esta forma, un error aceptable de acuerdo a los parámetros de diseño, establecidos para esta aplicación.

Pruebas de Operación del Prototipo con Sistema de Referencia Fijo.

La siguiente prueba, fue realizada con el fin de determinar el error y el retardo en la operación del prototipo implementado, teniendo una referencia fija. Para esto, se utilizaron exclusivamente las mediciones tomadas por el sistema VICON, con el fin de

comparar la orientación e inclinación teórica que debe tomar la plataforma seguidora de L.o.S. para replicar la línea de vista del usuario del casco, de acuerdo a la orientación e inclinación real del casco del usuario del prototipo; con la orientación e inclinación real, tomada por la plataforma seguidora de L.o.S en tiempo real.

Figura 29. Montaje de pruebas de funcionamiento del prototipo con referencia fija.

Para realizar esta prueba, se realizó un montaje como el de la Figura 29, el cual basado en la Figura 23, presentaba una distancia al plano de seguimiento de L.o.S. *(L)* de 5 metros, con distancias *l* y *d* de 30 cm, e iniciaba su operación y captura de datos del sistema VICON, con las líneas de vista del usuario del casco y de la plataforma seguidora de LoS, alineadas en un punto demarcado frente al usuario del casco, obteniendo los resultados observados en la Figura 30.

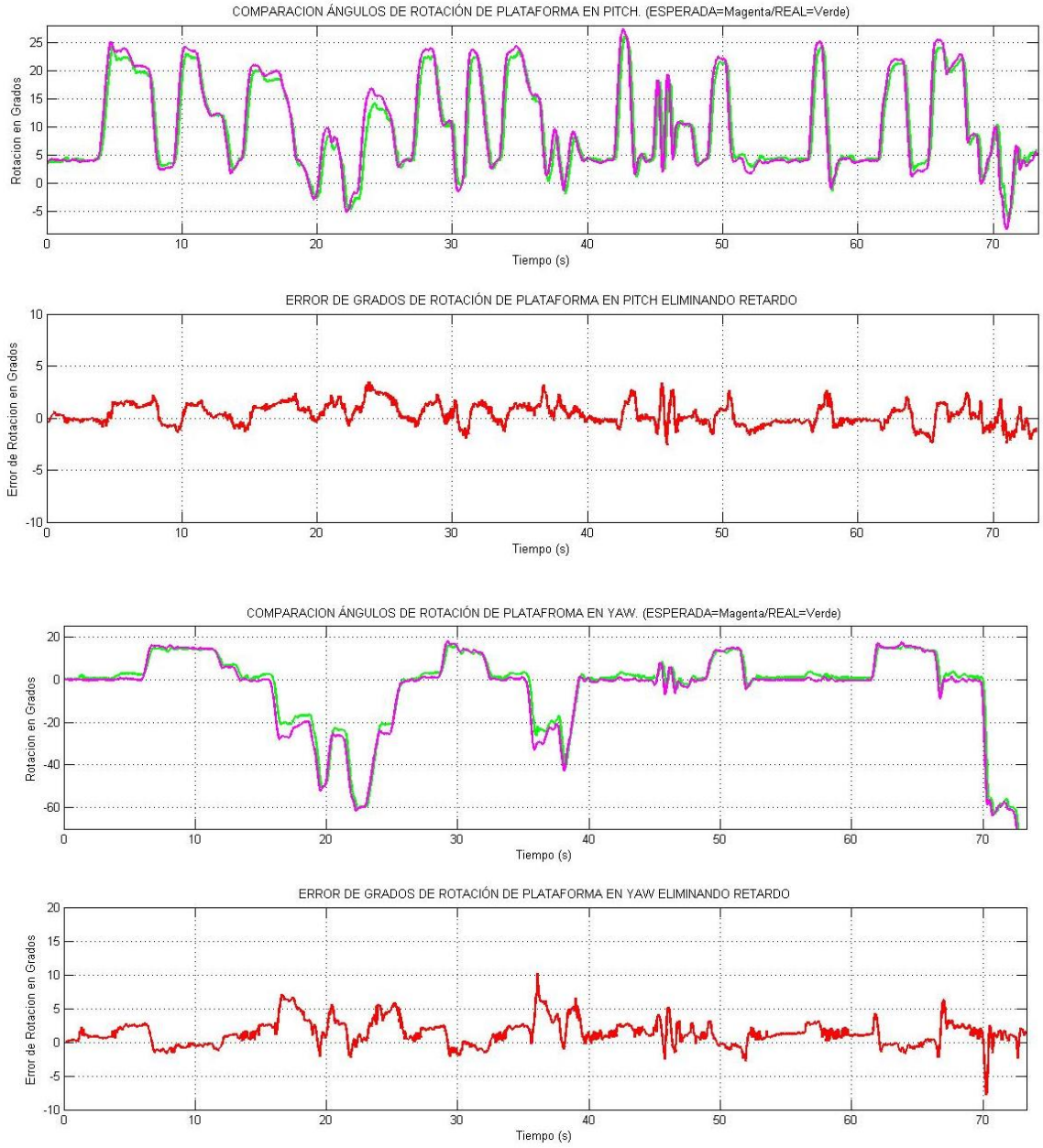


Figura 30. Pruebas de operación del prototipo con una referencia fija

En los resultados de la Figura 30, se puede determinar que el error en los ángulos de orientación en Yaw e inclinación en Pitch, tienen media cercana a cero, con raíz del error cuadrático medio (RMSE) de 1.06 Grados en PITCH y 2.25 grados en YAW, y un retardo de 135 milisegundos en promedio, el cual es el tiempo requerido por el sistema de

procesamiento del prototipo para realizar la estimación y ajustar la posición de la plataforma seguidora a una posición determinada, como se observa en la Figura 31, donde se realiza una ampliación a dos tramos de tiempo específicos de las pruebas anteriores. Es necesario aclarar que los valores de error encontrados, fueron calculados eliminando el retardo, igualando en el tiempo la señal teórica y la real.

La ejecución de esta prueba puede observarse en le siguiente video: https://www.youtube.com/watch?v=ky4VRg0yEok.

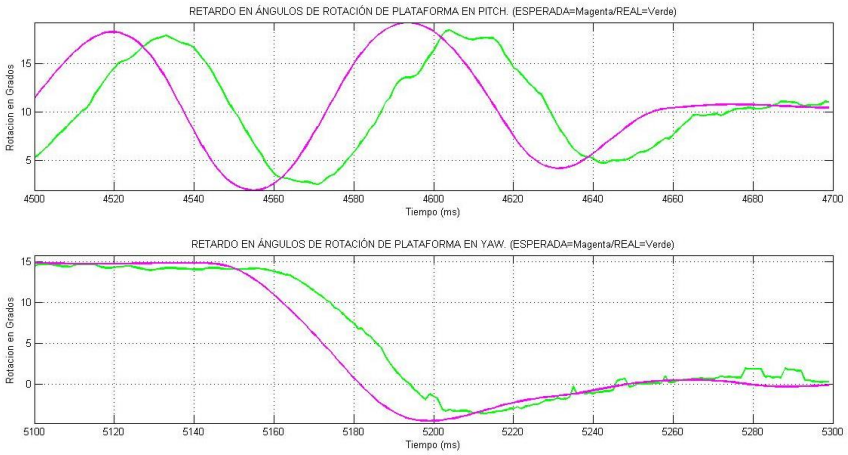


Figura 31. Retardo del prototipo en condiciones normales de operación.

Pruebas de Operación del Prototipo con Sistema de Referencia Móvil.

Por último, se realizó una prueba con el fin de determinar la afectación del cálculo de la diferencia entre los valores de orientación e inclinación de la IMU del casco y la IMU de referencia, girando las dos IMU de forma simultánea en la misma dirección, obteniendo los resultados presentados en la Figura 32.

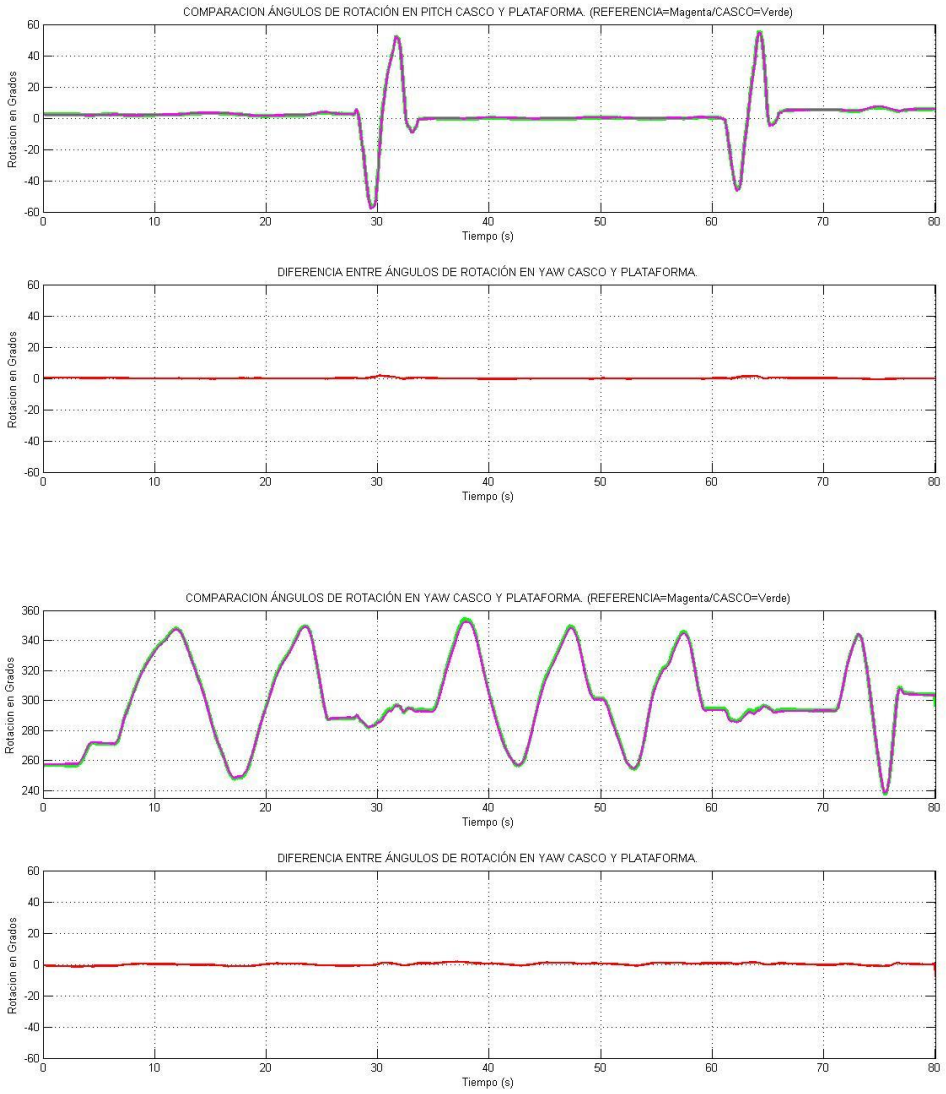


Figura 32. Comparación de ángulos de orientación e inclinación de IMU casco e IMU referencia.

En los resultados de la Figura 32, se puede observar en la parte superior de cada conjunto de gráficas, la comparación de la orientación e inclinación de la IMU del casco (verde) y la IMU de referencia (magenta), observando movimientos casi iguales. Por otra parte, en la parte inferior de ambos conjuntos de gráficas, se observa la diferencia entre los ángulos calculados para la IMU del casco y la IMU de referencia, obteniendo como se esperaba,

una señal constante muy cercana a cero, con una varianza de 0.1253 Grados^2 para el caso de Pitch y 0.5041 Grados^2 para el caso de Yaw.

Capítulo 6

Conclusiones.

En este trabajo se realizó el diseño e implementación del prototipo de un sistema que utiliza una plataforma con dos grados de libertad (Pitch y Yaw), para replicar la dirección de la línea de vista de un individuo en Tiempo Real, a través del sensado de movimiento de su casco, teniendo como referencia un sistema móvil, cumpliendo con los principales requisitos de diseño establecidos, logrando alcanzar algunas características destacadas como lo son: un retardo promedio de 135 milisegundos y raíz del error cuadrático medio (RMSE) de 1.06 Grados en PITCH y 2.25 grados en YAW utilizando componentes de bajo costo.

De igual forma se pudo evidenciar cómo la implementación del filtro de kalman angular para realizar fusión sensorial entre los acelerómetros, magnetómetros y giróscopos, permitió reducir la varianza ocasionada por el ruido de los sensores utilizados, en más del 95%, logrando obtener mediciones de mayor precisión, utilizando unidades de medición inercial de bajo costo.

El concepto desarrollado en la implementación de este prototipo, puede ser extendido y generalizado para realizar aplicaciones de bajo costo en vehículos terrestres o aéreos que utilicen un operario para dirigir la atención de cualquier dispositivo fijo a la plataforma, sobre un blanco o punto en el espacio; por ejemplo la activación de un arma para batir un objetivo, la señalización de un punto a otra plataforma para efectuar un rescate, etc.

Como trabajo futuro, se recomienda evaluar la incidencia de la velocidad máxima de los servomotores utilizados en la plataforma seguidora de L.o.S con el fin de reducir los picos de error observados en la Figura 30, y efectuar pruebas reales del prototipo dentro de una plataforma móvil que puede ser aérea o terrestre para comprobar su funcionalidad en condiciones normales de operación..

Bibliografía

[1] T. Jhonson, «What are survey-accurate visual simulations?,» Buid Media, 7 Septiembre 2015. [En línea]. Available: http://buildmedia.com/what-are-survey-accurate-visual-simulations/. [Último acceso: 22 Marzo 2016].

[2] Rafael - Advanced Defense Systems Ltd., «Eyeball: An Inertial Helmet Mounted Cueing System,» de *Position, Location and Navigation Symposium, 2008 IEEE/ION*, Monterey, CA, 2008.

[3] S. L. -. A. Y. -. M. K. -. M. Antonov, «Head Tracking for the Oculus Rift,» de *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 2014.

[4] T. Oskiper, «Augmented Reality Binoculars,» de *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Adelaide, SA, 2013.

[5] Y. L. E. Foxlin, «FlightTracker: a novel optical/inertial tracker for cockpit enhanced vision,» de *Proceedings of ISMAR'04*, Arlington,VA,USA, Nov 2004.

[6] Y. W. Y. L. Bin Luo, «Sensor Fusion Based Head Pose Tracking for Flight Cockpit System,» de *Image and Signal Processing, 2009. CISP '09. 2nd International Congress on*, Tianjin, 17-19 Oct. 2009.

[7] E. R. Clarence, «Helmet-Mounted Display: Design Issues for Rotary-Wing Aircraft,» de *Chapter 4: Visual Coupling, Tracking Systems*, Bellingham, Washintong, SPIE - The International Society for Optical Engineering, 1999, pp. 76-78.

[8] W. A. Hoff, «Fusion of Data from Head-Mounted and Fixed Sensors,» de *Submitted to the First International Workshop on Augmented Reality*, San Francisco, California., Noviembre 1, 1998.

[9] S. U. H. Y. a. H. T. Kiyohide Satoh, «Robust Vision-Based Registration Utilizing Bird's-Eye View with User's View,» de *Proceedings of ISMAR*, Tokyo, 2003.

[10] E. Foxlin, «Intertial Head-Tracker Sensor Fusion by a Complementary Separate-Bias Kalman Filter,» de *Virtual Reality Annual International Symposium*, Santa Clara, CA, 1996.

[11] M. H. a. Y. A. Eric Foxlin, «Miniature 6-DOF inertial system for tracking HMDs,» *SPIE, Helmet and Head-Mounted Displays III*, vol. 3362, nº 1, pp. 1-15, 1998.

[12] E. Foxlin, «Head-tracking relative to a moving vehicle or simulator platform using differential inertial sensors,» de *Proceedings of Helmet and Head-Mounted Displays V, SPIE Vol. 4021, AeroSense Symposium*, Orlando, FL, Abril 24 de 2000.

[13] N. Maluf, An introduction to microelectromechanical System Engineering, Boston - London: Artech - House, 2000.

[14] A. Papoulis, Probability, Random Variables and Stochastic Processes, Reno, NV, U.S.A.: McGraw Hill, 1984.

[15] Freescale Semiconductor, Inc., «Implementing a Tilt-Compensated eCompass Using Accelerometer and Magnetometer Sensors,» Freescale Semiconductor, Inc., Tempe, Arizona, 2012.

[16] J. L. David L. Hall, HANDBOOK OF MULTISENSOR DATA FUSION, New York: CRC Press, 2001.

[17] A. T. N. A. P. Rudolph van der Merwe, «Portland State University,» 12 Mayo 2014. [En línea]. Available: http://moodle.cecs.pdx.edu/pluginfile.php/3184/mod_resource/content/0/Lecture_Slides/Kalman-Wan.pdf. [Último acceso: 26 Marzo 2016].

[18] R. Kalman, «A New Approach to Linear Filtering and Prediction Problems,» *Transactions of the ASME–Journal of Basic Engineering,* vol. Serie D, nº 82, pp. 35-45, 1960.

[19] G. B. G. Welch, «An Introduction to the Kalman Filter,» Department of Computer Science, University of North Carolina, Chapel Hill, 2006.

[20] G. Ellis, Control System Design Guide, San Diego, California: Elsevier Academic Press, 2004.

[21] R. L. N. a. K. W. Greeley, Helmet-Mounted Display Design Guide, San Marcos, Texas: Crew Systems, 1997.

[22] Department of thr Army, Operator's Manual for UH-60L Helicopters, Washington D.C.: Department of thr Army, 2009.

[23] PHIDGETS, «Accelerometer Primer,» Phidgets, 03 Noviembre 2014. [En línea]. Available: http://www.phidgets.com/docs/Accelerometer_Primer. [Último acceso: 23 Mayo 2016].

[24] PHIDGETS, «Gyroscope Primer,» Phidgets, 02 Julio 2014. [En línea]. Available: http://www.phidgets.com/docs/Gyroscope_Primer. [Último acceso: 23 Mayo 2016].

[25] PHIDGETS, «Compass Primer,» Phidget, 14 Octubre 2014. [En línea]. Available: http://www.phidgets.com/docs/Compass_Primer. [Último acceso: 22 Mayo 2016].

[26] PHIDGETS, «Phidgets products for USB Sensing and Control - OS - Linux,» PHIDGETS, 6 Mayo 2014. [En línea]. Available: http://www.phidgets.com/docs/OS_-_Linux#Quick_Downloads. [Último acceso: 14 Mayo 2016].

[27] PHIDGETS, «1061 user guide,» PHIDGETS, 19 Abril 2016. [En línea]. Available: http://www.phidgets.com/docs/1061_User_Guide. [Último acceso: 2016 Mayo 15].

[28] C. E. Rash, Helmet Mounted Displays: Design Issues for Rotary-wing Aircraft, Bellingham, Washington: SPIE PRESS, 1999.

[29] E. T. R. a. M. J. Tovee, «Processing speed in the cerebral cortexand the neurophysiology of visual masking,» *Proceedings of the royal society B,* vol. I, nº 257, pp. 9-15, 1994.

Anexos

1. Link para observar videos de pruebas realizadas.
2. Aplicación en C Desarrollada.

1. LINK PARA OBSERVAR VIDEOS DE LAS PRUEBAS REALIZADAS.

- ***PRUEBA 1 (6 de Marzo de 2016).*** Este video realiza dos tomas con planos cerrados, mostrando en primer lugar como la plataforma seguidora de L.o.S. replica los movimientos del L.o.S. del casco y en segundo lugar compara sobre un plano de objetivos ubicado a 5 metros del casco, el láser que indica el L.o.S del casco y el láser que indica el L.o.S de la plataforma logrando un error máximo sobre los puntos de control de 5 cm. Esta prueba utilizó un sistema de referencia fijo.

  https://www.youtube.com/watch?v=ThyC0K9Nm8E

- ***PRUEBA 2 (6 de Mayo de 2016).*** Este video realiza una toma con un plano abierto, que muestra de forma simultánea el casco, la plataforma y el plano de objetivos sobre el cual se proyecta el L.o.S del casco y de la plataforma que se encuentra a una distancia de 5 metros, de igual forma se adicionan círculos de 5 centímetros de radio en los puntos de control, con el fin de establecer un parámetro para evaluar los resultados. Esta prueba utilizó un sistema de referencia fijo.

  https://www.youtube.com/watch?v=vfhu5LJbi7A

- ***PRUEBA 3 (8 de Mayo de 2016).*** Este video muestra la operación del prototipo en las dos condiciones probables, primero replicando los movimientos del casco utilizando un sistema de referencia fija y luego manteniendo la L.o.S del casco compensando con los movimientos requeridos al mover el sistema de referencia.

  https://www.youtube.com/watch?v=XljK9L-kZRo

- ***Pruebas prototipo con VICON. Referencia Fija (8 de Abril de 2016).*** Este video muestra de forma simultánea los movimientos del L.o.S del casco y de la plataforma, y los compara en tiempo real con los datos obtenidos con el sistema VICON.

  https://www.youtube.com/watch?v=ky4VRg0yEok

2. APLICACIÓN EN “C” DESARROLLADA.

```
1. /*
2.
   ================================================================
   ==============
3.  Name        : tesis.c
4.  Author      : Gabriel Andres Martin Velandia
5.  Version     : 1.0
6.  Copyright   : Your copyright notice
7.  Description : Toma de datos IMU PHIDGET, Ansi-style
8.
   ================================================================
   ==============
9.  */
10.
11.   //librerias requeridas
12.   #include <stdio.h>
13.   #include <stdlib.h>
14.   #include <math.h>
15.   #include <phidget21.h>
16.
17.   //Constantes utilizadas
18.   #define PI 3.141595
19.   #define LY  0.23
                  // Distancia de plataforma a casco sobre eje Y
20.   #define LZ  0.27
                  // Distancia de plataforma a casco sobre eje Z
21.   #define L  3.8
                        // Distancia de casco a blanco sobre eje Y
22.   #define FACT_YAW  L/(sqrt(((LY+L)*(LY+L))+(LZ*LZ)))
           // factor de ajuste en YAW
23.   #define PI 3.141595
24.   #define MAX 10000
                        // Cantidad de datos a almcenar
25.   #define MUESTREO 8.0
                  // Muestreo de IMU en milisegundos
26.   //VARIABLES DE CASCO PARA EL FILTRO DE KALMAN CASCO
27.   #define VAR_P_ACEL 0.00000102
                  // Varianza del proceso para acelerometros
28.   #define VAR_P_GYRO 10.0
                        // Varianza del proceso para giroscopos
29.   #define VAR_M_ACEL 0.009364611
                  // Coovarianza de la muestra para acelerometros
30.   #define VAR_M_GYRO 0.0412754
                  // Coovarianza de la muestra para giroscopos
31.   #define VAR_P_MAG 0.00000102
                  // Varianza del proceso para YAW
```

```
32.   #define VAR_P_GYROM 10.0
                  // Varianza del proceso para giroscopos en YAW
33.   #define VAR_M_MAG 0.03351852
                  // Coovarianza de la muestra para magnetometros
34.   #define VAR_M_GYROM 0.04983274
                  // Coovarianza de la muestra para giroscopos en
   YAW
35.
36.   //VARIABLES DE LA REFERENCIA PARA EL FILTRO DE KALMAN
   REFERENCIA
37.   #define VAR_P_ACEL_R 0.00000102
           // Varianza del proceso para acelerometros
38.   #define VAR_P_GYRO_R 10.0
                  // Varianza del proceso para giroscopos
39.   #define VAR_M_ACEL_R 0.009364611
           // Coovarianza de la muestra para acelerometros
40.   #define VAR_M_GYRO_R 0.0412754
                  // Coovarianza de la muestra para giroscopos
41.   #define VAR_P_MAG_R 0.00000102
                  // Varianza del proceso para YAW
42.   #define VAR_P_GYROM_R 10.0
                  // Varianza del proceso para giroscopos en YAW
43.   #define VAR_M_MAG_R 0.03351852
                  // Coovarianza de la muestra para magnetometros
44.   #define VAR_M_GYROM_R 0.04983274
           // Coovarianza de la muestra para giroscopos en YAW
45.
46.
47.   //VARIABLE PARA DEFINIR VALOR INICIAL DE PITCH Y YAW
48.   float yaw_inicial;
49.   float yaw_inicialr;
50.   float delta_yaw_ini;
51.   float pitch_inicial;
52.   float pitch_inicialr;
53.   float delta_pitch_ini;
54.   float yaw_platform[MAX];
55.   float pitch_platform[MAX];
56.
57.   //VARIALBES DEL FILTRO DE KALMAN PITCH DEL CASCO
58.   // Matriz de coovarianza del proceso (MATRIZ Q)
59.   float Q11=VAR_P_ACEL, Q12, Q21, Q22=VAR_P_GYRO;
60.   // Matriz de coovarianza de las mediciones (MATRIZ R)
61.   float R11=VAR_M_ACEL, R12, R21, R22=VAR_M_GYRO;
62.   //Matriz de coovarianza de la estimacion a priori (Pminus)
63.   float P11minus, P12minus, P21minus, P22minus;
64.   //Matriz de coovarianza de la estimacion a posteriori (P)
65.   float P11[MAX], P12[MAX], P21[MAX], P22[MAX];
66.   //Ganancia de KALMAN (MATRIZ K)
67.   float k11num, k11den1, k11den2, k11;
```

```
68.   float k12num, k12den1, k12den2, k12;
69.   float k21num, k21den1, k21den2, k21;
70.   float k22num, k22den1, k22den2, k22;
71.   //Valores anteriores
72.   float titahatminus, titapointhatminus;
73.   //Valores Futuros
74.   float titahat[MAX],   titapointhat[MAX];
75.
76.   //VARIALBES DEL FILTRO DE KALMAN ROLL CASCO
77.   // Matriz de coovarianza del proceso (MATRIZ QR)
78.   float QR11=VAR_P_ACEL, QR12, QR21, QR22=VAR_P_GYRO;
79.   // Matriz de coovarianza de las mediciones (MATRIZ RR)
80.   float RR11=VAR_M_ACEL, RR12, RR21, RR22=VAR_M_GYRO;
81.   //Matriz de coovarianza de la estimacion a priori (PRminus)
82.   float PR11minus, PR12minus, PR21minus, PR22minus;
83.   //Matriz de coovarianza de la estimacion a posteriori (PR)
84.   float PR11[MAX], PR12[MAX], PR21[MAX], PR22[MAX];
85.   //Ganancia de KALMAN (MATRIZ Kr)
86.   float kr11num, kr11den1, kr11den2, kr11;
87.   float kr12num, kr12den1, kr12den2, kr12;
88.   float kr21num, kr21den1, kr21den2, kr21;
89.   float kr22num, kr22den1, kr22den2, kr22;
90.   //Valores anteriores
91.   float rollhatminus, rollpointhatminus;
92.   //Valores Futuros
93.   float rollhat[MAX],   rollpointhat[MAX];
94.
95.   //VARIALBES DEL FILTRO DE KALMAN YAW CASCO
96.   // Matriz de coovarianza del proceso (MATRIZ QY)
97.   float QY11=VAR_P_MAG, QY12, QY21, QY22=VAR_P_GYROM;
98.   // Matriz de coovarianza de las mediciones (MATRIZ RY)
99.   float RY11=VAR_M_MAG, RY12, RY21, RY22=VAR_M_GYROM;
100.  //Matriz de coovarianza de la estimacion a priori (PYminus)
101.  float PY11minus, PY12minus, PY21minus, PY22minus;
102.  //Matriz de coovarianza de la estimacion a posteriori (PY)
103.  float PY11[MAX], PY12[MAX], PY21[MAX], PY22[MAX];
104.  //Ganancia de KALMAN (MATRIZ Kr)
105.  float ky11num, ky11den1, ky11den2, ky11;
106.  float ky12num, ky12den1, ky12den2, ky12;
107.  float ky21num, ky21den1, ky21den2, ky21;
108.  float ky22num, ky22den1, ky22den2, ky22;
109.  //Valores anteriores
110.  float yawhatminus, yawpointhatminus;
111.  //Valores Futuros
112.  float yawhat[MAX],    yawpointhat[MAX];
113.
114.  //VARIALBES DEL FILTRO DE KALMAN PITCH DEL REFERENCIA
115.  // Matriz de coovarianza del proceso (MATRIZ Q)
116.  float Q11r=VAR_P_ACEL_R, Q12r, Q21r, Q22r=VAR_P_GYRO_R;
```

```
117.  // Matriz de coovarianza de las mediciones (MATRIZ R)
118.  float R11r=VAR_M_ACEL_R, R12r, R21r, R22r=VAR_M_GYRO_R;
119.  //Matriz de coovarianza de la estimacion a priori (Pminus)
120.  float P11minusr, P12minusr, P21minusr, P22minusr;
121.  //Matriz de coovarianza de la estimacion a posteriori (P)
122.  float P11r[MAX], P12r[MAX], P21r[MAX], P22r[MAX];
123.  //Ganancia de KALMAN (MATRIZ K)
124.  float k11numr, k11den1r, k11den2r, k11r;
125.  float k12numr, k12den1r, k12den2r, k12r;
126.  float k21numr, k21den1r, k21den2r, k21r;
127.  float k22numr, k22den1r, k22den2r, k22r;
128.  //Valores anteriores
129.  float titahatminusr, titapointhatminusr;
130.  //Valores Futuros
131.  float titahatr[MAX],  titapointhatr[MAX];
132.
133.  //VARIALBES DEL FILTRO DE KALMAN ROLL REFERENCIA
134.  // Matriz de coovarianza del proceso (MATRIZ QR)
135.  float QR11r=VAR_P_ACEL_R, QR12r, QR21r, QR22r=VAR_P_GYRO_R;
136.  // Matriz de coovarianza de las mediciones (MATRIZ RR)
137.  float RR11r=VAR_M_ACEL_R, RR12r, RR21r, RR22r=VAR_M_GYRO_R;
138.  //Matriz de coovarianza de la estimacion a priori (PRminus)
139.  float PR11minusr, PR12minusr, PR21minusr, PR22minusr;
140.  //Matriz de coovarianza de la estimacion a posteriori (PR)
141.  float PR11r[MAX], PR12r[MAX], PR21r[MAX], PR22r[MAX];
142.  //Ganancia de KALMAN (MATRIZ Kr)
143.  float kr11numr, kr11den1r, kr11den2r, kr11r;
144.  float kr12numr, kr12den1r, kr12den2r, kr12r;
145.  float kr21numr, kr21den1r, kr21den2r, kr21r;
146.  float kr22numr, kr22den1r, kr22den2r, kr22r;
147.  //Valores anteriores
148.  float rollhatminusr, rollpointhatminusr;
149.  //Valores Futuros
150.  float rollhatr[MAX],  rollpointhatr[MAX];
151.
152.  //VARIALBES DEL FILTRO DE KALMAN YAW REFERENCIA
153.  // Matriz de coovarianza del proceso (MATRIZ QY)
154.  float QY11r=VAR_P_MAG_R, QY12r, QY21r, QY22r=VAR_P_GYROM_R;
155.  // Matriz de coovarianza de las mediciones (MATRIZ RY)
156.  float RY11r=VAR_M_MAG_R, RY12r, RY21r, RY22r=VAR_M_GYROM_R;
157.  //Matriz de coovarianza de la estimacion a priori (PYminus)
158.  float PY11minusr, PY12minusr, PY21minusr, PY22minusr;
159.  //Matriz de coovarianza de la estimacion a posteriori (PY)
160.  float PY11r[MAX], PY12r[MAX], PY21r[MAX], PY22r[MAX];
161.  //Ganancia de KALMAN (MATRIZ Kr)
162.  float ky11numr, ky11den1r, ky11den2r, ky11r;
163.  float ky12numr, ky12den1r, ky12den2r, ky12r;
164.  float ky21numr, ky21den1r, ky21den2r, ky21r;
165.  float ky22numr, ky22den1r, ky22den2r, ky22r;
```

```
166.  //Valores anteriores
167.  float yawhatminusr, yawpointhatminusr;
168.  //Valores Futuros
169.  float yawhatr[MAX],   yawpointhatr[MAX];
170.
171.  #ifndef WIN32
172.  #define __stdcall
173.  #define __stdcall1
174.  #endif
175.
176.  //Variables para registro de datos calculados
177.  float pitch[MAX], roll[MAX], yaw[MAX];
178.  float pitchw[MAX], rollw[MAX], yaww[MAX];
179.  float pitchr[MAX], rollr[MAX], yawr[MAX];
180.  float pitchwr[MAX], rollwr[MAX], yawwr[MAX];
181.  int j=0,k=0;
182.
183.  // Organizar datos adquiridos del casco para escribirlos en
   archivo *.csv
184.  int __stdcall SpatialDataHandler(CPhidgetSpatialHandle
   casco, void *userptr, CPhidgetSpatial_SpatialEventDataHandle
   *data, int count)
185.  {
186.      int i;
187.      //wx_offset=-1.134251, wy_offset=0.3272989,
   wz_offset=0.2664459;
188.      float ax[MAX], ay[MAX], az[MAX], wx[MAX], wy[MAX],
   wz[MAX], time[MAX];
189.      float roff=0, poff=0, thetay, phix, t_imuinit,
   wx_offset=-0.9029512, wy_offset=0.258547, wz_offset=0.3263491;
190.      float mx[MAX], my[MAX], mz[MAX];
191.      if(k>j){
192.      printf("R=%f P=%f Y=%f Delta = %d Datos = %d P ini=%f Y
   ini= %f P Plat=%f Y Plat=%f\r",rollhat[j-1], titahat[j-
   1],yawhat[j-1],k-
   j,k,delta_pitch_ini,delta_yaw_ini,pitch_platform[j-
   1],yaw_platform[j-1]);
193.      }
194.      else{
195.        printf("R=%f P=%f Y=%f     Delta = %d Datos = %d P
   ini=%f Y ini= %f P Plat=%f Y Plat=%f\r",rollhat[k-1],
   titahat[k-1],yawhat[k-1],k-
   j,k,delta_pitch_ini,delta_yaw_ini,pitch_platform[k-
   1],yaw_platform[k-1]);
196.        }
197.      fflush(stdout);
198.      // Para el caso que se obtenga mas de un paquete por
   evento
199.             for(i = 0; i < count; i++) {
```

```
200.                  FILE *file = (FILE *) userptr;
201.                  // Encabezado del archivo
202.                  //
   Time,A_X,A_Y,A_Z,Rolla,Pitcha,Yawa,W_X,W_Y,W_Z,Rollw,Pitchw,Ya
   ww,M_X,M_Y,M_Z,Pitchk,Rollk,Yawk
203.
204.                  if (j==1){
205.                       //Se define el tiempo de inicio de
   toma de datos t_imuinit
206.                       t_imuinit=(data[i]-
   >timestamp.seconds*1000.0 + (data[i]-
   >timestamp.microseconds)/1000.0);
207.                  }
208.
209.                  //Se definen las variables de interes
   tiempo, axyz, wxyz
210.                  time[j]=(data[i]->timestamp.seconds*1000.0
   + (data[i]->timestamp.microseconds)/1000.0)-t_imuinit;
211.                  ax[j]=(data[i]->acceleration[0]);
212.                  ay[j]=(data[i]->acceleration[1]);
213.                  az[j]=(data[i]->acceleration[2]);
214.                  wx[j]=data[i]->angularRate[0]-wx_offset;
215.                  wy[j]=data[i]->angularRate[1]-wy_offset;
216.                  wz[j]=data[i]->angularRate[2]-wz_offset;
217.
218.                  //Calculo de pitch y roll con acelerometro
219.
     pitch[j]=asin(ay[j]/sqrt(pow(ay[j],2)+pow(az[j],2)));
220.
     roll[j]=asin(ax[j]/sqrt(pow(ax[j],2)+pow(az[j],2)));
221.
222.
223.                  //Se filtran los datos de magnetometro, si
   son PUNK_DBL almacenan el dato anterior
224.                  if (data[i]->magneticField[0] == PUNK_DBL)
   {
225.                                       mx[j]=mx[j-1];
226.                                       my[j]=my[j-1];
227.                                       mz[j]=mz[j-1];
228.                                            } else {
229.                                       mx[j]=data[i]-
   >magneticField[0];
230.                                       my[j]=data[i]-
   >magneticField[1];
231.                                       mz[j]=data[i]-
   >magneticField[2];
232.                                       }
233.
```

```
234.                  //Calculo de yaw con magentometro y
   acelerometro
235.                  phix=atan2(ay[j],az[j]);
236.                  thetay=atan(-
   ax[j]/((ay[j]*sin(phix))+(az[j]*cos(phix))));
237.
238.
239.                  //calcula valor de YAW
240.                  yaw[j]=atan2((mz[j]*sin(phix))-
   (my[j]*cos(phix)),(mx[j]*cos(thetay))+(my[j]*sin(thetay)*sin(p
   hix))+(mz[j]*sin(thetay)*cos(phix)));
241.
242.                  //Conversion a grados
243.                  pitch[j]*=(180/PI)+poff;
244.                  roll[j]*=(180/PI)+roff;
245.                  yaw[j]*=(180/PI);
246.
247.                  //Ajuste de YAW entre 0 y 360 grados
248.                  if (yaw[j]<0){
249.                        yaw[j]=yaw[j]+360;
250.                  }
251.
252.                  //Calculo de PITCH, ROLL y YAW en base a
   giroscopo utilizando metodo de integracion
253.                  if (j==1){
254.                        pitchw[j]=pitch[j];
255.                        rollw[j]=roll[j];
256.                        yaww[j]=yaw[j];
257.                  }
258.                  else{
259.                        pitchw[j]=pitchw[j-
   1]+wx[j]*(MUESTREO/1000.0);
260.                        rollw[j]=rollw[j-1]-
   wy[j]*(MUESTREO/1000.0);
261.                        yaww[j]=yaww[j-
   1]+wz[j]*(MUESTREO/1000.0);
262.                  }
263.
264.                  //Condiciones iniciales del filtro de
   KALMAN para PITCH, ROLL y YAW del casco
265.                  if (j==1){
266.                        //PITCH
267.                        titahat[j] =pitch[j];
268.                        pitch_inicial=titahat[j];
269.                        titapointhat[j]=wx[j];
270.
     P11[j]=0.5;P12[j]=0.0;P21[j]=0.0;P22[j]=0.5;
271.                        //ROLL
272.                        rollhat[j] =roll[j];
```

```
273.                        rollpointhat[j]=-wy[j];
274.
     PR11[j]=0.5;PR12[j]=0.0;PR21[j]=0.0;PR22[j]=0.5;
275.                        //YAW
276.                        yawhat[j] =yaw[j];
277.                        yaw_inicial=yawhat[j];
278.                        yawpointhat[j]=wz[j];
279.
     PY11[j]=0.5;PY12[j]=0.0;PY21[j]=0.0;PY22[j]=0.5;
280.
281.                    }
282.                    else{
283.                    //filtro de KALMAN para PITCH del casco
284.                    titahatminus = titahat[j-1] +
   (MUESTREO/1000.0)*titapointhat[j-1];
285.                    titapointhatminus = titapointhat[j-1];
286.
     /*******************************************************/
287.                    P11minus = P11[j-1] +
   (MUESTREO/1000.0)*P21[j-1] + Q11;
288.                    P12minus = P12[j-1] +
   (MUESTREO/1000.0)*P22[j-1] - (MUESTREO/1000.0)*(P11[j-1] +
   (MUESTREO/1000.0)*P21[j-1]) + Q12;
289.                    P21minus = P21[j-1] + Q21;
290.                    P22minus = P22[j-1] - P21[j-
   1]*(MUESTREO/1000.0) + Q22;
291.
     /*******************************************************/
292.                    k11num = P11minus*R22 - P12minus*R21 +
   P11minus*P22minus - P12minus*P21minus;
293.                    k11den1 = P11minus*R22 - P12minus*R21 -
   P21minus*R12 + P22minus*R11;
294.                    k11den2 = R11*R22 - R12*R21 +
   P11minus*P22minus - P12minus*P21minus;
295.                    k11 = k11num/(k11den1 + k11den2);
296.
297.                    k12num = -P11minus*R12 + P12minus*R11;
298.                    k12den1 = P11minus*R22 - P12minus*R21 -
   P21minus*R12 + P22minus*R11 + R11*R22;
299.                    k12den2 = - R12*R21 + P11minus*P22minus -
   P12minus*P21minus;
300.                    k12 = k12num/(k12den1 + k12den2);
301.
302.                    k21num = (P21minus*R22 - P22minus*R21);
303.                    k21den1 = P11minus*R22 - P12minus*R21 -
   P21minus*R12 + P22minus*R11 + R11*R22;
304.                    k21den2 = - R12*R21 + P11minus*P22minus -
   P12minus*P21minus;
305.                    k21 = k21num/(k21den1+k21den2);
```

```
306.
307.                   k22num = -(P21minus*R12 - P22minus*R11 -
   P11minus*P22minus + P12minus*P21minus);
308.                   k22den1 = P11minus*R22 - P12minus*R21 -
   P21minus*R12 + P22minus*R11 + R11*R22;
309.                   k22den2 = - R12*R21 + P11minus*P22minus -
   P12minus*P21minus;
310.                   k22 = k22num/(k22den1+k22den2);
311.
     /*******************************************************/
312.                   titahat[j] = titahatminus -
   k11*(titahatminus - pitch[j]);
313.                   titahat[j] = titahat[j] -
   k12*(titapointhatminus - wx[j]);
314.                   titapointhat[j] = titapointhatminus -
   k21*(titahatminus - pitch[j]);
315.                   titapointhat[j] = titapointhat[j] -
   k22*(titapointhatminus - wx[j]);
316.
     /*******************************************************/
317.                   P11[j] = -P21minus*k12 - P11minus*(k11 -
   1);
318.                   P12[j] = -P22minus*k12 - P12minus*(k11 -
   1);
319.                   P21[j] = -P11minus*k21 - P21minus*(k22 -
   1);
320.                   P22[j] = -P12minus*k21 - P22minus*(k22 -
   1);
321.
322.                   //filtro de KALMAN para ROLL del casco
323.                   rollhatminus = rollhat[j-1] +
   (MUESTREO/1000.0)*rollpointhat[j-1];
324.                   rollpointhatminus = rollpointhat[j-1];
325.
     /*******************************************************/
326.                   PR11minus = PR11[j-1] +
   (MUESTREO/1000.0)*PR21[j-1] + QR11;
327.                   PR12minus = PR12[j-1] +
   (MUESTREO/1000.0)*PR22[j-1] - (MUESTREO/1000.0)*(PR11[j-1] +
   (MUESTREO/1000.0)*PR21[j-1]) + QR12;
328.                   PR21minus = PR21[j-1] + QR21;
329.                   PR22minus = PR22[j-1] - PR21[j-
   1]*(MUESTREO/1000.0) + QR22;
330.
     /*******************************************************/
331.                   kr11num = PR11minus*RR22 - PR12minus*RR21 +
   PR11minus*PR22minus - PR12minus*PR21minus;
332.                   kr11den1 = PR11minus*RR22 - PR12minus*RR21
   - PR21minus*RR12 + PR22minus*RR11;
```

```
333.                  kr11den2 = RR11*RR22 - RR12*RR21 +
   PR11minus*PR22minus - PR12minus*PR21minus;
334.                  kr11 = kr11num/(kr11den1 + kr11den2);
335.
336.                  kr12num = -PR11minus*RR12 + PR12minus*RR11;
337.                  kr12den1 = PR11minus*RR22 - PR12minus*RR21
   - PR21minus*RR12 + PR22minus*RR11 + RR11*RR22;
338.                  kr12den2 = - RR12*RR21 +
   PR11minus*PR22minus - PR12minus*PR21minus;
339.                  kr12 = kr12num/(kr12den1 + kr12den2);
340.
341.                  kr21num = (PR21minus*RR22 -
   PR22minus*RR21);
342.                  kr21den1 = PR11minus*RR22 - PR12minus*RR21
   - PR21minus*RR12 + PR22minus*RR11 + RR11*RR22;
343.                  kr21den2 = - RR12*RR21 +
   PR11minus*PR22minus - PR12minus*PR21minus;
344.                  kr21 = kr21num/(kr21den1+kr21den2);
345.
346.                  kr22num = -(PR21minus*RR12 - PR22minus*RR11
   - PR11minus*PR22minus + PR12minus*PR21minus);
347.                  kr22den1 = PR11minus*RR22 - PR12minus*RR21
   - PR21minus*RR12 + PR22minus*RR11 + RR11*RR22;
348.                  kr22den2 = - RR12*RR21 +
   PR11minus*PR22minus - PR12minus*PR21minus;
349.                  kr22 = kr22num/(kr22den1+kr22den2);
350.
     /*****************************************************/
351.                  rollhat[j] = rollhatminus -
   kr11*(rollhatminus - roll[j]);
352.                  rollhat[j] = rollhat[j] -
   kr12*(rollpointhatminus + wy[j]);
353.                  rollpointhat[j] = rollpointhatminus -
   kr21*(rollhatminus - roll[j]);
354.                  rollpointhat[j] = rollpointhat[j] -
   kr22*(rollpointhatminus + wy[j]);
355.
     /*****************************************************/
356.                  PR11[j] = -PR21minus*kr12 - PR11minus*(kr11
   - 1);
357.                  PR12[j] = -PR22minus*kr12 - PR12minus*(kr11
   - 1);
358.                  PR21[j] = -PR11minus*kr21 - PR21minus*(kr22
   - 1);
359.                  PR22[j] = -PR12minus*kr21 - PR22minus*(kr22
   - 1);
360.
361.                  //filtro de KALMAN para YAW del casco
```

```
362.                    yawhatminus = yawhat[j-1] +
   (MUESTREO/1000.0)*yawpointhat[j-1];
363.                    yawpointhatminus = yawpointhat[j-1];
364.
     /*****************************************************/
365.                    PY11minus = PY11[j-1] +
   (MUESTREO/1000.0)*PY21[j-1] + QY11;
366.                    PY12minus = PY12[j-1] +
   (MUESTREO/1000.0)*PY22[j-1] - (MUESTREO/1000.0)*(PY11[j-1] +
   (MUESTREO/1000.0)*PY21[j-1]) + QY12;
367.                    PY21minus = PY21[j-1] + QY21;
368.                    PY22minus = PY22[j-1] - PY21[j-
   1]*(MUESTREO/1000.0) + QY22;
369.
     /*****************************************************/
370.                    ky11num = PY11minus*RY22 - PY12minus*RY21 +
   PY11minus*PY22minus - PY12minus*PY21minus;
371.                    ky11den1 = PY11minus*RY22 - PY12minus*RY21
   - PY21minus*RY12 + PY22minus*RY11;
372.                    ky11den2 = RY11*RY22 - RY12*RY21 +
   PY11minus*PY22minus - PY12minus*PY21minus;
373.                    ky11 = ky11num/(ky11den1 + ky11den2);
374.
375.                    ky12num = -PY11minus*RY12 + PY12minus*RY11;
376.                    ky12den1 = PY11minus*RY22 - PY12minus*RY21
   - PY21minus*RY12 + PY22minus*RY11 + RY11*RY22;
377.                    ky12den2 = - RY12*RY21 +
   PY11minus*PY22minus - PY12minus*PY21minus;
378.                    ky12 = ky12num/(ky12den1 + ky12den2);
379.
380.                    ky21num = (PY21minus*RY22 -
   PY22minus*RY21);
381.                    ky21den1 = PY11minus*RY22 - PY12minus*RY21
   - PY21minus*RY12 + PY22minus*RY11 + RY11*RY22;
382.                    ky21den2 = - RY12*RY21 +
   PY11minus*PY22minus - PY12minus*PY21minus;
383.                    ky21 = ky21num/(ky21den1+ky21den2);
384.
385.                    ky22num = -(PY21minus*RY12 - PY22minus*RY11
   - PY11minus*PY22minus + PY12minus*PY21minus);
386.                    ky22den1 = PY11minus*RY22 - PY12minus*RY21
   - PY21minus*RY12 + PY22minus*RY11 + RY11*RY22;
387.                    ky22den2 = - RY12*RY21 +
   PY11minus*PY22minus - PY12minus*PY21minus;
388.                    ky22 = ky22num/(ky22den1+ky22den2);
389.
     /*****************************************************/
390.                    yawhat[j] = yawhatminus - ky11*(yawhatminus
   - yaw[j]);
```

```
391.                    yawhat[j] = yawhat[j] -
   ky12*(yawpointhatminus - wz[j]);
392.                    yawpointhat[j] = yawpointhatminus -
   ky21*(yawhatminus - yaw[j]);
393.                    yawpointhat[j] = yawpointhat[j] -
   ky22*(yawpointhatminus - wz[j]);
394.
     /*******************************************************/
395.                    PY11[j] = -PY21minus*ky12 - PY11minus*(ky11
   - 1);
396.                    PY12[j] = -PY22minus*ky12 - PY12minus*(ky11
   - 1);
397.                    PY21[j] = -PY11minus*ky21 - PY21minus*(ky22
   - 1);
398.                    PY22[j] = -PY12minus*ky21 - PY22minus*(ky22
   - 1);
399.
400.                    //Reinicia el filtro de kalman al cambiar
   de 360 a 0 grados
401.                    if(abs(yawhat[j]-yaw[j-1])>350){
402.                         yawhat[j]=yaw[j];
403.                    }
404.
405.                    //Define la Ubicacion de la plataforma
406.                    if(k-j>=0){
407.                    yaw_platform[j]=yawhat[j-1]-delta_yaw_ini-
   yawhatr[j-1];
408.                    pitch_platform[j]=titahat[j-1]-
   delta_pitch_ini-titahatr[j-1];
409.                    }
410.                    else{
411.                    yaw_platform[j]=yawhat[k-1]-delta_yaw_ini-
   yawhatr[k-1];
412.                    pitch_platform[j]=titahat[k-1]-
   delta_pitch_ini-titahatr[k-1];
413.                    }
414.
415.                    if(yaw_platform[j]>180){
416.                    yaw_platform[j]=yaw_platform[j]-360;
417.                    }
418.
419.                    if(yaw_platform[j]<-180){
420.                    yaw_platform[j]=yaw_platform[j]+360;
421.                    }
422.
423.                    }
424.
425.                    //Se imprime el archivo de datos del casco
426.                    if (j>=1){
```

```
427.                    fprintf(file, "%f,", time[j]);
428.                    fprintf(file, "%6f,%6f,%6f,", ax[j], ay[j],
   az[j]);
429.                    fprintf(file, "%6f,%6f,%6f,", roll[j],
   pitch[j], yaw[j]);
430.                    fprintf(file, "%6f,%6f,%6f,", wx[j], wy[j],
   wz[j]);
431.                    fprintf(file, "%6f,%6f,%6f,", rollw[j],
   pitchw[j], yaww[j]);
432.                    fprintf(file, "%6f,%6f,%6f,", mx[j], my[j],
   mz[j]);
433.                    fprintf(file, "%6f,%6f,%6f\n", titahat[j],
   rollhat[j], yawhat[j]);
434.                    fflush(file);
435.                    }
436.                    //Incrementa el puntero
437.                    j++;
438.
439.             }
440.
441.      return 0;
442.  }
443.
444.  // Organizar datos adquiridos de la referencia para
   escribirlos en archivo *.csv
445.  int __stdcall1 SpatialDataHandler1(CPhidgetSpatialHandle
   ref, void *userptr1, CPhidgetSpatial_SpatialEventDataHandle
   *data1, int count1)
446.  {
447.        int i;
448.        //wx_offset=-1.134251, wy_offset=0.3272989,
   wz_offset=0.2664459;
449.        float axr[MAX], ayr[MAX], azr[MAX], wxr[MAX],
   wyr[MAX], wzr[MAX], timer[MAX];
450.        float roffr=0, poffr=0,thetayr, phixr, t_imuinitr,
   wx_offsetr=-0.2748047, wy_offsetr=-0.1107412,
   wz_offsetr=0.3732657;
451.        float mxr[MAX], myr[MAX], mzr[MAX];
452.
453.        //printf("ROLL=%f PITCH=%f YAW=%f    Datos tomados =
   %d\r",rollr[k-1], pitchr[k-1],yawr[k-1],k);
454.        //fflush(stdout);
455.
456.        for(i = 0; i < count1; i++) {
457.                        FILE *file1 = (FILE *) userptr1;
458.                        // Encabezado del archivo
459.                        //
   Time,A_X,A_Y,A_Z,Rolla,Pitcha,Yawa,W_X,W_Y,W_Z,Rollw,Pitchw,Ya
   ww,M_X,M_Y,M_Z,Pitchk,Rollk,Yawk
```

```
460.
461.                         if (k==1){
462.                              //Se define el tiempo de inicio
   de toma de datos t_imuinit
463.                              t_imuinitr=(data1[i]-
   >timestamp.seconds*1000.0 + (data1[i]-
   >timestamp.microseconds)/1000.0);
464.                         }
465.
466.                         //Se definen las variables de interes
   tiempo, axyz, wxyz
467.                         timer[k]=(data1[i]-
   >timestamp.seconds*1000.0 + (data1[i]-
   >timestamp.microseconds)/1000.0)-t_imuinitr;
468.                         axr[k]=(data1[i]->acceleration[0]);
469.                         ayr[k]=(data1[i]->acceleration[1]);
470.                         azr[k]=(data1[i]->acceleration[2]);
471.                         wxr[k]=data1[i]->angularRate[0]-
   wx_offsetr;
472.                         wyr[k]=data1[i]->angularRate[1]-
   wy_offsetr;
473.                         wzr[k]=data1[i]->angularRate[2]-
   wz_offsetr;
474.
475.                         //Calculo de pitch y roll con
   acelerometro
476.
      pitchr[k]=asin(ayr[k]/sqrt(pow(ayr[k],2)+pow(azr[k],2)));
477.
      rollr[k]=asin(axr[k]/sqrt(pow(axr[k],2)+pow(azr[k],2)));
478.
479.                         //Se filtran los datos de
   magnetometro, si son PUNK_DBL almacenan el dato anterior
480.                         if (data1[i]->magneticField[0] ==
   PUNK_DBL) {
481.                                        mxr[k]=mxr[k-1];
482.                                        myr[k]=myr[k-1];
483.                                        mzr[k]=mzr[k-1];
484.                                             } else {
485.                                             mxr[k]=data1[i]-
   >magneticField[0];
486.                                             myr[k]=data1[i]-
   >magneticField[1];
487.                                             mzr[k]=data1[i]-
   >magneticField[2];
488.                                        }
489.
490.                         //Calculo de yaw con magentometro y
   acelerometro
```

```
491.                        phixr=atan2(ayr[k],azr[k]);
492.                        thetayr=atan(-
   axr[k]/((ayr[k]*sin(phixr))+(azr[k]*cos(phixr))));
493.
494.
495.                        //calcula YAW
496.                        yawr[k]=atan2((mzr[k]*sin(phixr))-
   (myr[k]*cos(phixr)),(mxr[k]*cos(thetayr))+(myr[k]*sin(thetayr)
   *sin(phixr))+(mzr[k]*sin(thetayr)*cos(phixr)));
497.
498.                        //Conversion a grados
499.                        pitchr[k]*=(180/PI)+poffr;
500.                        rollr[k]*=(180/PI)+roffr;
501.                        yawr[k]*=(180/PI);
502.
503.                        //Ajuste de YAW entre 0 y 360 grados
504.                        if (yawr[k]<0){
505.                             yawr[k]=yawr[k]+360;
506.                        }
507.
508.                        //Calculo de PITCH, ROLL y YAW en base
   a giroscopo utilizando metodo de integracion
509.                        if (k==1){
510.                             pitchwr[k]=pitchr[k];
511.                             rollwr[k]=rollr[k];
512.                             yawwr[k]=yawr[k];
513.                        }
514.                        else{
515.                             pitchwr[k]=pitchwr[k-
   1]+wxr[k]*(MUESTREO/1000.0);
516.                             rollwr[k]=rollwr[k-1]-
   wyr[k]*(MUESTREO/1000.0);
517.                             yawwr[k]=yawwr[k-
   1]+wzr[k]*(MUESTREO/1000.0);
518.                        }
519.
520.                        //Condiciones iniciales del filtro de
   KALMAN para PITCH, ROLL y YAW de la referencia
521.                        if (k==1){
522.                             //PITCH
523.                             titahatr[k] =pitchr[k];
524.                             pitch_inicialr=titahatr[k];
525.                             titapointhatr[k]=wxr[k];
526.
     P11r[k]=0.5;P12r[k]=0.0;P21r[k]=0.0;P22r[k]=0.5;
527.                             //ROLL
528.                             rollhatr[k] =rollr[k];
529.                             rollpointhatr[k]=-wyr[k];
```

```
530.
     PR11r[k]=0.5;PR12r[k]=0.0;PR21r[k]=0.0;PR22r[k]=0.5;
531.                              //YAW
532.                              yawhatr[k] =yawr[k];
533.                              yaw_inicialr=yawhatr[k];
534.                              yawpointhatr[k]=wzr[k];
535.
     PY11r[k]=0.5;PY12r[k]=0.0;PY21r[k]=0.0;PY22r[k]=0.5;
536.
537.                         }
538.                         else{
539.                         //filtro de KALMAN para PITCH de la
   referencia
540.                         titahatminusr = titahatr[k-1] +
   (MUESTREO/1000.0)*titapointhatr[k-1];
541.                         titapointhatminusr = titapointhatr[k-
   1];
542.
     /*****************************************************/
543.                         P11minusr = P11r[k-1] +
   (MUESTREO/1000.0)*P21r[k-1] + Q11r;
544.                         P12minusr = P12r[k-1] +
   (MUESTREO/1000.0)*P22r[k-1] - (MUESTREO/1000.0)*(P11r[k-1] +
   (MUESTREO/1000.0)*P21r[k-1]) + Q12r;
545.                         P21minusr = P21r[k-1] + Q21r;
546.                         P22minusr = P22r[k-1] - P21r[k-
   1]*(MUESTREO/1000.0) + Q22r;
547.
     /*****************************************************/
548.                         k11numr = P11minusr*R22r -
   P12minusr*R21r + P11minusr*P22minusr - P12minusr*P21minusr;
549.                         k11den1r = P11minusr*R22r -
   P12minusr*R21r - P21minusr*R12r + P22minusr*R11r;
550.                         k11den2r = R11r*R22r - R12r*R21r +
   P11minusr*P22minusr - P12minusr*P21minusr;
551.                         k11r = k11numr/(k11den1r + k11den2r);
552.
553.                         k12numr = -P11minusr*R12r +
   P12minusr*R11r;
554.                         k12den1r = P11minusr*R22r -
   P12minusr*R21r - P21minusr*R12r + P22minusr*R11r + R11r*R22r;
555.                         k12den2r = - R12r*R21r +
   P11minusr*P22minusr - P12minusr*P21minusr;
556.                         k12r = k12numr/(k12den1r + k12den2r);
557.
558.                         k21numr = (P21minusr*R22r -
   P22minusr*R21r);
559.                         k21den1r = P11minusr*R22r -
   P12minusr*R21r - P21minusr*R12r + P22minusr*R11r + R11r*R22r;
```

```
560.                        k21den2r = - R12r*R21r +
   P11minusr*P22minusr - P12minusr*P21minusr;
561.                        k21r = k21numr/(k21den1r+k21den2r);
562.
563.                        k22numr = -(P21minusr*R12r -
   P22minusr*R11r - P11minusr*P22minusr + P12minusr*P21minusr);
564.                        k22den1r = P11minusr*R22r -
   P12minusr*R21r - P21minusr*R12r + P22minusr*R11r + R11r*R22r;
565.                        k22den2r = - R12r*R21r +
   P11minusr*P22minusr - P12minusr*P21minusr;
566.                        k22r = k22numr/(k22den1r+k22den2r);
567.
     /*****************************************************/
568.                        titahatr[k] = titahatminusr -
   k11r*(titahatminusr - pitchr[k]);
569.                        titahatr[k] = titahatr[k] -
   k12r*(titapointhatminusr - wxr[k]);
570.                        titapointhatr[k] = titapointhatminusr
   - k21r*(titahatminusr - pitchr[k]);
571.                        titapointhatr[k] = titapointhatr[k] -
   k22r*(titapointhatminusr - wxr[k]);
572.
     /*****************************************************/
573.                        P11r[k] = -P21minusr*k12r -
   P11minusr*(k11r - 1);
574.                        P12r[k] = -P22minusr*k12r -
   P12minusr*(k11r - 1);
575.                        P21r[k] = -P11minusr*k21r -
   P21minusr*(k22r - 1);
576.                        P22r[k] = -P12minusr*k21r -
   P22minusr*(k22r - 1);
577.
578.                        //filtro de KALMAN para ROLL de la
   referencia
579.                        rollhatminusr = rollhatr[k-1] +
   (MUESTREO/1000.0)*rollpointhatr[k-1];
580.                        rollpointhatminusr = rollpointhatr[k-
   1];
581.
     /*****************************************************/
582.                        PR11minusr = PR11r[k-1] +
   (MUESTREO/1000.0)*PR21r[k-1] + QR11r;
583.                        PR12minusr = PR12r[k-1] +
   (MUESTREO/1000.0)*PR22r[k-1] - (MUESTREO/1000.0)*(PR11r[k-1] +
   (MUESTREO/1000.0)*PR21r[k-1]) + QR12r;
584.                        PR21minusr = PR21r[k-1] + QR21r;
585.                        PR22minusr = PR22r[k-1] - PR21r[k-
   1]*(MUESTREO/1000.0) + QR22r;
```

```
586.
     /*******************************************************/
587.                         kr11numr = PR11minusr*RR22r -
   PR12minusr*RR21r + PR11minusr*PR22minusr -
   PR12minusr*PR21minusr;
588.                         kr11den1r = PR11minusr*RR22r -
   PR12minusr*RR21r - PR21minusr*RR12r + PR22minusr*RR11r;
589.                         kr11den2r = RR11r*RR22r - RR12r*RR21r
   + PR11minusr*PR22minusr - PR12minusr*PR21minusr;
590.                         kr11r = kr11numr/(kr11den1r +
   kr11den2r);
591.
592.                         kr12numr = -PR11minusr*RR12r +
   PR12minusr*RR11r;
593.                         kr12den1r = PR11minusr*RR22r -
   PR12minusr*RR21r - PR21minusr*RR12r + PR22minusr*RR11r +
   RR11r*RR22r;
594.                         kr12den2r = - RR12r*RR21r +
   PR11minusr*PR22minusr - PR12minusr*PR21minusr;
595.                         kr12r = kr12numr/(kr12den1r +
   kr12den2r);
596.
597.                         kr21numr = (PR21minusr*RR22r -
   PR22minusr*RR21r);
598.                         kr21den1r = PR11minusr*RR22r -
   PR12minusr*RR21r - PR21minusr*RR12r + PR22minusr*RR11r +
   RR11r*RR22r;
599.                         kr21den2r = - RR12r*RR21r +
   PR11minusr*PR22minusr - PR12minusr*PR21minusr;
600.                         kr21r =
   kr21numr/(kr21den1r+kr21den2r);
601.
602.                         kr22numr = -(PR21minusr*RR12r -
   PR22minusr*RR11r - PR11minusr*PR22minusr +
   PR12minusr*PR21minusr);
603.                         kr22den1r = PR11minusr*RR22r -
   PR12minusr*RR21r - PR21minusr*RR12r + PR22minusr*RR11r +
   RR11r*RR22r;
604.                         kr22den2r = - RR12r*RR21r +
   PR11minusr*PR22minusr - PR12minusr*PR21minusr;
605.                         kr22r =
   kr22numr/(kr22den1r+kr22den2r);
606.
     /*******************************************************/
607.                         rollhatr[k] = rollhatminusr -
   kr11r*(rollhatminusr - rollr[k]);
608.                         rollhatr[k] = rollhatr[k] -
   kr12r*(rollpointhatminusr + wyr[k]);
```

```
609.                        rollpointhatr[k] = rollpointhatminusr
   - kr21r*(rollhatminusr - rollr[k]);
610.                        rollpointhatr[k] = rollpointhatr[k] -
   kr22r*(rollpointhatminusr + wyr[k]);
611.
     /*******************************************************/
612.                        PR11r[k] = -PR21minusr*kr12r -
   PR11minusr*(kr11r - 1);
613.                        PR12r[k] = -PR22minusr*kr12r -
   PR12minusr*(kr11r - 1);
614.                        PR21r[k] = -PR11minusr*kr21r -
   PR21minusr*(kr22r - 1);
615.                        PR22r[k] = -PR12minusr*kr21r -
   PR22minusr*(kr22r - 1);
616.
617.                        //filtro de KALMAN para YAW de la
   referencia
618.                        yawhatminusr = yawhatr[k-1] +
   (MUESTREO/1000.0)*yawpointhatr[k-1];
619.                        yawpointhatminusr = yawpointhatr[k-1];
620.
     /*******************************************************/
621.                        PY11minusr = PY11r[k-1] +
   (MUESTREO/1000.0)*PY21r[k-1] + QY11r;
622.                        PY12minusr = PY12r[k-1] +
   (MUESTREO/1000.0)*PY22r[k-1] - (MUESTREO/1000.0)*(PY11r[k-1] +
   (MUESTREO/1000.0)*PY21r[k-1]) + QY12r;
623.                        PY21minusr = PY21r[k-1] + QY21r;
624.                        PY22minusr = PY22r[k-1] - PY21r[k-
   1]*(MUESTREO/1000.0) + QY22r;
625.
     /*******************************************************/
626.                        ky11numr = PY11minusr*RY22r -
   PY12minusr*RY21r + PY11minusr*PY22minusr -
   PY12minusr*PY21minusr;
627.                        ky11den1r = PY11minusr*RY22r -
   PY12minusr*RY21r - PY21minusr*RY12r + PY22minusr*RY11r;
628.                        ky11den2r = RY11r*RY22r - RY12r*RY21r
   + PY11minusr*PY22minusr - PY12minusr*PY21minusr;
629.                        ky11r = ky11numr/(ky11den1r +
   ky11den2r);
630.
631.                        ky12numr = -PY11minusr*RY12r +
   PY12minusr*RY11r;
632.                        ky12den1r = PY11minusr*RY22r -
   PY12minusr*RY21r - PY21minusr*RY12r + PY22minusr*RY11r +
   RY11r*RY22r;
633.                        ky12den2r = - RY12r*RY21r +
   PY11minusr*PY22minusr - PY12minusr*PY21minusr;
```

```
634.                        ky12r = ky12numr/(ky12den1r +
   ky12den2r);
635.
636.                        ky21numr = (PY21minusr*RY22r -
   PY22minusr*RY21r);
637.                        ky21den1r = PY11minusr*RY22r -
   PY12minusr*RY21r - PY21minusr*RY12r + PY22minusr*RY11r +
   RY11r*RY22r;
638.                        ky21den2r = - RY12r*RY21r +
   PY11minusr*PY22minusr - PY12minusr*PY21minusr;
639.                        ky21r =
   ky21numr/(ky21den1r+ky21den2r);
640.
641.                        ky22numr = -(PY21minusr*RY12r -
   PY22minusr*RY11r - PY11minusr*PY22minusr +
   PY12minusr*PY21minusr);
642.                        ky22den1r = PY11minusr*RY22r -
   PY12minusr*RY21r - PY21minusr*RY12r + PY22minusr*RY11r +
   RY11r*RY22r;
643.                        ky22den2r = - RY12r*RY21r +
   PY11minusr*PY22minusr - PY12minusr*PY21minusr;
644.                        ky22r =
   ky22numr/(ky22den1r+ky22den2r);
645.
     /*******************************************************/
646.                        yawhatr[k] = yawhatminusr -
   ky11r*(yawhatminusr - yawr[k]);
647.                        yawhatr[k] = yawhatr[k] -
   ky12r*(yawpointhatminusr - wzr[k]);
648.                        yawpointhatr[k] = yawpointhatminusr -
   ky21r*(yawhatminusr - yawr[k]);
649.                        yawpointhatr[k] = yawpointhatr[k] -
   ky22r*(yawpointhatminusr - wzr[k]);
650.
     /*******************************************************/
651.                        PY11r[k] = -PY21minusr*ky12r -
   PY11minusr*(ky11r - 1);
652.                        PY12r[k] = -PY22minusr*ky12r -
   PY12minusr*(ky11r - 1);
653.                        PY21r[k] = -PY11minusr*ky21r -
   PY21minusr*(ky22r - 1);
654.                        PY22r[k] = -PY12minusr*ky21r -
   PY22minusr*(ky22r - 1);
655.
656.                        if(abs(yawr[k]-yawr[k-1])>350){
657.                             yawhatr[k]=yawr[k];
658.                        }
659.
```

```
660.                        //Ajuste del pitch y yaw de la
   plataforma
661.
      //yaw_platform[j]=atan(FACT_YAW*tan(yawhat[j-
   1]*PI/180))*180/PI;
662.
      //pitch_platform[j]=(atan((L*tan(titahat[j-1]*PI/180)-
   LZ)/(L+LY))+atan(LZ/L))*180/PI;
663.
664.                        }
665.
666.                        //Se imprime el archivo de datos de la
   referencia
667.                        if (k>=1){
668.                        fprintf(file1, "%f,", timer[k]);
669.                        fprintf(file1, "%6f,%6f,%6f,", axr[k],
   ayr[k], azr[k]);
670.                        fprintf(file1, "%6f,%6f,%6f,",
   rollr[k], pitchr[k], yawr[k]);
671.                        fprintf(file1, "%6f,%6f,%6f,", wxr[k],
   wyr[k], wzr[k]);
672.                        fprintf(file1, "%6f,%6f,%6f,",
   rollwr[k], pitchwr[k], yawwr[k]);
673.                        fprintf(file1, "%6f,%6f,%6f,", mxr[j],
   myr[j], mzr[j]);
674.                        fprintf(file1, "%6f,%6f,%6f\n",
   titahatr[k], rollhatr[k], yawhatr[k]);
675.                        fflush(file1);
676.                        }
677.                        k++;
678.        }
679.
680.        return 0;
681.  }
682.
683.  int main(int argc, char* argv[]) {
684.
685.      int result;
686.
687.      // Define variables de control de servomotores
688.
689.       //double curr_pos;
690.       const char *err;
691.       double maxVel, maxAccel;
692.
693.
694.      //Crea el archivo en que se escriben y guardan los
   datos del casco y la referencia
695.      FILE *file = fopen("casco_data.csv","w");
```

```
696.      FILE *file1 = fopen("ref_data.csv","w");
697.
698.      // Escribe el encabezado al inicio del archivo
699.      fprintf(file,
   "Time,A_X,A_Y,A_Z,Rolla,Pitcha,Yawa,W_X,W_Y,W_Z,Rollw,Pitchw,Y
   aww,M_X,M_Y,M_Z,Pitchk,Rollk,Yawk\n");
700.
   fprintf(file1,"Time,A_X,A_Y,A_Z,Rolla,Pitcha,Yawa,W_X,W_Y,W_Z,
   Rollw,Pitchw,Yaww,M_X,M_Y,M_Z,Pitchk,Rollk,Yawk\n");
701.
702.      //Declara un controlador casco, la referencia y un
   controlador de servos
703.       CPhidgetSpatialHandle casco = 0;
704.       CPhidgetSpatialHandle ref = 0;
705.       CPhidgetAdvancedServoHandle servo = 0;
706.
707.      // Crea el objeto casco, referencia y servo
708.      CPhidgetSpatial_create(&casco);
709.      CPhidgetSpatial_create(&ref);
710.      CPhidgetAdvancedServo_create(&servo);
711.
712.       // Conecta nuetra funcion de archivo-escritura de
   arriba con los eventos de acuerdo al flujo para el casco y la
   referencia
713.      CPhidgetSpatial_set_OnSpatialData_Handler(casco,
   SpatialDataHandler, (void *) file);
714.      CPhidgetSpatial_set_OnSpatialData_Handler(ref,
   SpatialDataHandler1, (void *) file1);
715.
716.       //Abre el objeto casco y servo para conectar
   dispositivos (casco 296173 / referencia 296382)
717.      CPhidget_open((CPhidgetHandle)casco, 296173);
718.      CPhidget_open((CPhidgetHandle)ref, 296382);
719.      CPhidget_open((CPhidgetHandle)servo, -1);
720.
721.       //llama el programa para esperar que el casco sea
   conectado
722.      if((result =
   CPhidget_waitForAttachment((CPhidgetHandle)casco, 10000)))
723.       {
724.       CPhidget_getErrorDescription(result, &err);
725.       printf("No se detecto IMU CASCO!, tipo de error:
   %s\n",err);
726.          return 0;
727.       }
728.       //llama el programa para esperar que referencia sea
   conectado
729.      if((result =
   CPhidget_waitForAttachment((CPhidgetHandle)ref, 10000)))
```

```
730.       {
731.       CPhidget_getErrorDescription(result, &err);
732.       printf("No se detecto IMU REFERENCIA!, tipo de error:
   %s\n",err);
733.          return 0;
734.       }
735.      //llama el programa para esperar que un dispositivo
   servo sea conectado
736.      if((result =
   CPhidget_waitForAttachment((CPhidgetHandle)servo, 1000)))
737.       {
738.       CPhidget_getErrorDescription(result, &err);
739.       printf("No se detecto Controlador de Servos, Tipo de
   error: %s\n", err);
740.       return 0;
741.       }
742.
743.      // calibra y establece valores iniciales de velocidad y
   aceleracion de motor cero (0) PITCH
744.
      CPhidgetAdvancedServo_setServoParameters(servo,0,560.00,245
   0.00,180.00,273.00);
745.       CPhidgetAdvancedServo_getAccelerationMax(servo, 0,
   &maxAccel);
746.       CPhidgetAdvancedServo_setAcceleration(servo, 0,
   maxAccel);
747.       CPhidgetAdvancedServo_getVelocityMax(servo, 0,
   &maxVel);
748.       CPhidgetAdvancedServo_setVelocityLimit(servo, 0,
   maxVel);
749.       printf("Maxima Velocidad 0: %f\n", maxVel);
750.       printf("Maxima Aceleracion 0: %f\n", maxAccel);
751.
752.      // calibra y establece valores iniciales de velocidad y
   aceleracion de motor siete (7) YAW
753.
      CPhidgetAdvancedServo_setServoParameters(servo,7,560.00,240
   0.00,180.00,273.00);
754.             CPhidgetAdvancedServo_getAccelerationMax(servo,
   7, &maxAccel);
755.             CPhidgetAdvancedServo_setAcceleration(servo, 7,
   maxAccel);
756.             CPhidgetAdvancedServo_getVelocityMax(servo, 7,
   &maxVel);
757.             CPhidgetAdvancedServo_setVelocityLimit(servo, 7,
   maxVel);
758.             printf("Maxima Velocidad 7: %f\n", maxVel);
759.             printf("Maxima Aceleracion 7: %f\n", maxAccel);
760.
```

```
761.
762.
763.      //Establece la velocidad de toma de datos para los
   eventos de la IMU en milisegundos (min 8ms)
764.       CPhidgetSpatial_setDataRate(casco, MUESTREO);
765.       CPhidgetSpatial_setDataRate(ref, MUESTREO);
766.
767.       //Establece parametros de calibracion de la brujula
   magnetica
768.       CPhidgetSpatial_setCompassCorrectionParameters(casco,0
   .32245, 0.01653, 0.23588, 0.02339, 2.96484, 2.86203, 3.47701,
   -0.08152, -0.03856, -0.07516, 0.00295, -0.04437, 0.00382);
769.       CPhidgetSpatial_setCompassCorrectionParameters(ref,0.3
   3673, 0.02460, 0.23972, -0.03675, 2.81121, 2.76034, 3.33769, -
   0.04998, -0.03705, -0.04844, 0.00489, -0.04341, 0.00583);
770.
771.       //Engancha el servo
772.       CPhidgetAdvancedServo_setEngaged(servo, 0, 1);
773.       CPhidgetAdvancedServo_setEngaged(servo, 7, 1);
774.
775.       //Calibra YAW
776.       if(j>1 && k>1){
777.            delta_yaw_ini=yaw_inicial-yaw_inicialr;
778.            delta_pitch_ini=pitch_inicial-pitch_inicialr;
779.       }
780.
781.       //Posiciona el servo de acuerdo a la IMU en PITCH y
   YAW utilizando valores filtrados con Kalman
782.       while(k<MAX){
783.       if(k>=j){
784.       CPhidgetAdvancedServo_setPosition (servo, 0, -
   pitch_platform[j-1]+90);//Ajuste de pitch para mover servo
785.       CPhidgetAdvancedServo_setPosition (servo, 7,
   yaw_platform[j-1]+90);//Ajuste de Yaw para mover servo
786.       }
787.       else{
788.       CPhidgetAdvancedServo_setPosition (servo, 0, -
   pitch_platform[k-1]+90);//Ajuste de pitch para mover servo
789.       CPhidgetAdvancedServo_setPosition (servo, 7,
   yaw_platform[k-1]+90);//Ajuste de Yaw para mover servo
790.       }
791.       }
792.
793.       //Desengancha, cierra el objeto casco, referencia y
   servo y los borra y cierra el archivo *.csv
794.      CPhidgetAdvancedServo_setEngaged(servo, 0, 0);
795.      CPhidget_close((CPhidgetHandle)casco);
796.      CPhidget_delete((CPhidgetHandle)casco);
797.      CPhidget_close((CPhidgetHandle)ref);
```

```
798.      CPhidget_delete((CPhidgetHandle)ref);
799.      CPhidget_close((CPhidgetHandle)servo);
800.      CPhidget_delete((CPhidgetHandle)servo);
801.      fclose(file);
802.      fclose(file1);
803.
804.      return 0;
805.  }
806.
807.
```