

Universidad Nacional de Educación a Distancia

Escuela Técnica Superior de Ingeniería Informática

Máster Universitario en Ingeniería de Sistemas y de Control

Trabajo de Fin de Master

IMPLEMENTACIÓN DE FILTROS DE KALMAN PARA CONTROL DE ACTITUD EN MICROCONTROLADOR RASPBERRY PI PICO

Autor:

Enrique FLORES MONTOYA

Directoras:

Raquel DORMIDO

Natividad DURO

5 de mayo de 2023

Resumen

Agradecimientos

Índice general

1. Introducción	1
1.1. Avances en UAVs and MAVs	1
1.2. Filtro de Kalman	3
1.3. Microcontroladores	5
2. Fundamentos Teóricos	9
2.1. Sistemas de referencia	9
2.1.1. Sistema de referencia inercial: (O_I, x_I, y_I, z_I)	9
2.1.2. Sistema de referencia ejes cuerpo: (O_b, x_b, y_b, z_b)	10
2.2. Formalismos para describir la actitud	10
2.2.1. Ángulos de Euler	10
2.2.2. Cuaternios	12
3. Dispositivo experimental	17
3.1. Raspberry Pi Pico	17
3.1.1. Especificaciones	17
3.1.2. Librerías	18
3.1.3. Picoprobe	19
3.2. Sensor MPU 9250	19
4. Caracterización de los sensores	21
4.1. Generalidades	21
4.1.1. Random walk	21
4.1.2. Dealineación interna de los sensores	21

4.2. Calibración del giróscopo.	21
4.3. Calibración del acelerómetro	22
4.4. Calibración del magnetómetro.	25
5. Extended Kalman Filter	31
5.1. Accelerometer data	33
5.2. Magnetometer data	37
5.3. Quaternion EKF Implementation	40
5.3.1. Prediction	40
5.3.2. Update	41

Capítulo 1

Introducción

1.1. Avances en UAVs and MAVs

Los vehículos autónomos o Unmanned Aerial Vehicles (UAVs) han experimentado un intenso desarrollo en las últimas décadas. Su avance se ha visto impulsado por las mejoras en las baterías y los motores sin escobillas, así como por el desarrollo de nuevos materiales compuestos y el abaratamiento y la miniaturización de la electrónica [boukoberine2019critical, hristov2016review, mazur2016clarity]. Los UAVs se emplean en diferentes aplicaciones, tanto civiles como militares: dragaminas (identificación de minas marinas), cobertura inalámbrica, mantenimiento de carreteras, planificación urbanística, prevención de desastres naturales y usos agrícolas. Se trata de un sector creciente cuyo valor de mercado se esperaba que alcanzase los 127 billones de dólares en 2020 [mazur2016clarity]. Existe una amplia variedad de configuraciones de UAV que van desde las aeronaves de ala fija a los vehículos de ala batida pasando por dispositivos de ala rotatoria y de tipo globo. Los vehículos de ala fija se utilizan para cubrir grandes áreas de terreno dada su mayor autonomía y alcance. En contraste, los vehículos de ala rotatoria basados en el uso de hélices son más adecuados para interiores y zonas de baja accesibilidad gracias a su mayor maniobrabilidad.

Independientemente de su configuración alar, los UAVs comparten una serie de elementos comunes que se resumen en el diagrama de bloques de la figura 1.1. Todo sistema de vuelo autónomo consta de un vehículo no pilotado, una estación de control en tierra que puede ser autónoma u operada manualmente y un sistema de telemetría y telecomandos que mantiene una comunicación entre la estación en suelo y el vehículo. Dentro de los elementos que conforman el vehículo, podemos distinguir los siguientes subsistemas,

1. **Planta de potencia y sistema de gestión de energía.** La mayoría de los vehículos de tamaño mediano y pequeño son alimentados por baterías. Los vehículos alimentados por baterías se caracterizan por una gran fiabilidad, baja firma térmica

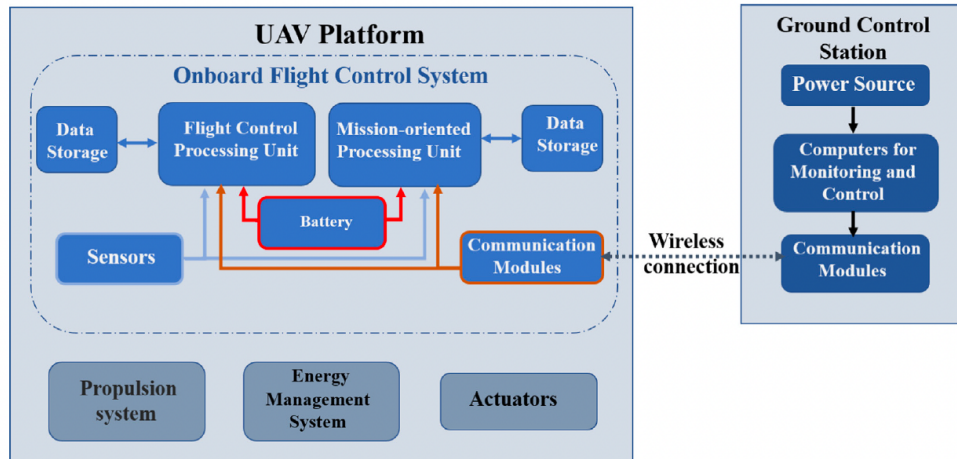


Figura 1.1: Diagrama de bloques de un UAV. (Fuente [boukoberine2019critical]).

y nulas emisiones de contaminantes. Los recientes avances en baterías de Ión Litio-Polímero (LiPo) han permitido lograr autonomías de hasta 90 minutos. Alternativamente, los UAVs eléctricos pueden ser alimentados por pilas de combustible. Las pilas de combustible producen energía eléctrica a partir de hidrógeno y se caracterizan por una mayor densidad energética que las baterías. No obstante, presentan una menor potencia específica [boukoberine2019critical]. Otra de las ventajas es que la recarga del depósito de combustible es mucho más rápida que la de las baterías. Finalmente, los vehículos de mayor tamaño emplean motores térmicos: motores alternativos o turbinas de gas como fuente de potencia. Un análisis más detallado sobre las fuentes de potencia y las estrategias de gestión de energía puede encontrarse en [boukoberine2019critical]. De ahora en adelante, nos centraremos en el análisis vehículos eléctricos alimentados por baterías.

2. **Sistema de propulsión y actuadores.** En los UAVs, el sistema de propulsión transforma la energía eléctrica proporcionada por las baterías en energía mecánica que permite al UAV desplazarse y desempeñar su misión. Generalmente, los UAVs utilizan un sistema de propulsión a hélice para mantenerse en vuelo. En las aeronaves de ala rotatoria, el sistema de propulsión proporciona la fuerza de sustentación que permite al vehículo elevarse del suelo. Por el contrario, en las aeronaves de ala fija, la fuerza de sustentación se genera en las alas y el sistema de propulsión se limita a compensar la resistencia aerodinámica permitiendo al vehículo mantener una velocidad de avance. La transformación de la energía eléctrica en energía mecánica se realiza mediante motores sin escobillas de corriente continua. Las aeronaves de ala fija requieren además de un sistema de actuadores que permita accionar las superficies de control aerodinámico como los alerones y el timón de cola. En los UAVs eléctricos de pequeño tamaño esto puede lograrse mediante el uso de servomotores.

3. **Sistema de control a bordo.** El sistema de control de vuelo a bordo consta de, como mínimo, una unidad de procesamiento y control de vuelo, un módulo de comunicación que permite al vehículo recibir órdenes de la estación en tierra y una serie de sensores que proporcionan al ordenador de a bordo información sobre el entorno y el estado del UAV. Además, el sistema de vuelo puede incluir una unidad de procesamiento de misión, un sistema de almacenamiento de datos y sensores adicionales. Estos últimos suelen estar relacionados con la misión del UAV y son independientes del control de vuelo *i. e.*, cámaras termográficas, medidores de la composición del aire, etc.

Dependiendo del objetivo de su misión, los UAVs modernos pueden ser generalmente operados tanto en modo manual como de manera completamente autónoma. Independientemente de como se realice la planificación de la trayectoria del vehículo a alto nivel, el control y la estabilización en vuelo requieren estimar su posición y su actitud para poder enviar las señales de control a los actuadores. Para ello, estos sistemas integran diferentes sensores tales como medidores de altitud barométricos, IMUs (Inertial Measurement Unit), GPSs y cámaras de baja resolución. No obstante, en general, se necesita procesar los datos de los sensores para estimar el estado del sistema. En particular, para la estimación de la actitud, la mayoría de los vehículos utilizan filtros de Kalman [zhang2005navigation, pettersson2015extended, trawny2005indirect].

1.2. Filtro de Kalman

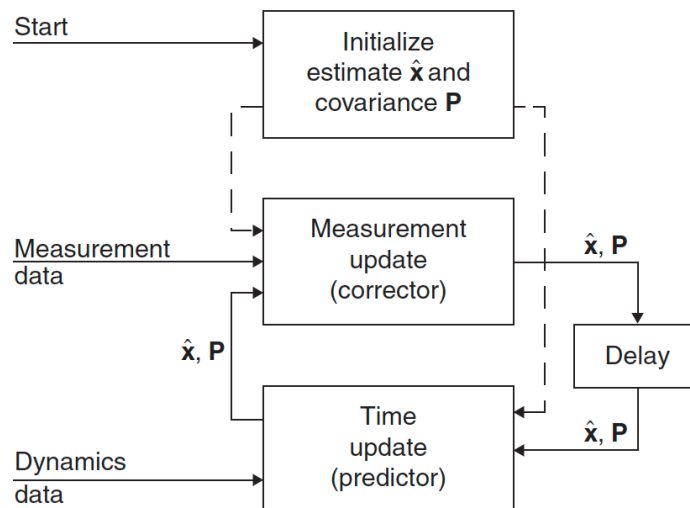


Figura 1.2: Diagrama de flujo de un filtro de Kalman. (Fuente [grewal2020kalman]).

El filtro de Kalman [chui2017kalman] fue desarrollado por Rudolf E. Kalman en

1960. En [kalman1960new], Kalman describe una solución recursiva para el problema del filtrado lineal de datos discretos. Un filtro de Kalman es un estimador en tiempo real que se utiliza para trazar el estado de un sistema dinámico con perturbaciones aleatorias a partir de mediciones ruidosas. El filtro es un procedimiento matemático que opera por medio de un mecanismo de predicción y corrección (<https://www.kalmanfilter.net/default.aspx>). En esencia, este algoritmo pronostica el nuevo estado a partir de su estimación previa añadiendo un término de corrección proporcional al error de predicción, de tal forma que este último es minimizado estadísticamente. La inicialización del filtro requiere asignar unos valores iniciales al vector de estados x y a su matriz de covarianza P , que representa la incertidumbre de la estimación inicial. Tras la inicialización, el algoritmo itera en bucle entre la fase de predicción y corrección. En gran medida, su popularidad emerge de su buen comportamiento en un gran número de aplicaciones reales.

La utilización de filtros de Kalman esta muy extendida tanto en el campo de la electrónica industrial como en la robótica [auger2013industrial, chen2011kalman]. Uno de los primeros usos del filtro de Kalman fue el procesado de los datos para el guiado y el control de trayectoria de las naves en el programa Apollo. Desde ese momento, debido en gran parte al avance en el cálculo digital, el filtro de Kalman ha sido objeto de una extensiva investigación y aplicación, particularmente en el área de la navegación autónoma y asistida, en rastreo de misiles y en economía. A día de hoy, los filtros de Kalman constituyen una parte esencial de los sistemas de navegación modernos tanto inerciales como por satélite [grewal2020kalman, grewal2014kalman]. El filtro se caracteriza por su bajo coste computacional y sus buenas propiedades recursivas. Es uno de los algoritmos de fusión de datos y sensores más comunes y más utilizados.

To be included

Comparación con el filtro complementario [higgins1975comparison, gui2015mems].

Diferentes tipos de filtros de Kalman [emer2020survey].

Paper sobre el filtro complementario [euston2008complementary]

La unidad de procesado y control de vuelo es la encargada de estimar el estado de la aeronave. La información sobre la actitud del UAV es utilizada por el algoritmo de control para mantener la estabilidad y seguir la trayectoria prevista. En un sistema de navegación inercial, los datos recogidos por la IMU (Inertial Measurement Unit) son procesados por el ordenador de vuelo para estimar la actitud del vehículo. La comunicación con los sensores y la implementación del filtro de Kalman se realizan en la unidad de procesado cuyo soporte físico (hardware) es una computadora o un microcontrolador.

1.3. Microcontroladores

Un microcontrolador es un sistema embebido [marwedel2021embedded] que está programado de manera secuencial y que, a diferencia de un microprocesador, no requiere de un sistema operativo (OS) [bolanakis2019survey]. Los microcontroladores consisten en un circuito digital síncrono que incorpora una unidad central de procesamiento (CPU), una memoria que almacena el programa (memoria de programa) y una memoria que contiene los datos que se generan durante la ejecución del código (memoria de datos).

El uso de microcontroladores está muy extendido en todo tipo de aplicaciones desde dispositivos médicos implantables, sistemas de procesamiento de control, dispositivos de instrumentación industrial y electrodomésticos [kondaveeti2021systematic]. Su utilización industrial se ha visto impulsada por su bajo consumo de potencia y por el abaratamiento de la electrónica debido al desarrollo de la tecnología de impresión de placas de circuitos.

Además, su gran versatilidad ha potenciado el uso de los microcontroladores para el prototipado y la educación [bolanakis2019survey]. A día de hoy, en el campo de la robótica existen un gran número de herramientas de prototipado accesibles al gran público tales como Arduino [kondaveeti2021systematic] o Raspberry pi [upton2016raspberrypi]. Estos dispositivos se caracterizan por su flexibilidad, bajo coste y accesibilidad. Los microcontroladores de uso extendido como el Arduino Uno incorporan conectividad USB para la alimentación, la comunicación y la carga del código (bootloader). Todos estos aspectos, junto con la existencia de bibliotecas *ready to use* han facilitado su extensión y fomentado la filosofía del DIY (Do It Yourself).

De entre todos los microcontroladores de bajo coste, Arduino es la tecnología más extendida, con una mayor comunidad de usuarios, soporte y bibliotecas. Existen diferentes modelos de circuitos impresos Arduino: Uno, Due, Mega, Nano, etc. El modelo más común es el Arduino Uno, que monta un procesador ATmega328 con una frecuencia de reloj de 16 MHz. La placa Arduino Uno cuenta con una memoria de programa de 32 kB, una memoria RAM de 2kB y una memoria EEPROM de 1kB. El microcontrolador posee 14 pines digitales de salida y entrada (I/O digital pins) y 6 pines de analógicos, además de dos pines de potencia a 5 V y 3.3 V. Las otras versiones de Arduino ofrecen un número diferente de entradas y salidas y/o niveles de memoria de programa y RAM marginalmente superiores [kondaveeti2021systematic].

Debido a sus limitadas prestaciones de memoria y capacidad de procesamiento, la tecnología Arduino no permite ni el uso extensivo de bibliotecas ni de grandes frecuencias de ejecución. A modo de ejemplo, consideremos que queremos incluir en el código principal del vehículo autónomo las bibliotecas para el control de los motores (`servo.h`), la comunicación por radio a través del chip NRF24L01 (`RF24.h`) y la lectura de los datos de la IMU MPU9250 (`MPU9250.h`). La mera incorporación de estas librerías implica un volumen de programa de **X kB**. Esto limita enormemente la extensión, y por tanto, la compleji-

dad y la sofisticación del código principal. La falta de memoria interna imposibilita, entre otros, la inclusión de bibliotecas para la implementación de un filtro de Kalman tales como (`Kalman_Filter_Library.h`). La utilización de cámaras de baja resolución que permitan, mediante el procesamiento de imágenes con herramientas de flujo óptico, la determinación de las velocidades de desplazamiento horizontal del vehículo queda totalmente descartada.

Otra de las fuertes limitaciones del microcontrolador Arduino Uno es que su chip, el ATmega328, que pertenece a la familia de microcontroladores AVR desarrollados por ATMEL [kunikowski2015overview], tiene una arquitectura de 8 bits. Esto implica que todas las operaciones básicas se realizan en paquetes de 8 bits. En aras de la comparación, las computadoras modernas y los celulares tienen típicamente arquitecturas de 64 y 32 bits respectivamente. Además, la mayoría de los microcontroladores del mercado carecen de Unidad de Procesado de Números de coma Flotante (FPU por sus siglas en inglés). Las operaciones con números de coma flotante están implementadas mediante software.

Por todas estas razones, la utilización de microcontroladores Arduino para el prototipado en vehículos autónomos condiciona enormemente sus prestaciones. En consecuencia, es necesario considerar el uso de microcontroladores con mayor capacidad de procesamiento y memoria interna. Además de Arduino, existen otros muchos microcontroladores disponibles en el mercado que se utilizan en aplicaciones de robótica tales como el ESP8266, el MSP430, el ESP32, el BBCmicrobit y el STM32. La Raspberry pi Pico es un microcontrolador recientemente desarrollado y comercializado por Raspberry. Las especificaciones detalladas pueden encontrarse en la sección 3. Su uso está menos extendido que el de Arduino pero su potencia y capacidad de procesamiento son mucho mayores. Esto lo convierte en un microcontrolador más adecuado para aplicaciones más exigentes donde se requieren mayores frecuencias de ejecución y un procesamiento de datos más complejo.

El objetivo del presente trabajo es abordar la sensorización de un sistema de navegación inercial. Para ello, se busca implementar un filtro de Kalman para la estimación de la actitud de un vehículo autónomo utilizando un microcontrolador Raspberry Pi Pico. La programación de este dispositivo requiere un mayor esfuerzo inicial debido a,

- su mayor complejidad y flexibilidad,
- la necesidad de construir un entorno para el *bootloading*,
- su menor implantación, que se traduce en un menor nivel de documentación.

No obstante, la Raspberry pi Pico es un microcontrolador con prestaciones suficientes para alojar un algoritmo de control, un protocolo de telecomandos-telemetría y una sensorización avanzada. Esto permitirá extender las funcionalidades de un futuro vehículo autónomo desarrollado en base a este microcontrolador. El trabajo busca abarcar todo el proceso de implementación desde el más bajo nivel (acceso a los registros de memoria del sensor) hasta un nivel medio (implementación y comparación entre diferentes filtros). El documento

se estructura de la siguiente forma: el capítulo 2 presenta los fundamentos teóricos, el capítulo 3 describe el montaje experimental, el capítulo 4 aborda la calibración de los sensores y el capítulo 5 describe la implementación del filtro de Kalman extendido.

Capítulo 2

Fundamentos Teóricos

2.1. Sistemas de referencia

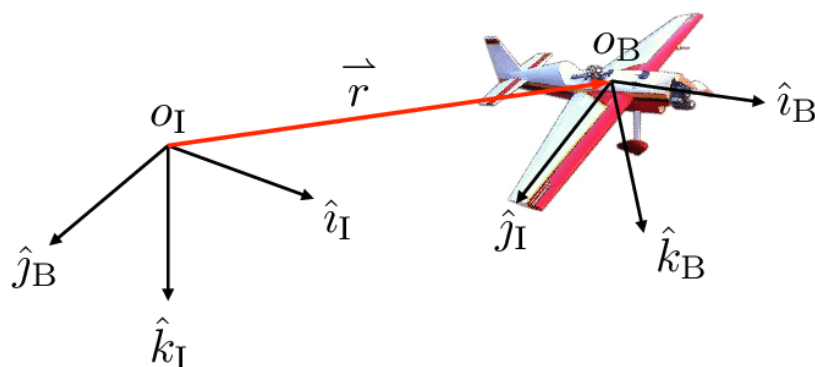


Figura 2.1: Sistemas de referencia inercial y ejes cuerpo. Fuente [mullen2016filtered].

En esta sección se presentan los sistemas de referencia empleados para definir la actitud de un vehículo aéreo. Los sistemas de referencia se denotan, de forma genérica por, $F(O, x, y, z)$ donde (O) indica el origen y (x, y, z) son los ejes del sistema de referencia. En la definición de la actitud de un vehículo es necesario considerar dos sistemas de referencia relevantes.

2.1.1. Sistema de referencia inercial: (O_I, x_I, y_I, z_I)

Es un sistema de referencia cartesiano inercial centrado en un punto arbitrario de la superficie terrestre, O_I . El eje x_I apunta en la dirección del norte geodésico, el eje y_I apunta en la dirección del este geodésico y el eje z_I apunta en la dirección del centro de la tierra. Este sistema de referencia, en ocasiones denotado por NED (North-East-Down) se utiliza

a menudo en para la navegación en aeronaves de pequeño tamaño. Generalmente, se toma el punto de despegue del vehículo como origen del sistema de coordenadas y se utiliza este sistema de referencia para describir la posición, la velocidad y la aceleración del vehículo con respecto a este punto.

2.1.2. Sistema de referencia ejes cuerpo: (O_b, x_b, y_b, z_b)

El sistema de referencia ejes cuerpo (O_b, x_b, y_b, z_b) está centrado en el centro de gravedad del vehículo. El eje x_b apunta hacia adelante y está contenido en el plano de simetría del vehículo, el eje y_b apunta a estribor (hacia la derecha desde el punto de vista de un observador que mira en la dirección del eje x_b) y el eje z_b apunta hacia abajo, en dirección al suelo, y esta también contenido en el plano de simetría del vehículo.

En los vehículos aéreos, las rotaciones alrededor de los ejes cuerpo se conocen respectivamente como balance, cabeceo y guiñada o *roll*, *pitch* y *yaw* por sus nombres en inglés. Estos ángulos describen la orientación del vehículo con respecto a los ejes inerciales (O_I, x_I, y_I, z_I) , lo que se conoce comúnmente como actitud del vehículo.

2.2. Formalismos para describir la actitud

Las rotaciones permiten pasar de unos sistemas de referencia a otros. Existen diferentes formalismos que se emplean para representar las transformaciones entre sistemas de referencia. El método más intuitivo para describir la orientación o actitud de un cuerpo en el espacio son los ángulos de Euler según la convención de Tait-Bryan. Sin embargo, los ángulos de Euler tienen ciertas limitaciones que hacen que los cuaternios sean el formalismo adoptado en los sistemas de estimación de la actitud. La descripción de la actitud mediante cuaternios puede traducirse a ángulos de Euler, lo que permite obtener un resultado más fácilmente interpretable.

2.2.1. Ángulos de Euler

Fundamentos

La forma más intuitiva de describir la actitud de un cuerpo en el espacio es mediante los ángulos de Euler. La orientación de un sistema de referencia con respecto a otro puede describirse mediante tres rotaciones sucesivas. Las rotaciones se representan mediante matrices ortonormales 3×3 ¹. La composición de rotaciones se obtiene multiplicando matrices

¹Una matriz M es ortogonal si y solo si $M^T = M^{-1}$ y además es ortonormal si cumple que su norma es la unidad, es decir, que el módulo de cada vector que la forma es la unidad.

de rotación individuales y la transformación inversa viene dada por la matriz transpuesta de la matriz de rotación. El orden en el que se realizan las rotaciones alrededor de los diferentes ejes se denomina convención. Nótese que las rotaciones finitas no son conmutativas. Existen diferentes convenciones de ángulos de Euler. La convención utilizada para describir la orientación de aeronaves y vehículos espaciales es la convención de Tait-Bryan zyx . En otros campos científicos como la Física de Partículas o la Astronomía, son usuales convenciones distintas como la zyz o la zxz . En aviónica, los ángulos de Euler se utilizan para describir la relación entre los ejes inerciales $-I$ y los ejes cuerpo $-b$,

- **Yaw** ψ : ángulo de rotación alrededor del eje z_I del sistema de referencia inercial. El rumbo del vehículo viene dado por el ángulo formado entre la proyección el eje x_b del vehículo sobre el plano formado por los ejes x_I y y_I y el eje x_I . La matriz de rotación asociada es,

$$C_\psi = \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

- **Pitch** θ : ángulo formado entre el plano $x_I - y_I$ y el eje x_b . La matriz de rotación correspondiente a este giro es,

$$C_\theta = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \quad (2.2)$$

- **Roll** ϕ : ángulo de rotación alrededor del eje x_b . La matriz de rotación asociada a esta transformación viene dada por,

$$C_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \psi & \sin \psi \\ 0 & -\sin \psi & \cos \psi \end{bmatrix} \quad (2.3)$$

La matriz de rotación global se obtiene mediante la composición de las tres rotaciones *yaw*, *pitch*, *roll* y viene dada por la multiplicación de las matrices C_ψ , C_θ y C_ϕ . De esta manera, la matriz de rotación que permite transformar las componentes de un vector expresadas en ejes inerciales a ejes cuerpo viene dada por,

$$C_I^b = C_\phi C_\theta C_\psi \quad (2.4)$$

donde el orden de las rotaciones tiene lugar de derecha a izquierda. La expresión de la matriz de rotación resultante es,

$$C_I^b = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \psi \\ \sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \sin \psi \sin \theta \sin \psi + \cos \phi \cos \psi & \sin \phi \cos \theta \\ \cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \sin \phi \sin \psi & \cos \phi \cos \theta \end{bmatrix} \quad (2.5)$$

dado que la matriz C_I^b es ortonormal, la transformación de un vector expresado en ejes cuerpo (O_b, x_b, y_b, z_b) a ejes inerciales (O_I, x_I, y_I, z_I) se realiza multiplicando por la matriz C_b^I , que satisface,

$$C_b^I = (C_I^b)^{-1} = (C_I^b)^T \quad (2.6)$$

Propagación de ángulos de Euler

El vector de velocidad angular del cuerpo con respecto al sistema de referencia inercial expresado en ejes cuerpo viene dado por,

$$\omega = [\omega_x \quad \omega_y \quad \omega_z] = [p \quad q \quad r] \quad (2.7)$$

La relación entre la velocidad angular y la variación temporal de los ángulos de Euler viene dada por,

$$p = \dot{\phi} - \dot{\psi} \sin \theta \quad (2.8)$$

$$q = \dot{\theta} \cos \phi + \dot{\psi} \cos \theta \sin \phi \quad (2.9)$$

$$r = -\dot{\theta} \sin \phi + \dot{\psi} \cos \theta \cos \phi \quad (2.10)$$

y la relación inversa se expresa como,

$$\dot{\phi} = p + (q \sin \phi + r \cos \phi) \tan \theta \quad (2.11)$$

$$\dot{\theta} = q \cos \phi - r \sin \phi \quad (2.12)$$

$$\dot{\psi} = (q \sin \phi + r \cos \phi) \frac{1}{\cos \theta} \quad (2.13)$$

Nótese que la ecuación (2.13) es singular cuando el ángulo de asiento (pitch) es $\theta = \pm\pi/2$. Esto limita el uso de los ángulos de Euler para representar la actitud de un cuerpo haciendo que los cuaternios unitarios sean la alternativa escogida por la mayoría de sistemas de integración de la actitud.

2.2.2. Cuaternios

Fundamentos

Los cuaternios son un formalismo que permite describir la actitud de un cuerpo y las rotaciones entre diferentes sistemas de referencia. Un cuaternio esta formado por cuatro componentes,

$$\mathbf{q} = [q_1 \quad q_2 \quad q_3 \quad q_4] \quad (2.14)$$

El cuaternio también puede expresarse como un número complejo con una componente real y cuatro componentes imaginarias,

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} \quad (2.15)$$

El producto entre dos cuaternios $\mathbf{q}_1 = [q_0 \ q_1 \ q_2 \ q_3]$ y $\mathbf{q}_2 = [a \ b \ c \ d]$ puede expresarse como el producto de matrices,

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \quad (2.16)$$

Los cuaternios pueden usarse para describir las rotaciones entre sistemas de referencia. Un cuaternio de componentes $[q_0 \ q_1 \ q_2 \ q_3]$ representa un giro de un ángulo α alrededor de un eje $\mathbf{t} = [t_x \ t_y \ t_z]$,

$$\mathbf{q} = \left[\cos \frac{\alpha}{2}, \mathbf{t} \sin \frac{\alpha}{2} \right] \quad (2.17)$$

Sea \mathbf{r}_b un vector cualquiera expresado en ejes cuerpo, este puede escribirse como un cuaternio cuya componente escalar es nula,

$$\mathbf{r}_b = 0 + r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k} \quad (2.18)$$

Para expresar el vector \mathbf{r}_b en ejes inerciales se aplica,

$$\mathbf{r}_I = \mathbf{q} \cdot \mathbf{r}_b \cdot \mathbf{q}^* \quad (2.19)$$

donde \mathbf{q}^* denota el cuaternio conjugado de \mathbf{q} . Esta operación también puede expresarse de forma matricial como,

$$\mathbf{r}_I = \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{C} \end{bmatrix} \mathbf{r}_b \quad (2.20)$$

donde \mathbf{C} es la matriz de rotación entre el sistema de referencia ejes cuerpo y el sistema inercial C_b^I . La matriz \mathbf{C} expresada en función de las componentes de \mathbf{q} viene dada por,

$$\mathbf{C} = \begin{bmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & (q_0^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & (q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{bmatrix} \quad (2.21)$$

de nuevo, la matriz \mathbf{C} es ortonormal y su transpuesta puede utilizarse para rotar un vector expresado en el sistema de referencia inercial al sistema de referencia ejes cuerpo.

Propagación de cuaternios

Dado un sistema de ejes cuerpo $-b$ que rota con un vector velocidad angular $\omega = [p \ q \ r]$ con respecto al sistema de referencia inercial $-I$ y sea \mathbf{q} el cuaternio que describe la orientación del sistema de referencia ejes cuerpo con respecto al sistema de referencia inercial, la variación temporal de \mathbf{q} viene dada por,

$$\dot{\mathbf{q}} = \frac{1}{2} \mathbf{q} \cdot \boldsymbol{\Omega} \quad (2.22)$$

donde $\boldsymbol{\Omega}$ se define como,

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 \\ p \\ q \\ r \end{bmatrix} \quad (2.23)$$

La ecuación (2.22) puede expresarse en forma matricial como,

$$\dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} 0 \\ p \\ q \\ r \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \frac{1}{2} \tilde{S}(\mathbf{q}) \omega \quad (2.24)$$

Esta expresión puede reescribirse para expresarla como el producto entre una matriz $S(\omega)$ y el cuaternio \mathbf{q} ,

$$\dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \frac{1}{2} S(\omega) \mathbf{q} \quad (2.25)$$

La integración de la ecuación (2.25) en el tiempo proporciona la evolución de la orientación del cuerpo descrita por el cuaternio \mathbf{q} en función de la velocidad angular del sistema $-b$ con respecto al sistema inercial $-I$, ω .

Relación entre los cuaternios y los ángulos de Euler

El cuaternio \mathbf{q} que representa la orientación del sistema ejes cuerpo $-b$ con respecto al sistema inercial $-I$ puede expresarse en función de los ángulos de Euler (ϕ, θ, ψ) como,

$$q_0 = \cos \frac{\phi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \quad (2.26)$$

$$q_1 = \sin \frac{\phi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} - \cos \frac{\phi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \quad (2.27)$$

$$q_2 = \cos \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} \quad (2.28)$$

$$q_3 = \cos \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} - \sin \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} \quad (2.29)$$

Igualmente, a partir del cuaternio \mathbf{q} que describe la orientación del sistema $-b$ podemos obtener los ángulos de Euler que definen la actitud del cuerpo,

$$\phi = \arctan \left[\frac{2(q_2q_3 + q_0q_1)}{1 - 2(q_1^2 + q_2^2)} \right] \quad (2.30)$$

$$\theta = \arcsin [2(q_0q_2 - q_1q_3)] \quad (2.31)$$

$$\psi = \arctan \left[\frac{2(q_1q_2 + q_0q_3)}{1 - 2(q_2^2 + q_3^2)} \right] \quad (2.32)$$

Las funciones arcotangente implementadas en los ordenadores solo producen resultados entre $\pm\pi/2$. Para poder generar todas las orientaciones es necesario sustituir la función `atan` por `atan2`, que tiene en cuenta el signo del numerador y del denominador. De nuevo, estas expresiones son singulares para $\theta \rightarrow \pm\pi/2$.

Capítulo 3

Dispositivo experimental

El dispositivo experimental utilizado para la implementación de un filtro de Kalman para la estimación de la actitud se representa en la figura 3.1. El circuito se monta sobre una placa de prototipado e incluye dos microcontroladores Raspberry Pi pico y la Unidad de Medición Inercial (IMU por sus siglas en inglés) MPU9250. A continuación se describen en detalle las características de los diferentes elementos del circuito.

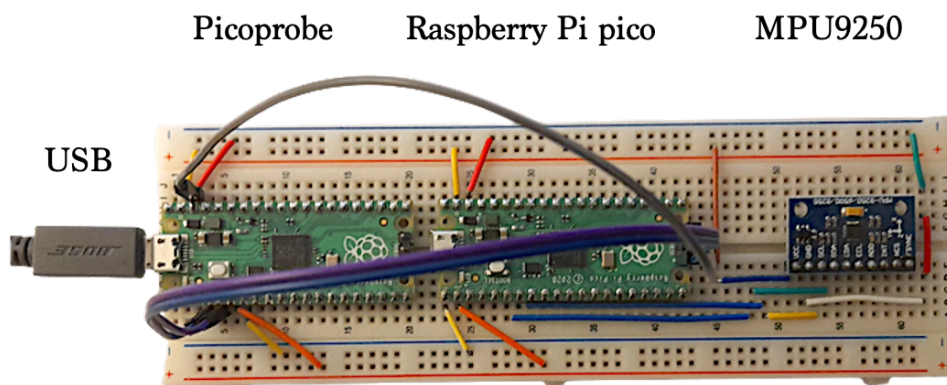


Figura 3.1: Montaje experimental

3.1. Raspberry Pi Pico

3.1.1. Especificaciones

La Raspberry pi Pico (<https://www.raspberrypi.com/products/raspberry-pi-pico/>) es un microcontrolador de alto rendimiento de tipo placa basado en el chip RP2040 diseñado por Raspberry Pi [[raspberrypiPico-datasheet](#)]. El microcontrolador Raspberry pi

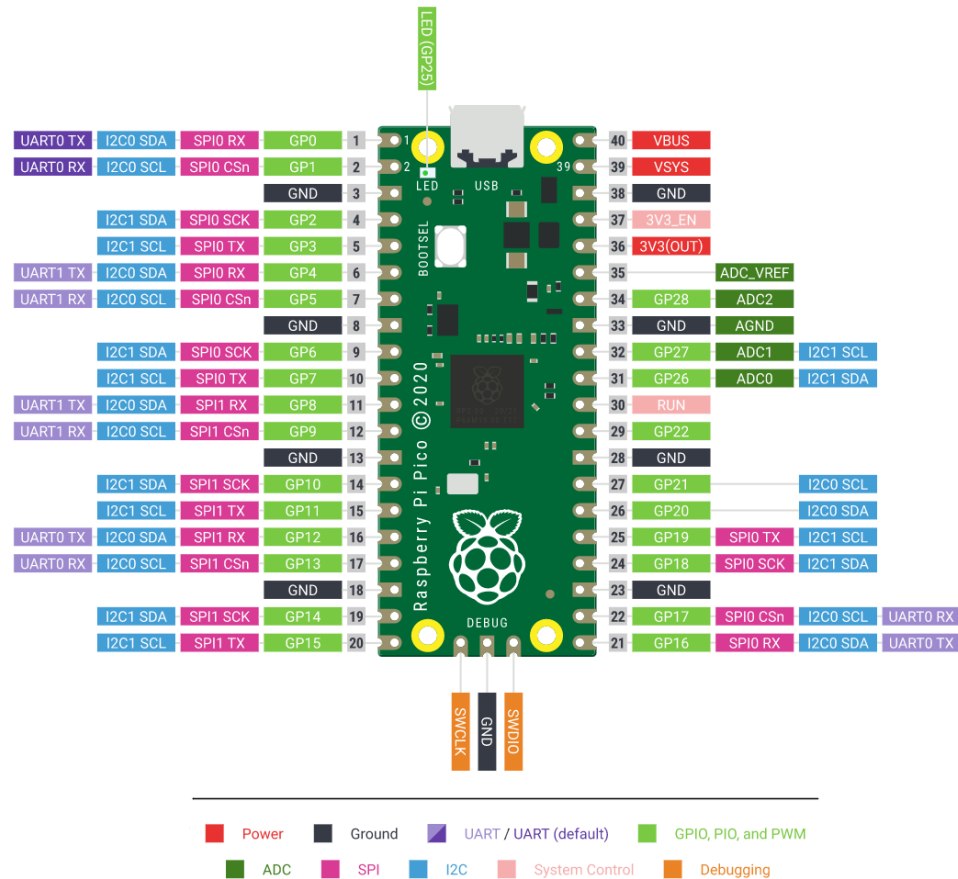


Figura 3.2: Raspberry pi Pico Pinout

Pico (RP π Pico) embarca un procesador de doble núcleo dual-core ARM-CORTEX-M0+ (hasta 133 MHz) con una memoria SRAM interna de 264kB. La arquitectura del procesador ARM-CORTEX-M0+ es de 32 bits. La RP π Pico cuenta con 2MB de memoria flash y un chip de *power supply* que soporta voltajes de entrada de entre 1.8 y 5.5 V. El microcontrolador (ver figura 3.2) presenta 26 pines GPIO (General Purpose In-Out) incluyendo pines programables y soportando los protocolos de comunicación I²C y SPI.

3.1.2. Librerías

La Raspberry pi Pico puede programarse en MicroPython [`micropython-raspberrypiPico`] o en C/C++ [`c/c++-raspberrypiPicot`]. En el presente trabajo se ha optado por la programación en C/C++. Para empezar a programar la Raspberry pi Pico en C/C++ es necesario definir un *environment* e instalar una serie de librerías tales como SDK, OpenOCD, GDB y picoprobe. El proceso de instalación y los pasos a seguir para configurar el *environment* están descritos en [`c/c++-raspberrypiPicot`].

La librería más importante es la librería SDK del chip RP2040 (<https://raspberrypi.github.io/pico-sdk-doxxygen/>) [`c/c++-raspberrypiPico`, `sdk-raspberrypiPico`]. La librería SDK (Software Development Kit) proporciona *headers*, funciones y métodos para escribir programas en dispositivos basados en el chip RP2040 tales como la Raspberry pi Pico. Estos métodos permiten controlar distintos elementos del hardware de la Raspberry pi Pico tales como los buses I2C, SPI y los *timers*.

Un único programa se ejecuta en el dispositivo cada vez bajo el método convencional `main()`. El sistema soporta las librerías estándar de C/C++ e incluye múltiples APIs (Application Programming Interface) para acceder al hardware del RP2040, incluyendo DMA, IRQs y una amplia variedad de pines programables y periféricos de función definida. Así mismo, la SDK provee librerías de alto nivel para manejar temporizadores, USB, sincronizaciones y programación multi-núcleo. La librería SDK utiliza CMake para compilar.

3.1.3. Picoprobe

Los archivos compilados pueden cargarse en el microcontrolador de dos formas. En primer lugar, es posible conectar la Raspberry pi Pico como si fuese un dispositivo de almacenamiento USB manteniendo pulsado el botón de BOOTSEL y conectando el cable USB. De esta manera podremos cargar los archivos compilados en la Raspberry copiándolos en la carpeta correspondiente. A continuación, deberemos desconectar el dispositivo y volver a enchufarlo para que el código cargado se ejecute. Este proceso es simple pero tedioso si se desean realizar pequeños cambios del código de manera constante.

La segunda opción es utilizar un segundo controlador Raspberry pi Pico conectado al ordenador por el puerto USB y a la Raspberry por el puerto UART. Esto permite utilizar `openOCD` para cargar los archivos compilados y resetear el microcontrolador desde la línea de comandos del ordenador. Este método, denominado `picoprobe` permite reprogramar el dispositivo de manera mucho más rápida y eficiente y es el que se ha adoptado en el presente trabajo. Los detalles pueden encontrarse en el Apéndice A de [`c/c++-raspberrypiPico`]. El circuito de la figura 3.1 muestra dos microcontroladores Raspberry pi Pico conectados en la configuración de `picoprobe`.

3.2. Sensor MPU 9250

El sensor MPU 9250 [`mpu9250datasheet`] es un módulo multi-chip que consiste en en dos *dies* integrados en un único paquete QFN (ver figura 3.3). El primero de los *dies* alberga un giroscopo y un acelerómetro de tres ejes. El segundo *die* contiene el magnetómetro de tres ejes AK8963. Por tanto, el sensor MPU 9250 es un dispositivo de *Motion Tracking* de 9 ejes que combina un giroscopo de 3 ejes, un acelerómetro de 3 ejes y un magnetómetro

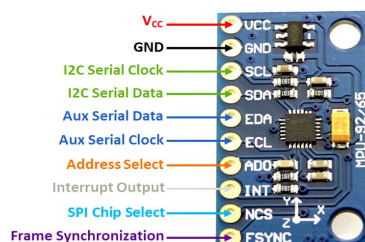


Figura 3.3: Sensor MPU 9250

de 3 ejes así como un Procesador de Movimiento Digital (DMP).

El MPU9250 puede conectarse a otros dispositivos mediante los buses de comunicación I²C (frecuencia de acceso a registros de memoria de 400 kHz) y SPI (frecuencia de acceso a registros de memoria de 1 MHz). El voltaje de operación del dispositivo es de 2.4 – 3.6 V. El sensor cuenta con nueve conversores analógico-digitales de 16 bits para digitalizar las lecturas del giróscopo, el acelerómetro y el magnetómetro. Para la detección precisa de movimientos rápidos y lentos, los diferentes sensores cuentan con distintos rangos de medida programables: ± 250 , ± 500 , ± 1000 , $\pm 2000^\circ/\text{sec}$ (dps) para el giróscopo, y ± 2 g, ± 4 g, ± 8 g, ± 16 g para el acelerómetro. Si no se especifica lo contrario, en lo que sigue los rangos de medida programados para el giroscopio y el acelerómetro son de $\pm 1000^\circ/\text{sec}$ y ± 4 g respectivamente. El magnetómetro tiene un rango de medida fijo de $\pm 4800\mu T$. El sensor cuenta con filtros digitales programables integrados, un reloj de precisión con una deriva del 1 % en el rango de temperatura $40^\circ\text{C} - 85^\circ\text{C}$, un sensor de temperatura e *interrupts* programables. Se trata de un sensor de bajo coste (a partir de 1.59 euros) lo que lo convierte en un buen candidato para la estimación de la actitud en vehículos autónomos de pequeño tamaño y en aplicaciones académicas y de prototipado.

La comunicación entre el MPU9250 y el microcontrolador se realiza mediante el protocolo de comunicación I²C [paret1997i2c]. En el contexto del presente trabajo se ha desarrollado una librería que permite acceder a las lecturas de los sensores que integra la Unidad de Medida Inercial. Esta librería se denomina `IMU.h` y permite inicializar los parámetros de acelerómetro y el giróscopo, así como aplicar la calibración determinada en el capítulo 4 a las medidas de los sensores. El acceso a los registros de memoria del sensor se realiza mediante las funciones `I2CwriteByte` e `I2Cread`.

Capítulo 4

Caracterización de los sensores

4.1. Generalidades

4.1.1. Random walk

Los sensores están sometidos a diferentes fuentes de ruido que afectan a los resultados obtenidos a la hora de estimar las variables de estado del sistema. El error en los sensores es especialmente relevante cuando las mediciones de los sensores se integran en el tiempo. Cuando una señal con ruido se integra en el tiempo los errores se van acumulando dando lugar a una deriva *drift* en la integral de la señal.

4.1.2. Dealineación interna de los sensores

4.2. Calibración del giróscopo.

La calibración del giróscopo se ha realizado en este caso utilizando un tiempo de muestreo de 50 ms y un rango de medida de $\pm 1000^\circ/\text{s}$. La calibración del giróscopo consiste principalmente en determinar los valores del bias. Para ello se miden los valores de la velocidad en los tres ejes con el sensor estático. El bias en cada dirección se determina calculando el valor medio de la señal. La figura 4.1 representa los valores de la velocidad angular medidos durante un intervalo de tiempo de 30 s con y sin bias.

La tabla 4.1 representa los valores del bias y de la desviación típica de las lecturas del giróscopo en cada eje.

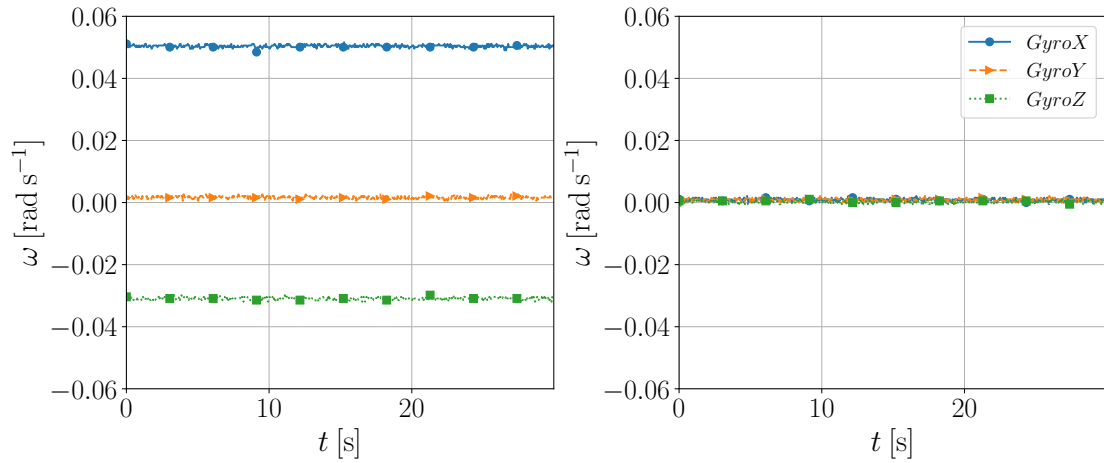


Figura 4.1: Mediciones estáticas del giroscopio: con bias (izquierda) y sin bias (derecha)

Eje	Bias [rad/s]	STD [rad/s]
ω_x	0.0504	4.199×10^{-4}
ω_y	0.0017	4.204×10^{-4}
ω_z	-0.0309	4.391×10^{-4}

Tabla 4.1: Caracterización del giróscopo.

4.3. Calibración del acelerómetro

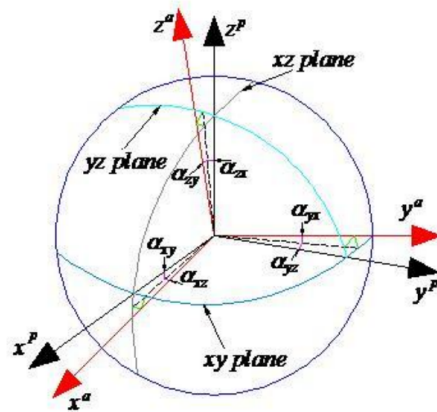


Figura 4.2: Errores de alineamiento entre los ejes del acelerómetro y el sistema de ejes cuerpo.

Idealmente, las direcciones en las que el sensor mide la aceleración deberían de ser

k_x	k_y	k_z	a_{yz} [rad]	a_{zy} [rad]	a_{zx} [rad]	b_x [m s ⁻²]	b_y [m s ⁻²]	b_z [m s ⁻²]
0.9957	0.9898	0.9666	0.01423	0.05225	0.00074	-0.9497	-1.5212	-1.5564

Tabla 4.2: Parámetros de calibración del acelerómetro

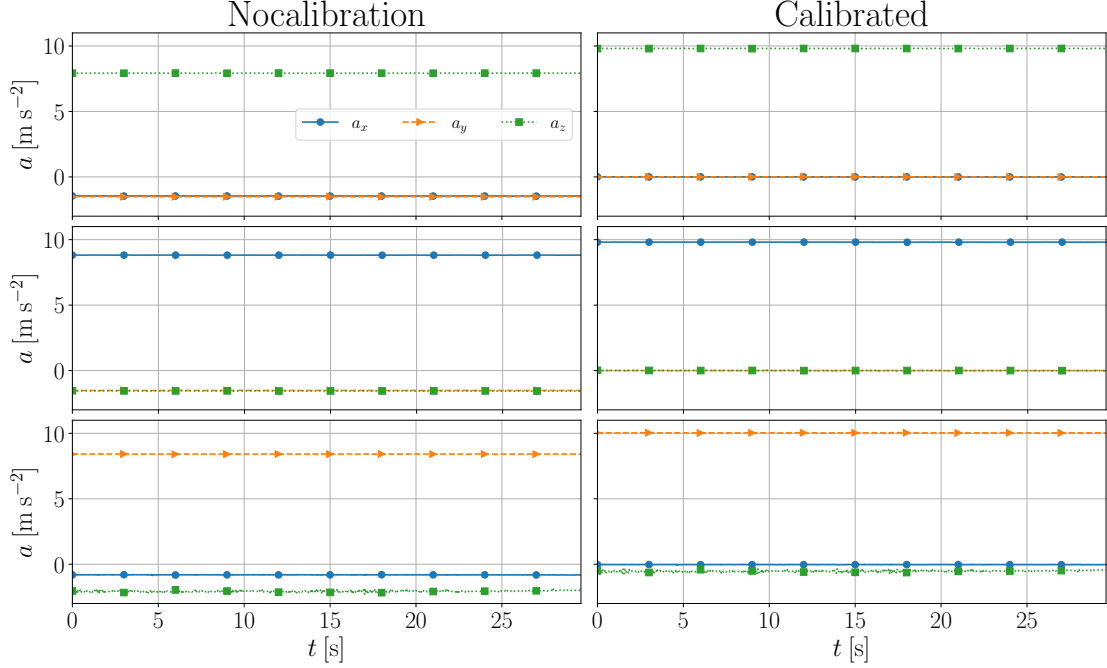


Figura 4.3: Lecturas estáticas del acelerómetro para tres orientaciones diferentes previas a la calibración (izquierda) y posteriores a la calibración (derecha).

ortogonales entre sí. Sin embargo, debido a las imprecisiones en la construcción de la IMU, a menudo existen desajustes en el alineamiento de los ejes. Esto se traduce en una falta de ortogonalidad entre los ejes del sensor que causa errores en las lecturas del acelerómetro. La figura 4.2 esquematiza los errores de desalineamiento entre los ejes del sensor y el sistema de ejes cuerpo $-b$. En esta sección se presenta un método para corregir los errores de alineamiento mediante de una calibración estática del acelerómetro. Los detalles del método pueden encontrarse en [tee2011triaxial]. En principio, se requiere conocer un total de seis ángulos de desviación, $(a_{xy}, a_{xz}, a_{yx}, a_{yz}, a_{zx}, a_{zy})$, entre los ejes del sensor y los ejes cuerpo para corregir los errores de ortogonalidad. Sin embargo, [skog2006calibration] mostraron que el modelo de error puede simplificarse si se asume que el eje x_b coincide con el eje x_a . Las lecturas del sensor pueden modelarse de acuerdo al siguiente modelo de error,

$$v_i = ST^{-1}g_i + b + \epsilon \quad (4.1)$$

donde v_i es una lectura bruta, i , del acelerómetro, S es una matriz diagonal que contiene los factores de escala, T es la matriz de desalineación, b es el bias constante de cada uno de

los ejes del sensor y ϵ es el ruido. El vector g_i es el vector de gravedad. La matriz T permite transformar un vector expresado en ejes acelerómetro $-a$ a ejes cuerpo $-b$. Su expresión es,

$$T = \begin{bmatrix} 1 & -a_{yz} & -a_{zy} \\ 0 & 1 & -a_{zx} \\ 0 & 0 & 1 \end{bmatrix} \quad (4.2)$$

A partir de la ecuación (4.1), un total de nueve parámetros recogidos en (4.3) deben ser determinados para corregir las lecturas del sensor.

$$[k_x \ k_y \ k_z \ a_{yz} \ a_{zy} \ a_{zx} \ b_x \ b_y \ b_z] \quad (4.3)$$

La ecuación (4.1) puede expandirse para obtener,

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}_i = \begin{bmatrix} b_x + k_x g_{xi} + a_{yz} k_x g_{yi} - k_x g_{zi} (a_{zy} - a_{yz} a_{zx}) \\ b_y + k_y g_{yi} + a_{zx} k_y g_{zi} \\ b_z + k_z g_{zi} \end{bmatrix}_i \quad (4.4)$$

El subíndice $-i$ denota la muestra i . Por tanto, utilizando un número i de muestras correspondiente al número de incógnitas en cada eje, la ecuación (4.4) puede descomponerse en tres sistemas lineales de ecuaciones,

$$\begin{bmatrix} v_{z1} \\ v_{z2} \end{bmatrix} = \begin{bmatrix} 1 & g_{z1} \\ 1 & g_{z2} \end{bmatrix} \begin{bmatrix} b_z \\ k_z \end{bmatrix} \quad (4.5)$$

$$\begin{bmatrix} v_{y1} \\ v_{y2} \\ v_{y3} \end{bmatrix} = \begin{bmatrix} 1 & g_{y1} & g_{z1} \\ 1 & g_{y2} & g_{z2} \\ 1 & g_{y3} & g_{z3} \end{bmatrix} \begin{bmatrix} b_y \\ k_y \\ k_{yzx} \end{bmatrix} \quad (4.6)$$

$$\begin{bmatrix} v_{x1} \\ v_{x2} \\ v_{x3} \\ v_{x4} \end{bmatrix} = \begin{bmatrix} 1 & g_{x1} & g_{y1} & -g_{z1} \\ 1 & g_{x2} & g_{y2} & -g_{z2} \\ 1 & g_{x3} & g_{y3} & -g_{z3} \\ 1 & g_{x4} & g_{y4} & -g_{z4} \end{bmatrix} \begin{bmatrix} b_x \\ k_x \\ k_{xyz} \\ k_{xzy} \end{bmatrix} \quad (4.7)$$

donde

$$a_{zx} = \frac{k_{yzx}}{k_y} \quad (4.8)$$

$$a_{yz} = \frac{k_{xyz}}{k_x} \quad (4.9)$$

$$a_{zy} = \frac{k_{xzy}}{k_x} - a_{yz} a_{zx} \quad (4.10)$$

Para determinar los coeficientes de calibración del acelerómetro (4.3) es necesario realizar n mediciones del campo gravitatorio en n orientaciones diferentes para las que se conozca

el valor teórico de la dirección del vector de gravedad \mathbf{g} . Siendo k el número de incógnitas por eje y n el número de orientaciones evaluadas, existen un total de,

$$\mathcal{C} = \frac{n!}{k!(n-k)!} \quad (4.11)$$

combinaciones de medidas por cada eje para calcular los coeficientes de calibración. Aumentando el número de direcciones evaluadas n es posible reducir el error en la estimación de (4.3). La tabla 4.2 muestra los parámetros de calibración obtenidos tras haber evaluado cuatro orientaciones del sensor. Nótese que los valores de los factores de escala son adimensionales y pueden utilizarse para reescalar el módulo total del vector gravedad.

La figura 4.3 representa las lecturas del sensor en tres orientaciones distintas junto con

4.4. Calibración del magnetómetro.

Localización	Latitud [N]	Longitud [W]	h_x^I [nT]	h_y^I [nT]	h_z^I [nT]	δh_x^I [nT]	δh_y^I [nT]	δh_z^I [nT]
Madrid	40° 25' 10"	3° 41' 31"	25749.4	97.1	36931.8	131	94	157
Toulouse	43° 36' 16"	1° 26' 39"	23869.8	637.5	39984.3	131	94	157

Tabla 4.3: Componentes del vector del campo magnético terrestre en ejes inerciales (NED) para dos localizaciones

El magnetómetro actúa como una brújula, proporcionando las tres componentes del vector del campo magnético terrestre medidas en ejes cuerpo $-b$.

$$\mathbf{h}_b = [h_x, h_y, h_z]_b \quad (4.12)$$

El vector en ejes cuerpo debe ser transformado a ejes inerciales para poderlo comparar con la dirección del campo magnético terrestre en esa posición, que es un vector de referencia conocido. La dirección del vector del campo magnético terrestre varía a lo largo de la superficie terrestre pero puede considerarse constante para pequeños desplazamientos. A partir de las coordenadas de un punto de la superficie terrestre es posible obtener los valores de las componentes del vector del campo magnético utilizando herramientas tales como <https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml?#igrfwmm>. En la tabla 4.3 se recogen las componentes del campo magnético para dos localizaciones distintas junto con sus incertidumbres.

La comparación entre las lecturas del magnetómetro y el valor de referencia del campo magnético puede utilizarse para estimar la actitud del sistema de referencia ejes cuerpo $-b$. En contraste con el acelerómetro, el vector de referencia del campo magnético tiene

componentes no nulas en el plano $x_I y_I$ por lo que permite estimar el ángulo de guiñada ψ . Otra de las ventajas del magnetómetro es que sus medidas no se ven afectadas por las aceleraciones externas, que sí afectan a las lecturas del acelerómetro.

A día de hoy existen diferentes tipos de sensores del campo magnético basados en diferentes tecnologías. Por una parte podemos encontrar los sensores magnetorresistivos anisotrópicos (AMR por sus siglas en inglés). En este tipo de sensores existen elementos resistivos cuya resistencia efectiva cambia cuando son sometidos a un campo magnético. Su sensibilidad al campo magnético depende de la dirección del campo magnético incidente con respecto a la orientación del elemento magnetorresistivo, de ahí la denominación de anisotrópicos. Para la medición del campo magnético es necesario disponer cuatro elementos magnetorresistivos en configuración de puente de Wheatstone y medir el voltaje de salida. Es posible encontrar más detalles sobre esta tecnología en [renaudin2010complete, freitas2007magnetoresistive].

Por otra parte se encuentran los sensores magnéticos de efecto Hall [ramsdén2011hall]. Los sensores magnéticos de efecto Hall son dispositivos electrónicos que detectan la presencia de un campo magnético utilizando el efecto Hall, descubierto por el físico estadounidense Edwin Hall en 1879. **El efecto Hall se refiere al fenómeno de generar un potencial de voltaje en un conductor cuando se coloca en un campo magnético perpendicular y fluye una corriente a través de él.** Los sensores de efecto Hall consisten en una delgada tira de material semiconductor (como arseniuro de galio o silicio) a través de la cuál se hace pasar una corriente eléctrica. Al aproximar el sensor a un campo magnético perpendicular a la dirección de la corriente, se generará un voltaje proporcional al producto de la fuerza del campo magnético y la corriente. Conociendo el valor de la corriente, es posible calcular la fuerza del campo magnético. De entre las diferentes tecnologías de detección del campo magnético, los sensores de efecto Hall son los más extendidos. Una de las principales razones es que es posible producir sensores de gran calidad utilizando las técnicas convencionales de construcción de circuitos integrados. Esto permite integrar el sensor dentro de un chip de manera sencilla y barata. **Los sensores de efecto Hall se utilizan ampliamente en una variedad de aplicaciones, incluyendo sistemas de control automotriz e industrial, robótica y electrónica de consumo. Son preferidos sobre otros tipos de sensores magnéticos en muchas aplicaciones porque son altamente sensibles, tienen un tiempo de respuesta rápido y no se ven afectados por factores ambientales como la temperatura, la humedad o la vibración.**

No obstante, el magnetómetro sufre de una serie de limitaciones que hacen que su calibración sea extremadamente importante para poderlo utilizar de forma efectiva en la estimación de la actitud del vehículo. Una descripción exhaustiva sobre las fuentes de error en los sensores de campo magnético magnetorresistivos puede encontrarse en [renaudin2010complete]. De forma resumida, este tipo de sensores están sometidos a errores en los factores de escala asociados a la falta de proporcionalidad entre la salida del sensor y el campo magnético detectado, errores de offset debido a la presencia de campos parásitos, errores debido a la sensibilidad cruzada entre ejes o desalineamiento y, finalmen-

te, a errores debido al ruido. De entre las fuentes de error que causan errores de offset y de desalineamiento podemos destacar los errores de hierro. Los errores de hierro están causados por las perturbaciones magnéticas originadas por la presencia de materiales ferromagnéticos y sistemas electromagnéticos en las proximidades del sensor. Estos errores están causados por los diferentes elementos de la planta donde se instala el sensor y dependen por tanto de su configuración. Esto hace que la calibración del sensor sea diferente en función de si este se instala en un placa de prototipado o en la planta final. Los errores de hierro se clasifican a su vez en errores de hierro duro y blando. Los errores de hierro duro están causados por fuentes de campo magnético cuyas componente del campo magnético son constantes en todas la direcciones. En otras palabras, son perturbaciones del campo magnético generadas por los diferentes subsistemas electrónicos presentes en las proximidades del sensor y que no dependen del campo magnético terrestre. Por otra parte, los errores de hierro blando surgen de la interacción entre el campo magnético terrestre y los elementos ferromagnéticos presentes en las proximidades del sensor. Estas perturbaciones tienen una naturaleza más compleja ya que dependen de la dirección del campo magnético terrestre incidente y, por tanto, de la orientación del sensor.

A continuación se presenta el modelo de error empleado para calibrar el magnetómetro y se describe el procedimiento de calibración. Sea \hat{h} el vector del campo magnético medido por el sensor y h el vector del campo magnético real en ejes cuerpo, el modelo de error de las medidas del magnetómetro que permite relacionar ambos términos viene dado por la ecuación (4.13).

$$\hat{h} = SM(A_{si}h + b_{hi}) + b_m + \epsilon \quad (4.13)$$

donde,

- S es la matriz diagonal de factores de escala,

$$S = \text{diag}(s_x, s_y, s_z) \quad (4.14)$$

Los factores de escala permiten modelar los errores de proporcionalidad entre la salida del sensor y las componentes del campo magnético en las tres direcciones del espacio. Además, este término permite normalizar la salida del sensor.

- M es la matriz de desalineamiento, modela los errores de ortogonalidad entre los ejes del sensor y viene dada por

$$M = N^{-1} = [\mathbf{m}_x \ \mathbf{m}_y \ \mathbf{m}_z]^{-1} \quad (4.15)$$

donde \mathbf{m}_x , \mathbf{m}_y y \mathbf{m}_z son vectores columna de dimensión tres que definen la orientación de los ejes del sensor $F(O_s, x_s, y_s, z_s)$ con respecto a los ejes cuerpo $F(O_b, x_b, y_b, z_b)$.

- b_m representa el sesgo (bias) en la salida del sensor modelado como un *offset* (desplazamiento)

$$b_m = [b_{m,x} \ b_{m,y} \ b_{m,z}]^T \quad (4.16)$$

- b_{hi} modela los errores de desviación magnética de tipo hierro duro. La presencia de campos magnéticos parásitos causados por fuentes de campo magnético presentes en las proximidades del sensor se representa mediante un desplazamiento o sesgo.

$$b_{hi} = [b_{hi,x} \quad b_{hi,y} \quad b_{hi,z}]^T \quad (4.17)$$

- A es la matriz que modela los errores de hierro suave causados por fuentes externas de campo magnético que cambian dependiendo tanto de la magnitud como de la dirección de las lecturas del sensor. La matriz A viene dada por,

$$A_{si} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad (4.18)$$

La ecuación (4.13) puede reescribirse como,

$$\hat{h} = A \cdot h + b + \epsilon \quad (4.19)$$

donde

$$A = SMA_{si} \quad y \quad b = SMb_{hi} + b_m \quad (4.20)$$

La matriz A combina los errores de factor de escala, desalineamiento y perturbaciones de hierro suave mientras que el vector b agrupa los errores de sesgo (bias).

A partir del modelo de error descrito podemos manipular la ecuación (4.20) para aislar el campo magnético h real en ejes cuerpo.

$$h = A^{-1}(h_m - b) \quad (4.21)$$

donde h es el vector del campo magnético real en ejes cuerpo y h_m es el vector del campo magnético medido por el sensor y sometido a las diferentes fuentes de error descritas. La calibración consiste en estimar los términos de la matriz A y del vector b para poder corregir las lecturas del sensor empleando la ecuación (4.21). De ahora en adelante, las lecturas del sensor se asumen normalizadas por el módulo del campo magnético de manera que el vector del campo magnético h satisface,

$$h^T h = 1 \quad (4.22)$$

Substituyendo la ecuación (4.21) en (4.22), se obtiene.

$$(h_m - b)^T (A^{-1})^T A^{-1} (h_m - b) = 1 \quad (4.23)$$

Se por Q el producto $Q = (A^{-1})^T A^{-1}$. Es fácil comprobar que Q es una matriz simétrica ya que es el producto de una matriz multiplicada por su transpuesta. Sustituyendo se obtiene,

$$(h_m - b)^T Q (h_m - b) = 1 \quad (4.24)$$

Calibración del magnetómetro

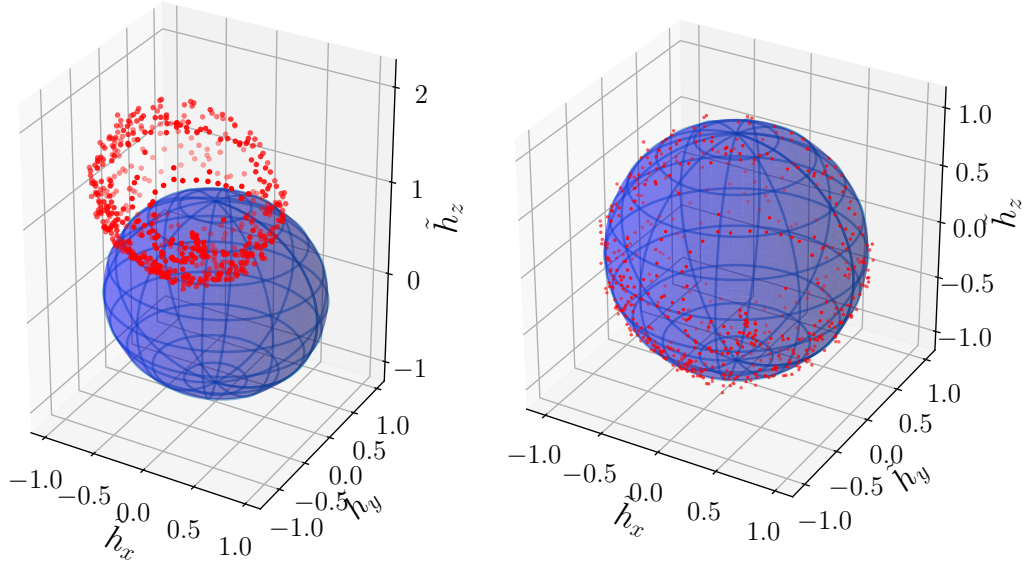


Figura 4.4: Lugar geométrico de los puntos del espacio descrito por el vector del campo magnético medido por el sensor cuando le se somete a rotaciones aleatorias en las tres direcciones del espacio. A la izquierda se representan las lecturas brutas previas a la calibración y una esfera de radio unitario centrada en el origen. A la derecha se representan esas mismas lecturas tras aplicar la calibración descrita. Las lecturas del magnetómetro se han normalizado con el módulo del vector del campo magnético en Toulouse (ver tabla 4.3.)

Expandiendo la ecuación anterior y simplificando se llega a,

$$h_m^T Q h_m - h_m^T Q b - b^T Q h_m + b^T Q b - 1 = 0 \quad (4.25)$$

dado que Q es una matriz simétrica, $b^T Q h = h^T Q b$. Por tanto, podemos reescribir la ecuación (4.25) como

$$h_m^T Q h_m - 2h_m^T Q b + b^T Q b - 1 = 0 \quad (4.26)$$

La ecuación (4.26) puede finalmente escribirse como la ecuación de una forma cuadrática

$$h_m^T Q h_m + h_m^T n + d = 0 \quad (4.27)$$

donde

$$Q = (A^{-1})^T A^{-1} \quad (4.28)$$

$$n = -2Qb \quad (4.29)$$

$$d = b^T Q b - 1 \quad (4.30)$$

El método de calibración del magnetómetro consiste en una calibración tridimensional que emplea el ajuste por mínimos cuadrados y se presenta con detalle en [kok2012calibration,

renaudin2010complete, magnusson2013state]. En ausencia de perturbaciones ni errores de desalineamiento, el módulo del vector del campo magnético medido por el sensor debe ser igual al módulo campo magnético terrestre independientemente de la orientación del sensor. En consecuencia, cuando el sensor se gira en los tres ejes del espacio, el vector del campo magnético medido por el sensor en ejes cuerpo debe describir una esfera de radio igual al módulo del vector del campo magnético. Debido a las diferentes fuentes de error descritas anteriormente, la traza del vector del campo magnético obtenida al girar el sensor formará un elipsoide.

La calibración consiste en encontrar los parámetros Q , n y d que satisfacen la ecuación (4.27). Se trata de un problema de ajuste de elipsoides utilizando las lecturas del sensor recopiladas para diferentes orientaciones. De entre los diferentes métodos de ajuste de elipsoides existentes es posible citar los algoritmos algebraicos [**li2004least, fitzgibbon1999direct**], geométricos [**cheng1999statistical**] y adaptativos [**markovsky2004consistent**]. En la presente implementación se ha utilizado la metodología descrita en [**li2004least**].

La calibración requiere medir el campo magnético en un gran número de orientaciones espaciales diferentes. A diferencia del acelerómetro, que requiere una calibración estática, el magnetómetro puede ser calibrado de forma dinámica, registrando los valores de las lecturas del magnetómetro durante un periodo de tiempo y girando el sensor en las tres direcciones del espacio. No obstante, debido a los ya mencionados errores de hierro duros y suaves, para ser efectiva la calibración debe realizarse directamente en la planta final. Esta operación puede resultar complicada en sistemas reales. Otros métodos de calibración 2D como el presentado por [**caruso1998applications**] permiten simplificar la fase de adquisición de datos para la calibración.

Tras ajustar los valores de Q , n y d utilizando los datos experimentales es posible calcular la matriz A aplicando,

$$b = -Q^{-1}n \quad (4.31)$$

$$A^{-1} = \frac{M^{1/2}}{\sqrt{n^T M^{-1} n - d}} \quad (4.32)$$

Capítulo 5

Extended Kalman Filter

In order to use the Kalman Filter, we first have to define the states that we want to use. This is why there are so many different kalman filter implementations out there. Every author out there is saying that using their chosen states, you will be able to achieve a better result. However, for the purpose of this tutorial, we are just going to implement a really simple (in comparison to other implementations) one which is similar to the one implemented in my other post for determining a 1 dimensional angle using kalman filter.

Although I said that this is simple relatively, it is actually not that simple. Moving from 2D to 3D is really a big jump as you have to start using vectors and matrices, but that's not something to be afraid of so let us press on.

The states that we will be using for this implementation is given as follows:

$$x = [q_0 \quad q_1 \quad q_2 \quad q_3 \quad b_1^g \quad b_2^g \quad b_3^g]^T \quad (5.1)$$

where

q_0 is the scalar term in the quaternion q_1, q_2, q_3 is the vector term in the quaternion b_1^g, b_2^g, b_3^g is the bias of the gyrometer in the x, y, z direction respectively (units is the same as angular velocity).

The bias term refers to how much the gyrometer would have drifted per unit time. For more information, you can read up my other post for a 1 dimensional kalman filter implementation.

For this implementation, we can write the states in a more concise and compact manner as shown below.

$$x = \begin{bmatrix} \tilde{q} \\ \tilde{b}^g \end{bmatrix} \quad (5.2)$$

where \tilde{q} is the quaternion and \tilde{b}^g is the bias vector

Now, we need to form an equation for the system dynamics. From section 3, we know

that

$$\dot{q} = \frac{1}{2}S(w)q = \frac{1}{2}S(q)w \quad (5.3)$$

where

$$S(w) = \begin{bmatrix} 0 & -w_1 & -w_2 & -w_3 \\ w_1 & 0 & w_3 & -w_2 \\ w_2 & -w_3 & 0 & w_1 \\ w_3 & w_2 & -w_1 & 0 \end{bmatrix} \quad (5.4)$$

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \quad (5.5)$$

$$S(q) = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \quad (5.6)$$

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \quad (5.7)$$

In order to compensate for the gyrometer bias, we will need to add in the bias term into equation (2).

$$\dot{q} = \frac{1}{2}S(w - b^g)q \quad (5.8)$$

$$= \frac{1}{2}S(w)q - \frac{1}{2}S(b^g)q \quad (5.9)$$

$$= \frac{1}{2} \begin{bmatrix} 0 & -w_1 & -w_2 & -w_3 \\ w_1 & 0 & w_3 & -w_2 \\ w_2 & -w_3 & 0 & w_1 \\ w_3 & w_2 & -w_1 & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 0 & -b^g_1 & -b^g_2 & -b^g_3 \\ b^g_1 & 0 & b^g_3 & -b^g_2 \\ b^g_2 & -b^g_3 & 0 & b^g_1 \\ b^g_3 & b^g_2 & -b^g_1 & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \quad (5.10)$$

Next, we will need to discretize the above equation so that we can implement it in our program that runs in discrete time. Here, we are simply going to use the first order linearized model to make things simpler. Therefore,

$$\dot{q}(k) = [q(k+1) - q(k)]/T \quad (5.11)$$

where T is the time taken between sample $k+1$ and k . Rearranging the equation,

$$q(k+1) = T\dot{q}(k) + q(k) \quad (5.12)$$

Substituting equation (3) into equation (4),

$$q(k+1) = \frac{T}{2}S(w)q(k) - \frac{T}{2}S(b^g)q(k) + q(k) \quad (5.13)$$

We can actually write equation (4) in a different manner as shown below such that it will be more meaningful when written in matrix form later.

$$q(k+1) = \frac{T}{2}S(q(k))w - \frac{T}{2}S(q(k))b^g + q(k) \quad (5.14)$$

Thus, the system state equation can be written as follows.

$$x_{k+1} = \begin{bmatrix} I_{4 \times 4} & -\frac{T}{2}S(q) \\ 0_{3 \times 4} & I_{3 \times 3} \end{bmatrix}_k x_k + \begin{bmatrix} \frac{T}{2}S(q) \\ 0_{3 \times 3} \end{bmatrix}_k w_k \quad (5.15)$$

$$\begin{bmatrix} q \\ b^g \end{bmatrix}_{k+1} = \begin{bmatrix} I_{4 \times 4} & -\frac{T}{2}S(q) \\ 0_{3 \times 4} & I_{3 \times 3} \end{bmatrix}_k \begin{bmatrix} q \\ b^g \end{bmatrix}_k + \begin{bmatrix} \frac{T}{2}S(q) \\ 0_{3 \times 3} \end{bmatrix}_k w_k \quad (5.16)$$

If you expand equation (7), you will find that the first row will resolve into equation (6), and the second row simply says that the bias at time $k+1$ is the same as the bias at time k . This may look silly at first because we are just saying that the bias is constant. However, the Kalman Filter alters the states of the equation thus the value of the bias actually changes over time when we actually implement the algorithm. For the sake of clarity, I am going to expand equation (7) so that there will be no confusion. Below is the expanded form of equation (7).

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ b_x^g \\ b_y^g \\ b_z^g \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & -\frac{T}{2}(-q_1) & -\frac{T}{2}(-q_2) & -\frac{T}{2}(-q_3) \\ 0 & 1 & 0 & 0 & -\frac{T}{2}(q_0) & -\frac{T}{2}(-q_3) & -\frac{T}{2}(q_2) \\ 0 & 0 & 1 & 0 & -\frac{T}{2}(q_3) & -\frac{T}{2}(q_0) & -\frac{T}{2}(-q_1) \\ 0 & 0 & 0 & 1 & -\frac{T}{2}(-q_2) & -\frac{T}{2}(q_1) & -\frac{T}{2}(q_0) \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ b_x^g \\ b_y^g \\ b_z^g \end{bmatrix}_k + \frac{T}{2} \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}_k \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}_k \quad (5.17)$$

Before we move on to the actual implementation, let us first take a look at how we can use the accelerometer and magnetometer data in order get the reference vectors.

5.1. Accelerometer data

The acceleration measured by the accelerometer module can be calculated if we know the acceleration of the body in the world frame, and the orientation of the body. Our

reference vector is going to be the gravity vector, and we know that this vector will always point downwards in the world frame. Therefore, if we know the orientation of the body, we can predict the acceleration that the accelerometer is going to measure. This is all based on the assumption that external forces are negligible thus the only force that acts on the body is the gravitational force. As a result of this assumption, you will find that once you move the body hectically, the algorithm will not perform well. There are of course ways to alleviate this problem, but we will not go into there for the purpose of this tutorial.

The above paragraph can be summarized into the following equation.

$${}^b a_m = {}^b R_w(-g) + {}^b e_a + {}^b b_a \quad (5.18)$$

where

${}^b a_m$ is the measured acceleration in the body frame by the accelerometer, ${}^b R_w$ is the rotation matrix for world frame to body frame, g is the gravitational vector in the world frame, $g = [0 \ 0 \ 1]_T$, ${}^b e_a$ is the accelerometer noise in the body frame and ${}^b b_a$ is the accelerometer bias in the body frame

Since we can determine all the variables on the right hand side (except for the noise term), it is possible to predict the measured acceleration. We can then use this prediction to compare with the actual measured acceleration to determine the error in our orientation. The Extended Kalman Filter (EKF) will then automatically help us convert this error into the bias term of the gyrometer. This is what I call the magic of the EKF because I do not need to know how the conversion is done. EKF handles it all as you will see in the later section.

For now, let us define some of the variables above. From the quaternion, it is possible to derive the rotation matrix with the following equation. Do read up on my previous post if you need more information on this.

$${}^b R_w = {}^b R_w^T = R(q)^T = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}^T \quad (5.19)$$

$${}^b R_w = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \quad (5.20)$$

Since we know that $g = [0 \ 0 \ 1]_T$, we can substitute equation (9) into equation (8) and then simplify equation (8) to get the following equation.

$$\begin{bmatrix} {}^b a_x \\ {}^b a_y \\ {}^b a_z \end{bmatrix} = - \begin{bmatrix} 2(q_1 q_3 - q_0 q_2) \\ 2(q_2 q_3 + q_0 q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} + \begin{bmatrix} {}^b b_x \\ {}^b b_y \\ {}^b b_z \end{bmatrix} + \begin{bmatrix} {}^b e_x \\ {}^b e_y \\ {}^b e_z \end{bmatrix} \quad (5.21)$$

*take note that in the above equation (10), g is a scalar. Notice in equation (10) that the matrix with quaternion terms is non-linear. In order to use the Kalman Filter, we have to write equation (10) in the form of $y = Cx + D$ where x is the state matrix as shown in equation (1) and y is the term on the left hand side of equation (10). You will realize that this is not possible because of the non-linearity. One possible solution is that we can linearize equation (10) near its “operating point”. This is exactly what the Extended Kalman Filter is doing, and also why you need to use the extended form of the Kalman Filter when working with rotation matrices because they are mostly non-linear. So, how do we go about linearizing equation (10)? We can rewrite equation (8), the unexpanded form of equation (10), evaluated at time t as shown below.

$$({}^b a_m)_k = -g h_a(q_k) + C_k \quad (5.22)$$

where $({}^b a_m)_k$ is the measured acceleration in the body frame by the accelerometer at time step k , g is the gravitational constant (take note that g is a scalar here), $h_a(q_k)$ represents a non-linear function in q_k , where q_k is the quaternion at time step k , C_k represents the other terms in equation (8), ${}^b e_a + {}^b b_a$, evaluated at time step k .

Just for clarity, I am going to write the values of $h_a(q_k)$ below so that there will be no misunderstandings.

$$h_a(q_k) = \begin{bmatrix} 2(q_1 q_3 - q_0 q_2) \\ 2(q_2 q_3 + q_0 q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}_k \quad (5.23)$$

With this, what we need to do now becomes much clearer. We have to linearize the nonlinear function $h_a(q_k)$, and one way of doing so is to find its Jacobian (gradient) at time $k - 1$. In doing so, we are assuming that the non-linear function is approximately linear between 2 points adjacent in time. This also means that the approximation will be better if you have a shorter timestep between your iterations.

In order to linearize a function, we will call upon the mighty Taylor Expansion as shown below.

$$f(x)|_{x=a} = f(a) + f'(a)(x - a) + \frac{f''(a)}{2}(x - a)^2 + \dots \quad (5.24)$$

where $f(x)$ is a non-linear function in x . Applying the above to our non-linear function, we will get the following equation.

$$h_a(q_k)|_{q_k=q_{k-1}} = h_a(q_{k-1}) + h'_a(q_{k-1})(q_k - q_{k-1}) + \dots \quad (5.25)$$

where,

$$h'_a(q_{k-1}) = \frac{\delta h_a(q)}{\delta q}|_{q=q_{k-1}} = \begin{bmatrix} \frac{\delta h_{a1}}{\delta q_0} & \frac{\delta h_{a1}}{\delta q_1} & \frac{\delta h_{a1}}{\delta q_2} & \frac{\delta h_{a1}}{\delta q_3} \\ \frac{\delta h_{a2}}{\delta q_0} & \frac{\delta h_{a2}}{\delta q_1} & \frac{\delta h_{a2}}{\delta q_2} & \frac{\delta h_{a2}}{\delta q_3} \\ \frac{\delta h_{a3}}{\delta q_0} & \frac{\delta h_{a3}}{\delta q_1} & \frac{\delta h_{a3}}{\delta q_2} & \frac{\delta h_{a3}}{\delta q_3} \end{bmatrix}_{k-1} \quad (5.26)$$

$$h'_a(q_{k-1}) = 2 \begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}_{k-1} \quad (5.27)$$

And there we have it! Let us now substitute equation (13) into equation (12).

$$h_a(q_k)|_{q_k=q_{k-1}} = \begin{bmatrix} 2(q_1 q_3 - q_0 q_2) \\ 2(q_2 q_3 + q_0 q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}_{k-1} + 2 \begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}_{k-1} \left(\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_k - \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_{k-1} \right) \quad (5.28)$$

Now that we have the linearized form of the non-linear equation, let us write out the whole linearized solution of equation (10).

$$\begin{bmatrix} {}^b a_x \\ {}^b a_y \\ {}^b a_z \end{bmatrix}_k = -g \left[\begin{bmatrix} 2(q_1 q_3 - q_0 q_2) \\ 2(q_2 q_3 + q_0 q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}_{k-1} + 2 \begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}_{k-1} \left(\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_k - \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_{k-1} \right) \right] + \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix}_k \quad (5.29)$$

Take note that g is a scalar term here. In the Python source code, you will notice that the C_k matrix is missing. This is because I calibrated the accelerometer at first to deal with the bias. In other words, I was working with a 0 bias accelerometer value so the C_k term becomes 0.

Let us now write equation (15) in a more compact matrix form so that it will be clearer.

$$y = Cx + D \quad (5.30)$$

where

$$y = ({}^b a_m)_k = \begin{bmatrix} {}^b a_x \\ {}^b a_y \\ {}^b a_z \end{bmatrix}_k \quad (5.31)$$

$$C = -g[2h'_a(q_{k-1})] = -2g \begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}_{k-1} \quad (5.32)$$

$$x = q_k = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_k \quad (5.33)$$

(here, the bias terms in equation (1) are treated as zeros)

$$D = -g[h_a(q_{k-1}) - 2h'_a(q_{k-1})(q_{k-1}) + C_k] \quad (5.34)$$

$$= -g \left[\begin{bmatrix} 2(q_1q_3 - q_0q_2) \\ 2(q_2q_3 + q_0q_1) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}_{k-1} - 2 \begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}_{k-1} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_{k-1} + \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix}_k \right] \quad (5.35)$$

We now have a system of linear equations, albeit being a little ugly. Actually, in the kalman filter implementation, we are only going to use matrix C (the Jacobian matrix) thus the rest of the terms are actually not needed. There are some mathematical proofs for this, but that is beyond the scope of this tutorial. We are now finally done with the accelerometer section.

5.2. Magnetometer data

The magnetometer provides us with a vector that is always pointing to the magnetic North. This vector changes (elevation of the vector changes depending on the altitude) over the surface of the Earth thus we will need a map of the vectors if we want a reference vector that works across the globe. However, in this project, we will be assuming that the change in the North reference vector is negligible since the sensor will not be moving around much (probably only within the room).

The magnetometer actually gives a 3 directional reference vector instead of a simple North heading in 2 dimensions. Due to this, there are some complications that arise. In this project, I will make the assumption that the accelerometer is accurate in providing the reference vector in the vertical plane, and the magnetometer data is accurate in providing the reference vector in the horizontal plane. As a result of this, we do not want the magnetometer data to affect the accelerometer data and vice versa. Therefore, we are going to remove 1 dimension from the magnetometer reference vector.

Firstly, we will calibrate our magnetometer based on my previous post so that we can get a 3 dimensional unit vector from the raw magnetometer data. If you follow the method in the previous post, you should be able to determine the values of A^{-1} and b so you can apply the calibration equation as follows.

$$h = A^{-1}(h_m - b) \quad (5.36)$$

where h is the actual magnetic field and h_m is the measurement reading from the magnetometer with errors. Next, in order to remove 1 dimension (the vertical plane) from the magnetometer vector, we have to transform the coordinates from the body frame (measurements are done in the body frame) to the world frame first (because the vertical

plane that we want to remove exist in the world frame). Therefore, we have to get the rotation matrix that converts the body frame to the world frame from our quaternion. This can be done through equation (1) from my previous post which is shown below as well.

$$r' = Cr \quad (5.37)$$

where

$$C = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \quad (5.38)$$

In this case, r will be the unit vector calibrated magnetometer data in the body frame. Therefore,

$$r = \begin{bmatrix} {}^bm_1 \\ {}^bm_2 \\ {}^bm_3 \end{bmatrix} \quad r' = \begin{bmatrix} {}^wm_1 \\ {}^wm_2 \\ {}^wm_3 \end{bmatrix} \quad (5.39)$$

${}^bm_1, {}^bm_2, {}^bm_3$ is the calibrated magnetometer data in the body frame's x, y, z direction respectively.

${}^wm_1, {}^wm_2, {}^wm_3$ is the calibrated magnetometer data in the world frame's x, y, z direction respectively.

Now, we can safely remove the z -axis from our data by letting ${}^wm_3 = 0$, then re-normalizing the vector such that it stays as a unit vector but only in 2 dimensions now. We will then rotate it back to the body frame and use the resulting vector (in place of the actual calibrated measured data) for our kalman filter update section later on. On a side note, you will find that even without doing all these, the kalman filter algorithm will work. However, whether it works better with or without is another story to be told another day.

In the accelerometer section, we have the gravity vector as our reference vector. On the other hand, we have the North vector as a reference for our magnetometer. For me, since the North vector points exactly in the direction of the negative y -axis when I placed the sensor parallel to my table, I simply took the negative y -axis (in the world frame) as my reference vector. A better approach would be perhaps to have a period of time for the initialization process to set the current North vector as the reference vector. In this way, all we have to do is place the object in the desired direction for it to use the direction as a reference for calculation.

As a result of setting the y -axis to be the reference vector, in place of $g = [0 \ 0 \ 1]_T$, I have $m = [0 \ -1 \ 0]_T$. (in the python code, i added in the negative sign into the gravity reference vector).

Moving on, once again, we need a linear equation for the output of our system in order for us to use the kalman filter. The output that we want to get here is the predicted

accelerometer and magnetometer data from our kalman filter states (quaternion). One way of doing so is through the rotation matrix which can be derived from a quaternion. Similar to the accelerometer, we can use the following equation to convert the magnetometer reading from the world frame to the body frame (so that we can compare it to the actual measured magnetometer value).

$${}^b m_m = {}^b R_w ({}^w m_r) + {}^b e_{mag} + {}^b b_{mag} \quad (5.40)$$

where

${}^b m_m$ is the measured magnetic field in the body frame by the magnetometer ${}^b R_w$ is the rotation matrix for world frame to body frame ${}^w m_r$ is the reference magnetic North vector in the world frame, where in my case ${}^w m_r = [0 \ -1 \ 0]_T$ ${}^b e_{mag}$ is the magnetometer noise in the body frame ${}^b b_{mag}$ is the magnetometer bias in the body frame

The rotation matrix ${}^b R_w$ is actually the same matrix as what was used in the accelerometer. Below is the exact same equation (9).

$${}^b R_w = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \quad (5.41)$$

Since we know that ${}^w m_r = [0 \ -1 \ 0]_T$, we can substitute equation (9) into equation (16) and then simplify equation (16) to get the following equation.

$$\begin{bmatrix} {}^b m_x \\ {}^b m_y \\ {}^b m_z \end{bmatrix} = - \begin{bmatrix} 2(q_1 q_2 + q_0 q_3) \\ q_0^2 - q_1^2 + q_2^2 - q_3^2 \\ 2(q_2 q_3 - q_0 q_1) \end{bmatrix} + \begin{bmatrix} {}^b b_x \\ {}^b b_y \\ {}^b b_z \end{bmatrix} + \begin{bmatrix} {}^b e_x \\ {}^b e_y \\ {}^b e_z \end{bmatrix} \quad (5.42)$$

From here, you can see that in order to obtain the Jacobian matrix for use in the Extended Kalman Filter, the steps are exactly the same as that for the accelerometer. As such, I am going to skip the derivations and just write down the value of the C matrix below here.

$$C = -2 \begin{bmatrix} q_3 & q_2 & q_1 & q_0 \\ q_0 & -q_1 & q_2 & -q_3 \\ -q_1 & -q_0 & q_3 & q_2 \end{bmatrix}_{k-1} \quad (5.43)$$

Here, notice that if we did not simplify the equations with the values of wmr for equation (16), we will get the exact same generalized Jacobian matrix from equation (8) (without the values of the vector g being substituted) as well. I will leave this as homework to you guys but the answer can be found in the python code anyway. I was lazy to write 2 different implementations for the accelerometer and the magnetometer so I combined them both. Anyway, we are now ready to get on with the EKF implementation finally!

5.3. Quaternion EKF Implementation

The equations that we are going to implement are exactly the same as that for the kalman filter as shown below. (This is taken directly from my earlier post except that i added k to the numbering to symbolize that it is the kalman filter equation)

Prediction

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \quad (5.44)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (5.45)$$

where \hat{x}_k^- is the priori estimate of x at time step k , P_k^- is the priori estimate of the error at time step k and Q is the process variance.

Update

$$K_k = \frac{P_k^- C^T}{CP_k^- C^T + R} \quad (5.46)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-) \quad (5.47)$$

$$P_k = (I - K_k C)P_k^- \quad (5.48)$$

where y_k is the actual measured state, \hat{x}_k is the posteri estimate of x at time step k , P_k is the posteri estimate of the error at time step k , K_k is the kalman gain at time step k and R is the measurement variance. Let us go through the equations in order.

5.3.1. Prediction

Equation 1K

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \quad (5.49)$$

If you know the kalman filter well, you would have realized that we have derived this equation already in the earlier part of this post. Below is a copy of the equation (7) from the kalman filter states section above except that I replace k with $k - 1$.

$$\begin{bmatrix} q \\ b^g \end{bmatrix}_k = \begin{bmatrix} I_{4 \times 4} & -\frac{T}{2}S(q) \\ 0_{3 \times 4} & I_{3 \times 3} \end{bmatrix}_{k-1} \begin{bmatrix} q \\ b^g \end{bmatrix}_{k-1} + \begin{bmatrix} \frac{T}{2}S(q) \\ 0_{3 \times 3} \end{bmatrix}_{k-1} w_{k-1} \quad (5.50)$$

From the above equation, we can determine our A and B matrices.

$$A = \begin{bmatrix} I_{4 \times 4} & -\frac{T}{2}S(q) \\ 0_{3 \times 4} & I_{3 \times 3} \end{bmatrix}_{k-1} \quad (5.51)$$

$$B = \begin{bmatrix} \frac{T}{2} S(q) \\ 0_{3 \times 3} \end{bmatrix}_{k-1} \quad (5.52)$$

where

$$S(q) = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \quad (5.53)$$

With all these information, we can now determine \hat{x}_k^- in equation (1K). Once this is done, we have to normalize the quaternion in our kalman filter states dues inaccuracies in the discretized calculation. Next up is equation (2k).

Equation 2K

$$P_k^- = AP_{k-1}A^T + Q \quad (5.54)$$

This equation is pretty straightforward. However, we have to decide an initial value for the P matrix and the Q matrix. You should play around with the values in the code to see how this affects the overall performance of the algorithm. You will realize that if you start with a P matrix with huge values, it may not converge to give you a coherent result. On the other hand, if you start with a P matrix with small values, it may take quite a while for the P matrix to converge to the actual solution. The Q matrix is the process variance, and it represents the inaccuracies of the model that we are using.

In the code, equation (1k) and (2k) are both done within the `predict()` method as shown below. (`predictAccelMag()` will be explained under equation (3k))

5.3.2. Update

Equation 3K

$$K_k = \frac{P_k^- C^T}{C P_k^- C^T + R} \quad (5.55)$$

In order to implement equation (3k), we first have to figure out what is our C matrix here. If you take a look at my previous post explaining the kalman filter using the pendulum example, you will know that the C matrix is a matrix to convert our kalman filter states to the measured variables. In the pendulum example, it just so happens that the measured variables are the same as the kalman filters states thus the C matrix is the identity matrix. However, in our case here, our measured variables are the direction of the gravity and

North vector while our kalman filter states are the quaternion and the gyro bias. They are essentially different parameters thus we have to figure out a C matrix which allows us to convert out state variables into the measured variables.

Now, if you had diligently read through everything in this post, you should realize that we have already derived our C matrix in the accelerometer and magnetometer section. For the Accelerometer,

$$C_a = -2 \begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}_{k-1} \quad (5.56)$$

Notice here that I removed the multiple g because the accelerometer data that I will be using is in units of g . For the Magnetometer,

$$C_m = -2 \begin{bmatrix} q_3 & q_2 & q_1 & q_0 \\ q_0 & -q_1 & q_2 & -q_3 \\ -q_1 & -q_0 & q_3 & q_2 \end{bmatrix}_{k-1} \quad (5.57)$$

In order to determine the C matrix, we would require the quaternion state from the previous iteration. Since the conversion of quaternion to rotation matrix from the world frame to the body frame is the same for the accelerometer and the magnetometer, there is actually a more generalized form of the C equation that will work for both the accelerometer and the magnetometer. I left that as a homework in the above section but the answer is actually already within the code under the method `getJacobianMatrix()`. For the purpose of clarity, I am once again going to write out the exact equation that is implemented within the python code.

$$\hat{y}_k^- = C \hat{x}_k^- \quad (5.58)$$

$$\begin{bmatrix} \hat{a}_m \\ \hat{m}_m \end{bmatrix}_k = \begin{bmatrix} C_a & 0_{3 \times 3} \\ C_m & 0_{3 \times 3} \end{bmatrix} \begin{bmatrix} q \\ b^g \end{bmatrix}_k \quad (5.59)$$

The last term that we have to determine in order to implement equation (3k) is the R matrix. This is the measurement variance and it represents how accurate our measurement data is. Similar to the Q matrix in equation (2k), this term is important yet difficult to determine. The values of these terms are usually determine through simulations or actual tests so in this tutorial, I am just going to use an arbitrary value which does not have any special meaning.

Equation 4K

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C \hat{x}_k^-) \quad (5.60)$$

Next up is equation (4k) which by now we should have all the terms except for y_k . This is the actual measurement values that we obtain from the sensor data. In my python code, this is expressed as shown below.

$$y_k = \begin{bmatrix} a_m \\ m_m \end{bmatrix} \quad (5.61)$$

Since we are making an addition to the quaternion, we have to normalize it again after this step to maintain a unit quaternion during the calculations.

Equation 5K

$$P_k = (I - K_k C) P_k^- \quad (5.62)$$

As for equation (5k), we already have all the variables ready so implementing it will be no problem at all. Equation (3k) to (5k) are all implemented within the `update()` method in the python code as shown below as well.