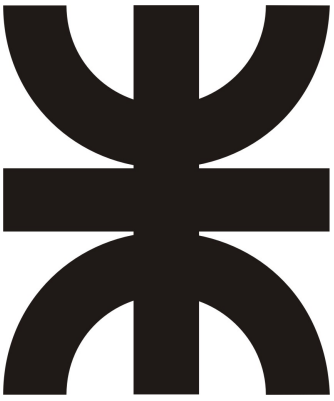
UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL CÓRDOBA

Fundamentos de Robótica móvil

Trabajo Práctico N°5:

Estimación de orientación con filtro de Kalman



Profesores:

- Prof. Dr. Ing Gaydou, David A.
- Prof. Dr. Ing. González Dondo, Diego.
- Prof. Dr. Ing. Perez Paina, Gonzalo F.

Alumnos:

- Dogliani, Matías / Legajo: 72.152
- Galfré, Alejo / Legajo: 71.827
- Nicolodi, Juan / Legajo: 66.875

-2021-

Índice

1. Introducción

En el presente informe se desarrolla el trabajo práctico 5 de Fundamentos de Robótica Móvil, donde se parte de un modelo balancín de un grado de libertad analizado anteriormente.

En un principio se comienza estudiando el proyecto llamado NaveGo que permite mediante simulaciones obtener mediciones reales de una IMU (Unidad de Medida Inercial). En dicho proyecto se analizaron las gráficas obtenidas para dos ejemplos de IMUs ya precargas, como así el funcionamiento de los scripts generadores de dichas mediciones.

Luego, de haber comprendido y analizado el proyecto anterior, se obtienen mediciones reales para una IMU en particular MicroStrain 3DM-GX1. Una vez obtenido estos datos se procede a diseñar un filtro de Kalman para poder estimar el ángulo de orientación del balancín.

2. Simulación con NaveGo

2.1. Ejemplo de integración de sensores con datos sintetizados

2. En este ejemplo se compara el rendimiento del sistema con dos IMUs diferentes utilizando datos sintéticos (simulados). En particular se compara la IMU modelo ADIS16405 y la IMU modelo ADIS16488, cuyos parámetros fueron obtenidos de sus hojas de datos y cargados al simulador. Para ello se realizan numerosas gráficas:

- En la primera, se evalúa el rendimiento del filtro de Kalman. Esto se hace verificando que las "innovaciones" provengan de una distribución de probabilidad normal con media cero y sin correlación entre ellas (proceso estocástico blanco). Luego, se grafica un histograma con las innovaciones, junto con una pdf (probability distribution function) correspondiente a una Gaussiana de referencia.

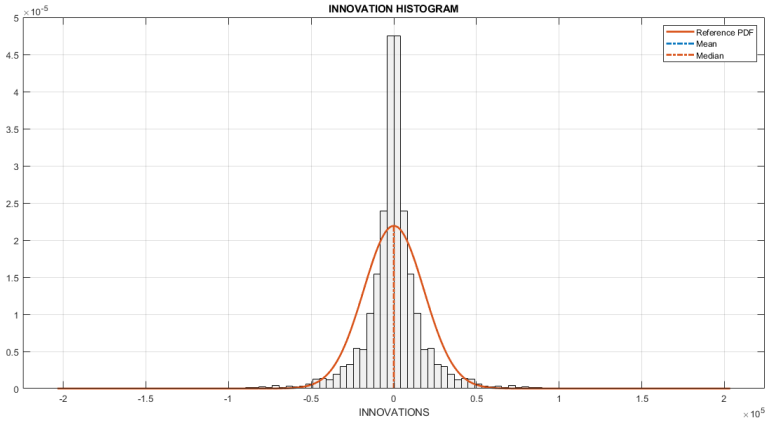


Figura 1: Rendimiento del filtro de Kalman.

- En la segunda se muestra la trayectoria en 3 dimensiones recorrida por el robot. Uno de los ejes corresponde a la altitud expresada en metros, otro a la latitud y el tercero a la longitud, ambos expresados en grados. A su vez, se muestran 3 trazos: Uno con línea discontinua y de color negro, que corresponde a los datos "reales", es decir, los que se programó como trayectoria para la simulación; otro con linea continua de color azul, que corresponde a las "mediciones" obtenidas con la IMU1; y el tercero con linea continua de color naranja, que corresponde a las "mediciones" obtenidas con la IMU2.

Se observa que, para el caso de la latitud y la longitud, todos los trazos coinciden casi a la perfección. En cuanto a la altitud, durante aproximadamente la primera mitad del recorrido, las mediciones obtenidas con la IMU2 son más cercanas a los datos "reales", pero luego, las mediciones obtenidas con la IMU1 prácticamente se igualan a dichos datos, y las de la IMU2 presentan un error por exceso constante.

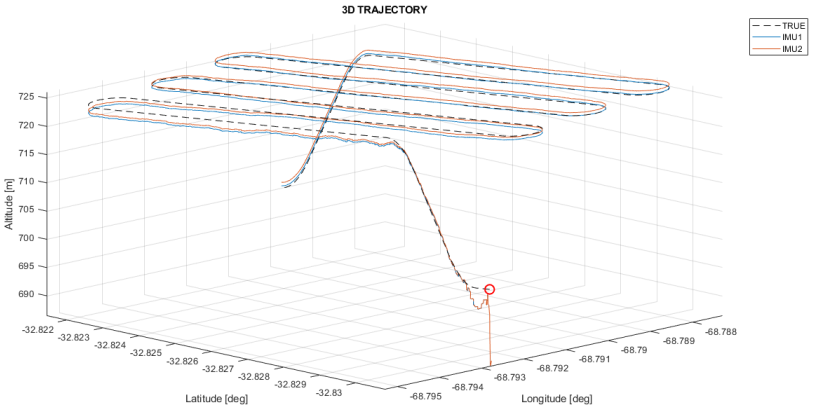


Figura 2: Trayectoria 3D.

- En la tercera, se muestra la misma trayectoria pero en dos dimensiones, considerando únicamente los ejes de latitud y longitud expresados en grados. Al igual que en la gráfica anterior, se muestran tres trazos diferentes, con los mismos colores, tipos de linea, y datos correspondientes. En esta se observa más claramente que el robot recorre un camino en "zig-zag" y se comprueba con mayor facilidad que, como se mencionó, todos los trazos coinciden casi a la perfección. Por su parte, la latitud toma valores comprendidos entre -32.8308° y -32.8215°, mientras que la longitud toma valores comprendidos entre -68.7953° y -68.7873°.

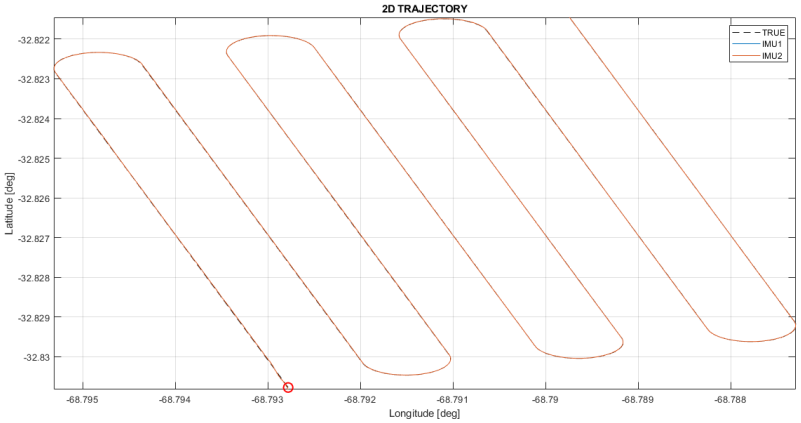


Figura 3: Trayectoria 2D.

- En la cuarta figura, se muestran los ángulos de roll, pitch y yaw en función del tiempo (gráficas de actitud). Nuevamente se muestran los 3 trazos con los mismos colores, tipos de lineas y datos correspondientes que en las gráficas anteriores. Se observa que el angulo de roll varía periódicamente entre -30° y 30°, con un periodo aproximado de 126 segundos. El ancho de los pulsos positivos y negativos es de aproximadamente 9 segundos, y entre ellos el ángulo se mantiene en 0° por aproximadamente 30 segundos.

  Por su parte el angulo de pitch presenta dos pulsos, uno positivo y uno negativo cuyas amplitudes son de 10°. El ancho de los pulsos es de aproximadamente 5 segundos, y entre ellos el ángulo se mantiene en 0° por aproximadamente 407 segundos.

  Por último, el ángulo de yaw varía periódicamente entre -15° y 165°, con un periodo aproximado de 126 segundos. El ancho de los pulsos tanto positivos como negativos es de aproximadamente 54 segundos.

  En los 3 casos se observa que las mediciones obtenidas con la IMU2 son más cercanas al valor "real". Por otro lado, se observa que el tiempo de simulación es de 437.245 segundos.

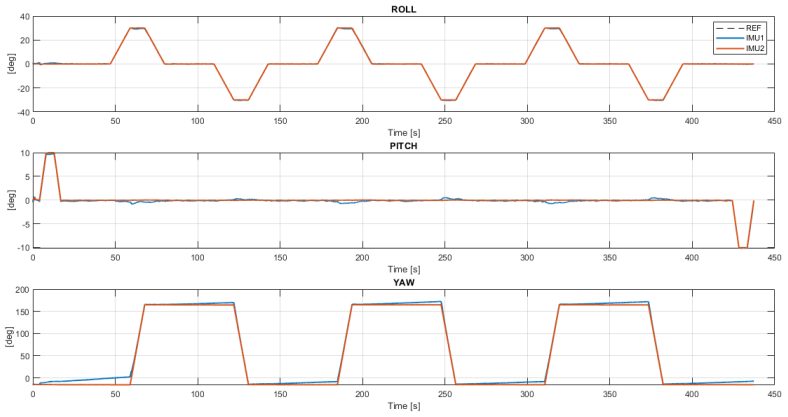


Figura 4: Actitud.

- En la quinta figura se muestra el error de los ángulos de roll, pitch y yaw en función del tiempo. Dicho error se calcula como la diferencia entre el valor "real" y el medido por cada una de las IMUs. Se observa que para los 3 casos, los valores obtenidos con la IMU2 son más cercanos al valor "real" que los obtenidos con la IMU1, puesto que el valor del error es próximo a cero. Esto coincide con lo mencionado en el punto anterior.

  En el caso del ángulo de roll, la IMU1 presenta un error máximo cercano a 1.4° a los 4 segundos de iniciada la simulación. Luego de eso, la magnitud del error permanece por debajo de 1°, y los mayores picos se dan en el lado negativo. En el caso del ángulo de pitch, la IMU1 presenta un error máximo cercano a -0.9° a 1 minuto de iniciada la simulación. Por último, se observa que el mayor error de la IMU1 se produce para el ángulo de yaw, donde alcanza un valor máximo cercano a 18° a los 59 segundos de iniciada la simulación. Además, luego de los primeros 5 segundos, todos los errores son positivos, con un valor mínimo aproximado de 0.6°.

  En estas gráficas se ha agregado además, un tercer trazo con linea discontinua y de color negro que corresponde a 3 desviaciones estándar de la IMU1. Los valores de error de la IMU2 siempre se encuentran dentro de estos límites, mientras que los de la IMU1 no, tal como se mencionó anteriormente.

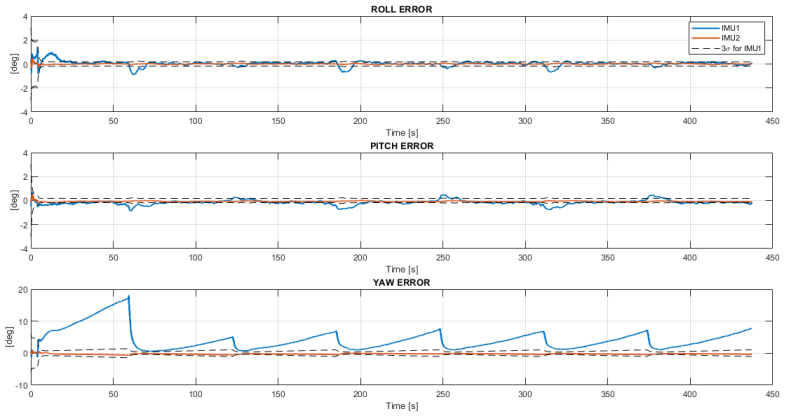


Figura 5: Errores en la actitud.

- En la sexta figura se muestran las velocidades respecto al norte, el este y hacia abajo en función del tiempo. Nuevamente se muestran los 3 trazos con los mismos colores, tipos de lineas y datos correspondientes que en las gráficas anteriores, donde además se incorporan puntos de color gris que corresponden a las mediciones entregadas por un GPS.

  Se observa que la velocidad respecto al norte varía periódicamente entre -15.45 m/s y 15.45 m/s, con un periodo aproximado de 126 segundos. El ancho de los pulsos tanto positivos como negativos es de aproximadamente 57 segundos.

  La velocidad respecto al este varía periódicamente entre -4.14 m/s y 4.14 m/s, con un periodo aproximado de 126 segundos. No obstante, en los bordes de la parte positiva de los pulsos se producen sobrepicos que alcanzan valores de 16 m/s. El ancho de los pulsos tanto positivos como negativos es de aproximadamente 54 segundos.

  Por último, la velocidad hacia abajo presenta dos pulsos, uno negativo y uno positivo cuyas amplitudes son de 2.78 m/s. El ancho de los pulsos es de aproximadamente 5 segundos, y entre ellos la velocidad se mantiene en 0 m/s por aproximadamente 407 segundos. En este caso, se observa que los valores obtenidos con la IMU1 son más cercanos a los valores “reales” que los obtenidos con la IMU2, cuya magnitud siempre es menor. Por otro lado, se observa que el tiempo de simulación es de 437.245 segundos.

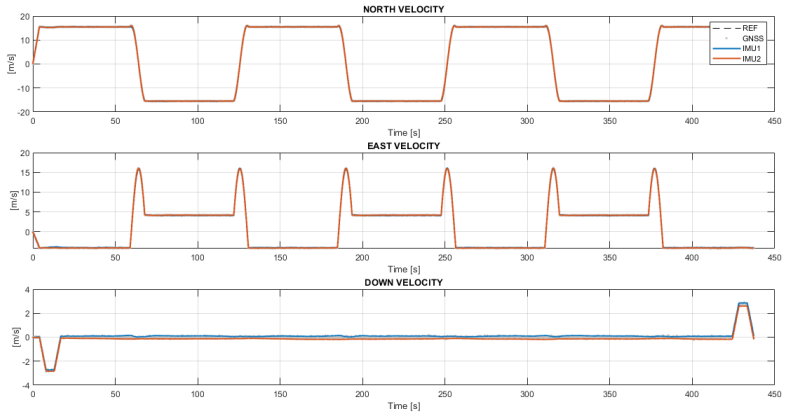


Figura 6: Velocidades.

- En la séptima figura se muestra el error de las velocidades respecto al norte, el este y hacia abajo en función del tiempo. Dicho error se calcula como la diferencia entre el valor "real" y el obtenido por medio de las IMUs y el GPS. Al igual que con los gráficos de error anteriores, se ha agregado un tercer trazo con linea discontinua y de color negro que corresponde a 3 desviaciones estándar de la IMU1.

  Se observa que para las velocidades respecto al norte y al este, los valores obtenidos con la IMU2 son más cercanos al valor "real" que los obtenidos con la IMU1, puesto que el valor del error es menor. No obstante, para la velocidad hacia abajo, ambos dispositivos presentan errores considerables, pero son mejores los resultados obtenidos con la IMU1.

  En el caso de la velocidad respecto al norte, la IMU1 presenta un error máximo cercano a -0.7 m/s a los 60 segundos de iniciada la simulación. Luego de eso, la magnitud del error permanece acotada dentro de los 3 sigmas durante la mayor parte del tiempo, excepto por algunos picos que se producen en el lado negativo pero cuya amplitud no supera los 0.4 m/s. El error correspondiente a la IMU2 permanece siempre dentro de los 3 sigmas.

  En el caso de la velocidad respecto al este, la IMU1 presenta un error máximo cercano a 0.27 m/s a los 15 segundos de iniciada la simulación. Luego de los 80 segundos de simulación, la magnitud del error se mantiene aproximadamente en el límite de los 3 sigmas durante la mayor parte del tiempo. El error correspondiente a la IMU2 permanece siempre dentro de los 3 sigmas.

  Por último, se observa que para el caso de la velocidad hacia abajo, los valores de error de las IMUs no oscilan alrededor de 0, sino que el correspondiente a la IMU1 es siempre positivo y con un valor aproximado de 0.1 m/s, mientras

que para la IMU2 es siempre negativo con un valor aproximado de -0.15 m/s . El valor máximo de error para la IMU1 es de aproximadamente 0.16 m/s y se produce a los 185 segundos, mientras que el valor máximo de error para la IMU2 es de aproximadamente -0.19 m/s y se produce a los 185 segundos. En este caso ambos errores se encuentran por fuera de los 3 sigmas prácticamente durante todo el tiempo.

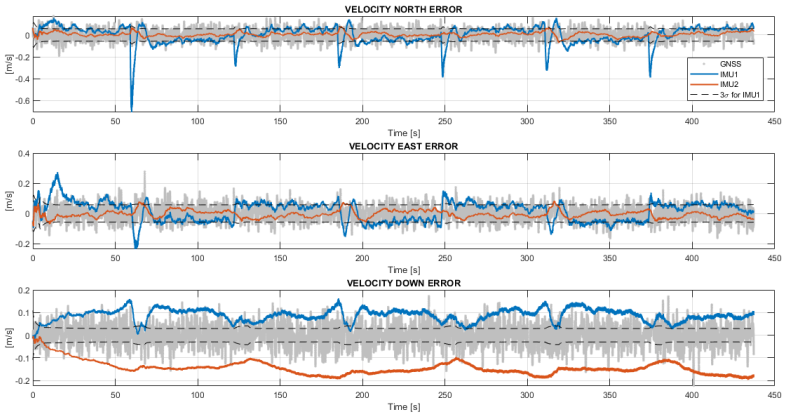


Figura 7: Errores en las velocidades.

- En la octava figura se muestra la latitud y la longitud expresadas en grados, y la altitud expresada en metros, todas en función del tiempo (gráficas de posición). Nuevamente se muestran los 3 trazos con los mismos colores, tipos de lineas y datos correspondientes que en las gráficas anteriores, donde además se incorporan puntos de color gris que corresponden a las mediciones entregadas por un GPS.

  Se observa que la latitud tiene la forma de una función triangular montada sobre una recta con pendiente positiva. Su valor oscila entre aproximadamente -32.831° y -32.821°, y tiene un periodo de aproximadamente de 126 segundos. La longitud también tiene la forma de estar montada sobre una recta con pendiente positiva, mayor a la del caso anterior. Su valor oscila entre aproximadamente -68.795° y -68.787°, y también tiene un periodo de aproximadamente de 126 segundos. Por su parte la altitud comienza con un valor de 700 m, y luego de aproximadamente 5 segundos comienza a crecer y alcanza los 725 m a los 17 segundos. Luego permanece en este valor hasta los 425 segundos, donde comienza a decrecer nuevamente hasta los 700 m.

  En el caso de la latitud y la longitud, las mediciones obtenidas con las dos IMUs son aproximadamente iguales entre sí y con el valor “real”. Para el caso de la altitud, hasta los 190 segundos aproximadamente, los valores más próximos a los “reales” son los obtenidos por la IMU2, y luego de ese momento,

son más cercanos los correspondientes a la IMU1. Por otro lado, se observa que el tiempo de simulación es de 437.245 segundos.

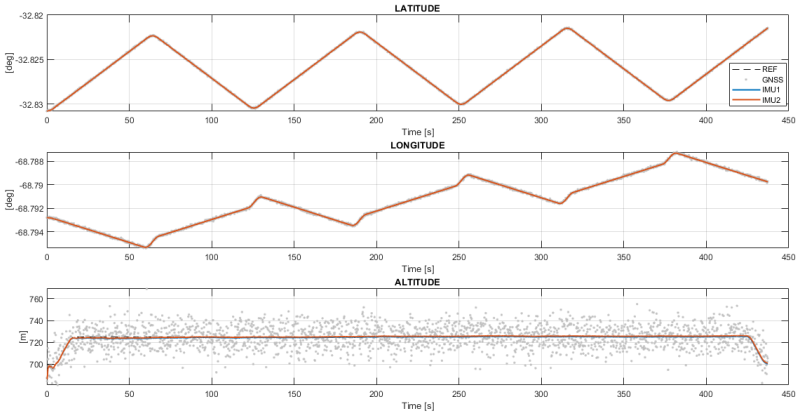


Figura 8: Posición.

- En la novena figura se muestra el error de la latitud, longitud y altitud en función del tiempo. Dicho error se calcula como la diferencia entre el valor "real" y el obtenido por medio de las IMUs y el GPS. Al igual que con los gráficos de error anteriores, se ha agregado un tercer trazo con linea discontinua y de color negro que corresponde a 3 desviaciones estándar de la IMU1.

  En el caso de la latitud, la magnitud del error es menor a 0.5° prácticamente durante todo el tiempo de simulación para ambas IMUs, y la diferencia entre las mismas llega como máximo a aproximadamente 0.06°. En el caso de la longitud, la magnitud del error es menor a 0.4° prácticamente durante todo el tiempo de simulación para ambas IMUs, y la diferencia entre las mismas llega como máximo a aproximadamente 0.015°. Por último, en el caso de la altitud, la magnitud del error es menor a 1 m prácticamente durante todo el tiempo de simulación para ambas IMUs. No obstante, el error correspondiente a la IMU2 está mas cerca de cero durante aproximadamente los primeros 190 segundos de simulación. Luego el error correspondiente a la IMU1 es el más cercano a cero.

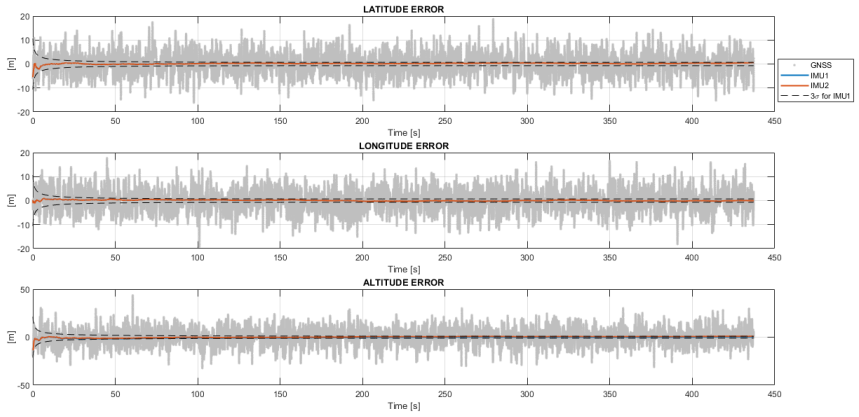


Figura 9: Errores en la posición.

- En la décima figura, se muestra la estimación por medio del filtro de Kalman del bias de los giróscopos en X, Y y Z. Nuevamente se muestran los 2 trazos con los mismos colores y tipos de lineas correspondientes a las IMUs y el trazo con linea negra discontinua que corresponde a 3 desviaciones estándar de la IMU1.

  En el caso del giróscopo en X, se observa que el bias correspondiente a la IMU1 tiene un valor máximo de aproximadamente 0.12° y un valor medio aproximado de 0.102°. Por su parte el bias correspondiente a la IMU2 tiene un valor máximo de aproximadamente -0.006° y un valor medio aproximado de -0.005°.

  En el caso del giróscopo en Y, se observa que el bias correspondiente a la IMU1 tiene un valor máximo de aproximadamente -0.073° y un valor medio aproximado de -0.062°. Por su parte el bias correspondiente a la IMU2 tiene un valor máximo de aproximadamente -0.009° y un valor medio aproximado de -0.007°.

  En el caso del giróscopo en Z, se observa que el bias correspondiente a la IMU1 tiene un valor máximo de aproximadamente 0.115° y un valor medio aproximado de 0.075°. Por su parte el bias correspondiente a la IMU2 tiene un valor máximo de aproximadamente -0.005° y un valor medio aproximado de -0.003°.

  En los 3 casos, la estimación del bias para la IMU2 se mantiene dentro del límite de los 3 sigmas, pero los valores para la IMU1 no.

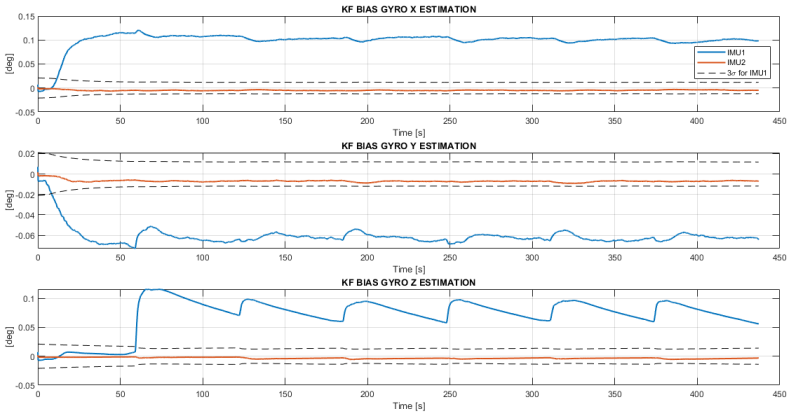


Figura 10: Estimación del bias de los giróscopos.

- En la undécima figura, se muestra la estimación por medio del filtro de Kalman del bias de los acelerómetros en X, Y y Z. Nuevamente se muestran los 2 trazos con los mismos colores y tipos de lineas correspondientes a las IMUs y el trazo con linea negra discontinua que corresponde a 3 desviaciones estándar de la IMU1.

  En el caso del acelerómetro en X, se observa que el bias correspondiente a la IMU1 tiene un valor máximo de aproximadamente -0.00376° y un valor medio aproximado de -0.0016°. Por su parte el bias correspondiente a la IMU2 tiene un valor máximo de aproximadamente -0.00198° y un valor medio aproximado de 0.0009°.

  En el caso del acelerómetro en Y, se observa que el bias correspondiente a la IMU1 tiene un valor máximo de aproximadamente -0.01088° y un valor medio aproximado de -0.006°. Por su parte el bias correspondiente a la IMU2 tiene un valor máximo de aproximadamente 0.00706° y un valor medio aproximado de 0.002°.

  En el caso del acelerómetro en Z, se observa que el bias correspondiente a la IMU1 tiene un valor máximo de aproximadamente -0.0626° y el bias correspondiente a la IMU2 tiene un valor máximo de aproximadamente 0.0512°.

  En el caso del acelerómetro en X, la estimación del bias para ambas IMUs se mantiene dentro del límite de los 3 sigmas, pero para los otros dos acelerómetros no.

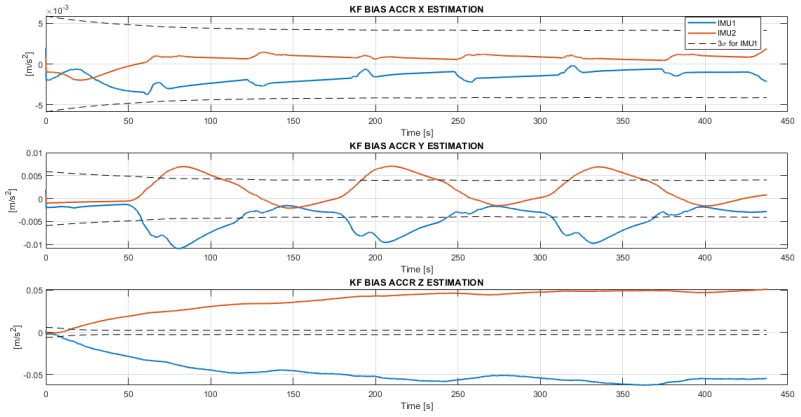


Figura 11: Estimación del bias de los acelerómetros.

2.2. Análisis de los scripts generadores de errores

3. La función `gyro_gen()` se utiliza para generar mediciones "realistas" de giróscopos a partir de datos de referencia y del perfil de error de la IMU. Esta recibe dos entradas:

- ref: Estructura de datos con la trayectoria verdadera (el valor real).
- imu: Estructura de datos con el perfil de error de la IMU.

La salida de esta función es una matriz de dimensiones Nx3 con mediciones simuladas de giróscopos ubicados en los ejes X, Y y Z en el marco de referencia del cuerpo y medidos en radianes.

En primer lugar, el script obtiene el tamaño del vector de tiempo, el cual es un miembro de la estructura ref, y lo almacena en la variable N. Luego, define una variable M, que es un vector fila con dos elementos: el primero es el valor de N que acabamos de obtener, y el segundo es 3. Seguidamente, comprueba que la estructura ref contenga un miembro llamado "wb":

- Si lo posee, significa que las tasas de giro están definidas y almacena el valor de este miembro en una variable llamada `gyro_b`.
- Caso contrario, debe obtenerlas desde las matrices de cosenos directores. Para ello, utiliza la función `gyro_gen_delta()`, a la cual se le pasan dos parámetros:
  - `DCMnb_m`, la cual es una matriz de dimensiones Nx9, donde cada una de las N filas contiene los 9 elementos de las matrices con los cosenos directores ordenadas como [a11 a21 a31 a12 a22 a32 a13 a23 a33].
  - t, el cual es un vector columna que contiene el tiempo en segundos.

Ambos parámetros son miembros de la estructura `ref`. Esta función devuelve una matriz de dimensiones Mx3 (donde M es el tamaño del vector de tiempo), que contiene los ángulos delta de los giróscopos X, Y y Z expresados en radianes. Esta matriz se almacena en una variable llamada `gyro_raw` a la cual luego se le agrega una fila al principio con tres ceros.

Puesto que en esta función se calculan las derivadas y esto introduce ruido, se deben filtrar los resultados obtenidos. Para ello se utiliza otra función llamada `my_sgolayfilt()`, a la cual se le pasa la matriz calculada anteriormente (`gyro_raw`). Esta aplica un filtrado Savitzky-Golay con tamaño variable y el resultado se almacena en una variable llamada `gyro_b`.

Luego, se declara una matriz de dimensión M (la cual fue definida anteriormente) rellena con ceros, y se almacena en una variable llamada `g_err_b`. Seguidamente, el script entra en un bucle que repetirá N veces (donde N es la cantidad de elementos del vector de tiempo). En cada iteración de este bucle primero se toma una de las filas del miembro `DCMnb_m` de la estructura `ref`, la cual, como se explicó anteriormente, contiene los 9 elementos correspondientes a una matriz de cosenos directores, y se la redimensiona para formar una matriz de 3x3 la cual se almacena en una variable denominada `dcmnb`.

Seguidamente se llama a una función denominada `earth_rate()`. Esta recibe como entrada la latitud en radianes (la cual es un miembro de la estructura `ref)` y devuelve la velocidad de rotación de la tierra en el marco de referencia de navegación y en rad/seg, la cual se almacena en una variable denominada `omega_ie_n`.

Después, se llama a una función denominada `transport_rate()` la cual recibe como entrada la latitud en radianes, la velocidad respecto del norte en m/s, la velocidad respecto del este en m/s, y la altura en m. Todos estos parámetros son miembros de la estructura `ref`. Esta función devuelve la velocidad de traslación de la tierra en el marco de referencia de navegación y en rad/s, la cual se almacena en una variable denominada `omega_en_n` .

A continuación, se realiza el producto matricial entre la matriz de 3x3 con los cosenos directores (dcmnb) y la suma entre las velocidades de rotación y de traslación de la tierra (`omega_ie_n` y `omega_en_n`). Finalmente, se traspone este resultado y se almacena en cada fila de la variable `g_err_b` definida antes de entrar al bucle.

A partir de este punto en el script, se comienzan a calcular los diversos ruidos que afectan las mediciones. En primer lugar se obtiene el bias estático como una variable aleatoria constante. Esto se logra utilizando una función denominada `noise_b_sta()`. La misma recibe dos parámetros como entrada: un escalar `b_sta` para definir los límites entre los que se encontrará el valor generado, y M que indica el número de filas de la matriz de salida. Esta matriz es de dimensión Nx3 y en cada una de sus N filas contiene los valores de los errores de bias estáticos en X, Y y Z, los cuales están formados a su vez por una parte determinística y una parte estocástica.

El primer parámetro enviado es un miembro de la estructura imu, y para el segundo se utiliza la longitud del vector de tiempo N. La salida de la función se almacena en una variable denominada `gb_sta`.

Seguidamente se simula el ruido blanco. Para ello en primer lugar se genera una matriz de dimensión M (definida anteriormente) con números aleatorios cuya distribución de probabilidad es de tipo normal o Gaussiana y se la almacena en una variable llamada `wn`. Luego, se define otra matriz de dimensión M, pero rellena con ceros, la cual se almacena en la variable `g_wn`. Finalmente, se entra en un bucle de 3 iteraciones en el cual se toma una columna de la matriz con números aleatorios (wn) y se la multiplica elemento por elemento con el miembro `a_std` de la estructura imu. El vector columna resultante se almacena en la columna correspondiente de la matriz `g_wn`.

Después, se simula el bias dinámico (o la inestabilidad del bias) como un modelo Gauss-Markov de primer orden. Para ello se define un diferencial de tiempo dt, al cual se asigna el recíproco del valor almacenado en el miembro freq de la estructura ref, y luego se llama a la función `noise_b_dyn()`. Esta recibe 4 parámetros de entrada: un vector fila de 3 elementos `b_corr`, el cual contiene los tiempos de correlación; un vector fila de 3 elementos `b_dyn` el cual contiene el nivel de los bias dinámicos; el periodo de muestreo dt y un vector fila de 2 elementos M, el cual contiene la dimensión de la matriz de salida. Dicha matriz contiene los bias dinámicos generados en el eje X, Y y Z. Los primeros 2 parámetros que se envían a la función son miembros de la estructura imu, y la salida de la función se almacena en una variable denominada `gb_dyn`.

Por último, se simula la "caminata aleatoria" de la velocidad. Para ello se utiliza la función `noise_rrw()`. Esta recibe 3 parámetros como entrada: un vector fila de 3 elementos rrw con el nivel de la caminata aleatoria, el periodo de muestreo dt (escalar), y un vector fila de 2 elementos M que determina la dimensión de la matriz de salida. Dicha matriz contiene el ruido generado en X, Y y Z, expresado en rad/s^2. El primer parámetro que se envía a esta función es un miembro de la estructura imu y la matriz devuelta se almacena en una variable denominada `g_rrw`.

Finalmente, para obtener el valor que simula una medición "realista" se suma al valor verdadero que mediría un giróscopo ideal (`gyro_b`), el ruido debido a la rotación y traslación de la tierra (`g_err_b`), el ruido blanco (`g_wn`), el valor de bias (`gb_sta`), la variación del bias (`gb_dyn`) y la caminata aleatoria de la velocidad (`g_rrw`). Este valor se almacena en una variable denominada `wb_sim`.

Por su parte, la función *acc_gen()* se utiliza para generar mediciones "realistas" de acelerómetros a partir de datos de referencia y del perfil de error de la IMU. Esta recibe dos entradas:

- ref: Estructura de datos con la trayectoria verdadera (el valor real).

- imu: Estructura de datos con el perfil de error de la IMU.

La salida de esta función es una matriz de dimensiones Nx3 con aceleraciones simuladas en los ejes X, Y y Z en el marco de referencia del cuerpo y medidas en m/s^2. Al igual que la función anterior, en primer lugar obtiene el tamaño del vector de tiempo, el cual es un miembro de la estructura ref, y lo almacena en la variable N. Luego, define una variable M, que es un vector fila con dos elementos: el primero es el valor de N que acabamos de obtener, y el segundo es 3.

Después, verifica si la estructura ref posee un miembro llamado "fb":

- Si este miembro existe significa que contamos con las aceleraciones, por lo que almacena su valor en una variable denominada `acc_b.`
- En caso contrario, se deben calcular dichas aceleraciones. En primer lugar se trata de hacerlo a partir de las velocidades. Para ello se verifica si la estructura ref posee un miembro denominado "vel". En caso de que este miembro exista, se calcula la derivada del mismo, y se divide cada uno de sus elementos por el vector de tiempo concatenado 3 veces. Este resultado se almacena en una variable denominada `acc_raw,` a la cual luego se le agrega una primera fila con tres ceros.

  Al igual que con la función anterior, al calcular las derivadas se produce ruido, el cual es necesario filtrar. Para ello se utiliza la función `my_sgolayfilt` mencionada anteriormente, y los resultados ya filtrados se almacenan en una variable denominada `acc_ned`. Por último, se llama a la función `acc_nav2body()` la cual transforma las aceleraciones desde el marco de referencia de navegación hasta el marco de referencia del cuerpo.

  Esta recibe dos entradas: una matriz con dimensiones Nx3 denominada `acc_n,` la cual contiene en cada una de sus filas las aceleraciones respecto al norte, este y hacia abajo en el marco de referencia de navegación; y una matriz de dimensiones Nx9 denominada `DCMnb_m`, la cual, como se mencionó anteriormente, contiene en cada una de sus N filas los 9 elementos de una matriz de cosenos directores entre el marco de referencia de navegación y del cuerpo. La salida de esta función es una matriz de dimensiones Nx3 denominada fb, la cual contiene en cada una de sus N filas las aceleraciones simuladas en X, Y y Z en el marco de referencia del cuerpo.

  A esta función se le envía como primer parámetro los resultados filtrados obtenidos anteriormente (`acc_ned`), y como segundo parámetro la matriz de cosenos directores la cual es un miembro de la estructura ref. El resultado se almacena en la variable `acc_b` mencionada anteriormente (la misma en la que hubiésemos guardado las aceleraciones en caso de estar definidas en la estructura).

- En el caso de que las velocidades tampoco estén definidas (es decir que el miembro vel no exista), se procede a calcular las aceleraciones a través de la posición. Para ello se utiliza la función `pllh2vned()` la cual genera tanto las velocidades como las aceleraciones en el marco de referencia de navegación a partir de la posición. Esta recibe como entrada la estructura ref, la cual contiene los datos de la trayectoria verdadera. Las salidas de esta función son 2 matrices de dimensiones Nx3 denominadas `vel_ned` y `acc_ned`, las cuales contienen en cada una de sus N filas las velocidades y las aceleraciones respecto al norte, este y hacia abajo en el marco de referencia de navegación respectivamente. Las primeras se expresan en m/s y las segundas en m/s^2.

  Como solo nos interesan las aceleraciones, guardamos únicamente la segunda matriz devuelta por la función en la variable denominada `acc_ned` (la cual contiene las aceleraciones en el marco de referencia de navegación). Por último, al igual que en el caso de calculo a través de las velocidades, se llama a la función `acc_nav2body()` la cual transforma las aceleraciones desde el marco de referencia de navegación hasta el marco de referencia del cuerpo. A esta se le envía como primer parámetro los resultados obtenidos anteriormente (`acc_ned`), y como segundo parámetro la matriz de cosenos directores la cual es un miembro de la estructura `ref`. El resultado se almacena en la variable `acc_b` mencionada anteriormente.

Obtenidos los valores "verdaderos" de las aceleraciones, es decir, aquellos que se medirían utilizando un acelerómetro ideal, se procede a afectarlos por los diversas fuentes de error. En primer lugar se calcula el efecto de la gravedad, la cual es una aceleración que puede considerarse constante y que esta dirigida verticalmente hacia abajo bajo ciertas condiciones. Para ello se utiliza la función `gravity()`. Esta recibe dos parámetros como entradas: Un vector columna de N elementos lat, el cual contiene las latitudes en radianes; y un vector columna de N elementos h, el cual contiene las alturas o altitudes en metros. Esta función devuelve un vector fila de N elementos, el cual contiene las aceleraciones debidas a la gravedad en el marco de referencia de navegación. Los parámetros enviados a esta función son miembros de la estructura `ref`, y el resultado que devuelve la misma es afectado por un signo negativo y almacenado en una variable denominada `grav_n`.

Seguidamente, se calcula el efecto coriolis. Para ello se utiliza la función `coriolis()` la cual recibe tres parámetros como entradas: Un vector columna de N elementos lat, el cual contiene las latitudes en radianes; una matriz de dimensiones Nx3 denominada vel, la cual contiene en cada una de sus N filas las velocidades respecto al norte, este y hacia abajo expresadas en m/s; y un vector columna de N elementos h, el cual contiene las alturas o altitudes en metros. Esta función devuelve una matriz de dimensiones Nx3, la cual contiene en cada una de sus N filas las aceleraciones de Coriolis en el marco de referencia de navegación y expresadas en m/s^2. Los parámetros enviados a esta función son miembros de la estructura `ref`, y el resultado que devuelve la misma es almacenado en una variable denominada `cor_n`.

Puesto que estas componentes se encuentran en el marco de referencia de navegación, es necesario llevarlas al marco de referencia del cuerpo. Para ello se definen en primer lugar, 2 matrices de dimensión M rellenas con ceros, las cuales se almacenan en las variables `grav_b` y `cor_b`. Luego, se entra en un bucle de N iteraciones (donde N es la longitud del vector de tiempo). En cada iteración de este bucle primero se toma una de las filas del miembro `DCMnb_m` de la estructura `ref`, la cual, como se explicó anteriormente, contiene los 9 elementos correspondientes a una matriz de cosenos directores, y se la redimensiona para formar una matriz de 3x3 la cual se almacena en una variable denominada `dcm_nb`.

Luego, se toma una de las filas de `grav_n`, se la traspone para obtener un vector columna y se realiza el producto matricial entre la matriz 3x3 definida anteriormente (`dcm_nb`) y dicho vector traspuesto, cuyo resultado se almacena en una variable denominada gb. De manera similar, se toma una de las filas de `cor_n`, se la traspone para obtener un vector columna y se realiza el producto matricial entre la matriz 3x3 definida anteriormente (`dcm_nb`) y dicho vector traspuesto, cuyo resultado se almacena en una variable denominada `corb`. Finalmente, estos resultados que están en forma de vectores columna vuelven a trasponerse para obtener vectores filas y se almacenan en cada una de las filas de las variables `grav_b` y `cor_b` definidas anteriormente.

A continuación, se calculan los diversos ruidos. En primer lugar se obtiene el bias estático como una variable aleatoria constante. Esto se logra utilizando una función denominada `noise_b_sta()`. La misma recibe dos parámetros como entrada: un escalar `b_sta` para definir los límites entre los que se encontrará el valor generado, y M que indica el número de filas de la matriz de salida. Esta matriz es de dimensión Nx3 y en cada una de sus N filas contiene los valores de los errores de bias estáticos en X, Y y Z, los cuales están formados a su vez por una parte determinística y una parte estocástica. El primer parámetro enviado es un miembro de la estructura imu, y para el segundo se utiliza la longitud del vector de tiempo N. La salida de la función se almacena en una variable denominada `ab_sta`.

Seguidamente se simula el ruido blanco. Para ello en primer lugar se genera una matriz de dimensión M (definida anteriormente) con números aleatorios cuya distribución de probabilidad es de tipo normal o Gaussiana y se la almacena en una variable llamada `wn`. Luego, se define otra matriz de dimensión M, pero rellena con ceros, la cual se almacena en la variable `a_wn`. Finalmente, se entra en un bucle de 3 iteraciones en el cual se toma una columna de la matriz con números aleatorios (wn) y se la multiplica elemento por elemento con el miembro `a_std` de la estructura imu. El vector columna resultante se almacena en la columna correspondiente de la matriz `a_wn`.

Después, se simula el bias dinámico (o la inestabilidad del bias) como un modelo Gauss-Markov de primer orden. Para ello se define un diferencial de tiempo dt, al cual se asigna el recíproco del valor almacenado en el miembro freq de la estructura `ref`,

y luego se llama a la función `noise_b_dyn()`. Esta recibe 4 parámetros de entrada: un vector fila de 3 elementos `b_corr`, el cual contiene los tiempos de correlación; un vector fila de 3 elementos `b_dyn` el cual contiene el nivel de los bias dinámicos; el periodo de muestreo dt y un vector fila de 2 elementos M, el cual contiene la dimensión de la matriz de salida. Dicha matriz contiene los bias dinámicos generados en el eje X, Y y Z. Los primeros 2 parámetros que se envían a la función son miembros de la estructura imu, y la salida de la función se almacena en una variable denominada `ab_dyn`.

Por último, se simula la "caminata aleatoria" de la velocidad. Para ello se utiliza la función `noise_rrw()`. Esta recibe 3 parámetros como entrada: un vector fila de 3 elementos rrw con el nivel de la caminata aleatoria, el periodo de muestreo dt (escalar), y un vector fila de 2 elementos M que determina la dimensión de la matriz de salida. Dicha matriz contiene el ruido generado en X, Y y Z, expresado en rad/s^2. El primer parámetro que se envía a esta función es un miembro de la estructura imu y la matriz devuelta se almacena en una variable denominada `a_rrw`.

Finalmente, para obtener el valor que simula una medición "realista", al valor verdadero que mediría un acelerómetro ideal (`acc_b`), se le resta la aceleración debida al efecto coriolis (`cor_b`), y se le suma la aceleración debida a la gravedad (`grav_b`), el ruido blanco (`a_wn`), el valor de bias (`ab_sta`), la variación del bias (`ab_dyn`) y la caminata aleatoria de la velocidad (`a_rrw`). Este valor se almacena en una variable denominada `fb_sim`. Nótese que el procedimiento para generar los ruidos es idéntico que en el caso de los giróscopos.

3. Simulación con sensor MicroStrain 3DM-GX1

Habiendo analizado los scripts que generan ruidos para simular mediciones realistas de sensores, se procede a aplicar los mismos métodos al modelo del balancín realizado en el trabajo anterior. En particular, se simula el sistema como si este utilizara la unidad de medición inercial modelo 3DM-GX1 del fabricante MicroStrain, para obtener medidas de la velocidad angular y las utilizara como realimentación para formar un sistema a lazo cerrado.

Para ello, en primer lugar, se genera un perfil de error con los datos de la IMU. Estos se obtienen de la hoja de datos del dispositivo brindada por el fabricante[1], y se cargan en una estructura de datos de GNU OCtave denominada DMGX1. Específicamente, se cargan la caminata aleatoria de los ángulos, la caminata aleatoria de las velocidades, y los valores de bias, tanto estáticos como dinámicos, de los giróscopos y acelerómetros que posee el dispositivo.

[1] http://files.microstrain.com/3DM-GX1%20Detailed%20Specs%20-%20Rev%201%20-%20070723.pdf

Luego, se aplica una serie de conversiones para llevar estos datos brindados por el fabricante a unidades del Sistema Internacional. Para ello se utiliza una función denominada `imu_si_errors()`, la cual recibe la estructura de datos que contiene los errores de la IMU y forma parte del toolbox NaveGo.

Seguidamente, se procede a generar cada uno de los ruidos que afectan las mediciones realizadas por un giróscopo. Se considera únicamente uno de los giróscopos que posee la unidad de medición (en particular el correspondiente al eje X), puesto que el movimiento del balancín está restringido a la rotación en una dirección. Por esta misma razón no se utiliza ningún acelerómetro tampoco.

Se genera ruido de bias estático, mediante la función `noise_b_sta()`; ruido blanco, mediante la función `randn()` y la desviación estándar del giróscopo; ruido de bias dinamico (o inestabilidad de bias) mediante la función `noise_b_dyn()`; y caminata aleatoria de la velocidad mediante la función `noise_rrw()`. El funcionamiento de cada una de ellas, así como sus parámetros de entrada y de salida, fueron analizados en el apartado anterior.

Acto seguido, se procede a sumar todas estas fuentes de errores para formar un "error total". Como se mencionó anteriormente, solo se considera el error del giróscopo correspondiente al eje X. Finalmente, cada uno de los elementos de este vector de error se suma a la velocidad angular (elemento 2 del vector de estados q, es decir q(2,)) que corresponde al valor "verdadero", dentro del bucle `for` que se utiliza para calcular la evolución temporal de los estados.

Las gráfica de posición y velocidad angular, así como del torque de control se muestra en la figura 12. El script utilizado para generar dichos gráficos se adjunta en el anexo.

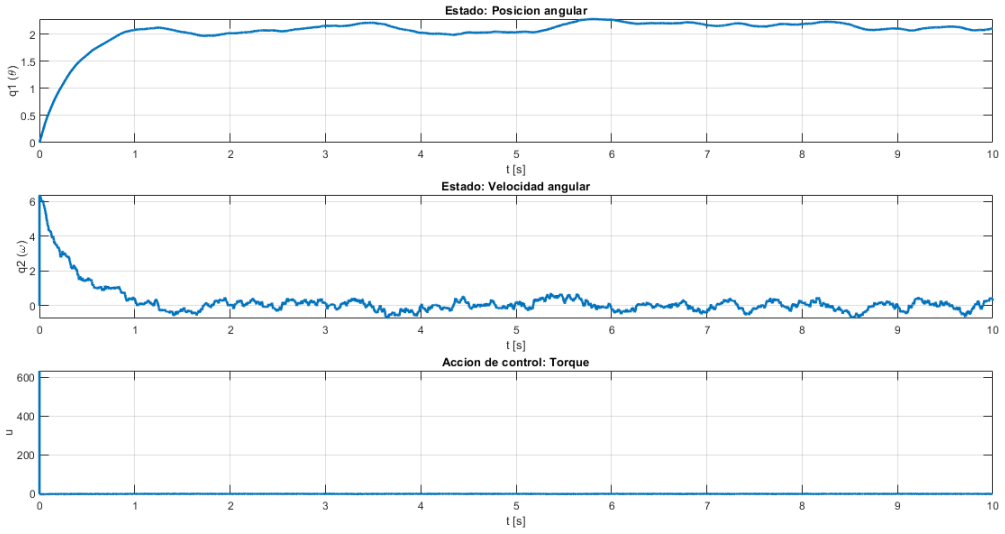


Figura 12: Resultados de la simulación del balancín.

4. Modelo de estimación

Teniendo en cuenta el modelo de estimación de la actitud de una aeronave presentado en [1] y el modelo del sistema del balancín desarrollado en trabajos anteriores, se diseña un filtro de Kalman definido por las expresiones 1 y 2

$$\mathbf{X_{k+1}} = \mathbf{F}x_k + \mathbf{G}u_k + \mathbf{L}w_k \tag{1}$$
$$z_k = \mathbf{H}x_k + v_k \tag{2}$$

Donde x_k el vector de estados del sistema, u es el vector de entrada al sistema y z es el vector de mediciones:

$$x_k = \begin{bmatrix}\Theta & b_\omega\end{bmatrix}^T \tag{3}$$
$$\omega_k = \begin{bmatrix}\eta_w & \eta_\alpha\end{bmatrix}^T \tag{4}$$

Estas expresiones de Filtro de Kalman son utilizadas para estimar el ángulo Θ y el bias de un giroscopio b_w, considerando $u = \omega_M$ y $z = \Theta_m$

La matriz **F** es la matriz que define la transición de un estado a otro, es decir, la evolución del sistema. La matriz **G** es la ganancia del vector de entrada a la planta y la matriz **H** relaciona el estado actual del sistema con la medición, que corresponde al modelo del sistema. La matriz **L** es la ganancia del ruido. Las matrices quedan definidas por las expresiones 5

$$F = \begin{bmatrix}1 & -dt\\ 0 & 1\end{bmatrix} \;;\; G = \begin{bmatrix}dt\\ 1\end{bmatrix} \;;\; L = \begin{bmatrix}dt\end{bmatrix} H = \begin{bmatrix}1 & 0\end{bmatrix} \tag{5}$$

5. Ruido de proceso(Q) y medición(R)

Para la determinación de la matrices de ruidos de proceso y medición se tomo como referencias la mediciones realizadas en [2]. En el enfoque propuesto se obtiene σ_θ y σ_ω del análisis de conjuntos de datos experimentales, y σ_α se ajusta experimentalmente según el comportamiento deseado del filtro.

Tomando como referencia estos valores se tiene:

$$\sigma_\theta = 19{,}97$$
$$\sigma_\omega = 2{,}96$$
$$\sigma_\alpha = 0{,}999$$

A partir de estos datos se puede obtener la matriz de ruido de proceso Q. Se estableció un tiempo de muestro $ts = 0{,}01312$.

$$Q = t_s \begin{bmatrix} \sigma_\omega^2 & \sigma_{\omega\alpha} \\ \sigma_{\omega\alpha} & \sigma_\alpha^2 \end{bmatrix} = \begin{bmatrix} 1{,}505 \times 10^{-3} & 0 \\ 0 & 1{,}721 \times 10^{-4} \end{bmatrix}$$

Para el calculo de R es mas fácil, ya que es un escalar

$$R = \sigma_\theta^2 = 398{,}62$$

6. Implementación de Filtro Kalman

Una vez ya diseñado el filtro de Kalman para nuestro sistema se puede implementarlo para realizar la predicción y la corrección de la medición. La **predicción** se implementa mediante las expresiones 6 y 7. El error de la predicción será la diferencia entre el valor real y el valor estimado.

$$\hat{\mathbf{x}}_\mathbf{k}^- = \mathbf{F}\hat{\mathbf{x}}_{\mathbf{k-1}} + \mathbf{G}u_{k-1} \tag{6}$$
$$\mathbf{P}_\mathbf{k}^- = \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{Q}_{k-} \tag{7}$$

La **corrección** se implementa mediante las ecuaciones 8,

$$K_k = \mathbf{P}_\mathbf{k}^-\mathbf{H}^T \cdot (\mathbf{H}\mathbf{P}_\mathbf{k}^-\mathbf{H}^T + \mathbf{R_k}^{-1} \tag{8}$$
$$\hat{\mathbf{x}}_\mathbf{k} = \hat{\mathbf{x}}_\mathbf{k}^- + K_k \cdot (z_k - \mathbf{H}\hat{\mathbf{x}}_\mathbf{k}^-) \tag{9}$$
$$\mathbf{P}_k = (\mathbf{I} - K_k\mathbf{H})P_k^- \tag{10}$$

Donde K_k : ganancia de Kalman.

Conociendo estas expresiones y ya habiendo obtenidos los valores necesarios, se implementa este modelo en GNU/Octave. El script utilizado se encuentra en el repositorio del trabajo [2]. Esta implementación de filtro se añadió a un ejemplo de `NaveGo` levemente modificado, para obtener la señal de medición con el ruido generado por las funciones propias de este toolbox.

```
for i = 2 : N -2

    %----------Agregado de ruido en la medicion
    ref.wb = [q(1,i-1);ax_zero(1,1);ax_zero(1,1)]';  %Generacion de tita ruidosa
    titaNoise = gyro_gen(ref, imu1);
    qNoise(1,i-1)=titaNoise(1,1);
    q(1,i-1)  =titaNoise(1,1);
    z =  q(:, i-1);                  %z = tita
    uk = q(:, i-1);                   %u = wm

  %----------Filtro
   %Prediccion
    x_hat(:,i) = F* x_hat(:,i-1) + G*uk(2);
    P_a = F*P*F' + Q;
    Kk = P_a* H' * inv( H*P_a*H' + R );
    %Correccion
    x_hat(:,i) =  x_hat(:,i) + Kk .* (z(1) - H*x_hat(:,i));
```

[2] Repositorio

```
    P= P_a - Kk * H * P_a;

    q(1, i-1)=x_hat(1,i-1);
    q(:, i) = A*q(:, i-1) + B*u(:, i-1);
    e =titaRef(:, i-1)    - q(1, i-1);
    e1=titaRef(:, i) - q(1, i);
    u(:, i) = (e1+td*((e1-e)/dt))*kp;

end
```

Se graficaron la posición angular con ruido de medición y la misma pero realimentada a través del filtro de Kalman. Esta comparación se puede observar en la figura 13

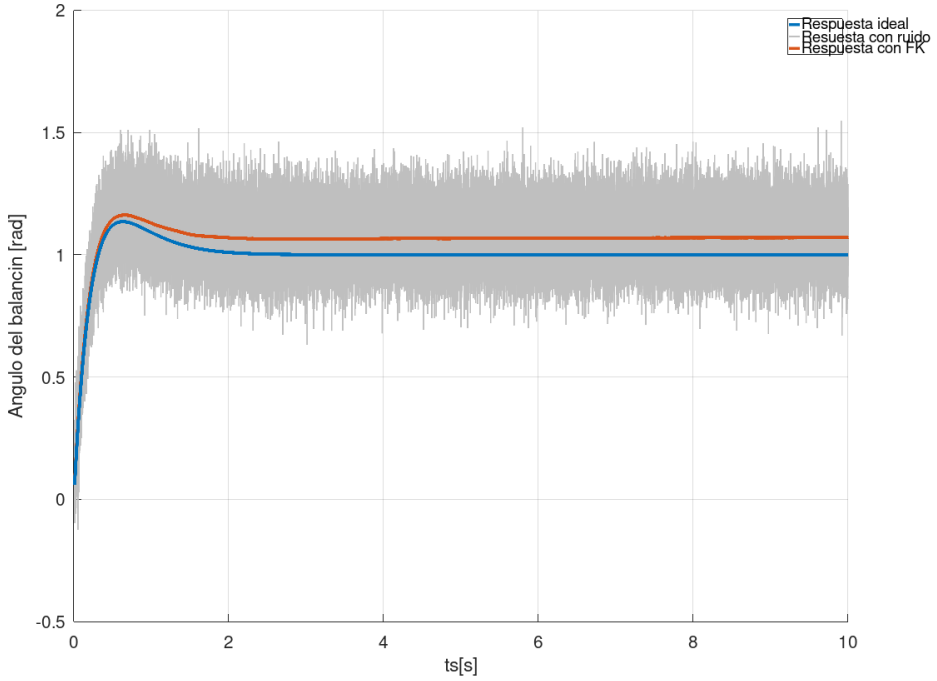


Figura 13: Gráfica de comparación Kalman vs Ideal vs Sin filtro

7. Conclusión

El simulador NaveGo permitió obtener el ruido generado particularmente para la IMU seleccionada. Esta herramienta nos permitió simular, en nuestra simulador de balancín, el ruido de la medición con la que se controla nuestro sistema.

Se pudo observar como las IMU no presentan un comportamiento ideal y que cómo el ruido afecta a cada una de sus mediciones dependerá de sus parámetros indicados en sus respectivas hojas de datos. En la figura 12 se puede observar como el control del sistema con una medición ruidosa generada particularmente por el modelo de IMU seleccionada, produce como salida una respuesta también ruidosa similar a la observar en el trabajo práctico anterior. En la figura 13 también se puede observar

lo mismo, con un ruido más notorio aunque también generado particularmente para la IMU especificada. La diferencia es que se utilizaron todas las funciones del propio ToolBox NaveGo.

En la misma figura 13 se puede observar como al añadir el filtro la respuesta obtenida se acerca a la respuesta ideal y mejora de una forma notable el ruido en la medición, sin embargo la respuesta aún sigue presentando un error respecto a la ideal.

Referencias

[1] RL Farrenkopf. Analytic steady-state accuracy solutions for two common spacecraft attitude estimators. *Journal of Guidance and Control*, 1(4):282–284, 1978.

[2] G Perez Paina, D Gaydou, J Redolfi, C Paz, and L Canali. Experimental comparison of kalman and complementary filter for attitude estimation. In *In Proceedings of 40th Argentine Conference on Informatics (JAIIO)*, pages 205–215, 2011.

A. Anexo A

```
clear all; close all; clc;

% Parametros de simulacion.
J = 100e-3          % Momento de inercia
T = 10              % Tiempo de simulacion
dt = 0.001          % Intervalo de muestreo
N = T/dt            % Indice maximo para estados discretos
ts = 0:dt:T-dt;  % Vector de tiempos discretos

%Parametros del sensor 3DM-GX1 (en unidades brindadas por el fabricante ; obtenidos de http://files.microstrain.com/3DM-
DMGX1.arw       = 3.5    .* ones(1,3);    % Angle random walks [X Y Z] (deg/root-hour)
DMGX1.arrw      = zeros(1,3);             % Angle rate random walks [X Y Z] (deg/root-hour/s)
DMGX1.vrw       = 0.4  .* ones(1,3);      % Velocity random walks [X Y Z] (m/s/root-hour)  [NOISE ACELEROMETRO]
DMGX1.vrrw      = zeros(1,3);             % Velocity rate random walks [X Y Z] (deg/root-hour/s)
DMGX1.gb_sta    = 0.7    .* ones(1,3);      % Gyro static biases [X Y Z] (deg/s)
DMGX1.ab_sta    = 10   .* ones(1,3);      % Acc static biases [X Y Z] (mg)
DMGX1.gb_dyn    = 0.02  .* ones(1,3);    % Gyro dynamic biases [X Y Z] (deg/s)
DMGX1.ab_dyn    = 0.2  .* ones(1,3);      % Acc dynamic biases [X Y Z] (mg)
DMGX1.gb_corr   = 100  .* ones(1,3);      % Gyro correlation times [X Y Z] (seconds)
DMGX1.ab_corr   = 100  .* ones(1,3);      % Acc correlation times [X Y Z] (seconds)
DMGX1.freq      = 1/dt;                 % IMU operation frequency [X Y Z] (Hz)
DMGX1.m_psd     = 0  .* ones(1,3);    % Magnetometer noise density [X Y Z] (mgauss/root-Hz) [ESTE NO SALE EN LA HOJA DE DAT

imu = imu_si_errors(DMGX1, dt);        % Funcion para convertir las unidades del fabricante a las del Sistema Internacion

%% Simulacion de ruidos

M = [N, 3];

%-------------------------------------------------------------------------
% Simulacion de bias estatico

gb_sta = noise_b_sta (imu.gb_sta, N);

%-------------------------------------------------------------------------
% Simulacion de ruido blanco

wn = randn(M);
g_wn = zeros(M);

for i=1:3

    g_wn(:, i) = imu.g_std(i) .*  wn(:,i);
end

%-------------------------------------------------------------------------
% Simulacion de bias dinamico (inestabilidad de bias)

gb_dyn = noise_b_dyn (imu.gb_corr, imu.gb_dyn, dt, M);

%-------------------------------------------------------------------------
% Simulacion de rate random walk

g_rrw = noise_rrw (imu.arrw, dt, M);

%-------------------------------------------------------------------------
%Error total en cada medicion de velocidad angular
error = g_wn(:,1) + gb_sta(:,1) + gb_dyn(:,1) + g_rrw(:,1); %Nos quedamos unicamente con los errores en X (usamos solo 1

% Vector de estado inicial.
% q(1) = theta (posicion angular) ; q(2) = theta_punto (velocidad angular)
q0 = [0; 0];

% Vector de estados e inicializacion.
q = zeros(2, N);
q(:, 1) = q0;

% Ahora, u es lo que le llega a la planta (luego de restarse a la entrada y
% multiplicar por Kp). Se calcula dentro del bucle for.
u = zeros(1, N);
%u = ones(1, N);

% Vector de theta de referencia (Entrada del sistema)
% Aca se aplican las acciones de control
%ref = zeros(1, N);
ref = ones(1, N);    %Escalon unitario

% Ganancia proporcional
%Kp = 0.1
Kp = 1
%Kp = 10

% Constante del compensador
%Td = 0.1
Td = sqrt(0.4)   %Td critico
%Td = 2

%En la primera iteracion, el punto anterior de la derivada no esta definido
u(1, 1) = ( ref(1, 1) - u(1,1) + Td * ( ref(1, 1) - u(1,1) )/dt ) * Kp;
```

```
% Matrices del sistema de estados discretizado.
A = [1, dt; 0, 1];
B = [0; dt/J];

% Bucle para calculo de los estados.
for i = 1 : (N-1)
  q(:, i+1) = A*q(:, i) + B*u(:, i);
  q(2, i+1) = q(2, i+1) + error(i);   %Agregamos ruido de medicion a la VELOCIDAD ANGULAR (medida por el giroscopo)
  u(:, i+1) = ( ref(:, i+1) - q(1, i+1) + Td * ( ref(:, i+1) - q(1, i+1) - ( ref(:, i) - q(1, i) ) )/ dt ) * Kp;
end

f1 = figure();
subplot(3, 1, 1); plot(ts, q(1, :), "linewidth", 2);
grid on; xlabel('t [s]'); ylabel('q1 (\theta)');
title('Estado: Posicion angular')
subplot(3, 1, 2); plot(ts, q(2, :), "linewidth", 2);
grid on; xlabel('t [s]'); ylabel('q2 (\omega)');
title('Estado: Velocidad angular')
subplot(3, 1, 3); plot(ts, u(1, :), "linewidth", 2);
grid on; xlabel('t [s]'); ylabel('u');
title('Accion de control: Torque')
```