Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sistema de Prototipado para Multirotores

Autor: Antonio González Rot

Tutor: José Ángel Acosta Rodríguez

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sistema de Prototipado para Multirotores

Autor:
Antonio González Rot

Tutor:
José Ángel Acosta Rodríguez
Profesor Titular

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado: Sistema de Prototipado para Multirotores

Autor: Antonio González Rot
Tutor: José Ángel Acosta Rodríguez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia por todo su apoyo tanto moral como económico sin el cual no estaría donde estoy ahora. A todos los profesores y maestros que he tenido durante mis años de educación en especial a todos aquellos que me inculcaron su amor por la ciencia, así como a quienes me enseñaron que la respuesta habitual no es siempre la correcta. A mis amigos por aguantar mis tonterías.

Antonio González Rot
Sevilla, 2018

Resumen

En este proyecto se tratará de realizar un autopiloto para la placa *Raspberry Pi* +Navio2 el cual se implementará en la *Raspberry Pi* y en MATLAB® + SIMULINK® .

El autopiloto se dividirá en dos partes claramente diferenciadas, por un lado el *software* encargado de leer los sensores y enviar los datos hacia MATLAB® implementado en la *Raspberry Pi* , y por otro lado el mezclador y estimador implementado en MATLAB® .

Abstract

The main theme of this project is to develop an autopilot for the *Raspberry Pi* + Navio2 hardware and implement it in the *Raspberry Pi* and in MATLAB® + SIMULINK® .

The autopilot will be divided in two parts, on the one hand, it will be a software reading and sending the sensor measurement from the *Raspberry Pi* to MATLAB® , where a motor mixer and a estimator will be implemented.

Índice Abreviado

Índice

Notación

gdl	Grados De Libertad
IMU	Inertial Measurement Unit. Unidad de Medida Inercial
RC	Radio Control
dps	Degrees Per Second. Grados Por Segundo. $^\circ/s$
SoC	Sistem on a Chip (Todos los componentes en el mismo chip, CPU, Ram, GPU, etc)
GPS	Global Position System. Sistema Global de Posicionamiento
UAV	Unmanned Aerial Vehicle. Vehículo Aéreo No Tripulado
EKF	Extended Kalman Filter. Filtro de Kalman Extendido
VICON	Sistema de captura de movimiento
Lidar	Sensor de distancia láser por tiempo de vuelo
NED	North, East, Down. Norte, Este, Abajo

Introducción

Dime y lo olvido, enséñame y lo recuerdo, involúcrame y lo aprendo

B. Franklin

Durante los últimos años se ha producido un gran repunte en la investigación y desarrollo de todo tipo de vehículos aéreos no tripulados y junto a ellos distintos *software* centrados no solo en la lectura de los sensores de abordo sino en el control de los mismos. El principal inconveniente de este tipo de *software* es que por lo general son especialmente complicados y demasiado genéricos por lo que si se intentan implementar algo en ellos fuera de lo que ya existe, esta tarea se vuelve realmente complicada lo que alarga el tiempo de desarrollo de forma más que considerable.

Algunos ejemplos de estos *software* son:

- PX4 [1] que tiene **515674 líneas** de código repartidas en sus **2270 ficheros**.
- Ardupilot [2] que tiene **901307 líneas** de código repartidas en sus **2880 ficheros**.

A pesar de estos inconvenientes la principal ventaja de estos *software* es que funcionan muy bien con una gran variedad de *hardware* por ejemplo PX4 puede funcionar con los autopilotos ”Navio[3]”, ”Pixhawk[4]” o ”Snapdragon Flight[5]” entre muchos otros [6].

Otra de las ventajas es que puede ser utilizado para distintas configuraciones de vuelo, por ejemplo, Arducopter da soporte para multirotores, helicópteros, aviones de ala fija, entre muchas otras, así como para vehículos tanto terrestres como acuáticos [7].

Objetivos del proyecto

Este proyecto se basará en el desarrollo de un *software* capaz leer los sensores que incorpora el autopiloto Navio2 y enviarlos a MATLAB® donde se implementarán los estimadores de posición y actitud así como los filtros correspondientes para su correcto funcionamiento junto con un pequeño mezclador para el control de bajo nivel de los motores.

Además de esto se desarrollará un programa encargado de lanzar varias copias de este *software* y monitorizar que todas ellas tengan un funcionamiento correcto, si esto no ocurriera se relanza una nueva copia y se repite el ciclo. De esta forma conseguimos incluir redundancia proporcionando mayor seguridad ante fallos mediante la paralelización de procesos.

Motivación del proyecto

La principal motivación que ha propiciado este proyecto es que hasta ahora se estaba utilizando un Firmware basado en PX4 [1] con algunas modificaciones para adaptarlas a nuestras necesidades. Sin embargo cada vez que se pretendía añadir algún modulo nuevo o modificar alguno ya existente teníamos que enfrentarnos a varios días, incluso semanas, de laboriosa integración con el objetivo de hacerlo funcionar de la forma deseada.

Con esta propuesta se ha conseguido pasar de 2270 ficheros de PX4 a 63 de los cuales solo habría que modificar 4 para incluir un nuevo sensor o una nueva característica además de añadir, si existiese, la librería del mismo. De esta forma se podría llegar a reducir de forma más que significativa el desarrollo. A esto además habría que añadir el modelo en SIMULINK® y las dos funciones que integra.

Materiales y Configuración necesaria

El que quiere algo conseguirá un medio, el que no una excusa

STEPHEN DOLLEY

En este capítulo se explicará los distintos materiales utilizados para el desarrollo del proyecto así como toda la configuración previa necesaria para su correcto funcionamiento. En primer lugar se pondrá de manifiesto los materiales utilizados junto con el software mínimo necesario para conseguir el objetivo marcado en este proyecto.

Navio2

Como placa de sensores se ha utilizado el autopiloto Navio2 [8], el cual se puede ver en Figura 2.1 y cuyas características se detallan desde la Tabla 2.1 a la Tabla 2.4.

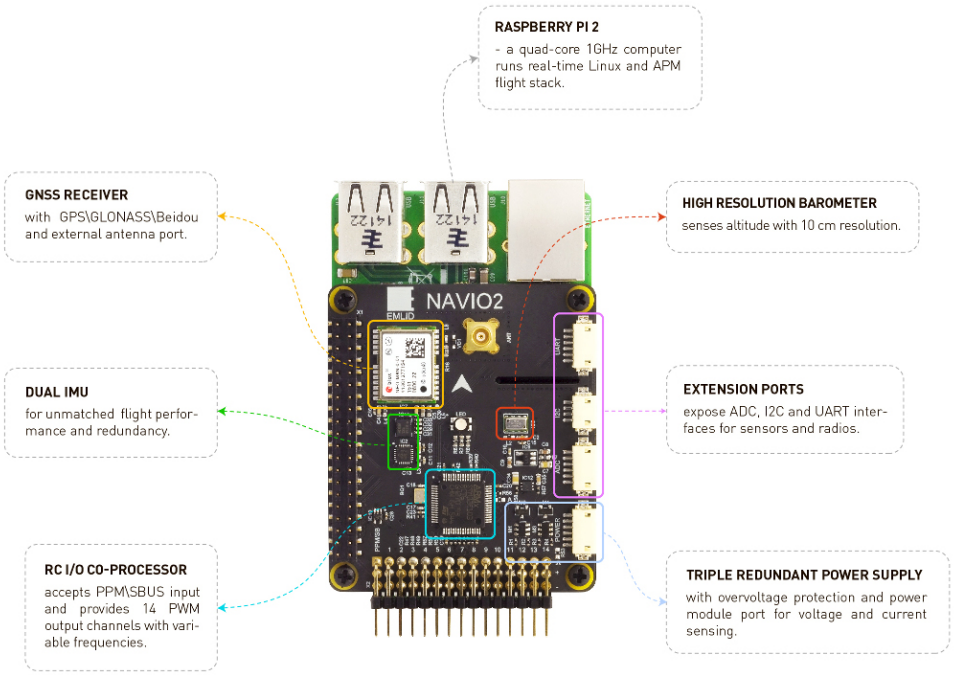

Figura 2.1 Placa Navio2.

Tabla 2.1 Especificación mecánica Navio2.

Características mecánicas	
Tamaño	55 x 66 mm
Peso	23 g
Temperatura operación	-40 ... +85 °C

Tabla 2.2 Especificaciones eléctricas Navio2.

Características eléctricas	
Voltaje alimentación	4.75-5.25V
Fuente de alimentación	USB, Módulo de potencia, Servo rail
Consumo de corriente	<150mA

Tabla 2.3 Puertos de la placa Navio2.

Puertos
Módulo de potencia
UART
I2C
ADC
12 salidas PWM para servo
Bus comunicación RC

Tabla 2.4 Sensores de la placa Navio2.

Sensores
IMU MPU9250 de 9 gdl
IMU LSM9DS1 de 9 gdl
Barómetro MS5611
GPS U-blox con GLONASS
Co-Procesador Cortex-M3 RC I/O
LED RGB

Principales características de los sensores

IMU MPU9250

IMU de 9 gdl que nos proporciona aceleración lineal, velocidad angular y medida del campo magnético en tres ejes, esta última con una precisión bastante baja. Además es capaz de medir la temperatura, para realizar las correcciones pertinentes.

Tabla 2.5 Especificaciones IMU MPU9250.

Sensor[9] Unidades:	Rango Gyro $(^\circ/s)$	Ruido Gyro Rate $dps/\sqrt{Hz}$	Rango Accel (g)	Rango Magnet. μT	Salida Digital	Alimentación Lógica (V)	Alimentación Operativa $(V \pm 5\%)$
	±250 ±500 ±1000 ±2000	0.01	±2 ±4 ±8 ±16	±4800	I^2C or *SPI*	1.7V to VDD or VDD	2.4V to 3.6V

IMU LSM9DS1

De nuevo una IMU de 9 gdl con que nos proporciona aceleración, velocidad angular y esta vez si, una buena medida del campo magnético. Al igual que el sensor anterior también proporciona una medida de la temperatura ambiente.

Tabla 2.6 Especificaciones IMU LSM9DS1.

Sensor[10] Unidades:	Rango Gyro $(^\circ/s)$	Rango Accel (g)	Rango Magnet. *gauss*	Salida Digital	Alimentación Lógica (V)	Alimentación Operativa $(V \pm 5\%)$
	±245 ±500 ±2000	±2 ±4 ±8	±4 ±8 ±12 ±16	I^2C or *SPI*	1.7V to VDD or VDD	2.4V to 3.6V

Barómetro MS5611

Sensor que nos proporciona una medida bastante precisa de presión y temperatura ambiente. Con la que posteriormente se puede hacer un cálculo de la altura.

Tabla 2.7 Especificaciones Barómetro MS5611.

	Sensor[11]	Rango	Resolución	Precisión
Presión $(mbar)$		10...1200	0.065 / 0.042 / 0.027 / 0.018 / 0.012 **(1)**	$\pm$1.5 **(2)**
Temperatura $(^{\circ}C)$		-40...+85	<0.01	$\pm$0.8
(1) ratio de sobremuestreo: 256 / 512 / 1024 / 2048 / 4096 **(2)** a 25$^{\circ}C$ y 750 $mbar$				

GPS Ublox-M8N

Sensor que nos proporciona las coordenadas de longitud y latitud terrestres junto a la altura sobre el nivel del mar y el elipsoide. También recibe del satélite la unidad de tiempo del reloj atómico y un valor de precisión de la posición horizontal y vertical.

Tabla 2.8 Especificaciones GPS Ublox-M8N.

Sensor[12] Unidades	Tipos	Tasa de actualización Hz	Precisión de la velocidad m/s	Precisión en el cabeceo $^{\circ}$	Precisión posición m
	GPS / GLONASS / GALILEO / BeiDou	5	0.05	0.3	2.5

Raspberry Pi 3 Model B

La placa Navio2 de forma individual sería inútil ya que está pensada para ser utilizada con el microcomputador *Raspberry Pi* 3 model B, Figura 2.2. En ella será necesario instalar una versión especial de "Raspbian", un sistema operativo en base Linux, proporcionado directamente por Emlid, los desarrolladores de Navio [13], así como las librerías para el control de los sensores también proporcionadas por Emlid [14].

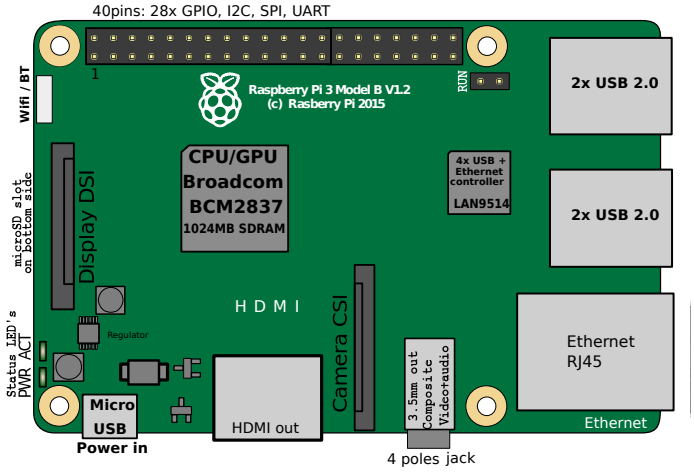

Figura 2.2 Microcomputador *Raspberry Pi* 3 Model B.

Características Hardware

Tabla 2.9 Características *Raspberry Pi* 3 Model B[15].

Arquitectura	SoC	CPU	Memoria (SDRAM)	USB
ARMv8-A (64/32-bit)	Broadcom BCM2837	1.2 GHz 64-bit quad-core	1 GB (shared with GPU)	4

Almacenamiento	Conectividad	Periféricos	Tamaño	Peso
MicroSDHC slot, USB Boot Mode	10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1	17× GPIO	85.60 × 56.5 mm	45 g

Características Software

La principal diferencia entre el Sistema Operativo que nos proporciona Emlid, es que se trata de un sistema preparado para tener el mejor funcionamiento posible en aplicaciones de tiempo real gracias a modificaciones del Kernel. Consiguiéndose latencias mucho menores que con el Raspbian original tal y como se ve en la Figura 2.3.

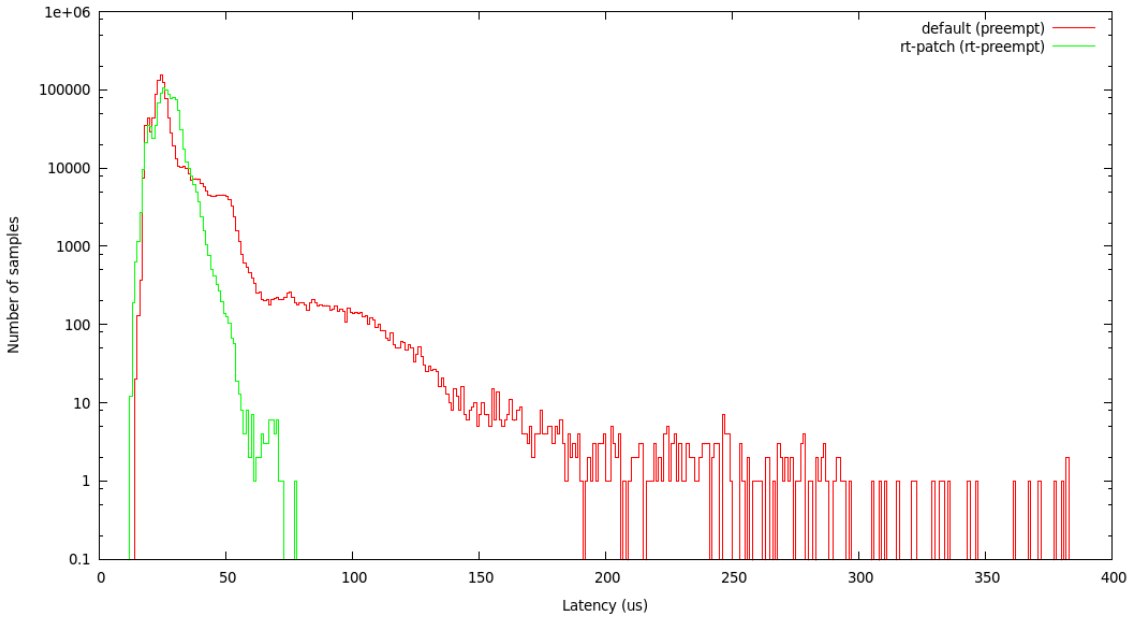

Figura 2.3 Comparación entre las latencias de raspbian y el SO de Emlid[16].

Por otro lado también incluye el acceso a todos los periféricos y sensores propios de la placa Navio2 de forma nativa. Permitiendo de esta forma comunicación inmediata con cada uno de ellos, la cual no sería posible con la version original de raspbian. Por ultimo es importante mencionar que esta versión no incluye interfaz gráfica ya que no es necesaria para aplicaciones en UAVs.

Intel NUC

Como ordenador principal de abordo se dispone de un Intel NUC[17], un pequeño ordenador en el que se encuentra instalado Ubuntu en su versión 16.04 LTS como sistema operativo junto a MATLAB® 2016b en el cual se implementarán los controladores y toda la etapa de recepción e interpretación de las medidas de los sensores enviadas desde Raspberry.

Características Hardware

Tabla 2.10 Características Intel NUC.

Procesador	Conexiones	Memoria Interna	Memoria Ram	Peso	Tamaño
Intel i7 séptima generación	micro HDMI, 2xUSB 2.0, 2xUSB 3.0, Ethernet, puerto M2	ssd 240 GB	16 GB	0,8 kg	11.1 x 11.5 x 5.1cm

Figura 2.4 Imagen de el ordenador Intel NUC.

Drivers para Navio2

Tanto si piensas que puedes, como si piensas que no, estás en lo cierto

Henry Ford

En este capítulo se expondrá todo el software propio desarrollado para el control y manipulación de Navio así como los elementos de seguridad de los mismos. En concreto se comentarán los programas ”navio_types.h”, ”DriversNavio2.c”, ”DriversNavio2.h” y ”Monitor.c” que son los que se han incluido a las librerías facilitadas por Emlid para el control de todos los sensores incorporados en Navio2.

En la Figura 3.1 se observa el esquema general de todo el software desarrollado y la interacción entre los distintos programas.

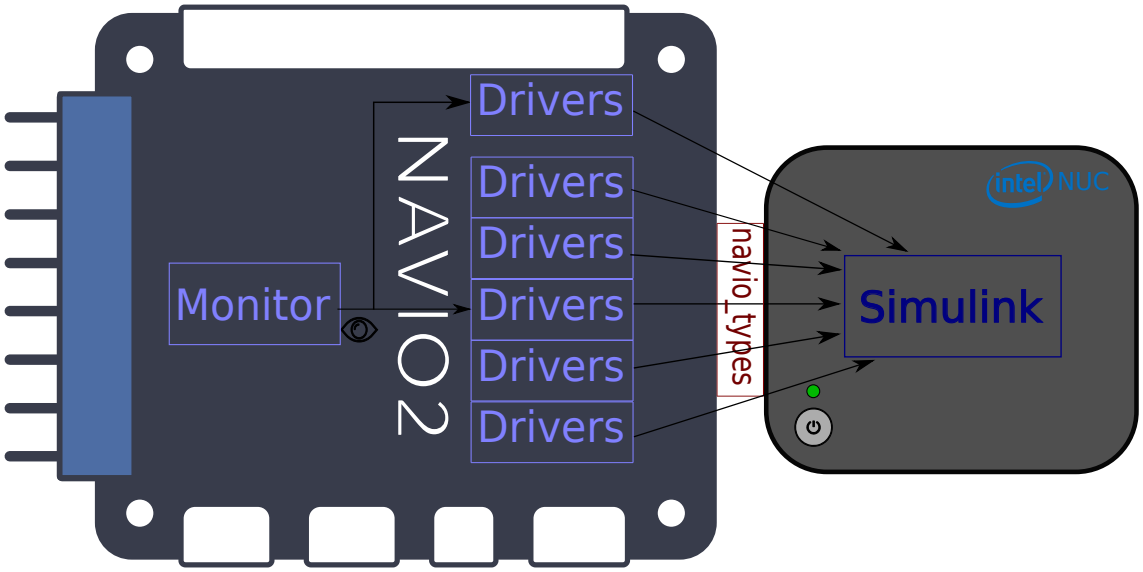

Figura 3.1 Esquema general del software desarrollado.

Se puede apreciar que existen dos elementos claramente diferenciados, por un lado se encuentra la *Raspberry Pi* encargada de medir los sensores que se incorporan en la placa Navio y enviarlos a SIMULINK® y por otro lado está SIMULINK® instalado en el ordenador NUC en el cual se analizarán las medidas recibidas, realizando a la vez las estimaciones de posición y actitud, así como el mezclador de los motores, todo este software se comentará con más detalle en el Capítulo 5. A continuación se comentará con detalle cada uno de los programas incluidos en la *Raspberry Pi* .

Cabe destacar que aunque en el software final se incluyen sensores externos a la placa Navio2, como son un VICON y un *Lidar*, estos no serán finalmente utilizados para las estimaciones por su ausencia para la realización de pruebas. Sin embargo se incluye en todos los casos para una futura implementación.

Navio_types

Como complemento a las definiciones y funciones proporcionadas por emlid se ha añadido una nueva librería que incluye la definición de las estructuras que almacenarán las medidas de los distintos sensores. De esta forma es mucho más fácil saber que tipo de dato se está escribiendo en cada momento además de facilitar el posterior envío vía UDP.

Ejemplo de las estructuras utilizadas en navio_types.h

A continuación se muestran algunos ejemplos de las estructuras utilizadas para almacenar los datos leídos de los sensores, en concreto se muestran las correspondientes a las IMUs y al GPS.

Código 3.1 Estructura para almacenar los datos de la IMU.

```
/* Estructura para almacenar las lecturas de las IMUs
   */
typedef struct
{
  float _temperature;
  float _ax;
  float _ay;
  float _az;
  float _gx;
  float _gy;
  float _gz;
  float _mx;
  float _my;
  float _mz;
}shm_imu;
```

Código 3.2 Estructura para almacenar los datos del GPS.

```
/* Estructura para almacenar la lectura del GPS*/
typedef struct
{
  double _time;
  double _long;
  double _lat;
  double _height_elips;
  double _height_sea;
  double _hor_accur;
  double _ver_accur;
}shm_gps;
```

Monitor

Esta aplicación es la encargada de arrancar tantos procesos hijos (*drivers*) como se le indique como argumento de linea de comandos. Una vez realizado esto se bloqueará a la espera de que alguno de estos procesos creados sufra algún problema en cuyo caso arrancará otro exactamente igual como se puede ver en la Figura 3.2.

Cabe mencionar que todos los procesos que arranca esta aplicación son idénticos a excepción de dos parámetros:

- El identificador propio de cada driver, para que cada uno utilice puertos UDP distintos.
- Y que a excepción del primer proceso que se crea, ninguno accede a la lectura del barómetro y el GPS ya quSe estos sensores solo permiten acceso por parte de un proceso de manera simultánea.

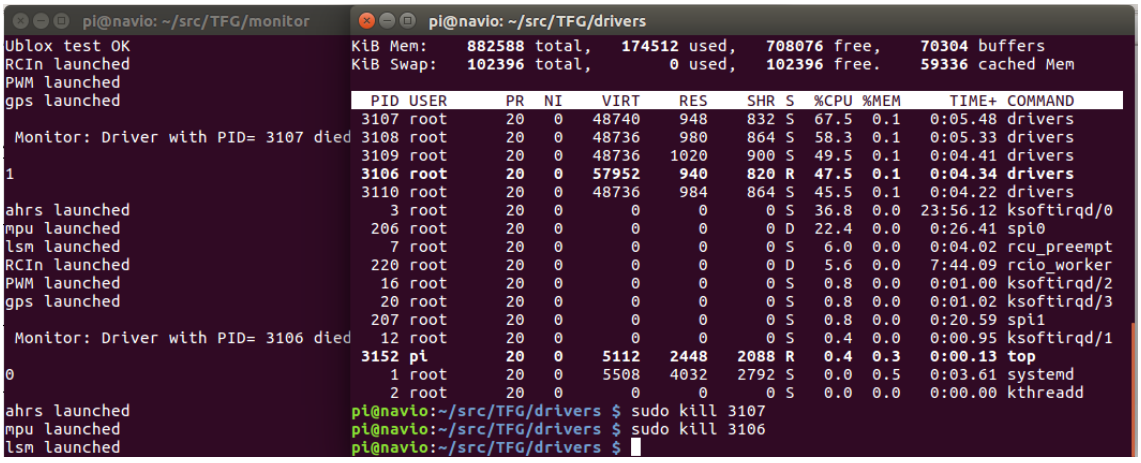

Figura 3.2 Proceso de espera y arranque del monitor.

En la figura anterior se puede ver como de forma manual se ha parado uno de los procesos que se habían arrancado y el monitor lo detecta de forma inmediata arrancándolo de nuevo.

Esquema de funcionamiento

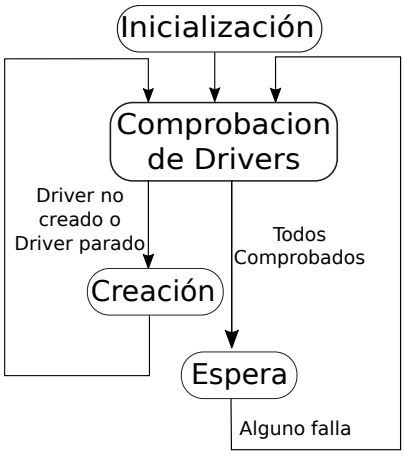

Figura 3.3 Esquema de funcionamiento proceso monitor.

A continuación se muestra la inicialización del proceso monitor, sonde se declaran las variables que se utilizarán durante el mismo así como la lectura del número de drivers que se deberán arrancar.

Código 3.3 Inicialización Monitor.

```
 [...]
/*Número máximo de procesos que se pueden iniciar */
#define max_proc 7

/*Hilo Principal */
int main( int argc, char *argv[])
{
  /* Declaración e inicializaci ón de variables */
  pid_t monitor;                  // PID de este proceso
  pid_t drivers[max_proc];        // PID del proceso arrancado
  pid_t died_driver;              // PID del proceso que falla
  int status;                     // Bandera que indica el motivo del fallo
  int flag[max_proc]={1,1,1,1,1,1,1,1}; // Bandera que indica si un proceso falla
                                  // 1-->Proceso parado, 0--> Iniciado
  int i;            // Contador de iteraciones para arrancar los procesos
  int N;            // Número de procesos a arrancar final
  char id[5];       // identificador del proceso arrancado

  /*Asignación de Número final de procesos a arrancar*/
  if(argc>1) // Si se recibe dato al iniciar
  {
    N=atoi(argv[1]);
    if(N>max_proc)//Si el numero deseado de procesos a arrancar es mayor se satura
      N=max_proc;
  }
  else  // Si no se recibe número de procesos a arrancar siempre se arrancan 3
    N=3;
 [...]
```

En el siguiente código se puede ver como se comprueba cada driver deforma individual para ver si este aún se encuentra en ejecución. En concreto se muestran las comprobaciones pertinentes de cada caso en concreto, es decir, el driver 0, que arrancará todos los sensores y el resto que no arrancarán ni el barómetro ni el GPS.

Código 3.4 Comprobación *drivers* Monitor.

```
[...]
    for(i=0;i<N;i++) // Se recorren el número de procesos que se desea arrancar
    {
      if(flag[i]==1)  // Si el proceso no está iniciado, se inicia
      {
        if((drivers[i] = fork()) < 0) // Si falla el fork se para todo
        {
          perror("fork failure");
          exit(1);
        }
      }
      if(drivers[i] == 0 && flag[i]==1 && i==0) //Si es el primer proceso el que
                                                //no está inicado

      else if(drivers[i] == 0 && flag[i]==1 && i>0) //Si el proceso no es el primero

      else   //Si es el proceso padre se indica que el hijo ya ha arrancado y se
             //continua a la siguiente iteración
      {
        flag[i]=0;
        continue;
      }

[...]
```

En el siguiente fragmento se muestra como es el proceso de arranque de cada driver, en el cual se obtiene el PID del mismo, se le asigna la máxima prioridad y se inicia con el identificador del mismo y los sensores que deseamos arrancar.

Código 3.5 Creación *drivers* Monitor.

```
[...]
          sprintf(id,"%d",i);        //Se le asigna el identificador
          drivers[i] = getpid();     //Se guarda el PID con el que arranca
          printf("driver with id=%s and pid=%d started\n",id,drivers[i]);
          setpriority(PRIO_PROCESS, drivers[i], -20);//Se le asigna prioridad máxima
         /*Se inicia con los sensores que pueden funcionar en paralelo
          *(Todos menos el barómetro y el GPS)*/
          execl("../drivers/drivers","drivers",id,"mpu","lsm","rc","pwm","ahrs",NULL);
          perror("execl() failure!\n\n");
          printf("This print is after execl() and should not have been executed if execl were successful!\n\n");
          _exit(1);
[...]
```

Por último se muestra como se realiza la espera del fallo de alguno de los drivers y la identificación de la misma una vez que esta ocurre.

Código 3.6 Espera *drivers* Monitor.

```
    died_driver=wait(&status);
    printf("\n Monitor: Driver with PID= %d died\n\n", died_driver);
    for(i=0;i<N;i++)//comprobacion del driver que ha fallado
    {
      if(drivers[i]==died_driver)
        flag[i]=1;
    }
```

Drivers

Este software recibe como argumento de linea de comandos un identificador propio, en función del cual asigna un puerto distinto para el envío y recepción de los *sockets* UDP con los que se comunica con SIMULINK®. Por otro lado también recibe los sensores que se quieren utilizar, de forma que solo se arrancan los sensores que realmente vamos a utilizar, para ello se inicia un hilo por sensor el cual una vez inicializado se queda en

un bucle leyendo dicho sensor. Este bucle se mantendrá activo mientras no se supere un cierto tiempo entre cada iteración, si esto llega a ocurrir el hilo sale de la espera, termina el bucle y salimos del hilo.

Por otro lado el hilo principal entra en un bucle infinito el cual en cada iteración comprueba si todos los hilos de los sensores siguen activos para en el caso de que no sea así arrancarlos de nuevo. Hecho esto, envía los datos leídos por los sensores cuyos hilos no han sufrido ningún problema mediante un *socket* UDP hacia MATLAB® junto a una bandera que indica si alguno de estos ha sufrido una parada para poder obviar sus datos de así necesitarlo. Al igual que el resto de hilos tiene un temporizador que en el caso de ser superado se termina el proceso.

Los *sockets* que utiliza el hilo main son los siguientes:

- **Puerto 8081+**$id_{proceso} * 3$: Datos de ambas IMUs, estimación de la actitud usando las funciones de Navio2, la diferencia de tiempo entre medidas y banderas que indica la caída de algunos de los hilos que controlan las IMUs.
- **Puerto 8082+**$id_{proceso} * 3$: Datos recibidos de la emisora RC junto a dos banderas que indican si ha habido algún error en los hilos que controlan los datos de la emisora y los motores.
- **Puerto 8083+**$id_{proceso} * 3$: Datos utilizados para la estimación de la posición, GPS, Barómetro, TotalStation, Lidar así como sus respectivas banderas para indicar la caída de algunos de los hilos.

Por otro lado, cabe destacar que todos los temporizadores encargados de que ningún hilo tarde más tiempo de lo debido en ejecutarse están realizados con señales ”posix” de tiempo real, las cuales llaman a un manejador (Código 3.7) el cual en función de la señal recibida, la cual es distinta para cada hilo, desactiva una bandera que para la ejecución del hilo correspondiente. Además se activa la bandera ”someone_died” que indica que se ha producido algún problema para que el hilo principal realice la comprobación del hilo caído para relanzarlo. Si el hilo que tiene el problema es el hilo principal se reinicia el proceso completo.

En el siguiente código se muestra un fragmento del manejador de la interrupción de sobretiempo de cualquiera de los hilos de lectura de los sensores, en concreto el hilo principal y el de lectura de la emisora. El resto es exactamente igual y solo difiere en la señal que se recibe.

Código 3.7 Manejador de temporizador de sobretiempo.

```
void timer_handler(int sig, siginfo_t *si, void *uc)
{
  someone_died=1; //Indicamos que algún hilo ha fallado
  /*Detectamos que hilo falla en función de la señal de fallo*/
  if(sig==SIGRTMIN)
  {
    main_exit=1;
    exit(0);
  }
  else if(sig==SIGRTMIN+1)
  {
    RC_exit=1;
  }
 [...]
```

Por ultimo cabe destacar que para la temporalización de la ejecución cíclica de cada uno de los hilos se ha probado con dos tipos de esperas:

- **”usleep”** que realiza una espera durante los μs especificados independientemente del tiempo de ejecución del propio hilo.
- **”clock_nanosleep”** que realiza una espera absoluta pudiendo sincronizar de esta forma los procesos de forma mucho más exacta.

Por ejemplo si la ejecución del hilo varía entre 1 y 2 ms y nosotros queremos una ejecución fija cada 4ms. Con clock_nanosleep podemos leer el tiempo absoluto de ejecución al principio del hilo y sumarle el tiempo que queremos que espere, de forma que realicemos la espera hasta ese tiempo absoluto independientemente del tiempo de ejecución. Además si la ejecución del hilo supera el tiempo de ejecución cíclico esperado directamente no se realiza ninguna espera. Mientras que usleep siempre realizaría una espera de 4ms al finalizar la ejecución del proceso por los que los tiempos sería prácticamente impredecibles.

En la Figura 3.4 se puede observar una comparativa de los tiempos de ejecución entre ambas opciones. Se observa que la ejecución con ”clock_nanosleep” es mucho más exacta y estable que la opción con usleep.

Por otro lado en el Código 3.8 se ve la forma en que se realiza esta espera.

Figura 3.4 clock_nanosleep vs usleep con tiempo objetivo 4ms.

A continuación, se muestra el mecanismo de espera implementado para la repetición cíclica de los hilos de lectura de los sensores.

Código 3.8 Espera entre ejecuciones.

```
    /*Cálculo del tiempo de espera*/
    clock_gettime(CLOCK_REALTIME,&slp);
    if ((slp.tv_nsec+RC_rate*1000)<(1000000000))
      slp.tv_nsec+=RC_rate*1000;
    else
    {
      slp.tv_sec+=1;
      slp.tv_nsec=slp.tv_nsec+RC_rate*1000-1000000000;
    }
[...]
    /*Espera para la repetición*/
    clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,&slp,NULL);
```

Esquema de funcionamiento hilo principal

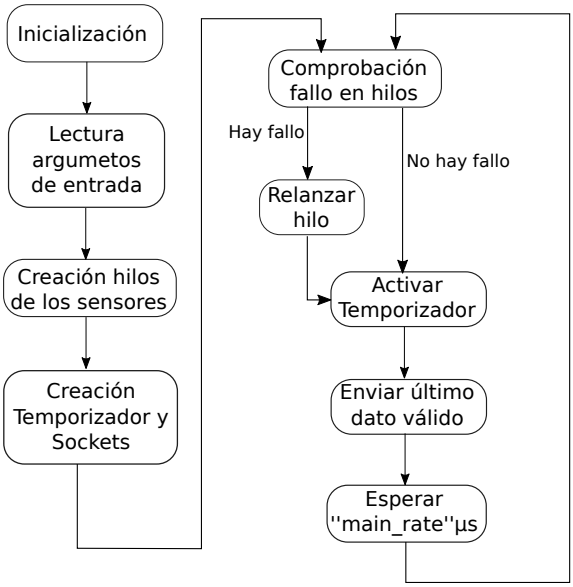

Figura 3.5 Esquema de funcionamiento hilo principal.

Aquí se puede observar la incialización de las variables necesarias para el correcto funcionamiento del hilo principal.

Código 3.9 Inicialización hilo Principal Drivers.

```
int main(int argc, char * argv[]) {
/*Declaración e inicializaci ón de variables */
    signal(SIGINT,quit_handler); // Asignación de manejador de interrupcion fin programa
    /* tabla que indica el sensor que se va a iniciar */
    bool sensor[10]={ false , false , false , false , false , false , false , false , false , false };
    /* Variables temporizadores */
    timer_t timerid; // Identificador
    struct sigevent sev; // Señal que indica el evento
    struct itimerspec its; // Valor del temporizador
    struct itimerspec its_cero; // Temporizador con valor 0
    sigset_t mask; // Máscara de la señal
    struct sigaction sa; // Accion de la señal
    /*Banderas que indican el fallo de un sensor*/
    bool mpu_fail, lsm_fail , baro_fail ,RC_fail ,PWM_fail, lidar_fail , gps_fail , total_fail ;
    struct timespec slp={0,0}; // Variable para almacenar el timepo de espera
    /* Diferencial de tiempo entre envío de señales */
    float dt;
    struct timeval tv;
    static unsigned long previoustime=0, currenttime=0;
[...]
```

En el siguiente código se muestra la forma en la que se detecta que sensores se deben arrancar a partir de la lectura de los parámetro de entrada al iniciar el proceso driver.

Código 3.10 Lectura argumentos de entrada.

```
[...]
/*Lectura de los sensores que se van a iniciar */
    if (argc > 2){ // Si se ha recibido como parámetro algo aparte del identificador
      /*
      * sensor[0]=>mpu
      * sensor[1]=>lsm
      * sensor[2]=>baro
```

```
        * sensor[3]=>rc
        * sensor[4]=>PWM
        * sensor[5]=>adc
        * sensor[6]=>ahrs  // only  if  mpu is  used
        * sensor[7]=>sf11c
        * sensor[8]=>  totalstation
        * sensor[9]=>gps
        * */
        for  ( int  i=2;i<argc;i++){  // Se comprueba cada parámetro para  ver  si  coincide
                                      // con  el  sensor
          if (strcmp(argv[i], "mpu")==0)sensor[0]=true;
          if (strcmp(argv[i], "lsm")==0)sensor[1]= true ;
          if (strcmp(argv[i], "baro")==0)sensor[2]= true ;
          if (strcmp(argv[i], "rc")==0)sensor[3]= true ;
          if (strcmp(argv[i], "pwm")==0)sensor[4]=true;
          if (strcmp(argv[i], "adc")==0)sensor[5]= true ;
          if (strcmp(argv[i], "ahrs")==0)sensor[6]= true ;
          if (strcmp(argv[i], "sf11c")==0)sensor[7]= true ;
          if (strcmp(argv[i], "  totalstation  ")==0)sensor[8]= true ;
          if (strcmp(argv[i], "gps")==0)sensor[9]= true ;
        }

      }else  if (argc==2){   // Si  solo  se  ha  recibido  el   identificador   de  proceso
                             // se  hace  un  inicio  por  defecto
        sensor[0]= true ;
        sensor[1]= true ;
        sensor[2]= true ;
        sensor[3]= true ;
        sensor[4]= true ;
        sensor[5]= false ;
        sensor[6]= true ;
        sensor[7]= false ;
        sensor[8]= false ;
        sensor[9]= true ;
      }
      else{  // Si  no  se  recibe  el   identificador   no  se  sigue  ejecutando  el  proceso
         printf (" falta   identificador   del  proceso\n");
         exit (0) ;
      }
      // Asignación del   identificador   de  proceso
      id=atoi(argv[1]) ;
       printf (" identicador  de  proceso=%d\n\n",id);
      if  (check_apm()) {
        return  1;
       }[...]
```

En este fragmento se muestra la creación de los drivers, en concreto se muestran las dos IMUs, así como la declaración de las estructuras necesarias las cuales son todas similares a lo mostrado en el primer comentario. El resto de sensores tienen un procedimiento de arranque similar a los ya mostrados.

Código 3.11 Creación hilos.

```
[...]
  /*** estructura   sensores***/
    /*typedef   struct
     *{
     *       IdentificadorSensor   _sensor;
     *      shm_TipoSensor _shmmsg; //definido  en  Navio_Types.h
     *}lsm9ds1_imu_str;*/

    /*Declaración de  estructuras   necesarias  para  almacenar  los  datos  de  cada  sensor*/
    mpu9250_imu_str mpu9250_imu;
    lsm9ds1_imu_str lsm9ds1_imu;
     baro_str  barometer;
     rcinput_str   rcin ;
     rcoutput_str   rcout ;
     adc_str   adc_signals ;
     sf11c_str   sf11c_signals ;
      totalStation_str      totalstation_signals   ;
     gps_str   gps_signals ;
```

```
    /*Se arranca mpu*/
    if (sensor[0]){
        printf("mpu launched\n");
       mpu9250_imu._mpu9250_imu.initialize();
        pthread_t mpu9250_imu_thread;
        if( pthread_create(&mpu9250_imu_thread, NULL, acquireMPU9250Data, (void*)&mpu9250_imu))
        {
            printf("Error: Failed to create mpu9250_imu thread\n");
            return 0;
        }
    }

    /*Se arranca lsm*/
    if (sensor[1]){
        printf("lsm launched\n");
       lsm9ds1_imu._lsm9ds1_imu.initialize();
        pthread_t lsm9ds1_imu_thread;
        if( pthread_create(&lsm9ds1_imu_thread, NULL, acquireLSM9DS1Data, (void*)&lsm9ds1_imu))
        {
            printf("Error: Failed to create barometer thread\n");
            return 0;
        }
    }
     [...] // identico para el resto de sensores
[...]
```

A continuación se puede observar como se crea el temporizador de sobretiempo y espera utilizado por el hilo principal.

Código 3.12 Creación Timer.

```
[...]
    /********** Se crea el ''watchdog'' del hilo principal **********/
    /*Se establece el manejador para la señal del temporizador*/
    sa.sa_flags = SA_SIGINFO;
    sa.sa_sigaction = timer_handler;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGRTMIN, &sa, NULL);

    /* Se bloquea la señal del timer temporalmente */
    sigemptyset(&mask);
    sigaddset(&mask, SIGRTMIN);
    sigprocmask(SIG_SETMASK, &mask, NULL);

    /* Creación del temporizador */
    sev.sigev_notify = SIGEV_SIGNAL;
    sev.sigev_signo = SIGRTMIN;
    sev.sigev_value.sival_ptr = &timerid;
    timer_create(CLOCK_REALTIME, &sev, &timerid);

    /* Establecimiento de los valores del temporizador*/
    its.it_value.tv_sec = 0;
    its.it_value.tv_nsec = main_rate*comp_factor*1000;
    its.it_interval.tv_sec = 0;
    its.it_interval.tv_nsec = 0;

    its_cero.it_value.tv_sec = 0;
    its_cero.it_value.tv_nsec = 0;
    its_cero.it_interval.tv_sec = 0;
    its_cero.it_interval.tv_nsec = 0;
[...]
```

El siguiente fragmento corresponde a la creación de los *sockets* necesarios para el envío de los datos vía UDP hacia SIMULINK® , en concreto se muestra la creación del *socket* encargado de enviar los datos de actitud, ya que el resto será idéntico a excepción del puerto utilizado.

Código 3.13 Creación *sockets* de envío.

```
[...]
    /********************* Declaración del socket **********************/
    /* Variables necesarias para el socket */
     struct sockaddr_in si_other_att ,si_other_RC, si_other_pos ;
     int s_att ,s_RC,s_pos, slen=sizeof( si_other_att );
    /* Estructuras a enviar */
     attitude_msg att_msg; // IMUs, AHRS, dt, bandera de fallo
    position_msg pos_msg; // Total , GPS, baro, lidar , bandera de fallo
    RCIn_msg RC_msg;      // Señal de radio recibida

    /*Creación del socket UDP*/
    if (( s_att =socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    {
      udp_die(" socket_att ");
    }

    /*Reserva de memoria de la estructura a enviar*/
    memset((char *) & si_other_att , 0, sizeof( si_other_att ));

    /* asignación de puertos a utilizar */
     si_other_att . sin_family = AF_INET;
     si_other_att . sin_port = htons(PORT_Att+(id*3));

    if ( inet_aton (SERVER , &si_other_att.sin_addr ) == 0)
    {
       fprintf ( stderr , " inet_aton () failed ( att )\n");
       exit (1);
    }
     [...] // idéntico para el resto de sockets
[...]
```

Además dentro del bucle de ejecución se comprobará que ningún hilo asociado a la lectura de los sensores haya sido abortado por sobre tiempo, en caso de que esto ocurriera se volverá a arrancar.

Código 3.14 Comprobación de fallo y re-lanzamiento de los sensores.

```
        /*Por defecto todos los sensores siguen funcionando*/
        mpu_fail= false ;
         lsm_fail = false ;
         baro_fail = false ;
        RC_fail= false ;
        PWM_fail=false;
         lidar_fail = false ;
        gps_fail = false ;
         total_fail = false ;
        /* si algun sensor ha fallado se comprueban todos para volver a arrancarlo */
        if (someone_died){
          someone_died=0;
           timer_settime ( timerid , 0, &its_cero , NULL);
          /*Se arranca ahrs*/
          if (mpu_exit){
            mpu_fail=true ;
             // ahrs_str get_ahrs ;
              printf ("ahrs restarted \n");
             mpu9250_imu._ahrs_required = true ;
          }
          /*Se arranca mpu*/
          if (mpu_exit){
             printf ("mpu restarted \n");
            mpu9250_imu._mpu9250_imu.initialize();
             pthread_t mpu9250_imu_thread;
             if ( pthread_create (&mpu9250_imu_thread, NULL, acquireMPU9250Data, (void *)&mpu9250_imu)){
                printf ("Error: Failed to create mpu9250_imu thread\n");
                return 0;
             }
          }
          [...]  // identico para el resto de sensores
```

Por último se muestra la recogida de datos y su actualización, siempre y cuando el hilo encargado siga funcionando, de los datos leídos de los sensores, así como el envío de los mismos utilizando los *sockets* creados anteriormente.

Código 3.15 Actualización y envío del último valor válido.

```
[...]
        // inicio de temporizador de timeout
        timer_settime(timerid, 0, &its, NULL);
        // Calculo del tiempo para la ejecucion ciclica
        clock_gettime(CLOCK_REALTIME,&slp);
        if ((slp.tv_nsec+main_rate*1000)<(1000000000))
          slp.tv_nsec+=main_rate*1000;
        else
        {
          slp.tv_sec+=1;
          slp.tv_nsec=slp.tv_nsec+main_rate*1000-1000000000;
        }

        // calculo del incremento de tiempo entre cada envío
        gettimeofday(&tv,NULL);
        previoustime = currenttime;
        currenttime = 1000000 * tv.tv_sec + tv.tv_usec;

        if(previoustime!=0){
          dt = (currenttime - previoustime) / 1000000.0;
          if(dt < 1/1300.0) usleep((1/1300.0-dt)*1000000);
          gettimeofday(&tv,NULL);
          currenttime = 1000000 * tv.tv_sec + tv.tv_usec;
          dt = (currenttime - previoustime) / 1000000.0;
        }else{
          dt=0;
        }

        //en caso de que no haya fallado el hilo se actualiza la estructura a enviar
        /* Restablecimiento del temporizador de espera */
        timer_settime(timerid, 0, &its, NULL);
        /*Tiempo a esperar, el actual más frecuencia del hilo */
        clock_gettime(CLOCK_REALTIME,&slp);
        if ((slp.tv_nsec+main_rate*1000)<(1000000000))
          slp.tv_nsec+=main_rate*1000;
        else
        {
          slp.tv_sec+=1;
          slp.tv_nsec=slp.tv_nsec+main_rate*1000-1000000000;
        }
        gettimeofday(&tv,NULL);
        previoustime = currenttime;
        currenttime = 1000000 * tv.tv_sec + tv.tv_usec;

        /*Cálculo del diferencial de tiempo entre iteraciones */
        if(previoustime!=0){
          dt = (currenttime - previoustime) / 1000000.0;
          if(dt < 1/1300.0) usleep((1/1300.0-dt)*1000000);
          gettimeofday(&tv,NULL);
          currenttime = 1000000 * tv.tv_sec + tv.tv_usec;
          dt = (currenttime - previoustime) / 1000000.0;
        }else{
          dt=0;
        }

        /*Reasignación de la lectura de los sensores al mensaje a enviar, si no ha
         *fallado el hilo del sensor correspondiente */

        /*Mensaje de actitud */
        if(!mpu_fail){
          att_msg.imu_mpu=mpu9250_imu._shmmsg;
          att_msg.quat_mpu=mpu9250_imu._quaternion;
        }
        if(! lsm_fail)
```

```
        att_msg.imu_lsm=lsm9ds1_imu._shmmsg;
      att_msg. fail_flags [0]=mpu_fail;
      att_msg. fail_flags [1]= lsm_fail ;
      att_msg.dt=dt;
       [...] // identico para todas las estructuras a enviar

      /*Envío de los datos por UDP*/
      if (sendto( s_att , &att_msg, sizeof (attitude_msg) , 0 , ( struct sockaddr *) & si_other_att , slen )==-1)
      {
        udp_die("sendto_Att ()");
      }

     [...] // identico para el resto de sockets
      /*Espera para repetir */
      clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,&slp,NULL);
    } // Fin while(1)
```

Esquema de funcionamiento hilo de sensor

El funcionamiento de cada uno de los hilos que se encargan de leer todos los sensores siguen el esquema mostrado en la Figura 3.6. Básicamente en cada iteración actualiza el valor de la variable recibida como parámetro, la cual es accesible desde el hilo principal, en un periodo máximo establecido por ”comp_factor”, que marca el máximo número de iteraciones que se pueden perder, y ”sensor_rate”, que marca periodo estándar de cada hilo.

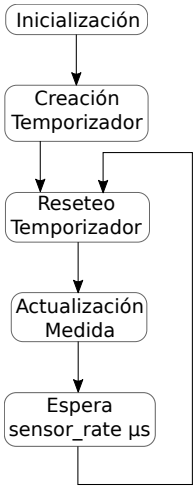

Figura 3.6 Esquema de funcionamiento hilo dedicado a leer los sensores.

Aquí se puede observar la creación de los temporizadores de sobretiempo y espera de cada uno de los hilos de los sensores. Como se puede observar esta es bastante similar a la del hilo principal.

Código 3.16 Creación temporizador de los sensores.

```
[...]
 /************** Se crea el ''watchdog'' del hilo de la Emisora ****************/
 /* Establecimiento del manejador del temporizador*/
 sa. sa_flags = SA_SIGINFO;
 sa. sa_sigaction = timer_handler ;
 sigemptyset(&sa.sa_mask);
 sigaction (SIGRTMIN+N, &sa, NULL);
 /* Block timer signal temporarily */
 sigemptyset(&mask);
 sigaddset (&mask, SIGRTMIN+N);
sigprocmask(SIG_SETMASK, &mask, NULL);
 /*Creación del temporizador*/
 sev. sigev_notify = SIGEV_SIGNAL;
 sev. sigev_signo = SIGRTMIN+N;
 sev. sigev_value . sival_ptr = &timerid;
```

```
    timer_create (CLOCK_REALTIME, &sev, &timerid);
    /* Establecimiento de los valores del temporizador */
    its . it_value . tv_sec = 0;
    its . it_value . tv_nsec = sensor_rate *comp_factor*1000;
    its . it_interval . tv_sec = 0;
    its . it_interval . tv_nsec = 0;
  [...]
```

La lectura de cada sensor es diferente en el sentido de que cada uno utiliza funciones y librerías propias, todas ellas proporcionadas por el fabricante de la Placa Navio2, Emlid. Aunque dado que la mayoría de los sensores utilizan el protocolo I2C todas las funciones se dedican básicamente a escribir y leer diferentes registros.

A continuación se muestra un ejemplo de como se leen los datos de la IMU LSM9DS1.

Código 3.17 Bucle hilo sensor(IMU LSM9DS1).

```
[...]
  while (! lsm_exit ) {
     timer_settime ( timerid , 0, &its , NULL);
     /*Cálculo del tiempo de espera*/
     clock_gettime (CLOCK_REALTIME,&slp);
     if (( slp . tv_nsec+lsm_rate*1000)<(1000000000))
        slp . tv_nsec+=lsm_rate*1000;
     else
     {
        slp . tv_sec+=1;
        slp . tv_nsec=slp . tv_nsec+lsm_rate*1000-1000000000;
     }

     /*Lectura de la medida*/
     lsm9ds1_imu->_lsm9ds1_imu.update();
     lsm9ds1_imu->_lsm9ds1_imu.read_accelerometer(&lsm9ds1_imu->_shmmsg._ax, &lsm9ds1_imu->_shmmsg._ay, &
       lsm9ds1_imu->_shmmsg._az);
     lsm9ds1_imu->_lsm9ds1_imu.read_gyroscope(&lsm9ds1_imu->_shmmsg._gx, &lsm9ds1_imu->_shmmsg._gy, &
       lsm9ds1_imu->_shmmsg._gz);
     lsm9ds1_imu->_lsm9ds1_imu.read_magnetometer(&lsm9ds1_imu->_shmmsg._mx, &lsm9ds1_imu->_shmmsg._my, &
       lsm9ds1_imu->_shmmsg._mz);
     lsm9ds1_imu->_shmmsg._temperature = lsm9ds1_imu->_lsm9ds1_imu.read_temperature();
     /*Espera para la repetici ón*/
     clock_nanosleep(CLOCK_REALTIME,TIMER_ABSTIME,&slp,NULL);
  } // Fin while(1)
[...]
```

Esquema de funcionamiento hilo de escritura en los motores

Este hilo se dedica exclusivamente en recibir la señal de control enviada por MATLAB® modificar la velocidad de los motores en función del valor recibido. Para esta comunicación se utiliza un *socket* UDP para cada proceso, dado que esto puede llegar a ocasionar que un mismo motor reciba distinta señal de PWM se ha incluido en este hilo un volcado a fichero de los valores que se escriben en los motores para poder realizar un análisis a posteriori en MATLAB® .

Por estos motivos su esquema de funcionamiento difiere un poco de la de los demás hilos dedicados a controlar el resto de sensores.

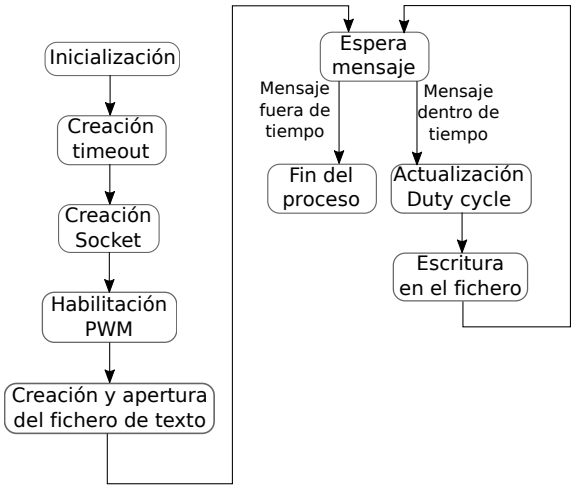

Figura 3.7 Esquema de funcionamiento hilo dedicado a controlar los motores.

En el siguiente fragmento se detalla la inicialización y declaración de todas las variables utilizadas para la recepción de los datos de control.

Código 3.18 Inicialización hilo PWM.

```
[...]
/* Declaración e inicializaci ón de variables */
    rcoutput_str * rco = ( rcoutput_str *)rcout;
    float pwm_out[6];
    int i;
    /* Variables para la declaraci ón del Socket de recepci ón*/
    struct sockaddr_in si_me, si_other;
    int s = sizeof( si_other );
    int recv_len;
    socklen_t slen;
    /* Declaración de la estructura de recepcion del valor de PWM*/
    RCOut_msg PWM_msg;
    /* Establecimiento del temporizador de recepci ón*/
    struct timeval tv;
    tv.tv_sec = 0;
    tv.tv_usec = PWM_rate*comp_factor;
[...]
```

A continuación se muestra un fragmento de código done se declara el *socket* de recepcion de datos desde SIMULINK® así como la condición de sobre tiempo del mismo.

Código 3.19 Creación *socket*.

```
[...]
    /*Creación del socket de recepci ón*/
    if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    {
        udp_die("socket_PWM");
    }
    /*Reserva de memoria*/
    memset((char *) &si_me, 0, sizeof(si_me));
    /*Asignación del puerto a utilizar */
    si_me.sin_family = AF_INET;
    si_me.sin_port = htons(PORT_PWM+id);
    si_me.sin_addr.s_addr = htonl(INADDR_ANY);
    /* Establecimiento del puerto de escucha UDP*/
    if( bind(s , ( struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
    {
        udp_die("bind_PWM");
    }
    /* Establecimiento del tiempo de espera máxima para recibir un dato*/
```

```
if ( setsockopt (s , SOL_SOCKET, SO_RCVTIMEO,&tv,sizeof(tv)) < 0)
{
    udp_die("PWM timeout setting");
}
[...]
```

En el siguiente fragmento se muestra la habilitación de los canales PWM a utilizar así como la del fichero donde se mostrarán los datos que cada proceso escribe en los motores para su posterior análisis.

Código 3.20 Habilitación PWM y apertura del fichero.

```
/* Inicializaci ón de los 6 motores a utilizar */
for(i = 0;i < 6; i++)
{
  rco->_pwm.initialize(i);
  rco->_pwm.set_frequency(i, 400);
  rco->_pwm.enable(i);
}

/* Variables para la escritura en fichero de los valores recibidos */
FILE *fp;
char filename[10];
 sprintf (filename,"f%d.txt",id);
fp = fopen ( filename, "w" );
/* Inicializaci ón del valor tiempo de cada iteracion para escribir en fichero */
 struct timespec tiempo={0,0};
/*Comentar hasta aquí si no se quiere escribir el valor en un fichero */
```

Por último se muestra la recepción de la señal de control de cada motor, con su posterior saturación de PWM y la escritura en el fichero declarado anteriormente del dato de cada motor y el tiempo en el que se realiza.

Código 3.21 Bucle hilo de escritura PWM.

```
    if (( recv_len = recvfrom(s, &PWM_msg, sizeof(RCOut_msg), 0, (struct sockaddr *) &si_other, &slen)) < 0)
    {
        udp_die("recvPWM()");
    }
    /*Lectura del valor de salida recibido para cada motor*/
    pwm_out[0] = PWM_msg.PWM._channel1;
    pwm_out[1] = PWM_msg.PWM._channel2;
    pwm_out[2] = PWM_msg.PWM._channel3;
    pwm_out[3] = PWM_msg.PWM._channel4;
    pwm_out[4] = PWM_msg.PWM._channel5;
    pwm_out[5] = PWM_msg.PWM._channel6;

    /* Establecimiento del ''Duty Cycle'' de cada motor*/
    for(i = 0;i < 6; i++)
    {
       fprintf (fp, "%f\t",pwm_out[i]); // Escritura en el fichero del valor
       /*Comentar línea anterior si no se quiere escibir en fichero */

       /* Saturaci ón del Duty Cycle*/
       if (pwm_out[i] < SERVO_MIN){
         rco->_pwm.set_duty_cycle(i, SERVO_MIN);
       }
       else
       {
         if (pwm_out[i] > SERVO_MAX)
           rco->_pwm.set_duty_cycle(i, SERVO_MIN);
         else
           rco->_pwm.set_duty_cycle(i, pwm_out[i]);
       }
    }
    /*Si no se desea escribir en ficheros comentar las dos siguientes líneas*/
    clock_gettime(CLOCK_REALTIME,&tiempo);//lectura del tiempo para representación de datos
     fprintf (fp, "%d.%d\n",tiempo.tv_sec,tiempo.tv_nsec); // escritura en fichero del tiempo
} // Fin while(1)
```

Análisis de las señales escritas en los ficheros

Como se ha comentado al principio de esta sección el volcado a ficheros se realiza con el objetivo de analizar las señales que recibe el autopiloto desde MATLAB® como señal de control.

Para el análisis de estos datos se han importado los ficheros a MATLAB® con el objetivo de reconstruir las señales de control recibidas por los motores, para ello se ha enviado desde SIMULINK® vía UDP dos tipos de señales, por un lado una señal continua modificada manualmente mediante un *slider*, Figura 3.8, y una señal senoidal, Figura 3.11 con el objetivo de comparar como influye la velocidad con la que cambia la señal y lo bruscamente que lo hace.

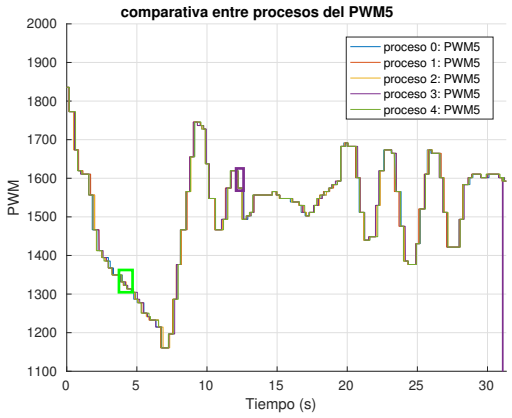

Figura 3.8 Señales de control recibidas por la *Raspberry Pi* , *slider*.

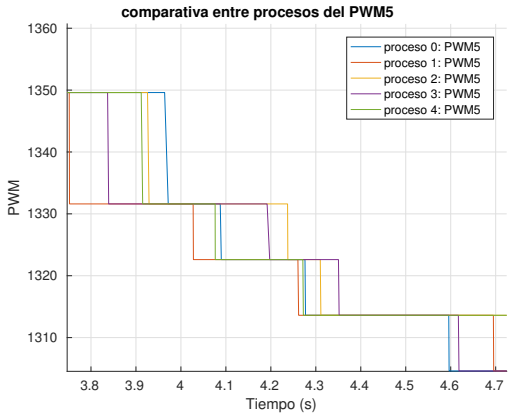

Figura 3.9 Detalle sobre el cuadrado verde.

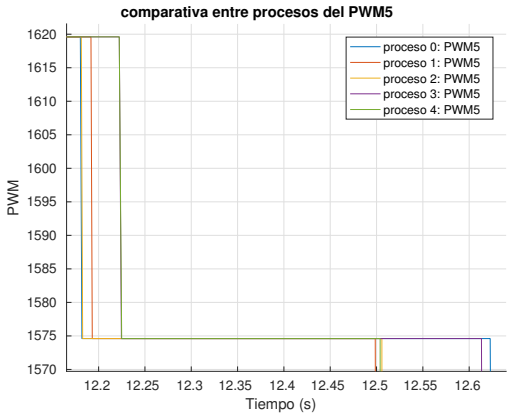

Figura 3.10 Detalle sobre el cuadrado violeta.

A vista de los resultados obtenidos con el *slider* se observa como cada proceso, al no estar sincronizados de ninguna forma, escriben a tiempos distintos. Sin embargo, se ve como con un cambio lento simplemente se produce algún retraso entre un proceso y otro lo que puede provocar que el primero en escribir un valor sea ignorado o sobre escrito con otro más antiguo. Sin embargo se observa como los retrasos son del orden de los 15-20ms. Lo que es más rápido que la propia dinámica de los motores.

Dado que este experimento no deja claro que ocurriría con una señal de control real y suave se ha llevado a cabo el siguiente experimento con una señal senoidal.

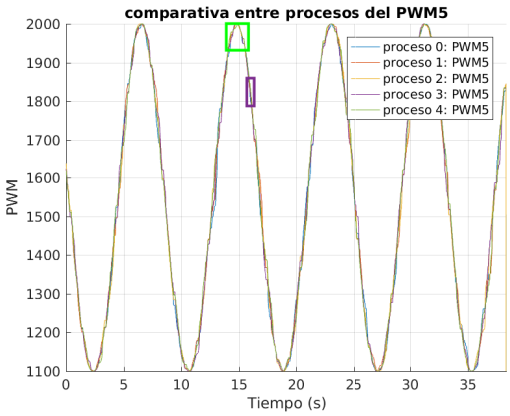

Figura 3.11 Señales de control recibidas por la *Raspberry Pi* , Señal sinusoidal.

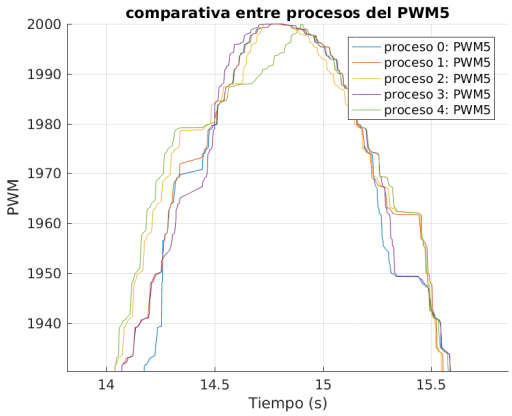

Figura 3.12 Detalle sobre el cuadrado verde.

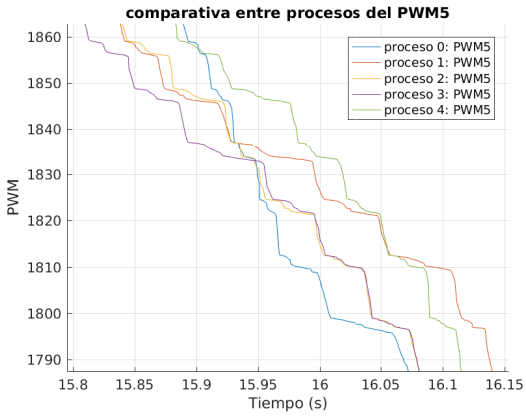

Figura 3.13 Detalle sobre el cuadrado violeta.

Con estos resultados se puede llegar a concluir que el tener tantos procesos escribiendo de forma simultánea puede inducir problemas de continuidad en la señal escrita a los PWM, ya que como se observa cada proceso escribe a ritmos muy diferentes llegando en algunos momentos a escribir valores realmente distintos aunque al igual que en el caso anterior los retrasos son del orden los 200ms.

No obstante sería interesante probar el comportamiento de los motores con esta configuración ya que según la precisión de los mismos puede ser o no un problema.

A pesar de esto existen dos posibles soluciones:

- Por un lado tener un único proceso recibiendo la señal de control.
- Sincronizar todos los procesos para que empiecen a la vez.

Estructuras enviadas

Como ya se ha comentado a lo largo de este capítulo todos los datos recogidos por los distintos hilos de los sensores son almacenado en diferentes estructuras y enviados hacia MATLAB® mediante un *socket* UDP. A continuación se comentarán los puertos utilizados para cada envío así como los tamaños y forma de los paquetes enviados.

Byte alignment

Sin embargo antes de continuar es necesario conocer el concepto de *Byte alignment*[18]. El cual se trata de un fenómeno que se da en la mayoría de compiladores de ”C” y que provoca que al crear una estructura esta se almacene, para optimización de los accesos a memoria, con un tamaño múltiplo del mayor dato de la estructura. Esto se hace especialmente relevante cuando se tiene una estructura con diferentes tipos de datos.

Por ejemplo si se tiene una estructura con datos tipo booleano, entero y doble, la estructura estará alineada en 8 bytes, por lo que cada conjunto de datos deberá ocupar múltiplos de 8 y cada dato siempre estará en el mismo registro no pudiendo dividirse en varios.

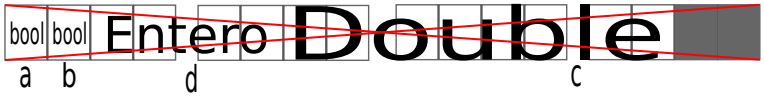

Figura 3.14 Registro de memoria estructura de forma errónea.

Digamos por ejemplo que tenemos las siguientes estructuras:

Código 3.22 Estructura ejemplo *Byte alignment*.

```
struct ejemplo1{
  bool a,b; //2bytes
  double c; //8bytes
   int d;     //4bytes
}
struct ejemplo2{
  bool a1,b1; //2bytes
   int d1;      //4bytes
  double c1;   //8bytes
}
```

Aunque ambas estructuras contengan el mismo tipo de datos por la forma en que se han definido ambas tendrán tamaños distintos, la estructura ”ejemplo1” ocupará un total de 24 bytes:

Figura 3.15 Registro de memoria estructura ejemplo1.

Esto es debido a que el dato tipo doble no puede almacenarse a continuación de los booleanos ya se tendría que empezar en medio de un registro lo cual es inaceptable. Y dado que el mayor tamaño de uno de los datos de la estructura es de 8 bytes tampoco puede empezar en el registro siguiente a los datos tipo booleano ya que esto implicaría que empezara en un registro no múltiplo de 8. De forma similar ocurre con el dato tipo entero, el cual debe ocupar un registro extra para cumplir con que la estructura tenga un tamaño total múltiplo de 8 bytes.

Por otro lado la segunda estructura ocupará un total de 16 bytes ya que el alineamiento será de la siguiente forma:

Figura 3.16 Registro de memoria estructura ejemplo2.

En esta ocasión los datos tipo booleano se agrupan de forma consecutiva ya que se alinean en múltiplos de 1 y el tipo entero se agrupa en el siguiente registro y no a continuación porque se fragmentaría la información en varios registros. Por último el tipo doble se almacena en el primer registro múltiplo de 8.

Datos de actitud

Por un lado se envían todos los datos que se utilizarán en MATLAB® para la estimación de la actitud del UAV, estos datos son:

- Datos de temperatura, aceleración, velocidad angular y campo magnético medidos por la IMU MPU9250. En total 10 datos de tipo ”Float” que corresponden a 4bytes cada uno, es decir 40 bytes en total.

- Datos de un estimador proporcionado por Emlid que nos proporciona la actitud en quaterniones. 4 datos de tipo ”Float” que corresponden a 4 bytes, en total 16 bytes, esta estimación no se utiliza ya que está pensada para aviones y movimientos lentos lo que la hace inútil para su uso en multirotores.
- Datos de aceleración, velocidad angular y campo magnético medidos por la IMU LSM9DS1. Estructura identica a la otra IMU, 40 bytes en total.
- Diferencia de tiempo entre medidas consecutivas. Un único dato tipo ”Float”.
- Dos banderas tipo booleano que indica un fallo en algunas de los dos hilos que miden las IMUs. En total 4 bytes debido al *byte alignment* ya que cada booleano tiene tamaño 1 byte.

Estos datos se almacenan en una única estructura, de tamaño total 100 bytes, que se envía por el puerto $8081 + id_{proceso} * 3$.

Datos de la emisora

Por otro lado se envían los datos recibidos desde la emisora y leídos por la placa Navio2.

- Datos de cada uno de los canales de la emisora, en total 14 datos de tipo ”short”, 2 bytes cada uno, 28 en total.
- Una bandera que indica si se ha producido un error en el hilo que realiza la lectura. Un boleano que junto a *byte alignment* de 2 ocupa un total de 2 bytes.

Esta única estructura de 30 bytes se envía por el puerto $8082 + id_{proceso} * 3$

Datos de posición

Por último se envían los datos concernientes a la estimación de la posición, en este caso:

- Datos recibidos de la ”TotalStation”, 4 datos tipo doble con un tamaño total de 32 bytes. Estos datos no se utilizan para ninguna estimación debido a que no se dispone de sensor.
- Posición global recibida del GPS. Medida de tiempo del satélite, datos de longitud y latitud, Altura sobre el elipsoide y el nivel del mar así como data de precisión horizontal y vertical. En total 7 datos tipo doble con tamaño de 56 bytes entre todos.
- Medida de presión y temperatura proporcionada por el barómetro para una estimación a posteriori de la altura. En total dos datos tipo ”float”, 8 bytes.
- Altura sobre el nivel del suelo medida por el Lidar, un dato tipo ”float”, 4 bytes. No se utiliza ya que no se dispone de sensor.
- Por último como en los casos anteriores una bandera por cada hilo de lectura que indica si su respectivo hilo ha sufrido algún problema. En total 4 bytes.

En total se envían 104 bytes por el puerto $8083 + id_{proceso} * 3$

Recepción y envio de datos en MATLAB®

Si quieres ir rápido ve solo. Si quieres llegar lejos ve acompañado

Proverbio Africano

En este capitulo se expondrán los esquemas utilizados en SIMULINK® junto a las funciones implementadas para la correcta recepción y análisis de datos recibidos desde la *Raspberry Pi* . Así como el esquema de envío desde SIMULINK® .

Recepción de datos

Para la recepción de los datos enviados desde la *Raspberry Pi* pi con la lectura de los sensores de Navio se ha utilizado el bloque de recepción UDP de SIMULINK® el cual se ha combinado con otros dos bloque en serie, *byte pack* y *byte unpack* que se encargan de crear un único paquete de todos los datos recibidos por el bloque de recepción en un único paquete que después será separado en cada uno de los tipos de datos que se esperan recibir.

Este esquema es necesario ya que mediante el bloque de recepción solo se puede recibir un tipo de dato en concreto por cada puerto y dado que en cada envío desde la *Raspberry Pi* hay como mínimo 2 tipos de datos distintos los datos distintos de tipo seleccionado no se podrán leer de forma correcta.

Por ejemplo si enviamos un entero y un booleano y seleccionamos en matlab que vamos a recibir un entero el booleano se interpretará como un entero por lo que de los 4 bits solo el primero tendrá información útil el resto será irrelevante pero hará que el dato sea un numero del orden de $\pm 2^{12}$. Si por otro lado especificamos que esperamos recibir un booleanos el entero se dividirá en 4 booleanos sin ningún tipo de coherencia.

Configuración bloque *UDP receive*

La forma más fácil de configurar este bloque, conociendo el total de bytes que se esperan recibir, es indicando que esperas recibir tantos datos tipo bool como bytes. Además es necesario indicar el puerto y crear un buffer de un paquete ya que si no el bloque esperá recibir N paquetes antes de expulsar uno, originando un severo retraso. Esta es la configuración que tenemos en la Figura 4.1.

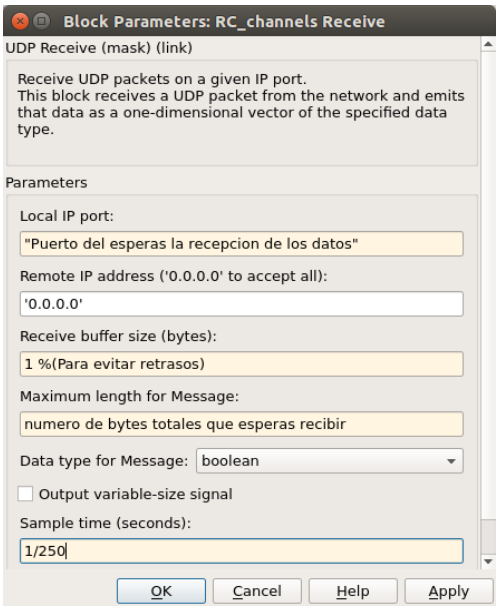

Figura 4.1 Configuración bloque *UDP receive*.

Otra forma de configurar el bloque es eligiendo cualquier otro tipo de dato que se espera recibir teniendo en cuenta que se elija un numero que sea el primer múltiplo del número de bytes del tipo de dato elegido mayor que el número total de bytes enviados.

Configuración bloques *byte pack/unpack*

La configuración del bloque de *byte pack* es sencilla simplemente debemos seleccionar el mismo tipo de dato que elegimos en el bloque *UDP receive* comentado anteriormente así como el tipo de alineamiento de byte que tenemos en los datos enviados. En la Figura 4.2 se observa el bloque.

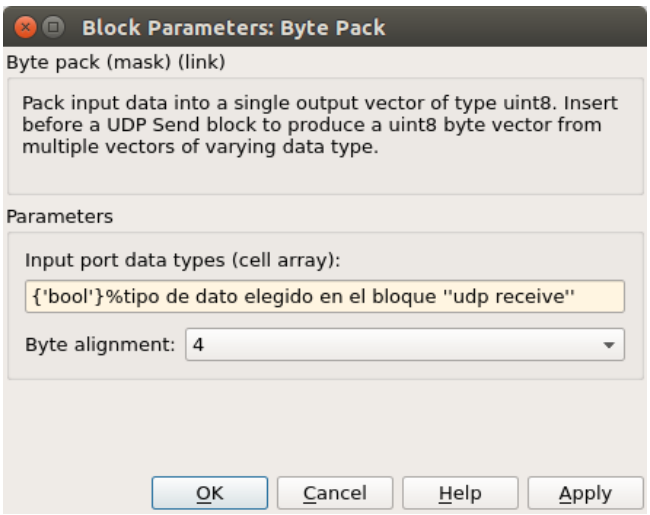

Figura 4.2 Configuración bloque *byte pack*.

Por otro lado en el bloque *byte unpack* donde realmente separamos los datos que hemos enviado desde la *Raspberry Pi* , como se ve en la Figura 4.3 en este bloque se debe seleccionar el número de datos de cada tipo que esperamos recibir así como el tipo de alineamiento de byte que tenemos.

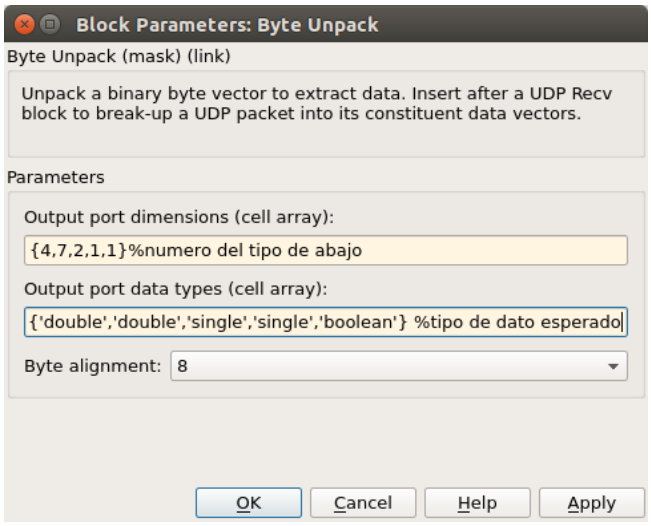

Figura 4.3 Configuración bloque *byte unpack*.

Esquema completo

En la Figura 4.4 se puede observar el esquema completo para la recepción de los datos desde la *Raspberry Pi* . Cabe destacar que se ha utilizado un *zero-order hold* para mantener el valor de los datos recibidos, además se ha realizado un enmascarado de este esquema para poder seleccionar el puerto a utilizar de forma que se pueda seleccionar desde fuera el proceso a leer de forma mucho mas rápida y sencilla. En la Figura 4.5 se puede apreciar la configuración del bloque de recepción de posición en este caso solo se pueden seleccionar los puertos desde 8083 hasta 8101 de 3 en 3, que es lo que correspondería a 7 procesos cada 3 tres puertos.

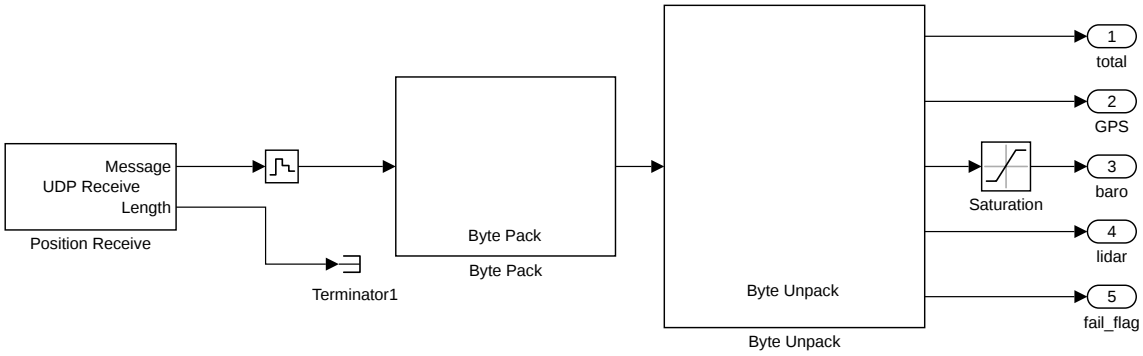

Figura 4.4 Modelo de SIMULINK® completo.

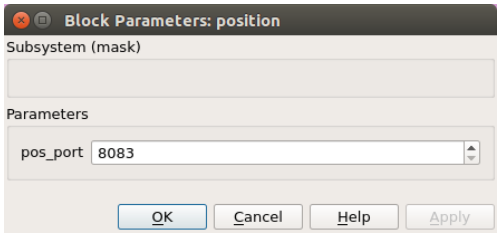

Figura 4.5 Configuración mascara de recepción, posición.

Envío de datos

Para el envío de datos la cosa se simplifica ya que en esta ocasión solamente necesitamos enviar un tipo de dato, *single*. Para ello utilizamos el bloque de SIMULINK® *UDP send* que recibe la dirección IP a la que

queremos enviar los datos y el puerto que se desea utilizar, Figura 4.4.

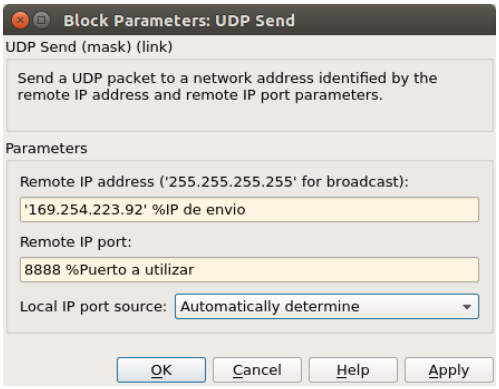

Figura 4.6 Configuración bloque *UDP send*.

Sin embargo, si se deseara enviar datos de tipos diferentes sería necesario realizar un *byte pack* para cada tipo y multiplexarlos entes del envío en el orden correcto en el que esperamos recibirlos en la *Raspberry Pi* .

Análisis e interpretación de los datos recibidos

Una vez recibidos los datos desde la *Raspberry Pi* en MATLAB® ya pueden ser analizados de forma sencilla. En primer lugar vamos a comparar los datos recibidos desde los diferentes procesos corriendo en paralelo en la *Raspberry Pi* . Para ello vamos a realizar pruebas con distintos niveles de prioridad para cada proceso.

Las distintas configuraciones de prioridad con las que se ha probado son las siguientes, siendo 20 la prioridad más baja y 0 la prioridad máxima posible, al nivel de los procesos del Sistema Operativo:

Tabla 4.1 Diferentes prioridades utilizadas para los procesos de medida.

	Prioridad 20	**Prioridad 15**	**Prioridad 0**
Configuración 1	Todos	-	-
Configuración 2	Todos menos id=0	id=0	-
Configuración 3	Todos menos id=0	-	id=0
Configuración 4	-	-	Todos

Todos los procesos con 20 de prioridad

En esta primera configuración se tratan a todos los procesos con un nivel de prioridad estándar, es decir, por debajo del resto de procesos propios del sistema operativo y al nivel de los procesos del ususario.

A continuación se va a analizar la medida de aceleración realizada por ambas IMUs.

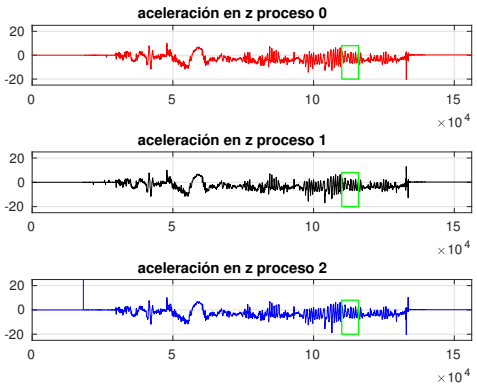

Figura 4.7 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos con prioridad 20 (imu LSM5611).

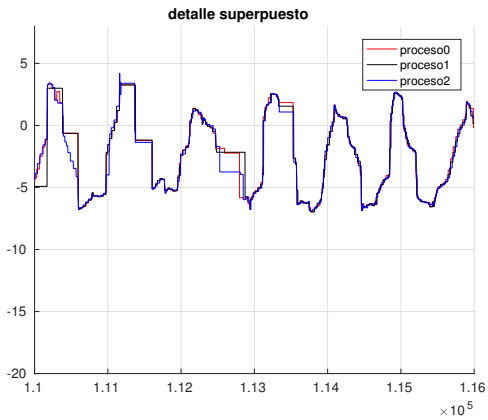

Figura 4.8 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu LSM5611).

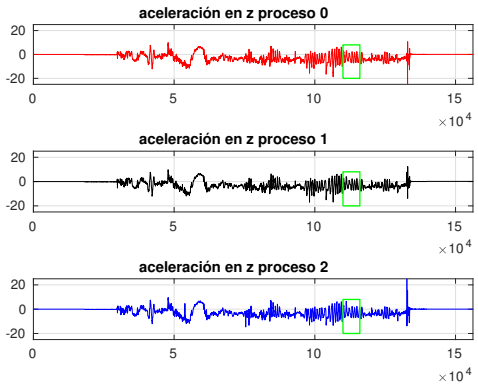

Figura 4.9 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos con prioridad 20 (imu MPU9250).

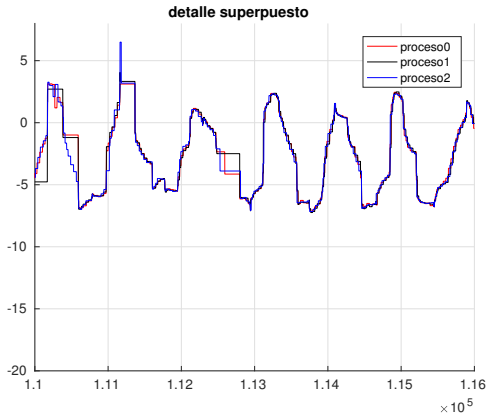

Figura 4.10 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu MPU9250).

La primera observación a destacar es que en todos los procesos no se da la misma medida de forma exacta, es decir al no estar sincronizados no dan una medida en el mismo tiempo aunque si que está dentro unos margenes bastante buenos para poder realizar algún tipo de filtrado o fusión.

Por otro lado se ve como en algunas ocasiones los procesos sufren algún tipo de problema lo que provoca cierto retraso en la llegada de una nueva medida.

Para tratar de solucionar estos problemas se proponen distintas configuraciones de prioridades de los procesos que se mostrarán a continuación a fin de buscar la mejor configuración de prioridades posibles.

Todos los procesos con 20 de prioridad y uno con 15

Debido a los problemas descubiertos en la configuración de prioridades por defecto y que han sido comentados en el apartado anterior, se decidió probar a bajar la prioridad de uno de los procesos, en concreto al proceso 0, para darle preferencia sobre el resto aunque dejándolo por debajo de los procesos propios del sistema operativo.

A continuación se muestran algunas gráficas en las que se pude comparar el comportamiento de los datos enviados por cada proceso:

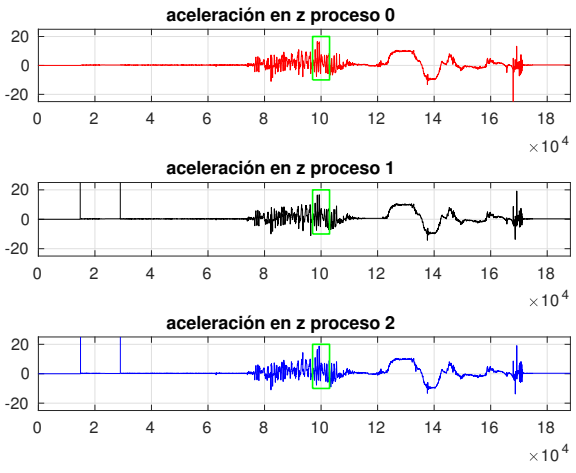

Figura 4.11 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos 1 con prioridad 15 (imu LSM5611).

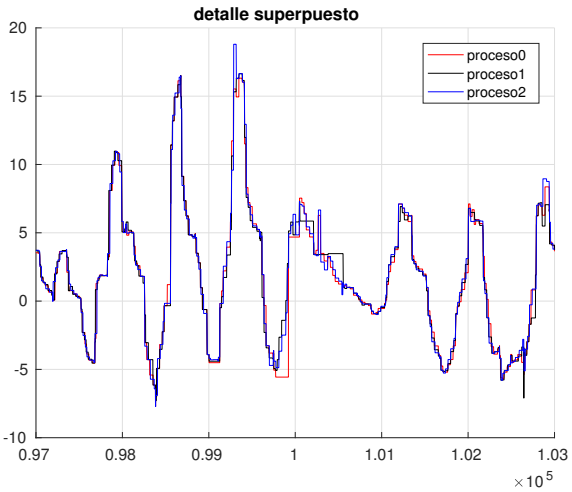

Figura 4.12 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu LSM5611).

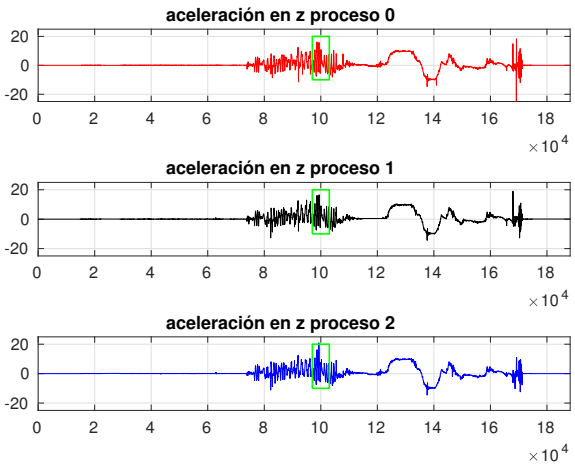

Figura 4.13 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos 1 con prioridad 15 (imu MPU9250).

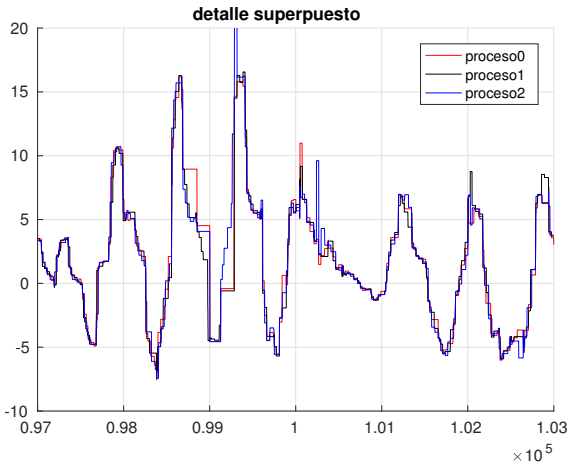

Figura 4.14 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu MPU9250).

En estas figuras se aprecia un comportamiento algo mejor que en el caso anterior sin embargo el proceso con mayor prioridad (proceso 0) no tiene una mejoría apreciable frente al resto, de hecho es incluso peor. Por este motivo se a obviado el estudio de subir la prioridad de todos los procesos a este nivel.

Todos los procesos con 20 de prioridad y uno con 0

Como siguiente prueba se propone subirle la prioridad al proceso 0 al nivel del resto de procesos del sistema operativo, es decir, asignarle prioridad 0, esta es además la máxima prioridad que se le puede asignar a un proceso del usuario.

Con esta configuración los datos recibidos han sido los siguientes:

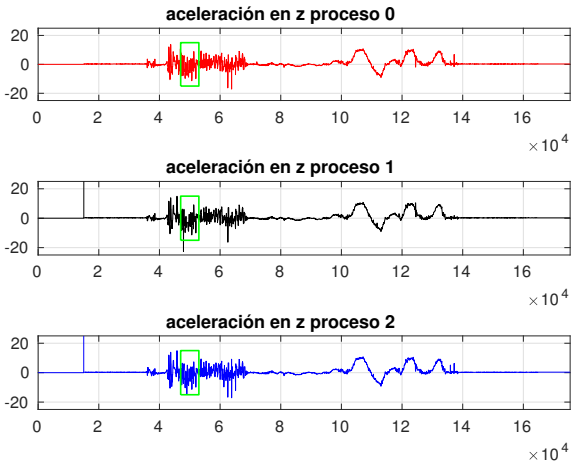

Figura 4.15 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos 1 con prioridad 0 (imu LSM5611).

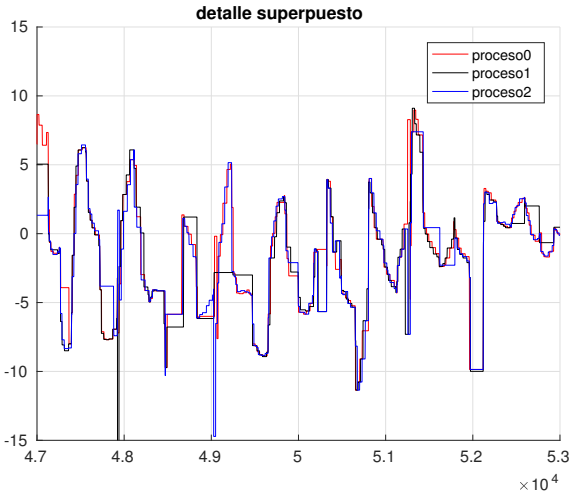

Figura 4.16 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu LSM5611).

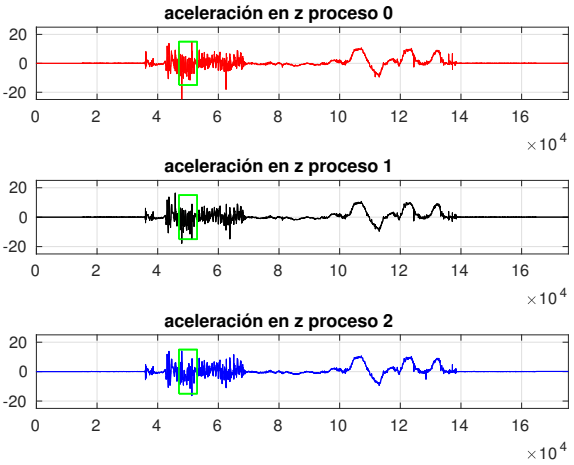

Figura 4.17 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos 1 con prioridad 0 (imu MPU9250).

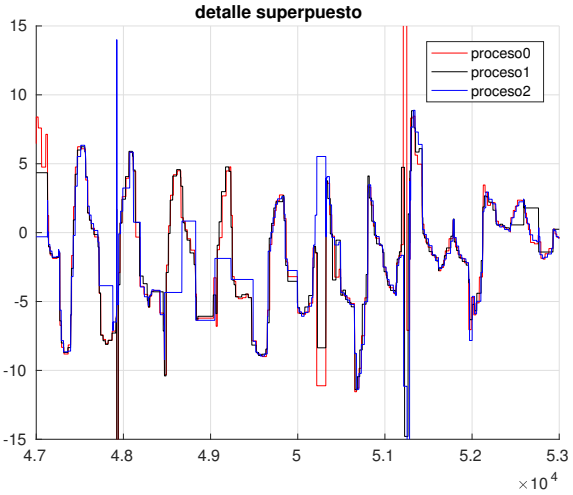

Figura 4.18 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu MPU9250).

En este caso observamos una clara mejoría en el proceso 0, vemos como no se produce ninguna perdida del mismo mientras que los otros dos si que hay algunas pérdidas esporádicas.

Todos los procesos con 0 de prioridad

Dados los resultados obtenidos en la configuración anterior se propone aumentar la prioridad de todos los procesos a fin de comprobar si de esta forma todos los procesos tienen el mismo comportamiento de forma de forma que se consiga un comportamiento mucho más estable en el mayor número de procesos posible.

A continuación se muestran los datos recibidos con todos los procesos a máxima prioridad:

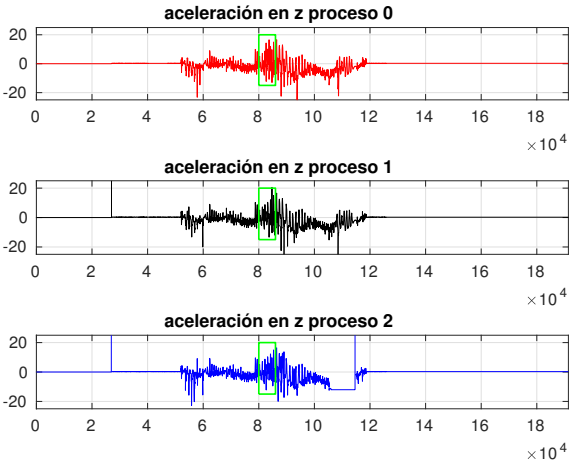

Figura 4.19 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos con prioridad 0 (imu LSM5611).

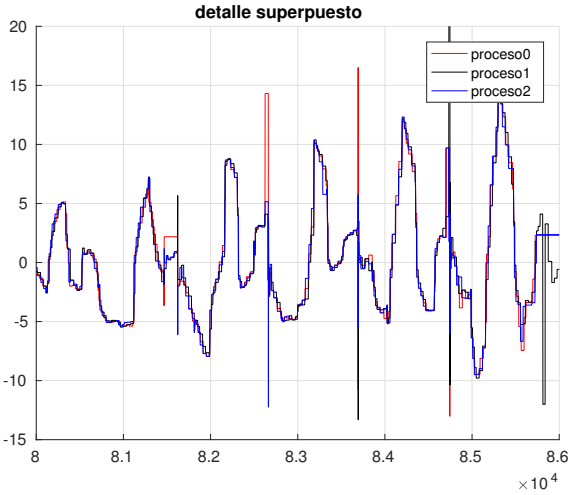

Figura 4.20 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu LSM5611).

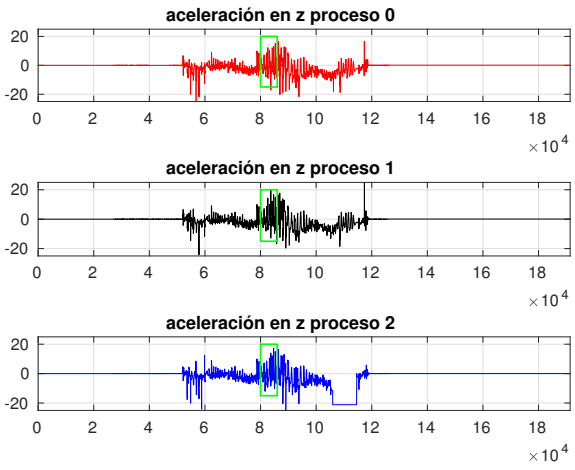

Figura 4.21 Comparación de las medidas de las aceleraciones en y de 3 procesos distintos con prioridad 0(imu MPU9250).

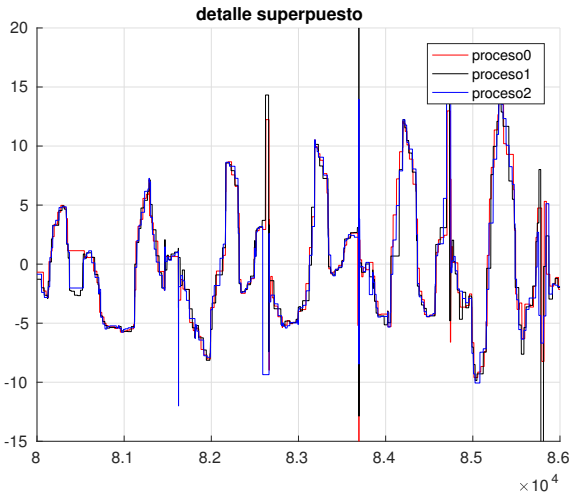

Figura 4.22 Detalle de las medidas de las aceleraciones en y sobre el cuadrado (imu MPU9250).

En base a los resultados mostrados se ha determinado que esta es la configuración ideal para implementar ya que aunque se producen alguna que otra pérdida de más respecto a solo tener un único proceso a máxima prioridad en este caso la mayoría de procesos mantienen su comportamiento lo que permitiría descartar la medida del proceso que falla mediante criterio de votación o mediante fusión sensorial.

Esquema completo de SIMULINK® implementado

A continuación, en la Figura 4.23 se muestra el esquema completo implementado en SIMULINK® donde se puede observar los 3 subsistemas encargados de la recepción de los datos en bruto desde la *Raspberry Pi* , así como los correspondientes al EKF y *motor mixer* que se comentarán en el próximo capítulo.

Todos los subsistemas encargados de la recepción, son similares al ya mostrado anteriormente en este capítulo y utilizan la misma metodología al igual que el bloque de envío que se encuentra a continuación del *motor mixer*.

Por otro lado tanto el *motor mixer* como el EKF contienen una función de MATLAB® con una pequeña adaptación de las entradas en el caso del EKF ya que estas funciones necesitan entrada de tipo doble por lo que es necesario la conversión de los datos recibidos. Además se obvia los datos de temperatura de las IMUs. Aún así estos subsistemas se mostrarán más adelante en este proyecto.(Sección 5.4, Sección 5.6)

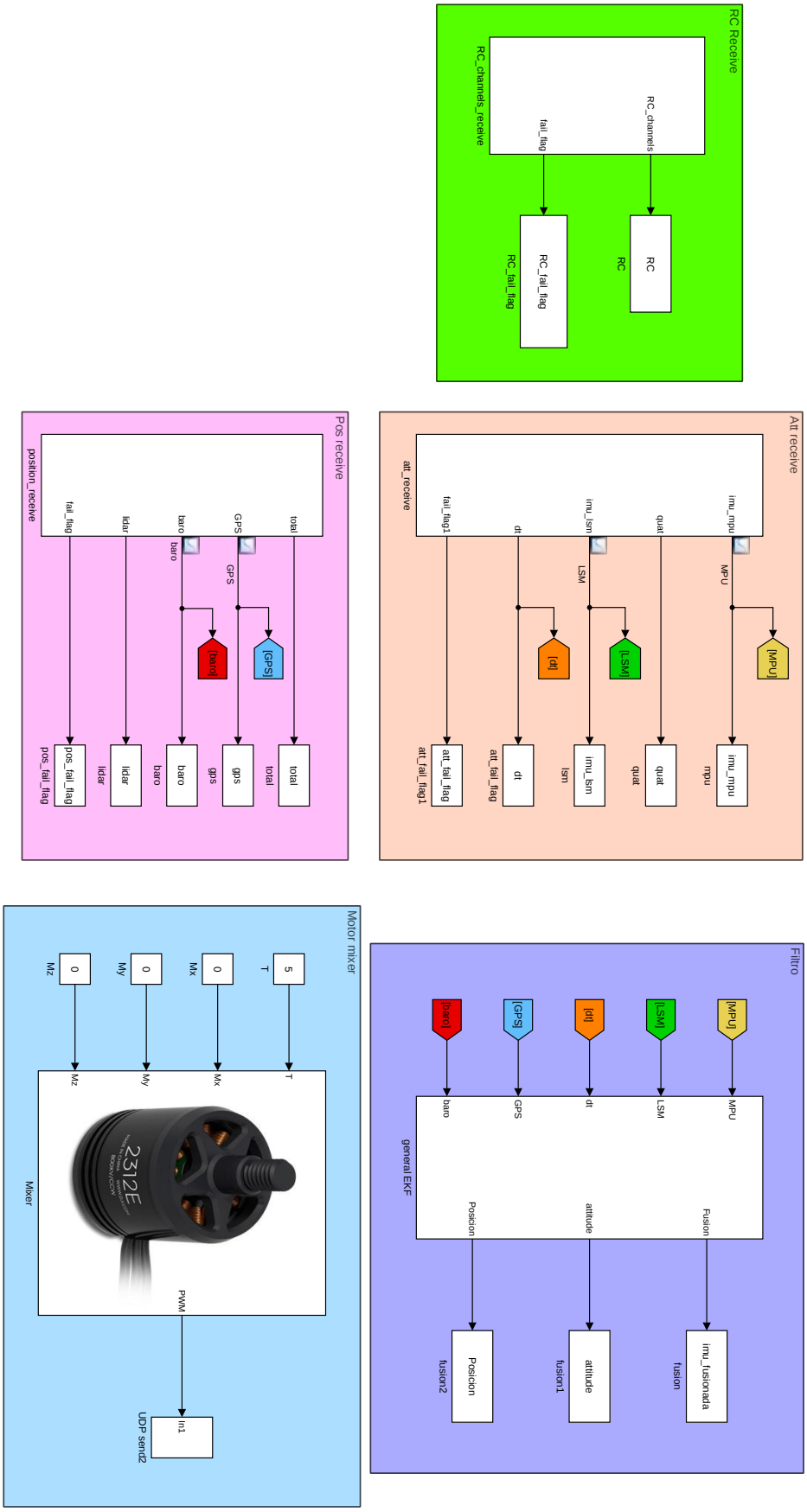

Figura 4.23 Esquema completo implementado en SIMULINK® .

Estimador y *motor mixer*

No temas renunciar a lo bueno para ir a lo grandioso

JOHN D. ROCKEFELLER

En este capitulo se expondrá todo lo relacionado con la adaptacion de los datos en bruto recibidos desde la *Raspberry Pi* . En resumen se expondrán los diferentes filtros desarrollados e implementados para las estimaciones de posición y orientación.

Filtro de Kalman para fusión de los datos procedentes de la IMU

Dado que disponemos de dos IMUs independientes lo ideal sería unificar las medidas de cada una de ellas. Para ello se propone el uso de un filtro de Kalman con el objetivo de realizar un filtro de paso bajo junto a la fusión.

Ecuaciones de Kalman generales

Como primera aproximación hacia un filtro de Kalman[19] en el que se fusionen todos los datos recibidos desde la placa Navio2 con el fin de realizar una estimación de la posición y actitud del UAV se ha realizado un filtro que unifica la medidas recibidas desde ambas IMU's con el fin de unificar ambos sensores en uno solo.

Esta misma medida podría extenderse a unificar los datos recibidos desde los distintos procesos una vez se tenga un filtro que realice la estimación de posición y actitud.

Etapa de predicción:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{5.1}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{5.2}$$

Etapa de actualización

$$K = P_k^- C^T [CP_k^- C^T + R]^{-1} \tag{5.3}$$

$$\hat{x}_k = \hat{x}_k^- + K(z - C\hat{x}_k^-) \tag{5.4}$$

$$P_k = (I - KC)P_k^- \tag{5.5}$$

Donde:
$A \in \mathbb{R}^{N\times N}$:= Matriz que contiene el modelo del sistema.
$\hat{x}_k^- \in \mathbb{R}^{N\times 1}$:= Estimación a priori de la medida en el instante actual.
$B \in \mathbb{R}^{N\times N}$:= Matriz que contiene el modelo de la entrada del sistema.
$u_{k-1} \in \mathbb{R}^{N\times 1}$:= Entrada del sistema en el instante k-1.
$P_k^- \in \mathbb{R}^{N\times N}$:= Estimación a priori de la Covarianza en el instante actual.

$Q \in \mathbb{R}^{N\times N}$:= Matriz constante que modela el ruido del proceso.
$K \in \mathbb{R}^{N\times M}$:= Ganancia de Kalman.
$C \in \mathbb{R}^{M\times N}$:= Modelo de observación que relaciona la medida con la estimación.
$R \in \mathbb{R}^{M\times M}$:= Matriz de varianza que modela el ruido de cada medida.
$z \in \mathbb{R}^{M\times 1}$:= Matriz que almacena las medidas realizadas en el instante k.
$I \in \mathbb{R}^{N\times N}$:= Matriz identidad.
$\hat{x}_k \in \mathbb{R}^{N\times 1}$:= Estimación a posteriori de la medida en el instante k.
$P_k \in \mathbb{R}^{N\times N}$:= Estimación a posteriori de la covarianza del proceso en el instante k.

Forma de las matrices utilizadas para fusionar las dos IMUs

A continuación se muestran las matrices implementadas para realizar la fusión de las medidas proporcionadas por ambas IMUs. Para ello se ha partido del filtro de kalman general comentado anteriormente, Ecuación 5.1 a Ecuación 5.5.

$$\hat{x}_k^- = \begin{pmatrix} \hat{a}_x \\ \hat{a}_y \\ \hat{a}_z \\ \hat{G}_x \\ \hat{G}_y \\ \hat{G}_z \\ \hat{M}_x \\ \hat{M}_y \\ \hat{M}_z \end{pmatrix}$$:= Estimaciones de acelerómetro, gyróscopo y magnetómetro.

$A = I_{9\times 9}$:=Como no se conoce el modelo del sistema directamente asignamos la estimación anterior a la actual.

$B = 0$, Dado que no tenemos ninguna entrada al sistema.

$Q = 1 \cdot 10^{-3} diag(2.87,3.75,4.60,0.003,0.001,0.001,57.1,44.3,37)$

Este valor de Q ha sido calculado como la covarianza del proceso dejando la *Raspberry Pi* completamente quieta. Y ha sido la matriz de inicio para el ajuste que se comentará en la próxima sección.

$R = diag(R_1 \cdots R_{18})$

IMU MPU9250	**IMU LSM9DS1**
$R_1 = 2.26 \cdot 10^{-3}$	$R_{10} = 7.62 \cdot 10^{-3}$
$R_2 = 1.99 \cdot 10^{-3}$	$R_{11} = 1.94 \cdot 10^{-3}$
$R_3 = 5.20 \cdot 10^{-3}$	$R_{12} = 2.02 \cdot 10^{-3}$
$R_4 = 7.06 \cdot 10^{-6}$	$R_{13} = 8.90 \cdot 10^{-6}$
$R_5 = 5.09 \cdot 10^{-6}$	$R_{14} = 1.15 \cdot 10^{-4}$
$R_6 = 7.11 \cdot 10^{-6}$	$R_{15} = 7.68 \cdot 10^{-6}$
$R_7 = 1.35 \cdot 10^{3}$	$R_{16} = 0.18$
$R_8 = 1.13 \cdot 10^{3}$	$R_{17} = 0.25$
$R_9 = 9.52 \cdot 10^{3}$	$R_{18} = 0.37$

Para el cálculo de la matriz R se ha dejado completamente estática los sensores durante aproximadamente 1.5 minutos y después se le ha calculado la varianza de cada uno de los sensores.

$$z = \begin{pmatrix} [Acc_{mpu}]_{3x1} \\ [Gyr_{mpu}]_{3x1} \\ [Mag_{mpu}]_{3x1} \\ [Acc_{lsm}]_{3x1} \\ [Gyr_{lsm}]_{3x1} \\ [Mag_{lsm}]_{3x1} \end{pmatrix}_{18x1}$$:= Almacenamos las medidas en primer lugar las medidas recibidas de la IMU MPU9250 después las de la IMU LSM9DS1

$C = \begin{pmatrix} I_{9\times 9} \\ I_{9\times 9} \end{pmatrix}$:= Con el objetivo de comparar la estimación con la medida de cada una de las IMUs.

Resultados obtenidos con la fusión de las IMUs

Con las matrices anteriores el resultado obtenido mediante la fusión sensorial se pude observar en las siguiente sucesión de gráficas:

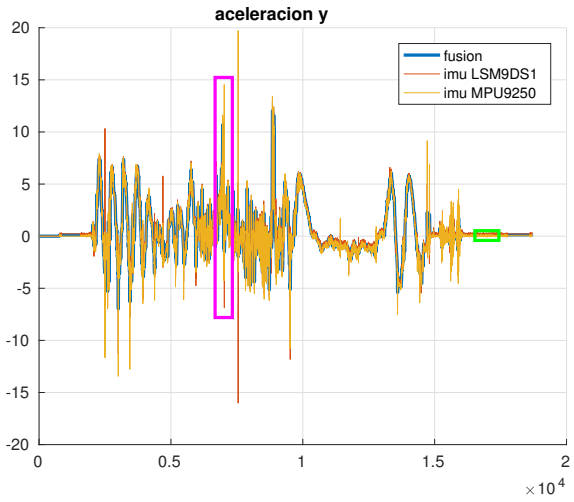

Figura 5.1 Comparación de la aceleración en y, de las dos IMUs y su fusión, Q=Covarianza.

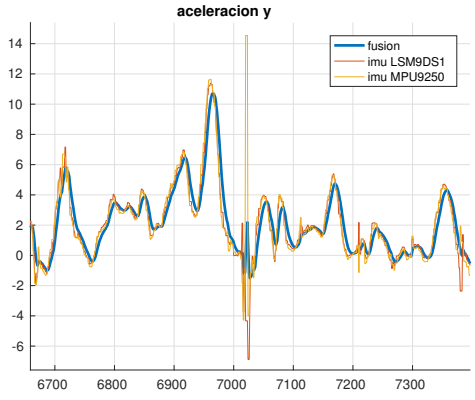

Figura 5.2 Detalle de las medidas de las aceleraciones en y, sobre el cuadrado violeta.

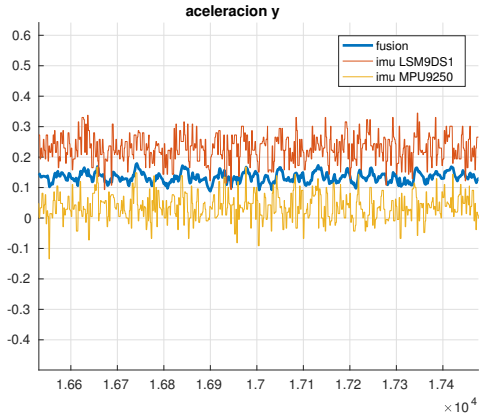

Figura 5.3 Detalle de las medidas de las aceleraciones en y, sobre el cuadrado verde.

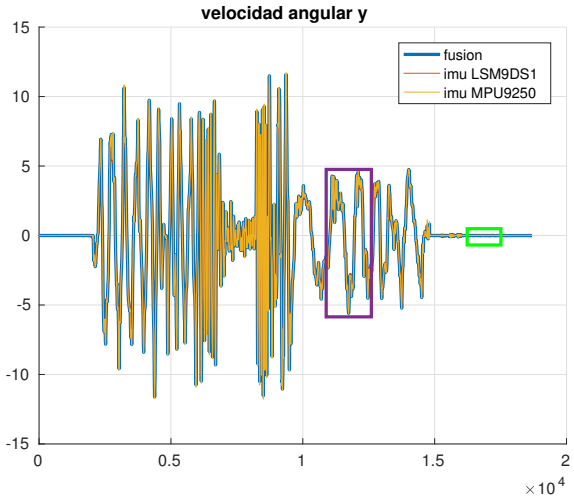

Figura 5.4 Comparación de la velocidad angular en y de las dos IMUs y, su fusión, Q=Covarianza.

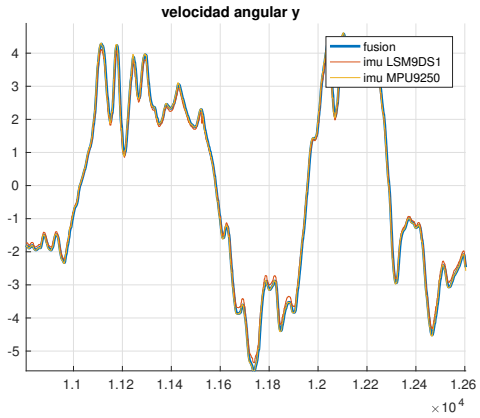

Figura 5.5 Detalle de las medidas de las velocidades angulares en y, sobre el cuadrado violeta.

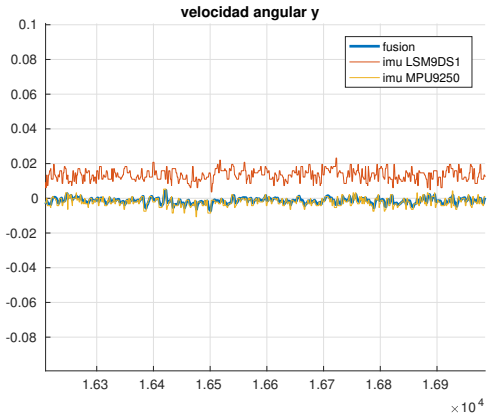

Figura 5.6 Detalle de las medidas de las velocidades angulares en y, sobre el cuadrado verde.

En base a estas gráficas se observa por un lado que de ambas medidas se consigue obtener una que ademas actúa como un pequeño filtro de paso de baja. Por otro lado se observa como en base a nivel de ruido se le da mayor credibilidad a la medida de uno de los sensores sobre el otro. Por ejemplo en la Figura 5.3 y sus detalles se observa como dado que el valor del ruido es bastante simular ($1.99 \cdot 10^{-3}$ vs $1.94 \cdot 10^{-3}$) la curva de la fusión se encuentra bastante por el centro. Sin embargo, en la Figura 5.6 se ve como la curva se adapta mucho mejor a la medida de la IMU MPU9250 ya que su nivel de ruido es 2 ordenes de magnitud menor que el de la otra IMU. ($1.15 \cdot 10^{-4}$ vs $5.09 \cdot 10^{-6}$)

Ajuste de la matriz Q para mejorar el filtrado de la señal

A continuación se ha llevado algunas variaciones de la matriz Q con el objetivo de ajustar el filtro. El resultado de estas variaciones se muestran en la Figura 5.7 y sus sucesivos detalles.

Algunos aspectos importantes de estas gráficas son que en ellas se muestran todas las simulaciones a la vez junto a la medida real de una de las dos IMUs con el objetivo de ver como afecta el valor de Q a el filtrado de la señal.

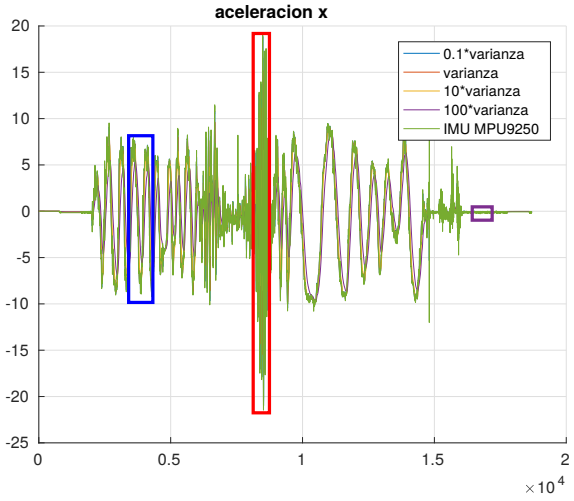

Figura 5.7 Comparación del filtrado para distintos valores de Q.

A continuación se muestran varios detalles sobre los cuadrados de la figura anterior con el objetivo de comentar diferentes movimientos y perturbaciones de la simulación realizada.

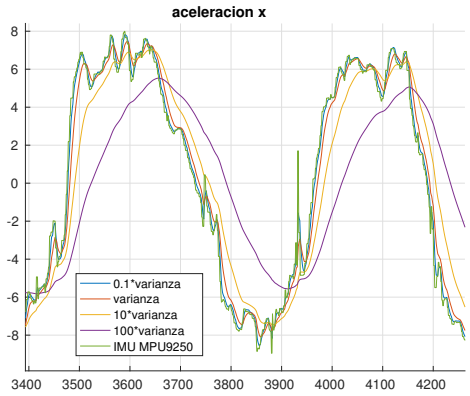

Figura 5.8 Detalle de las medidas de las aceleraciones en y, sobre el cuadrado azul.

En esta figura se observa en detalle un fragmento de la simulación con movimientos suaves. En ella se observa como a excepción de la simulación que tiene mayor Q, para el resto de valores de Q se sigue de forma razonablemente buena la señal real. Sin embargo se ve como a mayor Q mayor retraso en el seguimiento de la señal.

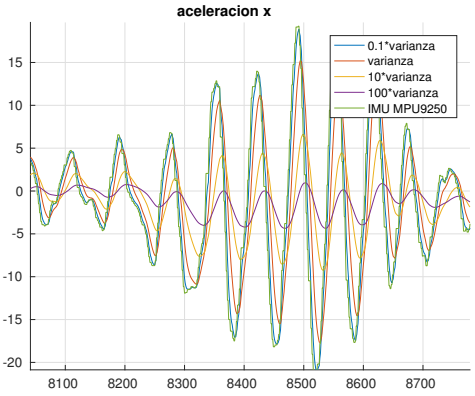

Figura 5.9 Detalle de las medidas de las aceleraciones en y, sobre el cuadrado rojo.

Aquí se muestra una zona con perturbaciones mucho mayores, en la que podemos ver como en esta ocasión la simulación amarilla con un valor de 10 veces la covarianza original no se adapta todo lo bien que nos gustaría a lquaterniona señal de forma que si nuestro sistema se sometiera a este tipo de movimiento quizás no sería adecuada, al igual que la señal violeta con 100 veces el valor de la Q original.

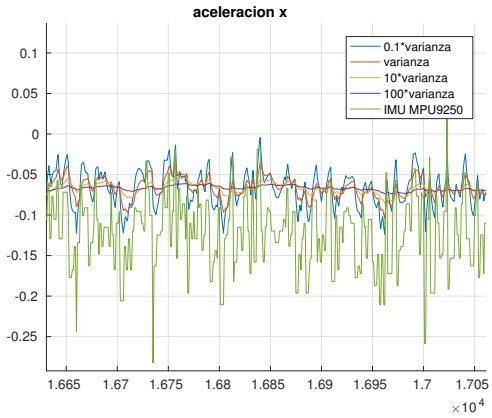

Figura 5.10 Detalle de las medidas de las aceleraciones en y, sobre el cuadrado violeta.

Por último se observa la zona en la que la *Raspberry Pi* se encuentra completamente parada. En esta ocasión dado que las señales se ajustan mejor a la real cuanto menor es el valor de la Q, vemos que la dos primeras señales tienen el mayor nivel de ruido mientras que la señal violeta apenas es afectada por el mismo.

En base a los resultados mostrados por los diferentes valores de Q vemos que la que mejor se adapta a los requisitos es la que se corresponde con la covarianza del proceso cuando lo dejamos en estático.

Filtro de Kalman para estimación de actitud

Para la estimación de la orientación del UAV se han considerado diferentes opciones, en primer lugar se ha propuesto realizar una medida directa de la posición en *roll* y *pitch* directamente a partir de las aceleraciones y de *yaw* a partir del magnetómetro y fusionarlo con la integración de la medida del giróscopo. La siguiente opción que se consideró fue realizar una integración de la medida dada por el giróscopo similar a lo realizado en [20] donde a partir de la velocidad angular se calcula la matriz de rotación y posteriormente la actitud del UAV. Por último se ha probado con el filtro de Kalman extendido propuesto en el capítulo 8 del libro[21], sin ninguna simplificación junto con la medida de actitud a partir de la IMU propuesta en la primera solución.

Estimación de la actitud de forma directa

Para esta primera aproximación de la orientación del multirotor se ha utilizado lo mismo que propone PX4 en su ”EKF_replay” el cual replica su estimador en matlab [22]. Este cálculo consiste en utilizar las aceleraciones y a partir de la medida de la gravedad deducir el angulo de inclinación. Dado que el cálculo se realiza en quaterniones posteriormente se convierte a ángulos de euler para su mejor comprensión. Esto nos proporcionaría solo la medida de *roll* y *pitch* para el cálculo del *yaw* este método no es válido para ello se utiliza el magnetómetro con el objetivo de calcular el ángulo en *yaw* respecto al norte magnético.

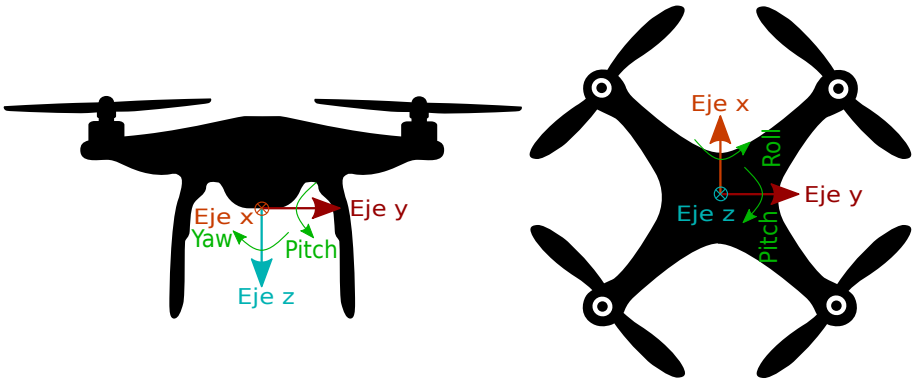

Figura 5.11 Dibujo indicativo de los ejes de *Roll*, *Pitch* y *Yaw*.

Estimación de *Roll* y *Pitch*

El cálculo de los ángulos de *roll* y *pitch* se basa en el cálculo del módulo de la aceleración en los ejes cuerpo ”x” e ”y” respecto del eje cuerpo ”z” con esto obtenemos la magnitud del giro total, primer valor de los quaterniones. A continuación si no estamos en un valor que pueda llegar a generar una indeterminación se calcula la proyección de las aceleraciones en ejes cuerpo sobre el eje ”z” inercial y la normalizamos. Con estos datos ya se puede calcula directamente el valor de la actitud en quaterniones por lo que lo siguiente será convertirlo a ángulos de euler.

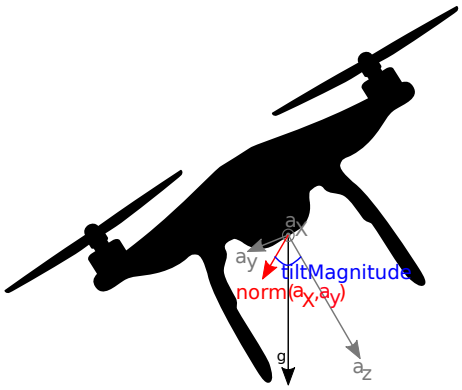

Figura 5.12 Dibujo explicativo del cálculo de *roll* y *pitch*.

Estimación del *Yaw*

Para el cálculo del ángulo de *yaw* lo primero que se realiza es una rotación respecto a los ángulos calculados anteriormente de *roll* y *pitch* para transformar las medidas del magnetómetro en ejes cuerpo a ejes inerciales para obtener una estimación de *yaw* real mediante la arcotangente entre el valor sobre el eje "y" y el eje "x".

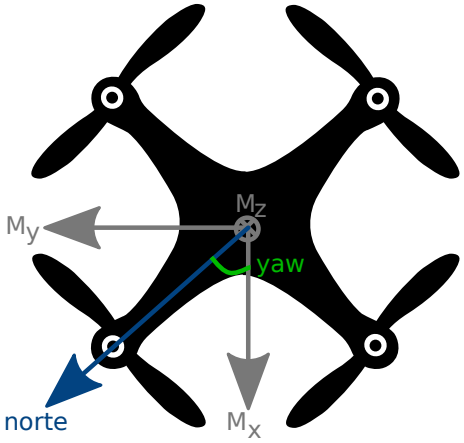

Figura 5.13 Dibujo indicativo del cálculo del *yaw*.

Cálculo en MATLAB®

En el siguiente fragmento de código se muestran el método que utiliza PX4 para la réplica de su filtro en matlab, la cual ha sido modificada para obtener además el ángulo de *yaw*. En ella se consigue calcular el *roll* y el *pitch* a partir de las aceleraciones y el *yaw* a partir del campo magnético.

Código 5.1 Cálculo de la actitud basado en PX4.

```
if( (xhat1(1)~=0 || xhat1(2)~=0 || xhat1(3)~=0) &&...
    (xhat1(7)~=0 && xhat1(8)~=0 && xhat1(9)~=0))
    tiltMagnitude = atan2(norm([xhat1(1);xhat1(2)]),xhat1(3));
    if (tiltMagnitude > 1e-3)
        tiltUnitVec = cross([xhat1(1);xhat1(2);xhat1(3)],[0;0;1]);
        tiltUnitVec = tiltUnitVec/norm([tiltUnitVec,tiltUnitVec]);
        tiltVec = tiltMagnitude*tiltUnitVec;
        quat = [cos(0.5*tiltMagnitude); tiltVec/tiltMagnitude*sin(0.5*
            tiltMagnitude)];
        quat=quat/norm(quat);
        [~, roll, pitch]=quat2angle(quat','ZYX');
    end
    mag=[xhat1(7) xhat1(8) xhat1(9)]*(eul2rotm([0 pitch roll]));
    yaw=atan2(mag(2),mag(1));
end
```

Filtro implementado para fusión y estimación directa de la actitud

A continuación se detalla el filtro utilizado para la fusión de las IMUs, idéntico al anterior, y la estimación de la actitud a partir de lo comentado previamente. Para ello simplemente se han ampliado las matrices correspondientes para los tres nuevos estados incluidos.

Etapa de predicción:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{5.6}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{5.7}$$

Etapa de actualización

$$K = P_k^- C^T [CP_k^- C^T + R]^{-1} \tag{5.8}$$

$$\hat{x}_k = \hat{x}_k^- + K(z - C\hat{x}_k^-) \tag{5.9}$$

$$P_k = (I - KC)P_k^- \tag{5.10}$$

$\hat{x}_k^- = \begin{pmatrix} \hat{a}_x \\ \hat{a}_y \\ \hat{a}_z \\ \hat{G}_x \\ \hat{G}_y \\ \hat{G}_z \\ \hat{M}_x \\ \hat{M}_y \\ \hat{M}_z \\ \hat{\phi} \\ \hat{\theta} \\ \hat{\psi} \end{pmatrix}$:= Estimaciones de acelerómetro, gyróscopo, magnetómetro y actitud.

$A := \begin{pmatrix} [I_{9x9}] & & & [\emptyset_{9x3}] \\ [\emptyset_{3x3}] & [dt \cdot I_{3x3}] & [\emptyset_{3x3}] & [I_{3x3}] \end{pmatrix}_{12x12}$:=Asignamos de forma directa la estimación anterior a la actual y en el caso de los ángulo además integramos la estimación de los giróscopos.

$B = 0$ Dado que no tenemos ninguna entrada al sistema.

$Q = 1 \cdot 10^{-3} \cdot diag(2.87, 3.75, 4.60, 0.003, 0.001, 0.001, 57.1, 44.3, 37, 14.5, 21.7, 37)$

$R = diag(R_1 \cdots R_{21})$

IMU MPU9250	**IMU LSM9DS1**	**Actitud directa**
$R_1 = 2.26 \cdot 10^{-3}$	$R_{10} = 7.62 \cdot 10^{-3}$	$R_{19} = 7.62 \cdot 10^{-3}$
$R_2 = 1.99 \cdot 10^{-3}$	$R_{11} = 1.94 \cdot 10^{-3}$	$R_{20} = 1.94 \cdot 10^{-3}$
$R_3 = 5.20 \cdot 10^{-3}$	$R_{12} = 2.02 \cdot 10^{-3}$	$R_{21} = 2.02 \cdot 10^{-3}$
$R_4 = 7.06 \cdot 10^{-6}$	$R_{13} = 8.90 \cdot 10^{-6}$	
$R_5 = 5.09 \cdot 10^{-6}$	$R_{14} = 1.15 \cdot 10^{-4}$	
$R_6 = 7.11 \cdot 10^{-6}$	$R_{15} = 7.68 \cdot 10^{-6}$	
$R_7 = 1.35 \cdot 10^{3}$	$R_{16} = 0.18$	
$R_8 = 1.13 \cdot 10^{3}$	$R_{17} = 0.25$	
$R_9 = 9.52 \cdot 10^{3}$	$R_{18} = 0.37$	

El cálculo de la matriz R y Q se ha realizado de forma similar a el filtro anterior en el que solo se fusionaba aunque se ha añadido los datos de la medida directa de actitud.

$z = \begin{pmatrix} [Acc_{mpu}]_{3x1} \\ [Gyr_{mpu}]_{3x1} \\ [Mag_{mpu}]_{3x1} \\ [Acc_{lsm}]_{3x1} \\ [Gyr_{lsm}]_{3x1} \\ [Mag_{lsm}]_{3x1} \\ Roll \\ Pitch \\ Yaw \end{pmatrix}_{21x1}$:= Almacenamos las medidas en primer lugar las medidas recibidas de la IMU MPU9250 después las de la IMU LSM9DS1, por ultimo la medida directa de la orientación según el Código 5.1

$C = \begin{pmatrix} [I_{9x9}] & [\emptyset_{9x3}] \\ [I_{9x9}] & [\emptyset_{9x3}] \\ [\emptyset_{3x9}] & [I_{3x3}] \end{pmatrix}_{12x12}$:= Con el objetivo de comparar la estimación con la medida de cada una de las IMUs y el cálculo de la orientación comentada anteriormente.

Esquema de funcionamiento de la función implementada

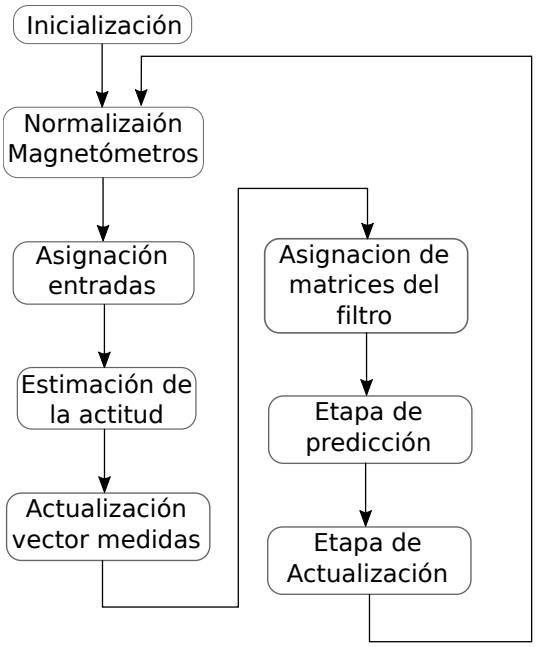

Figura 5.14 Esquema de funcionamiento de la función de estimación directa de la orientación.

Resultados estimación directa

Con el esquema de funcionamiento anterior y las matrices comentadas previamente se han obtenido los siguientes resultados.

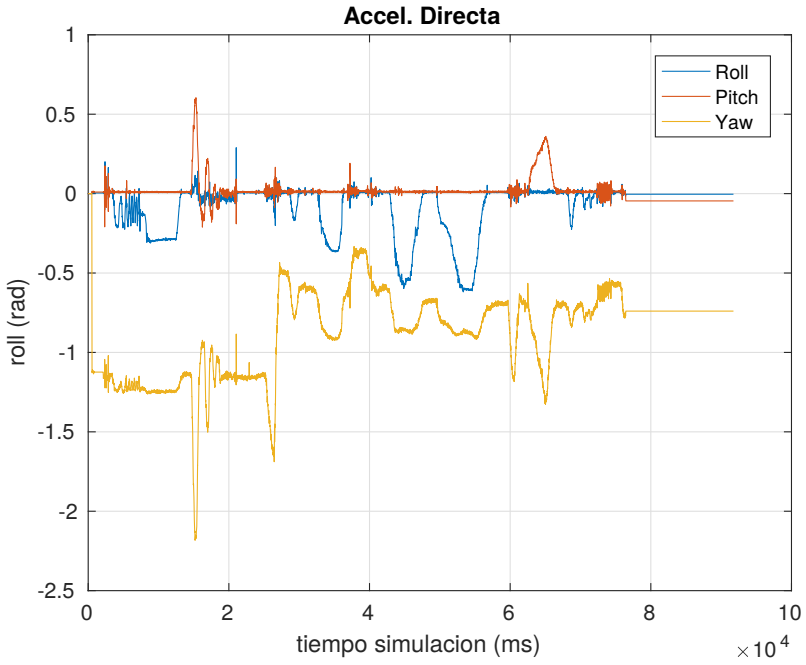

Figura 5.15 Resultados obtenidos de la estimación directa de la actitud.

En la Figura 5.15 se observa como la estimación no es del todo mala a excepción de que tiene bastante ruido a pesar de estar filtrada al limite de evitar grandes retrasos por lo que se debe buscar otra formar de

realizar la estimación. Además este método es bastante sensible a las aceleraciones lineales bruscas lo que en avión no es una gran problema pero que en un multirotor puede ocasionar grandes errores.

Estimación de la actitud basada en el artículo

Esta nueva estimación se basa en la propuesta realizada en el artículo [20] el cual propone una solución iterativa para el cálculo la orientación basado en sensores con diferentes frecuencias de medida, en su caso proponen como sensores una IMU y un VICON, el cual por si solo ya da una medida prácticamente perfecta. Las ecuaciones que ellos proponen, las cuales están en tiempo continuo son las siguientes:

Ecuaciones propuestas en el artículo

Predictor:

$$\dot{\Delta}(t) = \Delta(t)\Omega(t)_x \Delta(0) = \Delta_0 \tag{5.11}$$

$$y_i^p = \Delta(t)^T \Delta(t'_{k_i} - \tau_i) z_i(t)\, t \in [t'_{k_i}, t'_{k_i+1}) \tag{5.12}$$

Observador:

$$\dot{\hat{R}}(t) = \hat{R}(t) \left(\Omega(t) - P(t) \sum_{i=1}^{n} \left(L_i \left(\hat{y}_i(t) - y_i^P(t) \right) \right)_x \hat{y}_i(t) \right)_x \tag{5.13}$$

Donde:
$\Delta \in SO(3)$:= Estado interno del predictor.
$\Delta_0 \in SO(3)$:= Valor arbitrario como condición inicial.
Ω := Velocidad angular del UAV.
y_i^P := Salida del predictor.
z_i := Salidas anteriores del predictor.
τ_i := Retraso entre medidas.
$\dot{\hat{R}}$:= Salida del observador, matriz de rotación correspondiente en el instante t.
$\hat{R}(0) \in SO3$:= valor inicial de la observación.
P,L_i := matrices de ganancias definidas positivas.
$\hat{y}_i$:= Estimación de la iteración anterior.
El subíndice "i" indica el sensor que proporciona las medidas y el operador $(\bullet)_x$ es la matriz anti-simetrica de un vector $\mathbb{R}^3 \backslash (a)_x b = a \times b$ o lo que es lo mismo:

$$a_x = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}_x = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$$

Ecuaciones propias implementadas

Dado que nos encontramos en tiempo discreto y realmente no tenemos sensores con diferente frecuencia las ecuaciones anteriores deben ser modificadas ligeramente para garantizar convergencia. De este modo las ecuaciones realmente implementadas en MATLAB® son las siguientes:

La Ecuación 5.11 pasa a ser:

$$\Delta_k = \Delta_{k-1} + \Delta_{k-1} \Omega_x dt \tag{5.14}$$

La Ecuación 5.12 se convierte en:

$$y^P = \Delta_{k-1}^T \Delta_{k-1} att \tag{5.15}$$

Por último la Ecuación 5.13 queda de la siguiente forma:

$$\hat{R}_k = R_{k-1} + R_{k-1} \left(\Omega - P_1 \left(L(y_{k-1}^P - y_k^P)_x y_{k-1}^P \right)_x \right. \tag{5.16}$$

Además ahora se actualiza la estimación de la actitud a partir de la matriz R, obteniendo así los valores de los ángulos de Euler (*Roll*, *Pitch* y *Yaw*), **Donde:**

$$\Delta_0 = R_0 = \begin{pmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Ω := Estado correspondiente a la velocidad angular estimada previamente por la fusión de las dos IMUs con el filtro de Kalman de la Sección 5.1.

att := Medida directa de la orientación del UAV con el mismo método del apartado anterior.

y_{k-1}^{P} := Estimación actualizada con la obtención de los ángulos de Euler a partir de la matriz R.

P := Ruido del proceso asociado de la fusión de las IMUs. (Q(4:6,4:6))

L := Matriz identidad de 3x3.

Esquema de funcionamiento de la función implementada

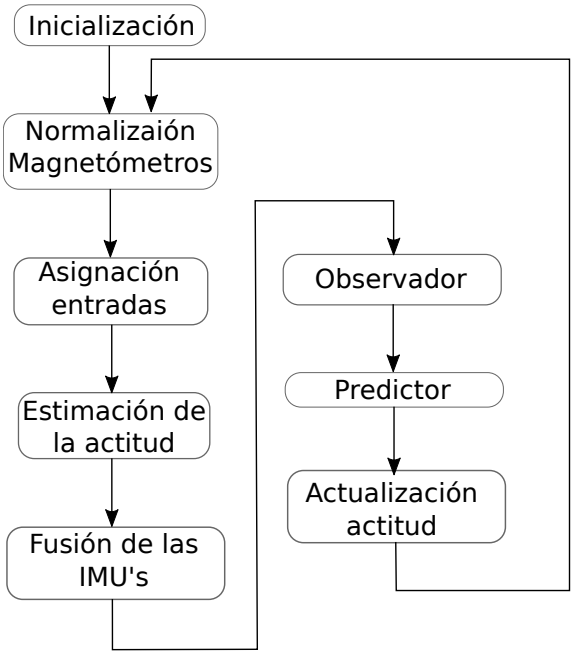

Figura 5.16 Esquema de funcionamiento de la función de estimación de la orientación basada en el artículo.

Resultados obtenidos con la estimación del artículo

Con las ecuaciones implementadas, Ecuación 5.14 a Ecuación 5.16 ya comentadas y utilizando los valores anteriores se han obtenido los siguientes resultados::

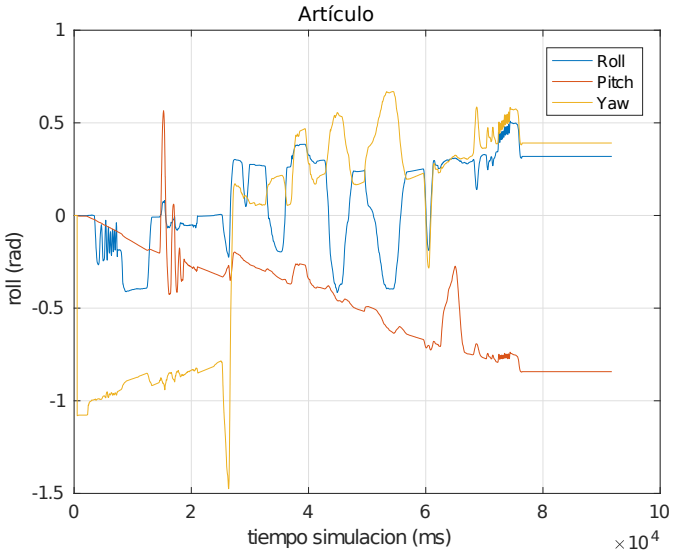

Figura 5.17 Resultados obtenidos de la estimación de la actitud basada en el artículo.

En la Figura 5.17 vemos como con este método los resultados no tienen prácticamente ruido alguno sin embargo dado que está ideado para tener como sensor un VICON, el cual, da una medida muchísimo más exacta y filtrada que los sensores de los que disponemos, este método de estimación introduce una deriva demasiado acentuada para que sea viable su uso.

Estimación de la actitud basada en el libro

Como último método para la estimación de la actitud se propone un filtro similar al que se propone en el capítulo 8 del libro "Small Unmanned Aircraft"[21] en el que se propone un Filtro de Kalman Extendido basado en el modelo de propagación de un UAV junto a un algoritmo para ejecutarlo a una frecuencia mayor que la frecuencia de llegada de nuevos datos de los sensores.

Ecuaciones propuestas en el libro

Filtro de Kalman Extendido

Mientras no llegan medidas nuevas (predictor):

$$\dot{\hat{x}} = f(\hat{x},u) \tag{5.17}$$

$$A = \frac{\partial f}{\partial x}(\hat{x},u) \tag{5.18}$$

$$\dot{P} = AP + PA^T + Q \tag{5.19}$$

Cuando llegan medidas nuevas (corrector):

$$C = \frac{\partial h}{\partial x^-}(\hat{x},u) \tag{5.20}$$

$$L = P^- C^T \left(R + CP^- C^T\right)^{-1} \tag{5.21}$$

$$\hat{x}^+ = L\left(y(t_n) - h(\hat{x}^-)\right) \tag{5.22}$$

$$P^+ = (I - LC)P^- \tag{5.23}$$

Modelo de propagación no lineal

$$\dot{\hat{\phi}} = p + q\sin(\phi)\tan(\theta) + r\cos(\phi)\tan(\theta) \tag{5.24}$$

$$\dot{\hat{\theta}} = q\cos(\phi) - r\sin(\phi) \tag{5.25}$$

Salida de los acelerómetros

$$y_{accel} = \begin{pmatrix} \dot{u} + qw - rv + g\sin(\theta) \\ \dot{v} + ru - pw - g\cos(\theta)\sin(\phi) \\ \dot{w} + pv + qu - g\cos(\theta)\cos(\phi) \end{pmatrix} \tag{5.26}$$

Donde:

$\hat{\phi}$:=Estimación del ángulo de *roll*.

$\hat{\theta}$:=Estimación del ángulo de *pitch*.

p,q,r :=Velocidades angulares respecto de los ejes x,y,z respectivamente.

u,v,w :=Velocidades lineales respecto de los ejes x,y,z respectivamente.

g := Gravedad.

A continuación en el libro se desarrollan estas expresiones para eliminar las aceleraciones y velocidades ya que en el caso que ellos utilizan dan por hecho que las aceleraciones no se conocen sin embargo en nuestro caso las aceleraciones son conocidas y proporcionadas por ambas IMUs mientras que las velocidades pueden ser obtenidas integrando las mismas. Por lo tanto las ecuaciones que se implementaran en nuestro caso particular serán estas directamente junto con la fusión de las IMUs.

Sin embargo lo que si se implementará será el algoritmo que se propone el cual solo corrige la estimación cuando llega una medida nueva de alguno de los sensores y mientras esto ocurre actualiza la estimación anterior sumandola al EKF expuesto anteriormente multiplicado por la siguiente ganancia:

Donde T_{sensor} es el periodo entre medidas del sensor y N es el numero de iteraciones entre dos muestras distintas consecutivas.

Cuando llega una nueva medida directamente ejecutamos el corrector del EKF comentado anteriormente.

Es importante mencionar que dado que para el cálculo de y_{accel} se precisa de la velocidad esta será calculada a partir de los acelerómetros e incluida como estado en el EKF.

Ecuaciones finales implementadas para la fusión y estimación de la actitud

De esta forma el filtro encargado de realizar la fusión y la estimación de actitud basado en este método será el siguiente:

Filtro de Kalman:

Cuando no hay nuevas medidas(Estimador):

$$x_k = x_{k-1} + \frac{dt_{imu}}{N} f(x_{k-1},u) \tag{5.27}$$

$$P_k = P_{k-1} + \frac{dt_{IMU}}{N}\left(AP_{k-1} + P_{k-1}A^T + Q\right) \tag{5.28}$$

Donde:

$x = \begin{pmatrix} [\hat{Acc}]_{3x1} \\ [\hat{Gyr}]_{3x1} \\ [\hat{Mag}]_{3x1} \\ [\hat{Vel}]_{3x1} \\ \hat{\phi} \\ \hat{\theta} \\ \hat{\psi} \end{pmatrix}_{15x1}$: Con $\hat{Acc}$,$\hat{Gyr}$ y $\hat{Mag}$ fusión de las IMUs y $\hat{Vel}$ fusión de las integracion de las aceleraciones de cada IMU. $\hat{\phi}$, $\hat{\theta}$, $\hat{\psi}$ estimación de la actitud del UAV.

$f(x,u) = \begin{pmatrix} \emptyset_{12x1} \\ p + q\sin(\hat{\phi}_{k-1})\tan(\hat{\theta}_{k-1}) + r\cos(\hat{\phi}_{k-1})\tan(\hat{\theta}_{k-1}) \\ q\cos(\hat{\phi}_{k-1}) - r\sin(\hat{\phi}_{k-1}) \\ 0 \end{pmatrix}_{15x1}$: La fusión la mantenemos igual que la original y corregimos principalmente la actitud mediante lo propuesto en el libro.

$A = \begin{pmatrix} [I_{12x12}] & & [\emptyset_{12x3}] & \\ [\emptyset_{1x12}] & q\cos(\hat{\phi}_{k-1})\tan(\hat{\theta}_{k-1}) - r\sin(\hat{\phi_{k-1}})tan(\hat{\theta_{k-1}}) & \frac{q\sin(\hat{\phi}_{k-1}) - r\cos(\hat{\phi}_{k-1})}{\cos^2(\hat{\theta}_{k-1})} & 0 \\ [\emptyset_{1x12}] & -q\sin(\hat{\phi}_{k-1}) - r\cos(\hat{\phi}_{k-1}) & 0 & 0 \\ [\emptyset_{1x12}] & 0 & 0 & 1 \end{pmatrix}_{15x15}$

$Q = 1\cdot 10^{-4}\cdot diag(1.29, 3.75, 4.60, 0.003, 0.001, 0.001, 571, 4439, 5461, 22.58, 19.85, 52.02, 74.73, 56.94, 469)$

Cuando llega una nueva medida (Corrector)

$$L = P_k^- C^T (R + CP_k^- C^T)^{-1} \tag{5.29}$$

$$x_k = x_k^- + L\left(z - h(x_{k-1},u)\right) \tag{5.30}$$

$$P_k = (I - LC)P_k^- \tag{5.31}$$

Donde:

$C = \begin{pmatrix} [I_{12x12}] & & [\emptyset_{12x3}] & \\ [I_{12x12}] & & [\emptyset_{12x3}] & \\ [\emptyset_{1x12}] & 0 & g\cos(\hat{\theta}_{k-1}) & 0 \\ [\emptyset_{1x12}] & -g\cos(\hat{\phi}_{k-1})\cos(\hat{\theta}_{k-1}) & g\sin(\hat{\phi}_{k-1})\sin(\hat{\theta}_{k-1}) & 0 \\ [\emptyset_{1x12}] & g\cos(\hat{\theta}_{k-1})\sin(\hat{\phi}_{k-1}) & g\cos(\hat{\phi}_{k-1})\sin(\hat{\theta}_{k-1}) & 0 \\ [\emptyset_{3x12}] & & [I_{3x3}] & \end{pmatrix}_{30x15}$

$R = diag(R_1 \cdots R_{30})$: si la norma de la ultima estimación de la fusión de los acelerómetros está dentro de un cierto umbral, en este caso (5,14);

$R = diag(R_1 \cdots R_{24}, 1 \cdot 10^4 \cdot (R_{25} \cdots R_{30}))$: si la aceleración es muy alta descartamos le damos menos credibilidad a la medida.

IMU MPU9250	**IMU LSM9DS1**	$\mathbf{y_{accel}}$
$R_1 = 2.26 \cdot 10^{-3}$	$R_{13} = 7.62 \cdot 10^{-3}$	$R_{25} = 2.19 \cdot 10^{-3}$
$R_2 = 1.99 \cdot 10^{-3}$	$R_{14} = 1.94 \cdot 10^{-3}$	$R_{26} = 1.27 \cdot 10^{-3}$
$R_3 = 5.20 \cdot 10^{-3}$	$R_{15} = 2.02 \cdot 10^{-3}$	$R_{27} = 5.13 \cdot 10^{-3}$
$R_4 = 7.06 \cdot 10^{-6}$	$R_{16} = 8.90 \cdot 10^{-6}$	**Actitud directa**
$R_5 = 5.09 \cdot 10^{-6}$	$R_{17} = 1.15 \cdot 10^{-4}$	$R_{28} = 1.06 \cdot 10^{-4}$
$R_6 = 7.11 \cdot 10^{-6}$	$R_{18} = 7.68 \cdot 10^{-6}$	$R_{29} = 1.69 \cdot 10^{-4}$
$R_7 = 1.35 \cdot 10^{3}$	$R_{19} = 0.18$	$R_{30} = 6.75 \cdot 10^{-4}$
$R_8 = 1.13 \cdot 10^{3}$	$R_{20} = 0.25$	
$R_9 = 9.52 \cdot 10^{3}$	$R_{21} = 0.37$	
$R_{10} = 2.26 \cdot 10^{-3}$	$R_{22} = 7.62 \cdot 10^{-3}$	
$R_{11} = 1.99 \cdot 10^{-3}$	$R_{23} = 1.94 \cdot 10^{-3}$	
$R_{12} = 5.20 \cdot 10^{-3}$	$R_{24} = 2.02 \cdot 10^{-3}$	

$$z = \begin{pmatrix} [Acc_{mpu}]_{3x1} \\ [Gyr_{mpu}]_{3x1} \\ [Mag_{mpu}]_{3x1} \\ [Vel_{mpu}]_{3x1} \\ [Acc_{lsm}]_{3x1} \\ [Gyr_{lsm}]_{3x1} \\ [Mag_{lsm}]_{3x1} \\ [Vel_{mpu}]_{3x1} \\ [y_accel(x_k^-)]_{3x1} \\ \hat{\phi}^- \\ \hat{\theta}^- \\ \hat{\psi}^- \end{pmatrix}$$

: $\hat{\phi}^-$, $\hat{\theta}^-$ y $\hat{\psi}^-$ calculado de forma directa como se comento previamente

$$h(x_{k-1},u) = \begin{pmatrix} [\hat{Acc}_{k-1}]_{3x1} \\ [\hat{Gyr}_{k-1}]_{3x1} \\ [\hat{Mag}_{k-1}]_{3x1} \\ [\hat{Vel}_{k-1}]_{3x1} \\ [y_accel(x_{k-1})]_{3x1} \\ \hat{\phi}_{k-1} \\ \hat{\theta}_{k-1} \\ \hat{\psi}_{k-1} \end{pmatrix}$$

Esquema de funcionamiento de la función implementada

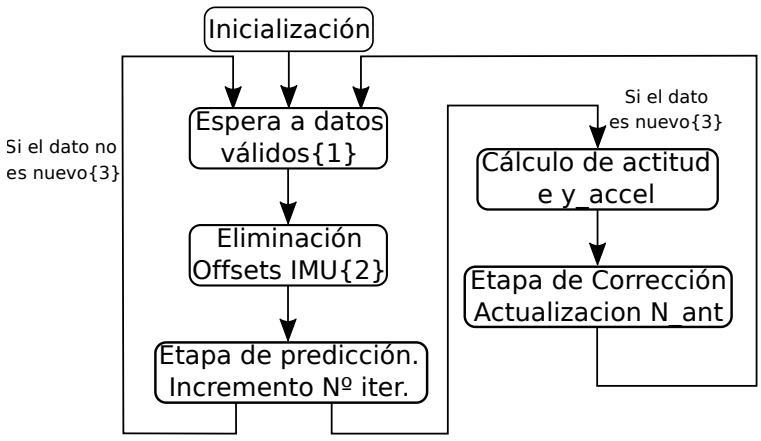

Figura 5.18 Esquema de funcionamiento de la función de estimación de la orientación basada en el libro.

Algunos comentarios generales sobre la realización de la funcion comentada en el esquema anterior:

{1} La espera de resultados validos se realiza comprobando que los valores de las aceleraciones son todos distintos de 0 que es el valor por defecto cuando no se reciben datos de la *Raspberry Pi* ya que el UAV siempre va a estar sometido a una aceleración en alguno de los ejes. Por otro lado también se considerará como dato invalido cuando alguno de los valores se mantenga constante durante más de 100 iteraciones consecutivas sin recibir datos diferentes, lo se puede considerar como que se ha perdido la conexión y el bloque *UDP receive* mantiene el ultimo valor recibido.

{2} Ambas IMUs presentan un cierto *offset* en los sensores de aceleración lineal y velocidad angular los cual puede inducir errores al integrar dichas mediciones, por ello se ha optado por su eliminación. Para estimar el valor de la corrección se ha dejado la *Raspberry Pi* con la placa Navio2 estática en varias posiciones para comprobar que el *offset* es similar en todas ellas y se ha realizado la media de todas las medidas recibidas. De esta forma en cada iteración se elimina dicha media.

{3} Por último la comprobación de si el dato utilizado para la nueva iteración es nuevo, se comprueba que sea diferente que el utilizado en la iteración anterior de esta forma se corrige solo cuando se recibe un dato nuevo.

Resultados obtenidos con la estimación del libro

Con estas matrices y la Ecuación 5.27 a la Ecuación 5.27 se han conseguido los siguientes resultados en la estimación de actitud.

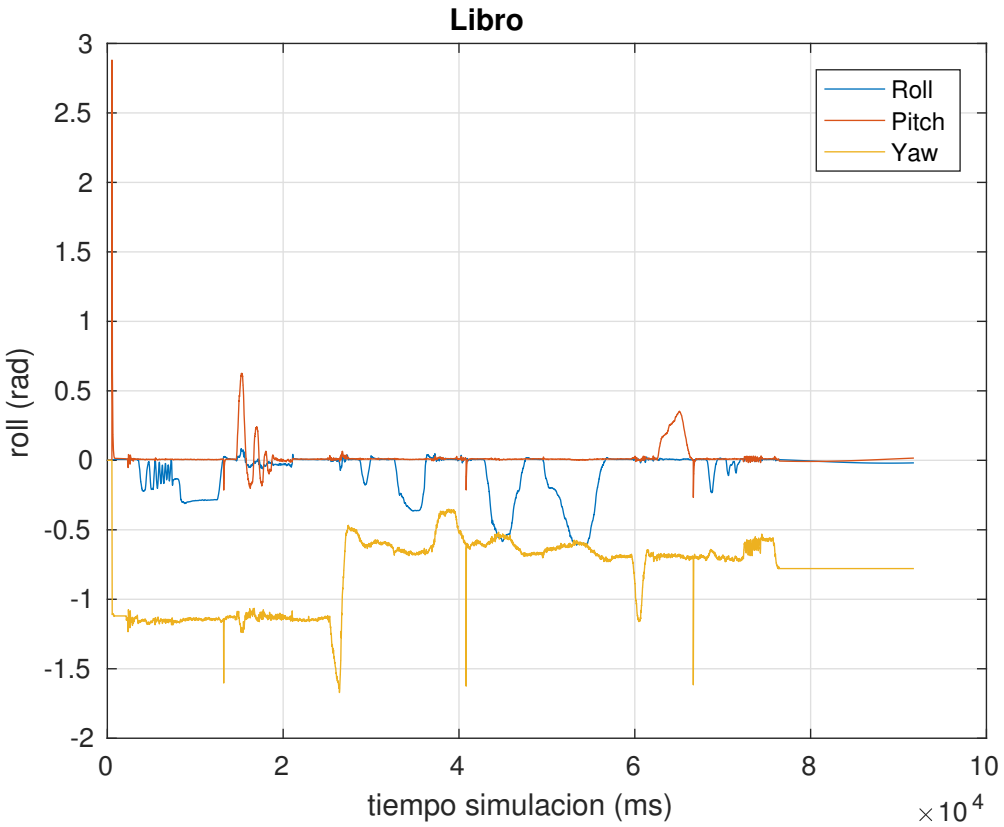

Figura 5.19 Resultados obtenidos de la estimación de la actitud basada en el libro.

En esta ocasión vemos como nos hemos desecho completamente de la deriva del caso anterior, además se observa como la estimación está suficientemente filtrada como para poder ser utilizada. Por otro lado cabe destacar que este método es considerablemente menos sensible a las aceleraciones lineales.

Comparación de las tres estimaciones

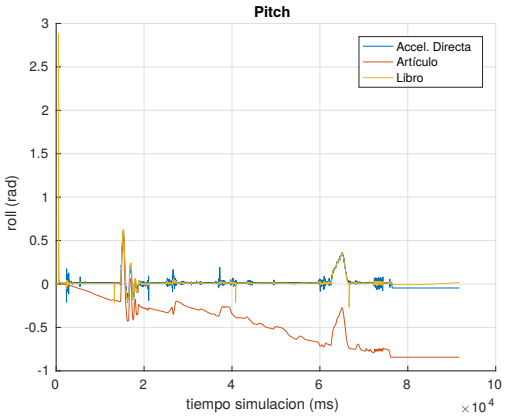

Figura 5.20 Comparación de la estimación de actitud en *Pitch*.

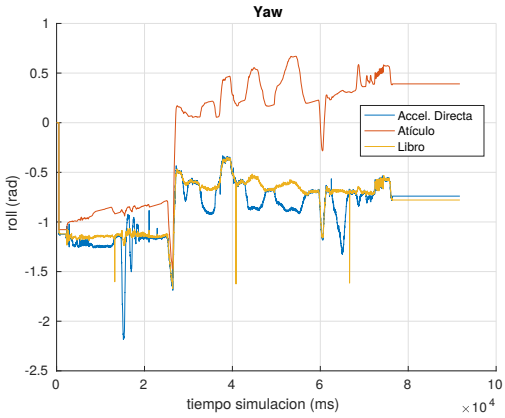

Figura 5.21 Comparación de la estimación de actitud en *Yaw*.

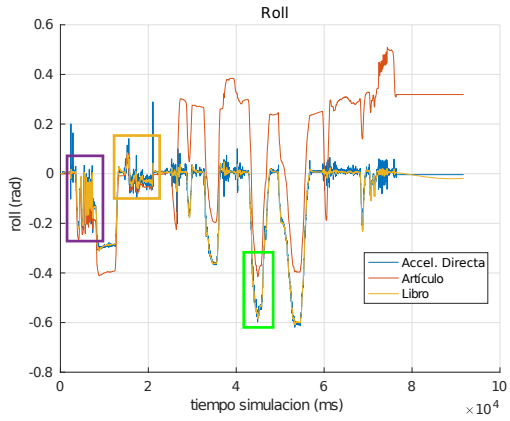

Figura 5.22 Comparación de la estimación de actitud en *Roll*.

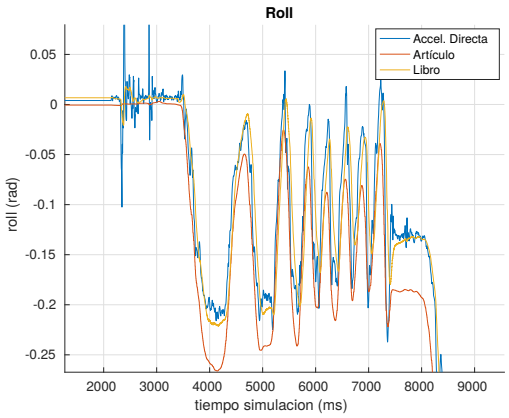

Figura 5.23 Comparación de la estimación de actitud en *Roll*, Zoom sobre el cuadrado morado.

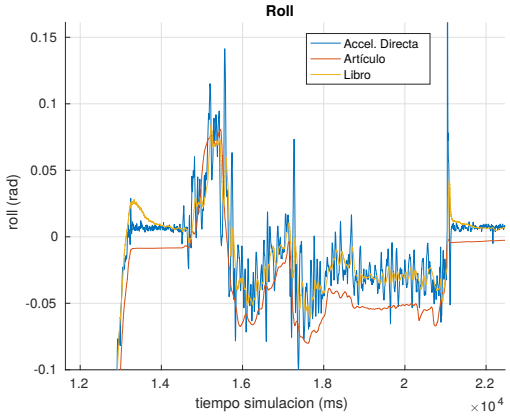

Figura 5.24 Comparación de la estimación de actitud en *Roll*, Zoom sobre el cuadrado amarillo.

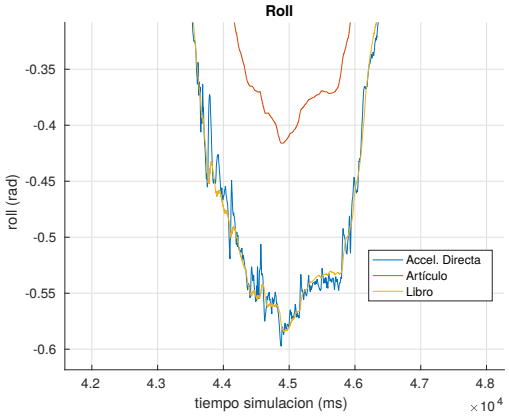

Figura 5.25 Comparación de la estimación de actitud en *Roll*, Zoom sobre el cuadrado Verde.

En base a los resultados obtenidos se puede llegar a concluir que la mejor opción para ser implementada es la estimación basada en el libro ya que frente a la basada en el artículo no tiene deriva a pesar de tener mayor ruido. Por otro lado frente a la estimación directa esta versión tiene bastante menos ruido además es menos sensible a la aceleraciones bruscas.

Filtro de Kalman para estimación de posición

Una vez conseguido un filtro para la estimación de la actitud el siguiente paso es incluir la estimación de posición, para ello se dispone de GPS y un Barómetro para utilizado en la estimación de la altura. Por tanto el primer paso será añadir estos sensores como entrada al sistema y aumentar las matrices para añadir los nuevos estados a las estimaciones.

Método de posicionamiento utilizado

Como medida de posición se van a utilizar coordenadas NED que permiten errores menores que las coordenadas absolutas ya que se trabaja en incremental respecto a una posición de origen. Además es independiente del posición geográfica lo que facilita un posible control de posición.

Este sistema de coordenadas nos proporciona el desplazamiento desde la posición de origen hacia el norte, el este y hacia abajo.

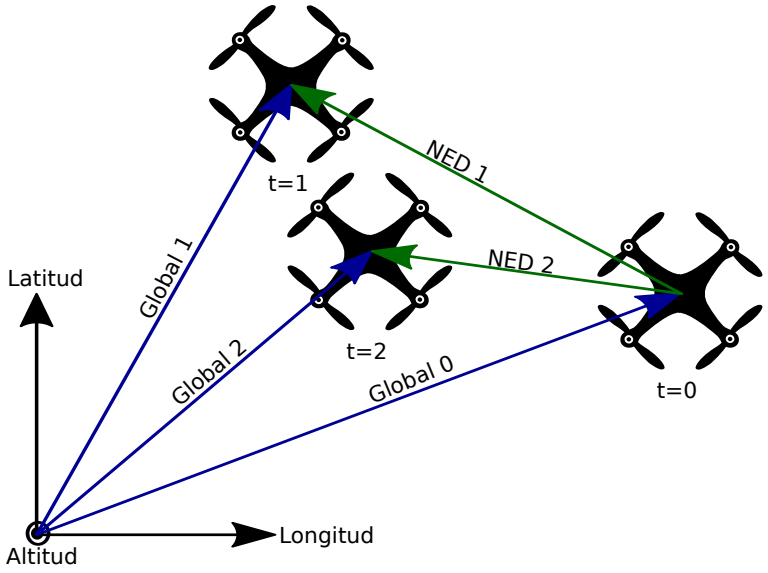

Figura 5.26 Explicación de las coordenadas NED.

Para el cálculo de estas coordenadas se espera a recibir valores válidos durante 10 iteraciones consecutivas en cuyo caso se almacenan los valores de esas coordenadas recibidas como valores de referencia e iniciales.

Por último cabe destacar que se ha utilizado el barómetro para la estimación de altura junto a las dos alturas que nos proporciona el GPS, sobre el mar y sobre el elipsoide. Aunque estas medidas por separado no son bastante diferentes al utilizar un sistema de coordenadas incremental respecto a la posición de inicio, los incrementos de todas estas medidas dan una aproximación bastante buena de la altura del UAV.

La conversión que se ha utilizado para obtener la altura a partir de la presión es la atmósfera estándar internacional[23]:

Partiendo de la presión en función de la altura:

$$P = P_0 \left(\frac{T_0 + a(h - h_0)}{T_0} \right)^{-\frac{g}{aR}} \tag{5.32}$$

Despejando la h:

$$\left(\frac{P}{P_0} \right)^{-\frac{aR}{g}} = \frac{T_0 + a(h - h_0}{T_0}$$

$$h_{baro}^{ref} = h_0 + \frac{T_0 \cdot \left(\frac{P}{P_0} \right)^{-\frac{aR}{g}} - T_0}{a} \tag{5.33}$$

Donde considerando que volamos en la troposfera, h<17km:

$P :=$ Presión atmosférica medida por el barómetro.

$P_0 = 101325Pa :=$ Presión base de la capa.

$T_0 = 288.15K :=$ Temperatura base de la capa de la atmósfera considerada.

$h :=$ Altura en la que nos encontramos.

$h_0 = 0m :=$ Altura geopotencial de la capa en la que nos encontramos.

$a = -6.5K/km :=$ Gradiente térmico.

$R = 287 \frac{m^2}{s^2 K} =$ Constante de individual del aire.

$g = 9.80665 m/s^2 :=$ Gravedad.

Las expresiones para el cálculo de las posiciones y velocidades en coordenadas NED se lleva a cabo a partir de las siguientes expresiones que utilizan las coordenadas de inicio y las recibidas en cada iteración. El cálculo se basa en el realizado en la réplica de PX4, [24].

Posición:

$$N = (lat - lat_{ref}) \cdot \frac{\pi}{180} \cdot Rm \tag{5.34}$$

$$E = (long - long_{ref}) \cdot \frac{\pi}{180} \cdot Rn \cdot \cos\left(lat_{ref} \cdot \frac{\pi}{180}\right) \tag{5.35}$$

Donde:

$$Rm = R \cdot \frac{(1 - flat^2)}{\left(1 - flat^2 \cdot \sin^2\left(lat_{ref} \cdot \frac{\pi}{180}\right)\right)^{3/2}}$$

$$Rn = \frac{R}{\left(1 - flat^2 \cdot \sin^2\left(lat_{ref} \cdot \frac{\pi}{180}\right)\right)^{1/2}}$$

$R = 6378137 :=$ Radio terrestre.
$A = 1/298.257223563$
$flat = \sqrt{A \cdot (2 - A)} :=$ Corrección por planicie terrestre.

$$D_{sea} = h_{sea}^{ref} - h_{sea} \tag{5.36}$$

$$D_{elip} = h_{elip}^{ref} - h_{elip} \tag{5.37}$$

$$D_{baro} = h_{baro}^{ref} - \left(h_0 + \frac{T_0 \cdot \left(\frac{P}{P_0}\right)^{-\frac{aR}{g}} - T_0}{a} \right) \tag{5.38}$$

Velocidad

$$V_N = \frac{N_k - N_{k-1}}{dt_{gps}} \tag{5.39}$$

$$V_E = \frac{E_k - E_{k-1}}{dt_{gps}} \tag{5.40}$$

$$V_D^{sea} = \frac{D_k^{sea} - N_{k-1}^{sea}}{dt_{gps}} \tag{5.41}$$

$$V_D^{elip} = \frac{D_k^{elip} - N_{k-1}^{elip}}{dt_{gps}} \tag{5.42}$$

Cabe destacar que no se ha utilizado la velocidad a partir del barómetro ya que su diferencial de tiempo en menos exacto y su medida es bastante más ruidosa lo que puede inducir errores mayores.

EKF implementado

Como en el Filtro comentado anteriormente y basado en el trabajo del libro[21] la etapa de corrección solo se ejecutará cuando llegue una medida nueva, en esta ocasión además esta corrección será distinta en función de si la medida que llega es el GPS o la IMU. En el caso de que sea el GPS se actualiza la posición y velocidad a partir de las coordenadas NED calculadas en el caso de que solo llegue la IMU se actualizará la posición a partir de la integración de la estimación de la aceleración rotada la actitud para hacerla coincidir con posiciones NED. De esta forma el EKF implementado tendrá por un lado una etapa común de predicción y una etapa de corrección que solo diferirá en la parte de la estimación de posición y velocidad NED. Por tanto el filtro queda de la siguiente forma:

Predicción:

$$x_k = x_{k-1} + \frac{dt_{imu}}{N} f(x_{k-1}, u) \tag{5.43}$$

$$P_k = P_{k-1} + \frac{dt_{IMU}}{N}\left(AP_{k-1} + P_{k-1}A^T + Q\right) \tag{5.44}$$

Donde:

$$x = \begin{pmatrix} \left[\hat{Acc}\right]_{3x1} \\ \left[\hat{Gyr}\right]_{3x1} \\ \left[\hat{Mag}\right]_{3x1} \\ \left[\hat{V_{Body}}\right]_{3x1} \\ \hat{\phi} \\ \hat{\theta} \\ \hat{\psi} \\ \left[\hat{P_{NED}}\right]_{3x1} \\ \left[\hat{V_{NED}}\right]_{3x1} \end{pmatrix}_{21x1}$$

$$f(x,u) = \begin{pmatrix} \emptyset_{12x1} \\ p + q\sin(\hat{\phi}_{k-1})\tan(\hat{\theta}_{k-1}) + r\cos(\hat{\phi}_{k-1})\tan(\hat{\theta}_{k-1}) \\ q\cos(\hat{\phi}_{k-1}) - r\sin(\hat{\phi}_{k-1}) \\ 0 \\ \emptyset_{6x1} \end{pmatrix}_{15x1}$$

$$A = \begin{pmatrix} [I_{12x12}] & & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & [\emptyset_{12x3}] \\ [\emptyset_{1x12}] & q\cos(\hat{\phi}_{k-1})\tan(\hat{\theta}_{k-1}) - r\sin(\hat{\phi_{k-1}})tan(\hat{\theta_{k-1}}) & \frac{q\sin(\hat{\phi}_{k-1}) - r\cos(\hat{\phi}_{k-1})}{\cos^2(\hat{\theta}_{k-1})} & 0 & [\emptyset_{1x3}] & [\emptyset_{1x3}] \\ [\emptyset_{1x12}] & -q\sin(\hat{\phi}_{k-1}) - r\cos(\hat{\phi}_{k-1}) & 0 & 0 & [\emptyset_{1x3}] & [\emptyset_{1x3}] \\ [\emptyset_{1x12}] & 0 & 0 & 1 & [\emptyset_{1x3}] & [\emptyset_{1x3}] \\ [\emptyset_{3x12}] & & [\emptyset_{3x3}] & & I_{3x3} & [\emptyset_{3x3}] \\ [\emptyset_{3x12}] & & [\emptyset_{3x3}] & & \emptyset_{3x3} & [I_{3x3}] \end{pmatrix}_{21x21}$$

$$Q = 1\cdot 10^{-4}\cdot diag(Q_1 \cdots Q_{21})$$

Aceleración	**Velocidad *Body***	**Velocidad NED**
$Q_1 = 1.29\cdot 10^{-4}$	$Q_{10} = 2.26\cdot 10^{-3}$	$Q_{19} = 1\cdot 10^{-3}$
$Q_2 = 3.75\cdot 10^{-4}$	$Q_{11} = 1.99\cdot 10^{-3}$	$Q_{20} = 1\cdot 10^{-3}$
$Q_3 = 4.6\cdot 10^{-4}$	$Q_{12} = 5.20\cdot 10^{-3}$	$Q_{21} = 1\cdot 10^{-3}$
Giróscopo	**Actitud**	
$Q_4 = 3\cdot 10^{-7}$	$Q_{13} = 7.47\cdot 10^{-3}$	
$Q_5 = 1\cdot 10^{-7}$	$Q_{14} = 5.69\cdot 10^{-3}$	
$Q_6 = 1\cdot 10^{-7}$	$Q_{15} = 4.69\cdot 10^{-3}$	
Magnetómetro	**Posición NED**	
$Q_7 = 5.72\cdot 10^{-2}$	$Q_{16} = 1\cdot 10^{-3}$	
$Q_8 = 44.4\cdot 10^{-2}$	$Q_{17} = 1\cdot 10^{-3}$	
$Q_9 = 54.6\cdot 10^{-2}$	$Q_{18} = 10\cdot 10^{-3}$	

Algunos detalles importantes son que como diferencial de tiempo se utiliza la diferencia de tiempo entre que se reciben datos, que coincide con la frecuencia de recepción de la IMU y la estimación de N es siempre el número de iteraciones entre la recepción de dos datos diferentes.

Corrección:

$$L = P_k^- C^T (R + CP_k^- C^T)^{-1} \tag{5.45}$$

$$x_k = x_k^- + L\left(z - h(x_{k-1}, u)\right) \tag{5.46}$$

$$P_k = (I - LC)P_k^- \tag{5.47}$$

Donde, en el caso de que llegue una nueva medida de GPS, las matrices tienen la siguiente forma:

$$C = \begin{pmatrix}
[I_{12x12}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & \\
[I_{12x12}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & \\
[\emptyset_{1x12}] & 0 & g\cos(\hat{\theta}_{k-1}) & 0 & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & -g\cos(\hat{\phi}_{k-1})\cos(\hat{\theta}_{k-1}) & g\sin(\hat{\phi}_{k-1})\sin(\hat{\theta}_{k-1}) & 0 & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & g\cos(\hat{\theta}_{k-1})\sin(\hat{\phi}_{k-1}) & g\cos(\hat{\phi}_{k-1})\sin(\hat{\theta}_{k-1}) & 0 & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & \\
[\emptyset_{3x12}] & & [I_{3x3}] & & [\emptyset_{3x3}] & & [\emptyset_{3x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & 1 & 0 & 0 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & 0 & 1 & 0 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & 0 & 0 & 1 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & 0 & 0 & 1 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & 0 & 0 & 1 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & 1 & 0 & 0 \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & 0 & 1 & 0 \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & 0 & 0 & 1 \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & 0 & 0 & 1
\end{pmatrix}_{39x21}$$

$R = diag(R_1 \cdots R_{39})$: si la norma de la ultima estimación de la fusión de los acelerómetros está dentro de un cierto umbral, en este caso (5,14);

$R = diag(R_1 \cdots R_{24}, 1 \cdot 10^4 \cdot (R_{25} \cdots R_{30}), R_{31} \cdots R_{39})$: si la aceleración es muy alta descartamos le damos menos credibilidad a la medida.

IMU MPU9250

$R_1 = 2.26 \cdot 10^{-3}$
$R_2 = 1.99 \cdot 10^{-3}$
$R_3 = 5.20 \cdot 10^{-3}$
$R_4 = 7.06 \cdot 10^{-6}$
$R_5 = 5.09 \cdot 10^{-6}$
$R_6 = 7.11 \cdot 10^{-6}$
$R_7 = 1.35 \cdot 10^{3}$
$R_8 = 1.13 \cdot 10^{3}$
$R_9 = 9.52 \cdot 10^{3}$
$R_{10} = 2.26 \cdot 10^{-3}$
$R_{11} = 1.99 \cdot 10^{-3}$
$R_{12} = 5.20 \cdot 10^{-3}$

IMU LSM9DS1

$R_{13} = 7.62 \cdot 10^{-3}$
$R_{14} = 1.94 \cdot 10^{-3}$
$R_{15} = 2.02 \cdot 10^{-3}$
$R_{16} = 8.90 \cdot 10^{-6}$
$R_{17} = 1.15 \cdot 10^{-4}$
$R_{18} = 7.68 \cdot 10^{-6}$
$R_{19} = 0.18$
$R_{20} = 0.25$
$R_{21} = 0.37$
$R_{22} = 7.62 \cdot 10^{-3}$
$R_{23} = 1.94 \cdot 10^{-3}$
$R_{24} = 2.02 \cdot 10^{-3}$

y_{accel}

$R_{25} = 2.19 \cdot 10^{-3}$
$R_{26} = 1.27 \cdot 10^{-3}$
$R_{27} = 5.13 \cdot 10^{-3}$

Actitud directa

$R_{28} = 1.06 \cdot 10^{-4}$
$R_{29} = 1.69 \cdot 10^{-4}$
$R_{30} = 6.75 \cdot 10^{-4}$

Posición NED

$R_{31} = 0.1$
$R_{32} = 0.1$
$R_{33} = 1$
$R_{34} = 1$
$R_{35} = 0.1$

Velocidad NED

$R_{36} = 0.1$
$R_{37} = 0.1$
$R_{38} = 0.1$
$R_{39} = 0.1$

$$z = \begin{pmatrix}
[Acc_{mpu}]_{3x1} \\
[Gyr_{mpu}]_{3x1} \\
[Mag_{mpu}]_{3x1} \\
[Vel_{mpu}]_{3x1} \\
[Acc_{lsm}]_{3x1} \\
[Gyr_{lsm}]_{3x1} \\
[Mag_{lsm}]_{3x1} \\
[Vel_{mpu}]_{3x1} \\
[y_accel(x_k^-)]_{3x1} \\
\hat{\phi}^- \\
\hat{\theta}^- \\
\hat{\psi}^- \\
N \\
E \\
D_{sea} \\
D_{elip} \\
D_{baro} \\
V_N \\
V_E \\
V_D^{sea} \\
V_D^{elip}
\end{pmatrix}$$

$$h(x_{k-1}, u) = \begin{pmatrix}
[\hat{Acc}_{k-1}]_{3x1} \\
[\hat{Gyr}_{k-1}]_{3x1} \\
[\hat{Mag}_{k-1}]_{3x1} \\
[\hat{Vel}_{k-1}]_{3x1} \\
[y_accel(x_{k-1})]_{3x1} \\
\hat{\phi}_{k-1} \\
\hat{\theta}_{k-1} \\
\hat{\psi}_{k-1} \\
\hat{N}_{k-1} \\
\hat{E}_{k-1} \\
\hat{D}_{k-1} \\
\hat{D}_{k-1} \\
\hat{D}_{k-1} \\
\hat{V}_{k-1}^N \\
\hat{V}_{k-1}^E \\
\hat{V}_{k-1}^D \\
\hat{V}_{k-1}^D
\end{pmatrix}$$

Si por el contrario no llega ninguna nueva medida de GPS las matrices tendrán la siguiente forma.

$$C = \begin{pmatrix}
[I_{12x12}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & & \\
[I_{12x12}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & & [\emptyset_{12x3}] & & \\
[\emptyset_{1x12}] & 0 & g\cos(\hat{\theta}_{k-1}) & 0 & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & & \\
[\emptyset_{1x12}] & -g\cos(\hat{\phi}_{k-1})\cos(\hat{\theta}_{k-1}) & g\sin(\hat{\phi}_{k-1})\sin(\hat{\theta}_{k-1}) & 0 & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & & \\
[\emptyset_{1x12}] & g\cos(\hat{\theta}_{k-1})\sin(\hat{\phi}_{k-1}) & g\cos(\hat{\phi}_{k-1})\sin(\hat{\theta}_{k-1}) & 0 & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & & \\
[\emptyset_{3x12}] & & [I_{3x3}] & & [\emptyset_{3x3}] & & [\emptyset_{3x3}] & & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & 1 & 0 & 0 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & 0 & 1 & 0 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & 0 & 0 & 1 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & 0 & 0 & 1 & [\emptyset_{1x3}] & \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & & 1 & 0 & 0 \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & & 0 & 1 & 0 \\
[\emptyset_{1x12}] & & [\emptyset_{1x3}] & & [\emptyset_{1x3}] & & 0 & 0 & 1
\end{pmatrix}_{37x21}$$

$R = diag(R_1 \cdots R_{39})$: si la norma de la ultima estimación de la fusión de los acelerómetros está dentro de un cierto umbral, en este caso (5,14);

$R = diag(R_1 \cdots R_{24}, 1 \cdot 10^4 \cdot (R_{25} \cdots R_{30}), R_{31} \cdots R_{37})$: si la aceleración es muy alta descartamos le damos menos credibilidad a la medida.

IMU MPU9250	**IMU LSM9DS1**	$\mathbf{y_{accel}}$	**Velocidad NED**
$R_1 = 2.26 \cdot 10^{-3}$	$R_{13} = 7.62 \cdot 10^{-3}$	$R_{25} = 2.19 \cdot 10^{-3}$	$R_{35} = 0.1$
$R_2 = 1.99 \cdot 10^{-3}$	$R_{14} = 1.94 \cdot 10^{-3}$	$R_{26} = 1.27 \cdot 10^{-3}$	$R_{36} = 0.1$
$R_3 = 5.20 \cdot 10^{-3}$	$R_{15} = 2.02 \cdot 10^{-3}$	$R_{27} = 5.13 \cdot 10^{-3}$	$R_{37} = 0.1$
$R_4 = 7.06 \cdot 10^{-6}$	$R_{16} = 8.90 \cdot 10^{-6}$	**Actitud directa**	
$R_5 = 5.09 \cdot 10^{-6}$	$R_{17} = 1.15 \cdot 10^{-4}$	$R_{28} = 1.06 \cdot 10^{-4}$	
$R_6 = 7.11 \cdot 10^{-6}$	$R_{18} = 7.68 \cdot 10^{-6}$	$R_{29} = 1.69 \cdot 10^{-4}$	
$R_7 = 1.35 \cdot 10^{3}$	$R_{19} = 0.18$	$R_{30} = 6.75 \cdot 10^{-4}$	
$R_8 = 1.13 \cdot 10^{3}$	$R_{20} = 0.25$	**Posición NED**	
$R_9 = 9.52 \cdot 10^{3}$	$R_{21} = 0.37$	$R_{31} = 0.1$	
$R_{10} = 2.26 \cdot 10^{-3}$	$R_{22} = 7.62 \cdot 10^{-3}$	$R_{32} = 0.1$	
$R_{11} = 1.99 \cdot 10^{-3}$	$R_{23} = 1.94 \cdot 10^{-3}$	$R_{33} = 0.1$	
$R_{12} = 5.20 \cdot 10^{-3}$	$R_{24} = 2.02 \cdot 10^{-3}$	$R_{34} = 1$	

$$z = \begin{pmatrix}
[Acc_{mpu}]_{3x1} \\
[Gyr_{mpu}]_{3x1} \\
[Mag_{mpu}]_{3x1} \\
[Vel_{mpu}]_{3x1} \\
[Acc_{lsm}]_{3x1} \\
[Gyr_{lsm}]_{3x1} \\
[Mag_{lsm}]_{3x1} \\
[Vel_{mpu}]_{3x1} \\
[y_accel(x_k^-)]_{3x1} \\
\hat{\phi}^- \\
\hat{\theta}^- \\
\hat{\psi}^- \\
P_x \\
P_y \\
P_z \\
D_{baro} \\
V_x \\
V_y \\
V_z
\end{pmatrix}
\qquad
h(x_{k-1},u) = \begin{pmatrix}
[\hat{Acc}_{k-1}]_{3x1} \\
[\hat{Gyr}_{k-1}]_{3x1} \\
[\hat{Mag}_{k-1}]_{3x1} \\
[\hat{Vel}_{k-1}]_{3x1} \\
[y_accel(x_{k-1})]_{3x1} \\
\hat{\phi}_{k-1} \\
\hat{\theta}_{k-1} \\
\hat{\psi}_{k-1} \\
\hat{N}_{k-1} \\
\hat{E}_{k-1} \\
\hat{D}_{k-1} \\
\hat{D}_{k-1} \\
\hat{V}^N_{k-1} \\
\hat{V}^E_{k-1} \\
\hat{V}^D_{k-1}
\end{pmatrix}$$

Resultados obtenidos

A continuación se muestran los resultados obtenidos con el filtro de posición, en concreto se muestran dos experimentos, 1 el que se ha dejado el UAV completamente estático, Figura 5.27, y otro en el que se ha

intentado realizar una trayectoria en L para ver su respuesta, Figura 5.28 y Figura 5.29.

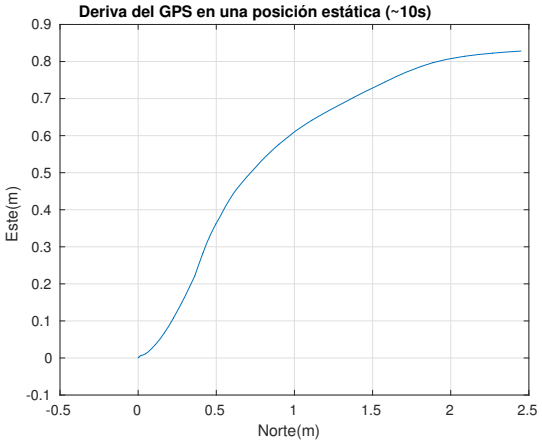

Figura 5.27 UAV estático durante 10s.

En base a este experimento se puede observar como existe una gran deriva en el filtro la cual sería interesante tratar de eliminar con algo más de tiempo en un futuro.

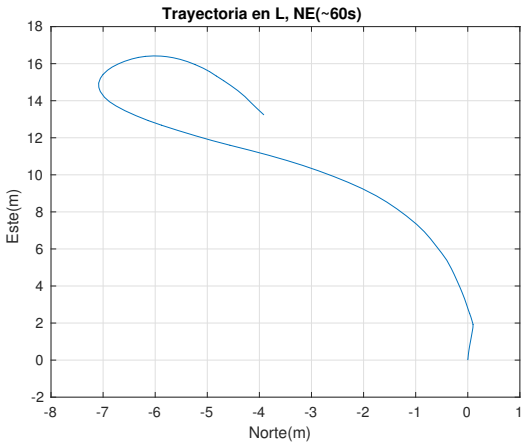

Figura 5.28 Trayectoria 2D en L.

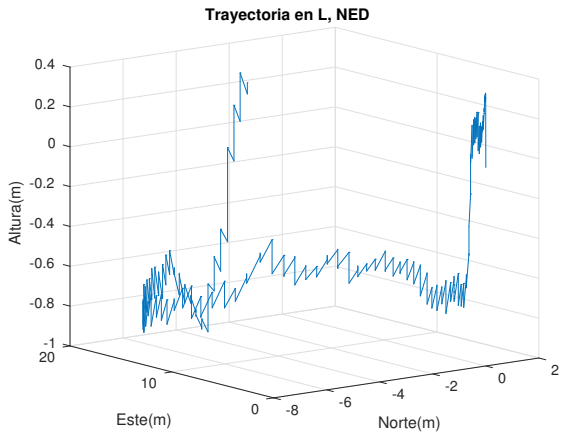

Figura 5.29 Trayectoria 3D en L.

Por otro lado vemos como a pesar de la deriva el filtro es capaz de reconocer la trayectoria seguida de forma aproximada. También se observa un ruido bastante alto en altitud provocado principalmente por el barómetro. En un futuro sería altamente prioritario arreglar el comportamiento de este filtro.

Posiblemente el error se encuentre en las correcciones realizadas por la integración de las IMUs las cuales parecen no llevarse a cabo debidamente, por el pequeño diferencia de tiempo entre medidas.

Subsistema EKF de SIMULINK®

A continuación se muestra el interior del subsistema correspondiente al filtro EKF de simulink.

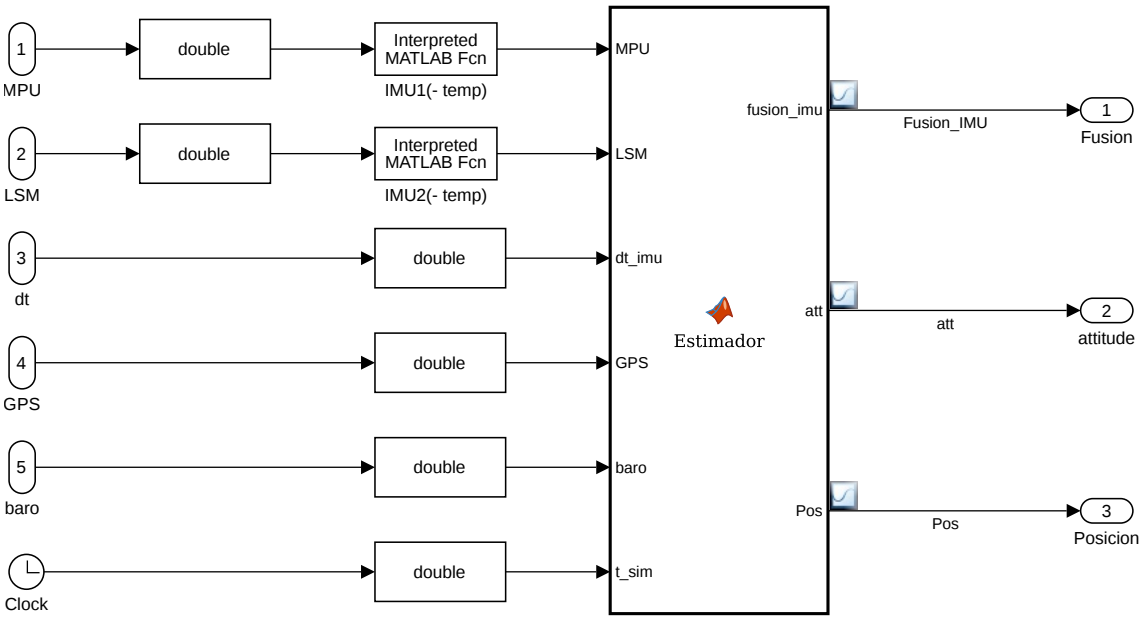

Figura 5.30 Interior del subsistema contenedor del EKF.

Motor mixer

Pseudoinversa

Como primera solución a un posible mezclador que a partir de las señales que entregue el controlador en empuje y torque sobre los tres ejes, se propone la pseudoinversa de la matriz que indica la contribución de cada motor a la señal de control correspondiente.

La matriz con las señales de control es la siguiente:

$$F = \begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} \tag{5.48}$$

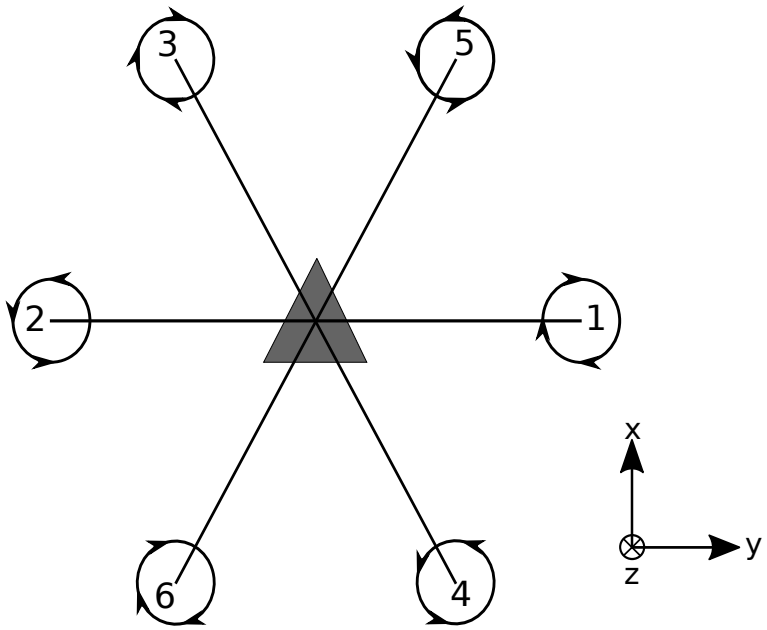

Figura 5.31 Configuración montada en el hexarotor.

En base a la Figura 5.31 se puede concluir que:

$$T = \sum_{1}^{6} T_i \tag{5.49}$$

$$\tau_x = -LT_1 + LT_2 + L\cos 60T_3 - L\cos 60T_4 - L\cos 60T_5 + L\cos 60T_6 \tag{5.50}$$

$$\tau_y = -0T_1 + 0T_2 + L\sin 60T_3 - L\sin 60T_4 + L\sin 60T_5 - L\sin 60T_6 \tag{5.51}$$

$$\tau_z = -bT_1 + bT_2 - bT_3 + bT_4 + bT_5 - bT_6 \tag{5.52}$$

En forma matricial:

$$F = D \cdot x \tag{5.53}$$

$$\begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -L & L & 0.5L & -0.5L & -0.5L & 0.5L \\ 0 & 0 & \sqrt{3}/2L & -\sqrt{3}/2L & \sqrt{3}/2L & -\sqrt{3}/2L \\ -b & b & -b & b & b & -b \end{pmatrix} \cdot \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{pmatrix} \tag{5.54}$$

Donde:
T := Empuje total requerido al UAV
τ_x :=Torque requerido respecto al eje x
τ_y :=Torque requerido respecto al eje y
τ_z :=Torque requerido respecto al eje z
T_i :=Empuje individual de cada uno de los rotores
L :=Longitud de cada brazo del UAV, en nuestro caso 0.55m.
b :=Fricción de cada pala con el aire que genera un momento contrario al sentido de giro.

Una vez planteado el problema queda claro que la solución al mismo no es única, ya que tenemos 2 grados de libertad, por ello como primera solución se propone la pseudoinversa que la solución que minimiza los empujes de cada uno. El principal problema que nos encontramos al utilizar esta solución es que la

solución obtenida no está acotado pudiendo pedirse empujes negativos o superiores a lo que cada motor puede proporcionar.

De esta forma la solución al problema queda:

$$x = D^{\dagger} F \tag{5.55}$$

$$\begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{pmatrix} = D^T * (D * D^T)^{-1}; \begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} \tag{5.56}$$

Esta solución es realmente fácil de implementar y apenas consume tiempo de computo lo que la hace ideal para ejecutarla en SIMULINK® , sin embargo dado que no se encuentra acotada es inviable implementarla en un sistema real ya que no podemos asegurar que de una solución válida.

Optimización de mínimos cuadrados

Otra posible implementación del mezclador puede ser mediante ”programación cuadrática” con el objetivo de optimizar la solución de mínimos cuadrados pero manteniendo una saturación.

La función a optimizar sería:

$$J = \frac{1}{2} x^T H x + f^T x \tag{5.57}$$

Con,

$x = \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{pmatrix}$

$H = I_{6x6}$

$f = \emptyset_{6x1}$

Y con las siguientes restricciones:

$$A_{eq} \cdot x = b_{eq} \tag{5.58}$$

$$lb \leq x \leq ub \tag{5.59}$$

Donde,

$$A_{eq} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -L & L & 0.5L & -0.5L & -0.5L & 0.5L \\ 0 & 0 & \sqrt{3}/2L & -\sqrt{3}/2L & \sqrt{3}/2L & -\sqrt{3}/2L \\ -b & b & -b & b & b & -b \end{pmatrix}$$

$b_{eq} = \begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix}$

$lb = T_{min}$:= Empuje mínimo de cada motor.

$ub = T_{max}$:= Empuje máximo de cada motor.

En esta ocasión nos encontramos con que somos capaces de saturar el empuje que puede dar cada motor aunque no siempre se encuentra una solución válida ya que se fuerza a que se cumpla la igualdad y al saturar puede no llegar a cumplirse.

Además no es viable su implementación en SIMULINK® ya que utiliza la función ”quadprog” que utiliza métodos numéricos para resolver la optimización y que no puede ser implementada en una ”matlab function” de SIMULINK® por lo que si se quiere ejecutar se debe incluir en una función interpretada lo que ralentiza en exceso la ejecución de la simulación.

Una posible mejora para conseguir siempre una solución válida es realizar la pseudoinversa, que se trata del caso óptimo cuando no tengamos saturación de ninguno de los motores y tratar de minimizar el error entre la salida deseada y posible teniendo en cuenta las saturaciones.

Para ello se propone la siguiente optimización:

$$J = \frac{1}{2}(A_{eq}T - F)^T H(A_{eq}T - F) + f'(A_{eq} - F) \tag{5.60}$$

Donde al desarrollar la expresión se llega al siguiente resultado:

$$J = \frac{1}{2}(T^T(A_{eq}^T H A_{eq})T) - F^T H A_{eq} T = \frac{1}{2}T^T \hat{H} T + f'T \tag{5.61}$$

Con la siguiente restricción:

$$lb \leq x \leq ub \tag{5.62}$$

Como se puede ver ahora no forzamos la igualdad ya que si no se consigue con la pseudoinversa no se puede conseguir cumpliendo las restricciones de empuje máximo y mínimo. Por otro lado mediante la matriz H se le da mucho más peso a minimizar el error en el empuje total que en los momentos sobre x,y,z. Por este motivo la matriz H tiene la siguiente forma:

$$H = \begin{pmatrix} 100 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

De esta forma nos aseguramos que siempre tenemos una solución válida que minimiza el error entre lo que pide en controlador y lo que somos capaces de conseguir con los motores que tenemos.

Sin embargo esto no soluciona el tiempo de ejecución ya que la iteración en la que la pseudoinversa no tenga solución válida tendremos que ejecutar este algoritmo lo que ralentizará en exceso el tiempo de ejecución.

Por lo que una posible solución podría ser tabular los resultados de el máximo número de iteraciones para evitar tener que ejecutar el algoritmo, o incluso inferir una función que aproxime los resultados del mismo.

Además de esto para intentar suavizar la transición entre la solución de la pseudoinversa y la optimización de mínimos cuadrados se ha modificado la Ecuación 5.61 añadiendole a la matriz $\hat{H}$ un termino constante de 5e-3. De esta forma se ha conseguido una transición mucho más suave. Por lo que la matriz implementada finalmente queda de la siguiente forma:

$$\hat{H} = A_{eq}^T \cdot diag(100,1,1,1,) \cdot A_{eq} + 5 \cdot 10^{-3} \cdot I_{6x6}$$

A continuación se muestra el empuje del motor 1 cuando se le pide al UAV un empuje total entre 5 y 55 N junto a un torque en "x" entre -20 y 20 N/m. Donde se debe tener en cuenta que el empuje máximo y mínimo que puede proporcionar cada motor es de 8.333N y 1N respectivamente.

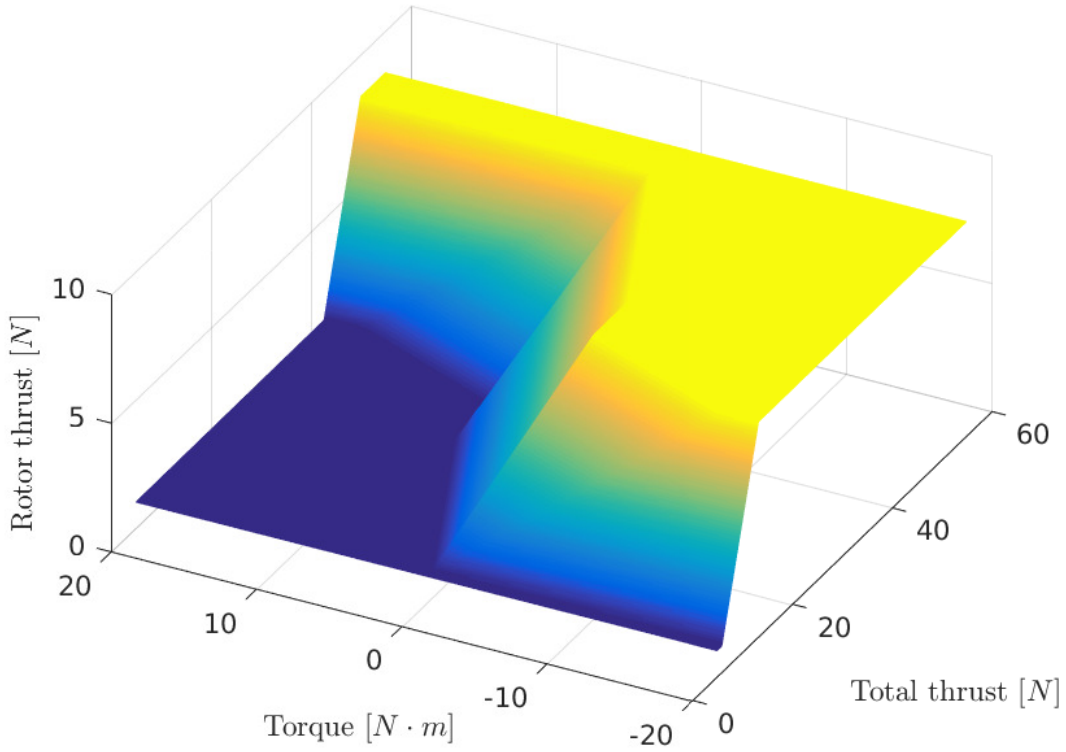

Figura 5.32 Empuje Motor 1.

En base a la figura anterior se puede llegar a concluir que es viable la determinación de una función con la que expresar el valor de empuje de cada motor en función de las entradas ya que todos los motores son similares. En definitiva implementar algo similar a lo desarrollado en [25],[26].

Subsistema *motor mixer* de SIMULINK®

A continuación se muestra el interior del subsistema correspondiente al *motor mixer* así como la máscara aplicada, en la que se definen los parámetros propios del UAV, como la longitud de los brazos o el empuje máximo y mínimo que deseamos, así como los valores propios de los motores utilizados, *duty cycle* máximo y mínimo.

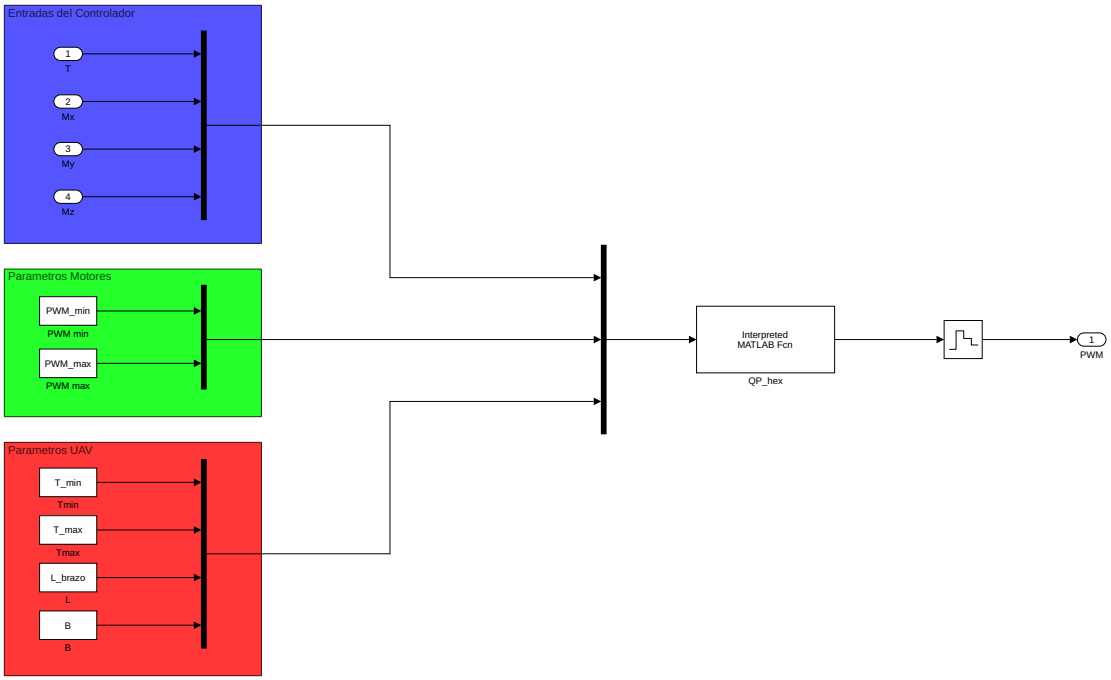

Figura 5.33 Interior del subsistema contenedor del *motor mixer*.

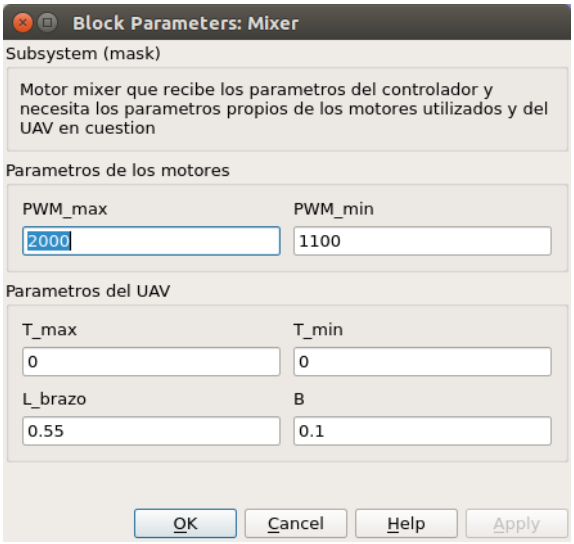

Figura 5.34 Máscara aplicada al mezclador.

Conclusiones y trabajo futuro

Si no sabes hacia que puerto zarpa tu barco, ningún viento te será favorable

SÉNECA

En este capitulo se mostrará una comparativa entre el autopiloto desarrollado durante este trabajo frente al que se ha estado utilizando hasta ahora, PX4. Con fin de ver la viabilidad de su sustitución. Ademas se desarrollarán las conclusiones y posibles mejoras y ampliaciones a lo ya realizado.

Comparación Autopiloto propio VS PX4

A continuación se mostrara una pequeña comparativa entre la salida del filtro de PX4, un más que conocido autopiloto y la salida del filtro realizado en este proyecto.

Comparación en actitud

En primer lugar se muestra la comparativa en actitud:

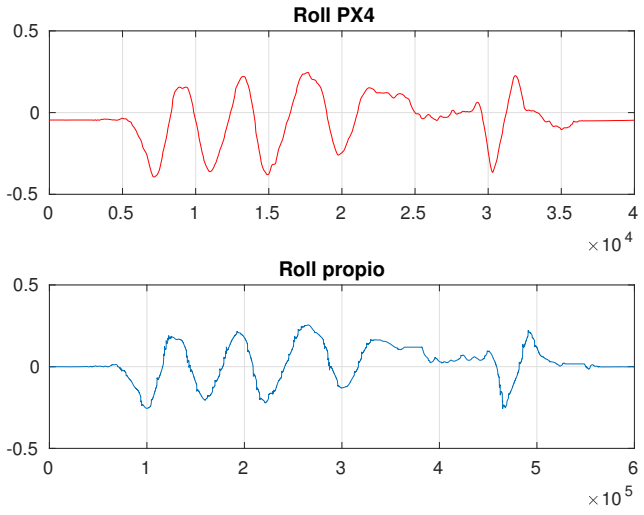

Figura 6.1 Comparación en Actitud de PX4 y el filtro propio, roll.

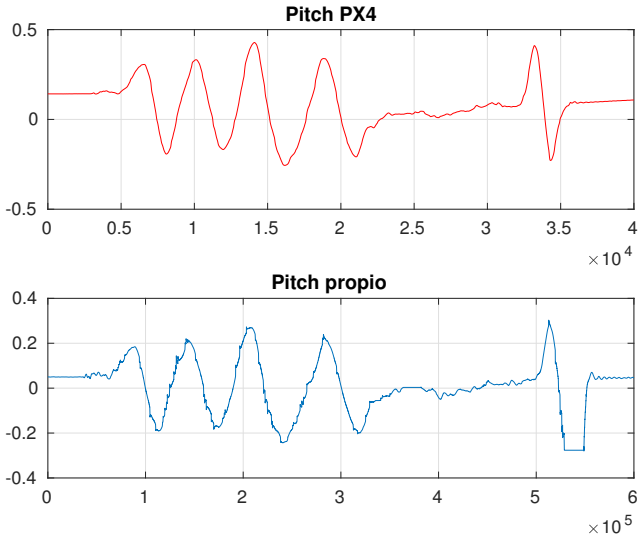

Figura 6.2 Comparación en Actitud de PX4 y el filtro propio, pitch.

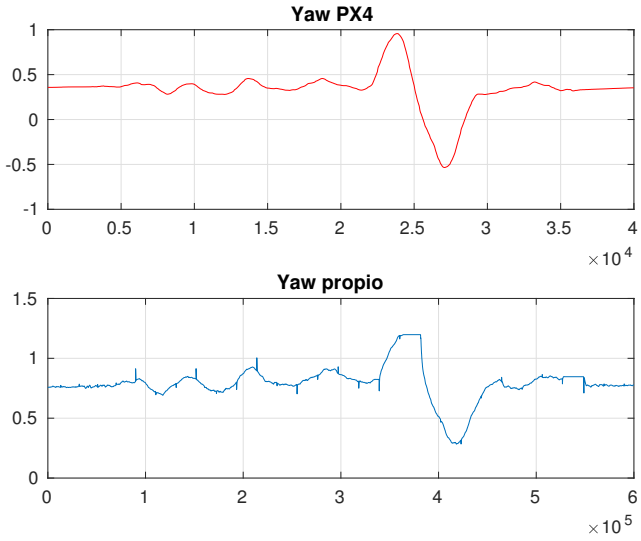

Figura 6.3 Comparación en Actitud de PX4 y el filtro propio, yaw.

En base a los resultados obtenidos anteriormente se ve el filtro desarrollado asemeja bastante bien el de PX4, sin embargo se observa como en el filtro propio es menos exacto en el sentido de que PX4 da valores mayores en los puntos máximos.

Otro detalle importante es que el filtro desarrollado es más ruidoso, sin embargo esto no se debe tomar como una realidad absoluta ya que el experimento se ha realizado con dos navios distintas ejecutando a la vez sendos algoritmos, sin embargo la correspondiente a la de PX4 se encontraba sobre una cama preparada para absorber las vibraciones y movimientos bruscos a modo de filtro paso bajo.

Conclusiones

Dado los resultados obtenidos en el proyecto creo que se puede concluir que la viabilidad de desarrollar un autopiloto propio desde cero es relativamente alta ya que se han conseguido resultados, que aunque no sean perfectos y necesiten bastante desarrollo y solución de algunos errores, sirven como punto de partida a un desarrollo mejorado.

De todo el proyecto se puede afirmar que la parte que ha resultado más complicada de llevar a cabo ha sido la estimación de la actitud del UAV ya que ha sido la parte que más investigación y pruebas a conllevado, aunque al final se han conseguido unos resultados considerablemente buenos, a falta de un mejor ajuste.

Por otro lado debido a la prioridad dada a conseguir una buena estimación en actitud se ha debido dejar en segundo plano el resto del proyecto lo que ha provocado tener que rebajar las pretensiones iniciales lo que ha provocado que el filtro de posición no pudiera ser bien ajustado obteniendo tan malos resultados aunque consiguiendo una aproximación suficientemente buena para una futura mejora.

A pesar de todo esto, el tiempo dedicado a la estimación de actitud a dado sus frutos ya que se ha conseguido, a falta un mejor ajuste, una aproximación bastante buena a lo que realiza PX4.

Trabajo futuro

Terminar filtro de posición

Como máxima prioridad de cara a futuras ampliaciones se propone completar el filtro de posición añadiendo a la estimación con el GPS la integración de las IMUs en lugar de mantenerlas completamente separadas con el objetivo de mejorar la deriva que de serie tienen ambas por separado. Además de una mejor estimación de la altura ya que actualmente se tiene demasiado rizado en la estimación.

Implementación de controlador

Lo primero que se debería hacer es probar el autopiloto desarrollado con un controlador real que le llegase a permitir volar de forma que se pudiera comprobar el correcto funcionamiento y la influencia o no de los posibles retrasos que pudieran existir así como del ruido de las medidas tomadas. Además esto permitiría poder realizar alguna prueba en vuelo.

Implementación del mezclador avanzado

Una vez que se tenga un controlador por sencillo que pudiera ser, algo interesante sería probar el mezclador comentado en la Subsección 5.5.2 donde se proponía la pseudoinversa como solución principal siempre y cuando no se produjeran saturaciones, en cuyo caso se realizaba la optimización de mínimos cuadrados de la diferencia entre el valor deseado y el más cercano posible con el objetivo de comprobar si los retrasos que genera la optimización son perceptibles.

En el caso de que no se pudiera ejecutar el mezclador propuesto se deberá pasar a calcular las ecuaciones de los planos correspondientes a los motores mostrados en la Figura 5.32 para realizar el cálculo del mezclador en función de los mismos y tratar de ahorrar tiempo de cómputo.

Mejor ajuste del estimador en el sistema real

Otra de las cosas que se podría llegar a mejorar una vez se disponga de un controlador y se pueda ejecutar el sistema completo es mejorar el ajuste de los filtros bien para mejorar los retrasos o los niveles de ruido. Por otro lado también sería interesante realizar algún programa con el que poder realizar la calibración de las IMUs de forma dinámica.

Mejoras en la seguridad del software

Por último sería altamente recomendable realizar test de esfuerzo de todo lo desarrollado para poner a prueba la estabilidad así como el correcto funcionamiento en diferentes situaciones. Por otro lado se debería de poder aprovechar la multiplicidad de procesos enviando las lecturas de los sensores.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] PX4. (2018, Marzo) Px4 pro autopilot software. [Online]. Available: https://github.com/PX4/Firmware

[2] Ardupilot. (2018, Marzo) Arduplane, arducopter, ardurover source. [Online]. Available: https://github.com/ArduPilot/ardupilot

[3] Emlid. (2018, Abril) Emlid. [Online]. Available: https://emlid.com/

[4] . PX4 Autopilot. (2018, Abril) Pixhaxk. [Online]. Available: https://pixhawk.org/

[5] . Qualcomm Technologies Inc. (2018, Abril) Snapdragon flight. [Online]. Available: https://developer.qualcomm.com/hardware/qualcomm-flight

[6] PX4. (2018, Marzo) Flight controller (autopilot) hardware. [Online]. Available: https://docs.px4.io/en/flight_controller/

[7] Ardupilot. (2018, Marzo) Ardupilot. [Online]. Available: https://en.wikipedia.org/wiki/ArduPilot

[8] Emlid. (2018, Marzo) Specifications section. [Online]. Available: https://emlid.com/navio/

[9] *MPU-9250 Product Specification*, InvenSense Inc., 6 2016, rev. 1.1. [Online]. Available: https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

[10] *LSM9DS1*, life.augmented, 12 2013, rev. 1. [Online]. Available: https://www.nordevx.com/datasheets/LSM9DS1-datasheet.pdf

[11] *MS5611-01BA03 Barometric Pressure Sensor*, Measurement Specialties ™, 10 2012. [Online]. Available: https://infusionsystems.com/support/MS5611-01BA03.pdf

[12] *NEO-M8*, Ublox, 10 2015, rev. 10. [Online]. Available: https://www.u-blox.com/sites/default/files/NEO-M8_DataSheet_(UBX-13003366).pdf

[13] Emlid. (2018, Abril) Raspberry pi configuration. [Online]. Available: https://docs.emlid.com/navio2/common/ardupilot/configuring-raspberry-pi/

[14] ——. (2018, Abril) C++ and python sensor examples for developers. [Online]. Available: https://github.com/emlid/Navio2

[15] Wikipedia. (2018, Abril) Raspberry pi. [Online]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi

[16] Emlid. (2018, Abril) Raspberry pi real-time kernel. [Online]. Available: https://emlid.com/raspberry-pi-real-time-kernel/

[17] Intel. (2018, Abril) Raspberry pi real-time kernel. [Online]. Available: https://www.intel.es/content/www/es/es/products/boards-kits/nuc.html

[18] S.-H. Haase. (2014, January) Alignment in c. [Online]. Available: https://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2013_2014/epc-14-haase-svenhendrik-alignmentinc-paper.pdf

[19] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960. [Online]. Available: https://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf

[20] A. Khosravian, J. Trumpf, R. Mahony, and T. Hamel, "Recursive attitude estimation in the presence of multi-rate and multi-delay vector measurements," pp. 3199–3205, July 2015.

[21] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft*. Hardcover, 2012.

[22] PX4. (2018, Junio) Aligntilt.m. [Online]. Available: https://github.com/PX4/ecl/blob/master/EKF/matlab/EKF_replay/Common/AlignTilt.m

[23] U. G. P. Office, "U.s. standard atmosphere," p. 241, 1976. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770009539.pdf

[24] PX4. (2018, Julio) Llh2ned. [Online]. Available: https://github.com/PX4/ecl/blob/372f9f430bb0edb901b3265ecd0cddf8ee4d1ffb/EKF/matlab/EKF_replay/Common/LLH2NED.m

[25] F. L. João C. Monteiro and L. Hsu, "Optimal control allocation of quadrotor uavs subject to actuator constraints," p. 500, 2016.

[26] G. J. J. Ducard and M.-D. Hua, "Discussion and practical aspects on control allocation for a multi-rotor helicopter," p. 95, 2011.