UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Final de Grado

Grado en Ingeniería Aeroespacial

DESARROLLO DE MIDDLEWARES PARA PILOTOS AUTOMÁTICOS DE BAJO COSTE

MANUAL DE PROGRAMACIÓN

Curso académico 2017-2018

Alumno: Andrés Masip, Miguel

Tutor: García-Nieto Rodríguez, Sergio

Índice general

Índice de figuras

Introducción

El presente documento recoge los aspectos principales para entender las modificaciones hechas en el codigo PX4 para el trabajo "Desarrollo de Middleware para pilotos automaticos de bajo coste".

Para el correcto seguimiento de este manual se deberá disponer de un mínimo de conocimientos sobre programación en C++, la arquitectura Linux y el entorno de compilación c-make. Además, será necesaria cierta familiaridad con conceptos sobre Fundamentos de Computadores y algún aspecto básico sobre los sensores presentes en un UAV (barómetros, giroscopos, acelerometros o magnetómetros).

El editor recomendado por la propia Developer Guide de PX4 sera el entorno QT Creator, aunque en este aspecto, casi todos los IDEs disponibles en el mercado poseen las funcionalidades demandadas para realizar los cambios propuestos.

Finalmente, como se ha detallado en la memoria de este proyecto o en el manual asociado, el acceso a la BeagleBone Blue se realizará mediante el entorno Cloud9. Esta característica puede no ser seguida por el usuario de este documento y en particular no condicionará los desarrollos propuestos en el archivo, ya que sólo se abarcará los cambios previos a la subida y ejecución de archivos.

Aspectos generales del PX4 Flight Stack

Entender la estructura general del código es fundamental para ser capaz de desarrollar el Middleware propuesto en este proyecto. Una vez conocida la misma, se localizarán los principales archivos que deben ser modificados para adaptarlos a las características básicas de la BeagleBone Blue. Estos archivos y sus funcionalidades quedan ejemplificados en el diagrama perteneciente a la figura 2.1.

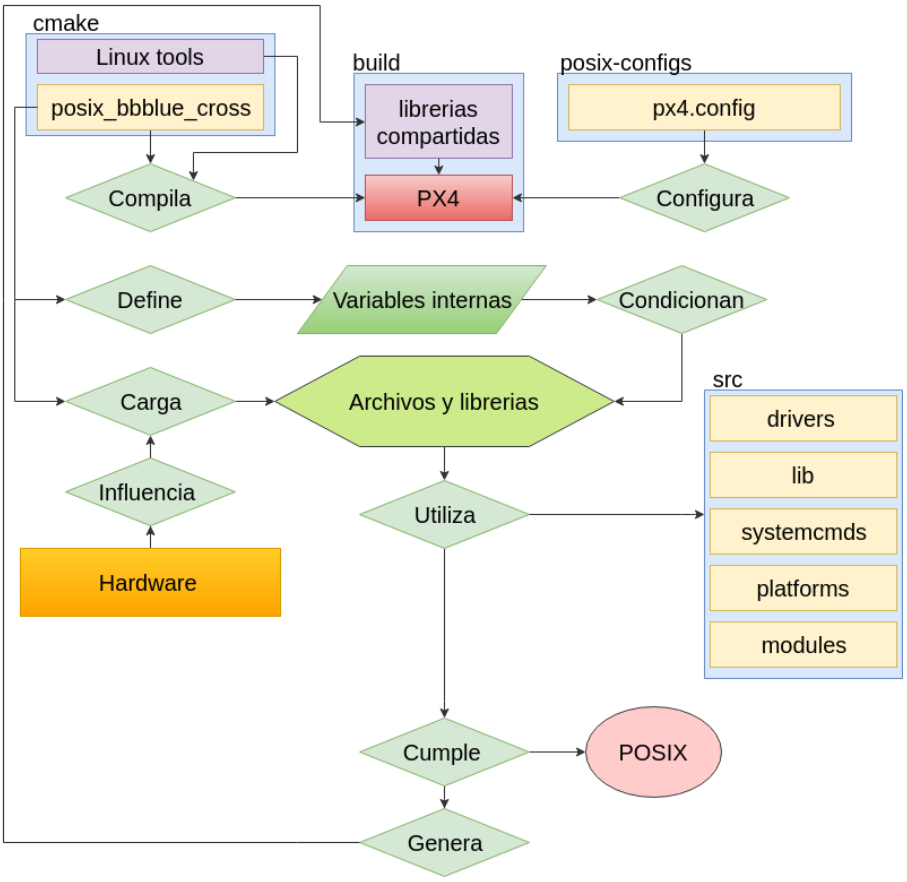

Figura 2.1: Diagrama de bloques para las intervenciones a realizar

El diagrama anterior favorece la comprensión de las modificaciones oportunas

aunque no es tan útil a nivel funcional como la figura 2.2 para entender el funcionamiento básico del código. Todos los bloques de la arquitectura se verán contenidos en el directorio src, puesto que será aquel que albergue las librería, comandos y módulos de interés. A lo largo de este proyecto se adaptará algún elemento de este directorio, así como se incluirán los archivos adecuados para la arquitectura BeagleBone Blue.

Las intervenciones según directorio de intervención[1] se detallaran de la siguiente forma:

- cmake: configuración de compilación. Carga drivers, modulos, comandos y añade definiciones básicas para la compilación.

- posix-configs: inicializa y configura el archivo una vez esta compilado. Documento que debe concordar con el archivo raíz de compilación y el hardware disponible.

- build: directorio objetivo de la compilación. Albergará el ejecutable y se construirá durante la fase de compilación. No se verá modificado ningún documento dentro de este directorio.

- src: carga todas las librerías, comandos y módulos necesarios. Dispone de muchas funcionalidades que se adaptarán a aquellos comandos y archivos incluidos por el documento de compilación. Se localizarán las mayores modificadas realizadas a nivel de programación.

La estructura por bloques, jerarquía y estándares (como POSIX) genera la posibilidad de extraer y configurar nuevos sensores o adaptar a otros protocolos los presentes en el DriverFramework con pequeñas modificaciones. Es decir, sólo se necesitará la adaptación de rutas y objetos de alto nivel, abstrayendo así la complejidad propia de registros y características de bajo nivel.

Puede ser interesante utilizar como modelo para los archivos a intervenir la Raspberry Pi con Navio 2 puesto que es la arquitectura disponible más parecida. Por otra parte, algunas modificaciones exigirán la generación de nuevos directorios o carpetas que serán particulares para cada hardware compatible. Se ha elegido el nombre de *bbblue* para estas disposiciones. Evidentemente, en el repositorio original previo a la realización de este proyecto estos directorios no existen y se han generado debido a la necesidad de ser añadidos para generar otra plataforma compatible.

[1] Evidentemente en algún directorio se realizarán diversas intervenciones

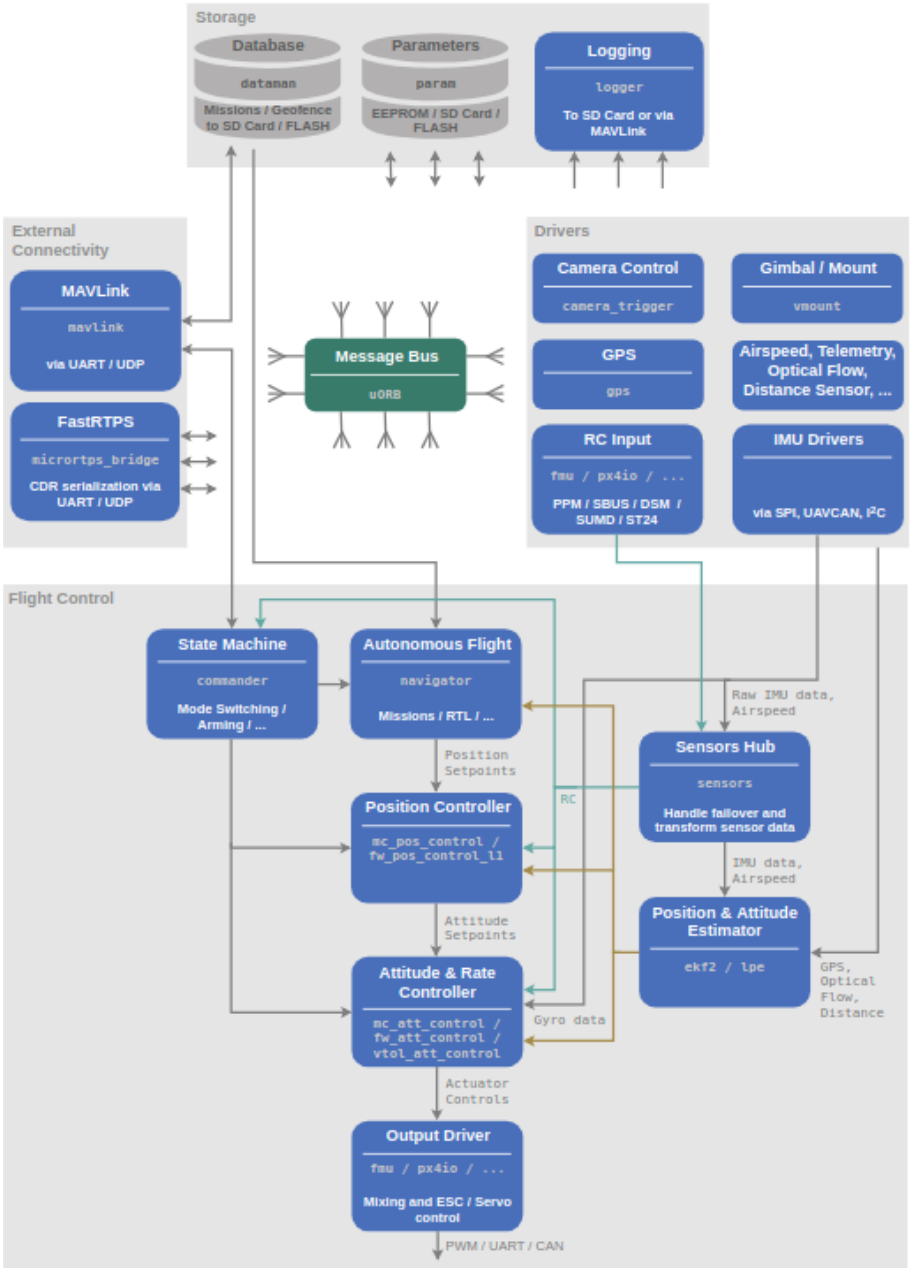

Figura 2.2: Arquitectura general del PX4 Flight Stack

Solución adoptada

En esta sección se detallarán todos los cambios e intervenciones realizadas en el código para generar las interfaces necesarias para poder ejecutar PX4 sobre la BeagleBone Blue. Las intervenciones de los archivos se detallarán en orden cronológico, no por importancia o directorio asociado.

Este documento se anexa a una memoria y a un manual de usuario. Algunos aspectos de programación o configuración final del código se han introducido en estos documentos. En este archivo solo se revisarán las intervenciones para compilar y hacer operativo un código PX4 sobre la BeagleBone Blue, sin repasar las condiciones de utilización o ejecución de la misma (manual de usuario). De la misma forma, en este archivo se adjuntarán códigos completos que por extensión no se han incluido en el documento principal o memoria.

Mapeado de la BeagleBone Blue

Archivo: board_config.h

Ruta: Firmware/src/drivers/boards/bbblue/

```
/**
* @file board_config.h
*
* BBBLUE internal definitions
*/

#pragma once

#define BOARD_OVERRIDE_UUID "BBBLUE   " // must be of length 12
    (PX4_CPU_UUID_BYTE_LENGTH)
#define BOARD_OVERRIDE_MFGUID BOARD_OVERRIDE_UUID

#define BOARD_NAME "BBBLUE"

#define ADC_BATTERY_VOLTAGE_CHANNEL 2
#define ADC_BATTERY_CURRENT_CHANNEL 3

#define BOARD_BATTERY1_V_DIV (10.177939394f)
#define BOARD_BATTERY1_A_PER_V (15.391030303f)

#define BOARD_MAX_LEDS 1 // Number external of LED's this board has

/*
* I2C busses
*/
#define PX4_I2C_BUS_EXPANSION 2
#define PX4_NUMBER_I2C_BUSES 1

#include <system_config.h>
#include "../common/board_common.h"
```

Cambios básicos e intervenciones que generan un archivo muy similar a otras arquitecturas Linux. Se ha definido el bus I2C, las entradas de voltaje y la ID relativa a la BeagleBone Blue.

3.1. Archivo de compilación

Archivo: posix_bbblue_common

Ruta: Firmware/cmake/common/

En este primer archivo, se inicializaran los módulos propios y necesarios para el código Esta parte es génerica y similar a otros dispositivos, particularizando aquellos sensores específicos. Las configuraciones relativas a *systemcmds* o *modules* no se han visto modificadas con respecto a otras arquitecturas Linux, ya que estos archivos cargarán funciones posteriores del código y no serán dependientes de la arquitectura de hardware presente[1].

```
# This file is shared between posix_bbblue_native.cmake
# and posix_bbblue_cross.cmake.

# This definition allows to differentiate if this just the usual POSIX
    build
# or if it is for the BBBLUE.
add_definitions(
-D__PX4_POSIX_BBBLUE
-D__DF_LINUX # For DriverFramework
-D__DF_BBBLUE # For DriverFramework
)

set(config_module_list
#
# Board support modules
#
#drivers/barometer
drivers/batt_smbus
drivers/differential_pressure
drivers/distance_sensor
#drivers/telemetry

modules/sensors

#platforms/posix/drivers/df_hmc5883_wrapper
#platforms/posix/drivers/df_isl29501_wrapper
#platforms/posix/drivers/df_lsm9ds1_wrapper
platforms/posix/drivers/df_bmp280_wrapper
platforms/posix/drivers/df_mpu9250_wrapper
#platforms/posix/drivers/df_trone_wrapper
```

[1]Constituyen parte de la capa Stack, la información que necesitarán ha sido procesada por otras partes del código a más bajo nivel.

```
#
# System commands
#
systemcmds/param
systemcmds/led_control
systemcmds/mixer
systemcmds/ver
systemcmds/esc_calib
systemcmds/reboot
systemcmds/topic_listener
systemcmds/tune_control
systemcmds/perf

#
# Estimation modules
#
modules/attitude_estimator_q
modules/position_estimator_inav
modules/local_position_estimator
modules/landing_target_estimator
modules/ekf2

#
# Vehicle Control
#
modules/fw_att_control
modules/fw_pos_control_l1
modules/gnd_att_control
modules/gnd_pos_control
modules/mc_att_control
modules/mc_pos_control
modules/vtol_att_control

#
# Library modules
#
modules/sdlog2
modules/logger
modules/commander
modules/dataman
modules/land_detector
modules/navigator
modules/mavlink
```

```
#
# PX4 drivers
#
drivers/linux_sbus
drivers/gps
#drivers/bbblue_adc
#drivers/navio_sysfs_rc_in
drivers/linux_gpio
drivers/linux_pwm_out
#drivers/navio_rgbled
drivers/pwm_out_sim
#drivers/rpi_rc_in

)

#
# DriverFramework driver
#
set(config_df_driver_list
bmp280
mpu9250
)
```

Como se puede observar en el código incluido, este archivo carga, define y marca las pautas del compilado del archivo. Se podrían ignorar todos aquellos módulos, sensores o drivers precedidos de almohadilla ya que no tendrán ninguna influencia en el código. Sin embargo, se incluirán todos y se comentarán o eliminarán las almohadillas en función de los archivos incluidos.

3.2. Archivo de configuración

Archivo:px4.config

Ruta: Firmware/posix-configs/bbblue

Este archivo activa los sensores, comandos y canales de comunicación en la etapa inicial. Se llamará junto al código principal al ejecutar el archivo y tendrá definido aquello previamente comentado, como serían los canales y puertos serie accedidos por el protocolo de comunicación o el orden de inicio de los sensores. Este archivo se puede ver modificado en disposiciones particulares, pudiendo por ejemplo cambiar los puertos de salida según se conecten para el UAV utilizado.

```
# bbblue config for a quad
uorb start
param load
param set SYS_AUTOSTART 4001
param set MAV_BROADCAST 1
param set MAV_TYPE 2
param set SYS_MC_EST_GROUP 2
param set BAT_CNT_V_VOLT 0.001
param set BAT_V_DIV 10.9176300578
param set BAT_CNT_V_CURR 0.001
param set BAT_A_PER_V 15.391030303
dataman start
#df_lsm9ds1_wrapper start -R 4
df_bmp280_wrapper start
df_mpu9250_wrapper start -R 10
#df_hmc5883_wrapper start
#df_ms5611_wrapper start
#navio_rgbled start
#gps start -d /dev/spidev0.0 -i spi -p ubx
#bbblue_adc start
sensors start
commander start
navigator start
ekf2 start
land_detector start multicopter
mc_pos_control start
mc_att_control start
mavlink start -x -u 14556 -r 1000000
mavlink stream -u 14556 -s HIGHRES_IMU -r 50
mavlink stream -u 14556 -s ATTITUDE -r 50
mavlink start -x -d /dev/ttyS3
mavlink stream -d /dev/ttyS3 -s HIGHRES_IMU -r 50
mavlink start -x -d /dev/ttyS2
mavlink stream -d /dev/ttyS2 -s ATTITUDE -r 50
#navio_sysfs_rc_in start
#linux_pwm_out start
logger start -t -b 200
mavlink boot_complete
```

3.3. Driver Framework

El DriverFramework es un marco genérico compatible con los estándares POSIX para generar e interactuar con los sensores presentes en el UAV donde se ejecute PX4. Sus principales ventajas se basan en disponer de archivos para gestionar los principales protocolos de comunicación (I2CDevObj,SPIDevObj) y sensores del mercado, haciendo viable pequeños ajustes para acomodar el driver disponible a las necesidades del proyecto. Este marco se ha descrito en profundidad a lo largo de la memoria principal anexada a este documento.

3.4. BMP280

Este sensor esta incluido y conectado mediante I2C en el DriverFramework. Sin embargo, se modificará el *device* que accede a este elemento puesto que estará mapeado de forma distinta en la BeagleBone Blue.

```
#if defined(__DF_BBBLUE)
#define BARO_DEVICE_PATH "/dev/i2c-2"
#else
#define BARO_DEVICE_PATH "/dev/iic-3"
#endif
```

Se comprueban los registros de acceso de este driver con la Datasheet o el comando i2cdump[2], donde se comprobará que coinciden con los presentes en:

Archivo: bmp280.cpp

Ruta: Firmware/src/lib/DriverFramework/drivers/bmp280

```
#define BMP280_REG_ID 0xD0
#define BMP280_REG_CTRL_MEAS 0xF4
#define BMP280_REG_CONFIG 0xF5
#define BMP280_REG_PRESS_MSB 0xF7
#define BMP280_ID 0x58

#define BMP280_BITS_CTRL_MEAS_OVERSAMPLING_TEMP2X 0b01000000
#define BMP280_BITS_CTRL_MEAS_OVERSAMPLING_PRESSURE8X 0b00010000
#define BMP280_BITS_CTRL_MEAS_POWER_MODE_NORMAL  0b00000011
#define BMP280_BITS_CONFIG_STANDBY_0MS5 0b00000000
#define BMP280_BITS_CONFIG_FILTER_OFF 0b00000000
#define BMP280_BITS_CONFIG_SPI_OFF 0b00000000
```

[2]Herramienta presente en los chips OMAP que permite obtener los registros activos del esclavo con la dirección introducida.

Archivo: bmp280.hpp

Ruta: Firmware/src/lib/DriverFramework/drivers/bmp280

```
// update frequency is 50 Hz (44.4-51.3Hz ) at 8x oversampling
#define BMP280_MEASURE_INTERVAL_US 20000

#define BMP280_BUS_FREQUENCY_IN_KHZ 400
#define BMP280_TRANSFER_TIMEOUT_IN_USECS 9000

#define BMP280_MAX_LEN_SENSOR_DATA_BUFFER_IN_BYTES 6
#define BMP280_MAX_LEN_CALIB_VALUES 26

// TODO: include some common header file (currently in drv_sensor.h).
#define DRV_DF_DEVTYPE_BMP280 0x42

#define BMP280_SLAVE_ADDRESS 0b1110110   /* 7-bit slave address */
```

Finalmente, la gestión de este sensor se realizará desde el archivo donde se llamará a los previamente introducidos. Es importante remarcar esta condición aunque realmente no se intervendrá en este archivo, puesto que sólo gestionará los objetos y la información obtenida del sensor.

Archivo: df_bmp280_wrapper

Ruta: Firmware/src/platforms/posix/drivers/

3.5. MPU9250

En el caso de la unidad inercial, serán necesarios más cambios. La estructura genérica de extracción de datos (variables internas) están ya desarrolladas en el DriverFramework. Sin embargo, el enlace SPI de este repositorio, no corresponde con el caso de la BeagleBone Blue.

Así en primer lugar se define y cambia la ruta de acceso. Proceso similar al caso anterior y favorecido por la posibilidad de comunicarse con una unidad inercial mediante un objeto I2C aunque no este adaptado para la IMU tratada. Por otra parte, en el segundo fragmento de este archivo se verá como se plantea un cambio de ruta e inclusión de nuevos archivos si se trata de la BeagleBone Blue.

Archivo: ImuSensor.hpp

Ruta: Firmware/src/lib/DriverFramework/framework/include

```
ImuSensor(const char *device_path, unsigned int sample_interval_usec,
    bool mag_enabled = false) :
#if defined(__IMU_USE_I2C)
I2CDevObj("ImuSensor", device_path, IMU_CLASS_PATH, sample_interval_usec),
#else
SPIDevObj("ImuSensor", device_path, IMU_CLASS_PATH, sample_interval_usec),
#endif
```

```
#if defined(__DF_BBBLUE)
#define __IMU_USE_I2C
#include "I2CDevObj.hpp"
#endif

#if defined(__IMU_USE_I2C)
#include "I2CDevObj.hpp"
#else
#include "SPIDevObj.hpp"
#endif

#if defined(__DF_QURT)
#include "dev_fs_lib_spi.h"
#define IMU_DEVICE_PATH "/dev/spi-1"
#elif defined(__DF_BEBOP)
#define IMU_DEVICE_PATH "/dev/i2c-mpu6050"
#elif defined(__DF_RPI)
#define IMU_DEVICE_PATH "/dev/spidev0.1"
#elif defined(__DF_EDISON)
#define IMU_DEVICE_PATH "/dev/spidev5.1"
#elif defined(__DF_OCPOC)
#define IMU_DEVICE_PATH "/dev/spidev1.0"
#elif defined(__DF_BBBLUE)
#define IMU_DEVICE_PATH "/dev/i2c-2"
#else
#define IMU_DEVICE_PATH "/dev/spidev0.0"
#endif
```

```
#if defined(__DF_RPI)
#include <linux/spi/spidev.h>
#define IMU_DEVICE_ACC_GYRO "/dev/spidev0.3"
#define IMU_DEVICE_MAG "/dev/spidev0.2"
#elif defined(__DF_RPI_SINGLE)
#define IMU_DEVICE_ACC_GYRO "/dev/spidev0.1"
#define IMU_DEVICE_MAG "/dev/spidev0.1"
#else
#define IMU_DEVICE_ACC_GYRO ""
#define IMU_DEVICE_MAG ""
#endif

#define IMU_CLASS_PATH "/dev/imu"
```

La siguiente modificación se basará en adaptar la comunicación SPI a I2C, por tanto, todo envío, registro y configuración se hará siguiendo las funciones empleadas en I2CDevObj[3].

Archivo: mpu9250.cpp

Ruta: Firmware/src/lib/DriverFramework/drivers/mpu9250

```
#if defined(__DF_BBBLUE)

#define __IMU_USE_I2C
#include <stdint.h>
#include <string.h>
#include "math.h"
#include "DriverFramework.hpp"
#include "MPU9250.hpp"
#include "MPU9250_mag.hpp"

#define MPU9250_SLAVE_ADDRESS 0x68

// update frequency 1000 Hz
#define MPU9250_MEASURE_INTERVAL_US 1000

#define MPU9250_BUS_FREQUENCY_IN_KHZ 400
#define MPU9250_TRANSFER_TIMEOUT_IN_USECS 9000


#define MPU9250_ONE_G 9.80665f
```

[3]Las diferencias I2CDevObj con SPIDevOBj han sido evidenciadas en el desarollo de la memoria principal.

```
#define MIN(_x, _y) (_x) > (_y) ? (_y) : (_x)

// Uncomment to allow additional debug output to be generated.
// #define MPU9250_DEBUG 1

using namespace DriverFramework;

int MPU9250::mpu9250_init()
{


/* Zero the struct */

m_sensor_data.accel_m_s2_x = 0.0f;
m_sensor_data.accel_m_s2_y = 0.0f;
m_sensor_data.accel_m_s2_z = 0.0f;
m_sensor_data.gyro_rad_s_x = 0.0f;
m_sensor_data.gyro_rad_s_y = 0.0f;
m_sensor_data.gyro_rad_s_z = 0.0f;
m_sensor_data.mag_ga_x = 0.0f;
m_sensor_data.mag_ga_y = 0.0f;
m_sensor_data.mag_ga_z = 0.0f;
m_sensor_data.temp_c = 0.0f;

m_sensor_data.read_counter = 0;
m_sensor_data.error_counter = 0;
m_sensor_data.fifo_overflow_counter = 0;
m_sensor_data.fifo_corruption_counter = 0;
m_sensor_data.gyro_range_hit_counter = 0;
m_sensor_data.accel_range_hit_counter = 0;

m_sensor_data.fifo_sample_interval_us = 0;
m_sensor_data.is_last_fifo_sample = false;

int result;
uint8_t bits = BIT_H_RESET;
result = _writeReg(MPUREG_PWR_MGMT_1, &bits, sizeof(bits));


if (result != 0) {
DF_LOG_ERR("reset failed");
}

usleep(100000);

DF_LOG_INFO("Reset MPU9250");
bits=0;
result = _writeReg(MPUREG_PWR_MGMT_1, &bits,sizeof(bits));
```

```
if (result != 0) {
DF_LOG_ERR("wakeup sensor failed");
}

usleep(1000);

result = _writeReg(MPUREG_PWR_MGMT_2, &bits,sizeof(bits));

if (result != 0) {
DF_LOG_ERR("enable failed");
}

// Se elimina el IF_DIS y el MST_RST



if (result != 0) {
}
usleep(1000);

// Reset and enable FIFO.
bits=BITS_USER_CTRL_FIFO_RST;
result = _writeReg(MPUREG_USER_CTRL,
&bits ,sizeof(bits));
bits=BITS_USER_CTRL_FIFO_EN;
result = _writeReg(MPUREG_USER_CTRL,
&bits ,sizeof(bits));
if (result != 0) {
DF_LOG_ERR("user ctrl 2 failed");
}

usleep(1000);

if (_mag_enabled) {
bits=BITS_FIFO_ENABLE_TEMP_OUT | BITS_FIFO_ENABLE_GYRO_XOUT
| BITS_FIFO_ENABLE_GYRO_YOUT
| BITS_FIFO_ENABLE_GYRO_ZOUT | BITS_FIFO_ENABLE_ACCEL
| BITS_FIFO_ENABLE_SLV0;
result = _writeReg(MPUREG_FIFO_EN,
&bits,sizeof(bits)); // SLV0 is configured for bulk transfer of mag data
    over I2C

} else {
bits=BITS_FIFO_ENABLE_TEMP_OUT | BITS_FIFO_ENABLE_GYRO_XOUT
| BITS_FIFO_ENABLE_GYRO_YOUT
| BITS_FIFO_ENABLE_GYRO_ZOUT | BITS_FIFO_ENABLE_ACCEL |
    BITS_ACCEL_CONFIG2_BW_41HZ;
result = _writeReg(MPUREG_FIFO_EN,
&bits,sizeof(bits));
```

```
DF_LOG_INFO("initializing mpu9250 driver without mag support");
}

if (result != 0) {
DF_LOG_ERR("FIFO enable failed");
}

usleep(1000);

/*
* A samplerate_divider of 0 should give 1000Hz:
*
* sample_rate = internal_sample_rate / (1+samplerate_divider)
*
* This is only used when FCHOICE is 0b11, FCHOICE_B (inverted) 0x00,
* therefore commented out.
*/
//uint8_t samplerate_divider = 0;
//result = _writeReg(MPUREG_FIFO_EN, samplerate_divider);
//if (result != 0) {
// DF_LOG_ERR("sample rate config failed");
//}
//usleep(1000);

#if defined(__DF_EDISON)
//Setting the gyro bandwidth to 250 Hz corresponds to
//8kHz sampling frequency which is too high for the Edison.
//Therefore, we use the gyro bandwith of 184 Hz which corresponds to 1kHz
    sampling frequency.
result = _writeReg(MPUREG_CONFIG,
BITS_DLPF_CFG_184HZ | BITS_CONFIG_FIFO_MODE_OVERWRITE,1);
#elif defined(__DF_RPI_SINGLE)
result = _writeReg(MPUREG_CONFIG,
BITS_DLPF_CFG_184HZ | BITS_CONFIG_FIFO_MODE_OVERWRITE,1);
#else
bits=BITS_DLPF_CFG_41HZ | BITS_CONFIG_FIFO_MODE_OVERWRITE;
result = _writeReg(MPUREG_CONFIG,&bits,sizeof(bits));
#endif

if (result != 0) {
DF_LOG_ERR("config failed");
}

usleep(1000);
bits=BITS_FS_2000DPS | BITS_BW_LT3600HZ;
result = _writeReg(MPUREG_GYRO_CONFIG, &bits ,sizeof(bits));
```

```
if (result != 0) {
DF_LOG_ERR("Gyro scale config failed");
}

usleep(1000);
bits=BITS_ACCEL_CONFIG_16G;
result = _writeReg(MPUREG_ACCEL_CONFIG, &bits,sizeof(bits));

if (result != 0) {
DF_LOG_ERR("Accel scale config failed");
}

usleep(1000);

bits=BITS_ACCEL_CONFIG2_BW_41HZ;

result = _writeReg(MPUREG_ACCEL_CONFIG2, &bits,sizeof(bits));

if (result != 0) {
DF_LOG_ERR("Accel scale config2 failed");
}

usleep(1000);

// Initialize the magnetometer inside the IMU, if enabled by the caller.
if (_mag_enabled && _mag == nullptr) {

if ((_mag = new MPU9250_mag(*this, MPU9250_MAG_SAMPLE_RATE_100HZ))
!= nullptr) {
// Initialize the magnetometer, providing the output data rate for
// data read from the IMU FIFO. This is used to calculate the I2C
// delay for reading the magnetometer.

result = _mag->initialize(MPU9250_MEASURE_INTERVAL_US);

if (result != 0) {
DF_LOG_ERR("Magnetometer initialization failed");
}

} else {
DF_LOG_ERR("Allocation of magnetometer object failed.");
}
}


// Enable/clear the FIFO of any residual data
reset_fifo();
```

```
// Clear Interrupt Status
clear_int_status();

return 0;
}

int MPU9250::mpu9250_deinit()
{
// Leave the IMU in a reset state (turned off).

uint8_t bits = BIT_H_RESET;
int result = _writeReg(MPUREG_PWR_MGMT_1,&bits,sizeof(bits));

if (result != 0) {
DF_LOG_ERR("reset failed");
}

// Deallocate the resources for the mag driver, if enabled.
if (_mag_enabled && _mag != nullptr) {
delete _mag;
_mag = nullptr;
}

return 0;
}

int MPU9250::start()
{
/* Open the device path specified in the class initialization. */
// attempt to open device in start()

int result = I2CDevObj::start();

if (result != 0) {
DF_LOG_ERR("DevObj start failed");
DF_LOG_ERR("Unable to open the device path: %s", m_dev_path);
return result;
}

result = _setSlaveConfig(MPU9250_SLAVE_ADDRESS,
MPU9250_BUS_FREQUENCY_IN_KHZ,
MPU9250_TRANSFER_TIMEOUT_IN_USECS);


if (result != 0) {
DF_LOG_ERR("Could not set slave config,%d",result);
}
```

```
uint8_t sensor_id;
result = _readReg(MPUREG_WHOAMI, &sensor_id, 4);


if (result != 0) {
DF_LOG_ERR("Unable to communicate with the MPU9250
    sensor.0x%X,0x%X,%d",MPUREG_WHOAMI,sensor_id,result);
goto exit;
}

if (MPU_WHOAMI_9250 != sensor_id && MPU_WHOAMI_9250_REAL != sensor_id) {
DF_LOG_ERR("MPU9250 sensor WHOAMI wrong: 0x%X, should be: 0x%X",
sensor_id, MPU_WHOAMI_9250);
result = -1;
goto exit;
}

result = mpu9250_init();

if (result != 0) {
DF_LOG_ERR("error: IMU sensor initialization failed, sensor read thread
    not started");
goto exit;
}
result = DevObj::start();

if (result != 0) {
DF_LOG_ERR("DevObj start failed");
return result;
}
exit:
return result;
}

int MPU9250::stop()
{
int result = mpu9250_deinit();

if (result != 0) {
DF_LOG_ERR(
"error: IMU sensor de-initialization failed.");
return result;
}

result = DevObj::stop();
```

```
if (result != 0) {
DF_LOG_ERR("DevObj stop failed");
return result;
}

// We need to wait so that all measure calls are finished before
// closing the device.
usleep(10000);

return 0;
}

int MPU9250::get_fifo_count()
{
int16_t num_bytes = 0;

int result = _readReg(MPUREG_FIFO_COUNTH, (uint8_t *) &num_bytes,
    sizeof(num_bytes));

if (result < 0) {
DF_LOG_ERR("FIFO count read failed");
return 0;
}

num_bytes = swap16(num_bytes);

return num_bytes;
}

void MPU9250::reset_fifo()
{

uint8_t bits = BITS_USER_CTRL_FIFO_RST | BITS_USER_CTRL_FIFO_EN;
int result = _writeReg(MPUREG_USER_CTRL, &bits, sizeof(bits));

if (result < 0) {
DF_LOG_ERR("FIFO reset failed");
}
}

void MPU9250::clear_int_status()
{
int result;
uint8_t int_status = 0;

result = _readReg(MPUREG_INT_STATUS, &int_status,sizeof(int_status));
```

```
if (result != 0) {
DF_LOG_ERR("Interrupt status clear failed");
}
}

void MPU9250::_measure()
{

uint8_t int_status = 0;
int result = _readReg(MPUREG_INT_STATUS, &int_status,sizeof(int_status));

if (result != 0) {
++m_sensor_data.error_counter;
return;
}

if (int_status & BITS_INT_STATUS_FIFO_OVERFLOW) {
reset_fifo();

++m_sensor_data.fifo_overflow_counter;
DF_LOG_ERR("FIFO overflow");

return;
}

int size_of_fifo_packet;

if (_mag_enabled) {
size_of_fifo_packet = sizeof(fifo_packet_with_mag);

} else {
size_of_fifo_packet = sizeof(fifo_packet);
}

// Get FIFO byte count to read and floor it to the report size.
int bytes_to_read = get_fifo_count() / size_of_fifo_packet
* size_of_fifo_packet;

// It looks like the FIFO doesn't actually deliver at 8kHz like it is
    supposed to.
// Therefore, we need to adapt the interval which we pass on to the
    integrator.
// The filtering is to lower the jitter that could result through the
    calculation
// because of the fact that the bytes we fetch per _measure() cycle
    varies.
_packets_per_cycle_filtered = (0.95f * _packets_per_cycle_filtered) +
    (0.05f * (bytes_to_read / size_of_fifo_packet));
```

```
if (bytes_to_read <= 0) {
++m_sensor_data.error_counter;
return;
}

// Allocate a buffer large enough for n complete packets, read from the
// sensor FIFO.
const unsigned buf_len = (MPU_MAX_LEN_FIFO_IN_BYTES /
    size_of_fifo_packet) * size_of_fifo_packet;
uint8_t fifo_read_buf[buf_len];

const unsigned read_len = MIN((unsigned)bytes_to_read, buf_len);
memset(fifo_read_buf, 0x0, buf_len);

result = _readReg(MPUREG_FIFO_R_W, fifo_read_buf, read_len);

if (result < 0) {
++m_sensor_data.error_counter;
return;
}
// According to the protocol specs, all sensor and interrupt registers
    may be read at 20 MHz.
// It is unclear what rate the FIFO register can be read at.
// If the FIFO buffer was read at 20 MHz, two effects were seen:
// - The buffer is off-by-one. So the report "starts" at
    &fifo_read_buf[i+1].
// - Also, the FIFO buffer seemed to prone to random corruption (or
    shifting), unless
//   all other sensors ran very smooth. (E.g. Changing the bus speed of
    the HMC5883 driver from
//   400 kHz to 100 kHz could cause corruption because this driver
    wouldn't run as regularly.
//
// Luckily 10 MHz seems to work fine.

for (unsigned packet_index = 0; packet_index < read_len /
    size_of_fifo_packet; ++packet_index) {

fifo_packet *report = (fifo_packet *)(&fifo_read_buf[packet_index *
    size_of_fifo_packet]);

/* TODO: add ifdef for endianness */
report->accel_x = swap16(report->accel_x);
report->accel_y = swap16(report->accel_y);
report->accel_z = swap16(report->accel_z);
report->temp = swap16(report->temp);
report->gyro_x = swap16(report->gyro_x);
report->gyro_y = swap16(report->gyro_y);
report->gyro_z = swap16(report->gyro_z);
```

```
// Check if the full accel range of the accel has been used. If this
    occurs, it is
// either a spike due to a crash/landing or a sign that the vibrations
    levels
// measured are excessive.
if (report->accel_x == INT16_MIN || report->accel_x == INT16_MAX ||
report->accel_y == INT16_MIN || report->accel_y == INT16_MAX ||
report->accel_z == INT16_MIN || report->accel_z == INT16_MAX) {
++m_sensor_data.accel_range_hit_counter;
}

// Also check the full gyro range, however, this is very unlikely to
    happen.
if (report->gyro_x == INT16_MIN || report->gyro_x == INT16_MAX ||
report->gyro_y == INT16_MIN || report->gyro_y == INT16_MAX ||
report->gyro_z == INT16_MIN || report->gyro_z == INT16_MAX) {
++m_sensor_data.gyro_range_hit_counter;
}

const float temp_c = float(report->temp) / 361.0f + 35.0f;

// Use the temperature field to try to detect if we (ever) fall out of
    sync with
// the FIFO buffer. If the temperature changes insane amounts, reset the
    FIFO logic
// and return early.
if (!_temp_initialized) {
// Assume that the temperature should be in a sane range of -40 to 85 deg
    C which is
// the specified temperature range, at least to initialize.
if (temp_c > -40.0f && temp_c < 85.0f) {

// Initialize the temperature logic.
_last_temp_c = temp_c;
DF_LOG_INFO("IMU temperature initialized to: %f", (double) temp_c);
_temp_initialized = true;
}

} else {
// Once initialized, check for a temperature change of more than 2
    degrees which
// points to a FIFO corruption.
if (fabsf(temp_c - _last_temp_c) > 2.0f) {
DF_LOG_ERR(
"FIFO corrupt, temp difference: %f, last temp: %f, current temp: %f",
(double)fabsf(temp_c - _last_temp_c), (double)_last_temp_c,
    (double)temp_c);
```

```
reset_fifo();
_temp_initialized = false;
++m_sensor_data.fifo_corruption_counter;
return;
}

_last_temp_c = temp_c;
}

m_sensor_data.accel_m_s2_x = float(report->accel_x)
* (MPU9250_ONE_G / 2048.0f);
m_sensor_data.accel_m_s2_y = float(report->accel_y)
* (MPU9250_ONE_G / 2048.0f);
m_sensor_data.accel_m_s2_z = float(report->accel_z)
* (MPU9250_ONE_G / 2048.0f);
m_sensor_data.temp_c = temp_c;
m_sensor_data.gyro_rad_s_x = float(report->gyro_x) * GYRO_RAW_TO_RAD_S;
m_sensor_data.gyro_rad_s_y = float(report->gyro_y) * GYRO_RAW_TO_RAD_S;
m_sensor_data.gyro_rad_s_z = float(report->gyro_z) * GYRO_RAW_TO_RAD_S;

if (_mag_enabled) {
struct fifo_packet_with_mag *report_with_mag_data = (struct
    fifo_packet_with_mag *)report;

int mag_error = _mag->process((const struct mag_data
    &)report_with_mag_data->mag_st1,
m_sensor_data.mag_ga_x,
m_sensor_data.mag_ga_y,
m_sensor_data.mag_ga_z);

if (mag_error == MAG_ERROR_DATA_OVERFLOW) {
m_sensor_data.mag_fifo_overflow_counter++;
}
}

// Pass on the sampling interval between FIFO samples at 8kHz.
m_sensor_data.fifo_sample_interval_us = 1000000 /
    MPU9250_MEASURE_INTERVAL_US
/ _packets_per_cycle_filtered;

// Flag if this is the last sample, and _publish() should wrap up the
    data it has received.
m_sensor_data.is_last_fifo_sample = ((packet_index + 1) == (read_len /
    size_of_fifo_packet));

++m_sensor_data.read_counter;
```

```
// Generate debug output every second, assuming that a sample is
    generated every
// 125 usecs
#ifdef MPU9250_DEBUG

if (++m_sensor_data.read_counter % (1000000 /
    m_sensor_data.fifo_sample_interval_us) == 0) {

DF_LOG_INFO("IMU: accel: [%f, %f, %f]",
(double)m_sensor_data.accel_m_s2_x,
(double)m_sensor_data.accel_m_s2_y,
(double)m_sensor_data.accel_m_s2_z);
DF_LOG_INFO("   gyro: [%f, %f, %f]",
(double)m_sensor_data.gyro_rad_s_x,
(double)m_sensor_data.gyro_rad_s_y,
(double)m_sensor_data.gyro_rad_s_z);
DF_LOG_INFO("  temp: %f C", (double)m_sensor_data.temp_c);
}

#endif

#ifdef MPU9250_DEBUG

if (_mag_enabled && mag_error == 0) {
if ((m_sensor_data.read_counter % 10000) == 0) {
DF_LOG_INFO("   mag: [%f, %f, %f] ga",
m_sensor_data.mag_ga_x, m_sensor_data.mag_ga_y, m_sensor_data.mag_ga_z);
}
}

#endif

_publish(m_sensor_data);
}
}
```

Además en el archivo al que se remite el expuesto previamente, se definirán las funciones que servirán para el los cambios propuestos en el archivo del magnetómetro. Por ejemplo, modificará _imu.ModifyReg a _imu.WriteReg según lo expuesto en el siguiente extracto.

Archivo: mpu9250.hpp

Ruta: Firmware/src/lib/DriverFramework/drivers/mpu9250

```
#pragma once

#include "ImuSensor.hpp"
#include "MPU9250_mag.hpp"

namespace DriverFramework
{
#define MPUREG_WHOAMI       0x75
#define MPUREG_SMPLRT_DIV   0x19
#define MPUREG_CONFIG       0x1A
#define MPUREG_GYRO_CONFIG  0x1B
#define MPUREG_ACCEL_CONFIG   0x1C
#define MPUREG_ACCEL_CONFIG2 0x1D
#define MPUREG_LPACCEL_ODR  0x1E
#define MPUREG_WOM_THRESH   0x1F
#define MPUREG_FIFO_EN      0x23
#define MPUREG_I2C_MST_CTRL   0x24
#define MPUREG_I2C_SLV0_ADDR 0x25
#define MPUREG_I2C_SLV0_REG   0x26
#define MPUREG_I2C_SLV0_CTRL 0x27
#define MPUREG_I2C_SLV1_ADDR 0x28
#define MPUREG_I2C_SLV1_REG   0x29
#define MPUREG_I2C_SLV1_CTRL 0x2A
#define MPUREG_I2C_SLV2_ADDR 0x2B
#define MPUREG_I2C_SLV2_REG   0x2C
#define MPUREG_I2C_SLV2_CTRL 0x2D
#define MPUREG_I2C_SLV3_ADDR 0x2E
#define MPUREG_I2C_SLV3_REG   0x2F
#define MPUREG_I2C_SLV3_CTRL 0x30
#define MPUREG_I2C_SLV4_ADDR 0x31
#define MPUREG_I2C_SLV4_REG   0x32
#define MPUREG_I2C_SLV4_DO  0x33
#define MPUREG_I2C_SLV4_CTRL 0x34
#define MPUREG_I2C_SLV4_DI  0x35
#define MPUREG_I2C_MST_STATUS 0x36
#define MPUREG_INT_PIN_CFG  0x37
#define MPUREG_INT_ENABLE  0x38
#define MPUREG_INT_STATUS  0x3A
#define MPUREG_ACCEL_XOUT_H   0x3B
#define MPUREG_ACCEL_XOUT_L   0x3C
#define MPUREG_ACCEL_YOUT_H   0x3D
#define MPUREG_ACCEL_YOUT_L   0x3E
#define MPUREG_ACCEL_ZOUT_H   0x3F
#define MPUREG_ACCEL_ZOUT_L   0x40
#define MPUREG_TEMP_OUT_H  0x41
#define MPUREG_TEMP_OUT_L  0x42
#define MPUREG_GYRO_XOUT_H  0x43
#define MPUREG_GYRO_XOUT_L  0x44
```

```
#define MPUREG_GYRO_YOUT_H  0x45
#define MPUREG_GYRO_YOUT_L  0x46
#define MPUREG_GYRO_ZOUT_H  0x47
#define MPUREG_GYRO_ZOUT_L  0x48
#define MPUREG_EXT_SENS_DATA_00 0x49
#define MPUREG_I2C_SLV0_DO  0x63
#define MPUREG_I2C_SLV1_DO  0x64
#define MPUREG_I2C_SLV2_DO  0x65
#define MPUREG_I2C_SLV3_DO  0x66
#define MPUREG_I2C_MST_DELAY_CTRL 0x67
#define MPUREG_SIGNAL_PATH_RESET 0x68
#define MPUREG_MOT_DETECT_CTRL 0x69
#define MPUREG_USER_CTRL    0x6A
#define MPUREG_PWR_MGMT_1  0x6B
#define MPUREG_PWR_MGMT_2  0x6C
#define MPUREG_FIFO_COUNTH  0x72
#define MPUREG_FIFO_COUNTL  0x73
#define MPUREG_FIFO_R_W    0x74

// Length of the FIFO used by the sensor to buffer unread
// sensor data.
#define MPU_MAX_LEN_FIFO_IN_BYTES 512

// Configuration bits MPU 9250
#define BIT_SLEEP     0x40
#define BIT_H_RESET      0x80
#define MPU_CLK_SEL_AUTO 0x01

#define BITS_USER_CTRL_FIFO_EN   0x40
#define BITS_USER_CTRL_FIFO_RST  0x04
#define BITS_USER_CTRL_I2C_MST_EN  0x20
#define BITS_USER_CTRL_I2C_IF_DIS  0x10
#define BITS_USER_CTRL_I2C_MST_RST  0x02

#define BITS_CONFIG_FIFO_MODE_OVERWRITE 0x00
#define BITS_CONFIG_FIFO_MODE_STOP 0x40

#define BITS_GYRO_ST_X     0x80
#define BITS_GYRO_ST_Y     0x40
#define BITS_GYRO_ST_Z     0x20
#define BITS_FS_250DPS     0x00
#define BITS_FS_500DPS     0x08
#define BITS_FS_1000DPS    0x10
#define BITS_FS_2000DPS    0x18
#define BITS_FS_MASK    0x18
// This is FCHOICE_B which is the inverse of FCHOICE
#define BITS_BW_3600HZ     0x02
```

```
// The FCHOICE bits are the same for all Bandwidths below 3600 Hz.
#define BITS_BW_LT3600HZ   0x00

#define BITS_DLPF_CFG_250HZ   0x00
#define BITS_DLPF_CFG_184HZ   0x01
#define BITS_DLPF_CFG_92HZ  0x02
#define BITS_DLPF_CFG_41HZ  0x03
#define BITS_DLPF_CFG_20HZ  0x04
#define BITS_DLPF_CFG_10HZ  0x05
#define BITS_DLPF_CFG_5HZ  0x06
#define BITS_DLPF_CFG_3600HZ 0x07
#define BITS_DLPF_CFG_MASK  0x07

#define BITS_FIFO_ENABLE_TEMP_OUT 0x80
#define BITS_FIFO_ENABLE_GYRO_XOUT 0x40
#define BITS_FIFO_ENABLE_GYRO_YOUT 0x20
#define BITS_FIFO_ENABLE_GYRO_ZOUT 0x10
#define BITS_FIFO_ENABLE_ACCEL   0x08
#define BITS_FIFO_ENABLE_SLV2  0x04
#define BITS_FIFO_ENABLE_SLV1  0x02
#define BITS_FIFO_ENABLE_SLV0  0x01

#define BITS_ACCEL_CONFIG_16G  0x18

// This is ACCEL_FCHOICE_B which is the inverse of ACCEL_FCHOICE
#define BITS_ACCEL_CONFIG2_BW_1130HZ 0x08
#define BITS_ACCEL_CONFIG2_BW_41HZ  0x03

#define BITS_I2C_SLV0_EN 0x80
#define BITS_I2C_SLV0_READ_8BYTES 0x08
#define BITS_I2C_SLV1_EN 0x80
#define BITS_I2C_SLV1_DIS 0x00
#define BITS_I2C_SLV2_EN 0x80
#define BITS_I2C_SLV4_EN 0x80
#define BITS_I2C_SLV4_DONE 0x40

#define BITS_SLV4_DLY_EN 0x10
#define BITS_SLV3_DLY_EN 0x08
#define BITS_SLV2_DLY_EN 0x04
#define BITS_SLV1_DLY_EN 0x02
#define BITS_SLV0_DLY_EN 0x01

#define BIT_RAW_RDY_EN   0x01
#define BIT_INT_ANYRD_2CLEAR 0x10

#define BITS_INT_STATUS_FIFO_OVERFLOW 0x10

#define BITS_I2C_MST_CLK_400_KHZ 0x0D
```

```
#ifndef M_PI_F
#define M_PI_F 3.14159265358979323846f
#endif

#if defined(__DF_EDISON)
// update frequency 250 Hz
#define MPU9250_MEASURE_INTERVAL_US 4000
#elif defined(__DF_RPI_SINGLE)
// update frequency 1000 Hz,if using rpi1,rpi zero,1000hz may be to
   higher,please reduce the frequency
#define MPU9250_MEASURE_INTERVAL_US 1000
#else
// update frequency 1000 Hz
#define MPU9250_MEASURE_INTERVAL_US 1000
#endif

// -2000 to 2000 degrees/s, 16 bit signed register, deg to rad conversion
#define GYRO_RAW_TO_RAD_S  (2000.0f / 32768.0f * M_PI_F / 180.0f)

// TODO: include some common header file (currently in drv_sensor.h).
#define DRV_DF_DEVTYPE_MPU9250 0x41

#define MPU_WHOAMI_9250    0x71 // 0x71
#define MPU_WHOAMI_9250_REAL  0x73 // 0x73

#pragma pack(push, 1)
struct fifo_packet {
int16_t accel_x;
int16_t accel_y;
int16_t accel_z;
int16_t temp;
int16_t gyro_x;
int16_t gyro_y;
int16_t gyro_z;
};
struct fifo_packet_with_mag {
int16_t accel_x;
int16_t accel_y;
int16_t accel_z;
int16_t temp;
int16_t gyro_x;
int16_t gyro_y;
int16_t gyro_z;
char mag_st1; // 14 mag ST1 (1B)
int16_t mag_x; // 15-16 (2B)
int16_t mag_y; // 17-18 (2B)
int16_t mag_z; // 19-20 (2B)
char mag_st2; // 21 mag ST2 (1B)
};
```

```
// This data structure is a copy of the segment of the above
    fifo_packet_with_mag data
// struture that contains mag data.
struct mag_data {
char mag_st1; // mag ST1 (1B)
int16_t mag_x; // uT (2B)
int16_t mag_y; // uT (2B)
int16_t mag_z; // uT (2B)
char mag_st2; // mag ST2 (1B)
};
#pragma pack(pop)

class MPU9250: public ImuSensor
{
public:
MPU9250(const char *device_path, bool mag_enabled = false) :
ImuSensor(device_path, MPU9250_MEASURE_INTERVAL_US, mag_enabled), // true
    = mag is enabled
_last_temp_c(0.0f),
_temp_initialized(false),
_mag_enabled(mag_enabled),
#if defined(__DF_EDISON)
_packets_per_cycle_filtered(4.0f), // The FIFO is supposed to run at 1kHz
    and we sample at 250Hz.
#else
_packets_per_cycle_filtered(8.0f), // The FIFO is supposed to run at 8kHz
    and we sample at 1kHz.
#endif
_mag(nullptr)
{
m_id.dev_id_s.devtype = DRV_DF_DEVTYPE_MPU9250;
// TODO: does the WHOAMI make sense as an address?
m_id.dev_id_s.address = MPU_WHOAMI_9250;
}
#if defined(__DF_BBBLUE)

// @return 0 on success, -errno on failure
int WriteReg(uint8_t reg, uint8_t *val, int time)
{
return _writeReg(reg, val, time);
}

int ReadReg(uint8_t address, uint8_t *val,int time)
{
return _readReg(address, val, time);
}
```

```
int SetSlaveConfig(uint32_t slave_address, uint32_t bus_frequency_khz,
    uint32_t transfer_timeout_usec)
{
return _setSlaveConfig(slave_address, bus_frequency_khz,
    transfer_timeout_usec);
}
//CUIDADO CON EL MODIFYREG

// @return 0 on success, -errno on failure
virtual int start() override;

// @return 0 on success, -errno on failure
virtual int stop() override;

#else

// @return 0 on success, -errno on failure
int writeReg(int reg, uint8_t val)
{
return _writeReg(reg, val);
}

int readReg(uint8_t address, uint8_t &val)
{
return _readReg(address, val);
}

int modifyReg(uint8_t address, uint8_t clearbits, uint8_t setbits)
{
return _modifyReg(address, clearbits, setbits);
}

// @return 0 on success, -errno on failure
virtual int start() override;

// @return 0 on success, -errno on failure
virtual int stop() override;

#endif

protected:
virtual void _measure() override;
virtual int _publish(struct imu_sensor_data &data) = 0;

private:
// @returns 0 on success, -errno on failure
int mpu9250_init();
```

```
// @returns 0 on success, -errno on failure
int mpu9250_deinit();

// @return the number of FIFO bytes to collect
int get_fifo_count();

void reset_fifo();

void clear_int_status();

float _last_temp_c;
bool _temp_initialized;
bool _mag_enabled;
float _packets_per_cycle_filtered;

MPU9250_mag *_mag;
};

}
// namespace DriverFramework
```

El archivo del magnetómetro iniciará la estructura _imu. En la construcción y escritura/lectura de registros utilizará las definiciones expuestas en el archivo previo, particularizado para la BeagleBone Blue. No sufrirá modificaciones tan grandes como el archivo mpu9250.cpp, sino que será un cambio en las funciones de interacción para ser coherentes y algún retoque o recombinación de funciones[4] para evitar errores. Además de forma similar al caso anterior, se comprobarán los registros y direcciones por defecto localizados en el archivo con aquellos dispuestos en la bibliografía[5].

Archivo: mpu9250_mag.cpp

Ruta: Firmware/src/lib/DriverFramework/drivers/mpu9250

```
#if defined(__DF_BBBLUE)

#include "MPU9250.hpp"
#include "MPU9250_mag.hpp"


using namespace DriverFramework;

// Uncomment to allow additional debug output to be generated.
// #define MPU9250_MAG_DEBUG 1
```

[4]Errores encontrados mediante un procedimiento de *debug*
[5]Ardupilot, Robotics Cape o Datasheet.

```
int MPU9250_mag::_convert_sample_rate_enum_to_hz(
enum mag_sample_rate_e sample_rate)
{
switch (sample_rate) {
case MPU9250_MAG_SAMPLE_RATE_100HZ:
return 100;

case MPU9250_MAG_SAMPLE_RATE_8HZ:
return 8;

default:
DF_LOG_ERR("Invalid mag sample rate detected.");
return -1;
}
}

int MPU9250_mag::_initialize(int output_data_rate_in_hz)
{

// Configure the IMU as an I2C master at 400 KHz
int result;
uint8_t bits= BITS_USER_CTRL_I2C_MST_EN;
result = _imu.WriteReg(MPUREG_USER_CTRL, &bits , sizeof(bits));

if (result != 0) {
DF_LOG_ERR("IMU I2C master enable failed.
    0x%X,0x%X",MPUREG_USER_CTRL,bits);
return -1;
}

bits = BITS_I2C_MST_CLK_400_KHZ;

result = _imu.WriteReg(MPUREG_I2C_MST_CTRL, &bits, sizeof(bits));

if (result != 0) {
DF_LOG_ERR("IMU I2C master bus config failed.");
return -1;
}

usleep(1000);

// First set power-down mode
result = write_reg(MPU9250_MAG_REG_CNTL2, BIT_MAG_CNTL2_SOFT_RESET);
```

```
if (result != 0) {
DF_LOG_ERR("MPU9250 soft reset failed.");
// Reset i2c master.
bits=BITS_USER_CTRL_I2C_MST_RST;

_imu.WriteReg(MPUREG_USER_CTRL,&bits,sizeof(bits)) ;
return -1;
}

usleep(1000);

// Detect mag presence by reading whoami regist


uint8_t b;
b=0;
// get mag version ID
int retVal = read_reg(MPU9250_MAG_REG_WIA, &b);

if (retVal != 0) {
DF_LOG_ERR("error reading mag whoami reg: %d", retVal);
return -1;
}

if (b != MPU9250_AKM_DEV_ID) {
DF_LOG_ERR("wrong mag ID %u (expected %u)", b, MPU9250_AKM_DEV_ID);
return -1;
}

// Get mag calibraion data from Fuse ROM
if (get_sensitivity_adjustment() != 0) {
DF_LOG_ERR("Unable to read mag sensitivity adjustment");
return -1;
}

// Power on and configure the mag to produce 16 bit data in continuous
   measurement mode.
int mag_mode;

if (_sample_rate == MPU9250_MAG_SAMPLE_RATE_100HZ) {
mag_mode = BIT_MAG_CNTL1_MODE_CONTINUOUS_MEASURE_MODE_2;

} else if (_sample_rate == MPU9250_MAG_SAMPLE_RATE_8HZ) {
mag_mode = BIT_MAG_CNTL1_MODE_CONTINUOUS_MEASURE_MODE_1;

} else {
DF_LOG_ERR("Unable to select a valid mag mode.");
return -1;
}
```

```
result = write_reg(MPU9250_MAG_REG_CNTL1, BIT_MAG_CNTL1_16_BITS |
    mag_mode);

if (result != 0) {
DF_LOG_ERR("Unable to configure the magnetometer mode.");
}

usleep(1000);

// Slave 0 provides ST1, mag data, and ST2 data in a bulk transfer of
// 8 bytes of data. Use the address of ST1 in SLV0_REG as the beginning
// register of the 8 byte bulk transfer.
bits=MPU9250_AK8963_I2C_ADDR | MPU9250_AK8963_I2C_READ;

result = _imu.WriteReg(MPUREG_I2C_SLV0_ADDR,
&bits,sizeof(bits));

if (result != 0) {
DF_LOG_ERR("MPU9250 I2C slave 0 address configuration failed.");
return -1;
}

bits=MPU9250_MAG_REG_ST1;

result = _imu.WriteReg(MPUREG_I2C_SLV0_REG, &bits, sizeof(bits));

if (result != 0) {
DF_LOG_ERR("MPU9250 I2C slave 0 register configuration failed.");
return -1;
}


bits=BITS_I2C_SLV0_EN | BITS_I2C_SLV0_READ_8BYTES;
result = _imu.WriteReg(MPUREG_I2C_SLV0_CTRL, &bits, sizeof(bits));

if (result != 0) {
DF_LOG_ERR("MPU9250 I2C slave 0 control configuration failed.");
return -1;
}

usleep(1000);
```

```
// Enable reading of the mag every n samples, dividing down from the
// output data rate provided by the caller.
int sample_rate_in_hz = _convert_sample_rate_enum_to_hz(_sample_rate);

if (sample_rate_in_hz <= 0) {
DF_LOG_ERR("Unable to convert the requested mag sample rate to Hz.");
return -1;
}

uint8_t i2c_mst_delay = output_data_rate_in_hz / sample_rate_in_hz;
bits=i2c_mst_delay;
result = _imu.WriteReg(MPUREG_I2C_SLV4_CTRL, &bits, sizeof(bits));

if (result != 0) {
DF_LOG_ERR(
"Unable to configure the I2C delay from the configured output data
    rate.");
return -1;
}

usleep(1000);

// Enable delayed I2C transfers for the mag on Slave 0 registers.
bits=BITS_SLV0_DLY_EN;
result = _imu.WriteReg(MPUREG_I2C_MST_DELAY_CTRL, &bits,sizeof(bits));

if (result != 0) {
DF_LOG_ERR("Unable to enable the I2C delay on slave 0.");
return -1;
}

usleep(1000);

return 0;
}

int MPU9250_mag::initialize(int output_data_rate_in_hz)
{
// Retry up to 5 times to ensure successful initialization of the
// sensor's internal I2C bus.
int init_max_tries = 5;
int ret = 0;
int i;

for (i = 0; i < init_max_tries; i++) {
#ifdef MPU9250_MAG_DEBUG
DF_LOG_INFO("Calling _initialize(%d)", output_data_rate_in_hz);
#endif
ret = _initialize(output_data_rate_in_hz);
```

```
if (ret == 0) {
break;
}

DF_LOG_ERR("mag initialization failed %d tries", i + 1);
usleep(10000);
}

if (ret == 0) {
#ifdef MPU9250_MAG_DEBUG
DF_LOG_INFO("mag initialization succ after %d retries", i);
#endif
_mag_initialized = true;

} else {
DF_LOG_ERR("failed to initialize mag!");
}

return ret;
}

int MPU9250_mag::get_sensitivity_adjustment()
{
// First set power-down mode
if (write_reg(MPU9250_MAG_REG_CNTL1, BIT_MAG_CNTL1_MODE_POWER_DOWN) != 0)
    {
return -1;
}
usleep(10000);

// Enable FUSE ROM, since the sensitivity adjustment data is stored in
// compass registers 0x10, 0x11 and 0x12 which is only accessible in Fuse
// access mode.
if (write_reg(MPU9250_MAG_REG_CNTL1,
BIT_MAG_CNTL1_16_BITS | BIT_MAG_CNTL1_FUSE_ROM_ACCESS_MODE) != 0) {
return -1;
}

usleep(10000);

// Get compass calibration register 0x10, 0x11, 0x12
// store into context.
for (int i = 0; i < 3; ++i) {

uint8_t asa;
```

```
if (read_reg(MPU9250_MAG_REG_ASAX + i, &asa) != 0) {
return -1;
}

// H_adj = H * ((ASA-128)*0.5/128 + 1)
//       = H * ((ASA-128) / 256 + 1)
// H is the raw compass reading.
_mag_sens_adj[i] = (((float)asa - 128.0f) / 256.0f) + 1.0f;
}

// Leave in a power-down mode
if (write_reg(MPU9250_MAG_REG_CNTL1, BIT_MAG_CNTL1_MODE_POWER_DOWN) != 0)
    {
return -1;
}

usleep(10000);

#ifdef MPU9250_MAG_DEBUG
DF_LOG_INFO("magnetometer sensitivity adjustment: %.3f %.3f %.3f",
_mag_sens_adj[0], _mag_sens_adj[1], _mag_sens_adj[2]);
#endif

return 0;
}



int MPU9250_mag::write_imu_reg_verified(int reg, uint8_t val, uint8_t
    mask)
{
int retVal;
uint8_t b;
int retry = 5;
bool err_seen;


uint8_t val1 = val;


while (retry) {
err_seen = false;
--retry;
retVal = _imu.WriteReg(reg, &val1, sizeof(val1));

if (retVal != 0) {
err_seen = true;
continue;
}
```

```
retVal = _imu.ReadReg(reg, &b, sizeof(b));

if (retVal != 0) {
err_seen = true;
continue;
}

if ((b & mask) != val) {
continue;

} else {
#ifdef MPU9250_MAG_DEBUG
DF_LOG_INFO("set_mag_reg_verified succ for reg %d=%d", reg, val);
#endif
return 0;
}
}

if (err_seen) {
DF_LOG_ERR("set_mag_reg_verified failed for reg %d. Error %d.",
reg, retVal);

} else {
DF_LOG_ERR("set_mag_reg_verified failed for reg %d. %d!=%d",
reg, val, b);
}

return retVal;
}

int MPU9250_mag::read_reg(uint8_t reg, uint8_t *val)
{
int retVal = 0;
uint8_t b = 0;

// Read operation on the mag using the slave 4 registers.
retVal = write_imu_reg_verified(MPUREG_I2C_SLV4_ADDR,
MPU9250_AK8963_I2C_ADDR | MPU9250_AK8963_I2C_READ, 0xff);

if (retVal != 0) {
return retVal;
}

// Set the mag register to read from.
retVal = write_imu_reg_verified(MPUREG_I2C_SLV4_REG, reg, 0xff);
```

```
if (retVal != 0) {
return retVal;
}

// Read the existing value of the SLV4 control register.
retVal = _imu.ReadReg(MPUREG_I2C_SLV4_CTRL, &b, sizeof(b));

if (retVal != 0) {
return retVal;
}

// Set the I2C_SLV4_EN bit in I2C_SL4_CTRL register without overwriting
    other
// bits. Enable data transfer, a read transfer as configured above.
b |= BITS_I2C_SLV4_EN;
// Trigger the data transfer
retVal = _imu.WriteReg(MPUREG_I2C_SLV4_CTRL, &b, sizeof(b));

if (retVal != 0) {
return retVal;
}

// Continuously check I2C_MST_STATUS register value for the completion
// of I2C transfer until timeout

int loop_ctrl = 1000; // wait up to 1000 * 1ms for completion

do {
usleep(1000);
retVal = _imu.ReadReg(MPUREG_I2C_MST_STATUS, &b, sizeof(b));

if (retVal != 0) {
return retVal;
}
} while (((b & BITS_I2C_SLV4_DONE) == 0x00) && (--loop_ctrl));

if (loop_ctrl == 0) {
DF_LOG_ERR("I2C transfer timed out");
return -1;
}

// Read the value received from the mag, and copy to the caller's out
    parameter.
retVal = _imu.ReadReg(MPUREG_I2C_SLV4_DI, val, sizeof(val));

if (retVal != 0) {
return retVal;
}
```

```
#ifdef MPU9250_MAG_DEBUG
DF_LOG_INFO("Mag register %u read returned %u", reg, *val);
#endif

return 0;
}

int MPU9250_mag::write_reg(uint8_t reg, uint8_t val)
{
int retVal = 0;
uint8_t b = 0;

// Configure a write operation to the mag using Slave 4.
retVal = write_imu_reg_verified(MPUREG_I2C_SLV4_ADDR,
MPU9250_AK8963_I2C_ADDR, 0xff);

if (retVal != 0) {
return retVal;
}

// Set the mag register address to write to using Slave 4.
retVal = write_imu_reg_verified(MPUREG_I2C_SLV4_REG, reg, 0xff);

if (retVal != 0) {
return retVal;
}

// Set the value to write in the I2C_SLV4_DO register.
retVal = write_imu_reg_verified(MPUREG_I2C_SLV4_DO, val, 0xff);

if (retVal != 0) {
return retVal;
}

// Read the current value of the Slave 4 control register.
retVal = _imu.ReadReg(MPUREG_I2C_SLV4_CTRL, &b, sizeof(b));

if (retVal != 0) {
return retVal;
}

// Set I2C_SLV4_EN bit in I2C_SL4_CTRL register without overwriting other
// bits.
b |= BITS_I2C_SLV4_EN;
// Trigger the data transfer from the byte now stored in the SLV4_DO
   register.
retVal = _imu.WriteReg(MPUREG_I2C_SLV4_CTRL, &b, sizeof(b));
```

```
if (retVal != 0) {
return retVal;
}

// Continuously check I2C_MST_STATUS regsiter value for the completion
// of I2C transfer until timeout.

int loop_ctrl = 1000; // wait up to 1000 * 1ms for completion

do {
usleep(1000);
retVal = _imu.ReadReg(MPUREG_I2C_MST_STATUS, &b,sizeof(b));

if (retVal != 0) {
return retVal;
}
} while (((b & BITS_I2C_SLV4_DONE) == 0x00) && (--loop_ctrl));

if (loop_ctrl == 0) {
DF_LOG_ERR("I2C transfer to mag timed out");
return -1;
}

#ifdef MPU9250_MAG_DEBUG
DF_LOG_INFO("Magnetometer register %u set to %u", reg, val);
#endif
return 0;
}

int MPU9250_mag::process(const struct mag_data &data, float &mag_ga_x,
    float &mag_ga_y, float &mag_ga_z)
{
#ifdef MPU9250_MAG_DEBUG
static int hofl_bit_counter = 0;
#endif

// Check magnetic sensor overflow HOFL bit set. No need to check the data
    ready bit, since
// the sample rate divider should provide new samples at the correct
    interval.
if (data.mag_st2 & BIT_MAG_HOFL) {
#ifdef MPU9250_MAG_DEBUG

if ((++hofl_bit_counter % 1000) == 0) {
DF_LOG_ERR("overflow HOFL bit set (x1000)");
}

#endif
```

```
return MAG_ERROR_DATA_OVERFLOW;
}

#ifdef MPU9250_MAG_DEBUG
//DF_LOG_INFO("Raw mag data: [%d, %d, %d]", data.mag_x, data.mag_y,
    data.mag_z);
#endif
// _mag_sens_adj[i] = (((float) asa[i] - 128.0) / 256.0) + 1.0f;

mag_ga_x = data.mag_x * _mag_sens_adj[0] * MAG_RAW_TO_GAUSS;
mag_ga_y = data.mag_y * _mag_sens_adj[1] * MAG_RAW_TO_GAUSS;
mag_ga_z = data.mag_z * _mag_sens_adj[2] * MAG_RAW_TO_GAUSS;


// Swap magnetometer x and y axis, and invert z because internal mag in
    MPU9250
// has a different orientation.
// Magnetometer X axis = Gyro and Accel Y axis
// Magnetometer Y axis = Gyro and Accel X axis
// Magnetometer Z axis = -Gyro and Accel Z axis
float temp_mag_x = mag_ga_x;
mag_ga_x = mag_ga_y;
mag_ga_y = temp_mag_x;

mag_ga_z = -mag_ga_z;

return 0;
}
```

3.6. Data validator

La compilación y ejecución del código puede llevar a errores relacionados con el tratamiento y verificación del código. Para el caso abordado se generó en primera instancia un error relacionado con el tiempo de lectura y actualización, el error $ACCEL\#0 : TIMEOUT$. En el siguiente extracto se puede establecer el origen del error en consola, una bandera que salta al comprobar el archivo referente a los sensores:

Archivo: voted_sensors_update.cpp

Ruta: Firmware/src/modules/sensors

```
bool VotedSensorsUpdate::check_failover(SensorData &sensor, const char
    *sensor_name)
{
if (sensor.last_failover_count != sensor.voter.failover_count()) {

uint32_t flags = sensor.voter.failover_state();

if (flags == DataValidator::ERROR_FLAG_NO_ERROR) {
int failover_index = sensor.voter.failover_index();

if (failover_index != -1) {
//we switched due to a non-critical reason. No need to panic.
PX4_INFO("%s sensor switch from #%i", sensor_name, failover_index);
}

} else {
int failover_index = sensor.voter.failover_index();

if (failover_index != -1) {
mavlink_log_emergency(&_mavlink_log_pub, "%s #%i fail: %s%s%s%s%s!",
sensor_name,
failover_index,
((flags & DataValidator::ERROR_FLAG_NO_DATA) ? " OFF" : ""),
((flags & DataValidator::ERROR_FLAG_STALE_DATA) ? " STALE" : ""),
((flags & DataValidator::ERROR_FLAG_TIMEOUT) ? " TIMEOUT" : ""),
((flags & DataValidator::ERROR_FLAG_HIGH_ERRCOUNT) ? " ERR CNT" : ""),
((flags & DataValidator::ERROR_FLAG_HIGH_ERRDENSITY) ? " ERR DNST" : ""));

// reduce priority of failed sensor to the minimum
sensor.priority[failover_index] = 1;
}
}

sensor.last_failover_count = sensor.voter.failover_count();
return true;
}

return false;
}
```

El lugar exacto donde se origina el error sera el siguiente documento, en cual lecturas incoherentes de los sensores generan que se superen los límites presentes y se activen las banderas.

Archivo: data_validator.cpp

Ruta: Firmware/src/lib/ecl/validation

```
float
DataValidator::confidence(uint64_t timestamp)
{
float ret = 1.0f;

/* check if we have any data */
if (_time_last == 0) {
_error_mask |= ERROR_FLAG_NO_DATA;
ret = 0.0f;

} else if (timestamp - _time_last > _timeout_interval) {
/* timed out - that's it */
_error_mask |= ERROR_FLAG_TIMEOUT;
ret = 0.0f;

} else if (_value_equal_count > _value_equal_count_threshold) {
/* we got the exact same sensor value N times in a row */
_error_mask |= ERROR_FLAG_STALE_DATA;
ret = 0.0f;

} else if (_error_count > NORETURN_ERRCOUNT) {
/* check error count limit */
_error_mask |= ERROR_FLAG_HIGH_ERRCOUNT;
ret = 0.0f;

} else if (_error_density > ERROR_DENSITY_WINDOW) {
/* cap error density counter at window size */
_error_mask |= ERROR_FLAG_HIGH_ERRDENSITY;
_error_density = ERROR_DENSITY_WINDOW;

}

/* no critical errors */
if (ret > 0.0f) {
/* return local error density for last N measurements */
ret = 1.0f - (_error_density / ERROR_DENSITY_WINDOW);

if (ret > 0.0f) {
_error_mask = ERROR_FLAG_NO_ERROR;
}
}

return ret;
}
```

Para subsanar este error se establecerán dos procedimientos:

Elevar la velocidad de procesamiento: siguiendo las instrucciones recogidas en https://github.com/mirkix/ardupilotblue.

1. Actualización de software: sudo apt update && sudo apt upgrade -y
2. Instalar software: sudo apt install -y bb-cape-overlays cpufrequtils
3. Fijar frecuencia a 1GHz: sudo sed -i 's/GOVERNOR=.ºndemand/GOVERNOR="performance/g' /etc/init.d/cpufrequtils
4. Actualizar archivos: cd /opt/scripts && sudo git pull
5. Maximizar la partición de la tarjeta micro-SD:
   sudo /opt/scripts/tools/grow_partition.sh
6. Instalar RT Kernel 4.9: sudo /opt/scripts/tools/update_kernel.sh –ti-rt-channel –lts-4_9
7. Especificar el directorio binario usado en arranque:
   sudo sed -i 's/#dtb=/dtb=am335x-boneblue.dtb/g' /boot/uEnv.txt
8. Reiniciar el sistema: sudo reboot

Modificar el validador de datos: en algún caso y debido al carácter reactivo[6] del código, el intervalo por defecto presente no sera suficiente para evitar el error, a pesar de haber subido la frecuencia de trabajo del procesador. Por tanto, mediante las siguientes líneas de código se podrá aumentar este *threshold* o umbral evitando el error de TIMEOUT.

Archivo: voted_sensors_update.cpp

Ruta: Firmware/src/modules/sensors

```
private:
uint32_t _error_mask{ERROR_FLAG_NO_ERROR}; /**< sensor error state */

uint32_t _timeout_interval{30000};  /**< interval in which the datastream
    times out in us*/

uint64_t _time_last{0};       /**< last timestamp */
uint64_t _event_count{0};     /**< total data counter */
uint64_t _error_count{0};     /**< error count */

int _error_density{0};        /**< ratio between successful reads and
    errors */
```

[6]Se adapta a la carga de trabajo

```
int _priority{0};       /**< sensor nominal priority */

float _mean[dimensions] {};     /**< mean of value */
float _lp[dimensions] {};     /**< low pass value */
float _M2[dimensions] {};     /**< RMS component value */
float _rms[dimensions] {};    /**< root mean square error */
float _value[dimensions] {};    /**< last value */
float _vibe[dimensions] {};     /**< vibration level, in sensor unit */

unsigned _value_equal_count{0};    /**< equal values in a row */
unsigned _value_equal_count_threshold{VALUE_EQUAL_COUNT_DEFAULT}; /**<
    when to consider an equal count as a problem */

DataValidator *_sibling{nullptr};  /**< sibling in the group */

static const constexpr unsigned NORETURN_ERRCOUNT = 100000; /**< if the
    error count reaches this value, return sensor as invalid 10000*/
static const constexpr float ERROR_DENSITY_WINDOW = 100.0f;  /**< window
    in measurement counts for errors */
static const constexpr unsigned VALUE_EQUAL_COUNT_DEFAULT = 50000; /**<
    if the sensor value is the same (accumulated also between axes) this
    many times, flag it 100*/

/* we don't want this class to be copied */
DataValidator(const DataValidator &) = delete;
DataValidator operator=(const DataValidator &) = delete;
};
```

Como se ha expuesto en los resultados de la memoria principal existen incoherencias en el vector de datos obtenido generando un problema de STALE o ECNT para el caso del magnetómetro y el barómetro respectivamente.

Conclusiones

Se han repasado las principales modificaciones necesarias en el *PX4 flight stack* para ser capaz de operar sobre la BeagleBone Blue. La multitud de drivers disponibles, sus estándares y los archivos como I2CDevObj han permitido adaptar y mapear la SBC de forma adecuada, permitiendo así la ejecución del código. Aunque la visualización de los archivos resulta compleja debido a la gran extensión de los ficheros y la multitud de pequeños cambios realizados.

El desarrollo integro de drivers quedaba fuera del alcance de cualquier parte de este documento. Como se ha podido observar, la mayoría de modificaciones son localizadas y pequeñas, basándose en cambios en variables, objetos, protocolos de comunicación, registros o rutas de acceso entre componentes.