EE2028 Group 04 Assignment 2 Report

Group members:

Ong Wei Heng [A0235044N]

Rani Dilipkumar Shiva Shankar [A0235167A]

Lab Day: Tuesday 9:00am to 12:00pm

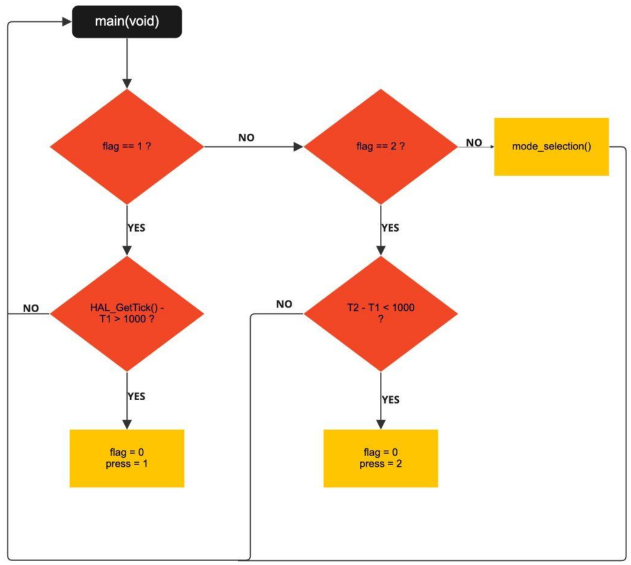
1. Introduction and objectives

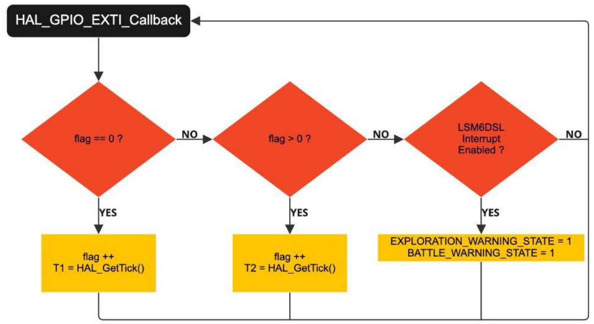
Pixie is a drone equipped with AI capabilities. It communicates back to a central base member called Cyrix. Pixie is able to transmit information such as temperature, humidity, pressure, acceleration, magnetic field, gyroscopic readings, warning messages as well as weapons information back to Cyrix. Our objective is to come up with a C program to collect the relevant data from the sensors on the STM32 board mounted on Pixie. While Pixie is doing its job, it reads, parses and displays these sensor readings with the appropriate magnitude and units via a dashboard using pygame framework. Additionally, an LED shows which mode/states Pixie is in. These modes/states include Exploration, Exploration Warning, Battle and Battle Warning. Sensor readings are collected through I2C protocol, LED, User Push Button with interrupt via GPIOs. To communicate back to Cyrix we have used UART protocol. We have also decided to add in a few external peripherals such as buzzer and an IR Sensor to make Pixie more realistic.

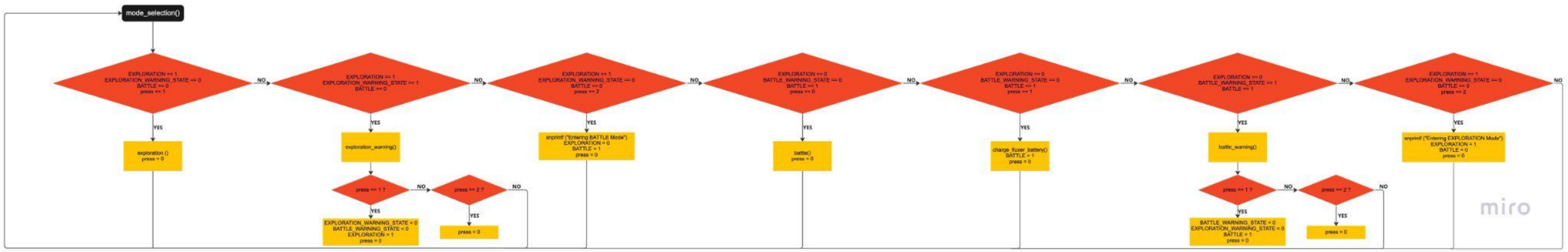
We have related the data produced by the sensor to our application purposes, from there we also set some real world thresholds for the sensors. When these threshold values are met they will trigger the board to a warning state, which will be explained later. Below is the table of how we intend to use our sensors.

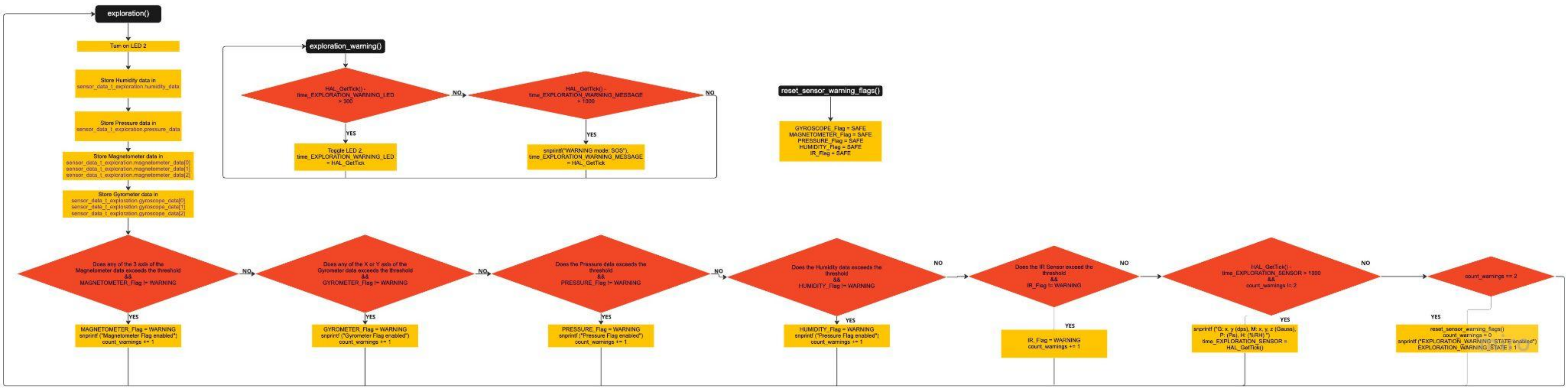
Temperature Sensor:	Measures the ambient temperature in (℃). Ensures that Pixie is operating at normal temperature.
Humidity Sensor:	Measures the ambient relative humidity in (%RH). Ensures that the environment is not too humid to short Pixie's circuit.
Pressure Sensor:	Measures ambient pressure in (Pa). Check if the atmospheric pressure is normal.
Accelerometer:	Mounted on top of Pixie. Measures all 3 axes in (m/s^2). Enabled free-fall interrupt when a free-fall is detected.
Gyrometer:	Measures all 3 axes in (dps) of changes. This allows Pixie's flight system to make adjustments to the power sent to each propeller.
Magnetometer:	Measures the 3 axis magnetic field strength in (Gauss). Allows Pixie to detect any strong magnetic field in the surroundings.

2. Flowcharts describing the system design and processes

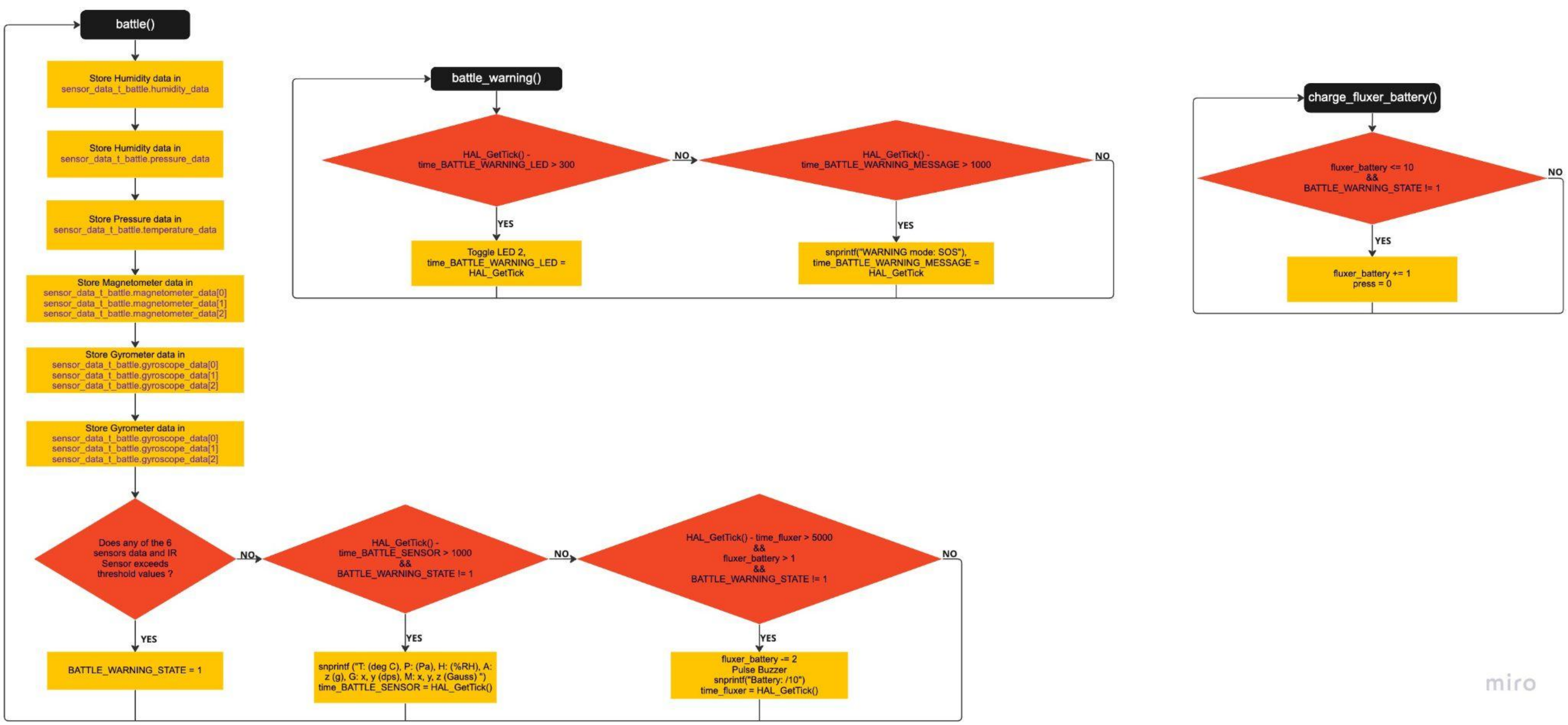








exploration()
Turn on LED 2
Store Humidity data in
sensor_data_t_exploration.humidity_data
Store Pressure data in
sensor_data_t_exploration.pressure_data
Store Magnetometer data in
sensor_data_t_exploration.magnetometer_data[0]
sensor_data_t_exploration.magnetometer_data[1]
sensor_data_t_exploration.magnetometer_data[2]
Store Gyrometer data in
sensor_data_t_exploration.gyroscope_data[0]
sensor_data_t_exploration.gyroscope_data[1]
sensor_data_t_exploration.gyroscope_data[2]
Does any of the 3 axis of the Magnetometer data exceeds the threshold && MAGNETOMETER_Flag != WARNING
YES
MAGNETOMETER_Flag = WARNING
snprintf ("Magnetometer Flag enabled")
count_warnings += 1
NO
Does any of the X or Y axis of the Gyrometer data exceeds the threshold && GYROMETER_Flag != WARNING
YES
GYROMETER_Flag = WARNING
snprintf ("Gyrometer Flag enabled")
count_warnings += 1
NO
Does the Pressure data exceeds the threshold && PRESSURE_Flag != WARNING
YES
PRESSURE_Flag = WARNING
snprintf ("Pressure Flag enabled")
count_warnings += 1
NO
Does the Humidity data exceeds the threshold && HUMIDITY_Flag != WARNING
YES
HUMIDITY_Flag = WARNING
snprintf ("Pressure Flag enabled")
count_warnings += 1
NO
Does the IR Sensor exceed the threshold && IR_Flag != WARNING
YES
IR_Flag = WARNING
count_warnings += 1
NO
HAL_GetTick() - time_EXPLORATION_SENSOR > 1000 && count_warnings != 2
YES
snprintf ("G: x, y (dps), M: x, y, z (Gauss), P: (Pa), H: (%RH) ")
time_EXPLORATION_SENSOR = HAL_GetTick()
NO
count_warnings == 2
YES
reset_sensor_warning_flags()
count_warnings = 0
snprintf ("EXPLORATION_WARNING_STATE enabled")
EXPLORATION_WARNING_STATE = 1
exploration_warning()
HAL_GetTick() - time_EXPLORATION_WARNING_LED > 300
YES
Toggle LED 2, time_EXPLORATION_WARNING_LED = HAL_GetTick
NO
HAL_GetTick() - time_EXPLORATION_WARNING_MESSAGE > 1000
YES
snprintf("WARNING mode: SOS"), time_EXPLORATION_WARNING_MESSAGE = HAL_GetTick
NO
reset_sensor_warning_flags()
GYROSCOPE_Flag = SAFE
MAGNETOMETER_Flag = SAFE
PRESSURE_Flag = SAFE
HUMIDITY_Flag = SAFE
IR_Flag = SAFE



battle()
Store Humidity data in
sensor_data_t_battle.humidity_data
Store Humidity data in
sensor_data_t_battle.pressure_data
Store Pressure data in
sensor_data_t_battle.temperature_data
Store Magnetometer data in
sensor_data_t_battle.magnetometer_data[0]
sensor_data_t_battle.magnetometer_data[1]
sensor_data_t_battle.magnetometer_data[2]
Store Gyrometer data in
sensor_data_t_battle.gyroscope_data[0]
sensor_data_t_battle.gyroscope_data[1]
sensor_data_t_battle.gyroscope_data[2]
Store Gyrometer data in
sensor_data_t_battle.gyroscope_data[0]
sensor_data_t_battle.gyroscope_data[1]
sensor_data_t_battle.gyroscope_data[2]
Does any of the 6 sensors data and IR Sensor exceeds threshold values ?
YES
BATTLE_WARNING_STATE = 1
NO
HAL_GetTick() - time_BATTLE_SENSOR > 1000 && BATTLE_WARNING_STATE != 1
YES
snprintf ("T: (deg C), P: (Pa), H: (%RH), A: z (g), G: x, y (dps), M: x, y, z (Gauss) ")
time_BATTLE_SENSOR = HAL_GetTick()
NO
HAL_GetTick() - time_fluxer > 5000 && fluxer_battery > 1 && BATTLE_WARNING_STATE != 1
YES
fluxer_battery -= 2
Pulse Buzzer
snprintf("Battery: /10")
time_fluxer = HAL_GetTick()
NO
battle_warning()
HAL_GetTick() - time_BATTLE_WARNING_LED > 300
YES
Toggle LED 2, time_BATTLE_WARNING_LED = HAL_GetTick
NO
HAL_GetTick() - time_BATTLE_WARNING_MESSAGE > 1000
YES
snprintf("WARNING mode: SOS"), time_BATTLE_WARNING_MESSAGE = HAL_GetTick
NO
charge_fluxer_battery()
fluxer_battery <= 10 && BATTLE_WARNING_STATE != 1
YES
fluxer_battery += 1
press = 0
NO
miro

3. Detailed implementation

```c
1  /* Includes ---------------------------------------------------------------
2  #include "main.h"
3  #include "../../Drivers/BSP/B-L475E-IOT01/stm32l475e_iot01_accelero.h"
4  #include "../../Drivers/BSP/B-L475E-IOT01/stm32l475e_iot01_tsensor.h"
5  #include "../../Drivers/BSP/B-L475E-IOT01/stm32l475e_iot01_gyro.h"
6  #include "../../Drivers/BSP/B-L475E-IOT01/stm32l475e_iot01_magneto.h"
7  #include "../../Drivers/BSP/B-L475E-IOT01/stm32l475e_iot01_psensor.h"
8  #include "../../Drivers/BSP/B-L475E-IOT01/stm32l475e_iot01_hsensor.h"
9  #include "stdio.h"
10 #include "stdlib.h"  // use stdlib.h header file to use abs() function.
11 #include "stdbool.h"
12 #include "string.h"
13 #include "math.h"
14 #include "ctype.h"
15
16 /* Declare Constants -------------------------------------------------------
17 #define GYRO_THRESHOLD 100.0 // based on testing
18 #define ACCEL_THRESHOLD -9.0 // based on testing
19 #define TEMP_THRESHOLD_MIN -20.0 // -20 degrees Celcius
20 #define TEMP_THRESHOLD_MAX 70.0 // 70 degrees Celcius
21
22 #define MAG_THRESHOLD 3.0 // Max is 4
23
24 #define HUM_THRESHOLD 30.0
25 #define PRES_THRESHOLD_MIN 90000.0
26 #define PRES_THRESHOLD_MAX 105000.0
27
28 #define WARNING 1
29 #define SAFE 0
30
31 #define MESSAGE_SIZE 300
32
33 /* Function Declarations ---------------------------------------------------
34 void SystemClock_Config(void);
35 static void UART1_Init(void);
36 static void MX_GPIO_Init(void);
37
38 static void mode_selection(void);
39 static void exploration(void);
40 static void exploration_warning(void);
41 static void battle(void);
42 static void battle_warning(void);
43 static void charge_fluxer_battery(void);
44 void reset_sensor_warning_flags(void);
45 void acc_interrupt_config(void);
```

```c
47 /* Global Variables --------------------------------------------------------
48 uint32_t T1, T2; // counting single and double press
49 UART_HandleTypeDef huart1; // huart1 variable of type UART_HANDLER
50 uint8_t flag = 0, press = 0, EXPLORATION = 1, EXPLORATION_WARNING_STATE = 0,
51         BATTLE = 0, BATTLE_WARNING_STATE = 0, count_warnings = 0,
52         fluxer_battery = 10;
53
54 char message_print[MESSAGE_SIZE]; // array buffer used for UART1 transmission
55
56 uint32_t time_EXPLORATION_SENSOR = 0;
57 uint32_t time_EXPLORATION_WARNING_LED = 0;
58 uint32_t time_EXPLORATION_WARNING_MESSAGE = 0;
59
60 uint32_t time_BATTLE_SENSOR = 0;
61 uint32_t time_BATTLE_WARNING_LED = 0;
62 uint32_t time_BATTLE_WARNING_MESSAGE = 0;
63 uint32_t time_BATTLE_LED = 0;
64 uint32_t time_fluxer = 0;
65
66 volatile uint8_t GYROSCOPE_Flag = SAFE, MAGNETOMETER_Flag = SAFE,
67         PRESSURE_Flag = SAFE, HUMIDITY_Flag = SAFE, IR_Flag = SAFE;
68
69 typedef struct sensor_data_t {
70     float temperature_data;
71     float humidity_data;
72     float pressure_data;
73     float altitude;
74
75     int16_t magnetometer_raw_data[3];
76     float magnetometer_data[3];
77
78     int16_t accelerometer_raw_data[3];
79     float accelerometer_data[3];
80
81     float gyroscope_raw_data[3];
82     float gyroscope_data[3];
83
84 } sensor_data_t;
85 // Initialize 2 structs for Exploration and Battle modes
86 sensor_data_t sensor_data_t_exploration;
87 sensor_data_t sensor_data_t_battle;
```

Line 1 - 14.

- Include all the library files for each sensor.

Line 17 - 31.

- Define sensor threshold values.

Line 34 - 45.

- Function prototypes are placed here to be used later.

Line 47 - 67.

- Declaration of global variables, for different modes, different states and for timers.

Line 69 - 87.

- Declaring a structure to place values of different data types.

```c
94  HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
95      if ((GPIO_Pin == BUTTON_EXTI13_Pin) && (flag == 0)) {
96          flag++;
97          T1 = uwTick;
98      } else if ((GPIO_Pin == BUTTON_EXTI13_Pin) && (flag > 0)) {
99          flag++;
100         T2 = uwTick;
101     }
102     if (GPIO_Pin == LSM6DSL_INT1_EXTI11_Pin)
103     {
104         EXPLORATION_WARNING_STATE = 1;
105         BATTLE_WARNING_STATE = 1;
106     }
107 }
```

```c
109 int main(void) {
110
111     /* Reset of all peripherals */
112     HAL_Init();
113     MX_GPIO_Init(); // initialize PB14, pin connected to LED2
114     UART1_Init(); // initialize UART1 for UART communication
115
116     /* Peripheral initializations using BSP functions for all sensors*/
117     BSP_ACCELERO_Init(); // initialize accelerometer
118     BSP_TSENSOR_Init();  // initialize temperature sensor
119     BSP_HSENSOR_Init();  // initialize humidity sensor
120     BSP_GYRO_Init();     // initialize gyroscope
121     BSP_PSENSOR_Init();  // initialize pressure sensor
122     BSP_MAGNETO_Init();  // initialize magnetometer
123     acc_interrupt_config(); // initialize Accelerometer interrupt
124
125     /* Exploration Mode: Print only once*/
126     //memset(message_print, 0, strlen(message_print));
127     snprintf(message_print, MESSAGE_SIZE, "Entering EXPLORATION Mode \r\n");
128     HAL_UART_Transmit(&huart1, (uint8_t*) message_print, strlen(message_print),
129             0xFFFF);
130
131     while (1) {
132         if ((flag == 1) && (uwTick - T1 > 1000)) {
133             // Detect a single press
134             flag = 0;
135             press = 1;
136         }
137
138         if ((flag == 2) && (T2 - T1 < 1000)) {
139             // Detect double press
140             press = 2;
141             flag = 0;
142         }
143
144         mode_selection();
145     }
146 }
```

main()

Line 94 - 107.

- External interrupt handler to handle interrupts from user push button via EXTI13 and LSM6DSL interrupt via EXTI11.

Line 109 - 128.

- In the main() function, HAL_Init() and UART1_Init are being initialized.
- All the ports and pins used are also being initialized under MX_GPIO_Init().
- Sensor initializations are done using the respective BSP functions.
- LSM6DSL interrupt registers are also configured.
- A message is being sent when entering into Exploration Mode.

Line 131 - 144.

- Verify that the user push button is being used and note how many times it has been pressed.
- Enter mode_selection() function.

```c
154 static void mode_selection() {
155     /* Exploration Mode */
156     if (EXPLORATION == 1 && EXPLORATION_WARNING_STATE == 0 && BATTLE == 0
157             && press <= 1) {
158         // Normal state
159         exploration();
160         press = 0;
161     } else if (EXPLORATION == 1 && EXPLORATION_WARNING_STATE == 1) {
162         // Come to the Warning State through interrupts or polling
163         exploration_warning();
164 
165         if (press == 1) {
166             // Clear the warning and go back to Exploration mode
167             BATTLE_WARNING_STATE = 0;
168             EXPLORATION_WARNING_STATE = 0;
169             EXPLORATION = 1;
170             press = 0;
171         } else if (press == 2) {
172             // Ignore it
173             press = 0;
174         }
175     } else if (EXPLORATION == 1 && EXPLORATION_WARNING_STATE == 0
176             && press == 2) {
177         // Change to Battle Mode
178         /* A message "Entering BATTLE mode" is sent once to Cyrix's Lab once
179          * immediately upon entering the BATTLE mode.
180          * The press flag is cleared later in mode_selection()
181          */
182         //memset(message_print, 0, strlen(message_print));
183         snprintf(message_print, MESSAGE_SIZE, "Entering BATTLE Mode \r\n");
184         HAL_UART_Transmit(&huart1, (uint8_t*) message_print,
185                 strlen(message_print), 0xFFFF);
186 
187         EXPLORATION = 0;
188         BATTLE = 1;
189         press = 0;
190     }
```

```c
192     /* Battle Mode */
193     if (EXPLORATION == 0 && BATTLE_WARNING_STATE == 0 && BATTLE == 1
194             && press == 0) {
195         // Battle state
196         battle();
197         press = 0;
198     } else if (EXPLORATION == 0 && BATTLE_WARNING_STATE == 0 && BATTLE == 1
199             && press == 1) {
200         /*  In BATTLE_MODE, without WARNING:
201          *  i.e., when Pixie is not sending 'SOS' message to Cyrix,
202          *  single press triggers BATTERY_CHARGING,
203          *  i.e., after single press, Fluxer is charged with 1/10 energy
204          *  of its capacity.*/
205         charge_fluxer_battery();
206         BATTLE = 1;
207         press = 0; // reset the press flag
208     } else if (BATTLE == 1 && BATTLE_WARNING_STATE == 1) {
209         battle_warning();
210 
211         if (press == 1) {
212             // Clear the warning and go back to Battle mode
213             BATTLE_WARNING_STATE = 0;
214             EXPLORATION_WARNING_STATE = 0;
215             BATTLE = 1;
216             press = 0;
217         } else if (press == 2) {
218             // Ignore it
219             press = 0;
220         }
221     } else if (BATTLE == 1 && BATTLE_WARNING_STATE == 0 && press == 2) {
222         // Change to EXPLORATION Mode
223         BATTLE = 0;
224         EXPLORATION = 1;
225         press = 0;
226     }
227 }
```

<u>mode_selection()</u>

Line 154-159.

- Check if exploration mode is active without any warning. If yes, executes the exploration function.

Line 161 - 174.

- Check if exploration mode is active with a warning state. If yes executes exploration warning function.
- Wait for push button to be pressed to clear any warning state

Line 175 - 189.

- Check if exploration mode is active without any warning and whether double press is active. If yes, execute mode change to BATTLE Mode.

Line 192 - 197.

- Check if battle mode is active without any warning. If yes, execute the battle function.

Line 198 - 207.

- Check if battle mode is active and whether the user button is being pressed once. If yes, charge the fluxer battery.

Line 208 - 220.

- Check if battle mode is active with a warning state. If yes, execute the exploration warning function.
- Wait for push button to be pressed to clear any warning state

Line 221 - 227.

- Check if battle mode is active without any warning and whether double press is active. If yes, execute mode change to EXPLORATION Mode.

```c
237 static void exploration(void) {
238
239     // Reset variables
240     sensor_data_t_exploration.humidity_data = 0;
241     sensor_data_t_exploration.pressure_data = 0;
242     sensor_data_t_exploration.magnetometer_raw_data[3] = 0;
243     sensor_data_t_exploration.magnetometer_data[3] = 0;
244     sensor_data_t_exploration.gyroscope_raw_data[3] = 0;
245     sensor_data_t_exploration.gyroscope_data[3] = 0;
246
247     // Read Humidity readings
248     sensor_data_t_exploration.humidity_data = BSP_HSENSOR_ReadHumidity();
249     // Read the pressure in units (Pascal)
250     // One hectopascal(hPa) is equal to exactly 100 Pascals.
251     sensor_data_t_exploration.pressure_data = BSP_PSENSOR_ReadPressure()
252             * 100.0f;
253
254     // Pass in the memory address to pDataXYZ Pointer to get XYZ magnetometer values.
255     BSP_MAGNETO_GetXYZ(sensor_data_t_exploration.magnetometer_raw_data);
256
257     sensor_data_t_exploration.magnetometer_data[0] =
258             (float) sensor_data_t_exploration.magnetometer_raw_data[0]
259                     / 1000.0f;
260     sensor_data_t_exploration.magnetometer_data[1] =
261             (float) sensor_data_t_exploration.magnetometer_raw_data[1]
262                     / 1000.0f;
263     sensor_data_t_exploration.magnetometer_data[2] =
264             (float) sensor_data_t_exploration.magnetometer_raw_data[2]
265                     / 1000.0f;
266
267     // Pass in the memory address to pDataXYZ Pointer to get XYZ gyroscope values.
268     BSP_GYRO_GetXYZ(sensor_data_t_exploration.gyroscope_raw_data);
269     sensor_data_t_exploration.gyroscope_data[0] =
270             sensor_data_t_exploration.gyroscope_raw_data[0] / 1000.0f;
271     sensor_data_t_exploration.gyroscope_data[1] =
272             sensor_data_t_exploration.gyroscope_raw_data[1] / 1000.0f;
273     sensor_data_t_exploration.gyroscope_data[2] =
274             sensor_data_t_exploration.gyroscope_raw_data[2] / 1000.0f;
```

```c
287     if ((abs((int) sensor_data_t_exploration.magnetometer_data[0])
288             >= MAG_THRESHOLD
289             || abs((int) sensor_data_t_exploration.magnetometer_data[1])
290                     >= MAG_THRESHOLD
291             || abs((int) sensor_data_t_exploration.magnetometer_data[2])
292                     >= MAG_THRESHOLD) && MAGNETOMETER_Flag != WARNING) {
293
294         // Set MAGNETOMETER_Flag to WARNING
295         MAGNETOMETER_Flag = WARNING;
296
297         snprintf(message_print, MESSAGE_SIZE, "Magnetometer Flag enabled \r\n");
298         HAL_UART_Transmit(&huart1, (uint8_t*) message_print, strlen(message_print), 0xFFFF);
299
300         count_warnings += 1;
301     }
302
303     if ((abs((int) sensor_data_t_exploration.gyroscope_data[0])
304             >= GYRO_THRESHOLD
305             || abs((int) sensor_data_t_exploration.gyroscope_data[1])
306                     >= GYRO_THRESHOLD) && GYROSCOPE_Flag != WARNING) {
307
308         // Set GYROSCOPE_Flag to WARNING
309         GYROSCOPE_Flag = WARNING;
310
311         snprintf(message_print, MESSAGE_SIZE, "Gyroscope Flag enabled \r\n");
312         HAL_UART_Transmit(&huart1, (uint8_t*) message_print, strlen(message_print), 0xFFFF);
313
314         count_warnings += 1;
315     }
316
317     if (((sensor_data_t_exploration.pressure_data <= PRES_THRESHOLD_MIN)
318             || (sensor_data_t_exploration.pressure_data >= PRES_THRESHOLD_MAX))
319             && PRESSURE_Flag != WARNING) {
320
321         // Set PRESSURE_Flag to WARNING
322         PRESSURE_Flag = WARNING;
323
324         snprintf(message_print, MESSAGE_SIZE,
325                 "Pressure Flag enabled P:%0.2f (Pa) \r\n",
326                 sensor_data_t_exploration.pressure_data);
327         HAL_UART_Transmit(&huart1, (uint8_t*) message_print, strlen(message_print), 0xFFFF);
328
329         count_warnings += 1;
330     }
```

<u>exploration()</u>

Line 237 - 245.

- Initialize members of sensor_data_t_exploration to 0 and set arrays for magnetometer and gyrometer readings to 0.

Line 248 - 274.

- Using sensor BSP libraries to get the measured values and storing them in the structure with the defined variables.

Line 287 - 359.

- Check if any of the sensor values exceed the defined threshold values.
- If threshold values are exceeded, the number of warnings is incremented and the sensor flag will be raised.
- Sensor flags stop any sensors that have already been captured from being recorded again.

```c
332     if ((sensor_data_t_exploration.humidity_data <= HUM_THRESHOLD)
333             && (HUMIDITY_Flag != WARNING)) {
334
335         // Set HUMIDITY_Flag to WARNING
336         HUMIDITY_Flag = WARNING;
337
338         //memset(message_print, 0, strlen(message_print));
339         snprintf(message_print, MESSAGE_SIZE,
340                 "Humidity Flag enabled , H:%0.2f (%%RH) \r\n",
341                 sensor_data_t_exploration.humidity_data);
342         HAL_UART_Transmit(&huart1, (uint8_t*) message_print, strlen(message_print), 0xFFFF);
343
344         count_warnings += 1;
345     }
346
347     // Code for IR sensor; CCK to reduce the distance, ACCK to increase the distance
348     uint8_t IR_sensor = HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_14);
349     if ((IR_sensor == 0) && (IR_Flag != WARNING)) {
350         IR_Flag = WARNING;
351
352         snprintf(message_print, MESSAGE_SIZE, "INFRARED SENSOR enabled \r\n");
353         HAL_UART_Transmit(&huart1, (uint8_t*) message_print, strlen(message_print), 0xFFFF);
354
355         count_warnings += 1;
356
357         IR_sensor = 1; // reset the state of the push button
358     }
359
360     // In EXPLORATION MODE, only those sensors mounted on Pixie are read periodically every ONE second
361     if ((HAL_GetTick() - time_EXPLORATION_SENSOR > 1000)
362             && (count_warnings != 2)) {
363         //memset(message_print, 0, strlen(message_print));
364         snprintf(message_print, MESSAGE_SIZE,
365                 "1, G:%0.2f:%0.2f (dps), M:%0.3f:%0.3f:%0.3f (Gauss), P:%0.2f (Pa), H:%0.2f (%%RH) \r\n",
366                 sensor_data_t_exploration.gyroscope_data[0],
367                 sensor_data_t_exploration.gyroscope_data[1],
368                 // sensor_data_t_exploration.gyroscope_data[2],
369                 sensor_data_t_exploration.magnetometer_data[0],
370                 sensor_data_t_exploration.magnetometer_data[1],
371                 sensor_data_t_exploration.magnetometer_data[2],
372                 sensor_data_t_exploration.pressure_data,
373                 sensor_data_t_exploration.humidity_data);
374         HAL_UART_Transmit(&huart1, (uint8_t*) message_print,strlen(message_print), 0xFFFF);
375
376         time_EXPLORATION_SENSOR = HAL_GetTick(); // reset the variable
377     }
```

```c
379     // EXPLORATION LED will always be ON
380     HAL_GPIO_WritePin(GPIOB, LED2_Pin, GPIO_PIN_SET);
381
382     if (count_warnings == 2) {
383
384         reset_sensor_warning_flags();
385         count_warnings = 0;
386
387         snprintf(message_print, MESSAGE_SIZE,"EXPLORATION_WARNING_STATE enabled \r\n");
388         HAL_UART_Transmit(&huart1, (uint8_t*) message_print,strlen(message_print), 0xFFFF);
389
390         // Set the EXPLORATION_WARNING_STATE flag to 1
391         EXPLORATION_WARNING_STATE = 1;
392     }
393 }
```

```c
395 static void exploration_warning(void) {
396     // Toggle WARNING LED every 3 seconds.
397     if ((HAL_GetTick() - time_EXPLORATION_WARNING_LED) > 3000) {
398         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
399         time_EXPLORATION_WARNING_LED = HAL_GetTick(); // reset time_EXPLORATION_WARNING_LED
400     }
401
402     // Send WARNING mode: SOS once every 1 second.
403     if ((HAL_GetTick() - time_EXPLORATION_WARNING_MESSAGE) > 1000) {
404         //memset(message_print, 0, strlen(message_print));
405         snprintf(message_print, MESSAGE_SIZE, "2, WARNING mode: SOS \r\n");
406         HAL_UART_Transmit(&huart1, (uint8_t*) message_print,
407                 strlen(message_print), 0xFFFF);
408         time_EXPLORATION_WARNING_MESSAGE = HAL_GetTick(); // reset time_EXPLORATION_WARNING_LED
409     }
410 }
```

Line 361 - 377.

- If the warning count is not equal to 2 and 1 second has elapsed, display sensor values to Cyrix via UART.

Line 380.

- Turn on LED 2.

Line 382 - 393.

- If the warning count is equal to 2, resets all sensor flags, sends a message to Cyrix and enables Exploration Warning State.

Line 395 - 410.

- If the Exploration Warning State is enabled, the exploration warning function is executed.
- Warning LED flashes 3 times every 1 second.
- Warning message is sent to Cyrix every 1 second.

```c
412 static void battle(void) {
413
414     // Reset variables
415     sensor_data_t_battle.temperature_data = 0;
416     sensor_data_t_battle.humidity_data = 0;
417     sensor_data_t_battle.pressure_data = 0;
418     sensor_data_t_battle.magnetometer_raw_data[3] = 0;
419     sensor_data_t_battle.magnetometer_data[3] = 0;
420     sensor_data_t_battle.gyroscope_raw_data[3] = 0;
421     sensor_data_t_battle.gyroscope_data[3] = 0;
422     sensor_data_t_battle.accelerometer_raw_data[3] = 0;
423     sensor_data_t_battle.accelerometer_data[3] = 0;
424
425     // Read Humidity readings
426     sensor_data_t_battle.humidity_data = BSP_HSENSOR_ReadHumidity();
427
428     /*  Read the pressure in units (Pascal)
429      *  One hectopascal(hPa) is equal to exactly 100 Pascals. */
430     sensor_data_t_battle.pressure_data = BSP_PSENSOR_ReadPressure() * 100.0f;
431
432     // Read Temperature Readings
433     sensor_data_t_battle.temperature_data = BSP_TSENSOR_ReadTemp();
434
435     // Pass in the memory address to pDataXYZ Pointer to get XYZ magnetometer values.
436     BSP_MAGNETO_GetXYZ(sensor_data_t_battle.magnetometer_raw_data);
437     sensor_data_t_battle.magnetometer_data[0] =
438             (float) sensor_data_t_battle.magnetometer_raw_data[0] / 1000.0f;
439     sensor_data_t_battle.magnetometer_data[1] =
440             (float) sensor_data_t_battle.magnetometer_raw_data[1] / 1000.0f;
441     sensor_data_t_battle.magnetometer_data[2] =
442             (float) sensor_data_t_battle.magnetometer_raw_data[2] / 1000.0f;
443
444     // Pass in the memory address to pDataXYZ Pointer to get XYZ gyroscope values.
445     BSP_GYRO_GetXYZ(sensor_data_t_battle.gyroscope_raw_data);
446     sensor_data_t_battle.gyroscope_data[0] =
447             sensor_data_t_battle.gyroscope_raw_data[0] / 1000.0f;
448     sensor_data_t_battle.gyroscope_data[1] =
449             sensor_data_t_battle.gyroscope_raw_data[1] / 1000.0f;
450     sensor_data_t_battle.gyroscope_data[2] =
451             sensor_data_t_battle.gyroscope_raw_data[2] / 1000.0f;
452
453     /* Pass in the memory address to pDataXYZ Pointer to get XYZ accelerometer values.
454      * The function below returns 16 bit integers which are 100 * acceleration(m/s^2).
455      * Convert to float to print the actual acceleration*/
456     BSP_ACCELERO_AccGetXYZ(sensor_data_t_battle.accelerometer_raw_data);
457     sensor_data_t_battle.accelerometer_data[0] =
458             sensor_data_t_battle.accelerometer_raw_data[0] / 100.0f;
459     sensor_data_t_battle.accelerometer_data[1] =
460             sensor_data_t_battle.accelerometer_raw_data[1] / 100.0f;
461     sensor_data_t_battle.accelerometer_data[2] =
462             sensor_data_t_battle.accelerometer_raw_data[2] / 100.0f;
```

```c
464     // read IR output here
465     uint8_t IR_sensor = HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_14);
466
467     /**
468      * @brief   Check if the sensors have reached their threshold, then if ANY
469      *          of the sensors have exceeded their maximum/minimum threshold,
470      *          go to the WARNING state.
471      * @steps   1. Raise flags if threshold is reached.
472      *          2. Type-cast variables explicitly to (int) to use abs()
473      *          3. Set the EXPLORATION_WARNING_STATE flag to 1.
474      */
475     if ((sensor_data_t_battle.temperature_data >= TEMP_THRESHOLD_MAX
476             || sensor_data_t_battle.temperature_data < TEMP_THRESHOLD_MIN)
477             || sensor_data_t_battle.humidity_data <= HUM_THRESHOLD
478             || (sensor_data_t_battle.pressure_data >= PRES_THRESHOLD_MAX
479                 || sensor_data_t_battle.pressure_data <= PRES_THRESHOLD_MIN)
480             || (abs(sensor_data_t_battle.magnetometer_data[0]) >= MAG_THRESHOLD)
481             || (abs(sensor_data_t_battle.magnetometer_data[1]) >= MAG_THRESHOLD)
482             || (abs(sensor_data_t_battle.magnetometer_data[2]) >= MAG_THRESHOLD)
483             || (IR_sensor == 0)) {
484         BATTLE_WARNING_STATE = 1;
485         IR_sensor = 1;
486     }
487
488     // In BATTLE MODE, only those sensors mounted on Pixie are read periodically every ONE second.
489     if (HAL_GetTick() - time_BATTLE_SENSOR > 1000
490             && BATTLE_WARNING_STATE != 1) {
491         //memset(message_print, 0, strlen(message_print));
492         snprintf(message_print, MESSAGE_SIZE,
493                 "3, T:%0.2f (deg C), P:%0.2f (Pa), H:%0.2f (%%RH), A:%0.2f (g), G:%0.2f:%0.2f (dps),
494                 sensor_data_t_battle.temperature_data,
495                 sensor_data_t_battle.pressure_data,
496                 sensor_data_t_battle.humidity_data,
497                 sensor_data_t_battle.accelerometer_data[2],
498                 sensor_data_t_battle.gyroscope_data[0],
499                 sensor_data_t_battle.gyroscope_data[1],
500                 // sensor_data_t_battle.gyroscope_data[2],
501                 sensor_data_t_battle.magnetometer_data[0],
502                 sensor_data_t_battle.magnetometer_data[1],
503                 sensor_data_t_battle.magnetometer_data[2]);
504         HAL_UART_Transmit(&huart1, (uint8_t*) message_print,
505                 strlen(message_print), 0xFFFF);
506
507         time_BATTLE_SENSOR = HAL_GetTick();
508     }
509
510     // Toggle WARNING LED every 1 second.
511     if ((HAL_GetTick() - time_BATTLE_LED) > 1000) {
512         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
513         time_BATTLE_LED = HAL_GetTick(); // reset time_BATTLE_LED
514     }
```

<u>battle()</u>

Line 412 - 423.

- Initializes members of sensor_data_t_exploration and declares arrays for magnetometer, accelerometer and gyrometer readings.

Line 426 - 462.

- Using sensor BSP libraries to get the measured values and store them in the structure with the defined variables.

Line 475 - 486.

- Check if temperature, humidity, pressure sensor and magnetometer values exceed the defined threshold values.
- If threshold values are exceeded, enable Battle Warning State.

Line 489 - 508.

- If Battle Warning State is not enabled and 1 second has elapsed, display sensor values to Cyrix via UART.

Line 511 - 514.

- Toggles LED 2 in battle mode.

```c
516     // Self firing Fluxer every 5s.
517     if (HAL_GetTick() - time_fluxer > 5000 && fluxer_battery > 1
518             && BATTLE_WARNING_STATE != 1) {
519
520         HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET); //On Buzzer
521
522         fluxer_battery -= 2;
523
524         time_fluxer = HAL_GetTick(); // reset time_fluxer
525
526         //memset(message_print, 0, strlen(message_print));
527         snprintf(message_print, MESSAGE_SIZE, "5, Battery: %d/10 \r\n",
528                 fluxer_battery);
529         HAL_UART_Transmit(&huart1, (uint8_t*) message_print,
530                 strlen(message_print), 0xFFFF);
531     }
532     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET); //Off Buzzer
533 }
```

```c
535 static void charge_fluxer_battery(void) {
536     // Fluxer recharging using PB.
537     if (fluxer_battery <= 10 && BATTLE_WARNING_STATE != 1) {
538         fluxer_battery += 1;
539         press = 0;
540     } else {
541         press = 0;
542     }
543 }
544
545 static void battle_warning(void) {
546     // Toggle WARNING LED every 3 seconds.
547     if ((HAL_GetTick() - time_BATTLE_WARNING_LED) > 3000) {
548         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
549         time_BATTLE_WARNING_LED = HAL_GetTick(); // reset time_EXPLORATION_WARNING_LED
550     }
551
552     // Warning Message sent every once a second
553     if ((HAL_GetTick() - time_BATTLE_WARNING_MESSAGE) > 1000) {
554         // send BATTLE mode: SOS
555         //memset(message_print, 0, strlen(message_print));
556         snprintf(message_print, MESSAGE_SIZE, "4, BATTLE mode: SOS \r\n");
557         HAL_UART_Transmit(&huart1, (uint8_t*) message_print,
558                 strlen(message_print), 0xFFFF);
559         time_BATTLE_WARNING_MESSAGE = HAL_GetTick(); // reset time_EXPLORATION_WARNING_LED
560     }
561 }
```

Line 516 - 533.

- Fluxer is being fired, and the remaining battery value is sent to Cyrix via UART every 5 seconds.
- Piezo Buzzer to indicate firing sound.

Line 535 - 543.

- The charge_fluxer_battery() checks if the battery level is below 10 and if the program is not in exploration mode, the fluxer battery increases by 1 unit.

Line 545 - 561.

- If the Battle Warning State is enabled, the battle warning function is executed.
- Warning LED flashes 3 times every 1 second.
- Warning message is sent to Cyrix every 1 second.

```c
568 void reset_sensor_warning_flags(void) {
569     GYROSCOPE_Flag = SAFE;
570     MAGNETOMETER_Flag = SAFE;
571     PRESSURE_Flag = SAFE;
572     HUMIDITY_Flag = SAFE;
573     IR_Flag = SAFE;
574 }
575
576 /**
577  * @brief   Set LSM6DSL to detect freefall and eanble INT1
578  * @note
579  * @retval  None
580  */
581 void acc_interrupt_config(void)
582 {
583     //1000 0000 set bit[7] to enable interrupts
584     SENSOR_IO_Write(LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW, LSM6DSL_ACC_GYRO_TAP_CFG1, 0x80);
585     //0000 1000 FF_Dur [4:0] = 00001 & FF_Ths [2:0] = 000
586     SENSOR_IO_Write(LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW, LSM6DSL_ACC_GYRO_FREE_FALL, 0x08);
587     //0001 0000 Enables bit 4 of MD1_CFG Register
588     SENSOR_IO_Write(LSM6DSL_ACC_GYRO_I2C_ADDRESS_LOW, LSM6DSL_ACC_GYRO_MD1_CFG, 0x10);
589 }
```

<u>reset_sensor_warning_flags()</u>
Line 568 - 574.
- Resets sensor flags during exploration mode.

<u>acc_interrupt_config()</u>
Line 584.
- Writes to LSM6DSL TAP_CFG1 register to enable interrupts.

Line 586.
- Writes to LSM6DSL Free-fall register to set up Free-fall threshold.

Line 588.
- Writes to LSM6DSL MD1_CFG1 register to enable interrupts on INT1.

```c
591 static void MX_GPIO_Init(void) //For LED and PB
592 {
593     GPIO_InitTypeDef GPIO_InitStruct = { 0 };
594
595     //GPIO Ports Clock Enable
596     __HAL_RCC_GPIOA_CLK_ENABLE();// Buzzer output
597     __HAL_RCC_GPIOB_CLK_ENABLE();// For LED
598     __HAL_RCC_GPIOC_CLK_ENABLE();// For Push Button
599     __HAL_RCC_GPIOD_CLK_ENABLE();// IR Sensor and LSM6D
600
601     //Configure GPIO pin Output Level // Pin Initializa
602     HAL_GPIO_WritePin(GPIOB, LED2_Pin, GPIO_PIN_RESET);
603
604     //Configure GPIO pin LED2_Pin  // Pin Configuration
605     GPIO_InitStruct.Pin = LED2_Pin;
606     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
607     GPIO_InitStruct.Pull = GPIO_NOPULL;
608     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
609     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
610
611     //Configuration of D8 as output for Buzzer
612     GPIO_InitStruct.Pin = ARD_D8_Pin;
613     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
614     GPIO_InitStruct.Pull = GPIO_NOPULL;
615     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct); // PB8
616
617     //Configuration of BUTTON_EXTI13_Pin (GPIO-C Pin-13
618     GPIO_InitStruct.Pin = BUTTON_EXTI13_Pin;
619     GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
620     GPIO_InitStruct.Pull = GPIO_NOPULL;
621     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
622
623     //Configuration of D2 as input for IR Sensor
624     GPIO_InitStruct.Pin = ARD_D2_Pin;
625     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
626     GPIO_InitStruct.Pull = GPIO_PULLUP; // Pull-UP
627     HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
628
629     // Configuration of LSM6DSL EXTI 11
630     GPIO_InitStruct.Pin = LSM6DSL_INT1_EXTI11_Pin;
631     GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
632     GPIO_InitStruct.Pull = GPIO_PULLUP;
633     HAL_GPIO_Init(GPIOD, &GPIO_InitStruct); // PD11
634
635     // Enable NVIC EXTI line 13
636     HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
637 }
```

```c
646 static void UART1_Init(void) {
647     /* Pin configuration for UART. BSP_COM_Init() can do this auto
648     __HAL_RCC_GPIOB_CLK_ENABLE();
649     GPIO_InitTypeDef GPIO_InitStruct = { 0 };
650     GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
651     GPIO_InitStruct.Pin = GPIO_PIN_7 | GPIO_PIN_6;
652     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
653     GPIO_InitStruct.Pull = GPIO_NOPULL;
654     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
655     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
656
657     /* Configuring UART1 */
658     huart1.Instance = USART1;
659     huart1.Init.BaudRate = 115200;
660     huart1.Init.WordLength = UART_WORDLENGTH_8B;
661     huart1.Init.StopBits = UART_STOPBITS_1;
662     huart1.Init.Parity = UART_PARITY_NONE;
663     huart1.Init.Mode = UART_MODE_TX_RX;
664     huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
665     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
666     huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
667     huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
668     if (HAL_UART_Init(&huart1) != HAL_OK) {
669         while (1)
670             ;
671     }
672 }
```

<u>MX_GPIO_Init()</u>
Line 591 - 637.
- Configures all ports and pins to respective inputs and outputs.
- NVIC EXTI line 10 - 15 is enabled too.

<u>UART1_Init()</u>
Line 646 - 672.
- Configures port and pin for UART communication with Cyrix.

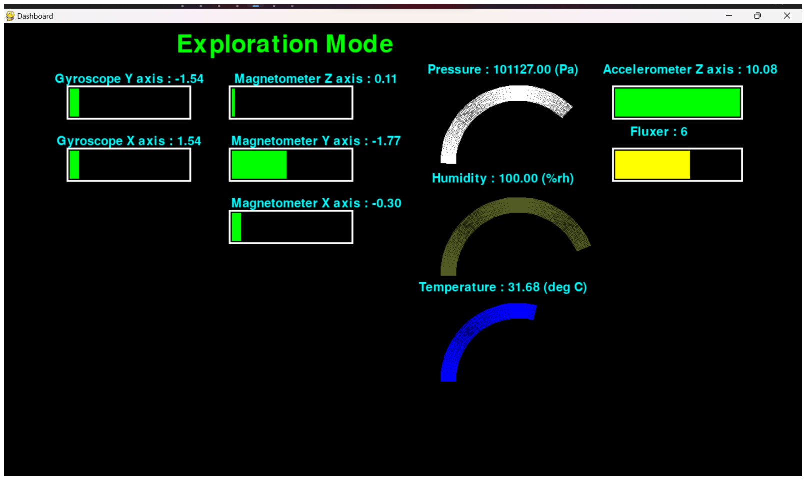
4. <u>Enhancement</u>

Infrared (IR) Sensor: We have opted for another infrared sensor to allow Pixie to detect obstacles along its path. We have connected the infrared sensor to GPIO Port (D), Pin (D2) on the base shield board.

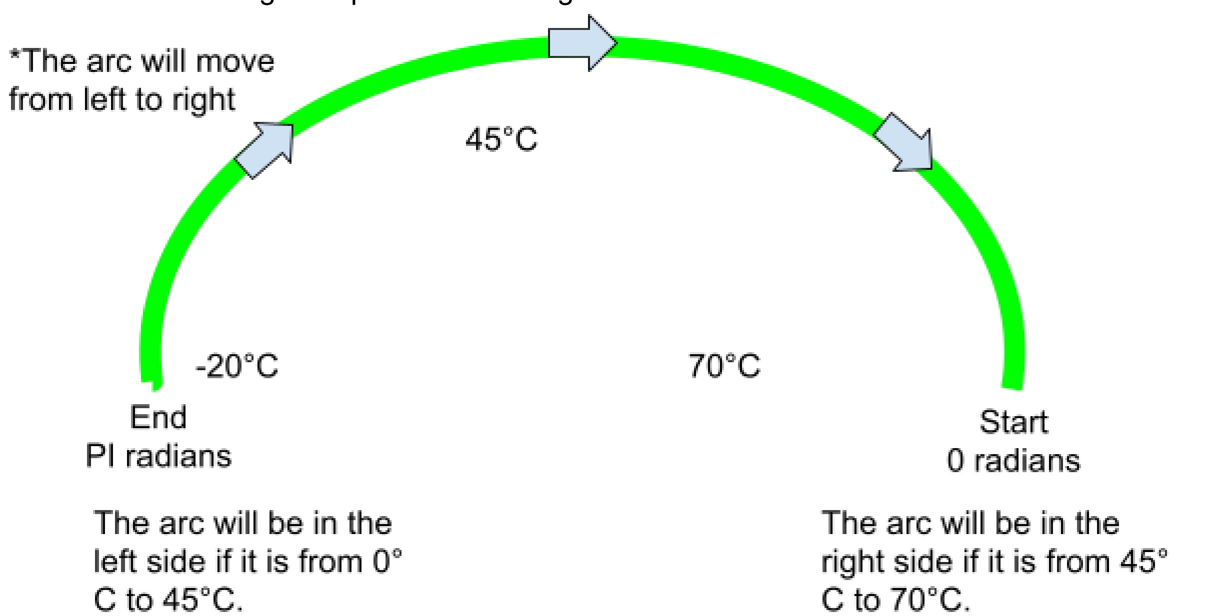
The infrared Obstruction Sensor Module has an IR transmitter and receiver mounted on the same board. IR energy is emitted and the receiver detects reflected IR energy to sense the existence of any obstacle in front. A potentiometer is located on the PCB of this electrical circuit. The potentiometer mounted on the board allows users to fine-tune the detecting range. Even in low light or full darkness, the sensor provides a very good and reliable response.

Piezo Buzzer: We have also opted to use a buzzer to connect it to the base shield board to simulate the sound of the fluxer being fired. Every 5s when the fluxer is being fired apart from showing the battery capacity left we will also hear a ‘pew’ sound from the buzzer.

Graphical User Interface:



A sensor dashboard is implemented to simulate how Cyrix will view the sensors in the lab. The dashboard is implemented using the pygame framework. The arcs representing pressure, humidity, and temperature are drawn using linear regression. For example, the temperature arc can be modeled using an equation of a straight line. **Y = mX + C**



Therefore, to display the arc moving at different temperatures, we simply need to change the start location of the arc from 0 radians (70°C) to PI radians (0°C).

Calculations

0 degrees celsius correspond to π radians.
40 degrees celsius corresponds to 0 radian.
The independent variable is the temperature and the dependent variable is the arc in radians.
(x1, y1) = (-20°C, π radians / 180°), (x2, y2) = (70°C, 0 radians / 0°)

$$m = \frac{y2 - y1}{x2 - x1} = \frac{0° - 180°}{70\,°C - (-20\,°C)} = -\frac{180}{90} = -2$$

We input one of the coordinates into the equation to determine the y intercept.

$0 = -2 * (70) + c$. Rearranging the equation, we get the c as 140, the straight line equation is $y = -2x + 140$

LSM6DSL Interrupt: We have implemented LSM6DSL Free-Fall interrupt. We configured the relevant registers required for the accelerometer to detect a free-fall. Firstly, we have to enable interrupts by setting TAP_CFG[7]. Secondly, we configure the threshold of the freefall by writing 000 to FREE_FALL[2:0]. Lastly, setting MD1_CFG[4] drives Free-Fall interrupt to INT1 pin.

5. Significant problems encountered and solutions proposed

Mode selection was not working the intended way we wanted to. While we were testing various conditions to change modes or to clear the warning states, we realized that the count of the number of times the PB being pressed within 1 second was not counted properly. To solve this issue we have to do some trouble shooting to isolate each part. We made sure that our PB was counted correctly before implementing the mode change. We have also come up with another variable called ‘press’ to be used by mode_selection(), instead of passing ‘flag’ used by the external interrupt into the function. From this we were able to learn that it is better to dedicate different variables for different uses.

We have also encountered a HardFault error. Upon troubleshooting and looking online we realized that our message to be transmitted to Cyrix was too long and we did not declare a large enough array to contain all the characters that are required to be printed. We used snprintf with a properly defined string size and this issue was corrected.

6. Issues or suggestions

While the project was interesting and challenging, we feel that the duration given to execute the assignment was a little rushed, this has not allowed us to look up more information of other sensor interrupts. Furthermore, since Pixie is a drone we felt that wireless communication could be included to make the project more realistic.

7. Conclusion

Overall, we have gained knowledge of how to utilize the I2C protocol, basic GPIO initialization, implement interrupts and draw flowcharts. Additionally, we discovered how crucial it is to study datasheets and user manuals in order to fully comprehend how external peripherals work and how to utilize them effectively. We also learned how crucial it is to use interrupts to recognize when a drone is in free fall so that the software can instantly cease operation and respond to the interrupt request more quickly than it could with polling. Most of the problems, including hard fault errors and logical inaccuracies, were resolved, and we discovered their root causes in the process of implementing this project.