Banana Milk Quadcopter Drone

Presented by Luis R, Jonathan I, Paul N, Rishita K

May 10,2019

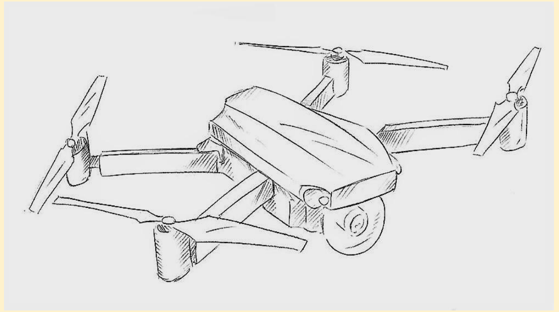
Agenda

- Introduction
- Hardware
  - -Block Diagram Design
  - -PCB Design
  - -PCB 1.0 V.S. PCB 2.0
- Software
  - -Code Workflow
  - -Code Explanation

Introduction

Our Focus

Quadcopter Drone that has the ability to elevate to a desired height and hover at such height



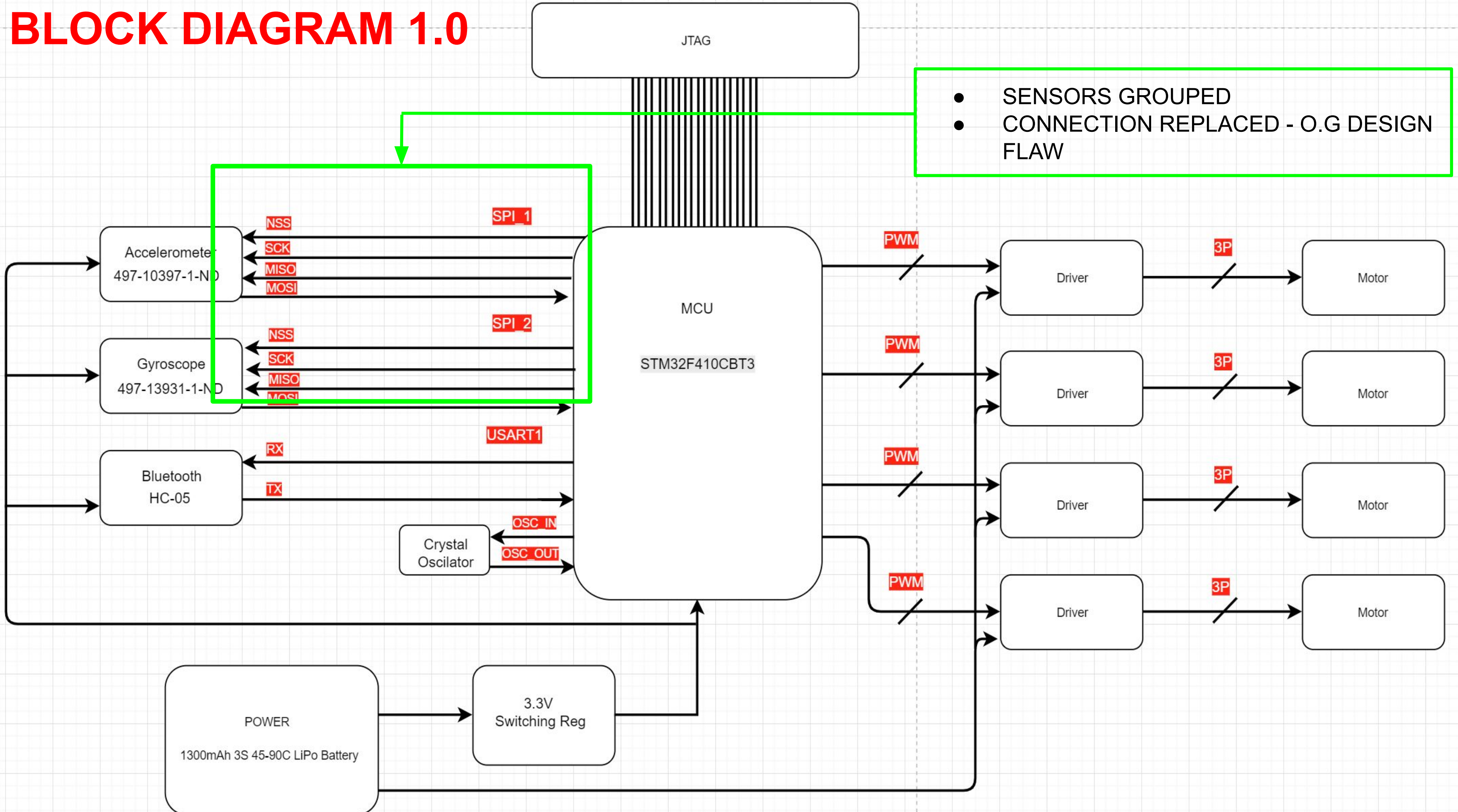
System's Sensors:
Gyroscope
Accelerometer

System's User Communication:
Bluetooth

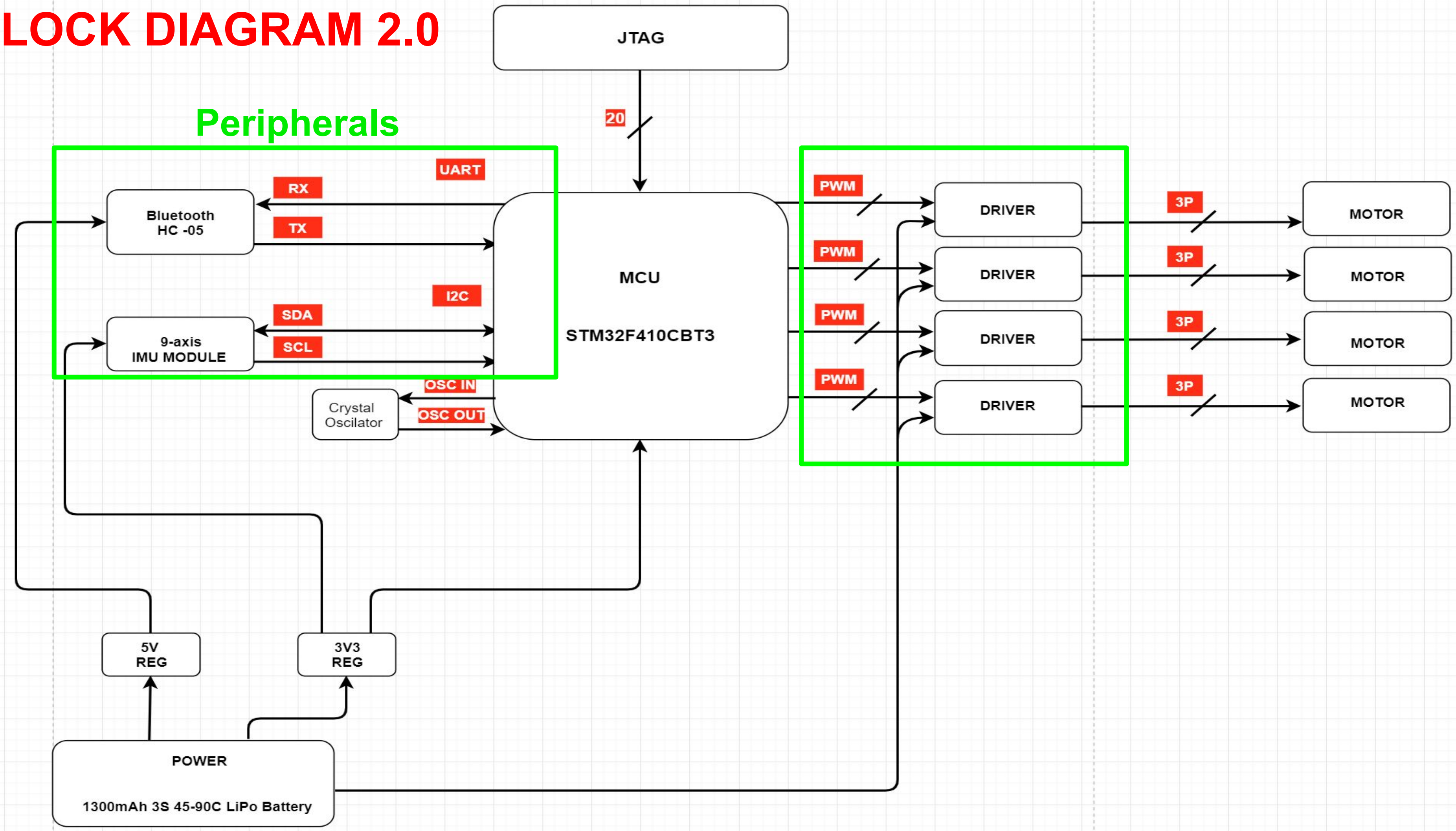
Hardware

BLOCK DIAGRAM 1.0

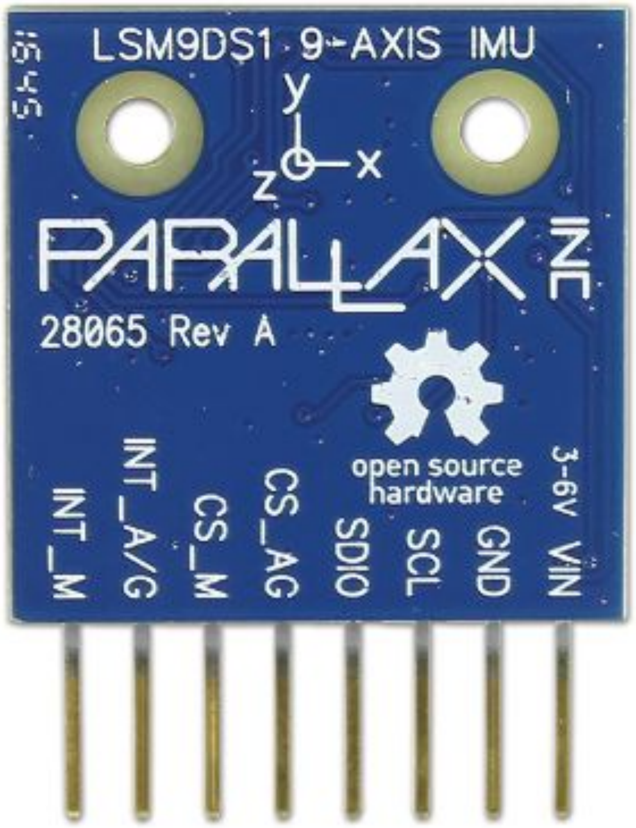
- SENSORS GROUPED
- CONNECTION REPLACED - O.G DESIGN FLAW



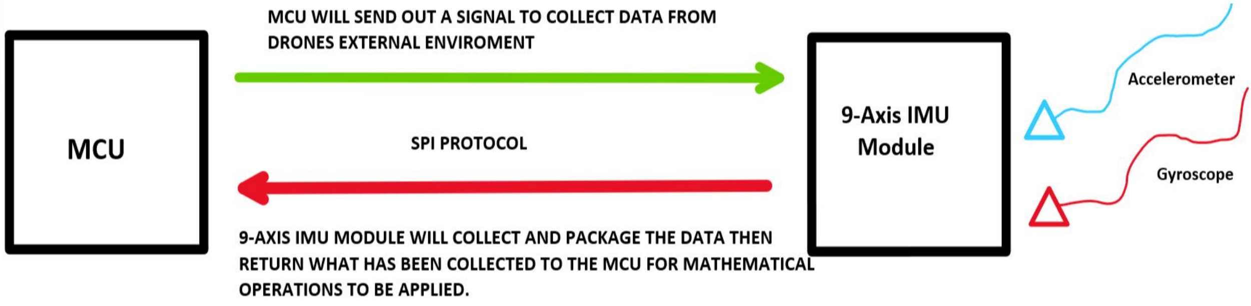
BLOCK DIAGRAM 2.0



LSM9DS1 9-axis IMU Module



- Sensor module required for PID control
- Measures linear acceleration, angular velocity, magnetic field strength, and temperature
- Visual representation of what is happening

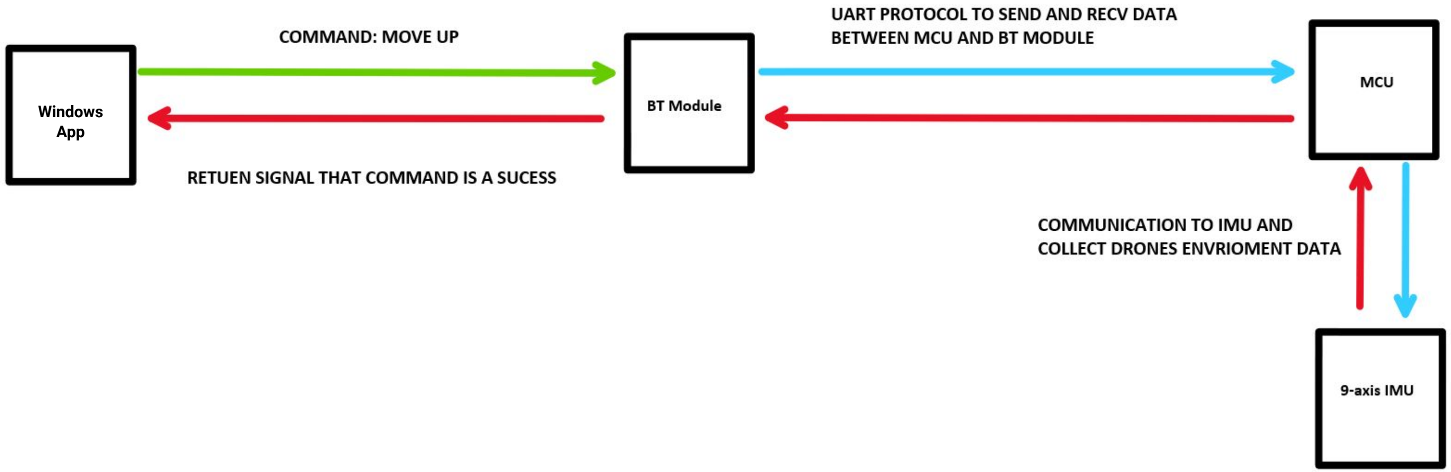


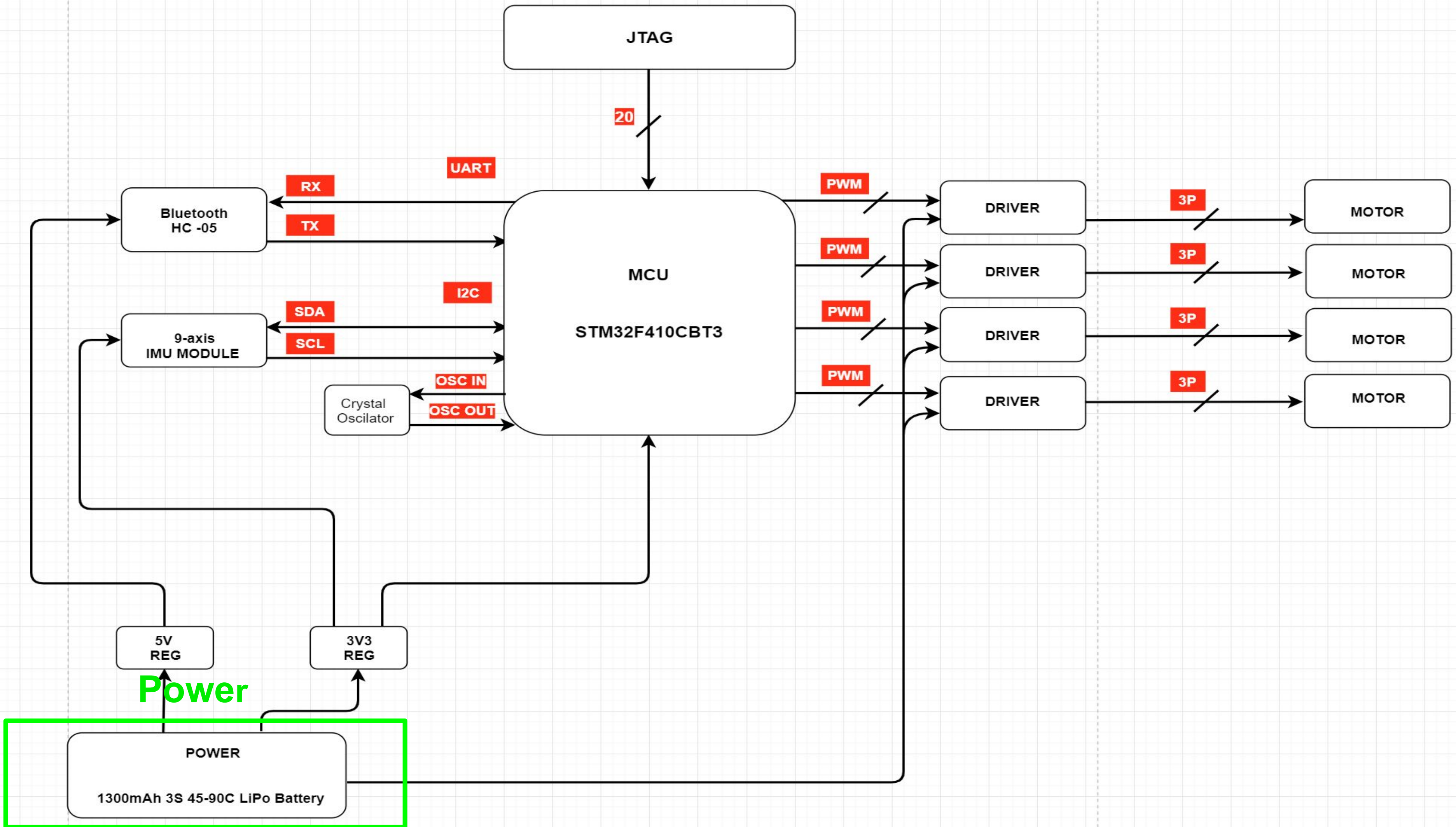
MCU WILL SEND OUT A SIGNAL TO COLLECT DATA FROM DRONES EXTERNAL ENVIROMENT
MCU
SPI PROTOCOL
9-Axis IMU Module
Accelerometer
Gyroscope
9-AXIS IMU MODULE WILL COLLECT AND PACKAGE THE DATA THEN RETURN WHAT HAS BEEN COLLECTED TO THE MCU FOR MATHEMATICAL OPERATIONS TO BE APPLIED.

Bluetooth Module - HC-05

- Bluetooth module needed to establish communication between our 2 systems

- A visual representation of what is happening :


COMMAND: MOVE UP
Windows App
BT Module
RETUEN SIGNAL THAT COMMAND IS A SUCESS
UART PROTOCOL TO SEND AND RECV DATA BETWEEN MCU AND BT MODULE
MCU
COMMUNICATION TO IMU AND COLLECT DRONES ENVRIOMENT DATA
9-axis IMU


JTAG
20
UART
RX
TX
Bluetooth HC -05
MCU
STM32F410CBT3
I2C
SDA
SCL
9-axis IMU MODULE
OSC IN
OSC OUT
Crystal Oscilator
PWM
PWM
PWM
PWM
DRIVER
DRIVER
DRIVER
DRIVER
3P
3P
3P
3P
MOTOR
MOTOR
MOTOR
MOTOR
5V REG
3V3 REG
Power
POWER
1300mAh 3S 45-90C LiPo Battery

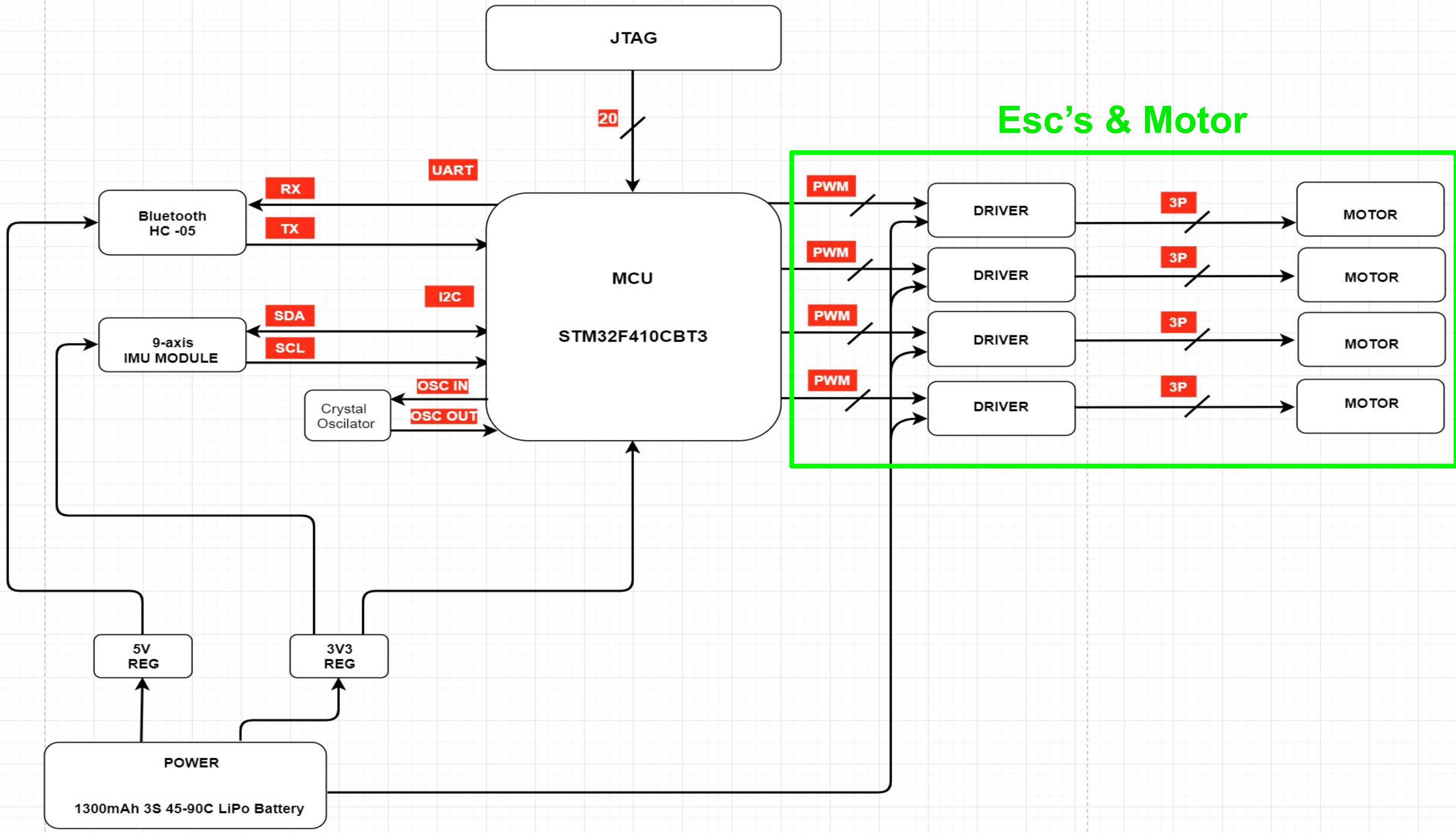
LiPo Battery 1300mAh 3S 45-90C

- Provides power to flight controller, motors, and ESCs
- Ability to dissipate/discharge power quick enough to the subsystems
- Voltage switching

VOLTAGE SWITCHING REGULATORS

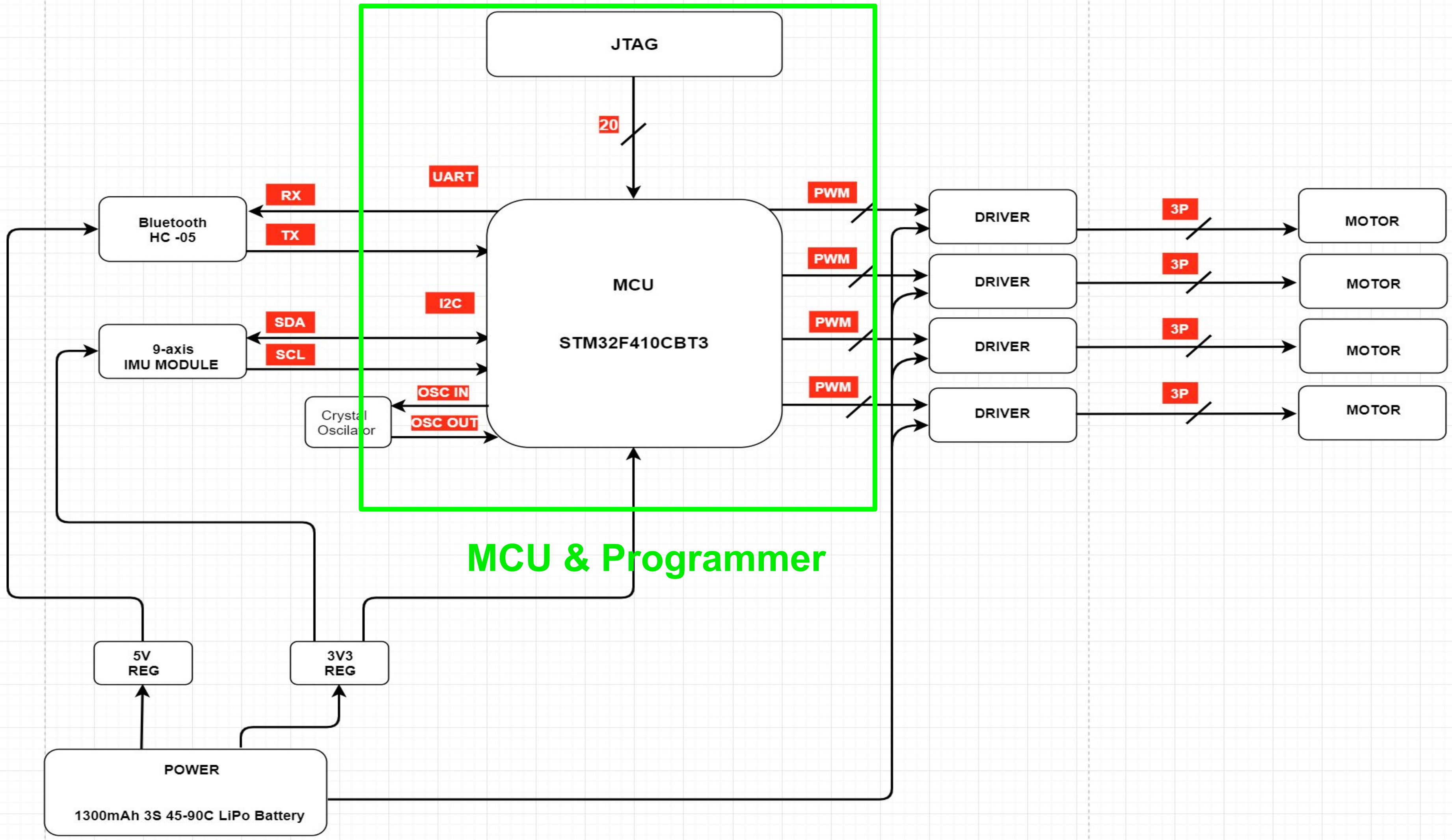
- Drop the voltage, efficiently, so that our parts can operate within their voltage requirements

- We needed 2 voltage regulators because 1 module required 5v to operate while the remain parts were able to operate at 3V3

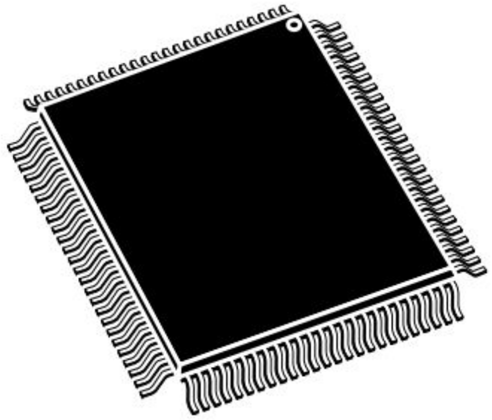

JTAG
20
Esc’s & Motor
UART
RX
TX
Bluetooth HC -05
MCU
STM32F410CBT3
I2C
SDA
SCL
9-axis IMU MODULE
OSC IN
OSC OUT
Crystal Oscilator
PWM
PWM
PWM
PWM
DRIVER
DRIVER
DRIVER
DRIVER
3P
3P
3P
3P
MOTOR
MOTOR
MOTOR
MOTOR
5V REG
3V3 REG
POWER
1300mAh 3S 45-90C LiPo Battery

30A ESCs with 2217-1000kV Brushless Motors

- ESCs and motors generate a lift force on the drone
- ESCs manage speed of motors and direction of rotation
- Three-phase system

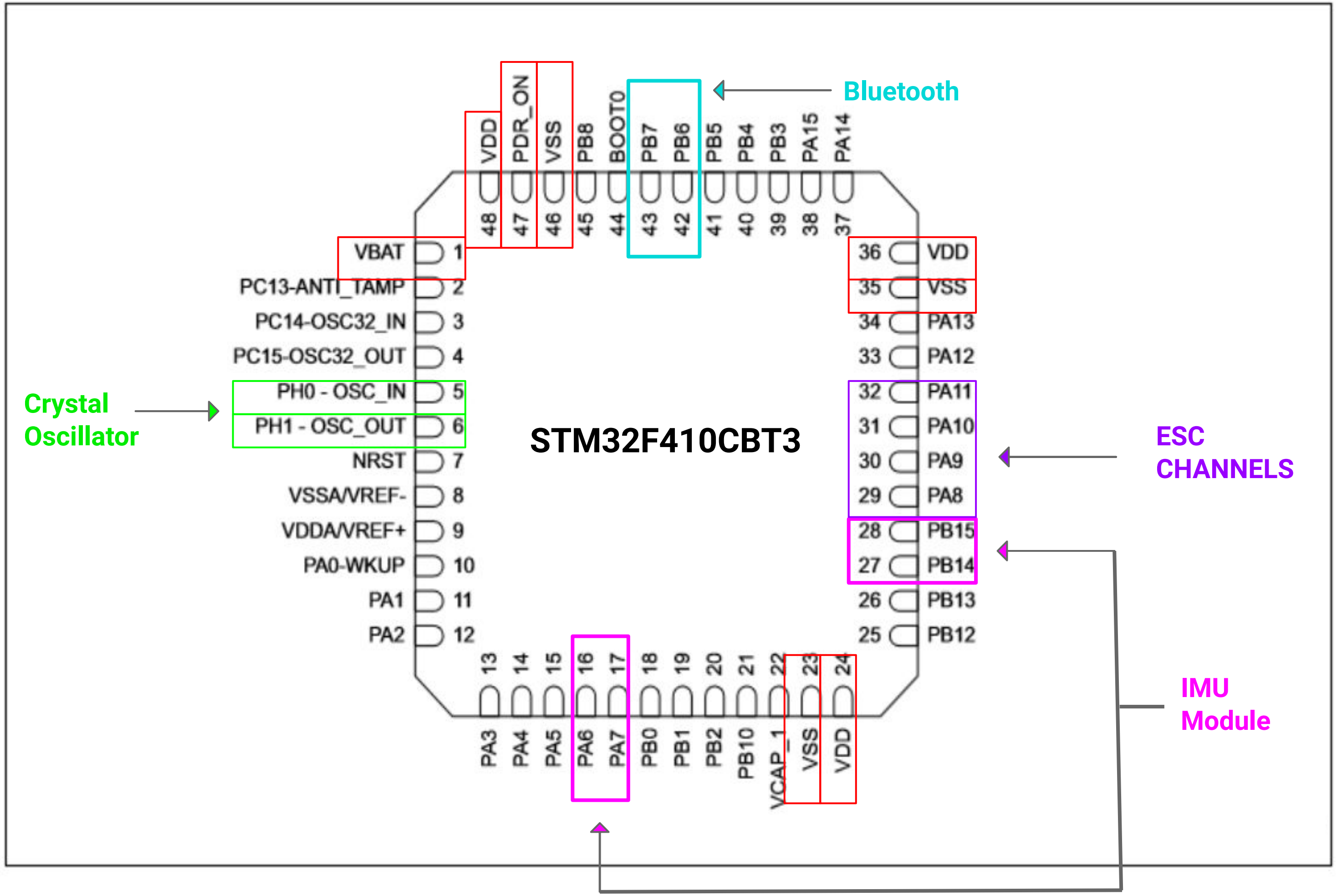

JTAG
20
UART
RX
TX
Bluetooth HC -05
MCU
STM32F410CBT3
I2C
SDA
SCL
9-axis IMU MODULE
OSC IN
OSC OUT
Crystal Oscilator
PWM
PWM
PWM
PWM
DRIVER
DRIVER
DRIVER
DRIVER
3P
3P
3P
3P
MOTOR
MOTOR
MOTOR
MOTOR
MCU & Programmer
5V REG
3V3 REG
POWER
1300mAh 3S 45-90C LiPo Battery

MICROPROCESSOR

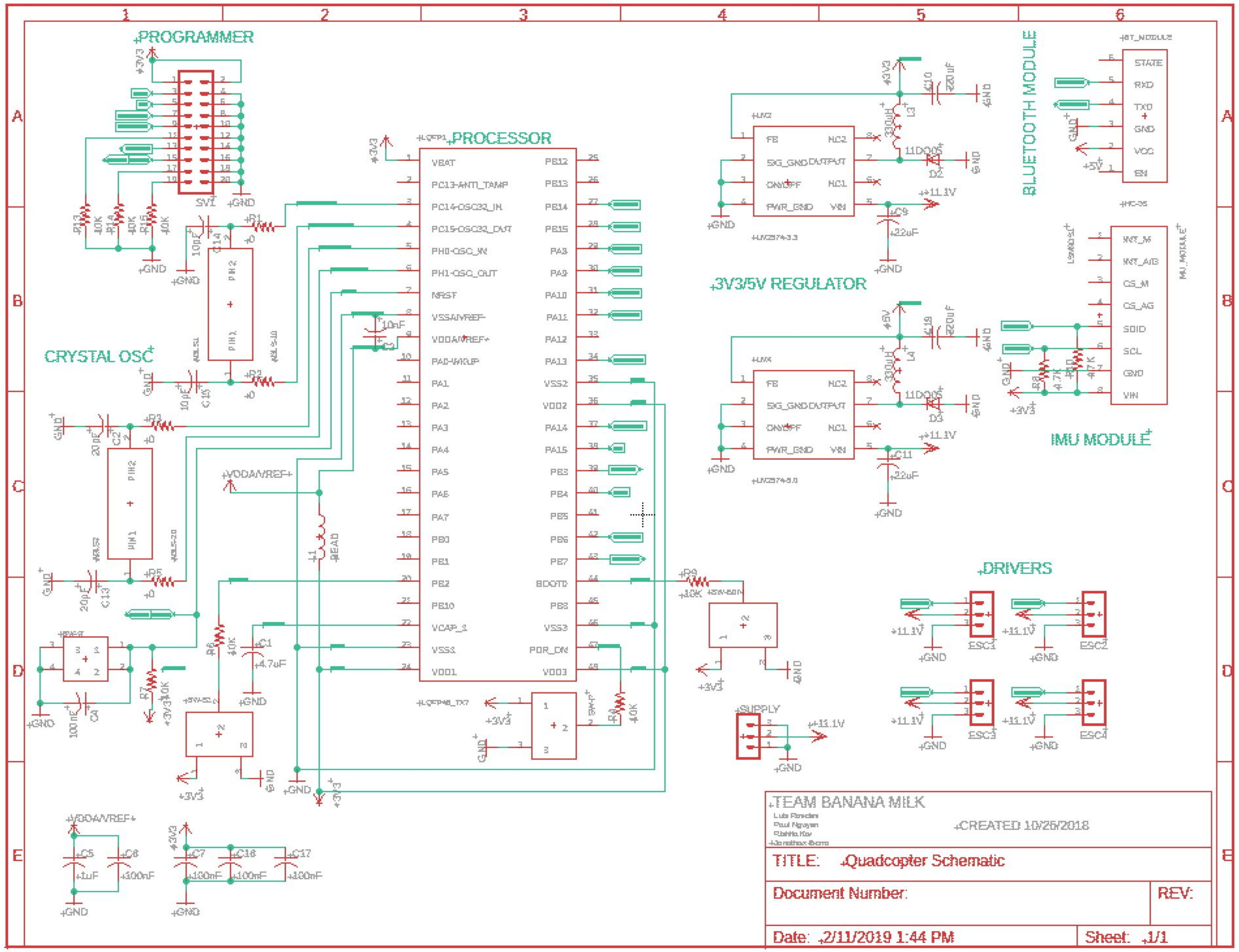

STM32F410CBT3

- Responsible for processing data from peripherals:
    - Gyroscope Sensor
    - Accelerometer Sensor
    - Bluetooth
- Performing software logic to communicate to other subsystems

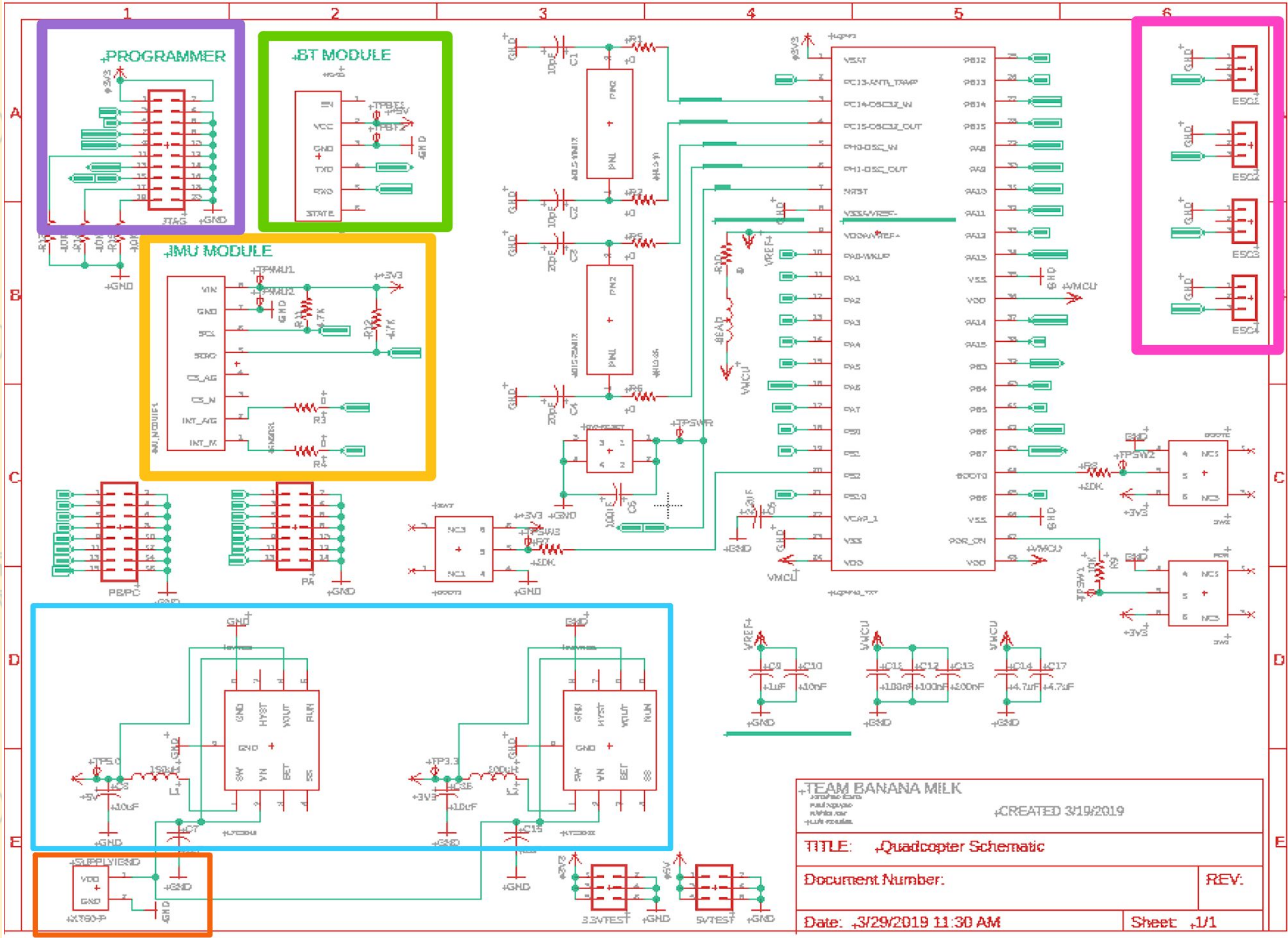
Subsystem-1: MCU



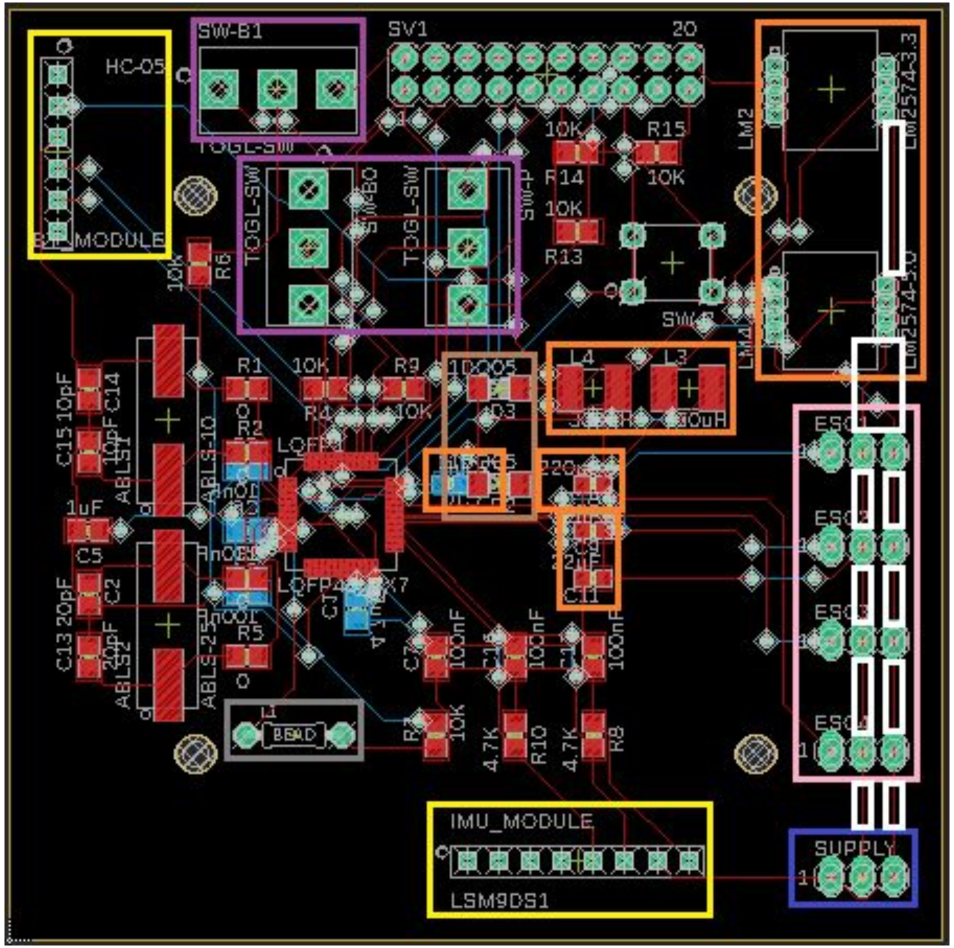
Schematic First Draft



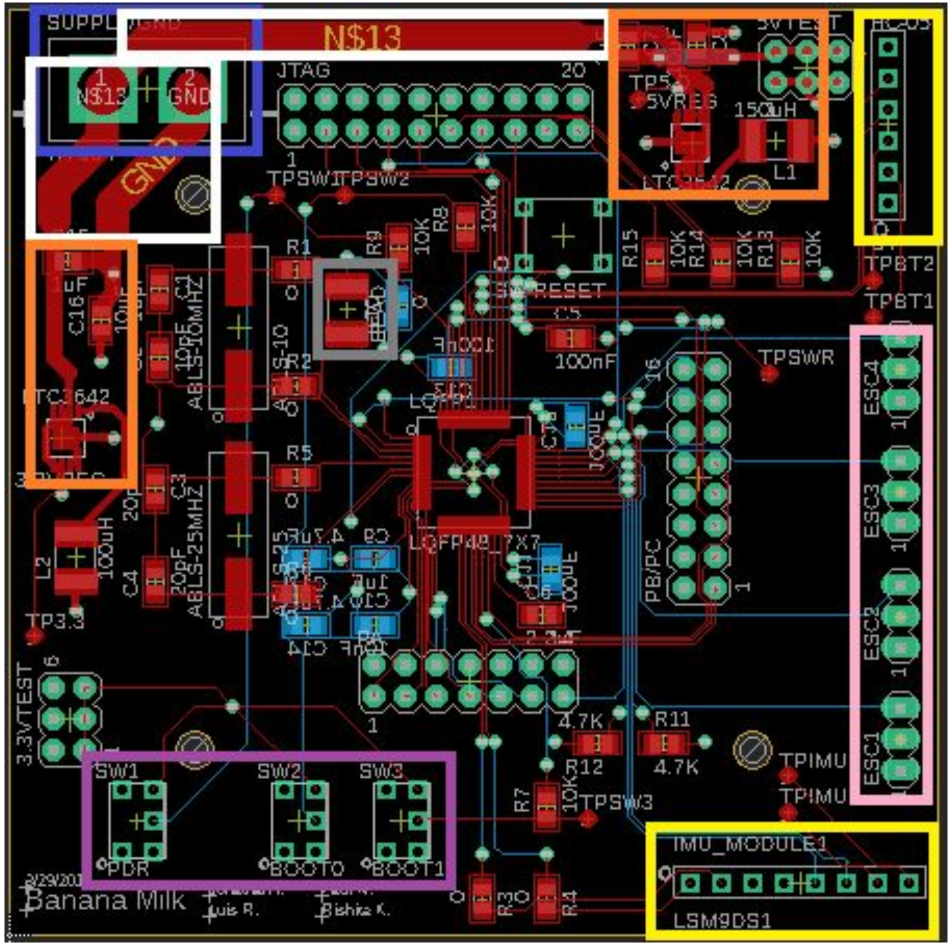
Schematic Second Draft



Layout
First Draft

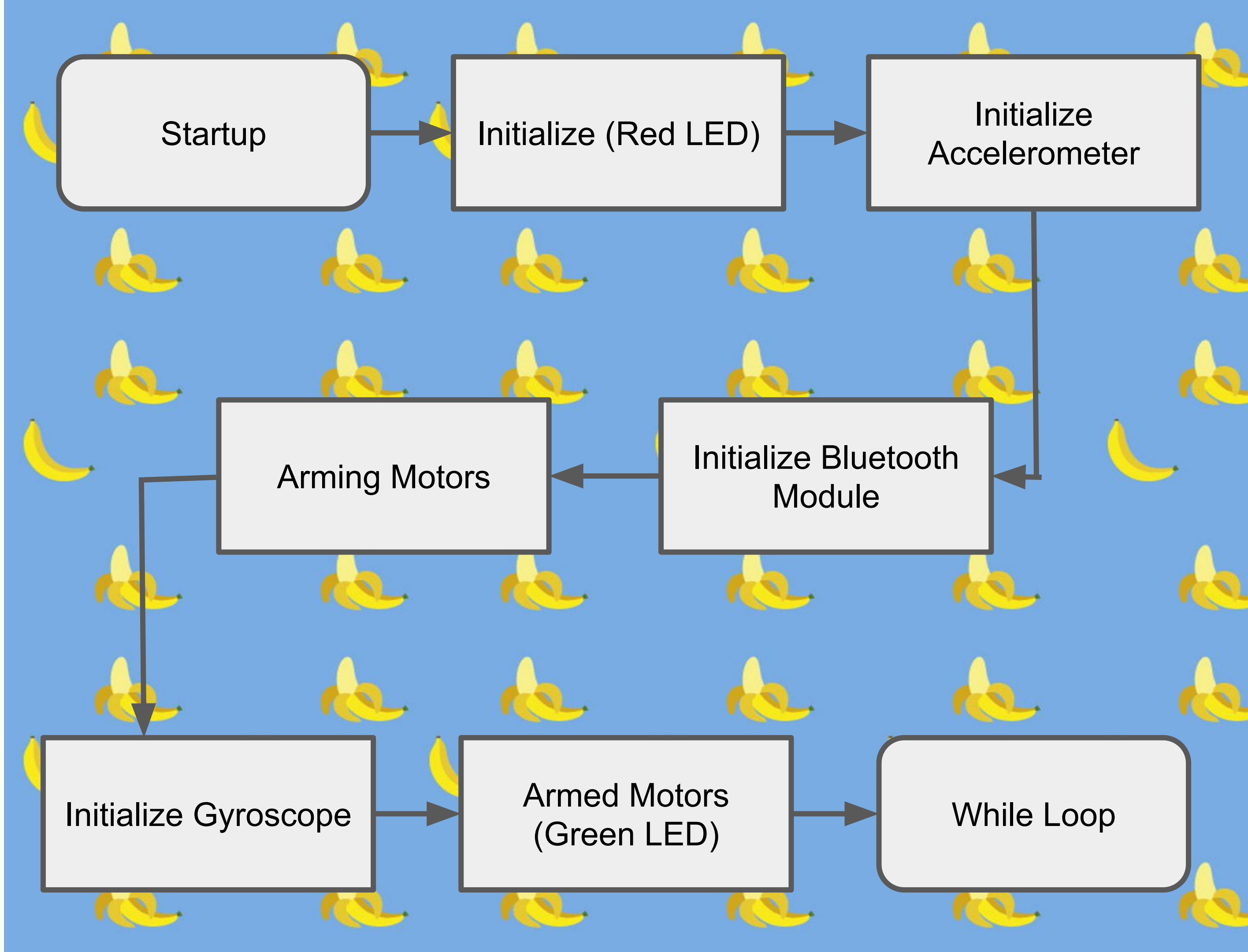


Layout
Second Draft

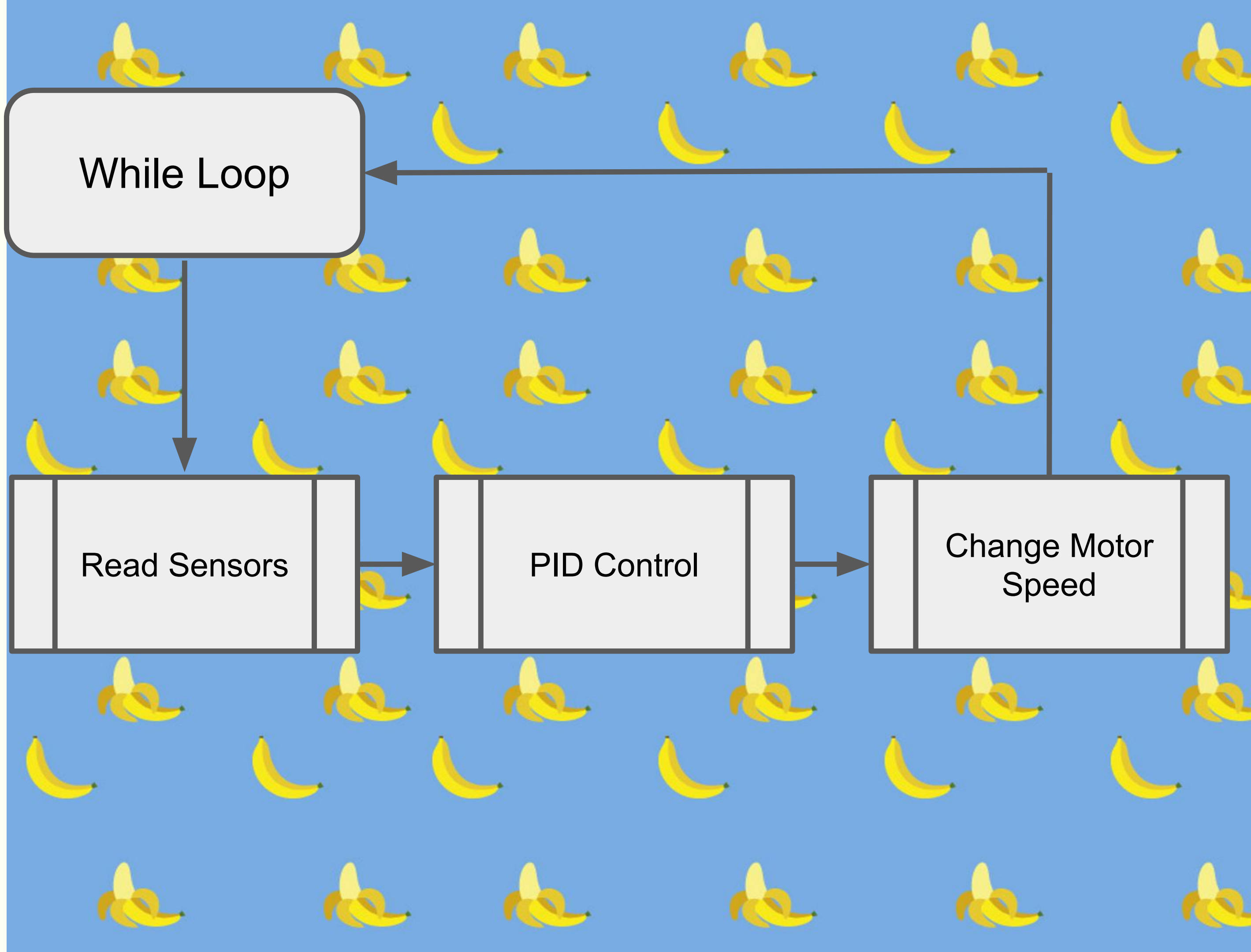


Software

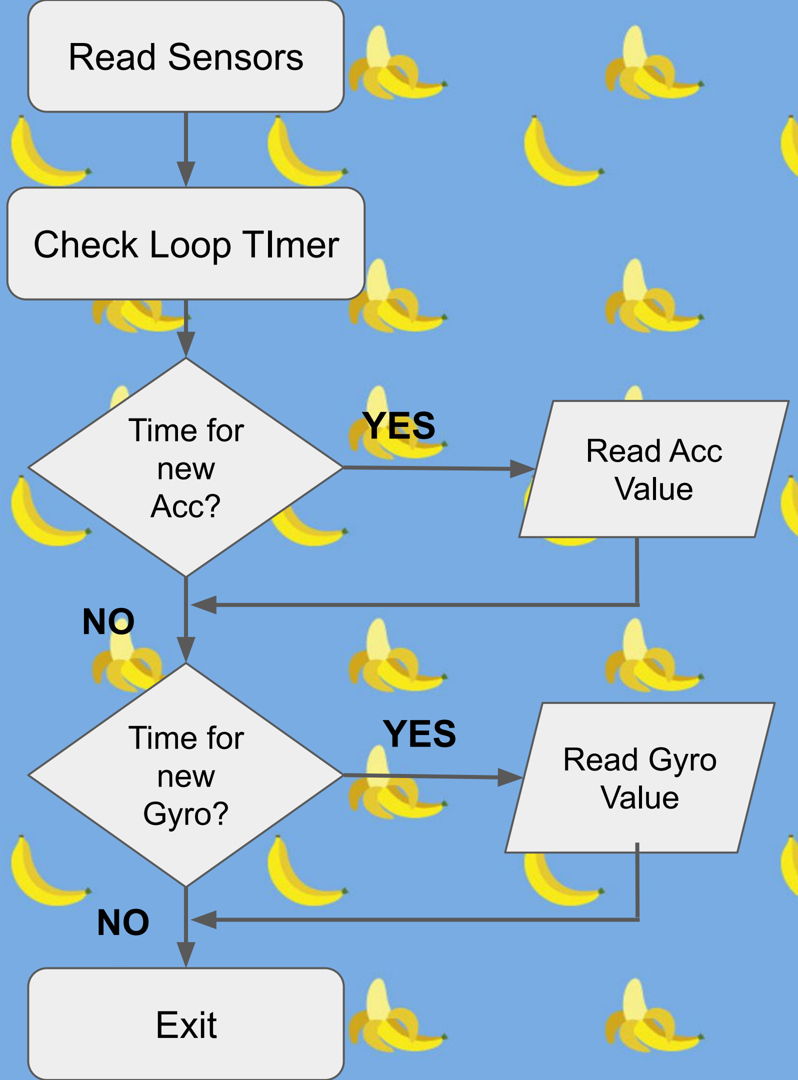
Code Workflow (Initialization Stage)



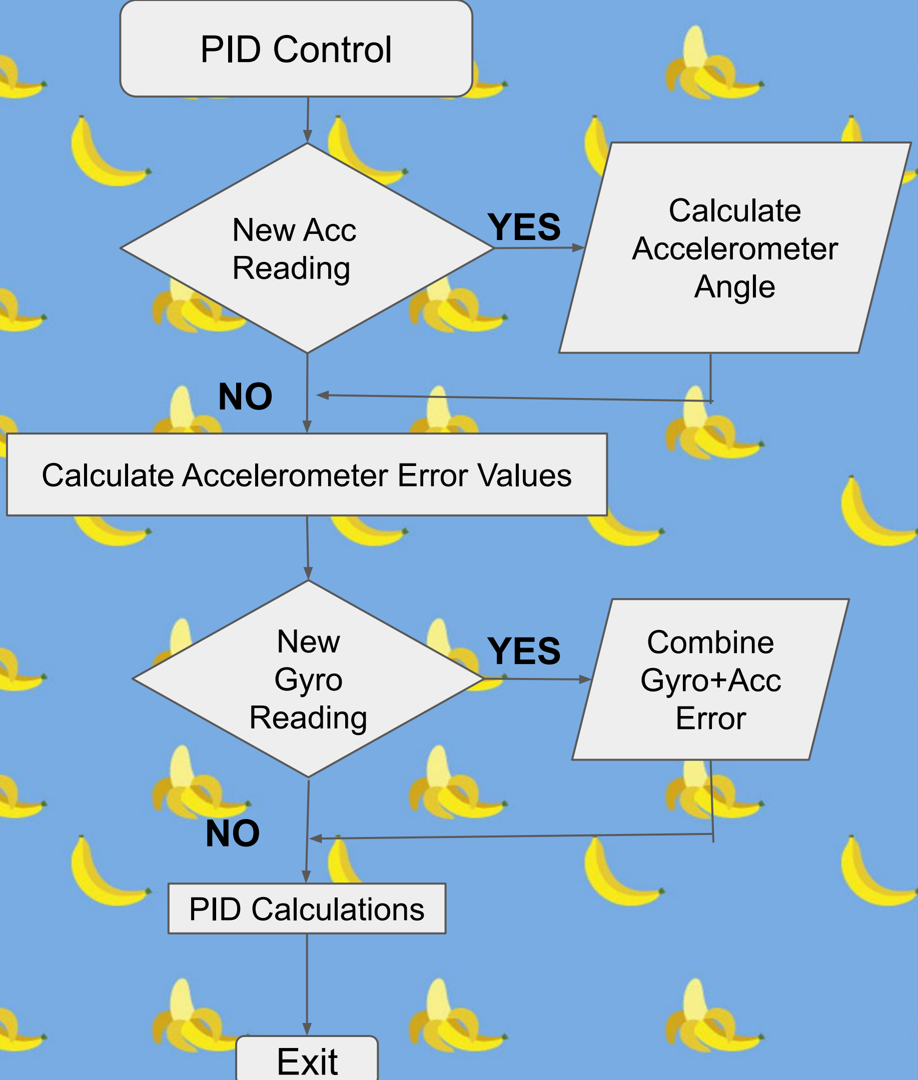
Code Workflow (Main Loop)



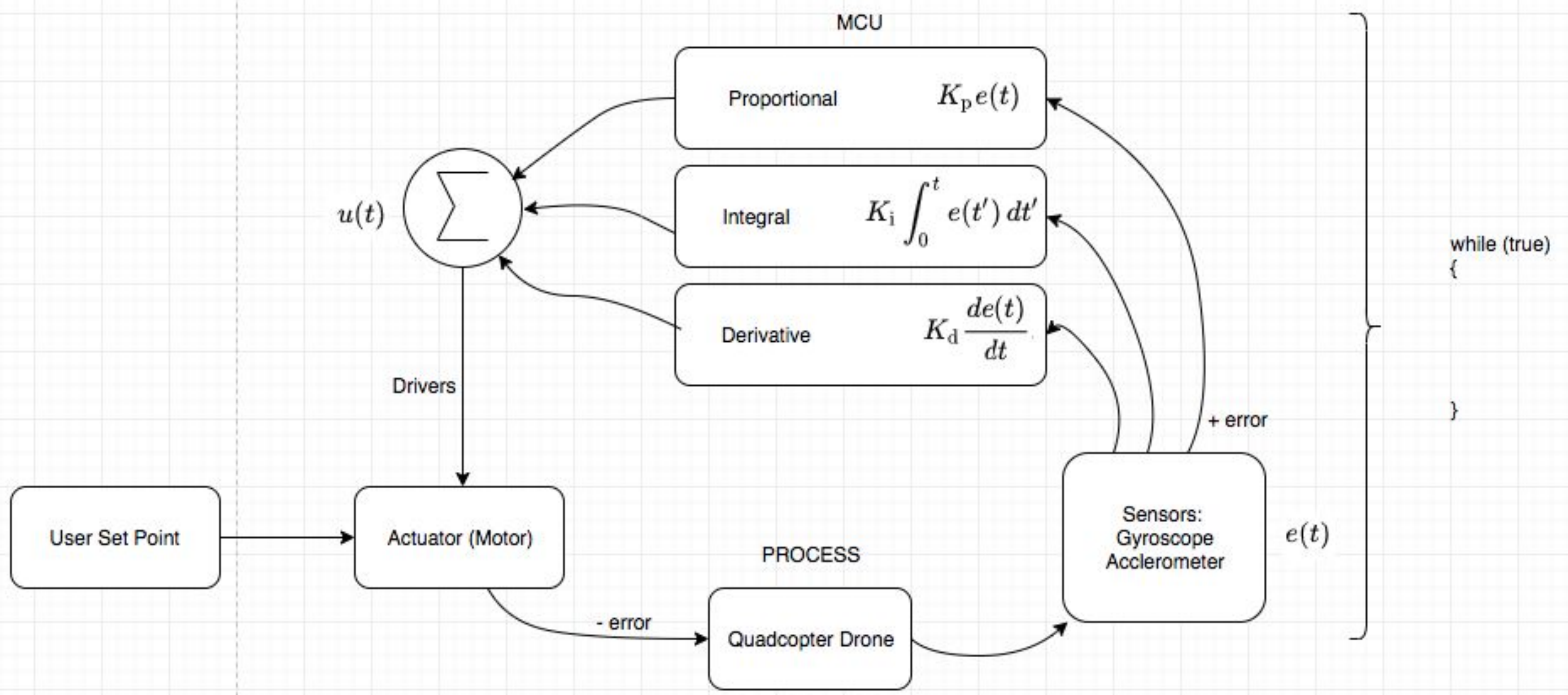
Code Workflow (Sensor Details)

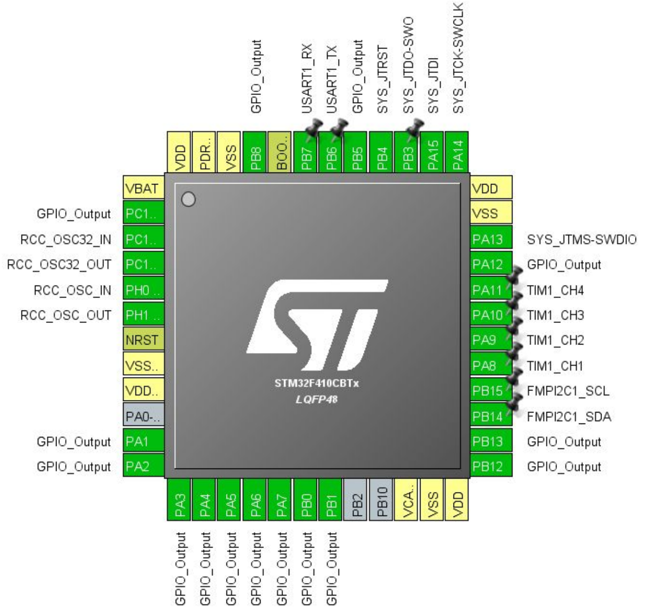


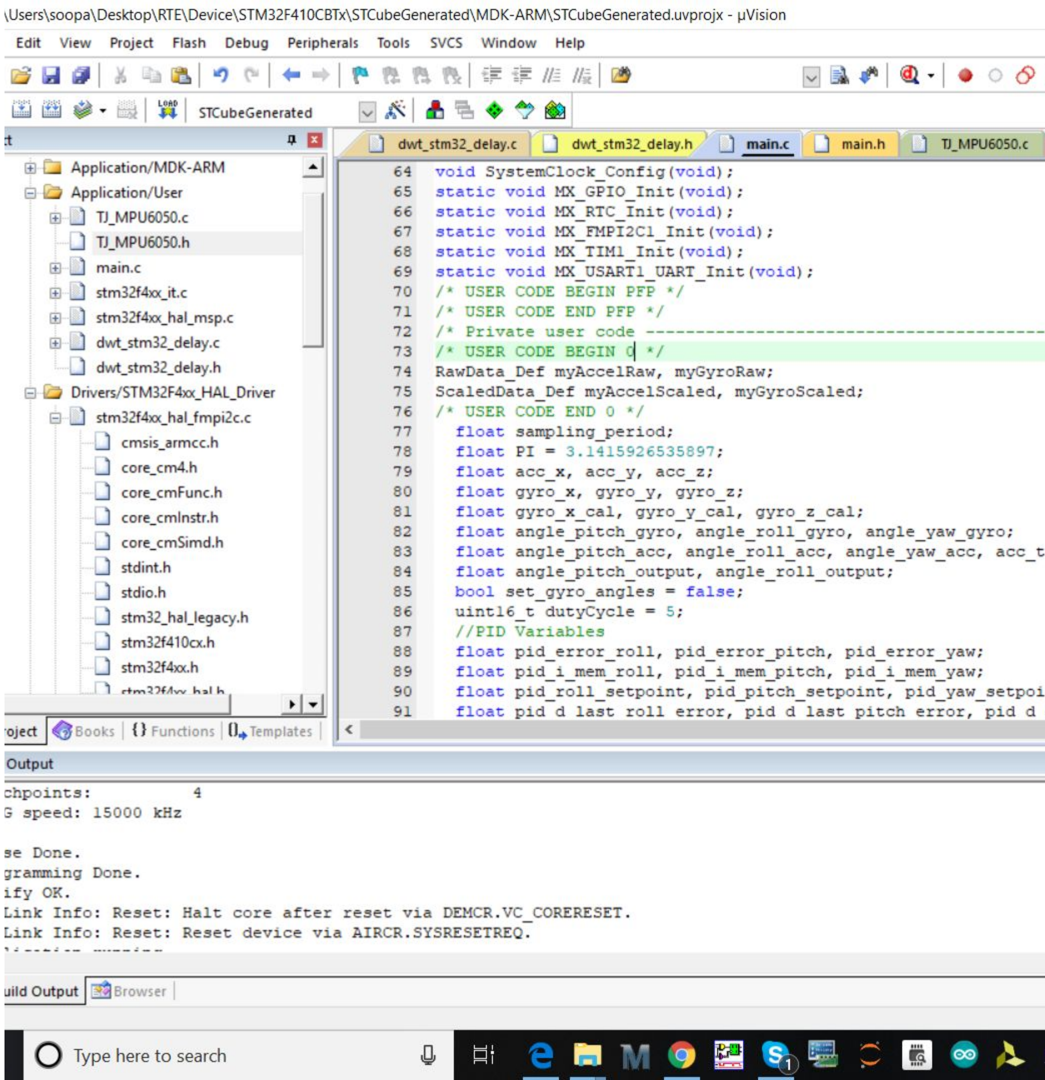
Code Workflow (Controller Details)



Code Workflow (PID Loop)







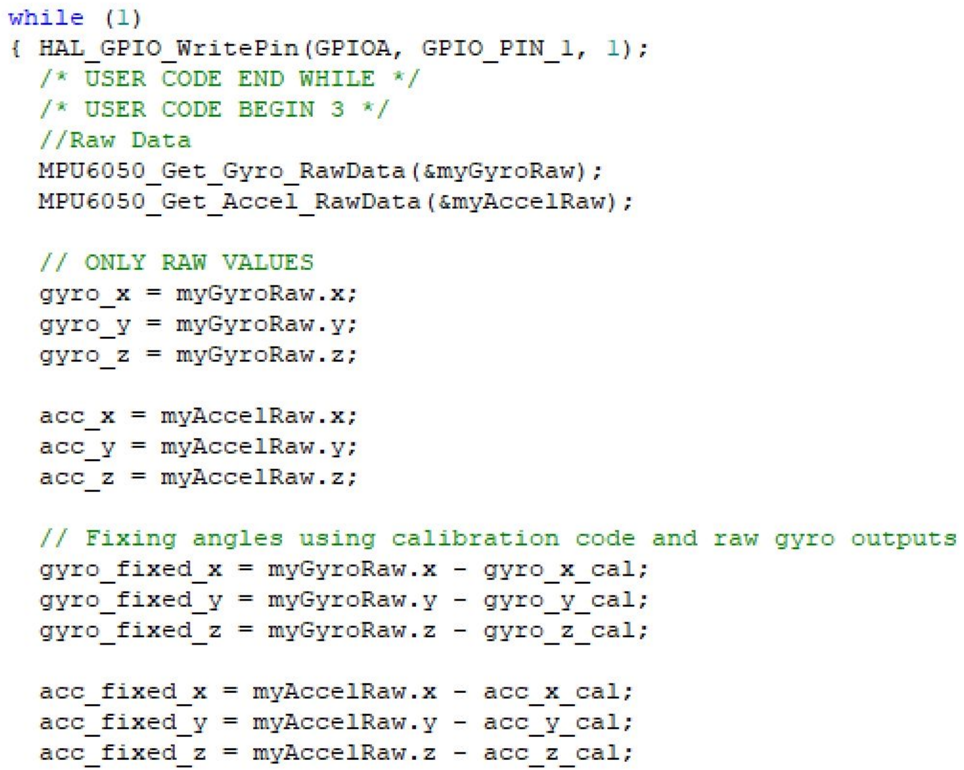
```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_RTC_Init(void);
static void MX_FMPI2C1_Init(void);
static void MX_TIM1_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------
/* USER CODE BEGIN 0 */
RawData_Def myAccelRaw, myGyroRaw;
ScaledData_Def myAccelScaled, myGyroScaled;
/* USER CODE END 0 */
  float sampling_period;
  float PI = 3.1415926535897;
  float acc_x, acc_y, acc_z;
  float gyro_x, gyro_y, gyro_z;
  float gyro_x_cal, gyro_y_cal, gyro_z_cal;
  float angle_pitch_gyro, angle_roll_gyro, angle_yaw_gyro;
  float angle_pitch_acc, angle_roll_acc, angle_yaw_acc, acc_t
  float angle_pitch_output, angle_roll_output;
  bool set_gyro_angles = false;
  uint16_t dutyCycle = 5;
  //PID Variables
  float pid_error_roll, pid_error_pitch, pid_error_yaw;
  float pid_i_mem_roll, pid_i_mem_pitch, pid_i_mem_yaw;
  float pid_roll_setpoint, pid_pitch_setpoint, pid_yaw_setpoi
  float pid_d_last_roll_error, pid_d_last_pitch_error, pid_d
```

```
chpoints:          4
G speed: 15000 kHz

se Done.
gramming Done.
ify OK.
Link Info: Reset: Halt core after reset via DEMCR.VC_CORERESET.
Link Info: Reset: Reset device via AIRCR.SYSRESETREQ.
```

stm32cubemx/keil uVision5

Reading Sensor Values from the MPU6050

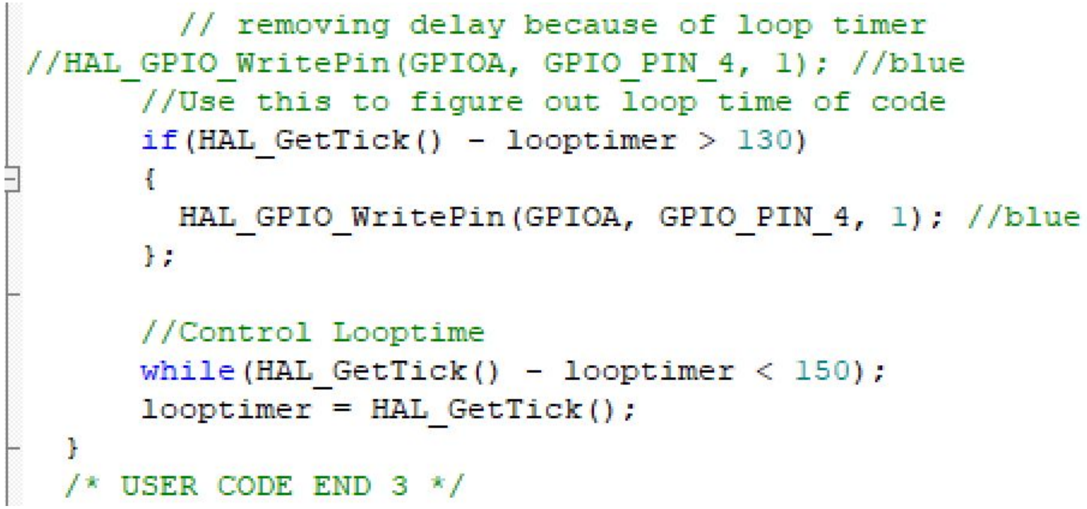
We adapted driver code in order to interfacing and obtain raw/scaled values



```
while (1)
{ HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 1);
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
  //Raw Data
  MPU6050_Get_Gyro_RawData(&myGyroRaw);
  MPU6050_Get_Accel_RawData(&myAccelRaw);

  // ONLY RAW VALUES
  gyro_x = myGyroRaw.x;
  gyro_y = myGyroRaw.y;
  gyro_z = myGyroRaw.z;

  acc_x = myAccelRaw.x;
  acc_y = myAccelRaw.y;
  acc_z = myAccelRaw.z;

  // Fixing angles using calibration code and raw gyro outputs
  gyro_fixed_x = myGyroRaw.x - gyro_x_cal;
  gyro_fixed_y = myGyroRaw.y - gyro_y_cal;
  gyro_fixed_z = myGyroRaw.z - gyro_z_cal;

  acc_fixed_x = myAccelRaw.x - acc_x_cal;
  acc_fixed_y = myAccelRaw.y - acc_y_cal;
  acc_fixed_z = myAccelRaw.z - acc_z_cal;
```

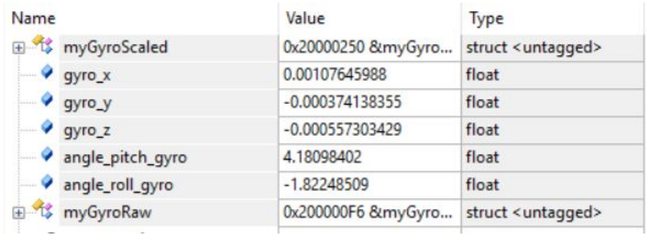
Looptimer

We tested the runtime of our code and chose a 150 ms looptimer. Getting this right is essential for gyroscope calculations.

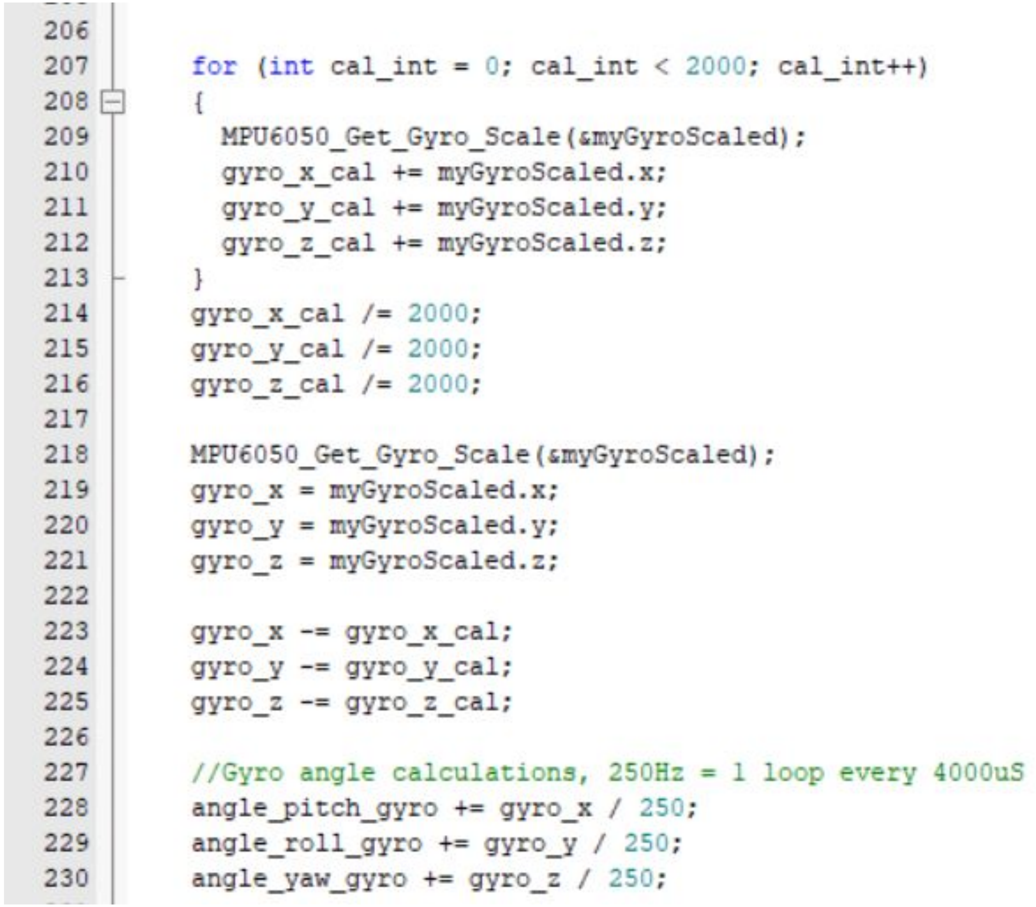


```
            // removing delay because of loop timer
//HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, 1); //blue
        //Use this to figure out loop time of code
        if(HAL_GetTick() - looptimer > 130)
        {
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, 1); //blue
        };

        //Control Looptime
        while(HAL_GetTick() - looptimer < 150);
        looptimer = HAL_GetTick();
   }
   /* USER CODE END 3 */
```

Gyroscope Angles



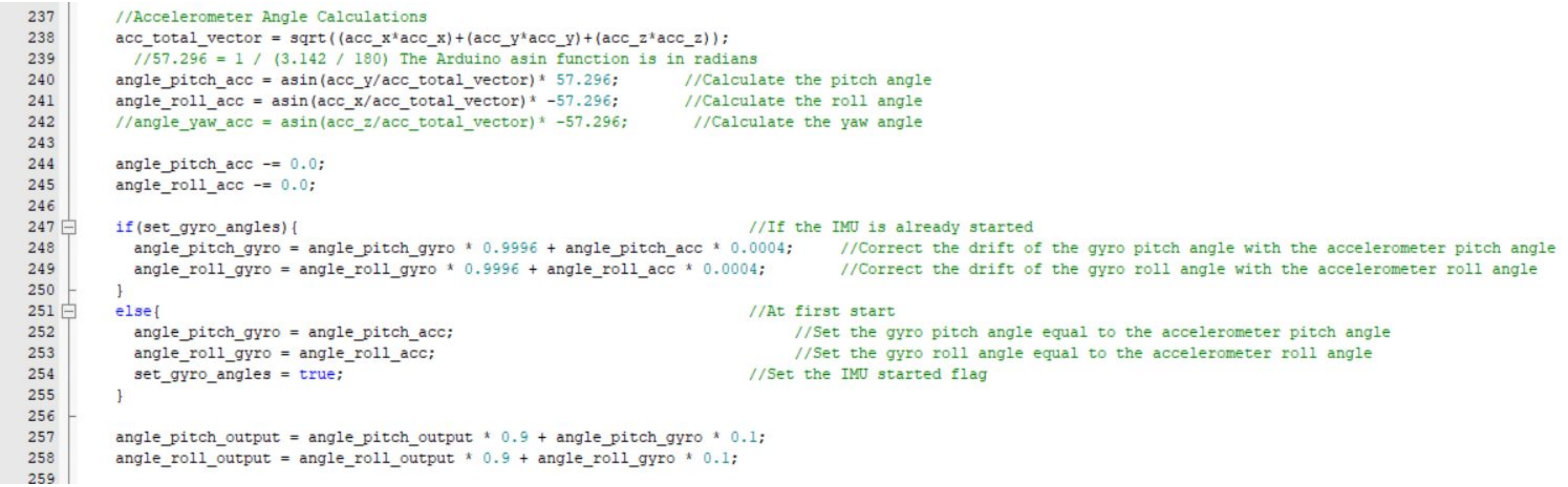
Name	Value	Type
myGyroScaled	0x20000250 &myGyro...	struct <untagged>
gyro_x	0.00107645988	float
gyro_y	-0.000374138355	float
gyro_z	-0.000557303429	float
angle_pitch_gyro	4.18098402	float
angle_roll_gyro	-1.82248509	float
myGyroRaw	0x200000F6 &myGyro...	struct <untagged>



```
    for (int cal_int = 0; cal_int < 2000; cal_int++)
    {
      MPU6050_Get_Gyro_Scale(&myGyroScaled);
      gyro_x_cal += myGyroScaled.x;
      gyro_y_cal += myGyroScaled.y;
      gyro_z_cal += myGyroScaled.z;
    }
    gyro_x_cal /= 2000;
    gyro_y_cal /= 2000;
    gyro_z_cal /= 2000;

    MPU6050_Get_Gyro_Scale(&myGyroScaled);
    gyro_x = myGyroScaled.x;
    gyro_y = myGyroScaled.y;
    gyro_z = myGyroScaled.z;

    gyro_x -= gyro_x_cal;
    gyro_y -= gyro_y_cal;
    gyro_z -= gyro_z_cal;

    //Gyro angle calculations, 250Hz = 1 loop every 4000uS
    angle_pitch_gyro += gyro_x / 250;
    angle_roll_gyro += gyro_y / 250;
    angle_yaw_gyro += gyro_z / 250;
```

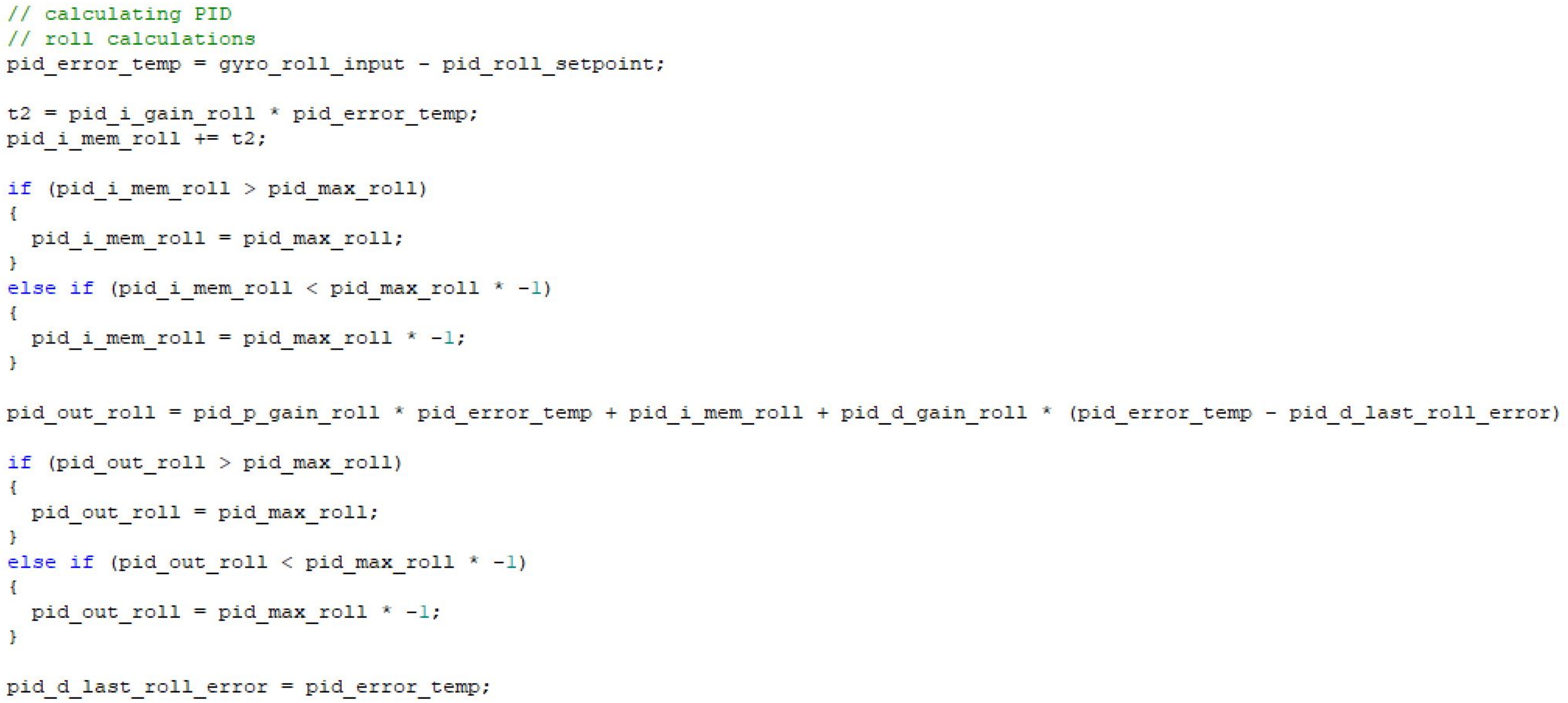
Accelerometer Angles and Consolidation

Gyro Angles + Accel Angles = robust IMU Measurement Angles for PID Calc



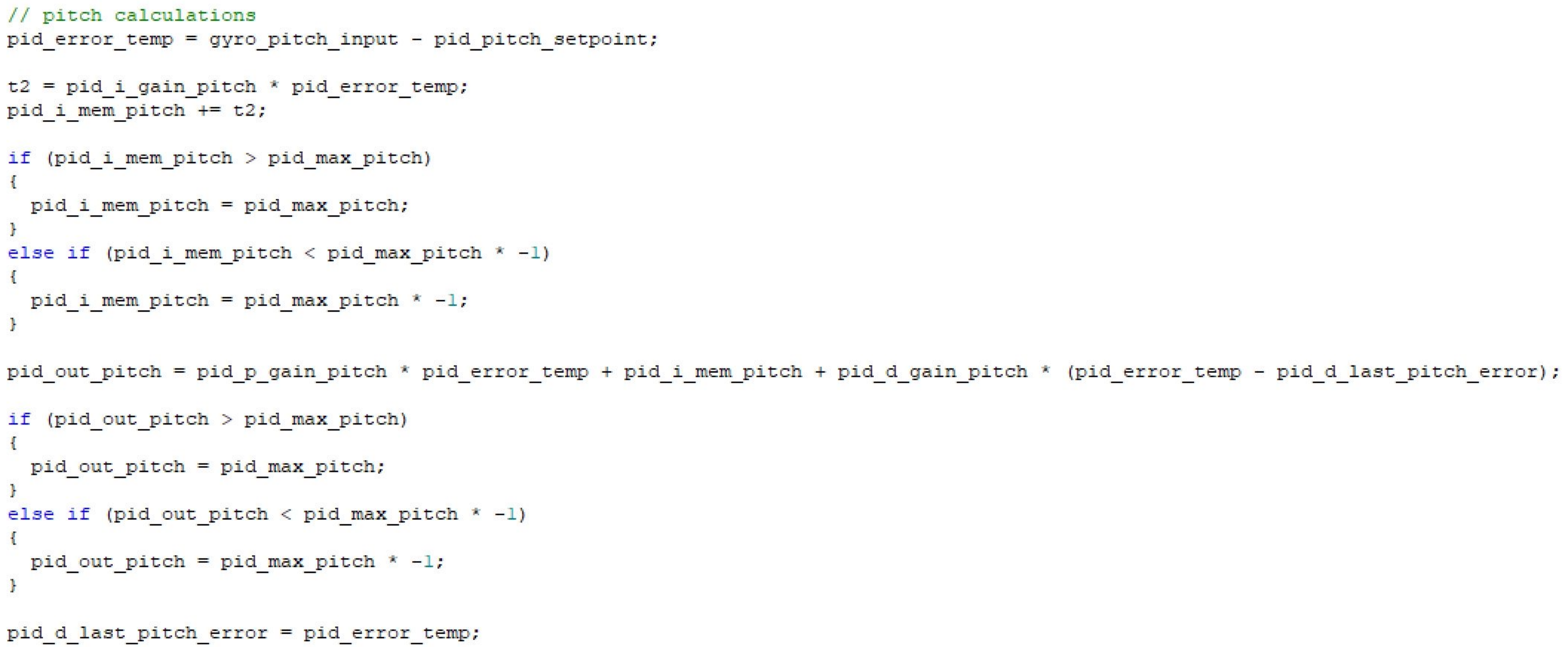
```
  //Accelerometer Angle Calculations
  acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));
    //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
  angle_pitch_acc = asin(acc_y/acc_total_vector)* 57.296;       //Calculate the pitch angle
  angle_roll_acc = asin(acc_x/acc_total_vector)* -57.296;       //Calculate the roll angle
  //angle_yaw_acc = asin(acc_z/acc_total_vector)* -57.296;       //Calculate the yaw angle

  angle_pitch_acc -= 0.0;
  angle_roll_acc -= 0.0;

  if(set_gyro_angles){                                            //If the IMU is already started
    angle_pitch_gyro = angle_pitch_gyro * 0.9996 + angle_pitch_acc * 0.0004;     //Correct the drift of the gyro pitch angle with the accelerometer pitch angle
    angle_roll_gyro = angle_roll_gyro * 0.9996 + angle_roll_acc * 0.0004;        //Correct the drift of the gyro roll angle with the accelerometer roll angle
  }
  else{                                                           //At first start
    angle_pitch_gyro = angle_pitch_acc;                                //Set the gyro pitch angle equal to the accelerometer pitch angle
    angle_roll_gyro = angle_roll_acc;                                  //Set the gyro roll angle equal to the accelerometer roll angle
    set_gyro_angles = true;                                       //Set the IMU started flag
  }

  angle_pitch_output = angle_pitch_output * 0.9 + angle_pitch_gyro * 0.1;
  angle_roll_output = angle_roll_output * 0.9 + angle_roll_gyro * 0.1;
```

PID Calculations (Roll)



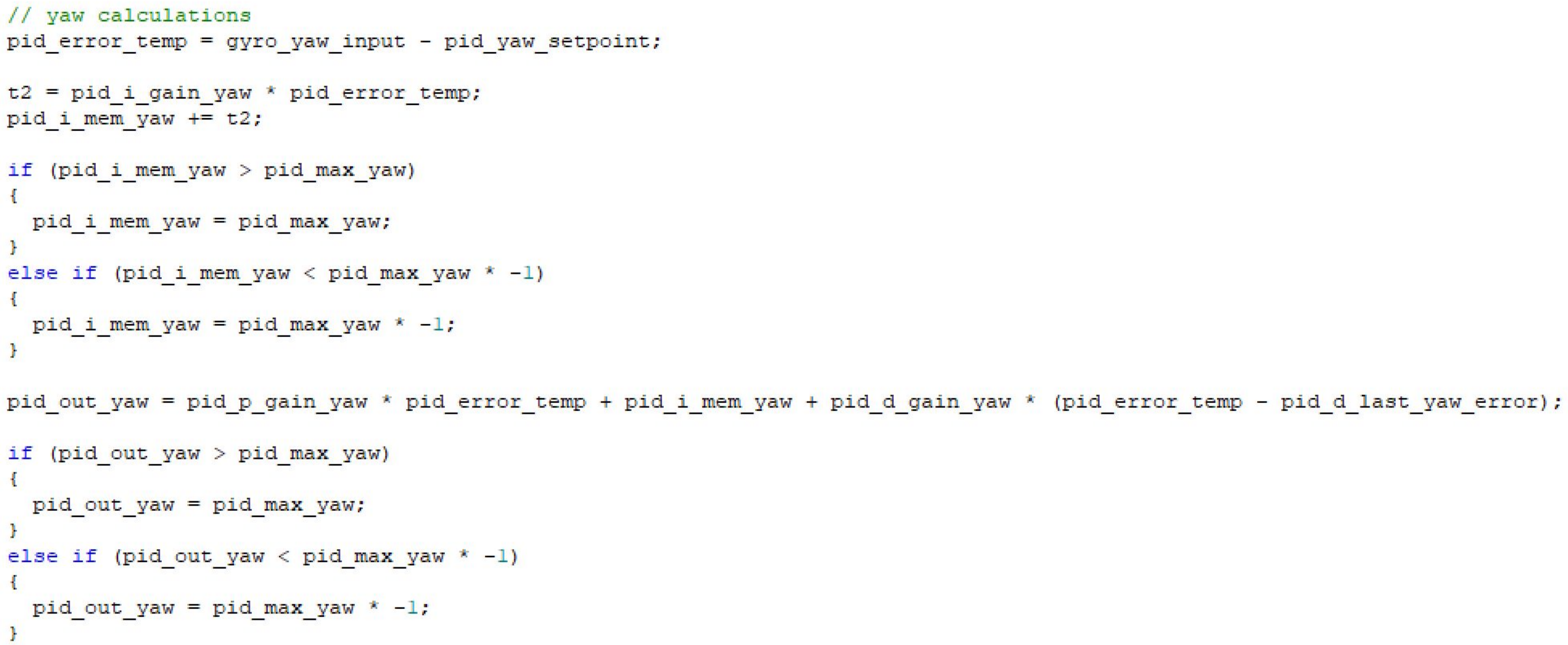
```
// calculating PID
// roll calculations
pid_error_temp = gyro_roll_input - pid_roll_setpoint;

t2 = pid_i_gain_roll * pid_error_temp;
pid_i_mem_roll += t2;

if (pid_i_mem_roll > pid_max_roll)
{
  pid_i_mem_roll = pid_max_roll;
}
else if (pid_i_mem_roll < pid_max_roll * -1)
{
  pid_i_mem_roll = pid_max_roll * -1;
}

pid_out_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll * (pid_error_temp - pid_d_last_roll_error)

if (pid_out_roll > pid_max_roll)
{
  pid_out_roll = pid_max_roll;
}
else if (pid_out_roll < pid_max_roll * -1)
{
  pid_out_roll = pid_max_roll * -1;
}

pid_d_last_roll_error = pid_error_temp;
```

PID Calculations (Pitch)



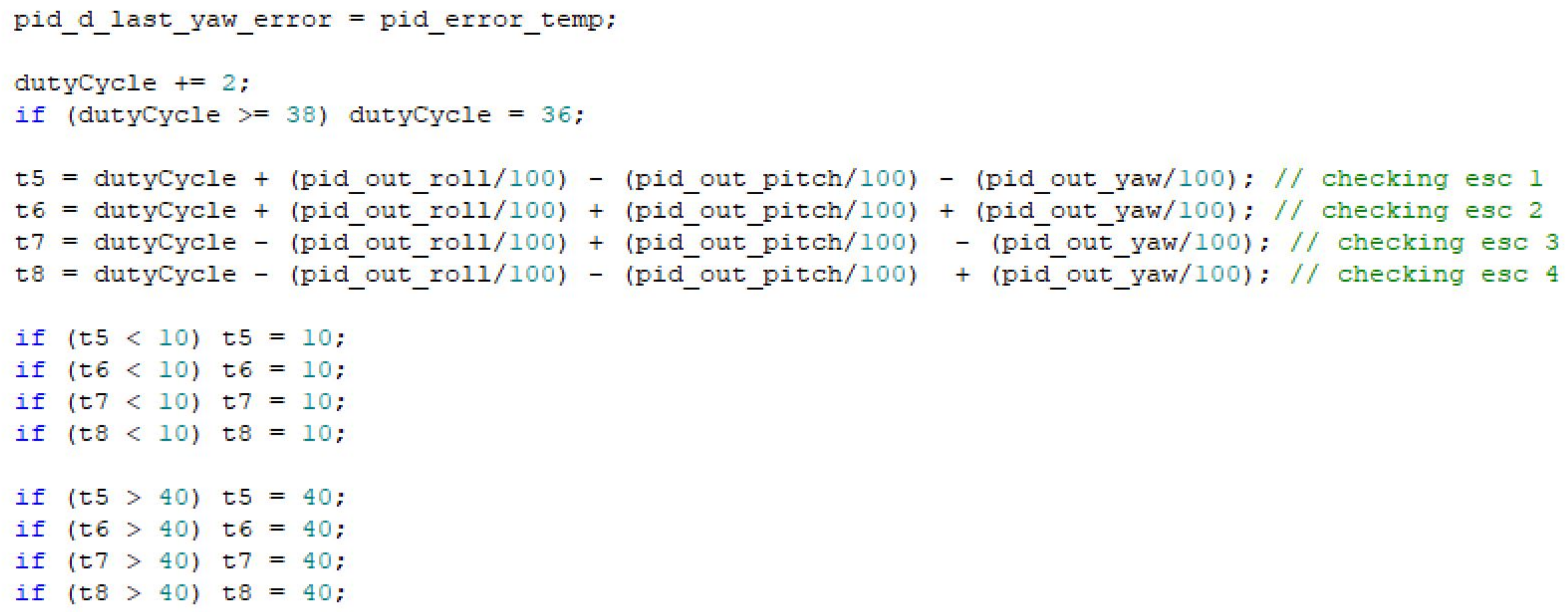
```
// pitch calculations
pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;

t2 = pid_i_gain_pitch * pid_error_temp;
pid_i_mem_pitch += t2;

if (pid_i_mem_pitch > pid_max_pitch)
{
  pid_i_mem_pitch = pid_max_pitch;
}
else if (pid_i_mem_pitch < pid_max_pitch * -1)
{
  pid_i_mem_pitch = pid_max_pitch * -1;
}

pid_out_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch * (pid_error_temp - pid_d_last_pitch_error);

if (pid_out_pitch > pid_max_pitch)
{
  pid_out_pitch = pid_max_pitch;
}
else if (pid_out_pitch < pid_max_pitch * -1)
{
  pid_out_pitch = pid_max_pitch * -1;
}

pid_d_last_pitch_error = pid_error_temp;
```

PID Calculations (Yaw)



```
// yaw calculations
pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;

t2 = pid_i_gain_yaw * pid_error_temp;
pid_i_mem_yaw += t2;

if (pid_i_mem_yaw > pid_max_yaw)
{
  pid_i_mem_yaw = pid_max_yaw;
}
else if (pid_i_mem_yaw < pid_max_yaw * -1)
{
  pid_i_mem_yaw = pid_max_yaw * -1;
}

pid_out_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp - pid_d_last_yaw_error);

if (pid_out_yaw > pid_max_yaw)
{
  pid_out_yaw = pid_max_yaw;
}
else if (pid_out_yaw < pid_max_yaw * -1)
{
  pid_out_yaw = pid_max_yaw * -1;
}
```

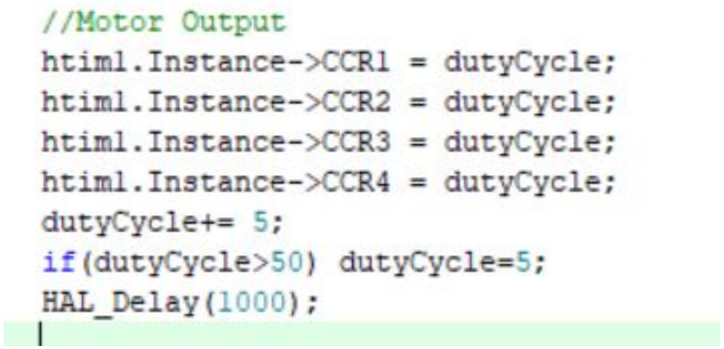
PID Corrected motor outputs



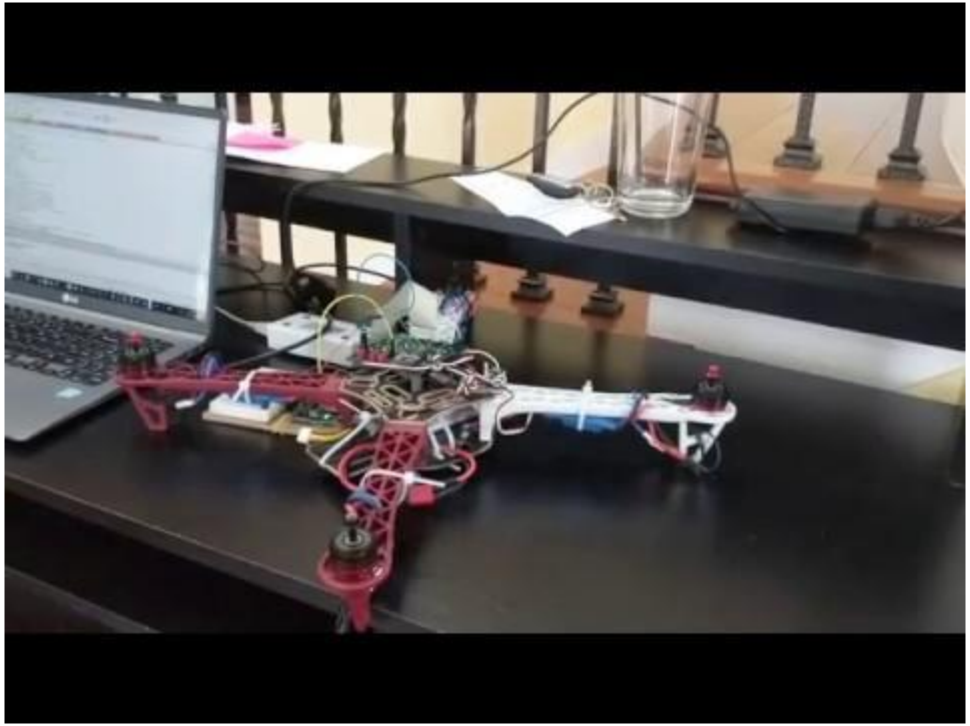
```
pid_d_last_yaw_error = pid_error_temp;

dutyCycle += 2;
if (dutyCycle >= 38) dutyCycle = 36;

t5 = dutyCycle + (pid_out_roll/100) - (pid_out_pitch/100) - (pid_out_yaw/100); // checking esc 1
t6 = dutyCycle + (pid_out_roll/100) + (pid_out_pitch/100) + (pid_out_yaw/100); // checking esc 2
t7 = dutyCycle - (pid_out_roll/100) + (pid_out_pitch/100)  - (pid_out_yaw/100); // checking esc 3
t8 = dutyCycle - (pid_out_roll/100) - (pid_out_pitch/100)  + (pid_out_yaw/100); // checking esc 4

if (t5 < 10) t5 = 10;
if (t6 < 10) t6 = 10;
if (t7 < 10) t7 = 10;
if (t8 < 10) t8 = 10;

if (t5 > 40) t5 = 40;
if (t6 > 40) t6 = 40;
if (t7 > 40) t7 = 40;
if (t8 > 40) t8 = 40;
```

Mock Fly

ST-Link/v2 Required for faster Debugging.



```
//Motor Output
htim1.Instance->CCR1 = dutyCycle;
htim1.Instance->CCR2 = dutyCycle;
htim1.Instance->CCR3 = dutyCycle;
htim1.Instance->CCR4 = dutyCycle;
dutyCycle+= 5;
if(dutyCycle>50) dutyCycle=5;
HAL_Delay(1000);
```



PID correction testing

Testing in the drone cage

Questions ??