Wrocław University of Science and Technology
Faculty of Electronics, Photonics and Microsystems

Intermediate Project

Final Report

Drone Flight Controller

Author:
Patryk Nowik

Date: 8th February 2022

Instructor:
dr inż. Witold Paluszyński

Main Goals:

- **Must do Goals**
  1. Working flight algorithm programmed in Python programming language
  2. Reading Data from *IMU* orientation sensor and altitude from pressure senor
  3. Connecting BLDC motors with ESC drivers and microcontroller
  4. Frame design in *CAD* software and connecting all parts
  5. Performing simple flight tests (During flight or on special testing platform - depending on current time frame)
- **Hope to do Goals**
  1. Adding RC controller for remote control

1 Introduction

The main goal of this project is to create working flight controller and to demonstrate its behaviour during quadcopter flight test. Schema which presents quadcopter in popular "X" configuration can be seen in figure 1.

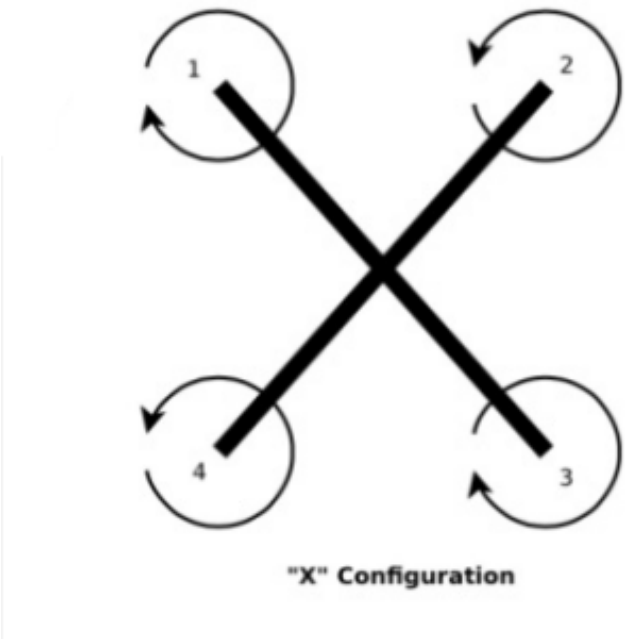

Figure 1: Quadcopter motors and orientation configuration

Project consists of mechanical, hardware and software stages. Mechanical part consists of designing a 3D model of drone frame and assembling all the elements necessary to connect parts and create a drone. During software part, there was coded flight algorithm including control algorithms. Final, hardware part, which purpose was to implement created algorithm on computer and assembling another necessary electronic parts like *IMU* sensor, was allowed to test software and created mechanics.

2 Description of the project stages

In this section there is described every project stage with work what was done in it.

2.1 Software section

First, the *Raspbian* operating system was installed on Raspberry Pi 4B computer, which allowed to properly program algorithm on this device. After setting up operating system early configuration was performed. At this stage was turned on *I2C* communication protocol, which is used to connect to *IMU* sensor.

For reading data from *BNO055* sensor separate module was created, which includes class with all registers declaration and needed methods for reading orientation data. Reading data is possible with using *SMbus* module, which is responsible for handling connection with *I2C* protocole. Below is the fragment of the code which initialize connection with sensor and is setting up correct operation modes.

```
1   def begin(self):
2     #Initialize I2C connection
3     self._bus = smbus.SMBus(1)
4
5     #Get into config mode
6     self._bus.write_byte_data(self._address,BNO055.BNO055_OPR_MODE_ADDR ,0
          x00)
7     time.sleep(0.03)
8
9     #Set up normal power mode
10    self._bus.write_byte_data(self._address, BNO055.BNO055_PWR_MODE_ADDR, 0
          x00)
11    time.sleep(0.03)
12
13    #Set up NDOF operating mode
14    self._bus.write_byte_data(self._address, BNO055.BNO055_OPR_MODE_ADDR, 0
          x0C)
15    time.sleep(0.03)
```

Operation mode of the *BNO055* is the NDOF mode, which according to the documentation [2] calculates absolute orientation of the sensor from the accelerometer, gyroscope and magnetometer data. In the class there is also a method which allow to read gyroscope data. This allowed to create control algorithm for all drone directions excluding height control [4]. Schematic diagram of this algorithm can be seen in figure 2.

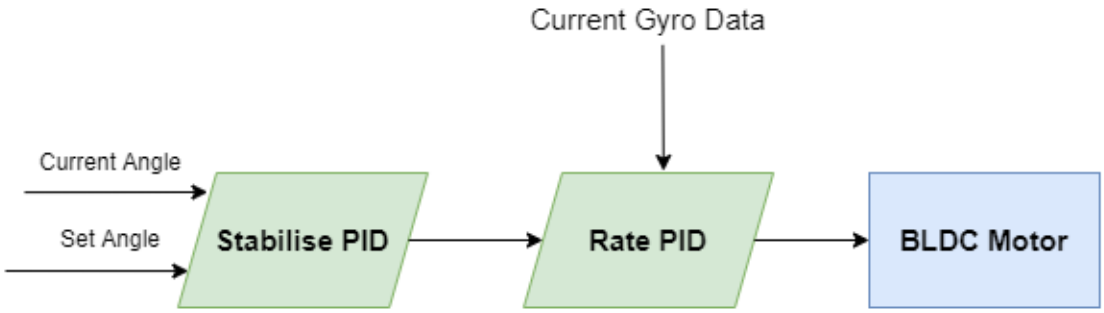

Figure 2: Drone motion control algorithm

This algorithm is using two PID controllers. First is called *stabilise* PID and it is responsible to control angles and return rotation speed. Second is called *rate* PID and uses gyroscope data and previous PID output. Thanks to it, algorithm is able to quickly react to changes in the error.

Second created class was made for reading pressure data from *BMP280* sensor. Below are presented three main functions which are used to calculate data, according to the sensor datasheet [1]. With the pressure data, current altitude can be calculated with accuracy to 1 meter. This allowed to monitor height of the drone, above the sea level and can be used in some future development of the project.

```
1   def read_raw(self):
2     data = self.read_B(self.RAW_DATA, 6)
3     _out = ((data[0] << 16 | data[1] << 8 | data[2])/16, (data[3] << 16 |
          data[4] << 8 | data[5])/16)
4
5     return tuple(_out)
6
7   def calculate_temp(self):
8     adc_t = self.read_raw()[1]
9
10    data = self.read_B(self.DIGITAL_TEMPERATURE, 6)
11    dig_T = struct.unpack('hhh', struct.pack('BBBBBB', data[0], data[1],
          data[2], data[3], data[4], data[5]))
12
13    var1 = ((adc_t) / 16384.0 - (dig_T[0]) / 1024.0) * (dig_T[1])
14    var2 = (((adc_t) / 131072.0 - (dig_T[0]) / 8192.0) * ((adc_t)/131072.0
          - (dig_T[0])/8192.0)) * (dig_T[2])
15    self.t_fine = int(var1 + var2)
16    cTemp = (var1 + var2) / 5120.0
17
18    return cTemp
19
20  def calculate_pressure(self):
21    temp = self.calculate_temp()
22    adc_p = self.read_raw()[0]
23    data = self.read_B(self.DIGITAL_PRESSURE, 18)
24    dig_P = struct.unpack('hhhhhhhhh', struct.pack('BBBBBBBBBBBBBBBBBB',
```

```
        data[0], data[1], data[2], data[3], data[4], data[5], data[6], data
        [7], data[8], data[9], data[10], data[11], data[12], data[13], data
        [14], data[15], data[16], data[17]))
25
26      var1 = (self.t_fine/ 2.0) - 64000.0
27      var2 = var1 * var1 * (dig_P[5]) / 32768.0
28      var2 = var2 + var1 * (dig_P[4]) * 2.0
29      var2 = (var2 / 4.0) + ((dig_P[3]) * 65536.0)
30      var1 = ((dig_P[2]) * var1 * var1 / 524288.0 + (dig_P[1]) * var1) /
            524288.0
31      var1 = (1.0 + var1 / 32768.0) * (dig_P[0])
32      if (var1 == 0.0): return 0 #avoid division by zero
33      p = 1048576.0 - adc_p
34      p = (p - (var2 / 4096.0)) * 6250.0 / var1
35      var1 = (dig_P[8]) * p * p / 2147483648.0
36      var2 = p * (dig_P[7]) / 32768.0;
37
38      return (p + (var1 + var2 + (dig_P[6]))) / 16.0) / 100 # pressure in hPa
```

Next module was created to implement algorithm, which is responsible for calculating thrust value for each motor, for certain directions. Equations for calculating motor mixing algorithm is presented below.

- **Motor 1 (CW - front left)** = Thrust - Yaw + Pitch - Roll
- **Motor 2 (CCW - front right)** = Thrust + Yaw + Pitch + Roll
- **Motor 3 (CCW - back left)** = Thrust + Yaw - Pitch - Roll
- **Motor 4 (CW - back right)** = Thrust - Yaw - Pitch + Roll

where,
CW - move clockwise
CCW - counter clockwise

Method created in that module, takes the orientation and gyroscope readings given by *BNO055* class and calculates thrusts for each of the motors, following equation from the motor mixing algorithm. After that certain function passes thrust values to *PWM* Pin in RPI. Such operation is presented in the figure 3.

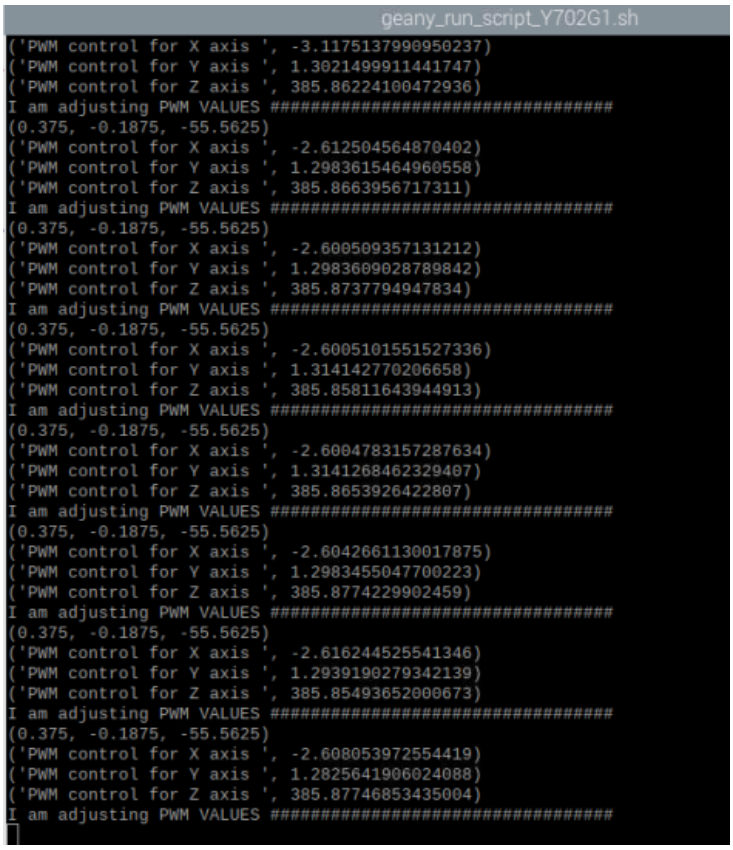

Figure 3: PWM calculating sequence for motor mixing algorithm

2.2 Mechanical section

To properly mount all the parts needed for drone motion, special frame was designed using *Inventor 2021* software provided by AUTODESK company. Designed frame was sliced using *Cura* program. This allowed to print this frame using 3D printer ENDER 3 with *PLA* material. Model of the frame is presented in the figure 4.

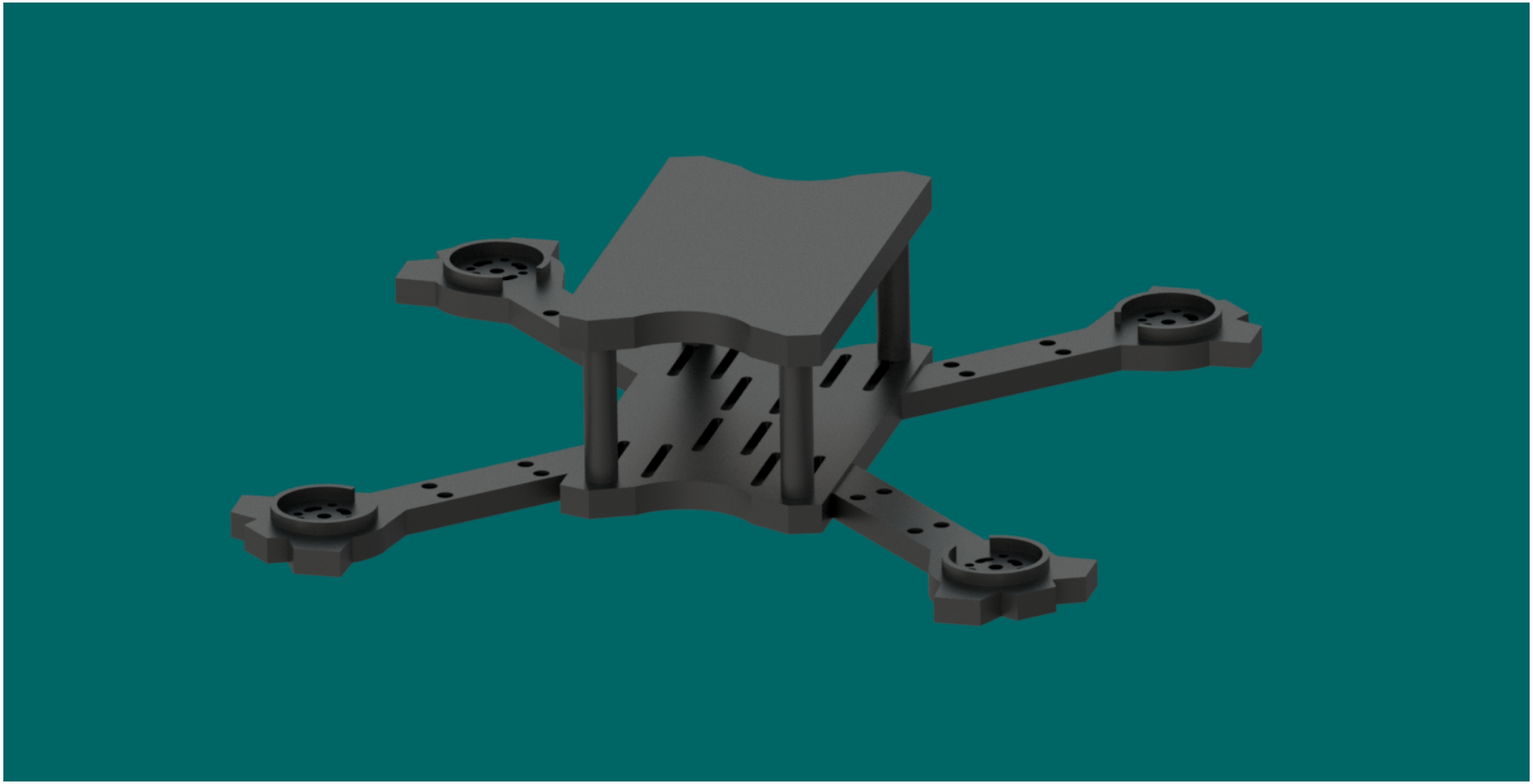

Figure 4: Project of drone frame

2.3 Hardware section

Main hardware components which was used to create a quadcopter are listed below.

- BLDC motors - *EMAX ECO II 2306 2400kV*
- ESC drivers for motors - *LittleBee - Spring 20A*
- Computer for algorithm calculations and hardware tasks - *Raspberry Pi 4B - 4GB version*
- Li-Po battery - 3S 3600mAh
- Orientation Sensor - *BNO055 Bosch Aboslute Orientation Sensor*
- Pressure sensor - *BMP280 Pressure Sensor*

Connection to Raspberry Pi computer was done by "headless mode". This solution allow to program computer remotely through ssh connection.
Both BNO055 and BMP280 sensors are mounted on single PCB board made by *DFrobot* [3] and allow to transfer data by *I2C* communication protocol which was enabled on Raspberry Pi. Description of the connection of DFrobot device to computer is in the table below.

DFrobot Pin	**Raspberry Pi 4B Pin**
Power	1 (3V3)
GND	6 (GND)
Data Line	3 (I2C1 SDA)
Clock Line	5 (I2C1 SCL)

ESC Drivers was mounted to RPI proper signal and ground pins. Also they were powered from Li-Po battery directly. Description of this connection can be seen below.

ESC Pin	**Raspberry Pi 4B Pin**
ESC1 GND	34 (GND)
ESC1 PWM	33 (PWM1)
ESC2 GND	30 (GND)
ESC2 PWM	35 (PWM1)
ESC3 GND	20 (GND)
ESC3 PWM	32 (PWM0)
ESC4 GND	14 (GND)
ESC4 PWM	12 (PWM0)

BLDC motors was connected to ESC output pins. Simple connection ensure rotation with clockwise direction. By replacing order of the random two wires, was obtained counterclockwise direction. Motors first and third have clockwise direction of rotation, while motors second and fourth counterclockwise. Such connection prevents quadcopter from twirling around its own z axis and meant to ensure stable position.
Power supply was ensured by soldering special breadboard, connected to Li-Po battery. Raspberry Pi was powered up by connecting voltage stabilizer and some capacitors to clear the signal.

3 Final Assembly

Finally parts from previous described stages was mounted together. In that way obtained quadcopter was ready to test and first flight. Picture taken of drone is presented in the figure 5.

Figure 5: Picture of mounted quadcopter

After powering up, it was impossible to connect to Raspberry Pi in headless mode, probably due to some disturbances in signal coming out from linear stabilizer. It was decided to power up computer with external power source After that calibration of ESC drivers was performed. Flight was very unstable due to uncalibrated PID controllers and still some ESC signal issues. Due to lack of RC remote controller it was difficult to adjust thrust which is needed for quadcopter to lift its own weight in air only from system console. To correctly test algorithm and tune PID controllers remote controller would be needed, which may be added in some future development of this project.

4 Summary

Quadcopter was mounted successfully. Whole project allowed to learn the principle which are standing behind the drone physics. Unfortunately not everything is working as it was expected. Due to lack of the remote controller, adjusting thrust was difficult and was not finished, therefore more advanced flight test could not also be performed. Unstable power supply to Raspberry Pi computer did not allow to connecting it from Li-Po battery. However, the whole drone is able to fly. In the future development of this project it may be possible to properly test the algorithm and adjust thrust of the quadcopter.

References

[1] Bosch Sensortec GmbH. *BMP280 Digital Pressure Sensor*. 2014.

[2] Bosch Sensortec GmbH. *BNO055 Intelligent 9-axis absolute orientation sensor*. 2014.

[3] DFrobot. Gravity 10dof ahrs user manual. `https://wiki.dfrobot.com/Gravity_BNO055_+_BMP280%20intelligent_10DOF_AHRS_SKU_SEN0253`.

[4] D. Such. Writing your own flight controller. `https://reefwing.medium.com/how-to-write-your-own-flight-controller-software-part-1-ac08b6ecc01e`, 2020.