CS 4140 | Embedded Systems Laboratory

25th of June 2019

TU Delft
Coordinated by Prof. Koen Langendoen

Final Report

Lennart de Jong	4440501	EWI	ES
Kaustubh Agarwal	4823168	EWI	ES
Koen Goedemondt	4291077	EWI	ES
Ibrahim Chahine	4737792	3ME	SC

Abstract

Drones, in the present day, are subjects of extensive application and research in diverse fields ranging from commercial, defense to entertainment. Through this project, a stabilization system for a small unmanned quad-rotor is conceived, designed and implemented. This work includes both an implementation of control on filtered raw sensor data and processed DMP measurements. Yaw Rate control is done with a P-controller and roll/pitch control is done with a PD controller. Aside from the trigonometric solution, done on the DMP, the raw sensor data are also filtered and used for control. A Butterworth filter is used on yaw rate measurements and Kalman filter is used on roll and pitch measurements. In addition, a Graphical User Interface is created for displaying the drone status in real time.

1 Introduction

This report presents the design and implementation of a quad-rotor stabilization system. For realizing stability, the drone uses an on-board nRF51822 SoC combined with a GY-86 sensor module. The sensor module contains a three-axis gyroscope, three-axis accelerometer and a barometer, which are all used for drone attitude control. The SoC also contains an analog-to-digital converter for reading battery voltage which is used to ensure flying safety. The drone is controlled by the help of a joystick connected to a PC, which is used as a base station and provides bidirectional UART communication link. The report is divided in sections where Section 2 discusses the system architecture and the overall design. Section 3 describes the implementation of the bidirectional communication protocol between the PC and quad-rotor system. It also discusses the control loop and filters used for stabilization and safety features. Following this, the GUI for displaying real time information about the overall system is discussed. Section 4 shows the experimental results of profiling and filtering. Finally, the conclusion reflects upon the final design and how well it performed during testing and the demonstration.

2 Architecture

A good design is important for system stability. A system architecture diagram provides the main model of the entire system and provides a basis for implementing all the different system functionality. Our system system architecture is shown in figure 1.

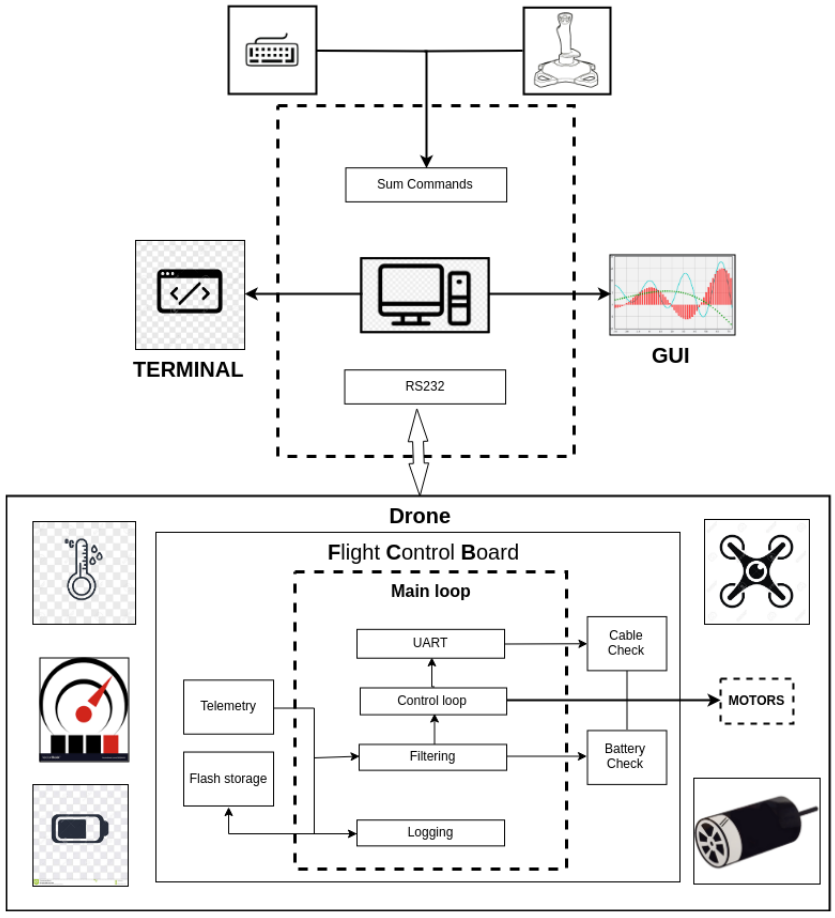

Figure 1: System architecture

3 Implementation

This section describes the implementation of the bidirectional communication between the quad-rotor system and the PC. It also describes our design choices while implementing the different control modes. Finally it explains the choice of filters for the RAW mode, system logging facility and a real-time Graphical user interface for visualization.

3.1 Communication

In order to achieve a stable flight, the quad-copter and the PC(base station) need to be synchronized. Even a small delay will cause the system to go out of synchronization resulting in a fatal crash. In order to achieve this we decided to design a dynamic serial communication protocol to keep the latency low. One way to do that is to divide communication into multiple states and allow other loop computations to keep going while the packet received/sent cycle is not complete. A queue was created, on both the PC and drone side, which would store every loop. Thus, by determining the different packet structures as shown in table 1 and table 2 below, 3 states were established for receiving a packet:

- We first receive a header (0xFF) and set the header flag to true. If not we de-queue and continue.
- We then check for a valid type (0x01,0x02 and 0x03), which sets the type flag to true and determines the expected packet length. If the byte received is of invalid type, we set the header flag to false and start the process again.
- If the queue receives a packet of a determined message length and both header and type flag are true (both Header and type bytes were received), de-queue the whole packet and check for CRC. Set the header and type flags to false to start again and store the information if the CRC confirms the package is correct. If the queue size is not equal to the required message length, continue receiving the rest of the bytes.

The following flowchart (2) summarizes the receiving part of communication.

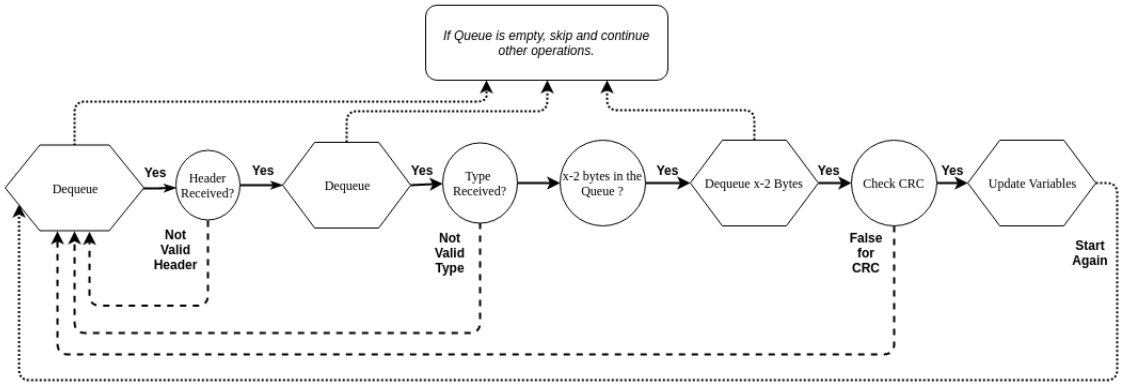

Figure 2: Flowchart that describes the different stages of receiving a message

The sending function takes the packet type (0x01,0x02,0x03,...) as an input , and sends complete packets at once. Packets are sent in a periodic fashion.

The base station(PC) sends a total of 3 packets with variable lengths and frequencies to the FCB as Shown in Table 1. The packet containing the setpoints is sent at 50 Hz while the other two packets are sent at 20 Hz. The packets are designed to be as small as possible and also include a CRC byte to check for bit flips and other transmission errors. The header was chosen to be 0xFF

and the type field varies according to different packets. The Toggle packet represents if the FCB is in DMP or in RAW mode.

Table 1: Packets sent from PC to FCB

Header	Type	Mode	Toggle	CRC
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

Header	Type	Mode	P	P1	P2	CRC
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

Header	Type	Mode	Yaw_set	Pitch_set	Roll_set	Lift_set	CRC
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

The Drone sends four packets of variable lengths to the PC as shown in Table 2(Logging packet not shown). Packet 3 contains the important sensor and battery values while packet 2 contains the current motor values, setpoints and the Controller values. All packets include CRC bytes for integrity checks and are as small as possible to keep the latency low.

Table 2: Packets sent from FCB to PC

Header	Type	Mode	Toggle	CRC
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

Header	Type	Motor Values	Controllers	Setpoints	CRC
1 Byte	1 Byte	8 Bytes	4 Bytes	4 Bytes	1 Byte

Header	Type	Phi	Theta	sp	sq	sr	Battery	CRC
1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte

3.2 Setting Motor Values

The roll, pitch, yaw and lift values received from the joystick are all sent with 8-bits of resolution. Originally, the joystick sends these values as 16-bit integers with a very high resolution, but are received as 8-bit unsigned integers. The 16-bit integers range from $[-32{,}768, 32{,}767]$, with 0 being the normal position of the joystick and 32,767 and $-32{,}768$ being maximum roll, pitch, yaw or lift in opposite directions respectively. The number is converted to $[0, 254]$, with 127 being the normal position of the joystick. These conversions happens on the PC side to avoid having to convert these numbers on the drone, as it would unnecessarily use computation time and slow down the main loop. On the drone, the 8-bit number is converted to a 16-bit unsigned and multiplied by 2 to increase sensitivity of the joystick. After which first the lift is added to the ae[ ] array, then pitch, yaw and roll. When subtracting motor values from the ae[ ] array, the current value is checked to be higher than than value it is subtracted from in order to avoid overflow. Finally, the motor values are checked to not be lower than 200, the minimum value at which they start spinning. This was done to avoid motors coming to standstill when steering heavily.

3.3 Safety Features

In order to operate the drone safely, two safety features were implemented. Namely, a battery voltage check to prevent battery damage and a cable disconnection check to prevent the drone becoming uncontrollable. Both of these safety features send the drone into panic mode after activation. According to the specifications of the batteries, the voltage is not allowed to drop below 10.5 V. If this happens, the battery could get damaged, and the voltage would drop quickly

causing the drone to crash. A reading of the battery voltage is provided by the ADC on the FCB, therefore this value was used to send to drone in panic mode if the battery voltage drops too low. When the motors rev up, the voltage of the batteries can drop below the safety threshold point to deliver the necessary current. To prevent this event from triggering panic mode, a recursive moving average algorithm was implemented which keeps the battery voltage reading stable. Additionally, the drone requires the serial connection to the PC to be always active. If this is not the case, the cable is assumed to be disconnected and the drone becomes uncontrollable. To detect whether the serial connection is still active, a timeout function was created which gets refreshed every time a packet is successfully received. If a packet is not received within a predefined amount of time - in this case 200 ms - the timeout function is triggered and the drone goes to Panic mode averting an uncontrollable drone.

3.4 Main Loop

The main loop which runs on the CPU of the drone is responsible for executing all tasks sequentially and within an acceptable time frame. The main loop also contains the state machine of different operating modes of the drone, most importantly panic, safe and full control mode. Panic mode can be reached from every state. The state machine is implemented as `switch...case` statement. A diagram of all activities performed by the main loop can be found below:

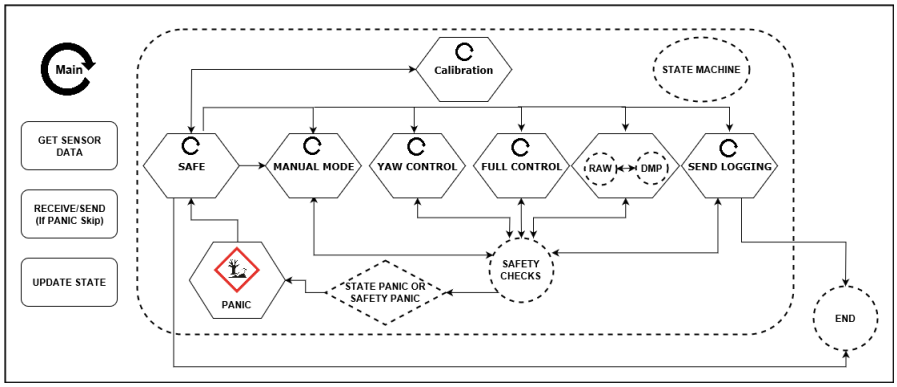

Figure 3: State Machine Diagram

3.5 Calibration mode

The sensor measurements provided by the MPU6050 are accurate in DMP, however they do have a bias that has to be corrected. The gyroscopes even have an active drift that requires active bias estimation to completely prevent this drift. In calibration mode, the first 100 measurements are ignored as initial inaccuracies in the sensor measurements were found if the mode is entered too quickly after switching between DMP and raw mode. The next 100 measurements of the accelerometer axes, gyroscope axes and pitch and roll angles are summed in their respective variables. After the 100 measurements they are normalized and cast to the calibration *global* variables, for instance 'sp0' for the roll rate. After completion of this step, the system returns to safe mode.

3.6 Control Loop

The system has two modes that require a control loop to operate: mode 4- Yaw mode and mode 5- Full control mode. The design of both loops is described in the following two subsections.

3.6.1 Yaw mode

In this mode only the yaw rate of the drone has to be controlled, unlike the roll and pitch, the yaw angle cannot be determined in the same manner since gravity does not act differently on the accelerometer if the yaw angle changes. By twisting the joystick it provides a setpoint for the yaw rate that the drone must follow, this setpoint is first shifted and scaled to the measurement range and then limited to allow fly-ability for the pilot. The DMP provides the current yaw rate, 'sr', of the drone and is subtracted from the setpoint creating the error between the desired and actual yaw rates as shown in Figure 4. This error is then multiplied by a tunable constant P and scaled back to motor values using bit-shift operations as in the case of a P controller. An overflow check is done to prevent incorrect values to be sent to the drone. Depending on the sign, the setpoint to motors 1 and 3 is increased/decreased and to motors 2 and 4 is decreased/increased to maintain a stable flight. We use a second order Butterworth filter in RAW mode for processing the raw sensor values.

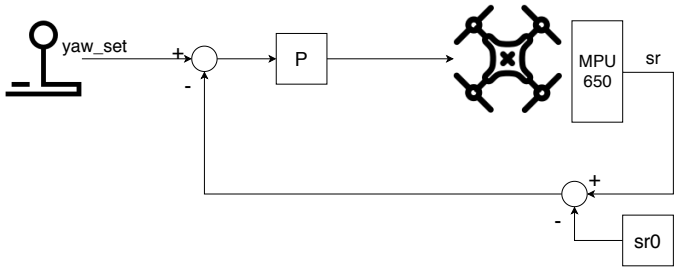

Figure 4: Proportional yaw rate control loop

3.6.2 Full Control mode

In full control mode both the angle and angular velocity of the pitch and roll have to be controlled around a setpoint from the joystick. This creates a problem since if we use two P controllers in series it will create a 180°phase lag. We overcame this problem by implementing a cascaded P controller with decoupled proportional gains as shown in Figure 5. This structure allows for independent tuning of P1 and P2 and is stable for $P2 > P1 > 1$ due to the rate controller reducing the phase shift. Tuning of P1 and P2 is done using the keyboard. When the MPU6050 is in DMP mode it provides both the angle and rates directly and only requires calibration, and as is directly used in the control loop. When it it set to RAW mode, a Kalman filter is used to obtain the angle and calibrate the rate measurement. The setpoints for both pitch and roll were translated to 30°at maximum extension for fly-ability purposes.

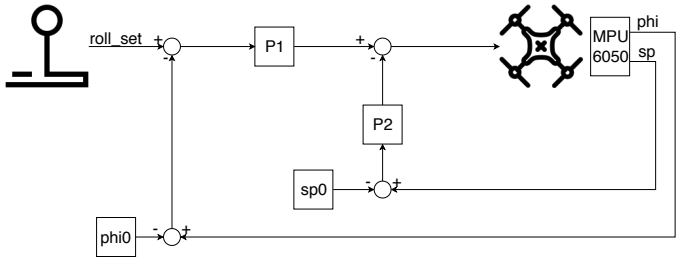

Figure 5: Roll control loop with DMP.

3.7 Filtering

In the DMP mode, the MPU-6050 module runs its own digital motion processing algorithm to correctly estimate the attitude of the drone. Without the DMP, the raw sensor values are susceptible to all kinds of noise. In the RAW mode we are supposed to design our own filters running at at least 500Hz to correctly estimate the angles and rates for all degrees of freedom of the drone. A low pass second order Butterworth filter was designed for estimating the yaw rate and a Kalman filter was designed to estimate the roll and pitch angles and actively estimate the gyroscope's bias.

3.7.1 Butterworth Filter

The unfiltered gyroscope measurements contains high frequency noise, this noise can be removed by the help of a low-pass filter which allows signal below its cut-off frequency to pass through and removes the higher frequencies. For the yaw rate we utilize a second order low-pass Butterworth filter with a cutoff frequency of 10Hz and a sampling frequency of 500Hz. The filter constants a0, a1, a2, b0 and b1 were calculated in Matlab using the Butter command. As the drone does not have a floating point arithmetic unit we resorted to fixed point arithmetic to implement the filter. We use 14 bit accuracy of the decimal number to allow for enough resolution for the filter to function as described in [1]. The implementation is listed below.

```
1 static int32_t a0 = 59;          static int32_t a1 = 119;
2 static int32_t b1 = -29863;      static int32_t b2 = 13716;
3
4     yaw_rate = sr - sr0;
5     int32_t x0_y = (int32_t) yaw_rate;
6     int32_t by0_y = (mul(a0, x0_y) + mul(a1, x1_y) + mul(a0, x2_y) - mul(b1, by1_y)
          - mul(b2, by2_y));
7     yaw_rate = (int16_t) by0_y;
8
9 x2_y = x1_y;     x1_y = x0_y;     by2_y = by1_y;   by1_y = by0_y;
```

3.7.2 Kalman Filter

In raw mode, pitch and roll have two problems that need to be addressed. Firstly the gyroscope measurements have an active drift and secondly we do not have a direct measurement of the angle. Both of these problems can be solved with the use of a Kalman filter. The Kalman filter dynamically estimates the bias on the gyroscope as well as provide the pitch and roll angle measurement by fusing the gyroscope and accelerometer values. The angle is obtained by integrating the gyroscope measurement using the loop time measured at not exactly 500Hz but 498Hz. The accelerometer at small angles approximates the actual angle and is then used to measure the prediction error. It also does not posses drift like the gyroscope and is therefore also used to correct the rate estimation. We used the same fixed point notation as Butterworth filter for the Kalman filter. We designed the it to run at 500 Hz and adjusted the bias term by tuning the constants C1 and C2.

```
static int32_t P2PHI = 366;    static int32_t C1 = 5;    static int32_t C2 = 25;

    // Roll Angle and Rate Kalman
    roll_rate = sp;
    int32_t say_cali = say - say0;
    int32_t p_est = (int32_t)(roll_rate << 14) - p_bias;
    kphi = kphi + mul(p_est, P2PHI);
    int32_t p_e = kphi - (say_cali << 14);
    kphi = kphi - (p_e>>C1);
    p_bias = p_bias + (divide(p_e, P2PHI)>>C2);

roll_rate = (int16_t)(p_est >> 14);      roll_angle = (int16_t)(kphi >> 14);
```

3.8 Logging

In addition to sending telemetry data to the PC, we also write the telemetry data on the on-board flash on the drone by calling the *write_log()* function. We use a packet of variable size as shown in Table 3 whose size can be altered based on the information to be logged in the system. The data is sent to the PC after pressing Key-9 on the keyboard to go to the logging state during safe mode. All logging packets are only sent after completion of the flight so that it doesn't introduce unnecessary latency during the flight. In the event of flash memory exhaustion, the flash memory is cleared and new data is written in it's place. The data is saved to a csv file which is then used for data visualization and debugging.

Table 3: Logging Packet

Header	Type	TimeStamp	Data	CRC
1 Byte	1 Byte	4 Bytes	Variable length	1 Byte

3.9 Graphical User Interface

To display the data in real time we write the entries on the terminal to a file. Creating and writing to the file is performed in the PC terminal. It was chosen to develop a web-browser base GUI, because of the ease of development and amount of available JavaScript libraries for plotting data and gauge graphics. For the back end, the python web framework flask is used. The built in development server is used to create a browser based front end for the GUI. The data is provided to the browser window by an endpoint which reads the file that is created by the PC terminal program. As the file is read, the file is deleted and the PC terminal program creates a new file with updated data. The endpoint is polled using GET requests in JavaScript and saved into a buffer. The data in the buffer is used by JavaScript running in the browser to update the gauges and textual data at a frequency of 10 Hz and the graphs at 5 Hz. The libraries used to create the GUI are: `chart.js` [2], `gauge.js` [3] and `Flask` [4].

4 Experimental Result

4.1 Profiling

We measured the loop time similar to the tik toc function in Matlab by calling the *get_time_us()* function at the start and end of the main loop. It was discovered that most of the loop time is taken up by receiving the sensor measurements from the MPU6050 in both DMP and RAW mode. In RAW mode we measured an average loop time of around 0.8ms with spikes of at most 1.3ms.

Table 4: Measured Loop Time in ms

Mode	Avg Loop Time (ms)- DMP Mode	Avg Loop time (ms)- RAW Mode
Manual	3.8428	0.7911
Yaw Control	3.9026	0.8153
Full Control	3.9771	0.9067

Table 5: Measured Duration of Modules in ms

Functionality	Loop Time (ms)- DMP Mode	Loop time (ms)- RAW Mode
Receive Packet	0.072-0.132	0.072-0.132
Send Packet	0.192-0.282	0.192-0.282
Read Data	3.21-3.54	0.62-0.65
Full Control	0.048	0.048
Kalman Filter	0.055	0.055
Butterworth Filter	0.033	0.033

4.2 Filtering

The filters were implemented in Matlab and used data sets acquired from the drone through logging. This allowed for easier tuning of the filter values. We then ran the filters on the board and logged the raw measurements and the filter ones as shown in Figures 6,7 and 8. The Butterworth filter works as intended and removes the high frequency noise at the cost of phase delay. The Roll rate measurement and estimation only vary slightly at the end due to bias compensation and is functioning properly. The Roll angle measurement also functions but it a bit noisy still. Sadly we did not have the time to test with an added Butterworth filter to the accelerometer measurements. We also could not test the filters with the drone before the demonstration and during the demo it was discovered that in raw mode the value sent to the motors should be higher to allow proper tuning in raw mode.

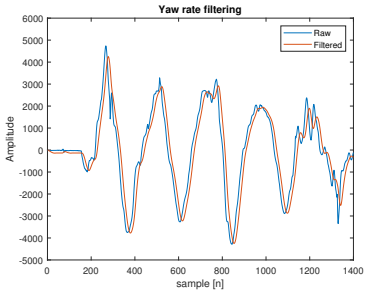

Figure 6: Raw sensor data before and after the Butterworth filter.

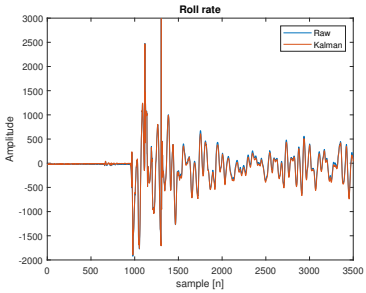

Figure 7: Raw measured rate and estimated rate.

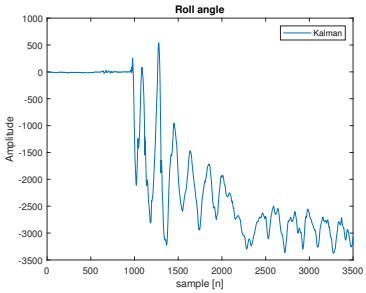

Figure 8: Roll angle output of the Kalman filter.

5 Conclusion & Evaluation

This report describes the design and implementation of a software module to pilot a tethered drone using a joystick. Throughout the design phases of this project, compromises in the scalability and given requirements appeared as design challenges. A dynamic and robust communication link between the PC and drone was implemented and tested with a custom packet protocol and CRC Checks. The lengths of the packet were chosen to reduce latency as much as possible. The packets were sent to the drone periodically at 20Hz and 50Hz depending on the type, the drone sent telemetry packets periodically at 20Hz to the PC. Safety checks in case of sudden cable disconnection or battery depletion were successfully implemented. These come with great importance for the safety of both the drone and the user. The yaw rate of the drone is controlled using a tunable P controller. The pitch and roll angles and rates were correctly controlled using two cascaded decoupled P controllers (PD) also with tunable gains. To implement filters, a second order Butterworth filter was implemented and verified for raw mode at 500Hz. The Kalman filter was implemented at 500Hz without the addition of a Butterworth filter on the accelerometer. A browser based GUI was implemented using JavaScript with a Python backend and performed well at 10 Hz refresh rate. The final size of the binary running on the FCB was 49 kB.

However, while creating the final product some possible points of improvement emerged. Planning, preparing specific testing scenarios before the lab sessions was not always up to par. This was in part due to task division being a too vague at times. Creating an adequate division beforehand could have induced greater progress. Concerning the implementation, the design still has room for improvement : better mapping between control and setting the motor values to allow for more accurate tuning. Moreover, adding aperiodic communication for the gain tuning and mode switching to communication reduce overhead, since in the current implementation the same data is sent many times over.

6 Team member contributions

Koen - Supported implementation of communication protocol and logging, main function and state machine, functions for setting motor values, safety checks, GUI.

Lennart - Designed and implemented control functions and filters. Assisted in communication protocol implementation and PC terminal functionalities.

Ibrahim - Designed and implemented the communication protocol, PC terminal code and safety checks. Assisted in control system design.

Kaustubh - Developed the on-board logging feature and worked on implementation of filters for raw mode. Supported in the implementation of the communication protocol and integrity checking.

References

[1] Sait Izmit. “Floating-Point Support for Embedded FPGA Platform with 6502 Soft-Processor”. In: (June 2019).

[2] Github contributors. *chart.js*. URL: `https://www.chartjs.org/` (visited on 06/2019).

[3] Bernard Kobos. *gauge.js*. URL: `https://bernii.github.io/gauge.js/` (visited on 06/2019).

[4] Armin Ronacher. *Flask (A Python Microframework)*. URL: `http://flask.pocoo.org/` (visited on 06/2019).