Embedded Systems Laboratory

Stefanos Koffas *4915747*
Beni Kovács *4770552*
Pranjal Singh Rajput *4788087*
Miguel Pérez Ávila *4937724*

May 2019

1 Abstract

In this report, we present the implementation of the quadcopter stabilization problem for TU Delft course CS4140, done by Group 14. Besides implementing the basic functionality, we have developed an interactive GUI for analyzing and visualizing the signals at runtime. All safety measures are taken care of, and different modes of operations including the full control, and raw mode are implemented. Kalman and Butterworth filters are designed and used for operation in raw mode. The drone can be operated in both tethered and wireless mode the following with an Android phone.

2 Introduction

In this report, we explain the system architecture and its software components. Then, we continue with a short summary of the development methods and the explanation of the implementation of the software. Finally, we present our experimental results and conclude our work.

3 Architecture

In this section, we discuss the hardware and the software architecture and various components used in the system.

3.1 Hardware architecture and interfaces

The system consists of several hardware components that include ARM Cortex M0 based SoC, the nRF51822 microcontroller, that acts as an interface between the sensor modules and the 4 motors. The sensor module is GY-86, that consists of 2 parts, an MPU-6050 that has a 3-axis accelerometer, and a 3-axis gyroscope, & an MS5611 barometer. The MPU-6050 operates in two different modes, DMP, and RAW mode. In DMP mode, the frequency of operation is fixed at 100Hz and the raw sensor readings are fused and converted into the Euler angles on the microcontroller, thereby giving less noisy data. Whereas, in RAW mode, the frequency of operation is user selectable and it ranges from 32Hz to 8kHz. We set this frequency at 1kHz. The sensor readings are directly pushed into the FIFO without applying any data fusion algorithm.

A 4000mAh LiPo battery is used to supply the voltage to the FCB and for driving the 4 motors. The battery level must be checked regularly and should be used above 10.5V. The microcontroller sends the RPM setpoints to an F330, electronic speed controller, which thereby uses these values to drive the motors. The MCU communicates with the speed controller using PWM signals. A Bluetooth Low Energy (BLE) module is used for wireless communication between the drone and the android app.

On the PC side, both joystick and keyboard can be used to control the drone movement, while only the keyboard is used for setting the control parameters such as proportional constants, changing the mode of communication, start data logging, etc. We created a GUI to visualize and analyze live data from the drone on the screen.

In wireless mode, an Android phone with Android *version 7.1.1* is used for operating the drone in different modes. For this, the smartphone's inbuilt accelerometer and the magnetometer sensor data is read and are fused together to calculate roll, pitch and yaw control values using an Android API. The calculated data is then sent to the drone through a Bluetooth Low Energy communication channel.

3.2 Software architecture

3.2.1 MCU architecture

The MCU software architecture consists of 4 different tasks as we present in Figure 2. First, we have an initialization part that is in charge of setting up the system, assigning the initial value to all the peripheral, the global variables and the state of the drone mode. After that, the drone enters in the execution of the main loop, where it continues a fixed execution of the tasks. First, we check the battery level and whether we have received a message from the PC before the timeout pass, in order to continue the normal execution or enter in the panic mode (if it is not in safe mode). Then we execute the reading sensor task and then the control loop task. Although, this reading of new measurements is done after we check the MPU6000 chip is ready and has a new data available. If this does not happen we do not execute the control loop, because it would compute the same values again. The Fourth task is in charge of checking the arrival of new messages and process them. This task realizes a different process depending on the message arrived. The messages are explained in Table 1.

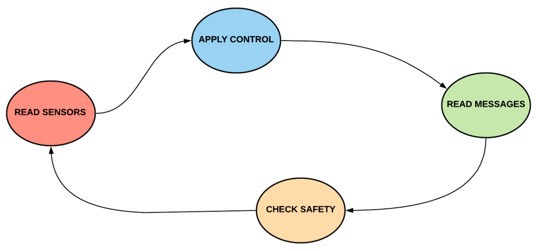


Figure 2: Flow execution of the MCU software architecture.

3.2.2 Qt user interface

The UI application consists of 3 threads as presented in figure 3. First, a background thread handles the connection to the joystick and sends the events to the main thread. The main thread consists of 2 forms. The main form handles key input and provides visual feedback for the user. A second form shows live charts that can be used for debugging. There is a similar additional form that can be opened after dumping the log data from the microcontroller, for quick visualization.

4 Development methods

During the development, our primary goal was to maximally exploit the available laboratory time to fine-tune the controllers in the software. Where it was possible (for example for the protocol and the filters), parts of the program were developed and comprehensively tested independently on PC. Additionally, before the labs, all source codes were tested on the flight controller board. To overcome the lack of rotors, we used live charts in order to make the tuning of the controllers possible with preliminary values off-flight.

Additionally, besides this approach adds a lot to the safety of testing, because no code runs first on the drone (although faults may still be possible by mal-tuning of the controllers).

5 Implementation

This section gives in-depth explanation about the implementation of the software components.

5.1 Communication

We use a general, platform-independent communication protocol that works on both the PC and microcontroller. The protocol itself was developed in C. Testing was carried out by connecting to instances of PC programs to each other, in order to boost the speed of the development, keeping in mind to use only microcontroller compatible code. Later, the code was adapted to Android, however, it was translated there to Java to solve the mobile development without having to use the native SDK.

5.1.1 Protocol Implementation

As it is clearly shown in table 1, a protocol with variable length messages is used. Apart from the fields that are shown in table 1, two more bytes are used for every packet that is transmitted. Therefore, the size of every packet that is exchanged between the two sides varies from 5 to 43 bytes.

The first byte of every message contains the character '[' which denotes its beginning. The next field after the start byte contains its ID. In particular, different letters are used to characterize the messages that are exchanged between the microcontroller and the PC side. The last field of every packet contains its checksum. Thus, for every packet that is received

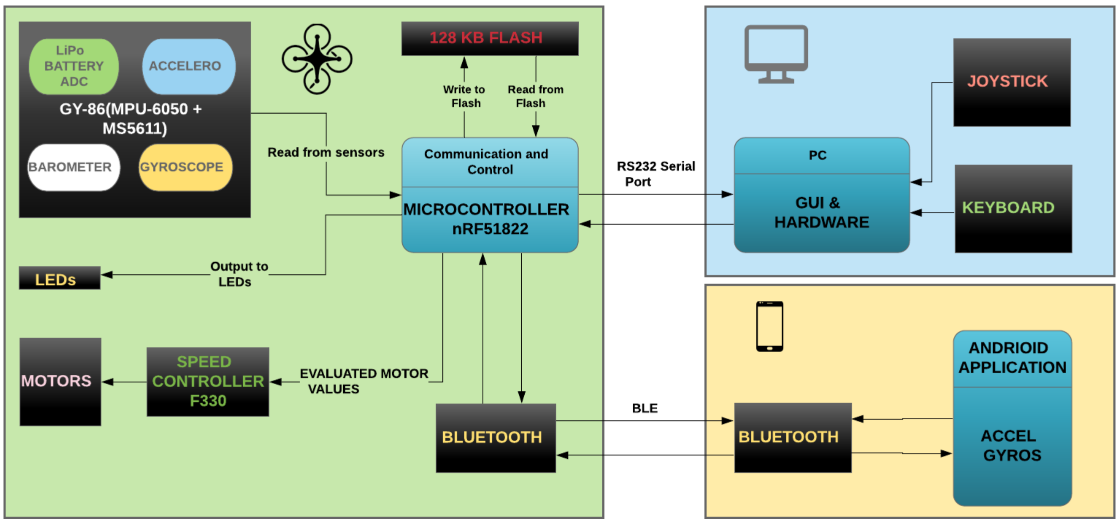


Figure 1: Overall System Architecture

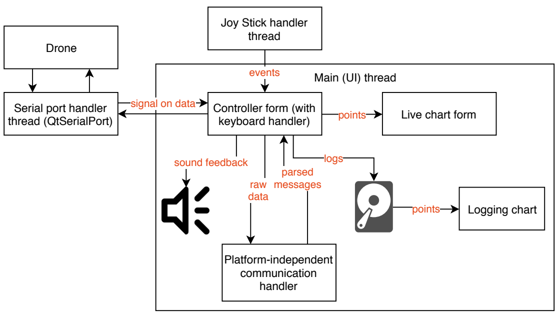


Figure 3: The architecture of the Qt application.

in both parties, its checksum is calculated and if it does not match with this field the packet is discarded as it may be modified unintentionally by the communication channel. It should be noted here, that for every message type the length is known for both parties. Thus, we always know how to access the last field of the received packet to check its checksum.

The connection between the drone and the PC side will be considered alive as long as messages are received on both sides. More specifically, if in the microcontroller side no message is received for 500 ms the drone will go into *panic* mode. Similarly, if no message is received by the microcontroller in the PC side for 500 ms the PC immediately puts the microcontroller into panic mode.

In order to keep the client and the microcontroller in sync for every message that is sent from the PC side, an acknowledgment is sent back from the microcontroller. Only then, the UI updates its state. For example, the UI will not update the lift value that it reports even if the 'Up' button is pressed unless the microcontroller acknowledges this change.

5.1.2 Logging

For the logging functionality in the microcontroller, the protocol contains 3 different messages. As shown in the table 1 in order to start and end the logging, a message with ID 'l' should be sent with the appropriate parameters. In particular, to start logging in the parameter 's' is sent and to stop the logging 'x' is sent. The microcontroller replies to the PC with the corresponding parameter in its payload. Thus, it is known on the PC side that the logging has been started or stopped. If the log dump is requested, the first message that will be sent from the PC side will request the size in bytes of the log file. In this way, it becomes clear in the PC side how many messages should be exchanged between the two parties in order to complete the transfer of the log file. After the size is known in the PC side the log file is transferred from the microcontroller line by line. Every line contains 40 bytes of data. More specifically, it contains the following variables:

- `time`
- `phi, theta, psi`
- `sp, sq, sr`
- `sax, say, saz`
- `bat_volt`

Key	**Description**	**Request Params**	**Response Params**	**Req. Len. (bytes)**	**Resp. Len. (bytes)**
s	transfer the battery state	none	`bat_volt`: the battery voltage	0	2
a	get values for debug	none	motor speed (ae[0] - ae[3]) and yaw (sr, sr_ofst)	0	12
m	control values from keyboard, joystick or phone	lift, roll, pitch, yaw, Yaw control gain, P1 and P2 controller gains	lift, roll, pitch, yaw, Yaw control gain, P controller gains P1 and P2	8	8
c	mode switch	'c': calibrate 'p': panic 'm': manual 'y': yaw 'f': full 'h': height control 'r': raw mode 'w': wireless 's': safe mode	'c': calibrate 'p': panic 'm': manual 'y': yaw 'f': full 'h': height control 'r': raw mode 'w': wireless 's': safe mode	1	1
r	sets the value of PI2PHI	PI2PHI nominator and PI2PHI denominator	PI2PHI nominator and PI2PHI denominator	4	4
l	logging control commands	's': start logging 'x': stop logging	's': start logging 'x': stop logging	1	1
b	get the size of the log file	none	the size of the log file	0	4
d	messages for the log dump transfer	none	lines of the log file	0	40

Table 1: Protocol summary

- `motor[0], ..., motor[3]`
- `pressure`
- `temperature`

To request a new line of the log file in the PC side a message with ID 'd' is sent to the microcontroller. When the number of bytes that are received in the PC side match with the total number of bytes that are written in the microcontroller no more lines are requested by the PC and the log dump has been successfully transferred to the PC.

5.2 Safety

This section summarizes the safety features of this aircraft: before performing a critical mode change (from safe mode to a mode where the rotors can be energized), all the lift, roll, pitch, yaw values must have zero values. Upon changing mode, the user interface loudly reads the name of the entered mode to give additional feedback about the operation for the user.

The safety feature on the microcontroller side are the following: when the system is not in safe in panic mode (so it is in an active mode) and there are no control messages through either the serial port or Bluetooth, after 0.5 seconds it goes to panic mode. Another case when the panic mode is activate is if the battery level drops under a certain threshold for 0.4 seconds.

If the lift value is 0, it is not possible to move the rotors even if we are in full control mode or raw control mode and there is a pitch or roll movement requested. Finally, in order to protect the motors, if based the lift value, the motor signals would be above 180, it cannot go below 180 so the motors will not stop even with aggressive controller parameters.

5.3 Modes

From the UAV theory of operation it is known that the force in the z-axis (Z) and the moments to be controlled by the roll (L), pitch (M) and yaw (N) are given by the following formulas:

$$Z = -b(ae_1^2 + ae_2^2 + ae_3^2 + ae_4^2) \tag{1}$$

$$L = b(ae_4^2 - ae_2^2) \tag{2}$$

$$M = b(ae_1^2 - ae_3^2) \tag{3}$$

$$N = d(ae_2^2 + aae_4 - ae1^2 - ae_3^2) \tag{4}$$

where b and d are drone specific constants and ae_i controls the duty cycle of the PWM signals that are applied to the engines of the drone. In order to control the drone the above equations have been inverted and solved according the ae_i values. The resulting system is shown in equations 5 to 8.

$$ae_1 = \sqrt{-\frac{N}{4d} - \frac{Z}{4b} + \frac{M}{2b}} \tag{5}$$

$$ae_2 = \sqrt{\frac{N}{4d} - \frac{Z}{4b} - \frac{L}{2b}} \tag{6}$$

$$ae_3 = \sqrt{-\frac{N}{4d} - \frac{Z}{4b} - \frac{M}{2b}} \tag{7}$$

$$ae_4 = \sqrt{\frac{N}{4d} - \frac{Z}{4b} + \frac{L}{2b}} \tag{8}$$

To avoid using slow floating point libraries a custom `square_root` function was implemented and used for those calculations.

Square-root function We implement a bisection (interval halving) method that works with 32 bit integers. To compute the square root of n, the algorithm initially sets $min = 0$ and $max = \frac{n}{2}$. Then it finds the half h of the interval and checks if n is between min^2 and h^2 or h^2 and max^2 and changes the interval accordingly. When $min + 1 >= max$ holds, the algorithm is terminated and returns min as the square root of n. Its drawback is that based on the input, its running time may vary (for larger number it takes more time).

5.3.1 Manual mode

For the manual mode the original values of `lift`, `roll`, `pitch` and `yaw` are first scaled by some factors (based on the feedback we received during the preliminary demonstrations) and then are used in the equations 5 to 8 for the calculation of the corresponding ae_i values.

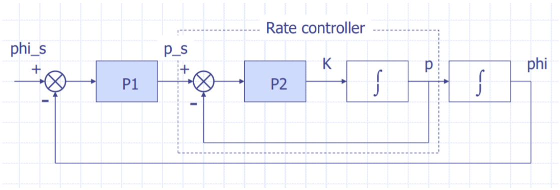


Figure 4: Cascade P controller.

5.3.2 Calibration mode

For the calibration mode, we designed a message from the PC side that enables this mode in the microcontroller. When this message is received the calibration procedure is started. A total of 255 sensor readings are taken from the sensors for *phi*, *theta*, *psi*, *sp*, *sq*, *sr*, *sax*, *say*, *saz*,and *height*. Then the average of all values is computed and later are subtracted from the new sensor readings. This is done to clear any kind of offset if its there even when the drone is in a stable position when kept on the ground.

Note that this calibration procedure is only used for calibrating the values coming from the DMP, and the raw mode calibrates itself on activation.

5.3.3 Yaw control mode

For the yaw control a simple P controller. First we compute the yaw error:

$$\text{err}_{yaw} = \text{ref}_{yaw} - tsr \cdot YC1$$

where ref_{yaw} is the reference value for yaw tsr is the actual yaw value from the sensor and $YC1$ is constant to tune the sensitivity. Then we use the following equation to determine N for equations 5 to 8:

$$N = \text{err}_{yaw} \cdot yawP$$

where $yawP$ is an online-tunable constant.

5.3.4 Full control mode

For the full mode the yaw P controller is extended with two cascade controllers for the pitch and the roll. For this purpose, we implemented two P controller in cascade for each of these events as shown in Figure 4[2].

5.3.5 Raw mode

The raw mode was implemented in order to read the raw data coming from the sensor, especially the MPU6000, that gives the measures of the accelerometer and gyroscope. When this

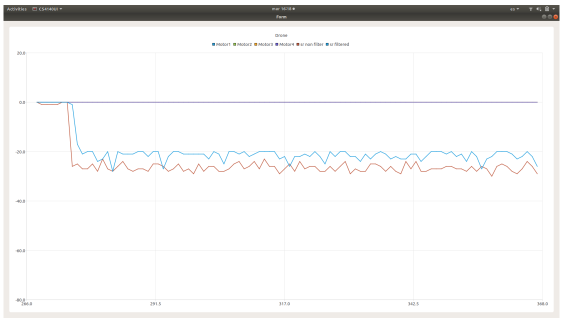


Figure 5: Response of the low pass filter with 25Hz cut frequency to no movements in the system.

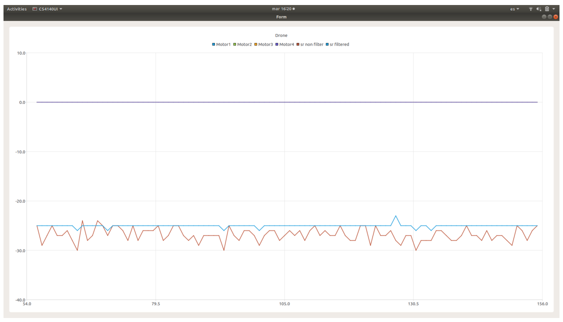


Figure 6: Response of the low pass filter with 50Hz cut frequency to no movements in the system.

mode is activated, the system sets up again the MPU6000 configuration, turning the DMP(Digital Motion Processing) off. We use a sampling frequency of 1 KHz in this mode for reading the sensor values.

We tried to implement a timer interrupt in order to obtain a fix time-stamp from the measures, but we were unable to read the sensor values from the interrupt routine so we ended up reading the sensors in the main loop.

After that, we process the measures with a Butterworth filter for the yaw and a Kalman filter for the roll and the pitch. For designing the Butterworth filter we use the applet given in the resource webpage of the course [5] and the information obtained for the fixed-point arithmetic in [1].

We designed different first order low pass filters at 25Hz and 50Hz cut frequencies and we tested their results in the system. In Figures 5 and 6 we can observe the results of the filtered yaw data when no movements are applied to the system. The 50 Hz filter obtain better results in terms of a more stable offset.

The coefficients of the filter are $a_0 = 1, a_1 = 1$ & $b_0 = 0.7265425280$. So in order to reduce the transformation error to a fixed point arithmetic, we used a 14 fraction bits.

Only the control loops input signal are computed differently in raw mode, so we use the same equations as full control mode to calculate the motor signals. To make this possible, the Kalman-filter use a custom scaling of the values, instead of the traditional bit shifting way. Our benchmarks showed that the implementation of the run time of the Kalman-filters takes insignificant time in the control loop, so we can afford these multiplication and division operations without harming the control loop's efficiency.

Figure 7 present the unfiltered and filtered angle values from the Kalman-filter. We can observe a maximal 30 milliseconds of delay in the filtered value in the worst case.

5.3.6 Wireless mode

The wireless mode was implemented to interface with Android phones. The drone can be controlled from an android app named *Flydrone* using a mobile phone and it mimics the phone movement. The app source code can be compiled with Android Studio and Gradle, and is supported on Android version 4.3 and above.

The communication between the nRF SOC and the PC using a wired duplex serial link is replaced by the Nordic UART Service (NUS) that emulates a serial port Bluetooth Low Energy (BLE). The main challenge was to downsize the channel bandwidth in case of Bluetooth communication. For this, only necessary data such as lift, roll, pitch, yaw values and control parameters like p_{yaw}, p_1, p_2, only are sent and receive between the drone and the phone.

The mobile phone contains a 3-axis accelerometer and a 3- sensor axis magnetometer. The accelerometer sensor reports the acceleration of the device in m/s^2 in 3 sensor axis that includes the physical acceleration and the gravity. The *sensor type* used is *SENSOR_TYPE_ACCELEROMETER* that gives values in x, y, z directions. These values are 0 when device is kept flat on the table. Moving towards left will give negative X- acceleration and to the right will give positive X-acceleration. Similarly, tilting the device forward will give positive Y-acceleration and tilting backwards will give negative Y-acceleration. For reading the magnetometer sensor values, *sensor type* used is *SENSOR_TYPE_MAGNETIC_FIELD* which gives

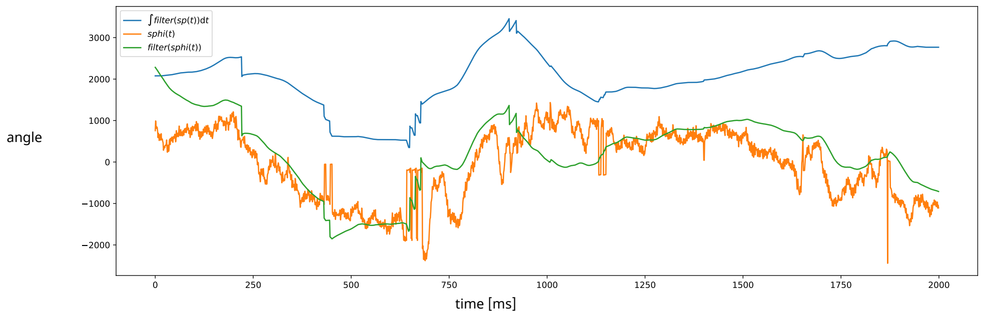


Figure 7: Kalman-filter

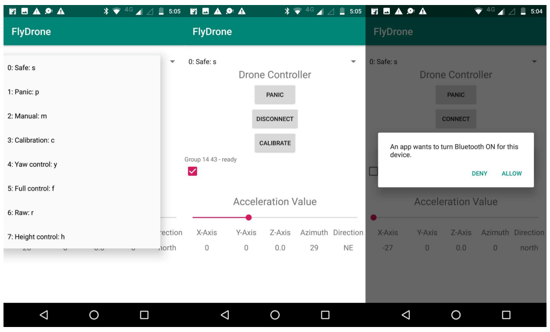


Figure 8: Bluetooth connection must be allowed in app, Flydrone android app GUI, Different modes available to select in the android app

the magnetic field value in *micro-Tesla(uT)* in the 3-sensor axes i.e. in x, y, z directions. We used an Android inbuilt API for calculating the azimuth value, which is the amount of rotation of the device around an axis perpendicular to the mobile screen. For this, the accelerometer and the magnetometer sensor values are fused together and a rotation matrix is calculated. This rotation matrix gives the x, y, and the azimuth values. These values are very noisy, so we applied a non-linear function and then amplified the values that are required for the drone acceleration. The azimuth values are then converted to degrees, and are then used for the *yaw* control of the drone. Rotating device to the right will give positive yaw acceleration and to the left will give negative yaw acceleration.

For connecting the Bluetooth module with the mobile app we used source codes from the Android-nRF-UART app version 2.0.1 [4] that helps in sending and receiving the ASCII and UTF-8 text strings from mobile to the Bluetooth module. We send a message from the PC side that enables the wireless mode. Wireless mode i.e mode 8, can only be switched in from the safe mode. After switching into mode 8, user interface loudly reads the name of the entered mode. From this time the PC does not send any messages, however it can request the drone to change back to wired mode (the drone still listens for the communication from the PC).

The app tries to enable the Bluetooth of the device, which must be allowed in any case. Also, the location access must be switched on in order to scan the available Bluetooth Low Energy devices in the mobile's range. Using the *Connect* button, Bluetooth connection with the drone can be established. A *Ready* message is displayed if the connection is successful, else *Not ready* is displayed and the module needs to be reconnected. After the successful connection with the drone, the checkbox needs to be selected for starting the communication with the microcontroller. Any mode of communication i.e. from mode 0 to mode 7 can be selected from the dropdown list provided in the top right corner of the app as shown in the figure. A *Calibrate* button is provided which resets the roll, pitch and yaw values in the mobile to 0 (if there's any). For controlling the lift of the drone, a seek bar is provided with the values ranging from 0 to 100. The pitch of the drone can be done by tilting the phone forward or backward, the rolling movement can be done by moving the phone either to the left or to the right. For the yaw movement, the phone should be rotated left or right along the axis perpendicular to the surface of the phone.

5.3.7 Height control

The height controller uses the data from the barometer, filtered with a first order infinite impulse response (IIR) filter. However, this sensor seemed to be quite inaccurate, as shown in figure 9 where we do a single lift of one meter between time steps 160 and 240. The raw signal is clearly very noisy. Filter 1 removes most of the noise, however, we can still observe a constant drift in the signal. Filter 2 filters more aggressively, therefore, fails to track the signal correctly. In order to solve these issues we have experimented with the development of a Kalman-like filter based on both accelerometer and barometer readings, however, finally, it was not included in the demonstration's a code.

For security reasons, the height control can change the value of the lift reference only by 33%. It should clearly be enough to keep the drone in height in expected operational circumstances and a properly-picked lift reference.

5.4 User interface

The user interface was implemented using the QT framework. We decided to use this framework because it has an extensive set of APIs available and a big community providing several examples for all the functionalities [3]. In the Figures 10 and 11 we show the GUI application. The main window is Figure 10 in which we can read the different values from the control of the drone, and also we can start the data logging and print it. In Figure 11 we present the real time chart used for debugging, which has the same format as are used to print the data logging.

Class variables hold the values set by the keyboard and the joystick: the joystick is read on a background thread, that sends Qt signals to the the UI when new events arrive. We use standard keypress events to handle keyboard input that is implemented according to the specification in the lab manual.

The main form has a timer that calls a function periodically, that sends the current control values computed by summarizing the the joystick and the keyboard values to the microcontroller. The period depends on the mode, in debug mode we use a lower frequency (the messages are send once every 450 ms) to be able to observe the exchanged messages, while in production mode the communication is fast (a message is sent every 20-30 ms, depending on the mode and if want to use the live charts) to decrease latency.

For debugging and safety purposes, the user interface shows only the values that are acknowledged by the microcontroller: this is useful in several ways: we always how long the reaction time of the drone is, and we can immediately see communication issues in the user interface, i.e. if incorrect or no values are present on the progress bars.

6 Experimental Results

Two kinds of benchmark methods were used: all benchmarks that measure the time of embedded C code fragments were run on the microcontroller. The communication response time benchmarks are based on measurements on the PC.

Table 2 summarizes the benchmarks executed on the microcontroller. It is based on 1000 samples for each code fragment.

Clearly, we can observe that in raw mode the execution time event including the filters is significantly lower than when we use the MPU for filtering. In raw mode we've measured an average sensor reading + filtering time of 873 μseconds, therefore even if the controller runs at 1 kHz, produces only about 87% memory consumption.

The communication benchmarking was done on the PC side. The testing was done by sending 1000 rounds of control change and state request values to the microcontroller and waiting for acknowledgement with the same values. About 0.7% of the messages were lost (in this case we did not receive a response with a valid checksum or no response at all: in the case the message was corrupted in the direction from the PC to the MC). Since this a very small amount of lost message, we've dropped the idea of re-sending the lost messages, because the controller value change messages are sent frequently enough and the messages loose their value with time. Therefore applying a queue would have more drawbacks. The state change request messages are sent while the state know by the PC and the MC is not equal, so if a state change request message is lost, it's still sent again until it succeeds. Our measurements showed that it took on average 13 milliseconds with a standard deviation of 9 milliseconds for a message to be acknowledged.

6.1 Scheduling

The benchmarks clearly show that communication can sometimes block the main thread for more than a millisecond. After noticing this, we tried to use the user timer interrupts to run the sensor readings and filters periodically, however, the microcontroller immediately crashed when we called the code for sensor reading from

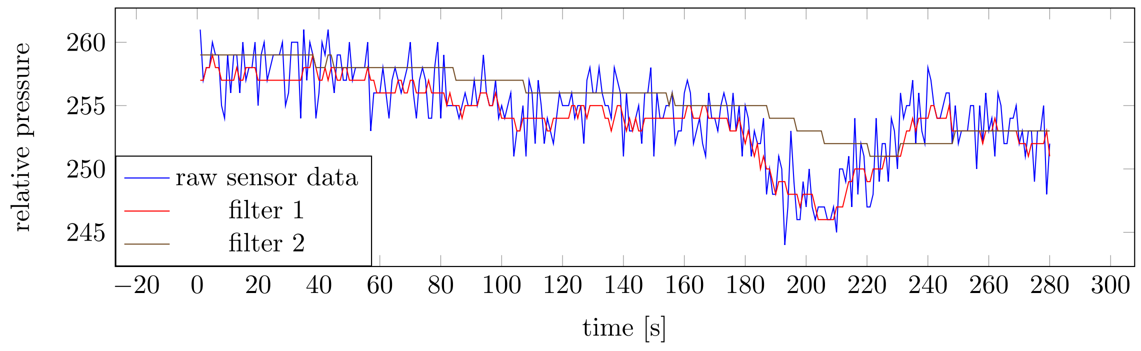
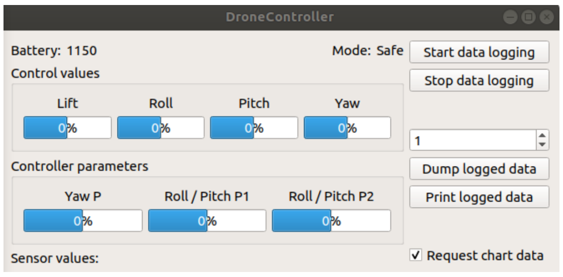


Figure 9: The height signal



Battery: 1150

Mode: Safe

Figure 10: Image of the GUI application.

code fragment	execution time [μs]			
	min.	avg.	max.	σ
reading DMP sensor values	3426	3641	4395	241
reading raw sensor values	704	764	1077	72
filters & control DMP mode	8	13	321	31
filters & control raw mode	90	109	409	33
serial receive	9	164	1260	334
bluetooth receive	9	78	836	207
time between control loop executions in DMP mode	9116	10067	10995	456
time between control loop executions in raw mode	727	900	1600	181

Table 2: Execution time of selected code fragments

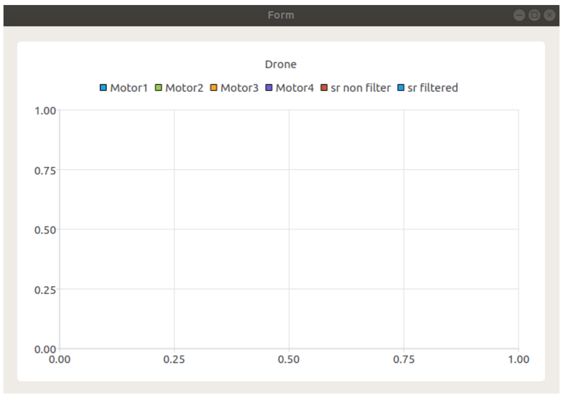


Figure 11: Image of the real time charts

the interrupt (although the new data flag was set).

7 Conclusion

To conclude our work we have successfully implemented a draw controller software with several modes: the full control mode showed relatively high stability compared to the inaccuracy of the sensors. In raw mode, using the Kalman filters and careful tuning of the parameters we were able to achieve higher stability that made the drone able to do basic hovering for shorter periods from our hand. Unfortunately, due to the lack of a proper and accurate interrupt-based scheduling, the raw mode showed some oscillations on longer flights so we were unable to take off from ground with the drone. The wireless communication works reliably, and height control was also implemented with partial success.

7.1 Participation in the code

Table 3 summarizes the contribution of the team members to each implemented function in the project's source codes.

Task Name	Beni	Stefanos	Pranjal	Miguel
Communication prot.	X	X		
Reading sensor				X
Calibration	X		X	
Data logging		X		
Yaw control	X		X	
Full control	X			
Raw mode	X			X
Butterworth Filter				X
Kalman Filter	X			
Scheduling	X			X
Wireless	X		X	
Android wireless	X		X	
Android comm. prot.			X	
Android sensor read			X	
Android GUI impl.	X		X	
Qt app core & UI	X			
Qt threading	X			
Qt live charts	X			
Qt joystick handling	X			
Qt protocol controller	X	X		
Qt data logging		X		
Qt data logging charts				X

Table 3: Contribution of team members

References

[1] Sait Izmit. Floating-Point Support for Embedded FPGA Platform with 6502 Soft-Processor.

[2] Koen Langendoen. Lecture 4 slides. http://wwwtmp.st.ewi.tudelft.nl/koen/cs4140/lect04-Control.pdf. Accessed on 23.06.2019.

[3] The Qt Company Ltd. Qt examples and tutorials. https://doc.qt.io/qt-5/qtexamplesandtutorials.html. Accessed on 23.06.2019.

[4] NordicPlayground. Android-nrf-uart. https://github.com/NordicPlayground/Android-nRF-UART, Oct 2015. Accessed on 15.05.2019.

[5] Koen Langendoen Sujay Narayana. Cs4140 project resources. http://wwwtmp.st.ewi.tudelft.nl/koen/cs4140/Resources/index.html. Accessed on 23.06.2019.