Delft University of Technology

Embedded Systems Laboratory
CS4140ES

Embedded Systems Lab Group 21

Authors:

February 26, 2022

Abstract

This report introduces the design of embedded software for controlling and stabilizing an unmanned aerial vehicle. The embedded software for the UAV consists of two parts. The first part runs on a PC control station while the second part is on the microcontroller on the drone, which is implemented as a finite state machine (FSM). Different tasks are performed in the FSM in a round robin fashion. Seven working modes were successfully implemented and tested on the unmanned vehicle as elaborated in this report.

1 Introduction

In this course, we have designed embedded software to control and stabilize an unmanned aerial vehicle (UAV), the Quadruple quadrotor. The quadrotor (also referred to as 'drone') has seven working modes[1]. The drone can be controlled with flight inputs from a joystick and these inputs can be trimmed with the keyboard. The real time values of motors, mode of operation and battery voltage can be monitored using a Graphical User Interface (GUI) running on the PC control station.

The embedded software is designed on an NRF chip which uses an ARM Cortex M0 processor. The communication between the PC control station and the drone is established using a RS232 link. The drone's attitude[2] and attitude rates are measured using the MPU6050 inertial measurement unit. External disturbances on the drone during flight are counteracted by a proportional control loop implemented on the drone. We also experimented with drone control in a 'raw' mode which uses raw data from the MPU6050 instead of the data from it's on-board Digital Motion Processor (DMP). A Kalman filter and a Butterworth filter are implemented to acquire accurate sensor outputs from the MPU in this mode. Finally, we also implement a height control mechanism, a so called semi-autopilot, that enables the drone to maintain it's height.

This report details the implementation as well as the results of our demonstrator. Stability tests were carried out with the drone and profiling of the embedded system was done to check the speed of the control response. Section 2 details our system architecture and Section 3 covers the different implementations. We elaborate on the experimental results in section 4 and make some conclusions in section 5.

2 Architecture

Software architecture is divided in two parts, *control station* and *drone*. *Control station* is software running on the PC, which handles serial communication with the drone, joystick and keyboard inputs and displaying data on the GUI. *Drone* part of the software implements the finite state machine which handles all aspects of drone's operation. It controls mode switching, implements the control loop, sensor acquisition and filtering, battery monitoring, logging and motor control. The overall software architecture is shown in Figure 1.

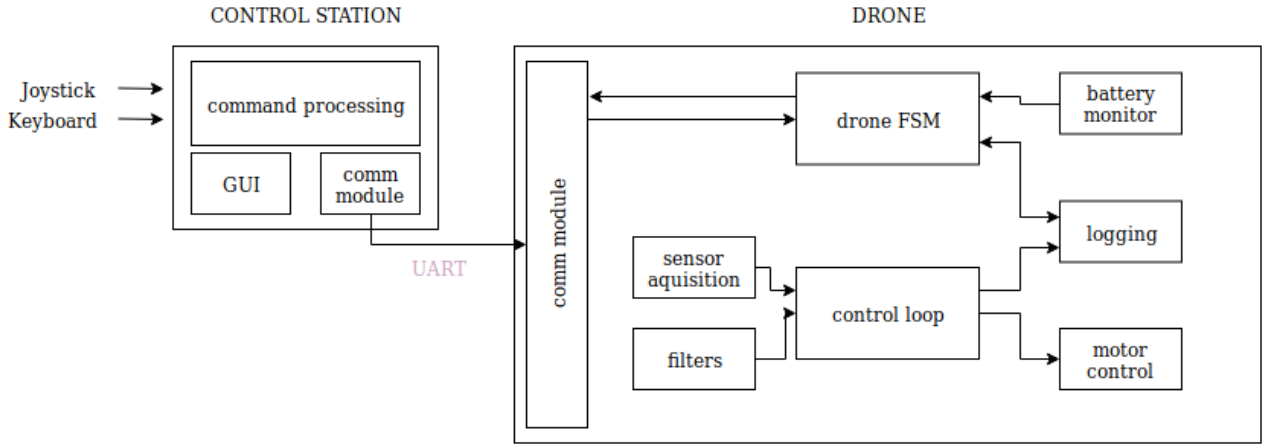

Figure 1: Software architecture diagram

[1] safe mode, panic mode, manual mode, calibration mode, yaw-controlled mode, full-controlled mode, raw mode, height-controlled mode

[2] drone attitude ϕ, θ, ψ

2.1 Control station

Control station is implemented in a round-robin fashion, where the following tasks are performed:

- keyboard event handling
- joystick event handling
- serial data rx/tx
- packet decoding
- periodic packet sending

Keyboard buttons trigger corresponding actions, creating a packet and adding it to the transmit queue. Joystick buttons are handled similarly. However, joystick axis movements and keyboard trimming buttons, only update the flight parameters (roll, pitch, yaw, lift) which are sent periodically.

The number of bytes sent/received in each loop iteration is bounded, to prevent unexpected communication delays. Incoming data is stored in an rx_queue and processed separately with packet decoding state machine. Data in tx_queue (packets waiting to be sent) is being sent gradually in each loop iteration.

2.2 Drone

Drone software is implemented as a finite state machine, where each state corresponds to one mode of operation. In every loop iteration, incoming and outgoing serial data is controlled and packets are being processed with an independent, packet decoding state machine (same implementation as used in control station).

Seven modes of operation of the drone are implemented: *safe*, *panic*, *manual*, *yaw-control*, *full-control*, *raw* and *height-control*.

- *Safe mode* is a stationary mode (not reachable when flying) which ensures that all motors are shut down and this mode only reacts to mode change packets.
- *Panic mode* is reachable from all flying modes and is triggered with a mode packet, when emergency button is pressed.
- *Manual mode* is the most basic flying mode which maps flight parameters (16-bit values obtained from joystick+keyboard) to the motors linearly.
- *Yaw-control* mode maintains yaw stability by utilizing yaw rate values (obtained from onboard MPU) and implementing a simple rate controller.
- *Full-control* mode, in addition to the previously described yaw mode, implements controllers for roll and pitch as well. All the sensor values are obtained from MPU and no signal filtering is needed in this mode.
- *Raw mode* is the most computationally intensive mode, which performs the same task as full-control mode without using the MPU's DMP.

Filtering of the signals is described in further detail in Section 3. Besides the modes mentioned above, there are several modules which run independently of the mode. Battery monitoring, reads battery voltage level and switches to panic mode when voltage drops below 10.5V. Logging module sends information about drone's current mode, battery voltage and motor speeds to the control station, at the rate of 5 Hz. In drone's software, several interrupts are being utilized: UART interrupt (serial communication RX), timer interrupt (motor value updating, sensors), and ADC interrupt (battery monitoring).

2.3 Communication Interface

The communication between control station and drone is packet-based, where packets are sent both periodically and on-demand. The packet structure can be seen in Figure 2.

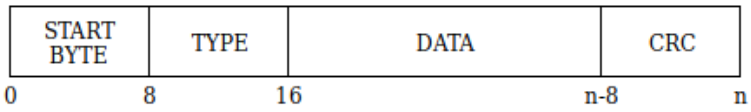

Figure 2: Packet structure

There are four types of packets implemented: *flight*, *mode*, *control* and *logging*.

- Flight packets are sent periodically, updating the flight parameters (roll, pitch, yaw, lift) of the drone and serving as a liveness check. These packets are sent at a frequency of 50Hz from the PC to the drone. No acknowledgement mechanism exists for these packets since a few packet misses is non-critical.
- Mode packets are sent on-demand when switching between drone's modes of operation. The drone acknowledges these packets and sends out a response about it's mode switch.
- Control packets are sent on-demand and are used for updating the gain values in the control loop. The drone acknowledges these packets and sends out a response about it's new control gains.
- Logging packets are used for both periodic reporting to the PC as well as transferring logging data stored on flash memory.

Both the control station and the drone implement the same software module for processing packets. A *packet decoding state machine* reads bytes from the receiving queue and decodes the packets. Upon a successful CRC check, the appropriate action is taken based on the content of the packet. For example, when drone receives a mode packet, it will switch to the given mode. More details about the packets used can be seen in the Appendix.

3 Implementation

The system was implemented in a team, where each of the four members imlpemented several parts of the software. Jelger implemented datalogging and profiling modules, height controlled mode and CRC checking functionality. Ji Hang developed the GUI, sensor calibration and worked on control loop design (with Snehal). Snehal developed the drone's state machine, fixed-point arithmetic, filters and controllers. Jure designed the communication protocol, implemented the control station and maintained the overall project structure (software architecture).

3.1 Safety

To make sure the quadrotor is safe for use, even when something unexpected happens, a few safety precautions have been implemented in the software. These are the following:

- *Communication failure detection*:

  The communication from the PC to quadrotor is verified to make sure that communication is still working as it should. This is implemented by counting the number of received packets every 50mS. When there are zero packets received during this period, and the drone is in any flight mode[3], it will automatically go to panic mode and land in a safe manner. This system protects against cable disconnects, but also against a crash on the PC side.

- *Mode switching protection*:

  Whenever a mode switch is requested to any flight mode, the joystick position and keyboard trim values are checked before the mode switch is accepted. This is implemented to make sure the quadrotor enters a flying mode in a safe way.

- *Battery under-voltage protection*:

  The battery voltage is monitored once per second, and when the voltage goes below 10.5 volts, the quadrotor will go to panic mode, and a low battery indication will be given in the terminal and the RED LED will turn ON.

[3]Flight modes are: manual control mode, yaw control mode, full control mode, height control mode

- *Emergency stop*:

  To be able to stop the quadrotor quickly if necessary, an emergency stop is implemented. When either the escape button or '1' is pressed, the quadrotor will enter panic mode to safely stop in case of emergency.

3.2 Controls

The control implementation includes two parts. The first part is the joystick input to motor mapping. The second part is the actual control loop.

- *Motor Mapping*: The input from the joystick is taken as the set point for the drone. The input value of the joystick is a signed 16 bits integer (range from -32,768 to 32,767). The input value is converted to a signed 8 bits integer (range from -128 to 127) by shifting right by 8 bits. Finally, the shifted input values change the value of corresponding actuator signals $ae1$ to $ae4$ according to the dynamic equations of the drone.
- *Control Loop*: The implementation of control loops consists of two types of control. The first type is a proportional control loop for *yaw-control*. The second type is a cascaded control loop for *roll-control* and *pitch-control*.
  - The proportional control loop compares the difference between the set point of yaw (converted input value from the joystick) and the actual yaw rate sr. After testing, the optimal p for proportional control loop was chosen as 24.
  - The structure of the cascaded control loop is shown in Figure 3. The outer loop of the cascaded controller compares the difference between the set point of roll/pitch and the actual drone attitude ϕ/θ. The inner loop of the cascaded controller compares the output of the outer loop and the actual roll/pitch rate. After testing, the optimal values of $p1$ and $p2$ were found to be 17 and 46 respectively.

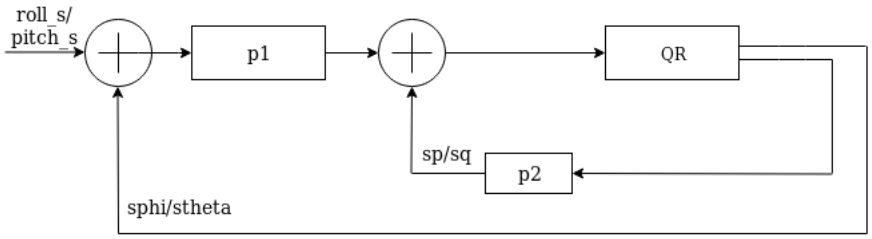

Figure 3: Cascade Control Loop

3.3 Calibration

In order to acquire a more trustworthy control loop, sensor calibration is implemented to obtain the offsets of different sensor values. The implementation calculates the average of 512 samples of sensor values when the drone is static and in level attitude. Those averages are stored in corresponding offsets. When comparing the sensor values and the set points of the joystick for control, the offsets are removed from these raw sensor values.

3.4 Filters

Another mode implemented on the system was the 'raw' mode which involved using raw data from the sensors as opposed to data from the sensor's on-board Digital Motion Processor (DMP). The advantage of running in Raw mode is that it enables us to sample the sensor (the MPU accelerometer and gyro) as a much higher rate. Moreover, this allows us to filter the noisy raw sensor data as per our own requirements as will be elaborated in this section.

- DMP mode sampling rate: 100 Hz
- Raw mode sampling rate: 500 Hz
- Butterworth Cut-Off frequency: 5 Hz (2^{nd} Order)

Although the sampling rate can be set much higher (up to 8 KHz), we used a smaller sampling frequency keeping in mind the time restrictions of our implementation on the microcontroller. As will be elaborated in the profiling section, the rate at which our state machine and all it's functions can be run is about 1.2 KHz.

However, when datalogging is enabled, this rate drops down to about 550Hz. Considering this limitation and understanding that datalogging is necessary to test the performance of filters, the sampling rate was set to 500Hz.

There are two types of filters that were necessary and are implemented for our project: A low-pass filter and a Kalman filter.

3.4.1 Low-pass Butterworth Filter

The raw values obtained from the sensor can be quite noisy and thus to avoid vibrations, need to be filtered to values realistic for actual quadrotor dynamics. Thus a low-pass filter was implemented to smooth the sensor signals to be used for control. Although the angle estimate is quite noisy too, the low-pass filter is only implemented for the gyro rates since a different Kalman filter implementation is used for estimating the angles (detailed in the next subsection).

An efficient low-pass moving average filter is the Butterworth filter. It achieves good results while requiring little memory to store previous sensor values. We implemented a second order Butterworth filter with a cut-off frequency of 5Hz. The second order nature ensures a sharper cut-off while 5Hz was chosen as an optimal cut-off frequency based on observations from the data log and oscillations observed in the sensor angular rates.

Implementation was done using fixed point arithmetic (with a precision of 14 decimal places) to implement the 2nd order butterworth equations using the calculated values for the butter parameters via MATLAB. Note that while all the angular rates were filtered out for smoother control, we still used the raw values of the angular rates for the Kalman filter implementation as will be explained in the next subsection.

The results from the Butter-worth filter implementation will be elaborated in the section 'Experimental results'.

3.4.2 Kalman Filter

To estimate angles from the raw accelerometer and gyro data, we need to solve two problems. The angular rate obtained from the gyro could be integrated over time to provide the angle estimate. However, the angular rate and the estimated angle have the tendency to drift, especially due to the integration. On the other hand, the raw acceleration values sax and say can be used to get a rough estimate for the pitch and roll angle respectively (for small angles). However, the raw acceleration values are very noisy especially due to the vibrations of the drone while it flies.

Thus, a Kalman filter is used to get the best of both worlds and fuse the accelerometer and gyro estimates to get a drift-free and smooth estimate of the roll and pitch angles. Implementation was done in a similar fashion to the reference 'kalman_control' provided for the course and we used fixed point arithmetic (with a precision of 14 decimal places). The Kalman parameters were tested out using the raw sensor data from our data logs and simulating via MATLAB.

The results from the Kalman filter implementation will be elaborated in the section 'Experimental results'.

3.5 Datalogging

In order to datalog periodically (when datalogging is enabled), snapshots of important parameters are written to flash memory in every iteration of the state-machine. This means that dataloggging is done at the fastest possible speed in DMP mode, and in RAW mode the speed is synchronized with the filters, (and therefore runs at 500Hz).

The following parameters are recorded in the log data: timestamp, current mode, motor speeds, ϕ, θ, ψ, sp, sq, sr, sax, say, saz, p, q, r.

This takes 37 bytes per log, which means approximately 3500 logs can be written until the flash is full (approximately 7 seconds of datalogging at 500Hz).

Log data that is written to the flash memory can be send back to the PC only when the drone is in safe mode. In this case, datalogs are send to the PC using datalog-packets (start-byte, type byte, 37 data bytes, crc8). On the PC side, this data is saved to a .dat file in the gnuplot format. The logdata can also be visualized using the graphical user interface.

3.6 Height control

In order to realize the height control, the output of the barometric pressure sensor is used to get an indication of the current height. At the moment height control gets turned on, the current barometer value gets stored as a setpoint and the height control loop is used to constantly adjust motor speeds in order to keep the height at this

setpoint. The control loop used for height control consists of a single p-controller which takes raw (unfiltered) values from the barometer, and outputs an offset value which is added to the current motor speed values. This is done to make sure the other functions, like rolling and yawing are still functioning in height control mode.

3.7 Graphical User Interface (GUI)

The implemented graphical user interface is shown in Figure 4. It was designed in *Qt Creator* using C++. The real time values of motor speeds, inputs of joystick, battery, and modes are displayed. Moreover, the log data of the sensor values is also displayed in the GUI, which can provide an overview of the result of the implemented filters. On the PC side, the current values of motor speeds, input of joystick, and battery are written as a line in a file at a frequency of 40 Hz. Inside the GUI, we parse the line inside the file. The separated values after line parsing are set to the corresponding widgets at the update frequency of 40 Hz.

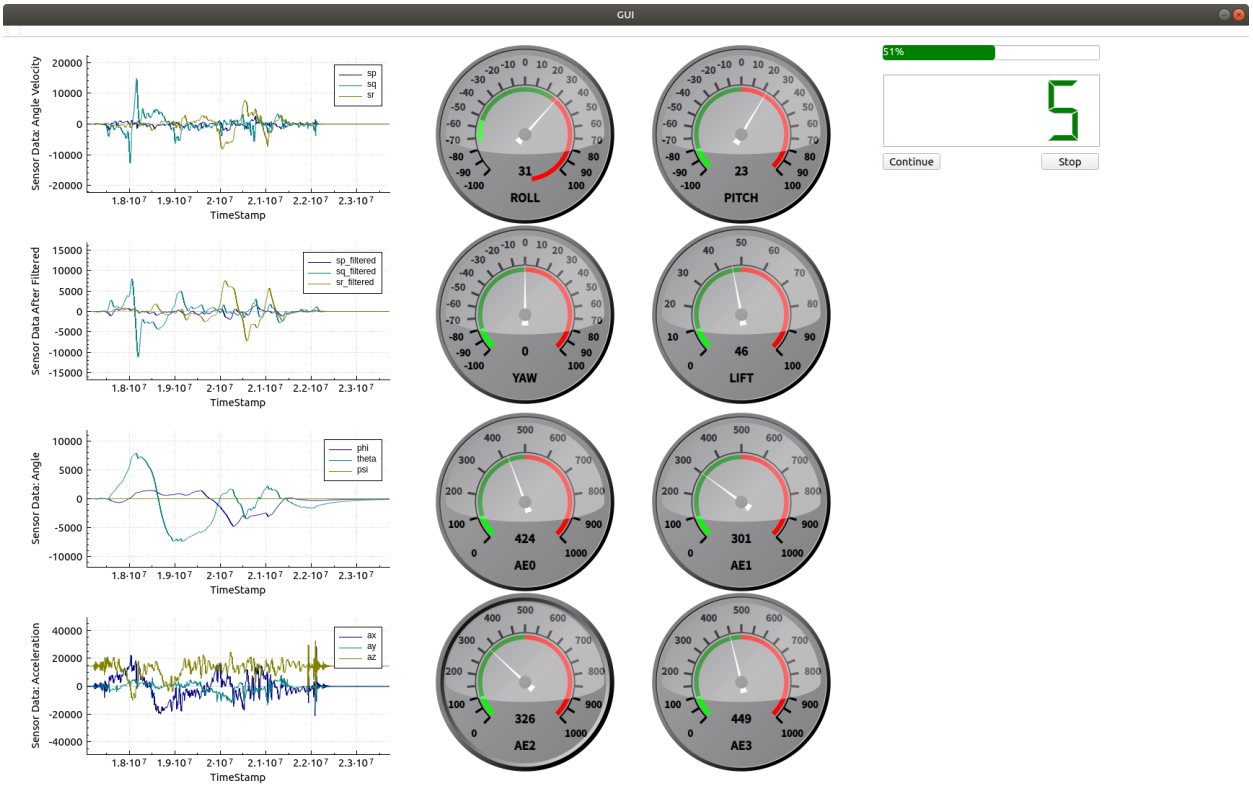

Figure 4: Graphical User Interface

4 Experimental results

4.1 Filters

To test the filter results in raw mode, we made use of the datalogging feature of our implementation. In our implementation, it is possible to run the sensor acquisition and filters at a high rate (at say 1 KHz). However, with datalogging enabled, the sensor acquisition has to be slower in order to meet the timing constraint since datalogging takes time. We thus log the raw as well as filtered sensor data after every run of the filters at 500 Hz. The filter outputs can be seen in Figures 5, 6 and 7.

- *Butterworth*: Upon testing with both 10 Hz and 5 Hz cut-off frequencies, the 5 Hz filter performed better since our rates (especially pitch and roll) were quite noisy and can lead to a lot of instability in the angular rate control with P2. A second order Butterworth filter was chosen for the same reason.
- *Kalman*: The Kalman filter was tuned with different values of C1 and C2 and the optimal values were chosen as 256 and 262144 respectively (this was of course implemented as bit shifts of 8 and 18 respectively). With these settings, we were able to get a reasonable level of performance and the fused output is visualized

in Figures 6 and 7. One drawback of the filter is that it introduces a significant **phase delay of about 266 ms** (measured using MATLAB) between the actual angle and the filtered angle estimate.

- The time taken to run the filters was minimal relative to the total execution time of the state machine and all timing constraints were met even with our high precision fixed point implementation.

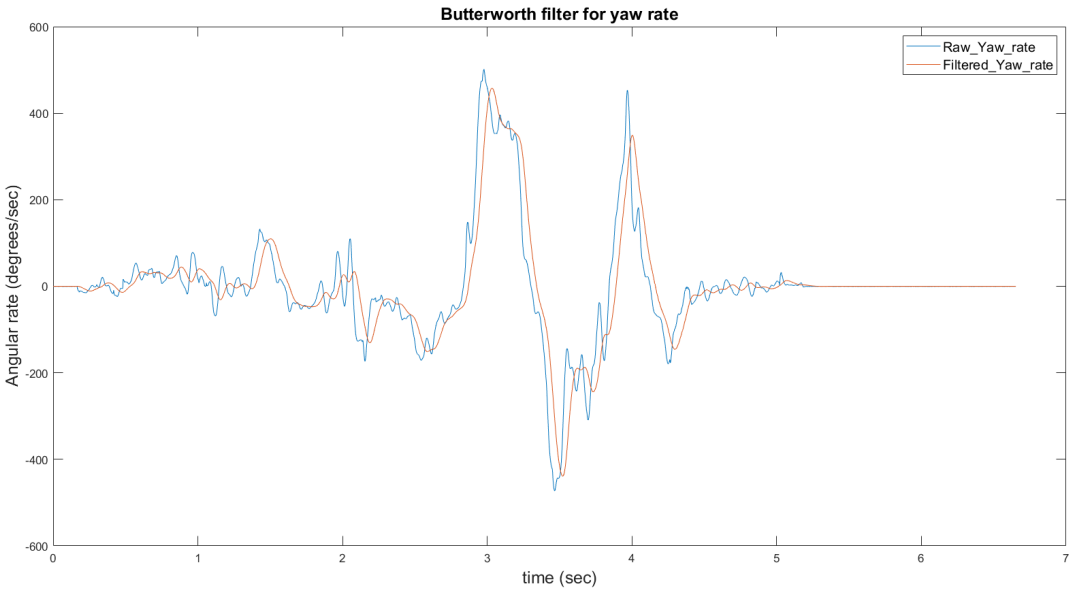

Figure 5: Butterworth Filter: Yaw Rate (2nd order 5Hz cut-off)

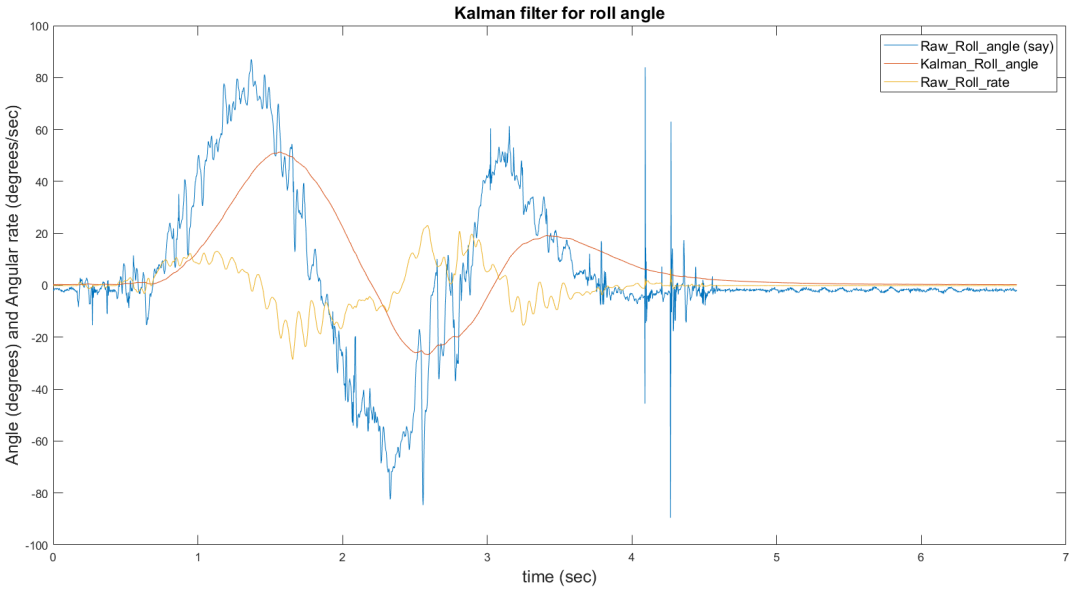

Figure 6: Kalman Filter: Roll (Phase delay 266ms)

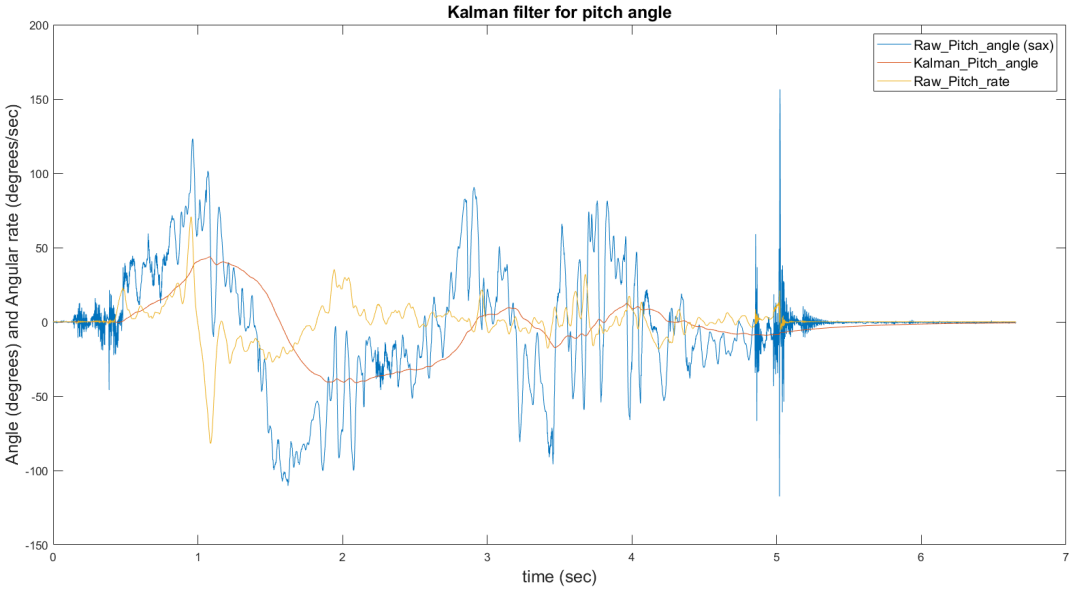

Figure 7: Kalman Filter: Pitch (Phase delay 266ms)

4.2 Stability and Control

Tests were carried out on the Quadruple drone to test whether our control implementation was able to stabilize the drone.

- In DMP mode (i.e. filtered sensor values from the MPU), the yaw-control as well as the full-control mode of the drone ran smoothly and gave good stability. After trimming, the drone is able to reject oscillations and fly accurately as per the inputs provided by the joystick.
- In Raw mode, we were able to match the performance seen in DMP mode for angular rate control (yaw, pitch P2, roll P2). The Angle control performance however was not as good as the DMP mode. This is primarily due to the Kalman filter implementation which causes some phase delay in the system. This can cause oscillations during operation especially for large gain values. However, we were still able to achieve reasonable levels of performance with a smaller P1 gain for the roll and pitch control.
- Height control works as expected with the motor speed reduction and increase observed when the drone's height is above and below the setpoint respectively. The barometer data, however, only changes for quite a significant change in height and thus we obtain a relatively coarse height adjustment. In the future, the barometer data could be fused with the z-axis accelerometer data to provide a better height estimation.

4.3 Software Profiling

The profiling of the software was done using a timing mechanism implemented on the drone using the 'get_time_us()' function.

Hard constraints on running the control loop for this project were met (10ms for dmp mode and 2ms for raw mode). The profiled control frequency in both dmp and raw mode are provided in Table 1.

Task	dmp mode	raw mode
FSM execution (ms)	3.7	0.78
Control loop frequency (Hz)	270	1282
Controller execution time (ms)	0.01	0.01
Filters execution time (ms)	0.061	0.061
Kalman phase delay (ms)	N.A.	266

Table 1: Software profile

5 Conclusion

In this course, we acquired the experience of designing a complete embedded software application. We have designed our communication protocol and successfully used it for serial communication with the drone. Control loops and filters were also successfully implemented in our design to control and stabilize the drone. Due to the lacking float-point arithmetic of the chosen CPU, fixed-point arithmetic was also implemented and used for calculations in filters. In order to evaluate the filters, datalogging was implemented to plot curves of the actual sensors values and the filtered sensor values. A visualization of real time values of motor values, sensors values, and drone attitude was achieved by a graphical user interface.

Some design choices in our implementation are shown in Table 2. Lines of code are measured by *cloc*, shown in Table 3 (not taking Makefiles into account).

motor speed	min speed (rpm)	max speed (rpm)
	200	850
flight communication	communication frequency (Hz)	
	50	
datalogging	logging frequency (Hz)	
	500	
fixed-point arithmetic	decimal points	remaining points
	14	18

Table 2: Overview of design parameters

	lines	folder
Control station	846	(control_station/controller)
Drone	1426	(drone/main)
Drone drivers	4679	(drone/drivers)
Shared	312	(shared)
GUI	723	(GUI)

Table 3: Code size

Appendix

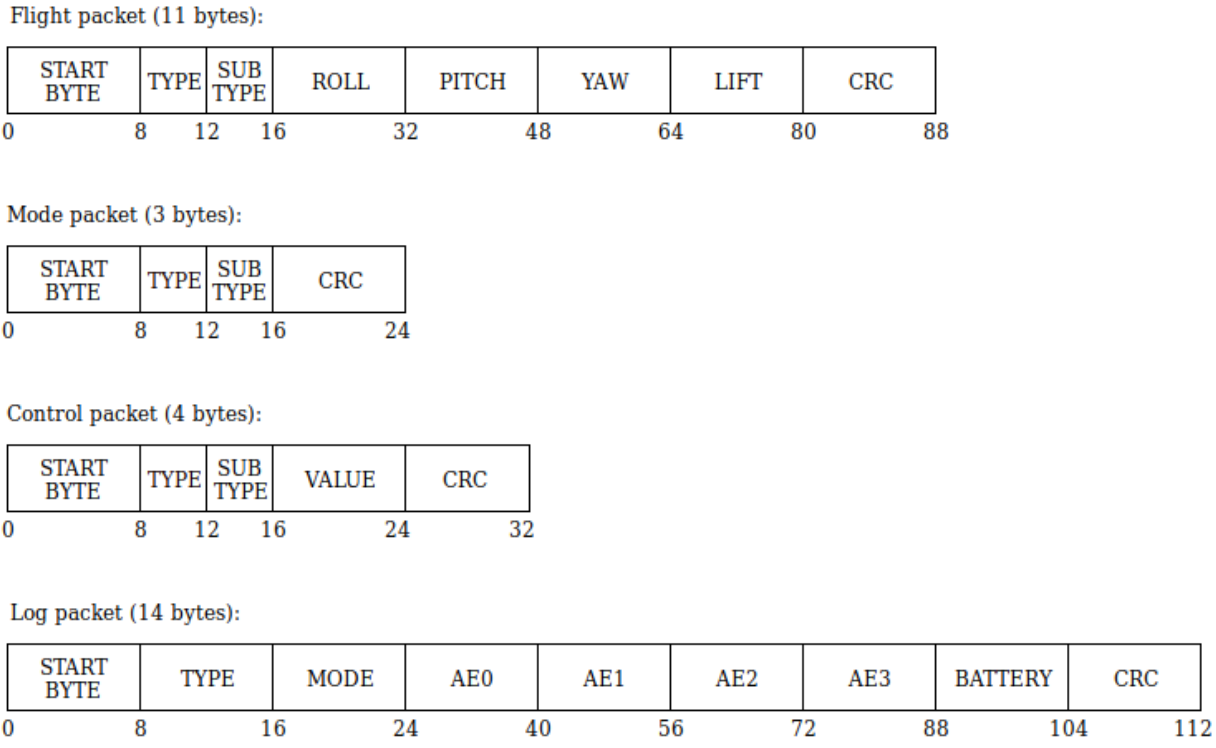

Figure 8: Packet types