Technical Report Embedded Systems Lab

Jelle Komen (5837995), Jiacong Li (5656354), Sukanya Pal (5851424), Marijn Sluijs (5071364)

Group 14. TU Delft, April 16, 2023

Abstract

This report describes the development of a system to control a quadrupel drone, in order to fly with user control and provide stabilization. The system gives the user the ability to send commands to the drone with a joystick and keyboard. The system will control the drone based on the commands, and sends back relevant data to the computer. Multiple flying modes are developed to fly the drone, the first one being the manual mode, where the drone is controlled without any stabilization. In yaw controlled mode, the yawing of the drone is stabilized, meaning that the drone will counter-act when the drone is yawed by wind or another disturbance. In full control mode the pitch and roll are also stabilized. In raw mode, the drone is stabilized by filtering the sensor values using a Kalman filter, instead of using the values filtered by the DMP (Digital Motion Processor). In the last mode, the height control mode, the height of the drone is controlled using the barometer.

Introduction

The Embedded Systems Lab project provided us with a unique and exciting challenge to design and develop a system that can stabilize and fly a quadrupel drone, using a combination of control theory and embedded software. The system had to be written in the programming language Rust.

Our main objective was to integrate a flight control board (nRF51822) with a sensor module and motor controllers, and develop software that can receive input from a computer via joystick and keyboard commands. The input will then be translated into a command that the drone can understand, and sent to the drone via a specific protocol over a serial link. On the drone side, the command will be extracted and executed accordingly.

To ensure safe drone operation, we also implemented two important safety features: a panic mode and a safe mode. In the panic mode, the drone will gradually slow down until it comes to a complete stop, while in the safe mode, the motors will remain at zero rpm at all times. These safety features will be essential in case of an emergency or drone malfunction.

Throughout the project, we relied on our knowledge of control theory and embedded software to create a system that can fly and stabilize the drone with high accuracy and precision. Overall, the Embedded Systems Lab project was a challenging but rewarding experience that allowed us to develop practical skills in the field of control theory and embedded systems.

Architecture

Functional block diagram of the system architecture

The architecture of the system developed to control the quadrupel drone is shown in Figure 1. It includes all interfaces and software components that have been written. On the left side, the PC interface including its components is shown. The inputs of the joystick and keyboard are mapped to a command and combined into a settings logic command. This settings logic command is placed into a packet together with the CRC of the command, to give the receiver of the packet the ability to check the command for errors. The packet is then written over USB to the drone. The settings logic command is also shown in a GUI on the screen, giving the user the ability to see the settings (Figure 3.

On the drone, the packet is checked for errors using CRC. If the packet is received correctly, the command is extracted from the packet and the drone is controlled. A datalog is created, which includes the motor values, sensor values and delay, and the datalog is sent to the PC over USB, to be displayed in the GUI.

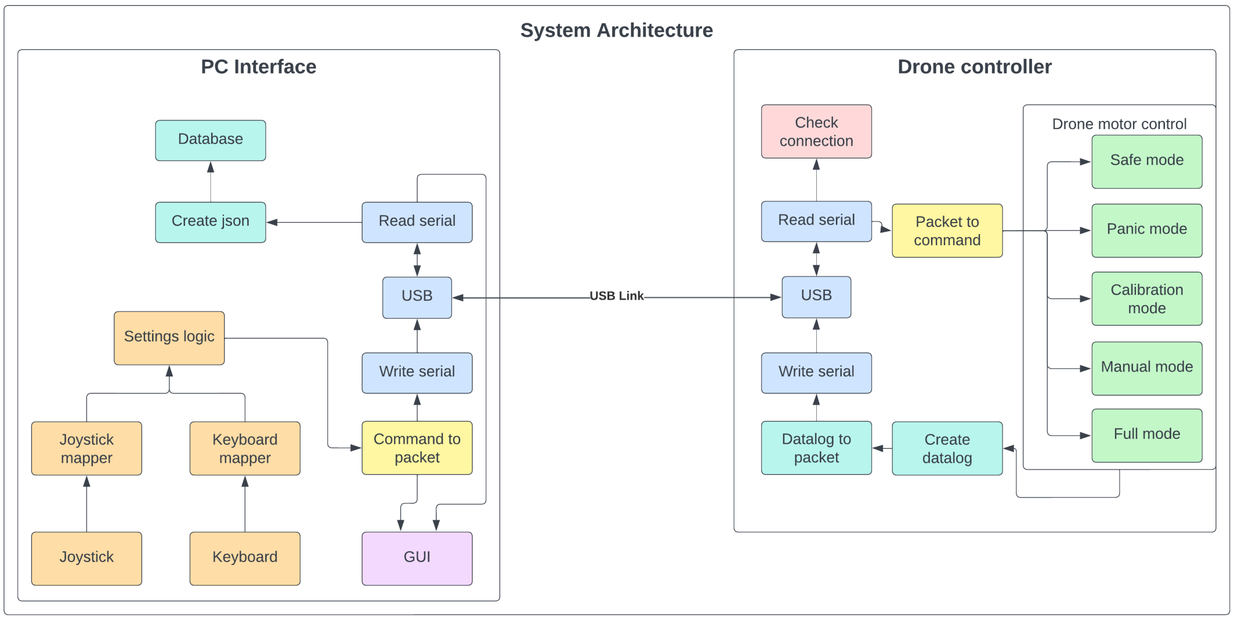


Figure 1: Functional block diagram of the system.

Implementation

Main Loop of Drone

This part will introduce the main operating loop of the drone. The main operating loop refers to the infinite loop that the drone will always execute with all actions of the drone contained. Once the code is uploaded to the drone, the drone performs initialization and then executes the loop until interrupted by a user command or by disconnection of the cable. The operation loop can be mainly divided into three parts: command processing, sensor data measurement, and datalog generation.

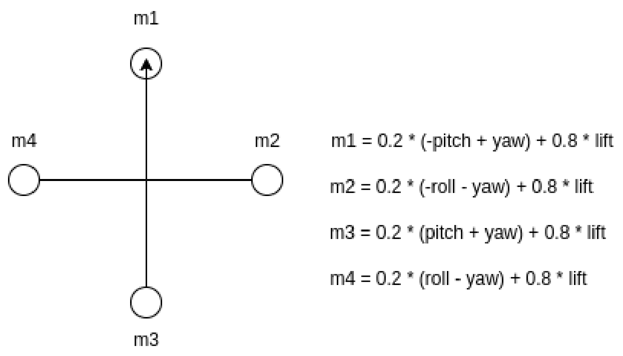
After the drone receives user instructions, it will switch modes and take corresponding actions according to the corresponding instructions. If users want to switch between modes, the users will need to set the drone to safe mode first. When the command to go to safe mode is given, the drone will automatically execute panic mode and finally return to safe mode to ensure that the drone's rotors can stop rotating. After the drone performs mode switching and corresponding operations in each cycle, it will collect sensor data, record and send data logs to the PC.

The final results show that even if the drone runs in raw mode, the mode with the most computations, when the control frequency is set to 100 Hz, the idle time of each cycle is still about 5ms, so task management algorithms like round robin are not applied.

When the cable is disconnected, the drone will automatically execute panic mode. The cable connection between the drone and PC is checked by counting how often a packet is not received in a row. The PC sends packets to the drone at a rate of 100 Hz. When the drone has not received a packet for 30 ms (corresponding to not receiving 3 packets in a row), it will conclude that the cable is disconnected and execute panic mode.

Motor Assign

Motor Assign refers to changing the speed of different motors according to the command sent by the PC to realize pitch, roll, yaw and lift. We tried to use Matlab modeling to find the relationship between motor speed and expected torque at the beginning, but due to the lack of many physical parameters of the drone, we finally decided to use the relationship between user expectations and drone motor speed directly. The specific relationship is as follows:



m1 = 0.2 * (-pitch + yaw) + 0.8 * lift

m2 = 0.2 * (-roll - yaw) + 0.8 * lift

m3 = 0.2 * (pitch + yaw) + 0.8 * lift

m4 = 0.2 * (roll - yaw) + 0.8 * lift

Figure 2: Motor assign formulas of Cross-shaped quadcopter drone.

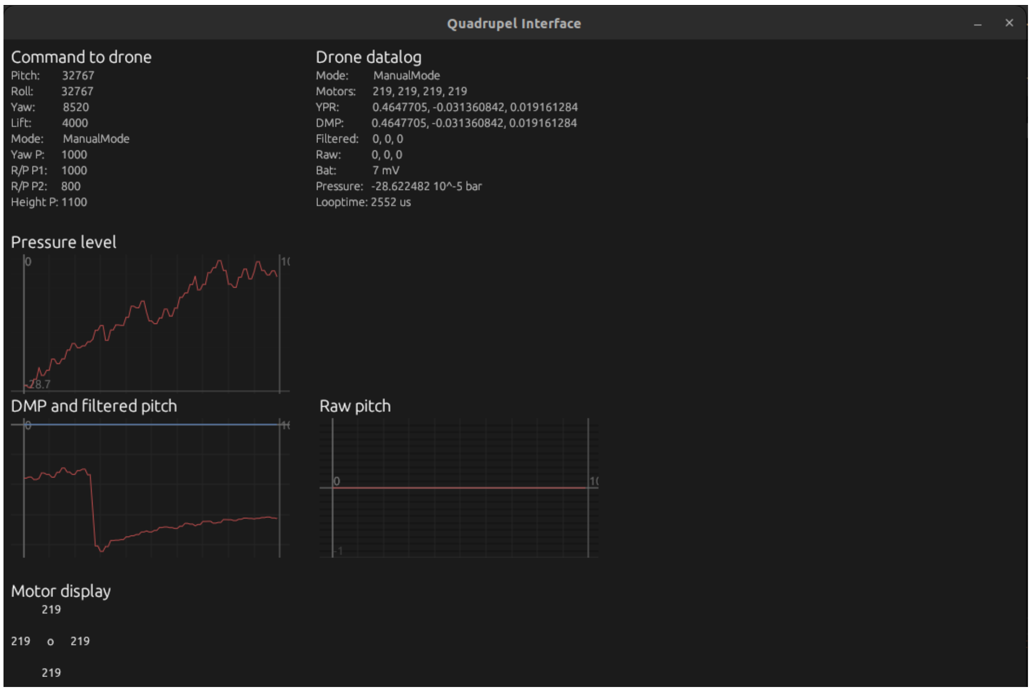
All joystick or keyboard inputs are normalized and applied in the above formula. The normalized results have different physical meanings in different modes, which will be introduced in the next section. As you can see, the variables used to achieve different actions have different weights in the formula. The part used to achieve lift accounts for 80% of the motor speed assign, because most of the motor speed is used to generate upward thrust against gravity.

Data Logging

The commands from the PC to the drone are stored as JSON files with the name of the real-time timestamp, such that the data can be examined later if something goes wrong with the drone. On the drone side, the data is stored in the flash memory of the board. When the storage is full, the data is dumped and sent to the PC. And the flash memory is erased to store new data.

GUI

The commands that are being sent from PC to drone and the datalogs sent by the drone to PC are displayed in a GUI to show the data to the user. A screenshot of the GUI is shown in Figure 3. There are three plots in the GUI. The first plot 'Pressure level' shows the height of the drone, and the other two plots show the comparison of Kalman filtered values with DMP output and raw sensor data. For the comparison, the pitch angles of the drone is used.



```
Quadrupel Interface

Command to drone            Drone datalog
Pitch:     32767            Mode:      ManualMode
Roll:      32767            Motors:    219, 219, 219, 219
Yaw:       8520             YPR:       0.4647705, -0.031360842, 0.019161284
Lift:      4000             DMP:       0.4647705, -0.031360842, 0.019161284
Mode:      ManualMode       Filtered:  0, 0, 0
Yaw P:     1000             Raw:       0, 0, 0
R/P P1:    1000             Bat:       7 mV
R/P P2:    800              Pressure:  -28.622482 10^-5 bar
Height P:  1100             Looptime:  2552 us

Motor display
      219
219    o    219
      219
```

Figure 3: GUI to display command to drone and datalog data.

User input normalization and PID control

The user input normalization is done to unify the input of the joystick and keyboard into four variables: yaw, pitch, roll, lift, with values ranging from -1 to 1, and assign different physical meanings according to subsequent calculation needs. The advantage of this is that it can facilitate subsequent code implementations in different modes. Besides, it is unnecessary to consider the input's physical meaning when assigning the motor speed. Although the floating point operation will reduce the overall operating efficiency, the results show that the system idle time is about 5ms in the fixed time cycle, so the impact can be ignored. Normalized results have different physical meanings in different modes, as shown in Table 1.

In addition to the user input, the sensor output will be normalized, and finally the result obtained by making a difference with the user input will be the input into the PID controller. An input of 1 means the position state of the system has changed significantly, whereas 0 means it does not change compared with the last loop cycle. Sometimes the sensor output value is greater than the maximum value of the physical meaning we give to the normalized result, resulting in an input to the PID controller with an absolute value greater than 1, which means that the system needs to be adjusted a lot. But due to clipping, this has no effect on the maximum value of the output.

	yaw	**pitch**	**roll**	**lift**
manual	*	*	*	*
yaw_control	rate: [0,360] deg/s	*	*	*
full_control / raw	rate: [0,360] deg/s	angles: [0,30] deg	angles: [0,30] deg	*
height_control	rate: [0,360] deg/s	angles: [0,30] deg	angles: [0,30] deg	height [0,20] m

Table 1: Physical meaning of normalized results in different modes. * means the value is only used as motor speed scale.

The PID controller used is an incremental PID, instead of a traditional positional PID, shown in Equation 1. Compared with the positional PID, the output of the incremental PID is the amount of change that should be made in each cycle rather than the state that the system should be in. In other words, the superposition of the output of the incremental PID is the positional PID output.

$$K_p \cdot (e(k) - e(k-1)) + K_i \cdot e(k) + K_d \cdot (e(k) - 2 \cdot e(k-1) + e(k-2)) \tag{1}$$

The biggest advantage of using incremental PID is that it can reduce calculation when using an I controller. Positional PID needs to superimpose the past error input in each cycle in order to calculate the integral. However, incremental PID only needs the current input. The I controller is implemented in full control mode, which saves a lot of computing time. In addition, when performing double-loop control on pitch and roll, we adjusted the inner-loop PID controller to the feedback path to facilitate subsequent parameter adjustments.

Filters

In this section, we will discuss how we used filters to process data from the drone's sensors to obtain reliable measurements of the drone's orientation and altitude. Specifically, we used a Kalman filter, which is a recursive algorithm that combines a predicted state of the system with measurements from the sensors to obtain an improved estimate of the system's state.

Angle filtering

The Kalman filter fuses data from the gyroscope and accelerometer to obtain a more accurate estimate of the drone's angle. The gyroscope measures the angular rate, while the accelerometer measures the gravitational force acting on the drone. However, relying only on the gyroscope can cause drift due to small errors, while the accelerometer cannot measure the rotation rate.

The Kalman filter solves this problem by combining the measurements from both sensors in a way that takes advantage of their strengths. It models the system as a set of equations and uses Bayesian statistics to estimate the most likely state of the system given the measurements.

The Kalman filter predicts the current state of the system based on the previous state and the control inputs (the gyroscope readings). It then updates the state estimate using the measurements from the accelerometer. The filter uses process noise and measurement noise parameters to account for the errors in the measurements and the model.

Mathematically, the Kalman filter equations for angle filtering can be found here [1].

Altitude filtering

Altitude filtering estimates the altitude of a vehicle using data from sensors. For example, a barometer measures air pressure to estimate altitude, while an accelerometer provides data on the drone's acceleration, which can be used to calculate velocity and altitude. One approach to altitude filtering is to use a Kalman filter, which estimates altitude based on measurements of altitude and velocity, using a state-space model and matrices to represent the system and its uncertainty. During the prediction step, the Kalman filter estimates the next state based on the current state and control input, and computes the predicted state covariance matrix. During the update step, the Kalman filter computes the Kalman gain, updates the state estimates and covariance matrix based on the measured altitude, and provides an estimate of the vehicle's altitude that accounts for measurement noise and system dynamics.

Recalibration of raw sensor mode and height control mode

For the raw sensor mode and height control mode, every time when drone goes into these two modes, the drone executes self-calibration for the first 5 seconds.

Final Results

Checklist

In Table 2 a checklist of the finished and unfinished components of the system is shown. All components have been completed, except for the wireless mode.

Basic Part				**Bonus Part**	
Functions	State	Functions	State	Functions	State
Protocol	Y	Panic Mode	Y	Kalman Filter & Plot	Y
Joystick & Keyboard Input	Y	Manual Mode	Y	Raw Sensor Mode	Y
User Interface	Y	Calibration Mode	Y	Height Control Mode	Y
Datalog	Y	Yaw Control Mode	Y	Fancy GUI	Y
Safe Mode	Y	Full Control Mode	Y	Wireless Mode	N

Table 2: Checklist of finished components (Y = finished, N = unfinished).

Size of the Rust code

The size of the compiled Rust code being uploaded to the drone is 62.52kb when compiling in development mode (using the command **cargo run**). When the Rust code is compiled in release mode (using the command **cargo run –release**), the size is 46.19kb.
The Rust code consists of 29 Rust files, including 224 functions, with a total of 3573 lines.

Control speed of the system

Control frequency

The control frequency of the drone control loop has been set to 100 Hz. For raw mode the drone control loop has been tested at 200 Hz, this gave more accurate results from the Kalman filter. The computation time of one iteration of the control loop in raw mode is just under 5 ms, giving the opportunity to run the loop at 200 Hz.

Average loop time of drone control modes

In Table 3 the average loop time of the drone control loop in various control modes is shown.

Control mode	**Average Loop Time (ms)**
manual_mode	4.267
yaw_control_mode	4.695
full_control_mode	5.872
raw_mode	4.581
height_control_mode	5.781

Table 3: Average loop time of the drone control loop in various control modes.

Latency of various blocks within the drone control loop

The latency of various blocks in the drone control loop has been measured, the results are shown in Table 4.

Blocks in drone control loop	Latency (ms)
Battery check	0.025
Read packet sent over serial	0.065
Check USB connection	0.021
Motor PWM calculation manual mode	0.091
Motor PWM calculation yaw control mode	0.537
Motor PWM calculation full control mode	1.521
Motor PWM calculation raw sensor mode	1.341
Motor PWM calculation height control mode	1.605
Read DMP	2.260
Raw sensor data to angles calculation	1.297
Kalman filter	1.393
Create datalog	0.034
Write serial data	0.023

Table 4: Latency of various blocks within the drone control loop.

Conclusion

In this project, all the safety features for the drone were first implemented and then all the other control modes were designed. The five modes that have been demonstrated are - manual mode, yaw control mode, full control mode, raw mode, and height control mode. All the team members had the opportunity to work on the embedded systems project for the first time. We learned how to combine general coding with the specific constants and requirements of the hardware. Individual contributions have been shown in the table below. Overall, this was a great learning curve for all of us, which will definitely be helpful for our future endeavors.

Individual contributions of each team member

In Table 5 the individual contributions of each team member to the system are shown.

	Jelle	**Jiacong**	**Sukanya**	**Marijn**
Joystick input	X			X
Keyboard input	X			X
Serial protocol	X			X
Serial transmission pc	X			X
PC Interface			X	X
GUI			X	X
Database	X		X	X
Drone control loop		X		X
Serial transmission drone				X
Calibration mode		X		
Full control mode		X		
Height control mode		X		
Manual mode		X		X
Panic mode		X		
Raw sensor mode	X	X		
Yaw control mode		X		
Kalman filter	X			
PID controllers	X	X		
Drone struct	X	X		

Table 5: Individual contributions

Bibliography

[1] “A practical approach to kalman filter and how to implement it,” Sep 2012.

Appendix A

The following table shows the LEDs that should be on in the corresponding modes

	Yellow	Green	Red	Blue
safe_mode	on	off	off	blink
panic_mode	off	off	on	
calibration_mode	off	off	off	
manual_mode	on	on	off	
yaw_control_mode	off	on	on	
full_control_mode	off	on	off	
raw_mode	on	off	on	
height_control_mode	on	on	on	

Appendix B

The following table shows the paths for corresponding function

Function	Path
Protocol	dronecode/src/drone_transmission.rs runner/src/interface/pc_pransimission.rs protocol/src/lib.rs
User Input	runner/src/interface/joystik_mapper.rs runner/src/interface/keyboard_mapper.rs runner/src/interface/setting_logic.rs
User Interface	runner/src/interface/interface.rs
Control Loop	dronecode/src/control.rs
Motor Assign	dronecode/src/working_mode/motors.rs
Datalog	dronecode/src/working_mode/log_storage_manager.rs runner/src/interface/database.rs
Safe Mode	dronecode/src/working_mode/safe_mode.rs
Panic Mode	dronecode/src/working_mode/panic_mode.rs
Manual Mode	dronecode/src/working_mode/manual_mode.rs
Calibration Mode	dronecode/src/working_mode/calibration_mode.rs
Yaw Control Mode	dronecode/src/working_mode/yaw_control.rs
Full Control Mode	dronecode/src/working_mode/full_control.rs
Kalman Filter	dronecode/src/working_mode/kalman.rs
Raw Sensor Mode	dronecode/src/working_mode/raw_sensor_mode.rs
Height Control Mode	dronecode/src/working_mode/height_control_mode.rs
GUI	runner/src/interface/gui.rs runner/src/interface/plotter_pistion.rs