Engineering 100-Quadcopter Lab 2: Introduction to Autonomous Flight Control

In the previous lab you learned about the components in a quadcopter, and were introduced to the software used for flight control. Now you will use that knowledge and lecture material to control and optimize the quadcopter for take-off, maintaining a constant altitude, and landing. In addition, you will

- learn about different sensors used for estimating the altitude of the quadcopter,
- use your knowledge of control techniques to optimize controller parameters for a smooth and rapid take-off and landing, and
- export logs from your quadcopter to the ground control software for further processing.

1 Background Material

To stay at a target altitude, the quadcopter must measure its distance from the ground, move to the desired altitude, and remain there. There is a barometer built into the board, which is commonly used to measure altitude. Alternatively, we have learned in Lab 1 how an ultrasonic sensor can be used to measure distance from the ground. We will use each of these sensors to control altitude. First, we will cover several aspects of flight control.

1.1 PID control and how it is used for altitude control

As explained in lecture, a closed-loop controller generally takes as input the error (e) between the target state of a system (also called *Plant*) and the measured state, and generates as output a control signal ($u(e)$) that is a function of this error. This is done continuously or at discrete times; as new measurements are taken, new control values are generated in response. This allows the controller to adapt to the state of the system. This feedback control process is illustrated in Figure 1.

A PID controller generates the control signal $u(t)$ using $e(t)$:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}. \tag{1}$$

As you can see, the control signal consists of three components: the proportional, integral and derivative terms, summarized as follows.

- The proportional (P) term increases with error. It has the effect of *pushing* hard when the error is large but at the risk of overcorrecting the error. The P controller usually works well in most situations. But in certain circumstances, for example a sudden increase of the wind against the quadcopter, the final target can never be reached. Increasing the control gain K_p tends to reduce but not eliminate the steady-state error, which is the eventual gap between

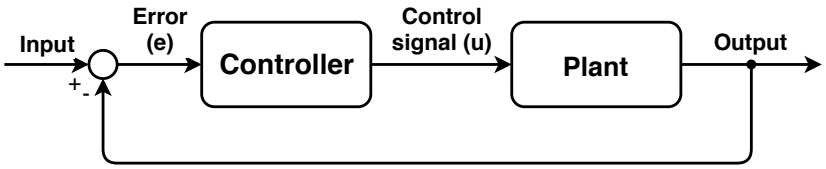

Figure 1: Feedback control process.

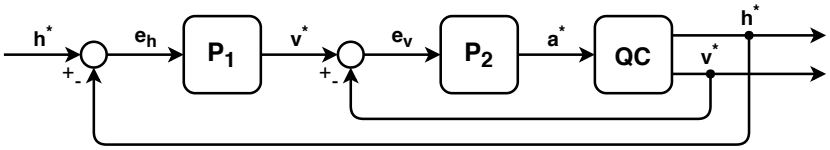

Figure 2: Two P-controllers in series.

the state of the system at a distant time in the future (i.e., steady-state), and the target state. With P, alone, the system may also oscillate or over-react to transient changes that might be caused by sensor errors.

- The integral (I) term accounts for errors that accumulated in the past. Persistent errors can eventually build up through the integrator to produce large signals, forcing the controller to exert more control to reduce the error. As a result, the integral controller can help reduce the stead-state error that is likely to occur with the P controller alone. This term also tends to make the system more *sluggish*. In some circumstances, it also produced oscillations. Once large, this term can take a long time to *unwind*, i.e., return to zero.

- The derivative (D) term is proportional to the error's rate of change. It can be used to *anticipate* future error, for if the error is increasing fast, even if the error magnitude remains very small, one can anticipate a large error in the future. This term tends to dampen the system and decrease overshoot. On the other hand, it has no effect on the steady-state error; this is because persistent but constant error has zero rate of change.

The quality of a PID controller hinges on the right combination of the three gain parameters for a particular control task. In this lab you will have the opportunity to tune and experiment with some parameter choices and observe their effect on the altitude control of the quadcopter.

The controller found in ArduPilot for controlling the quadcopter's altitude is illustrated in Figure 2.

The first of the two, P_1, is a proportional controller that converts the error in altitude/height (the difference between the target altitude/height h^* and the measured height h) into a control signal for target velocity (v^*). The second controller, P_2, then uses this target velocity and the measured velocity to generate a proportional control signal for target acceleration (v^*). Integrating acceleration (e.g., accelerometer readings) over time is one way to measure velocity. Integrating velocity over time yields the change in altitude. We can also measure the (change in) altitude using the built-in barometer or the external ultrasonic sensor. In practice, ArduPilot combines all available measurements to obtain robust estimates, and uses them to drive the depicted controllers.

Now we explain how the target acceleration is used to produce motor outputs as shown in Figure 3. A PI controller uses the error between the target acceleration and measured acceleration

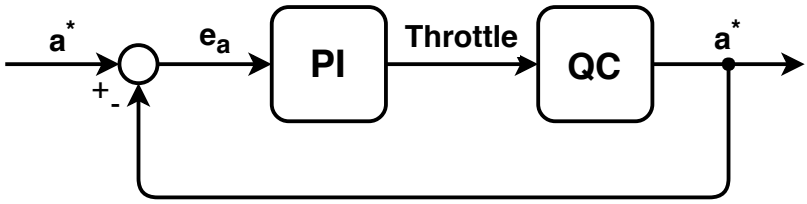

Figure 3: PI controller.

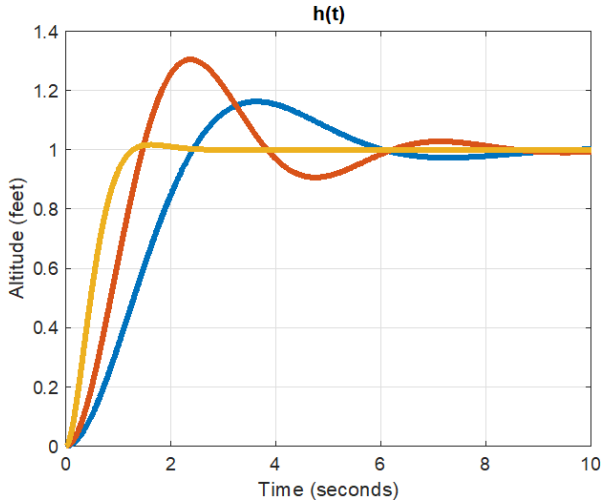

Figure 4: Altitude control timeseries.

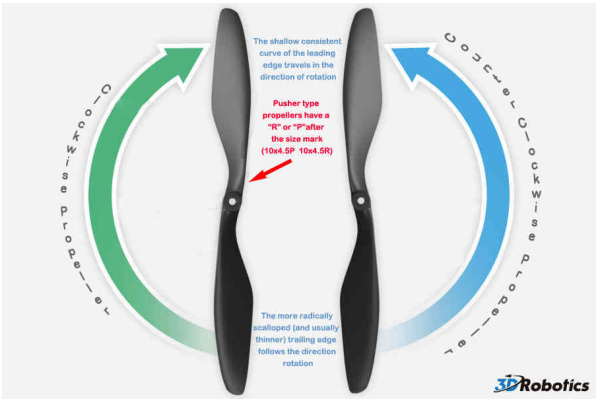

Figure 5: Propeller rotation directions. Image credit: `http://ardupilot.org/copter/docs/connect-escs-and-motors.html`.

(i.e., rotated accelerometer readings, according to the pitch and roll of the aircraft) to control the motor speed, and hence the throttle.

Figure 4 shows simulated examples of the described control mechanism being used to control altutude with a target of one foot. The parameters used in the simulation have been intentionally omitted. In this set of lab exercises we will leave the P_2 controller intact but you will be asked to tune both P_1 and PI and to observe their impact on the quadcopter's ability to take off, land, and hold a steady altitude.

1.2 Propellers

A propeller converts rotational motion into thrust. A quadcopter relies on four propellers to provide it with upward thrust. However, to prevent the entire frame from spinning out of control, two of the propellers rotate in the opposite direction from others, resulting in a controllable angular velocity that is often held at zero. You should know the direction of each rotor in your quadcopter from the previous lab. Note that clockwise propellers are also different from counter-clockwise propellers. A propeller spinning against its intended direction will result in downward thrust, crashing the quadcopter into the ground. Clockwise propellers are often called pushers, and are marked with a

R or P. You can also recognize the type of the propellers from its shape, as shown in Figure 5. The side with a consistent curve should be the leading edge when rotating in the correct direction, while the other thinner side is the trailing edge.

2 Pre-Lab (30 Points)

You will have two weeks to complete this lab. Note the due dates on Canvas; the pre-lab will be due by 11:59pm the day before the lab on the first week, and you will have one week following the second week to complete the in-lab and post-lab.

Question 2.1. (10 points) Describe two different methods for measuring vertical speed using the sensors on your quadcopter.

Question 2.2. (10 points) Download the lab files and find a MATLAB script that simulates the P1 and P2 controllers from the above background material, named `altitude_controller.m`. This program plots the response of the system when the target altitude is one foot. Set the proportional gains K_{p1} and K_{p2} to 2 and 1, respectively. Now run the script.[1] What is the peak altitude of the response? How long does it take for the response to reach steady-state (i.e., stop oscillating)? Submit an image of the response that this program generates.

Question 2.3. (10 points) The quadcopter has two types of propellers; one spins counter-clockwise and one spins clockwise. Please identify two traits that distinguish the two types.

3 In-Lab (40 Points)

In this lab, you will fly the quadcopter for the first time. To reduce the probability of injury and crashing into surrounding objects, you will attach (tether) the quadcopter to a 10-pound weight that restricts it to remain within a small (approximately three feet) radius of the weight. If the tether is damaged or untied, inform lab staff. Follow these safety instructions for the remainder of the lab.

- Wear eye protection for the entire duration of the lab.
- Only connect propellers when you are about to fly the quadcopter, and always remove them after your flight(s).
- Always connect the motor battery **after** attaching it to the weight. Make sure you are outside the quadcopter's range before initiating take-off, and while the motors are armed and spinning.
- After each landing, wait until the motors have been turned off, and then disconnect the motor battery; they should remain disconnected until the next flight.
- Make sure motors are not powered when the quadcopter is not tethered. If you need to test the motors untethered, e.g., the test procedure in Lab 2, remove propellers **before** attaching the motor battery.

Note that there are three precautions you can take to prevent an accident from happing: (1) disconnect the motor battery, (2) remove the propellers, and (3) tether the quadcopter. You must

[1] You can use the CAEN computer labs to run MATLAB code. Alternatively, you can install MATLAB on your own computer for free (`http://caenfaq.engin.umich.edu/software-for-students/matlab-for-students`), or use MATLAB Online (`https://www.mathworks.com/products/matlab-online.html`) after registering for an account.

always take at least one (two when possible) of these precautions, e.g., disconnect the battery **and** remove propellers when the quadcopter is sitting on your lab bench. You will also be given a separate flight checklist that you need to follow when flying in the lab from now on.

Before starting, connect Mission Planner to ArduPilot and load your parameter list from the end of the previous lab, or load the defaults given to you at the start of Lab 2. Check the messages tab to see if the copter has any errors. Anything that refers to GPS or Compass can be ignored, but errors such as 'accel calibration needed' should be recognized and resolved. It is a good practice to calibrate the accelerometer on every start up. If during flight, the HUD does not show a near-zero pitch and roll, then the 'Calibrate Level' option on the accel calibration window can be used as a simple solution to zero out the roll and pitch parameters.

Make sure that both batteries have sufficient voltage, and that they are strapped securely to the frame. If any of the batteries slips off during flight, it will most likely result in a crash, and may result in a fire and explosion. Also, make sure that the test leads from the batteries are tucked in so the props cannot hit them.

3.1 Part 1: Altitude holding using a barometer

Turn on the main board, and connect Mission Planner to ArduPilot. First make sure your quadcopter is only using the barometer for measuring altitude by going to `Initial Setup` → `Optional Hardware` → `Range Finder`, and selecting `None` from the dropdown menu. Reset the software (by pressing the `MOD` button on the main board) for the changes to take effect, and reconnect Mission Planner to ArduPilot.

Go to the `Quick` tab in the `Flight Data` screen and examine the barometer readings. If you do not see this reading in Mission Planner, you can double-click on any of the displayed measurements, and select `alt` to show the barometer reading in the selected slot. Also select and display `verticalspeed`, the velocity computed from barometer readings. Look at how the readings change over time. Lift the quadcopter from its initial position, and examine whether the barometer can detect changes in altitude. Note that the barometer is calibrated every time the software is launched (i.e., when the main board is turned on, or restarted), to record the initial position of the quadcopter, corresponding to an altitude of zero. Examine how readings change over a few minutes.

Question 3.1. (5 points) Comment on the accuracy of barometer readings for estimating absolute altitude, and detecting changes in altitude (i.e., vertical velocity). Describe the errors for each case, as well as whether the measurement can be used reliably for flight control indoors.

Attach your quadcopter to the provided weight, and place it on the ground. You may use the carabiner, but please have lab staff tie monofilament[2]. Check the flying range of the quadcopter by manually moving it in a circle, with the attached line completely extended, and ensure that it cannot hit any surrounding objects. Connect the 3-cell battery to power the motors. You should hear a 4-beep tone indicating that the motors have been armed. Move out of the quadcopter's flying radius. Your instructor will then help you setup a joystick that can be used to land the quadcopter or disarm the motors as a safety measure.

Checkpoint 1. (3 points) Ask your instructor to help you setup a joystick on Mission Planner and double check your setup before proceeding.

Go to `Flight Data` → `Scripts`. Click `Select Script` and find the file named `AltHold.py` in the lab files. This program allows you to test the quadcopter's ability to take-off and maintain a certain altitude. Click `Run Script` to initiate the script. You can now issue the following commands.

- **Takeoff**: The motors will start spinning. After a 5-second pause, the quadcopter will start ascending to 30 cm.
- **Land**: The quadcopter will land and then disarm the motors.
- **Panic Button**: Pressing this button, or the Esc key on your keyboard will immediately disarm (turn off) the motors. Only use this option as a last resort, e.g., if a crash is inevitable; it can damage the quadcopter.

Note that you can also trigger the last two commands from a joystick. Next, identify the correct propeller type for each motor using what you learned in the pre-lab, and in the previous lab. Motors with silver nuts are counter-clockwise motors, while those with black nuts rotate clockwise. Note that the motors use self-tightening nuts for holding the propellers. The threads on each motor are configured such that holding the nut still, while rotating the propeller in the correct direction, must result in fastening the nut, hence the name self-tightening. This prevents the nut from loosening over time, which might cause a crash if it comes off during flight. However, even with self-tightening nuts, there is a risk of propellers detaching if they are not first manually tightened. Put the propellers on your quadcopter, and securely fasten the propeller nuts using the available wrenches. Secure any loose wires that might get caught in spinning propellers.

Initiate take-off and observe the quadcopter's response for up to a minute, and then tell it to land. Monitor the barometer readings in this interval. Note that the quadcopter may drift on the tether, because nothing else is controlling its position. The accelerometer will try to keep the board level, but vibrations can lead to errors, which in turn make the quadcopter move in random directions. Additionally, slight differences between the angle of the board and the frame might cause it to drift in a certain direction. Land the quadcopter if it starts acting erratically at any point. Report your observations below.

[2]Many knots that work fine in string will slip in monofilament.

Question 3.2. (5 points) Describe the quadcopter's motion upon take-off. Did it take off from the ground? If it did, did it stay at a fixed altitude? If it did, was that fixed altitude the pre-configured 30 cm?

3.2 Part 2: Altitude holding using an ultrasonic sensor

We will next try altitude holding using an external, downward facing ultrasonic sensor. Go to `Initial Setup` → `Optional Hardware` → `Range Finder` and select `Maxbotix I2C` from the drop-down menu. Restart ArduPilot (by pressing the `MOD` button on the main board) and reconnect Mission Planner to ArduPilot. In the `Quick` tab under the `Flight Data` screen click on the barometer reading and select `sonarrange`. You should be able to see the sonar reading on the control panel. Try moving/lifting the quadcopter with your hand and you should see the readings change.

Now use the provided script to fly the quadcopter again, and compare its ability to maintain a constant height with the previous exercise. Fly the quadcopter for up to a minute, and report your findings below.

Question 3.3. (5 points) Describe the quadcopter's motion. Did it take off? If so, did it stay at a fixed altitude? If it did, was it the specified altitude of 30 cm?

Next you will examine the flight data logs to inspect the response of the quadcopter. Under the `Flight Data` screen, go to `DataFlash Logs` and click `Download DataFlash Log`. Select and download the latest log from the main board. Close the window and click on `Review a Log`. Select the file you just downloaded. You can plot and view the data collected by ArduPilot during flight. Open the group called `CTUN` and select the ultrasound sensor readings (`SAlt`), and the desired altitude (`DSAlt`). Also compare these readings to the barometer (`Alt`). You can also explore other available measurements, such as IMU readings.

To save separate log files, you should `MOD` the copter between tests. This will be very helpful in part 3 of this lab as well as in Lab 4. Alternatively you could have one large continuous log file, but it can become difficult to navigate the output.

Checkpoint 2. (3 points) Show the collected flight data logs to your instructor.

Question 3.4. (5 points) Look at the ultrasound readings and the target altitude. How long does it take for the quadcopter to reach the target altitude, once the take-off command has been issued (i.e., `DSAlt` is set to 30 cm)? The x-axis for the logs is samples, and the sampling rate is 200 samples per second. The version of Mission Planner we are using has bugs related to changing the time axis. The most straightforward way to complete the task is to leave the display as samples and covert using the sampling rate yourself.

3.3 Part 3: Tuning the altitude controller

In this part we will experiment with different controller gain parameters. We will not change the gain in the P_2 controller, but will modify both P_1 and PI. This means there are three parameters for you to tune: the proportional gain K_{p1} in P_1, and the proportional gain K_p and the integral gain K_i in the PI controller. In Mission Planner, go to `Config/Tuning` → `Full Parameter List`. The gains can be found under `POS_Z_P` (K_{p1}), `ACCEL_Z_P` (K_p), and `ACCEL_Z_I` (K_i). Make sure that you start with the following parameters: $K_{p1} = 1.0$, $K_p = 0.5$, $K_i = 0.5$. Also ensure that the `RNGFND_GAIN` parameter is set to 1.0 under the `Full Parameter List` tab.

We will first focus on the PI controller. You might have noticed that the controller is sluggish. This is due to a sub-optimal ratio between K_i and K_p. Tune this ratio by changing K_i until you are satisfied with the response of your quadcopter. Setting this parameter too high will result in oscillation. After each flight, you can download the flight logs to compare the results with those of a previous flight. It is probably a good idea to reset the software before each flight, so that logs are written in a new slot. After tuning K_i, try increasing (or decreasing) K_p and K_i together, while keeping their ratio constant, and determine the appropriate gains for this controller. Note that the maximum allowed values for K_p and K_i are 1.5 and 3, respectively.

Question 3.5. (6 points) Report your optimized parameters for the PI controller.

Your next task is to find the optimal K_{p1}. You will do this using simulations generated by the MATLAB script you used in the pre-lab. Open the script and set K_{p2} to the correct value from the `Full Parameter List` tab in Mission Planner, under the `VEL_Z_P` parameter. Find the highest gain for K_{p1} that does not result in oscillation. About a 2% overshoot is alright.

Now open the full parameter list in Mission Planner, and search for `POS_Z_P` and `RNGFND_GAIN`. To correctly modify P_1, you must set both to this K_{p1} value. Take off with the tuned parameter you found in MATLAB. Further increase or decrease the gain in small steps to find the highest value that does not result in oscillation.

Checkpoint 3. (4 points) Show your instructor how you can achieve a fast and smooth take-off with your optimized parameters.

Question 3.6. (4 points) Report the value you computed from MATLAB, and the final value for K_{p1}. How long does it take for the quadcopter to reach its target altitude with the tuned parameters?

Before you leave, download the flight log corresponding to take-off with your tuned parameters. You will need this file for completing the post-lab. You should also save your params file as you did at the end of lab 2.

4 Remote Student Recommendations

All team members, including remote students, should actively participate in the lab. This section recommends tasks that might be most practical for remote participants to lead. These are recommendations: you are permitted to divide work in other ways.

- Remote students should be familiar with all safety precautions, especially when it comes to flying the quadcopter. Be sure that every member is acting safely and responsibly.
- Subsection 3.1 and 3.2: These sections are heavily focus on interfacing with and observing the quadcopter. Be sure to encourage your team to have their screen shared during the work with mission planner. During flights, insist to have the camera pointed at the quadcopter to be able to see and contribute to the questions about its response to changes.
- Subsection 3.3: The controller tuning portion of this lab has the potential for participation with the MATLAB sections. MATLAB simulations, similar to what was needed in the Pre-Lab, needs to be done here. Play an active role in using these simulations to find optimum values. You can share your screen with your team to show them the graphs that back up any values you claim will work well.

5 Post-Lab (30 points)

Question 5.1. (6 points) Submit a screenshot of a plot showing how the quadcopter reaches the target altitude, using the log file you collected at the end of the lab. Also submit an image of a MATLAB simulation using the same controller gains for P1 and P2 as ArduPilot. Comment on the similarities and differences between the simulation and actual flight.

Question 5.2. (6 points) Briefly explain what overshoot and steady-state error mean.

Question 5.3. (6 points) Describe a situation where a barometer cannot provide an accurate estimate of the altitude, and one where it must be chosen over a downward facing proximity (ultrasonic or LIDAR) sensor. You can use both indoor or outdoor situations.

Question 5.4. (6 points) You have learned about P, PI, and PID controllers in lectures. We have also discussed the usage of different controllers in ArduPilot above. Why do you think an integral term is used for controlling the throttle of the quadcopter, but not for determining the target velocity and acceleration (P_1 and P_2)? Hint: Try to analyze what happens when the quadcopter has reached its target state (acceleration, velocity, or altitude), i.e., what happens when the error is zero for each of these controllers.

Question 5.5. (6 points) ArduPilot tries to keep the quadcopter level (according to readings from the accelerometer) when it is hovering. Assuming that our goal is to keep the quadcopter stationary (i.e., prevent it from drifting horizontally), is this approach an open-loop or closed-loop controller? Why? Comment on the effectiveness of this technique using what you observed in the lab. What sensor(s) should you add to the quadcopter, if any, to be able to implement a closed-loop controller for keeping the quadcopter stationary?