AE 483
UAV Navigation and Control
Lab Manual

Lab 3

Flight Control

3.1 Objectives

In this lab, you will write full 6 degree-of-freedom control policies which enable two basic maneuvers: hover and translation in a straight line at a fixed altitude. In particular, you will

- Simulate full attitude control in MATLAB, and implement your attitude controller on-board the vehicles in C
- Simulate hover control in MATLAB, and implement your hover controller on-board the vehicles in C
- Visualize and compare simulation and flight results.

3.2 Tasks

3.2.1 Attitude Control

The goal in this task is to simulate full attitude control in MATLAB, and then implement this controller on-board the hummingbird vehicle in C. This will require us to write, in MATLAB, the full equations of motion which have been derived in lecture. After writing these equations of motion, you will write an independent LQR controller to regulate each of the rotational axes: roll, pitch, and yaw.

When your simulated controller produces stable results, you will implement the controller on-board the hummingbird vehicle. You will write your LQR controller in C, and test it's performance on the real vehicle during flight experiments.

3.2.2 Position Control

The goal in this task is to simulate position control in MATLAB, and then implement this controller for the hummingbird vehicle. Recall that the onboard computer runs it's control loop at 1000 Hz, and the motion capture system runs at 100 Hz. This motivates our decision to use a decomposed control strategy.

From the previous task, you have already written the core functionality making up the quadrotor simulator in MATLAB, the equations of motion, and an inner loop attitude controller. In this task, you will add to this a simple *outer loop* controller, a controller that uses 3D position information from the motion capture system and generates desired angles and thrust to be sent to the inner loop controller on the vehicle. Also, just as before, we will develop the outer loop controller in MATLAB, and when we are satisfied with the behavior in simulation, we will move to integrate the controller with the hummingbird vehicle to perform full flight tests.

3.3 Procedure

3.3.1 Simulate Attitude Control

1. Download `Lab3_codes.zip` from the course website. This zip archive contains a folder called `QuadrotorSim` containing MATLAB programs which provide the infrastructure of a simulator for the full 6-DOF quadrotor model you derived in lecture and homework. Extract this entire folder to a location on your `U:` drive. In particular, the script `simulate_this` will run the simulator. Your first task is to define the equations of motion in the function `Quad_EOM`.

2. Once you have equations of motion, the next step is to create your attitude controller. To do this, you will modify the template code provided in `Inner.m`. In designing your attitude controller, recall Lab 2 where we linearized the equations of motion about a nominal attitude. We will repeat this process from Lab 2, and instead of focusing only on the pitch axis, we will consider each of the roll, pitch, and yaw axes. After linearizing, and considering each axis independently, we can write down the following collection of state and control vectors, and linearized state-space equations of motion:

$$x_{roll} = \begin{bmatrix} \theta_3 \\ \dot{\theta}_3 \end{bmatrix} \qquad u_{roll} = \begin{bmatrix} u_1 \end{bmatrix}$$

$$\dot{x}_{roll} = A_{c,roll} x_{roll} + B_{c,roll} u_{roll}$$

$$x_{pitch} = \begin{bmatrix} \theta_2 \\ \dot{\theta}_2 \end{bmatrix} \qquad u_{pitch} = \begin{bmatrix} u_2 \end{bmatrix}$$

$$\dot{x}_{pitch} = A_{c,pitch} x_{pitch} + B_{c,pitch} u_{pitch}$$

$$x_{yaw} = \begin{bmatrix} \theta_1 \\ \dot{\theta}_1 \end{bmatrix} \qquad u_{yaw} = \begin{bmatrix} u_3 \end{bmatrix}$$

$$\dot{x}_{yaw} = A_{c,yaw} x_{yaw} + B_{c,yaw} u_{yaw}$$

where $A_{c,i}$ and $B_{c,i}$ are the continuous time linearized equation of motion matrices for the i-th rotational degree of freedom.
Now, using these linearized equations, create a discrete-time Linear-Quadratic Regulator (LQR) that drives each axis from arbitrary initial conditions to $(\theta_i, \dot{\theta}_i) = (0, 0)$, i.e. regulate each rotation axis to zero. To do this, modify the template code in `Inner.m`. Assume initial condition (set in `simulate_this`)

$$x^i(0) = \begin{bmatrix} \pi/6 \\ 0 \end{bmatrix}.$$

To create these controllers, use the MATLAB `dlqr` command. Recall that the LQR controller has the form

$$u = -Kx$$

```
[K,S,e] = dlqr(A,B,Q,R)
```

where A and B are from the **time-discretized** linearized equations of motion, Q and R define the quadratic cost function.

3. In `Inner.m` create a discrete-time LQR controller that drives the vehicle to a non-zero attitude, i.e. non-zero setpoint regulation, where the desired state of the vehicle is given by

$$\begin{bmatrix} \theta_1 \\ \dot{\theta}_1 \end{bmatrix} = \begin{bmatrix} a \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} \theta_2 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} \theta_3 \\ \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} c \\ 0 \end{bmatrix}$$

for some $a, b, c > 0$. Assume initial conditions (set in `simulate_this`)

$$\begin{bmatrix} \theta_1(0) \\ \dot{\theta}_1(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} \theta_2(0) \\ \dot{\theta}_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} \theta_3(0) \\ \dot{\theta}_3(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

3.3.2 Simulate Position Control

1. Create your outer loop position controller in MATLAB. To do this, you will modify the template code provided in `Outer.m`. To design the outer loop controller, we will linearize the equations of motion about hover, where the vehicles position, angles, velocities, and inputs are zero, except for the thrust force which is equal to the force required to counteract gravity. After linearizing, we had the following system of linearized equations of motion:

$$\begin{aligned} \delta\ddot{o}_1 &= -g\delta\theta_2 \\ \delta\ddot{\theta}_2 &= \frac{1}{J_2}\delta u_2 \end{aligned}$$

$$\begin{aligned} \delta\ddot{o}_2 &= g\delta\theta_3 \\ \delta\ddot{\theta}_3 &= \frac{1}{J_1}\delta u_1 \end{aligned}$$

$$\begin{aligned} \delta\ddot{o}_3 &= -\frac{1}{m}\delta u_4 \\ \delta\ddot{\theta}_1 &= \frac{1}{J_3}\delta u_3 \end{aligned}$$

At this point, we can see that altitude control (Z-control) and yaw control are already completely independent of the rest of the state, i.e. their equations of motion do not include any other states. This is not the case for horizontal position control (X-Y-control). For X-Y control, we will do the following: assume that $\delta\theta_2$ and $\delta\theta_3$ can be instantly achieved. This is motivated by the inner loop attitude controller which runs at 1000 Hz, which does rapidly regulate these angles to desired values. Given this assumption, we can then notice that we can achieve desired translational *acceleration* simply by choosing $\delta\theta_2$ and $\delta\theta_3$. In other words, we now have four completely decoupled sets of linearized equations of motion:

$$\begin{aligned} \delta\ddot{o}_1 &= -g\delta\theta_2 \\ \delta\ddot{o}_2 &= g\delta\theta_3 \\ \delta\ddot{o}_3 &= -\frac{1}{m}\delta u_4 \\ \delta\ddot{\theta}_1 &= \frac{1}{J_3}\delta u_3 \end{aligned}$$

Note that the yaw controller is already implemented in the onboard, inner loop, controller. Note that none of the translational equations depend on yaw, thus we will simply hold a yaw angle of zero, i.e. $\delta\theta_{1_{des}} = 0$. The remaining equations, the systems which govern translation in the x, y, and z directions can now be treated just as we are used to from the inner loop. Just as before, we will create three sets of state space equations, and create LQR controllers for these systems which drive the positions $\delta o_1, \delta o_2, \delta o_3$ to zero.

$$x_{o_1} = \begin{bmatrix} \delta o_1 \\ \delta \dot{o}_1 \end{bmatrix} \qquad u_{o_1} = \begin{bmatrix} \delta \theta_2 \end{bmatrix}$$

$$\dot{x}_{o_1} = A_{c,o_1} x_{o_1} + B_{c,o_1} u_{o_1}$$

$$x_{o_2} = \begin{bmatrix} \delta o_2 \\ \delta \dot{o}_2 \end{bmatrix} \qquad u_{o_2} = \begin{bmatrix} \delta \theta_3 \end{bmatrix}$$

$$\dot{x}_{o_2} = A_{c,o_2} x_{o_2} + B_{c,o_2} u_{o_2}$$

$$x_{o_3} = \begin{bmatrix} \delta o_3 \\ \delta \dot{o}_3 \end{bmatrix} \qquad u_{o_3} = \begin{bmatrix} \delta u_4 \end{bmatrix}$$

$$\dot{x}_{o_3} = A_{c,o_3} x_{o_3} + B_{c,o_3} u_{o_3}$$

To create these controllers, use the MATLAB `dlqr` command. Recall that the LQR controller has the form

$$u = -Kx$$

and note that the `dlqr` command has the following calling syntax

```
[K,S,e] = dlqr(A,B,Q,R)
```

where A and B are from the **time-discretized** linearized equations of motion, Q and R define the quadratic cost function.

3.3.3 C code: AscTec SDK

In this section, you will proceed to write the C code, that you will flash on to the quadrotor.

1. Open the AscTec Eclipse application on your computer. For your workspace, select `workspace_lab3` from `Lab3_codes.zip` that you downloaded from Piazza. You will add our Inner and Outer loop control in the `lab3()` function in `lab.c`.
2. Fill in the gains for the Outer Loop. Use the gains from your simulation.
3. Use the position errors and velocities to compute the desired angles and thrust. Keep in mind that we will include the effects of yaw in the real system, and hence the desired roll and pitch will also depend on yaw.
4. Include offsets in the desired roll and pitch angles to account for biases in the onboard IMU. You will tune these offsets later during flight.
5. Fill in the gains for the Inner Loop. Again, use the gains from your simulation.

6. Use the attitude errors and attitude rates to compute the inner loop control output.
7. Configure the build targets. Select "Project → Clean Project". Make sure there are no errors at the Console window in Eclipse. If there were no errors, you should see a recently created main.hex file in your workspace folder.
8. Show the code to your TA.
9. Flash the code on to the quadrotor by following the instructions mentioned in Lab 1 manual.

3.3.4 C code: Ground Station

1. Open the solution file (.sln file) from `AE483GroundStation_Lab3` in **Visual Studio 2010**.
2. In `main.c`, change the IP address to the PC IP at your work station. Refer to Lab 1 manual for instructions on changing the IP address.
3. Place the readme.txt file from `Lab3_codes.zip` in your local directory (U drive).
4. Show the code to your TA and then proceed to flight.

3.4 Checkoff Points

In lab, show the TA your:

1. `QuadrotorSim` files `Quad_EOM`, `Inner.m` and `Outer.m`.
2. Onboard C code, `lab.c`, which regulates 3-DOF attitude (roll, pitch, and yaw).

3.5 Report

Write a lab report which answers the following problems.

Problem 3.1. Run your MATLAB simulation with an initial offset of [4, 4, 4] meters in your position states. Keep zero initial conditions for all other states. Submit plots of data from your MATLAB simulation.

Problem 3.2. Append your simulation codes and also the lines you added in the AscTec C code (**not** the entire code).

Problem 3.3. Explain the need for the variables *u1_offset* and *u2_offset*. Note down the values that you used.

Note: Your report should carefully describe anything that went “wrong” with the experiments, or any mismatch between the data and what you saw with your own eyes. It should be **written with your lab group**, and submitted by email to the **ae483.fall16+*TA-name*@gmail.com** with the subject line “Lab 3 Report” and the attached report should be named with the last names of all group members “*Lastname1-Lastname2-Lastname3*-Lab-3.pdf”.