Programming Exercise 2:
Quadrotor Control in the Plane

1 Introduction

In this exercise, you will be implementing the PD controller discussed in lecture to control the motion of the quadrotor in the Y-Z plane. Before starting on this programming exercise, we strongly recommend watching the video lectures, completing the review questions for the associated topics, and reading through this handout.

Just as you've done in the previous programming exercise, you will need to download the starter code and unzip its contents into the directory in which you wish to complete the assignment.

2 System Model

2.1 Coordinate Systems

The coordinate systems and free body diagram for the planar model of a quadrotor are shown in Fig. 1. The inertial frame, $\mathcal{A}$, is defined by the axes $\mathbf{a}_2$, and $\mathbf{a}_3$. The body frame, $\mathcal{B}$, is attached to the center of mass of the quadrotor with $\mathbf{b}_2$ coinciding with the preferred forward direction and $\mathbf{b}_3$ perpendicular to the rotors pointing vertically up (see Fig. 1).

2.2 Dynamics

For a quadrotor modeled in the $Y - Z$ plane, its orientation is defined by a roll angle, ϕ. It is assumed that its pitch and yaw angles are 0. You will need the rotation matrix for transforming components of vectors in $\mathcal{B}$ to components of vectors in $\mathcal{A}$:

$$^{\mathcal{A}}[R]_{\mathcal{B}} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}. \tag{1}$$

We will denote the components of angular velocity of the robot in the body frame by $\dot{\phi}$:

$$^{\mathcal{A}}\omega_{\mathcal{B}} = \dot{\phi}.$$

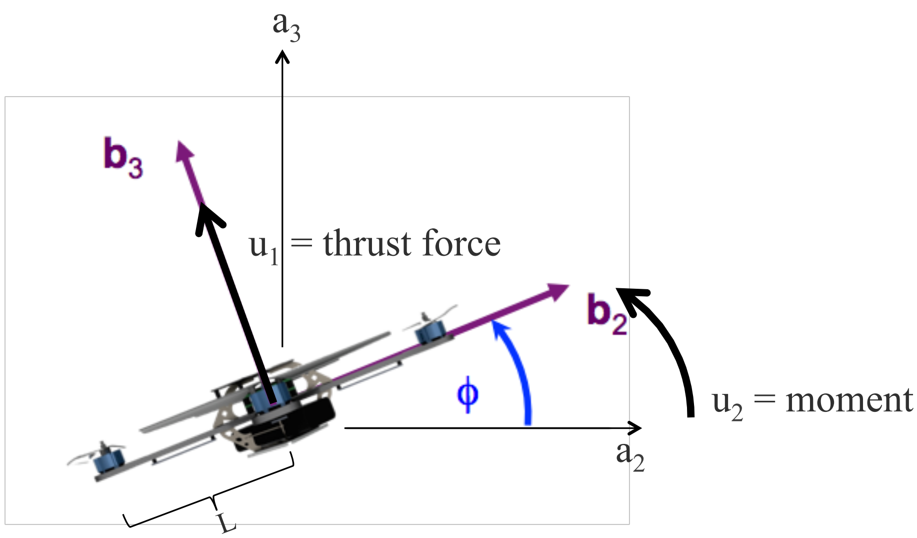


Figure 1: Planar quadrotor model and the coordinate systems.

In the planer model of the quadrotor, we only consider the thrust force on two of the rotors. The quadrotor has two inputs: the thrust force (u_1) and the moment (u_2). u_1 is the sum of the thrusts at each rotor

$$u_1 = \Sigma_{i=1}^{2} F_i \ ,$$

while u_2 is proportional to the difference between the thrusts of two rotors

$$u_2 = L(F_1 - F_2) \ .$$

Here, L is the arm length of the quadrotor.
Let $\mathbf{r} = [y, z]^T$ denote the position vector of planar quadrotor in $\mathcal{A}$. The forces on the system are gravity, in the $-\mathbf{a}_3$ direction, and the thrust force, in $\mathbf{b}_3$ direction. Hence, by Newton's Equations of Motion,

$$m\ddot{\mathbf{r}} = m \begin{bmatrix} \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ -mg \end{bmatrix} +^{\mathcal{A}} [R]_{\mathcal{B}} \begin{bmatrix} 0 \\ u_1 \end{bmatrix} = \begin{bmatrix} 0 \\ -mg \end{bmatrix} + \begin{bmatrix} -u_1 \sin(\phi) \\ u_1 \cos(\phi) \end{bmatrix} . \tag{2}$$

The angular acceleration is determined by Euler's equation of motion

$$I_{xx}\ddot{\phi} = L(F_1 - F_2) = u_2$$

As a result, the system model can be written as,

$$\begin{bmatrix} \ddot{y} \\ \ddot{z} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} -\frac{1}{m}\sin(\phi) & 0 \\ \frac{1}{m}\cos(\phi) & 0 \\ 0 & \frac{1}{I_{xx}} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{3}$$

3 Controller

3.1 Linearization

The dynamic model of the quadrotor (Eq. 3) is nonlinear. However, a PD controller is designed for a linear system. To use a linear controller for this nonlinear system, we first linearize the equation of motions about an equilibrium configuration.
In the case of the quadrotor, the equilibrium configuration is the hover configuration at any arbitrary position y_0, z_0, with zero roll angle. The corresponding thrust force needed to hover at this configuration is exactly mg, while the moment must be zero. Explicitly, the values of the related variables at the hover configuration are

$$y_0, z_0, \phi_0 = 0, u_{1,0} = mg, u_{2,0} = 0$$

To linearize the dynamics, we replace all non-linear function of the state and control variables with their first order Taylor approximations at the equilibrium location. In this case, the non-linear functions are $\sin(\phi)$ and $\cos(\phi)$. Near $\phi = 0$, $\sin(\phi) \approx \phi$ and $\cos(\phi) \approx 1$

$$\begin{aligned}
\ddot{y} =& - g\phi \\
\ddot{z} =& - g + \frac{u_1}{m} \\
\ddot{\phi} =& \frac{u_2}{I_{xx}}
\end{aligned}$$

Let r denote a state variable, either y, z or ϕ. We can find the commanded acceleration of that state, $\ddot{r}_c$, corresponding to a (PD) controller as follows. Define the position and velocity errors as

$$\begin{aligned}
e_p =& r_T(t) - r \\
e_v =& \dot{r}_T(t) - \dot{r}
\end{aligned}$$

We want error to satisfy the following differential equation, which will result in convergence of the error for some value subof k_p and k_d.

$$(\ddot{r}_T(t) - \ddot{r}_c) + k_p e_p + k_v e_v = 0 \tag{4}$$

From this, we can see that

$$\ddot{r}_c = \ddot{r}_T(t) + k_p e_p + k_v e_v, \tag{5}$$

where k_p and k_v are proportional and derivative gains respectively.
As a result, the inputs u_1, u_2, can be derived as:

$$u_1 = mg + m\ddot{z}_c = m\{g + \ddot{z}_T(t) + k_{v,z}(\dot{z}_T(t) - \dot{z}) + k_{p,z}(z_T(t) - z)\} \tag{6}$$

$$u_2 = I_{xx}\ddot{\phi}_T(t) = I_{xx}(\ddot{\phi}_c + k_{v,\phi}(\dot{\phi}_c - \dot{\phi}) + k_{p,\phi}(\phi_c - \phi)) \tag{7}$$

$$\phi_c = -\frac{\ddot{y}_c}{g} = -\frac{1}{g}(\ddot{y}_T(t) + k_{v,y}(\dot{y}_T(t) - \dot{y}) + k_{p,y}(y_T(t) - y)) \tag{8}$$

3.2 Hover Controller

Hovering is the special case of which the desired position is constant and the desired roll is zero. From $\mathbf{r}_T(t) = \mathbf{r}_0 = [y_0, z_0]^T, \phi_T(t) = \phi_0, \dot{\mathbf{r}}_T(t) = \ddot{\mathbf{r}}_T(t) = 0$ we get:

$$\begin{aligned}
u_1 =& m[g - k_{v,z}\dot{z} + k_{p,z}(z_0 - z)] \\
u_2 =& I_{xx}(\ddot{\phi}_c + k_{v,\phi}(\dot{\phi}_c - \dot{\phi}) + k_{p,\phi}(\phi_c - \phi)) \\
\phi_c =& -\frac{1}{g}(k_{v,y}(-\dot{y}) + k_{p,y}(y_0 - y))
\end{aligned}$$

3.3 Trajectory Controller

For trajectory following, given the desired trajectories for each state and their derivatives, $r_T(t), \dot{(r)}_T(t), \ddot{(r)}_T(t)$, the inputs u_1, u_2 can be calculated using Equations 6-8.

4 Gain Tuning

Until now you have been tuning rather simple controllers with a small number of gains. The controller in this assignment is slightly more complex, containing four gains (two for position and two for attitude) and the controller in your last assignment will contain 12 gains (six for position and six for attitude). A trial and error approach to tuning such a controller will be inefficient. Hence, we introduce a more systematic approach to gain tuning, the *Ziegler-Nichols method*.

4.1 Ziegler-Nichols method

The Ziegler-Nichols method is a method used to tune P, PI, PD, and PID controllers. We will only cover this method as applied to PD controllers. The method is outlined as follows:

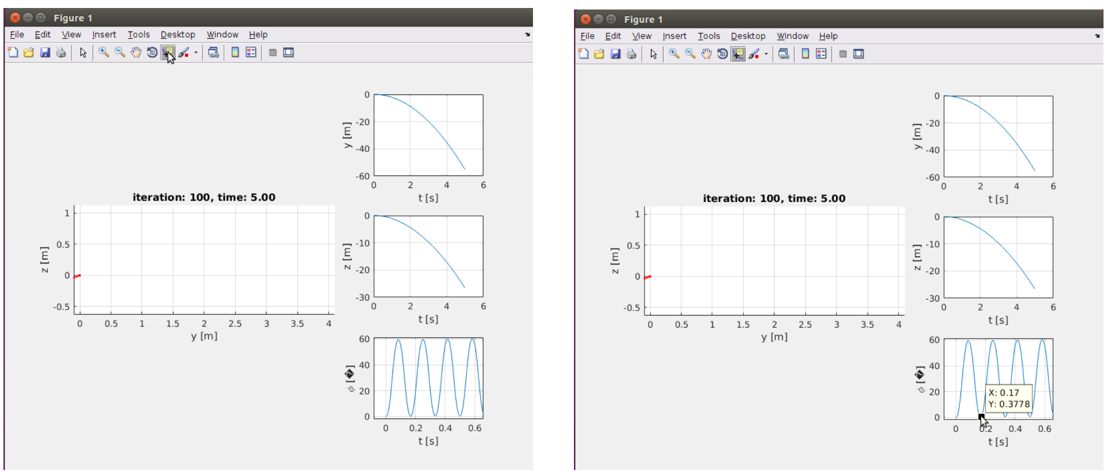
1. Set the proportional and derivative gains to zero
2. Turn the proportional gain up until the system outputs sustained oscillations.
3. Note the value of the proportional gain. This value is called the *ultimate gain*, K_u.
4. Measure T_u, which is the period of the oscillations.
5. Set the proportional gain to $K_p = 0.8K_u$.
6. Set the derivative gain to $K_d = T_u/8$.

If you follow the above procedure and the controller response is too slow, try scaling the proportional and derivative gains up by the same constant until you get a sufficiently quick response.

4.2 Tuning your controller

Because you are implementing a hierarchical control system in which the attitude controller forms the "inner loop", it is recommended that you first tune the attitude controller and then the position controller. You can tune the attitude controller by setting all of your position gains to zero and setting a desired roll angle to a certain value, say $\pi/6$.

We provide you with output plots so that you can determine when you have reached the ultimate gain and the period of oscillation. MATLAB provides all of the tools necessary to analyze these plots. After running your simulation and zooming into the plot (using the "Zoom In" tool on the Figure Toolbar), click the "Data Cursor" tool on the Figure Toolbar, as shown in Figure 2(a). Then click on the angle response plot in the location of the first period to get the xy coordinates of that point (the x coordinate is the period T_u), as shown in Figure 2(b).



(a) click on the "Data Cursor" tool

(b) then click on the point whose coordinates you wish to know

Figure 2: using MATLAB plotting tools to determine T_u

When tuning the position gains, you will wish to use the step input trajectory handle and lengthen the duration of the simulation. To select the step input trajectory handle, enter the line:

```
trajhandle = @traj_step;
```

in `runsim.m` before `simulation_2d` is called and comment out every other trajectory handle assignment. To change the run time of the simulation, open `simulation_2d.m`, scroll down to the initial conditions and change the value of the `t_total` variable.

5 Assignment

5.1 Files included in this exercise

[$\star$] `controller.m` - Controller for planar quadrotor.

`evaluate.p` - Code that evaluates your controller.

`runsim.m` - Test script to be called for testing.

`simulation_2d.m` - Simulation code that will be called by runsim.

`submit.m` - Script which calls the `evaluate` function for getting submission results.

`trajecories/` - Folder includes example trajectories for testing your controller.

`utils/` - Folder containing helper functions which you can use.

$\star$ indicates files you will need to implement

5.2 Tasks

You will need to complete the implementation of a PD controller in the file `controller.m` so that the simulated quadrotor can follow a desired trajectory. Before implementing your own functions, you should first try to run `runsim.m` in your MATLAB environment. If you see a quadrotor falling from position $(0, 0)$, then the simulator works on your computer and you may continue with other tasks. Again, the quadrotor falls because the default outputs of the `controller.m` function are all zeros, and no thrust is applied.

To test different trajectories, you will need to modify the variable `trajhandle` inside the `runsim.m` to point to the appropriate function. Examples are provided inside the `runsim.m` file.

5.3 Submission and Grading

To submit your results to our server, you need to run the command `submit` in your MATLAB command window. A script will then evaluate your controller on two trajectories and generate output files (files with the type `.mat`) to be uploaded to the web UI. There will be one output file for each test case. The evaluation is based on how well your controller can follow the desired trajectories. For each trajectory, we have upper and lower thresholds for the cumulative position error while following the trajectory. If your error is below the lower threshold, you get full points while if the error is larger than the upper threshold you get zero. If the error is in between the two, we just do a linear interpolation for the score. The thresholds for the two trajectories are shown in the following table:

Part	**Submitted File**	**Error thresholds**	**Points**
Line trajectory	line.mat	0.08, 0.15	20
Sine wave trajectory	sine.mat	0.10, 0.20	30
Total Points			50

You may submit your results multiple times, and we will count only the highest score towards your grade.