Programming Exercise 3:
3D Controller and Trajectory Generation

1 Introduction

In this exercise, you will be extending your PD controller from the previous assignment to control a quadrotor in 3D. You will then generate the time parameterized trajectories such that the quadrotor will navigate through all of the given waypoints. 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.

Again, you will need to download the starter code and unzip its contents into the directory in which you wish to complete the exercise.

2 System Model

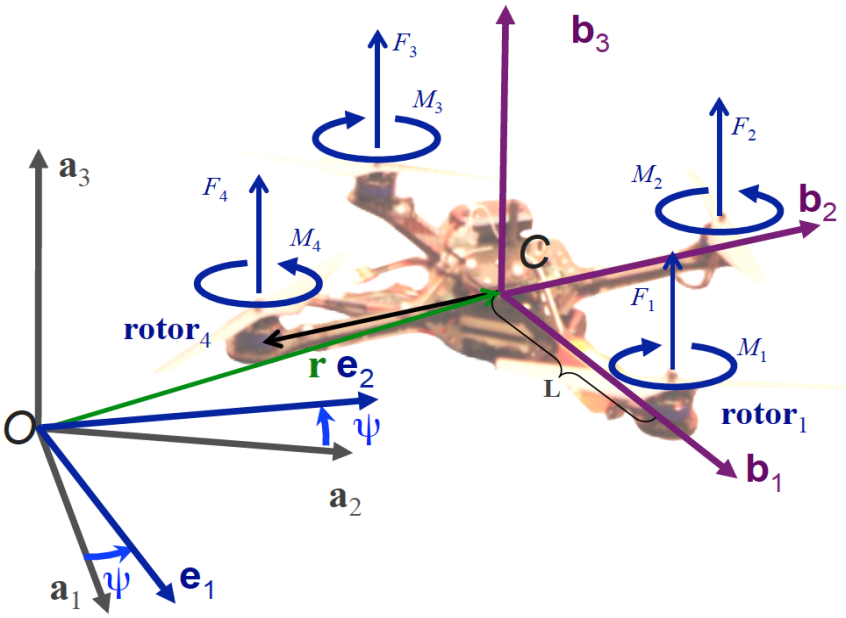

Figure 1: Quadrotor model with the body-fixed and the inertial reference frames.

2.1 Coordinate Systems

The coordinate systems and free body diagram for the quadrotor are shown in Fig. 1. The inertial frame, $\mathcal{A}$, is defined by the axes $\mathbf{a}_1, \mathbf{a}_2$, and $\mathbf{a}_3$ around the origin (O). The body frame, $\mathcal{B}$, is attached to the center of mass of the quadrotor with $\mathbf{b}_1$ coinciding with the preferred forward direction and $\mathbf{b}_3$ perpendicular to the plane of the rotors pointing vertically up during perfect hover (see Fig. 1). These vectors are parallel to the principal axes. The center of mass is C. Rotor 1 is a distance L away along $\mathbf{b}_1$, 2 is L away along $\mathbf{b}_2$, while 3 and 4 are similarly L away along the negative $\mathbf{b}_1$ and $\mathbf{b}_2$ respectively.
Since $\mathbf{b}_i$ are principal axes, the inertia matrix referenced to the center of mass along the $\mathbf{b}_i$ reference triad, I, is a diagonal matrix. This matrix will be provided to you as variable `params.I`.

2.2 Motor Model

Each rotor has an angular speed ω_i and produces a vertical force F_i according to

$$F_i = k_F \omega_i^2. \tag{1}$$

Experimentation with a fixed rotor at steady state shows that $k_F \approx 6.11 \times 10^{-8} \frac{N}{rpm^2}$. The rotors also produce a moment according to

$$M_i = k_M \omega_i^2. \tag{2}$$

The constant, k_M, is determined to be about $1.5 \times 10^{-9} \frac{Nm}{rpm^2}$ by matching the performance of the simulation to the real system.

2.3 Rigid Body Dynamics

We will use $Z - X - Y$ Euler angles to model the rotation of the quadrotor in the world frame. To get from $\mathcal{A}$ to $\mathcal{B}$, we first rotate about $\mathbf{a}_3$ through the yaw angle, ψ, to get the triad $\mathbf{e}_i$. A rotation about the $\mathbf{e}_i$ through the roll angle, ϕ, gets us to the triad $\mathbf{f}_i$ (not shown in the figure). A third pitch rotation about $\mathbf{f}_2$ through θ results in the body-fixed triad $\mathbf{b}_i$. You will need the rotation matrix for transforming components of vectors in $\mathcal{B}$ to components of vectors in $\mathcal{A}$:

$${}^{\mathcal{A}}[R]_{\mathcal{B}} = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \tag{3}$$

We will denote the components of angular velocity of the robot in the body frame by p, q, and r:

$${}^{\mathcal{A}}\omega_{\mathcal{B}} = p\mathbf{b}_1 + q\mathbf{b}_2 + r\mathbf{b}_3.$$

These values are related to the derivatives of the roll, pitch, and yaw angles according to

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{4}$$

Newton's Equations of Motion Let $\boldsymbol{r}$ denote the position vector of C in $\mathcal{A}$. The forces on the system are gravity, in the $-\boldsymbol{a_3}$ direction, and the forces from each of the rotors, F_i, in the $\boldsymbol{b_3}$ direction. The equations governing the acceleration of the center of mass are

$$m\ddot{\boldsymbol{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{5}$$

We will define our first input u_1 to be

$$u_1 = \Sigma_{i=1}^{4} F_i.$$

Euler's Equations of Motion In addition to forces, each rotor produce a moment perpendicular to the plane o frotation of the blade, M_i. Rotors 1 and 3 rotate in the $-\boldsymbol{b_3}$ direction while 2 and 4 rotate in the $+\boldsymbol{b_3}$ direction. Since the moment produced in the quadrotor is opposite to the direction of rotation of the blades, M_1 and M_3 act in the $\boldsymbol{b_3}$ direction while M_2 and M_4 act in the $-\boldsymbol{b_3}$ direction. L is the distance from the axis of rotation of the rotors to the center of mass of the quadrotor.

The angular acceleration determined by the Euler equations is

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{6}$$

We can rewrite this as:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{7}$$

where $\gamma = \frac{k_M}{k_F}$ is the relationship between lift and drag given by Equations (1-2). Accordingly, we will define our second set of inputs to be the vector of moments $\boldsymbol{u_2}$ given by:

$$\boldsymbol{u_2} = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}.$$

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

$$I_{xx}\ddot{\phi} = L(F_2 - F_4) = u_2$$

3 Robot Controllers

3.1 The Nominal State

Our controllers are derived by linearizing the equations of motion and motors models (5 – 2) at an operating point that corresponds to the nominal hover state, $\boldsymbol{r} = \boldsymbol{r}_0$, $\theta = \phi = 0$,

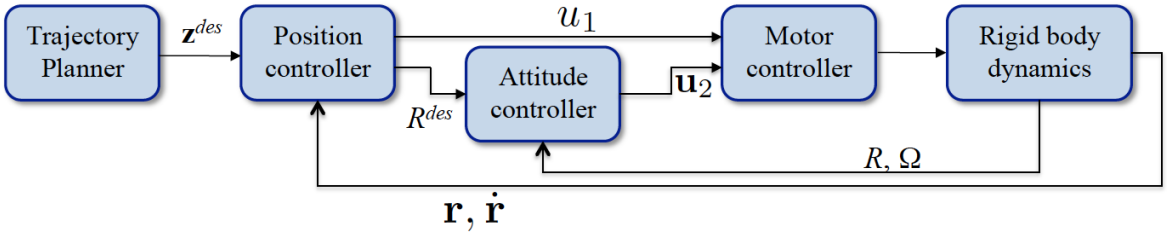

Figure 2: The position and attitude control loops.

$\psi = \psi_0$, $\dot{\boldsymbol{r}} = 0$, and $\dot{\phi} = \dot{\theta} = \dot{\psi} = 0$, where the roll and pitch angles are small ($c\phi \approx 1, c\theta \approx 1, s\phi \approx \phi$, and $s\theta \approx \theta$). At this state the lift from the propellers is given by:

$$F_{i,0} = \frac{mg}{4},$$

The nominal values for the inputs at hover are $u_{1,0} = mg$, $\boldsymbol{u}_{2,0} = \mathbf{0}$.

Linearizing (5), we get:

$$\begin{aligned} \ddot{r}_1 &= g(\Delta\theta \cos\psi_0 + \Delta\phi \sin\psi_0) \\ \ddot{r}_2 &= g(\Delta\theta \sin\psi_0 - \Delta\phi \cos\psi_0) \\ \ddot{r}_3 &= \frac{1}{m} u_1 - g \end{aligned} \tag{8}$$

Linearizing (6), we get:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{9}$$

In other words, the equations of motion are decoupled in terms of angular accelerations. Each component of angular acceleration depends only on the appropriate component of $\boldsymbol{u}_2$.

3.2 Position and Attitude Control

The control problem is to determine the four inputs, $\{u_1, \boldsymbol{u}_2\}$ required to hover or to follow a desired trajectory, $\boldsymbol{z}_{des}$. As shown in Figure 2, we will use errors in the robot's position to drive a position controller from (8) which directly determines u_1. The model in (8) also allows us to derive a desired orientation. The attitude controller for this orientation is derived from the model in (9).

The transformation of the inputs into $\{u_1, \boldsymbol{u}_2\}$ is straightforward and is described in [1]. The inner attitude control loop uses onboard accelerometers and gyros the control the roll, pitch, and yaw and runs at approximately 1 kHz, while the outer position control oop uses estimates of position and velocity of the center of mass to control the trajectory in three dimensions at 100-200 Hz.

Attitude Control We now present a a proportional plus derivative (PD) attitude controller to track a trajectory in $SO(3)$ specified in terms of a desired roll, pitch and yaw angle. Since our development of the controller will be based on linearized equations of motion, the attitude must be close to the nominal hover state where the roll and pitch angles are small.

Near the nominal hover state the proportional plus derivative control laws take the form:

$$\boldsymbol{u}_2 = \begin{bmatrix} k_{p,\phi}(\phi_{des} - \phi) + k_{d,\phi}(p_{des} - p) \\ k_{q,\theta}(\theta_{des} - \theta) + k_{d,\theta}(q_{des} - q) \\ k_{r,\phi}(\psi_{des} - \psi) + k_{d,\psi}(r_{des} - r) \end{bmatrix} \tag{10}$$

Position Control In the next subsection, we will present two position control methods that use the roll and pitch angles as inputs to drive the position of the quadrotor. In both methods, the position control algorithm will determine the desired roll and pitch angles, θ_{des} and ϕ_{des}, which can be used to compute the desired speeds from (10). The first, a hover controller, is used for station-keeping or maintaining the position at a desired position vector, $\boldsymbol{r}_0$. The second tracks a specified trajectory, $\boldsymbol{r}_T(t)$, in three dimensions. In both cases, the desired yaw angle, is specified independently. It can either be a constant, ψ_0, or a time varying quantity, $\psi_T(t)$. We will assume that the desired trajectory:

$$\boldsymbol{z}_{des} = \begin{bmatrix} \boldsymbol{r}_T(t) \\ \psi_T(t) \end{bmatrix},$$

will be provided by a trajectory planner as an input to specify the trajectory of the position vector and the yaw angle we are trying to track.

3.3 Hover Controller

For hovering, $\boldsymbol{r}_T(t) = \boldsymbol{r}_0$ and $\psi_T(t) - \psi_0$. The command accelerations, $\ddot{r}_{i,des}$, are calculated from a proportional plus derivative (PD) controller. Define the position error in terms of components using the standard reference triad $\boldsymbol{a}_i$ by:

$$e_i = (r_{i,T} - r_i).$$

In order to guarantee that this error goes exponentially to zero, we require

$$(\ddot{r}_{i,T} - \ddot{r}_{i,des}) + k_{d,i}(\dot{r}_{i,T} - \dot{r}_i) + k_{p,i}(r_{i,T} - r_i) = 0, \tag{11}$$

where $\dot{r}_{i,T} = \ddot{r}_{i,T} = 0$ for hover.

From (8) we can obtain the relationship between the desired accelerations and roll and pitch angles. Given that $\delta\theta = \theta - \theta_0 = \theta$ and $\delta\psi = \psi - \psi_0 = \psi$, we can write

$$\begin{aligned} \ddot{r}_{1,des} &= g(\theta_{des}\cos\psi_T + \phi_{des}\sin\psi_T) \\ \ddot{r}_{2,des} &= g(\theta_{des}\sin\psi_T - \phi_{des}\cos\psi_T) \\ \ddot{r}_{3,des} &= \frac{1}{m}u_1 - g. \end{aligned} \tag{12}$$

For hover, the last equation yields:

$$u_1 = mg + m\ddot{r}_{3,des} = mg - m(k_{d,3}\dot{r}_3 + k_{p,3}(r_3 - r_{3,0})) \tag{13}$$

The other two equations can be used to compute the desired roll and pitch angles for the attitude controller:

$$\phi_{des} = \frac{1}{g}(\ddot{r}_{1,des}\sin\psi_T - \ddot{r}_{2,des}\cos\psi_T) \tag{14a}$$

$$\theta_{des} = \frac{1}{g}(\ddot{r}_{1,des}\cos\psi_T + \ddot{r}_{2,des}\sin\psi_T) \tag{14b}$$

The desired roll and pitch velocities are taken to be zero.

$$p_{des} = 0 \tag{15a}$$

$$q_{des} = 0 \tag{15b}$$

Since the yaw, $\psi_T(t)$ is prescribed independently by the trajectory generator, we get:

$$\psi_{des} = \psi_T(t) \tag{16a}$$

$$r_{des} = \dot{\psi}_T(t) \tag{16b}$$

These equations provide the setpoints for the attitude controller in (10).

Note that the attitude controller must run an order of magnitude faster than the position control loop in order for this to work. In practice as discussed in [1], the position controller runs at 100 Hz, while the inner attitude control loop runs at 1 kHz.

3.4 3D Trajectory Control

The 3-D Trajectory Controller is used to follow three-dimensional trajectories with modest accelerations so the near-hover assumptions hold. To derive this controller we follow the same steps as in (11) except that $\dot{r}_{i,T}$ and $\ddot{r}_{i,T}$ are no longer zero but are obtained from the specification of the trajectory. If the near-hover assumption holds and the dynamics are linear with no saturation on the inputs, a controller that generates the desired acceleration $\ddot{r}_{i,des}$ according to (11) is guaranteed to drive the error exponentially to zero. However, it is possible that the commanded trajectory has twists and turns that are too hard to follow perfectly because of errors in the model or limitations on the input thrusts. We suggest a modification that allows the robot to follow the path even if it may not be able to follow the time-parameterized trajectory.

Let $\boldsymbol{r}_T$ denote the closest point on the desired trajectory to the current position $\boldsymbol{r}$, and let the desired velocity and acceleration obtained by differentiating the specified trajectory be given by $\dot{\boldsymbol{r}}_T$ and $\ddot{\boldsymbol{r}}_T$ respectively. Let the unit tangent vector of the trajectory (unit vector along $\dot{\boldsymbol{r}}_T$ be $\hat{\boldsymbol{t}}$. The unit normal to the trajectory, $\hat{\boldsymbol{t}}$, is derived by the differentiating the tangent vector with respect to time or arc length, and finally the unit binormal vector is the cross product $\hat{} = \hat{\boldsymbol{t}} \times \hat{\boldsymbol{n}}$. We define the position and velocity errors according to the following equations:

$$\boldsymbol{e}_p = ((\boldsymbol{r}_T - \boldsymbol{r}) \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}} + ((\boldsymbol{r}_T - \boldsymbol{r}) \cdot \hat{\boldsymbol{b}})\hat{\boldsymbol{b}}$$

and

$$\boldsymbol{e}_v = \dot{\boldsymbol{r}}_T - \dot{\boldsymbol{r}}$$

Note that here we ignore position error in the tangential direction and are only considering position error in the plane that is normal to the curve at the closest point. Once again we calculate the commanded acceleration, $r_{i,des}$, from the PD feedback as shown in (11). In vector notation, this is:

$$(\dot{\boldsymbol{r}}_T - \dot{\boldsymbol{r}}_{des}) + \boldsymbol{k}_d \boldsymbol{e}_v + \boldsymbol{k}_p \boldsymbol{e}_p = 0, \tag{17}$$

Finally we use (14 - 16) to compute the desired roll and pitch angles. Again we refer the readers to [1] for more details including experimental results obtained from these controllers.

4 Trajectory Generation

From lecture, we know that a necessary condition for a minimum snap trajectory is that it is a 7th order polynomial. If we are given a set of $n+1$ waypoints $w_0, \ldots, w_n$, the minimum snap trajectory is a piecewise polynomial composed of n 7th order polynomials. Each polynomial piece p_i travels between a pair of waypoints w_{i-1} and w_i and takes a known amount of time T_i to complete, where $i = 1, \ldots, n$.

Let $S_0 = 0$ and, for $i = 1, \ldots, n, S_i = \sum_{k=1}^{i} T_k$. That is, S_i is the time it takes to reach waypoint w_i from waypoint w_0. Then, the polynomial p_i has the form:

$$p_i(t) = \alpha_{i0} + \alpha_{i1}\frac{t - S_{i-1}}{T_i} + \alpha_{i2}\left(\frac{t - S_{i-1}}{T_i}\right)^2 + \cdots + \alpha_{i7}\left(\frac{t - S_{i-1}}{T_i}\right)^7 \tag{18}$$

To obtain the complete equation for the piecewise trajectory, we need to solve for all the coefficients α_{ij}. There are $8n$ such coefficients. These coefficients must meet a series of constraints. First, the polynomials must go through the given waypoints, giving $2n$ constraints:

$$p_i(S_{i-1}) = w_{i-1} \text{ for all } i = 1, \ldots, n$$
$$p_i(S_i) = w_i \text{ for all } i = 1, \ldots, n$$

Second, the quadrotor must start and stop at rest, giving two more constraints:

$$\dot{p}_1(S_0) = \dot{p}_n(S_n) = 0$$

Next, the velocities and accelerations must be continuous between each segment, giving us an additional $n - 1$ constraints:

$$\dot{p}_i(S_i) = \dot{p}_{i+1}(S_i) \text{ for all } i = 1, \ldots, n-1$$

We can also do this with the accelerations to get $n - 1$ more constraints:

$$\ddot{p}_i(S_i) = \ddot{p}_{i+1}(S_i) \text{ for all } i = 1, \ldots, n-1$$

So far we have $4n$ constraints. We need $4n$ more in order to fully define all of the coefficients of our trajectory. How will we get these constraints? We get them from specifying continuity in higher derivatives of the trajectory at the intermediate waypoints ($n - 1$ constraints for each derivative), and from specifying the higher derivatives of the trajectory to be zero at the endpoints (2 constraints per derivative). So, to get $8n$ constraints, we can specify that the 3rd through 6th derivatives are continuous, giving us $4n - 4$ more constraints:

$$p_i^{(k)}(S_i) = p_{i+1}^{(k)}(S_i) \text{ for all } i = 1, \ldots, n-1 \text{ and } k = 3, \ldots, 6$$

and then we can specify that the acceleration and jerk are zero at the endpoints, bringing us up to $8n$ constraints:

$$p_1^{(k)}(S_0) = p_n^{(k)}(S_n) = 0 \text{ for } i = 2, 3$$

To recap, the set of constraints needed to solve for the coefficients of the piecewise polynomial are:

$$p_i(S_{i-1}) = w_{i-1} \text{ and } p_i(S_i) = w_i \text{ for all } i = 1, \ldots, n \text{ } (2n \text{ constraints}) \tag{19}$$

$$p_1^{(k)}(S_0) = p_n^{(k)}(S_n) = 0 \text{ for all } k = 1, \ldots, 3 \text{ (6 constraints)} \tag{20}$$

$$p_i^{(k)}(S_i) = p_{i+1}^{(k)}(S_i) = 0 \text{ for all } k = 1, \ldots, 6 \text{ } (6n - 6 \text{ constraints}) \tag{21}$$

We can see that now, the number of constraint equations matches the number of unknown coefficients α_{ij}.

First, convert the equations to matrix form (you will have to work this out on your own):

$$A\alpha = b, \tag{22}$$

where α is a vector containing the unknown coefficients α_{ij} and A and b are matrices representing all constraints. The coefficients can be obtained by solving:

$$\alpha = A^{-1}b \tag{23}$$

Refer to the MATLAB tutorials from Week 1 if you need help solving this matrix equation.

5 Assignment

5.1 Files included in this exercise

[⋆] `controller.m` - Controller for a full 3D quadrotor.

[⋆] `traj_generator.m` - Trajectory generator for generating a trajectory passing through all the given waypoints.

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

`simulation_3d.m` - Simulation code that will be called by runsim.

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

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

`traj_helix.p` - Pre-defined helical trajectory for testing your controller.

`traj_line.p` - Pre-defined line trajectory for testing your controller.

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

⋆ indicates files you will need to implement

5.2 Tasks

5.2.1 Controller

You will need to complete the implementation of the PD controller in the file `controller.m`. Before implementing your own functions, you should first try running `runsim.m` in your MATLAB environment. If you see a quadrotor falling from starting position, then the simulator is running correctly and you may continue with other tasks.

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

5.2.2 Trajectory Generation

You will need to implement the function `traj_generator.m` to generate a trajectory through a series of way points. You may use the method outlined in section 4 or your own method of interpolating smoothly. The provided function gives an example trajectory which jumps directly between the waypoints. If you execute this, you can see the undesirable effects of giving a controller a non-smooth trajectory.

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 first two parts evaluate your controller based on how well it can follow the given trajectories. Similar to the 2D quadrotor assignment, 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 trajectories are shown in the following table:

Part	**Submitted File**	**Error Thresholds**	**Points**
Line trajectory	line.mat	0.2, 0.4	10
Helical trajectory	helix.mat	2.0, 4.0	20

The third part requires you to provide your own trajectory generator and use your controller to follow a trajectory passing through a given set of waypoints. The evaluation for this part is based on how close the robot passes to the given waypoints and how long it takes to complete the trajectory. To count a waypoint as being cleared, the robot has to pass within 0.05 m of the waypoint. You are given 5 waypoints that your trajectory has to pass through and clearing each one earns you 16% of the total score. The time taken for finishing the trajectory is responsible for the remaining 20%. Similar to the error thresholds for the other two parts, we have time thresholds here. A completion time of less than 15 seconds gets the full 20% while anything above 20 seconds results in no points for completion time.

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

References

[1] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP Multiple Micro-UAV Testbed," *IEEE Robotics and Automation Magazine*, 2010.