Trajectory Planning for a Quadrotor

MEE 4411/5411 Project 2, Phases 2 and 3

"Due:" Tuesday, December 5

1 Optimal Trajectory Planning

As discussed in lecture, we will find optimal polynomial trajectories for the quadrotor to follow through the environment. The starter code provides 3D mapping and A* implementations, and it is your task to take the resulting waypoints from the A* planner and use them to create a continuous, polynomial trajectory for the robot. The remainder of this document will detail this process.

1.1 Order of the System

As discussed in lecture, the quadrotor is a mixture of a second order system (in the roll, pitch, yaw, and z directions) and a fourth order system (in the x and y directions). You are welcome to treat the quadrotor as a second, third, or fourth order system. All of these are commonly used in practice.

The Lagrangian for an nth order system is

$$\mathcal{L}(x^{(n)}, \ldots, \dot{x}, x, t) = \left(x^{(n)}\right)^2. \tag{1}$$

We then plug this into the Euler-Lagrange equation,

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{x}} + \ldots + (-1)^n \frac{d^n}{dt^n}\frac{\partial \mathcal{L}}{\partial (x^{(n)})} = 0, \tag{2}$$

which yields the result that

$$x^{(2n)} = 0. \tag{3}$$

A polynomial trajectory of order $2n-1$ would then be optimal, and so it only remains to determine the $2n$ coefficients of the polynomial,

$$x(t) = c_{2n-1}t^{2n-1} + \ldots + c_1 t + c_0. \tag{4}$$

For the quadrotor, a second, third, or fourth order system leads to third, fifth, or seventh order polynomials, respectively.

2 Determining the Coefficients

For the quadrotor, we need to plan trajectories in x, y, z, and yaw. As we saw in the lectures on quadrotor control, the roll and pitch angles are determined by the desired linear accelerations. Due to a property of the quadrotor called differential flatness, we can *independently* plan trajectories in the x, y, z, and yaw directions.[1]

[1] Mellinger, D., & Kumar, V. (2011, May). Minimum snap trajectory generation and control for quadrotors. In IEEE International Conference on Robotics and Automation (ICRA) (pp. 2520-2525).

2.1 Trimming the Points

The output of A* has many points along the path. To reduce the number of segments that must be planned, one thing that may be of help is to use the `trim_path` function from Project 1, Phase 4 to remove the intermediate points and only keep the points where the path changes direction. You are of course welcome to modify this in any way that is helpful. If you don't wish to use `trim_path`, then simply comment it out (line 28 in `runsim`).

2.2 Time Length

The first thing to do is to determine the time length of each segment. This can be done in many ways. One option is to pick a desired average velocity. For long segments this will work well. However, for short segments, where the robot does not have the time to accelerate up to the desired velocity, this will cause the robot to accelerate too quickly which will lead the linear controller to fail. Instead, for these short segments you can, for example, just pick a fixed time length to complete the segment or pick a slower desired velocity. Given a desired average velocity, the time of a segment is the distance divided by the desired velocity. Remember to use the total distance, not just the distance in x, y, or z.

2.3 Single-Segment Coefficients

For a single segment, we need to define enough boundary conditions in order to solve for the $2n$ coefficients of the polynomial. The most natural ones are to ensure that the segment will start and end at the desired location and will start and stop at rest. This gives us

$$\begin{array}{lllll} x(0) = x_s & \dot{x}(0) = 0 & \ddot{x}(0) = 0 & \cdots & x^{(n)}(0) = 0 \\ x(T) = x_f & \dot{x}(T) = 0 & \ddot{x}(T) = 0 & \cdots & x^{(n)}(T) = 0, \end{array}$$

where the trajectory starts at x_s at time $t = 0$ and ends at x_f at time $t = T$. This gives us $2n$ equations to solve for the $2n$ coefficients.

We know that

$$\begin{aligned} x(t) &= c_{2n-1}t^{2n-1} + \ldots + c_1 t + c_0 \\ \dot{x}(t) &= (2n-1)c_{2n-1}t^{2n-2} + \ldots + c_1 \\ &\vdots \end{aligned}$$

Putting these together with the boundary conditions, we can create a set of linear equations of the form, for example, with a second order system

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ T^3 & T^2 & T & 1 \\ 0 & 0 & 1 & 0 \\ 3T^2 & 2T & 1 & 0 \end{bmatrix} \begin{bmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} x_s \\ x_f \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

We can then solve this system of linear equations for the vector of unknown coefficients. Also note that for a single segment, the matrix on the left will be the same for all of the different components, x, y, z, and yaw. So you only need to construct this matrix once for a segment. In fact, this can be sped up even more by combining all of the

equations into one of the form

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ T^3 & T^2 & T & 1 \\ 0 & 0 & 1 & 0 \\ 3T^2 & 2T & 1 & 0 \end{bmatrix} \begin{bmatrix} c_{x,3} & c_{y,3} & c_{z,3} & c_{\psi,3} \\ c_{x,2} & c_{y,2} & c_{z,2} & c_{\psi,2} \\ c_{x,1} & c_{y,1} & c_{z,1} & c_{\psi,1} \\ c_{x,0} & c_{y,0} & c_{z,0} & c_{\psi,0} \end{bmatrix} = \begin{bmatrix} x_s & y_s & z_s & \psi_s \\ x_f & y_f & z_f & \psi_f \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

2.4 Multi-Segment Trajectories

For multiple segments, there are multiple options. One is to break the multiple segments into a collection of individual segments, where each starts and stops at rest. In the case, each segment can be solved independently using the equations from before.

The other option is to create a single trajectory that passes through each waypoint. In this case, the equations at the waypoints can be used to create continuity, as discussed in lecture. This will couple all of the segments together, so they must all be solved for simultaneously with one large equation.

You are also welcome to come up with your own strategy. If you do this option, the only requirement is that you satisfy the assumptions of the data format in the paragraph below and in Section 3.

Regardless of the approach that we take, we will reset the timer for each segment to start at 0, meaning the initial time for each segment is 0 and the final time for each segment is the duration of that segment. So segment 1 will start at $t = 0$ and go until time $t = T_1$. Then segment 2, instead of starting at $t = T_1$ will start at $t = 0$ and end at $t = T_2$. While there is technically nothing wrong with the first approach, resetting the timer to 0 for each segment will lead to better numerical stability when computing the polynomial coefficients. Because of this, I will expect the coefficients to be solved for in this manner.

3 Code Requirements

You are required to fill in the function `computeCoefficients`. This is called inside of the `trajectory_generator` function. I encourage you to look at the `trajectory_generator` function to see how the coefficients are used to find the desired position, velocity, etc.

In `computeCoefficients`, you need to output two variables. The first is a vector `DT`. This is a row vector with the time duration of each segment. For example, the third element `DT(3)` is the time it takes the robot to go along the third segment, which connects the third and fourth waypoints in `path`.

The second output in a cell array `C` containing the coefficients of the polynomial trajectories. The cell array has one element per segment, and the format of each of those elements in a $2n \times 4$ matrix of coefficients, like the one found in (6). The columns contain the coefficients for the x, y, z, and yaw trajectories, respectively. The rows contain the coefficients of the polynomial in *descending* order. So c_{2n-1} is in the first row and c_0 is in the bottom row.

The starter code initializes a cell array and sets the values to be dummy variables. You can look at the syntax in the starter code for information about how to work with cell arrays. We also discussed these in class several weeks ago.

As always, the main function is `runsim`. Within `runsim` you can switch between several different pre-defined cases using the `scenario` variable. You are welcome to create your own scenarios as desired, but these pre-defined ones are known to have valid paths from the starts to the goals. `runsim` will show you a plot of both the A* path and the trajectory that you have planned. I encourage you to use these visualizations to help you determine how well your planner is working.

Also within `runsim` you can switch between Phases 2 and 3 using the variable `phase`. Using `phase = 2` will call the function `plot_trajectory`. This will take the output of your `computeCoefficients` function and use it to visualize the resulting trajectory. It will show you 5 figures:

1. 3D view of the environment, A* path, and your polynomial trajectory
2. Positions over time along with the waypoints
3. Linear velocities over time
4. Linear accelerations over time
5. Yaw and yaw angular velocity over time

You can use these figures to help visualize your path. I encourage you to make sure that it does not collide with any obstacle. Also be sure to look at the maximum velocity, acceleration, etc.

3.1 Controller Modifications

Using `phase = 3` will call the function `test_trajectory`. This will take the path that you compute and pass it to the robot to follow. In order for this to work, you should be sure to copy over your `controller.m` file from Phase 1 and put it in this folder (replacing the default `controller.m` that I have included). You will also need to modify your controller slightly from Phase 1.

Recall that in Phase 1 the desired velocities and accelerations were all 0 since the goal was to hover in place. In this Phase we wish to follow a trajectory over time so the desired velocities and accelerations will be non-zero. To account for this, we need to add in the following terms, shown in red:

$$\ddot{x} = \ddot{x}^{\text{des}} + k_{d,x}(\dot{x}^{\text{des}} - \dot{x}) + k_{p,x}(x - x^{\text{des}}) \tag{7}$$

$$\ddot{y} = \ddot{y}^{\text{des}} + k_{d,y}(\dot{y}^{\text{des}} - \dot{y}) + k_{p,y}(y - y^{\text{des}}) \tag{8}$$

$$\phi^{\text{des}} = \frac{1}{g}(\ddot{x}\sin\psi - \ddot{y}\cos\psi) \tag{9}$$

$$\theta^{\text{des}} = \frac{1}{g}(\ddot{x}\cos\psi + \ddot{y}\sin\psi) \tag{10}$$

$$u_1 = m\left(g + \ddot{z}^{\text{des}} + k_{d,z}(\dot{z}^{\text{des}} - \dot{z}) + k_{p,z}(z - z^{\text{des}})\right) \tag{11}$$

$$u_2 = I_{xx}\left(k_{d,\phi}(0 - \dot{\phi}) + k_{p,\phi}(\phi^{\text{des}} - \phi)\right) \tag{12}$$

$$u_3 = I_{xx}\left(k_{d,\theta}(0 - \dot{\theta}) + k_{p,\theta}(\theta^{\text{des}} - \theta)\right) \tag{13}$$

$$u_4 = I_{zz}\left(k_{d,\psi}(\dot{\psi}^{\text{des}} - \dot{\psi}) + k_{p,\psi}(\psi^{\text{des}} - \psi)\right) \tag{14}$$

Note that the desired angular velocity in the roll and pitch directions remains 0 in (12) and (13). This is because we want the robot to achieve a desired roll and pitch angle to achieve the desired linear accelerations. So once the robot reaches the desired roll and pitch angle, we want it to stay there. However, the desired yaw velocity in (14) comes from the output of your trajectory generation function.

As with Phase 1, the equations for u_1 and u_4, (11) and (14), respectively, can be computed independently. However, for the other two control inputs, we need to first compute (7) and (8), plug those in to (9) and (10), and then finally plug those into (12) and (13). All of these desired velocities and accelerations are available to you inside of the `controller` starter code that I provided to you.

3.2 Submission

When you are finished with Phase 2 you may submit your code by putting it all into a single `.zip` file and emailing it to pdames@temple.edu with subject line "MEE4411 Project 2.2" (clicking the previous link will automatically start a correctly formatted email for you). This link will do the same for Phase 3: pdames@temple.edu with subject line "MEE4411 Project 2.3" Prof. Dames will test your submission and send you a list of items to fix. You are encouraged to update your code to fix any bugs, however each student is limited to 3 submissions per phase of the project.