MEAM620 Project 1 Report

Group 9: Yiren Lu, Myles Cai, Wudao Ling, Xuanyu Zhou

May 7, 2017

1 Introduction

This project is intended to control the motion of the Unmanned Aerial Vehicle in the real world to follow a specific trajectory smoothly and quickly. In this report, we will discuss how we design the controller and trajectory generator to reach this purpose.

2 Controller Description

Our purpose is to keep the yaw rate of the vehicle be zero during the whole motion, which is a "hover-like" status. We applied a proportional plus derivative controller here, and we could get the command accelerations $\ddot{r}_i^{des}$ by using the following equation:

$$\ddot{r}_i^{des} = \ddot{r}_{i,T} + k_{d,i}(\dot{r}_{i,T} - \dot{r}_i) + k_{p,i}(r_{i,T} - r_i) \tag{1}$$

where $\ddot{r}_{i,T}$, $\dot{r}_{i,T}$, and $r_{i,T}$ are the desired acceleration, velocity and position of the designed trajectory. Therefore, here we have 6 gains needed to be tuned during the experiment. The thrust force could be calculated as:

$$u_1 = mg + m\ddot{r}_3^{des} \tag{2}$$

We could get the desired roll, pitch and yaw angles by:

$$\phi^{des} = \frac{1}{g}(\ddot{r}_1^{des}\sin\psi_T - \ddot{r}_2^{des}\cos\psi_T) \tag{3}$$

$$\theta^{des} = \frac{1}{g}(\ddot{r}_1^{des}\cos\psi_T + \ddot{r}_2^{des}\sin\psi_T) \tag{4}$$

$$\psi_{des} = \psi_T \tag{5}$$

then we apply another PD controller to the attitude control and we could get u_2 by:

$$u_2 = I\begin{bmatrix} k_{p,\phi}(\phi^{des} - \phi) + k_{d,\phi}(p^{des} - p) \\ k_{p,\theta}(\theta^{des} - \theta) + k_{d,\theta}(q^{des} - q) \\ k_{p,\psi}(\psi^{des} - \psi) + k_{d,\psi}(r^{des} - r) \end{bmatrix} \tag{6}$$

where p, q, r are angular velocities, and p^{des} and q^{des} are all set to be 0. And $r^{des} = \dot{\psi}_T$. Then we have another 6 more gains here to be tuned. All of the formulas above are from the handouts of the project. Totally, we have 12 parameters needed to be tuned. During the simulation and real experiment, we find the following gains works fine for our vehicle.

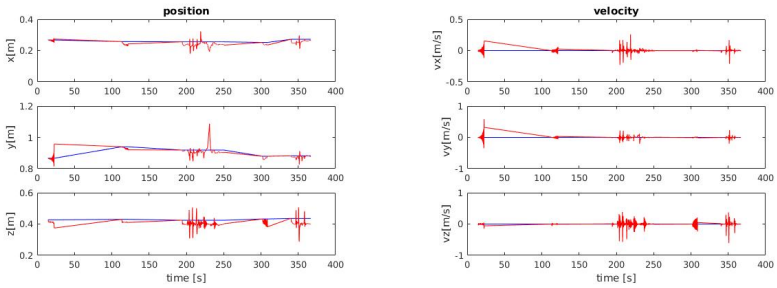
$$k_{p,1} = 5, k_{p,2} = 5, k_{p,3} = 20$$

$$k_{d,1} = 5, k_{d,2} = 5, k_{d,3} = 10$$

$$k_{p,\phi} = 3000, k_{p,\theta} = 3000, k_{p,\psi} = 3000$$

$$k_{d,\phi} = 300, k_{d,\theta} = 300, k_{d,\psi} = 300$$

Since in the real experiment, we are given the attitude controller, we only need to tune the gains for position controller. We could find that the gains for z-axis control is greater than that of the other two axises. It is because it needs to compensate the gravity force. And Figure 1 are the plots of comparison between actual and desired velocity and position during hovering.
The fluctuation here are the responses of the vehicle with respect to the small disturbances, and we could find that the gains that we choose work fine.



(a) Position comparison (b) Velocity comparison

Figure 1: Comparison between actual response and desired command

3 Trajectory Generator

Our team aims to generate smooth and safe trajectory following the requested waypoints in project 1 phase 4. We adopted both 3rd order minimum acceleration trajectory and 7th order minimum snap trajectory.

3.1 Speed Profile

With given waypoints in path cell, we first determine appropriate flight time for entire path and each segments, which further determine the speed profile. The total time is calculated by dividing total length of the path by the predefined speed constant. After testing, our team set speed constant to 0.5. Then, we distribute total time to each segment time based on the path segment length's square root. In this way, the quadrotor will go faster along long segment while fly slower along short segment (which generally means many turns). This speed profile is proved to be safe in both MATLAB simulation and real world testing.

3.2 Minimum Acceleration Trajectory

One of the most common trajectories is minimum acceleration trajectory. It consists of segmental 3rd order polynomials and achieves the balance between speed, safety and computational economy. Our team designed a minimum acceleration trajectory with continuity of velocity at waypoints.
Provided that there are m segments in the path, $4 * m$ constraints are necessary to solve m 3rd order:
4 constraints for start and end point

$$x_s(t=0) = p_s; \dot{x_s}(t=0) = 0;$$

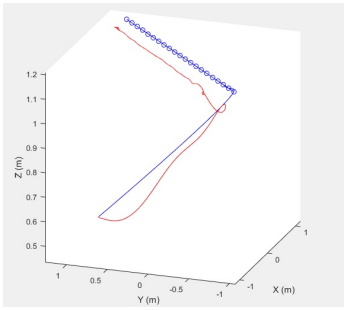
$$x_e(t=T) = p_e; \dot{x_e}(t=T) = 0;$$

$4 * (m-1)$ constraints for other $m-1$ waypoints

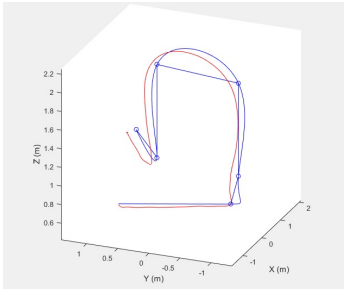
$$x_i(t_i) = x_{i-1}(t_i) = p_i;$$

$$\dot{x_i}(t_i) = \dot{x_{i-1}}(t_i); \ddot{x_i}(t_i) = \ddot{x_{i-1}}(t_i);$$

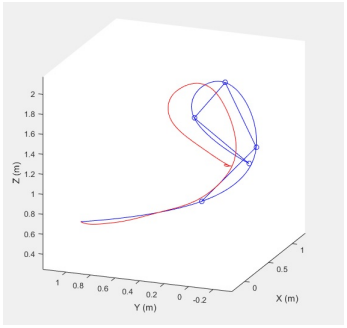
For continuity, the position and velocity of adjacent trajectories at waypoints should be equal (acceleration = 0) and 1 equation here means 2 constraints for 2 trajectories.
With enough constraints, the trajectories in x coordinate are then obtained by MATLAB matrix computation $x = A^{-1}b$. Similarly, the trajectories in y and z coordinate can be obtained.



(a) Task 1's desired and actual trajectory



(b) Task 2's desired and actual trajectory



(c) Task 3's desired and actual trajectory

Figure 2: Minimum acceleration trajectories generated with continuity constraints (blue curve)

Table 1: Minimum Acceleration Trajectory

	Time planned (s)	Time actual (s)	Len. planned (m)	Len. actual (m)	Avg. velocity (m/s)
Path 1	11.1507	13.8507	5.6223	5.6225	0.4059
Path 2	18.24	20.04	9.6071	9.6104	0.4796
Path 3	8.0405	9.7205	4.2314	4.2314	0.4353

3.3 Minimum Snap Trajectory

Our team also tested with the minimum snap trajectory, segmental 7th order polynomials. Also, we tried different constraints: continuity at waypoints and zero at waypoints.
Provided that there are m segments in the path, $8 * m$ constraints are necessary to solve 7th order trajectory:
8 constraints for start and end point

$$x_s(t=0) = p_s; \dot{x_s}(t=0) = 0; \ddot{x_s}(t=0) = 0; \dddot{x_s}(t=0) = 0;$$

$$x_e(t=T) = p_e; \dot{x_e}(t=T) = 0; \ddot{x_e}(t=T) = 0; \dddot{x_e}(t=T) = 0;$$

- Continuity : $8 * (m-1)$ constraints for other $m-1$ waypoints

$$x_i(t_i) = x_{i-1}(t_i) = p_i;$$

$$\dot{x_i}(t_i) = \dot{x_{i-1}}(t_i); \ddot{x_i}(t_i) = \ddot{x_{i-1}}(t_i); \dddot{x_i}(t_i) = \dddot{x_{i-1}}(t_i);$$
$$x_i^{(4)}(t_i) = x_{i-1}^{(4)}(t_i); x_i^{(5)}(t_i) = x_{i-1}^{(5)}(t_i); x_i^{(6)}(t_i) = x_{i-1}^{(6)}(t_i);$$

  The position, velocity, acceleration and jerk of adjacent trajectories at waypoints should be equal (snap = 0) and 1 equation here means 2 constraints for 2 trajectories.

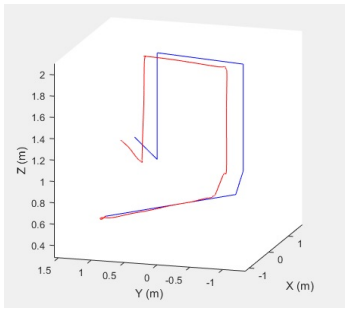
- Zero : $8 * (m-1)$ constraints for other $m-1$ waypoints

$$x_i(t_i) = x_{i-1}(t_i) = p_i;$$

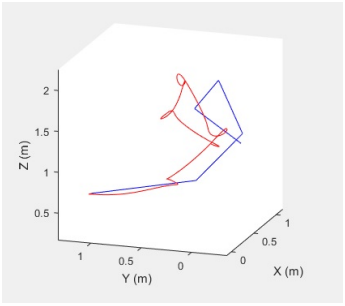
$$\dot{x_i}(t_i) = 0; \ddot{x_i}(t_i) = 0; \dddot{x_i}(t_i) = 0;$$
$$\dot{x_{i-1}}(t_i) = 0; \ddot{x_{i-1}}(t_i) = 0; \dddot{x_{i-1}}(t_i) = 0;$$

  The position should be equal, the velocity, acceleration and jerk of adjacent trajectories at waypoints should be 0.

With enough constraints, the trajectories in x coordinate are then obtained by MATLAB matrix computation $x = A^{-1}b$. Similarly, the trajectories in y and z coordinate can be obtained.



(a) Task 2's desired and actual trajectory (with zero constraints))



(b) Task 3's desired and actual trajectory (with continuity constraints)

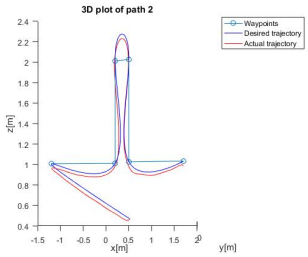
Figure 3: Minimum snap trajectories generated (blue curve)

Table 2: Minimum Snap Trajectory

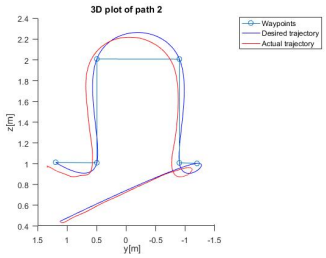
	Time planned	Time actual	Len. planned	Len. actual	Avg. velocity
Path 2	28.95	30.25	9.2709	9.2709	0.3065
Path 3	8.30	10.30	4.6356	5.4552	0.5296

4 Actual v. Desired Position Plots

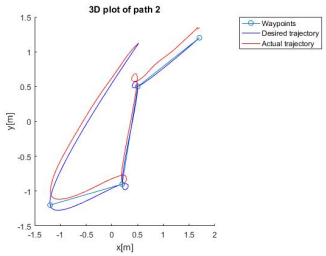
In addition to the three above 3D plots that show the trajectories, we've taken screenshots of the 2D views in the x, y, and z planes for the path 2 trajectory to more clearly show the errors in the various directions.



(a) XZ-plane view of trajectory



(b) YZ-plane view of trajectory



(c) XY-plane view of trajectory

Figure 4: Various plane-wise views of the trajectory for path 2

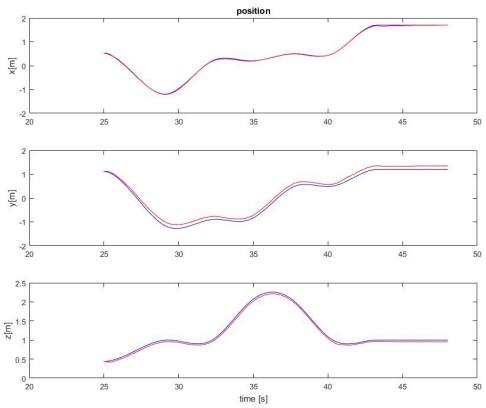


Figure 5: X,Y,Z specific trajectory tracking plots

What can be gleaned from these three plots are how the error might be different in each direction, and which type of errors might be able to be accounted for by further tuning the PD controller.

5 Discussion

5.1 Controller

Studying Figure 4 and 5, it seems that there is only a minor steady-state error, which would be adjusted best by adding an integral part to the controller and making it a PID. It also appears that there is no large deviation among the direction, but only in the xZ-plane did the quadrotor actually manage to pass through the majority of the waypoints. A possible way to address this is to tune the X and Y directions separately, as we had left the controller gains equal for the two. Looking specifically at Figure 5, we can see that tracking in the x-direction is relatively accurate, while tracking in the y-direction is not. In fact, the actual y-direction tracking consistently hovers above the expected, and would probably benefit the most from the introduction of an integral part to the controller. As for the z-direction, the quadrotor hovers below the expected height very consistently. This too would benefit from an integral term, but might also be addressed by increasing the Kp coefficient. Another possible reason for this error occurring is that the weight of the quadrotor is slightly heavier than we inputted and expected. By making fewer assumptions (X, Y directions not identical, weight should be adjusted between quadrotors), and adding an integral term, all of the errors should be fixed.

5.2 Trajectory

Different type of trajectory for the same task are different in velocity, acceleration, jerk or etc. Minimum snap trajectory generates smooth acceleration profile while minimum acceleration trajectory can change drastically in acceleration (large jerk).
With different constraints, trajectories generated are totally different. As observed in Minimum snap trajectories of task 2 and 3 (Figure 3), continuity constraints generate smooth and thus fast trajectory because there is a velocity at waypoints. In contrast, ones generated by zero constraints have collision-free guarantee and are easier to evaluate waypoints hitting.

5.3 Improvements

- For trajectories generators with continuity constraints, the generated trajectories tend to be smooth and sometime a bit away from the straight path connecting the waypoints. In this case the actual trajectories may hit obstacles(figure 2(b)). To avoid this scenario, we can add some control points along the path. To be concrete, given line segment (a, b) of path , we can add 0.1*a + 0.9*b and 0.9*a + 0.1*b to the waypoints. The control points limit the planned path of the trajectory generator to desired path at turning points and hence decrease the chance of hitting obstacles.

- Due to limited lab time, we have not fully tuned the controller gains and speed profile. However, they are crucial to the performance. We believe optimizing the controller and speed profile can dramatically increase the stability and speed.