Project Report: Quadrotor Planning and Control

Team 12: Haoxiang You, Juan Cervino, Bruce Lee, and Kun Huang

I. Introduction and System Overview

The objective for the in person lab was to implement a path planner, trajectory generator, and controller on a physical quadrotor to fly through several different environments without collision. The main difference from the previous projects in the course is that in the in person lab the robot was real as opposed to a simulated environment. This allowed our group to become familiar with the adaptations that must be made for differences between the simulated and real world environments. In particular, our group was able to identify the main changes, and adapt both the gains, and some parameters of the trajectory generation to be robust to the uncertainty present in the real world setting. Through this project, we demonstrated the robots capabilities to plan a trajectory given a map of the environment, and execute control actions to follow this trajectory at fairly high speeds (up to 3m/s for a cube trajectory!)

The experiment setup consists of a quadcopter in an environment constituted by several obstacles placed according to three provided JSON files. Put together, these JSON files described a closed trajectory which was divided into 3 sub-trajectories. These subtrajectories were executed successively – the ending point of a trajectory became the starting point of the next trajectory.

The hardware used was a crazyfile 2.0 quadrotor, an antenna, a computer, and a VICON system. The sensing is performed both by an IMU on the quadcopter, which measures both angular velocities and acclerations, and by the VICON system, which makes use of VICON tags on the quadcopter to measure the tag locations, and fuses these measurements into estimates of the position and orientation of the quadcopter. The bulk of the computation is performed by the computer, which receives position and orientation information from the VICON system. It then uses these estimtates to compute the thrust and desired orientation of the quadcopter. There are transmitted to the crazyflie, where a a low level attitude controller is used to compute the motor commands.

The information exchanged between the quadrotor, computer, and VICON system are as follows: measurements of the angular velocity and accleration provided by the onboard IMU are sent from the quadcopter to the computer, position and orientation measurements are sent from the VICON system to the computer, and thrust and desired orientation values are sent from the computer to the quadcopter.

II. Controller

The controller was composed of two loops. The outer loop captures the error between desired position, and the true position. It then generates a desired force vector aligned with this error, and a thrust given by the projection of this desired force vector onto the z-axis of the quadcopter. The force vector output by the outer loop then serves as an input to the inner loop. To ensure convergence, the inner loop should run faster than the outer loop.

The outer loop uses geometric intuition to generate the desired force vector. Ideally, the force generated by the rotors should cancel out both the gravity and the errors in the position. In particular, we have the following equations for the desired force:

$$\ddot{\mathbf{r}}_{des} = \ddot{\mathbf{r}}_T - K_d(\dot{\mathbf{r}} - \dot{\mathbf{r}}_T) - K_p(\mathbf{r} - \mathbf{r}_T) \tag{1}$$

$$\mathbf{F}_{des} = m\ddot{\mathbf{r}}_{des} + m\begin{bmatrix}0 & 0 & g\end{bmatrix}^\top, \tag{2}$$

where $\mathbf{r}_T$ is the 3D position vector of trajectory expressed in world frame, $\mathbf{r}$ is our current position expressed in world frame, m is the mass of the quadcopter in kg, and K_p and K_d are proportional and derivative gains, respectively.

The actual force in the world frame and thrust generated in the body frame are related by the orientation of the quadcopter as $\mathbf{F} = R\begin{bmatrix}0 & 0 & u_1\end{bmatrix}^\top$, where u_1 is the thrust. Therefore, the thrust should attempt to make $\mathbf{F} = \mathbf{F}_{des}$. The best we can do in this regard is to set $u_1 = \begin{bmatrix}0 & 0 & 1\end{bmatrix} R^\top \mathbf{F}_{des}$.

Seeing as we can only generate thrust along the z-axis of the quadcopter, the inner loop attempts to align the z-axis of the quadcopter with the desired thrust vector. In particular, we define the desired orientation $R_{des} = \begin{bmatrix}\mathbf{b}_1^{des}, \mathbf{b}_2^{des}, \mathbf{b}_3^{des}\end{bmatrix}$, where $\mathbf{b}^{des}$ are the coordinates of each axis in body frame expressed in world frame.

First, we let the z axis of body frame be aligned with desired force:

$$\mathbf{b}_3^{des} = \frac{\mathbf{F}_{des}}{\|\mathbf{F}_{des}\|}. \tag{3}$$

Then we can calculate the second axis by the fact that it should both perpendicular to the $\mathbf{b}_3$ and the axis

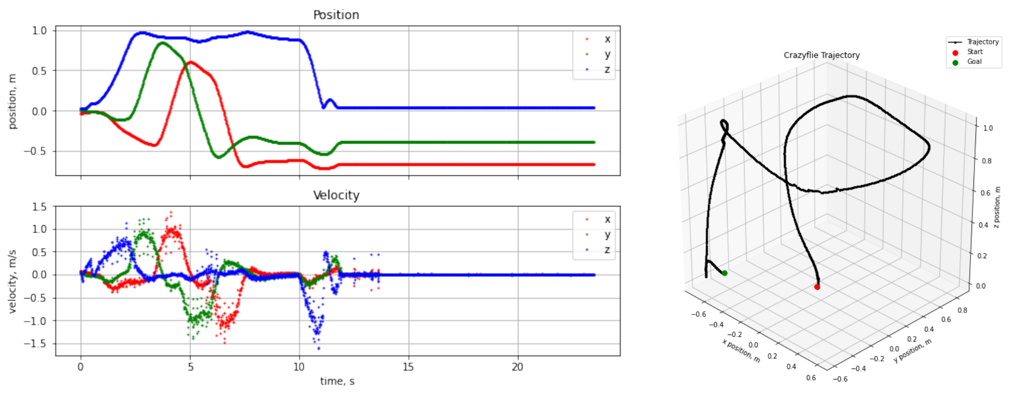

Fig. 1: Position and velocity of the quadrotor vs. time for a cube trajectory.

defined by desired yaw angle:

$$\mathbf{a}_\psi = \begin{bmatrix} \cos\psi_{des} \\ \sin\psi_{des} \\ 0 \end{bmatrix}, \mathbf{b}_2^{des} = \frac{\mathbf{b}_3^{des} \times \mathbf{a}_\psi}{\left\| \mathbf{b}_3^{des} \times \mathbf{a}_\psi \right\|}. \tag{4}$$

The remaining axis can be obtained by the cross product between the calculated axis:

$$\mathbf{b}_1^{des} = \mathbf{b}_2^{des} \times \mathbf{b}_3^{des} \tag{5}$$

After we obtained the desired rotation matrix we can compose our attitude controller. In contrast to our position controller, the attitude error is not given by difference of rotation matrix directly, because the rotation matrix has only three degrees of freedom, but is composed of nine elements. The actual orientation error is taken by the formula below:

$$\mathbf{e}_R = \frac{1}{2}(R_{des}^T R - R^T R_{des})^\vee \tag{6}$$

The moment along each axis can now be calculated as:

$$\mathbf{u}_2 = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = I(-K_R \mathbf{e}_R - K_\omega \mathbf{e}_\omega) \tag{7}$$

where $\mathbf{e}_\omega$ is the error vector for angular velocity, I is the inertial matrix in units $kg \cdot m^2$, and K_R and K_ω are proportional and derivative gains, respectively.

The values of K_d and K_p that we used are:

$$K_p = \texttt{diag}(3, 3, 3), \quad K_d = \texttt{diag}(2.75, 2.75, 2.75).$$

The unit for K_d is 1/s, the unit for K_p is $1/s^2$. As K_p captures the error for the position, it helps to reduce the rising time and steady state error. Meanwhile, K_d captures the error in first derivative term, so it helps to reduce overshot.

The gains K_R and K_ω serve a similar function to K_p and K_d, respectively. They are in units of $1/s^2$ and $1/s$ and the values used for final implementation were:

$$K_R = \texttt{diag}(1600, 1600, 900),$$
$$K_w = \texttt{diag}(79.2, 79.2, 79.2).$$

We note that the attitude controller is not directly used in the physical experiments. Instead, our thrust u_1 and a quaternion describing R_{des} are sent to a low-level attitude controller on the drone, from which motor commands are calculated and applied.

We used almost the same controller in the simulation as on the physical quadcopter. However, we did tune parameters in order to for fly the map 3 during the second part of the in-person lab, where the quadrotor worked in simulation but hit the box when implemented on the real robot. The reason for the collision was a steady state error in position. Plausible reasons may be: the motor in the real quadrotor requires more time to implement the command we give to it, whereas motors in the simulator will reach the command immediately. Another reason may be the parameters of quadrotor not exact meet what we specified in the simulator due to manufacturing problems. Indeed, the change that we made to improve the performance of the quadcopter on Map 3 was to increase the estimate of the mass used by the controller by 7.5% from what was provided.

In Figure 1 we show how the position and velocity of the quadrotor change through time in a cube trajectory flight, and the visualization of the actual trajectory. Our controller is able to effectively eliminate the steady state error along the z-axis, however, there is some shift in the x and y axes. As we discussed above, we carefully chose the control gains in order to reduce overshot, rising time and steady state error. This is supported by the response curve in Figure 1 as the result closely resembles a critical damping performance, reducing both rising time and settling time as a result.

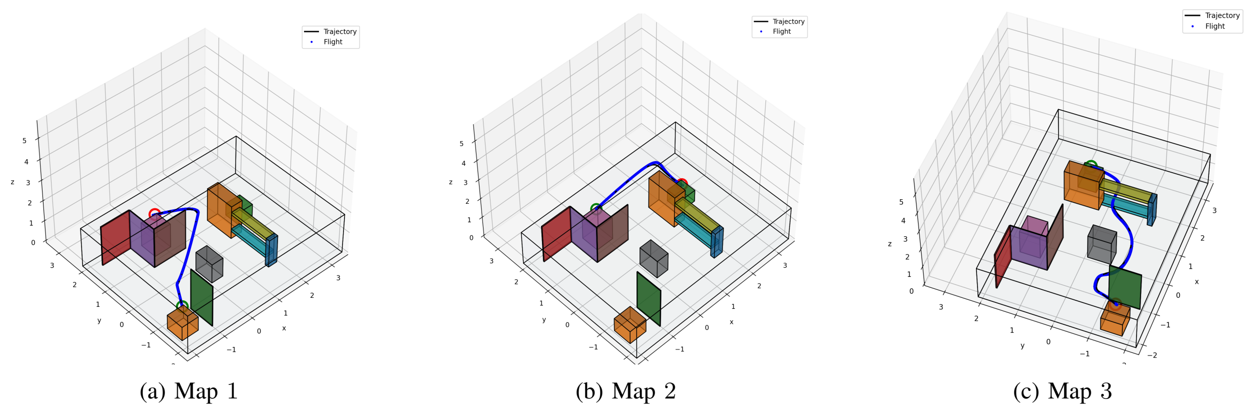

Fig. 2: Planned trajectory and true trajectory for the 3 maps from the second session in simulation.

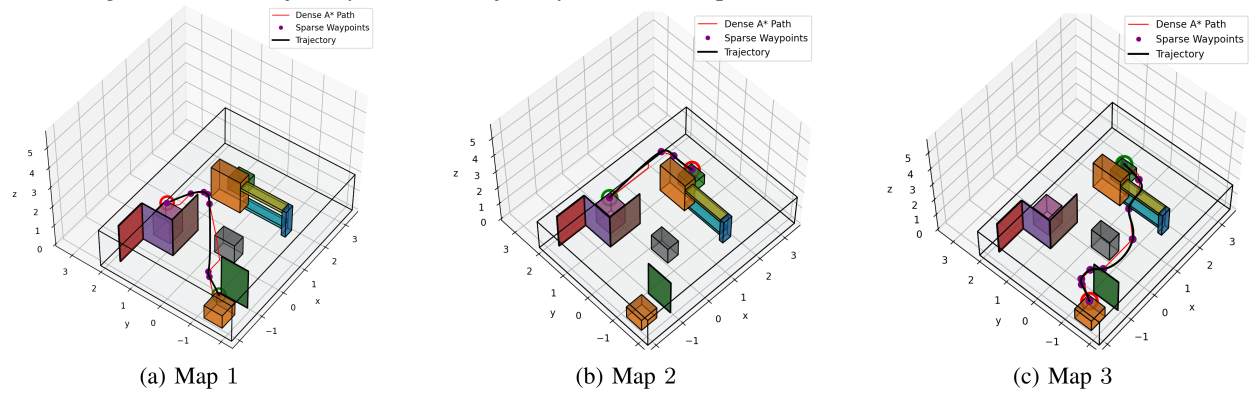

Fig. 3: Sparse waypoints, dense A^* path and planned trajectory for the 3 maps from the second session in simulation.

III. Trajectory Generator

The first step to generate a trajectory is to select some resolution and partition the map into voxels. The resolution chosen for this purpose was $[0.125, 0.125, 0.125]$ meters. Next, we create an occupancy map by marking the voxels which contain an obstacle as occupied. To account for the fact that i) the quadcopter is not a point mass, and ii) the controller will not be perfect, we expand the obstacles by some positive margin before marking the voxels. The margin used was 0.25 meters.

Given our occupancy map, the next step is to find a discrete path from the start location to the goal location. This may be done with a standard graph search algorithm, such as Djikstra's algorithm.

Djikstra's algorithm gives a very dense set of waypoints. It is difficult to plan a trajectory through this set of waypoints, motivating us to prune waypoints that give little information. A principled way to do this is with the Ramer-Douglas-Peucker (RDP) algorithm, which takes a parameter ϵ. This algorithm discards points on the line which don't change the path by more than ϵ at any point. Therefore, the resulting path has fewer waypoints, yet the path is changed very little. We set $\epsilon = 0.05$ meters.

To generate the trajectory for our sparse set of waypoints, we first set some desired velocity $v = 1\frac{m}{s}$. We then allocate the time between waypoints as the distance between waypoints divided by this velocity. We are left with a set of sparse set of waypoints $p_0, \ldots, p_n$, and desired times $t_0, t_1, \ldots t_n$ to reach them.

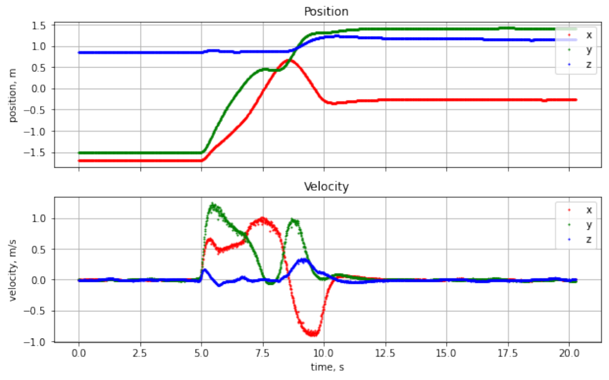

Fig. 4: Quadcopter response for the map 1 from the second session.

We attempt to find a minimum acceleration trajectory for each segment, subject to some constraints. In particular, we consider segments of the form

$$w(t) = c^w_{3,i}(t-t_i)^3 + c^w_{2,i}(t-t_i)^2 + c^w_{1,i}(t-t_i) + c^w_{0,i}$$

for $t \in [t_i, t_{i+1}]$, where t_i is the desired start time for the ith segment, and w may equal x, y, or z. Note that we solve for the x, y and z trajectories separately, however the processes are identical. We will therefore focus only on one coordinate, and drop the superscript w. The bare minimum constraints that we must consider are that the initial position is the start position, the final position is the goal position, the start and end velocities are zero, and the trajectory is continuous between segements. To ensure continuity between segments, we must have that the position and the velocity at the end of one segment are equal to the position and velocity at the start of the next segment. Combining these, we have

$$\begin{aligned}
&c_{0,0} = \texttt{start}, \quad c_{1,0} = 0 \\
&c_{3,n-1}(t_n - t_{n-1})^3 + c_{2,n-1}(t_n - t_{n-1})^2 \\
&\qquad + c_{1,n-1}(t_n - t_{n-1}) + c_{0,n-1} = \texttt{end}, \\
&3c_{3,n-1}(t_n - t_{n-1})^2 + 2c_{2,n-1}(t_n - t_{n-1}) \\
&\qquad + c_{1,n-1} = 0, \\
&c_{0,i+1} = c_{3,i}(t_{i+1} - t_i)^3 + c_{2,i}(t_{i+1} - t_i)^2 \\
&\qquad + c_{1,i}(t_{i+1} - t_i) + c_{0,i} \text{ for } i = 0, \ldots, n-2 \\
&c_{1,i+1} = 3c_{3,i}(t_{i+1} - t_i)^2 + 2c_{2,i}(t_{i+1} - t_i) + c_{1,i} \\
&\qquad \text{for } i = 0, \ldots, n-2.
\end{aligned}$$

We will denote the above constraints by $A_{eq}c = b_{eq}$. In order to avoid hitting obstacles, we must also stay close to the trajectory defined by taking a straight path between waypoints. This is handled with corridor constraints. The corridor constraints are implemented by taking 100 evenly spaced time increments τ_j between 0 and t_n. At each of these times, we constrain the coefficients such that the distance of the trajectory from the constant speed trajectory at this time is smaller than the corridor width, dentoed $\texttt{tol}$. We set $\texttt{tol} = 0.1$. This gives us the following inequality constraints.

$$\Bigg| c_{3,i}(\tau_j - t_i)^3 + c_{2,i}(\tau_j - t_i)^2 + c_{1,i}(\tau_j - t_i) + c_{0,i} - p_i + \frac{(p_{i+1} - p_i)(\tau_j - t_i)}{v} \Bigg| \leq \texttt{tol} \text{ for } j = 1, \ldots, 100,$$

where t_i is the largest segment start time smaller than τ_j. These constraints can be separated into two affine inequality constraints by requiring that both the component inside the absolute value, and the negative of that are smaller than $\texttt{tol}$. This set of constraints will be denoted $A_{ineq}c \leq b_{ineq}$ With this set of constraints, we will have an underconstrained problem, so we minimize the acceleration over the trajectory. In particular, by taking the second derivative of the trajectory, squaring it, and integrating over time, we get the objective $H(c) = \sum_{i=0}^{n-1} 4c_{2,i}^2(t_{i+1} - t_i) + 6c_{2,i}c_{3,i}(t_{i+1} - t_i)^2 + 12c_{3,i}^2(t_{i+1} - t_i)^3$. Then solving the following optimization problem provides coefficients defining a trajectory:

$$\underset{c}{\text{minimize}} : H(c) \quad \text{s.t.} \quad A_{eq}c = b_{eq},\ A_{ineq}c \leq b_{ineq}.$$

The fact that we minimize the acceleration encourages smooth position and velocity for the planned trajectory. Removing exact equality constraints at waypoints, and instead requiring that the trajectory be close to the original trajectory also encourages smoothness. The smoothness of the trajectories can be seen from the position and velocities over time on Map 1, in Figure 4.

The trajectory that we get out of the above optimization problem is feasible for each map. The reason why is that the sparse waypoints we get is close to that determined by our graph search (how close depends on the RDP parameter ϵ) and the end trajectory is close to the constant velocity trajectory through these sparse waypionts (exactly how close depends on the corridor constraint parameter $\texttt{tol}$). The values $\epsilon = 0.5$ and $\texttt{tol} = 0.1$ sum to 0.15, which is smaller than the margin passed to the graph search algorithm. This guarantees that as long as the graph search returned a feasible path, and the trajectory is well behaved in between the points where corridor constraints are enfored, we will have a feasible path. The feasibility of the paths for each map are shown in the 3D plots in Figure 2.

IV. Maze Flight Experiments

In the second session, we were able to successfully fly the quadrotor with no collision. In this section, we will show the quadrotor response for each of the 3 maps. From the 3 responses in figure 4, it can be seen that no collision took place. Note that the lines are smooth, which correspondes to a trajectory through open air. We see from Figure 4 that there is quite significant tracking errors. The error between the z position and the desired position is particularly bad, almost 0.5 meters on maps 1 and 2. This led us to increase the mass parameter for map 3, leading to a reduction in the tracking error. This information could have been used to select the margin while planning trajectories, but this would not have worked for such drastic tracking error. The trajectories could have been slightly faster, but making them too much more aggressive would have led to collisions with obstacles, as seen by the close proximity's to barriers on Map 1 and 3. One step that could have improved the speed and reliability on the maze would have been to tune the mass parameter even further so that there was even less tracking error.

V. Conclusion

To conclude, we were able to successfully fly a physical quadrotor through interesting environments. Flying the quadrotor in 3 different maps showcases our group's robot skills. With one more lab session, we think it would be exciting to try and perform the state estimation ourselves, rather than receiving accurate state estimates.