Project 1.4 Report: Quadrotor Planning and Control

Group 10: Greg Campbell, Malavika Manoj, Shall Chee Shih

I. Introduction and System Overview

The objective of this lab was to demonstrate the robustness of our simulation code on a physical quadrotor. Through projects 1.1-1.3, we developed python code for path planning, trajectory generation, and quadrotor control, and tested the code in an idealized simulation. We had also previously tested our quadrotor control on the physical system to tune the control gains. The next step taken in this lab was therefore to demonstrate that this combination of code was robust enough to work beyond simulation: in the real world.

Our code was tested on a CrazyFlie 2.0 that had its precise location monitored by an indoor VICON system. The VICON system was tracking the quadrotor location via the retro-reflective spheres and sending the data back to computer. The CrazyFlie was measuring accelerations and moments (using an accelerometer and gyroscope respectively), communicating with the computer, and applying motor inputs. The computer prior to flight was solving for desired positions at each point in time, and during flight it was comparing actual positions (from VICON) with desired positions, applying position control, and sending the desired motor inputs to the CrazyFlie. The CrazyFlie is therefore only receiving motor speeds for each discretized moment in time, and is sending back moment and acceleration measurements.

II. Controller

A. Position Controller with Equations

We decided to use a geometric nonlinear controller, which is built based on geometric intuition. The following was adapted from the project 1.1 handout [1].

We start out from the PD controller in compact vector form:

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

where K_d and K_p are diagonal, positive definite derivative and proportional gain matrices, r is the present position, r_T is the present desired position, and $\ddot{r}^{des}$ is the commanded acceleration.

Then, we calculate the total commanded force F^{des} (including gravity): $F^{des} = m\ddot{r}^{des} + [0, 0, mg]^T$.

Next, we calculate u_1 by projecting F^{des} onto the third column of the desired rotation matrix b_3: $u_1 = b_3^T F^{des}$.

To calculate u_2, we decide to align b_3 with F^{des}, and align b_1 to match the desired yaw ψ_T, since the quadrotor can only produce thrust along b_3 axis.Therefore, b_3^{des} can be calculated by: $b_3^{des} = \frac{F^{des}}{\|F^{des}\|}$.

Then, b_2^{des} can be calculated by using the cross product between b_3^{des} and a_ψ: $b_2^{des} = \frac{b_3^{des} \times a_\psi}{\|b_3^{des} \times a_\psi\|}$, where the yaw direction in the plane $a_\psi = [\cos\psi_T, \sin\psi_T, 0]^\mathsf{T}$.

Next, the desired rotation matrix R^{des} can be obtained by combining the b^{des} vectors:$R^{des} = [b_2^{des} \times b_3^{des}, b_2^{des}, b_3^{des}]$.

We measure the error in orientation $R^{des^T} \times R$ by the following error vector [2]: $e_R = \frac{1}{2}(R^{des^T} R - R^T R^{des})^V$.

At last, we get the control input:$u_2 = I(-K_R e_R - K_\omega e_\omega)$, where I is the inertia, the error in angular velocities $e_\omega = \omega - \omega^{des}$, K_R and K_ω are diagonal gains matrices. We set ω^{des} to zero in this project.

B. Control Gains

In the lab, we found out that the real environment was different from the simulator, so we tried several gains matrices to improve the performance of the quadrotor. We decided to use the following diagonal gains matrices in the lab:

$$K_d = \begin{bmatrix} 8.5 & 0 & 0 \\ 0 & 8.5 & 0 \\ 0 & 0 & 8 \end{bmatrix} K_p = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix} \tag{2}$$

As for the units, the unit of K_d is $\frac{m/s^2}{m/s}$, which can be simplified to $\frac{1}{s}$; the unit of K_p is $\frac{m/s^2}{m}$, which can be simplified to $\frac{1}{s^2}$. The units of K_R and K_ω both are $\frac{N}{kg \cdot m^2 \cdot rad/s}$. Proportional term K_p has a spring (capacitance) response, and as K_p increases, generally, rise time will decrease, overshoot will increase and steady-state error will decrease. Derivative term K_d has a dashpot (resistance) response, and as K_d increases, generally, overshoot will decrease and settling time will decrease.

C. Attitude Controller

Our attitude controller is not being used, and we used the attitude controller on-board the CrazyFile instead. We can directly command the angle, while we cannot command the moments. Generally, the attitude controller on CrazyFile is much less responsive than the attitude controller in simulator.

D. Adjustments for Differences between Simulation and Experiment

(a) Resolution changes from [0.25, 0.25, 0.25] to [0.125, 0.125, 0.125].

(b) Margin changes from 0.5 to 0.3.

(c) Max acceleration changes from 19 m/s^2 to 1 m/s^2.

We do these adjustments mainly to ensure safety, since the map in real world is different from the simulation.

E. Controller Performance

As shown in figures in part IV (Fig. 6), the steady state error in x axis and z axis is very small, but there is a small steady state error in y axis. We believe the reason is that the battery on the quadrotor changes the mass center of it, and therefore the symmetric assumption we made on the quadrotor results in some errors. To solve this problem, we can modify the position of the battery or take the battery into account when designing the controller. Besides, the damping ratio is over-damped meaning that there is no overshoot. Likewise, settling time is an insignificant measure. Rise times were low for code 2 and on the order of 0.5 second for code 1. The result demonstrates our controller's ability to adhere to smooth trajectories consistently.

III. Trajectory Generator

We were able to get the best trajectory using minimum jerk with constraints for continuity at waypoints.

Waypoints: From generated A* path, beginning from the starting point, we skipped subsequent points one at a time, densely sampling the path directly to the next new point. The absence of an obstacle on this path means we can directly connect this new point to the start point. By doing this repeatedly until a collision occurs on the sampled path, we have the farthest point(on the current A* path) to which we can connect directly from the current point. We then repeat the process from the new point, until we connect to the goal.

Time allocation: We used constant acceleration to calculate the time between waypoints, using the formula $2\sqrt{\frac{d}{a}}$, where d is the distance between the two waypoints and a is the chosen constant acceleration. This assumes low initial velocity, and could fail in extreme circumstances. When we used constant speed, the acceleration was very large for smaller paths, which made the quad unstable.

Equations: We solve for minimum trajectory with continuity of velocity, acceleration, jerk and snap at all waypoints between start and goal. We get 6m equations for a four segment path (described below), which we compose into the form Ax=B to solve. For a trajectory with 'm' segments between waypoints, A is of the order $6m \times 6m$, x and B are of order $6m \times 1$. The 6m equations

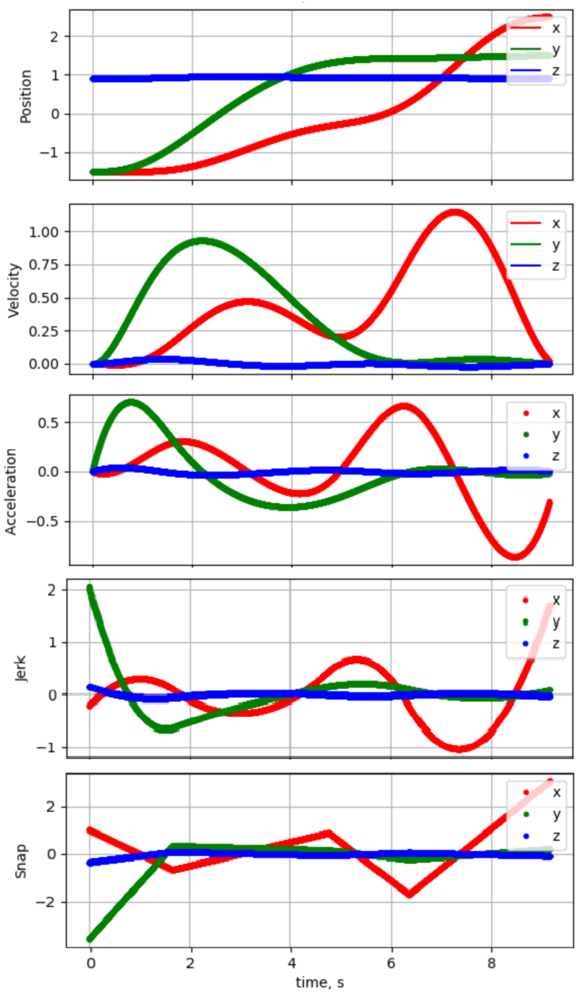

Fig. 1: Position and time derivatives up to snap (minimum jerk) in simulation

for a multi segment trajectory are described below.

Start and end positions for each segment (2m equations):

$c_{n,5}t_s^5 + c_{n,4}t_s^4 + c_{n,3}t_s^3 + c_{n,2}t_s^2 + c_{n,1}t_s + c_{n,0} = p_{nstart}$

$c_{n,5}t_e^5 + c_{n,4}t_e^4 + c_{n,3}t_e^3 + c_{n,2}t_e^2 + c_{n,1}t_e + c_{n,0} = p_{nend}$

in which, $n = 1, 2, \cdot\cdot\cdot, m$, t_s is the start time of each segment, t_e is the end time of each segment, the end position of last segment is the same as the start position of the next segment, so the continuity at waypoint between two segments is guaranteed.

Velocity continuity (m-1 equations):

$5c_{n,5}t_n^4 + 4c_{n,4}t_n^3 + 3c_{n,3}t_n^2 + 2c_{n,2}t_n + c_{n,1} - 5c_{n-1,5}t_n^4 - 4c_{n-1,4}t_n^3 - 3c_{n-1,3}t_n^2 - 2c_{n-1,2}t_n - c_{n-1,1} = 0$

Acceleration continuity (m-1 equations):

$20c_{n,5}t_n^3 + 12c_{n,4}t_n^2 + 6c_{n,3}t_n + 2c_{n,2} - 20c_{n-1,5}t_n^3 - 12c_{n-1,4}t_n^2 - 6c_{n-1,3}t_n - 2c_{n-1,2} = 0$

Jerk continuity (m-1 equations):

$60c_{n,5}t^2 + 24c_{n,4}t + 6c_{n,3} - 60c_{n-1,5}t^2 - 24c_{n-1,4}t - 6c_{n-1,3} = 0$

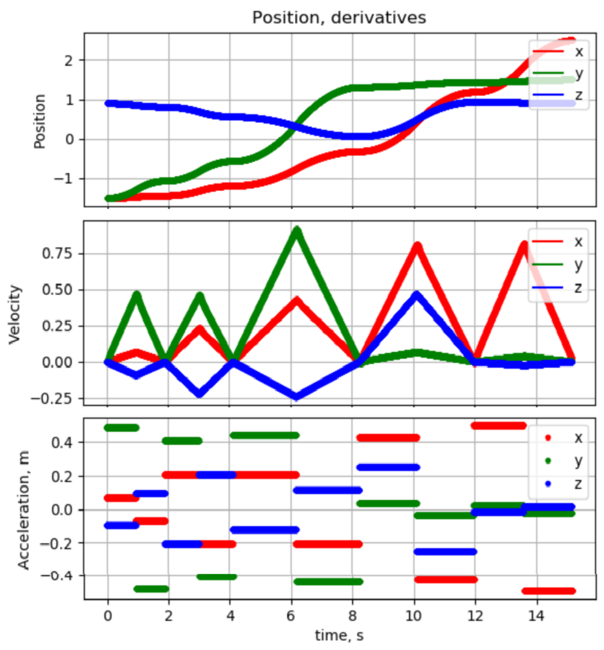

Fig. 2: Position, velocity, acceleration (constant acceleration) in simulation

Snap continuity (m-1 equations):
$120c_{n,5}t + 24c_{n,4} - 120c_{n-1,5}t - 24c_{n-1,4} = 0$
in which, $n = 2, 3, \cdots, m$.
Velocity=0 at start and goal (2 equations):
$5c_{1,5}t_s^4 + 4c_{1,4}t_s^3 + 3c_{1,3}t_s^2 + 2c_{1,2}t_s + c_{1,1} = 0$
$5c_{m,5}t_e^4 + 4c_{m,4}t_e^3 + 3c_{m,3}t_e^2 + 2c_{m,2}t_e + c_{m,1} = 0$
in which, t_s is the start time of the first segment, t_e is the end time of the last segment.
Acceleration=0 at start and goal (2 equations):
$20c_{1,5}t_s^3 + 12c_{1,4}t_s^2 + 6c_{1,3}t_s + 2c_{1,2} = 0$
$20c_{m,5}t_e^3 + 12c_{m,4}t_e^2 + 6c_{m,3}t_e + 2c_{m,2} = 0$
in which, t_s is the start time of the first segment, t_e is the end time of the last segment.
For the trajectory generated from these equations, we get smooth profiles for velocity, acceleration and jerk as shown in Fig. 1. All derivatives upto snap are continuous and crackle becomes discontinuous.

For trajectory generated for minimum jerk, stopping at all waypoints, we strictly follow the generated A* path, while the smooth profiles generated with the above equations deviates from the A* path when smoothing the trajectory. We also ran another trajectory in lab, which was derived using constant acceleration and requiring the quadrotor to stop at each waypoint. We get continuous velocity and discontinuous acceleration for this trajectory, as shown in Fig. 2.

IV. Maze Flight Experiments

Our group ran two separate sets of code to navigate the maze. The first set of code (hence force "code 1") stopped at each waypoint and then designated a constant acceleration, while the second set of code ("code 2") designated a constant snap throughout. It is worth noting that a bug was introduced into to trajectory generator of code 1 prior to its running, leading to non-ideal pathing; it was still able to complete the maps. The controllers in each set of code were identical. Three-dimensional plots of each map for each code can be found in Fig. 3 through Fig. 5 below. The path goes from the green circle to the red circle.

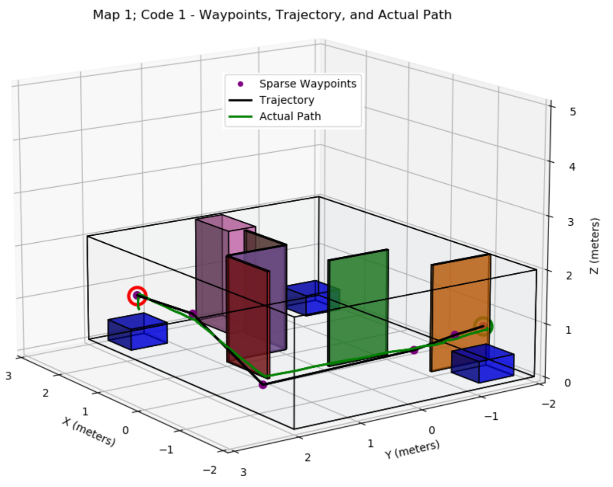

Fig. 3: 3D plot for map 1 using code 1.

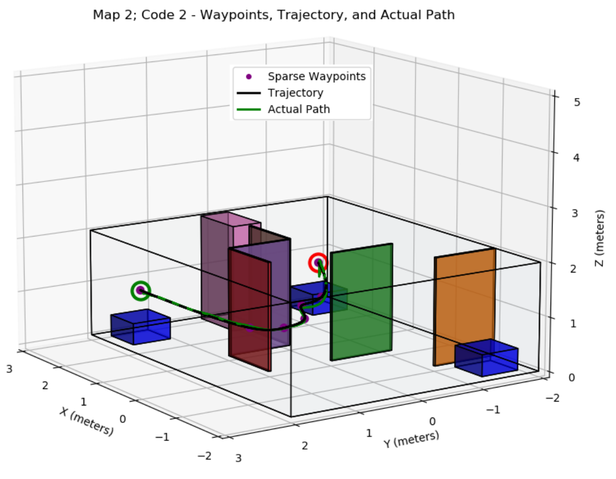

Fig. 4: 3D plot for map 2 using code 2.

It is apparent in the three-dimensional plots that the real quadrotors followed the planned trajectories reasonably well. However, there was some deviation. This deviation is explored quantitatively using Fig. 6 below, which compare the actual position and velocities of the quadrotors navigating map 1 to the planned trajectories.

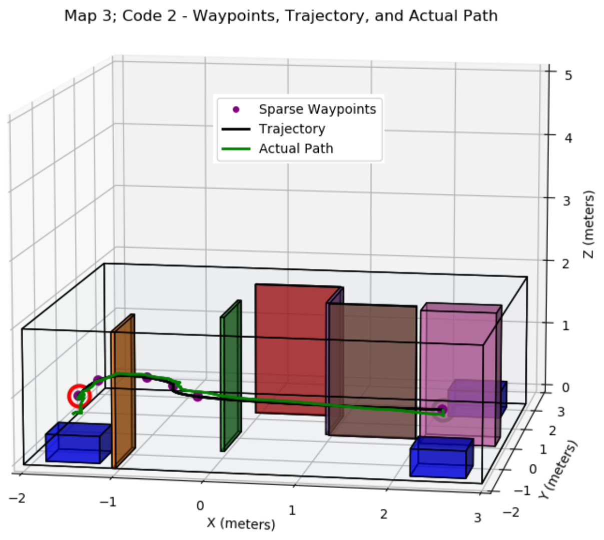

Fig. 5: 3D plot for map 3 using code 2.

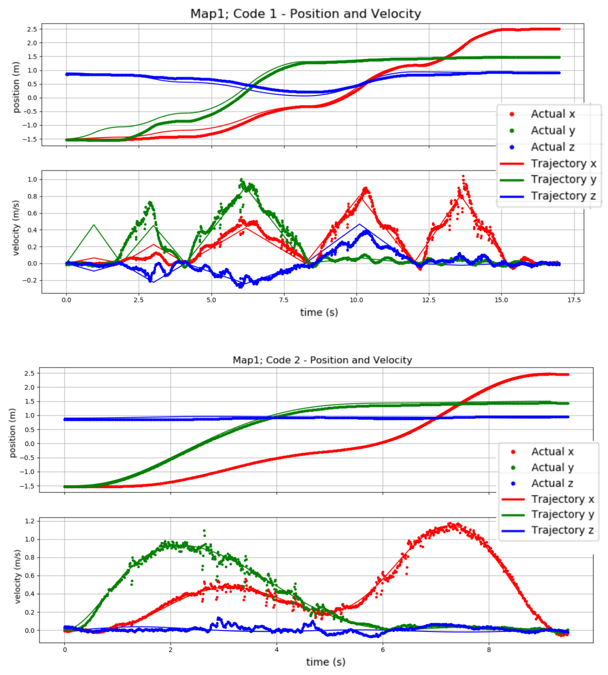

Fig. 6: Position and Velocity data for each set of code navigating Map 1.

Code 1 may have had a different first waypoint, or fails to appreciate the y component of the required move, as it doesn't make any change in the y direction on its first move. It then overcompensates with the y velocity change on the second move to catch back up. This leads to an error of about 0.5 meters over the first 2.5 seconds. The z error is minimal until the quadrotor starts dropping low, at which point the quadrotor stays too high. This could be due to ground effects, which would provide additional lift that was not accounted for in the controller. The maximum z position and x position error were less than 0.25 meters.

Code 2 tracked the requested trajectory much better. This could be because it wasn't asking for infinite jerk at transition points (which code 1 did). After a small lag in y velocity at the beginning of the route, code 2 followed the trajectory positions and velocities nearly perfectly. The maximum error was approximately 0.1 meters in the y and z positions, and practically nonexistent in the x position.

Given the errors for each of the codes, it is clear that using continuous snap (code 2) is a superior option for reducing error. Also, having an optimal trajectory could play a role in this improvement. Assuming that we use code 2, we could assume that the quadrotor always stays within 0.1 meters of the planned trajectory. We could therefore adjust our robot margin to the robot radius plus this envelope of 0.1 meters on each side for a total margin of 0.45 meters. At this envelope size, we would expect to be fully safe from all obstacles.

For code 2 in particular, the trajectory could definitely have been more aggressive in terms of a higher acceleration. The fear with larger accelerations is that it could lead to a higher error, as error based on lag times would be scaled directly. That being said, there was no point where the quadrotor seemed near to colliding. I would be wary flying code 1 much faster, as it was close to bottoming out at one point.

Plenty of changes could be made to code 1 to improve its performance. First off, the bug in the trajectory generation has already been fixed so that it uses an ideal trajectory. Furthermore, code 1 stops at each waypoint by design. This requirement causes the quadrotor to lose a lot of momentum throughout the flight. Code 2 seems pretty near ideal. If there were some way to increase the transmission speed of data that could potentially drive down lag times. The best way to decrease flight time would be to choose a more aggressive acceleration.

If we were to return to lab, we would want to run code 2 more aggressively to try and decrease flight time. We'd also be interested in trying to relax constraints (maybe allow the quad to fly over obstacles?) to see if there could be even faster or more direct routes to get from start to finish, given only the physical limitations of the room.

References

[1] MEAM 620 Team, "MEAM 620 Project 1 Phase 1: Modeling and Control of a Quadrotor," January 29, 2020

[2] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in Proc. of the IEEE Int. Conf. on Robotics and Automation, Shanghai, China, May 2011