MEAM 620 Project Report: Quadrotor Planning and Control

Team 11: Dingding Zheng, Shane Rozen-Levy, Huaiyu Chen

I. Introduction and System Overview

In this project, we designed algorithms to control a quadrotor. Controlled by a geometric non-linear controller, the quadrotor could follow a predefined trajectory and reached the goal without collision. We got the optimal trajectory by applying Down-sampling algorithm (shown in section. III) to the shortest path which was generated by A*. This lab focused on demonstrating the planning aspect of robot capabilities. During the lab, the CrazyFlie 2.0 was used for realistic demonstrations. A microcomputer is responsible for low-level control and estimation, while the onboard IMU provides feedback of angular velocities and accelerations. The attitude and thrust commands are sent to the quadrotor via the CrazyRadio after being computed in python.

II. Controller

A. Introduction

In this project, we implemented a Geometric Non-linear controller. This kind of controllers perform well with aggressive maneuvers since these maneuvers have high-speeds and sharp turns.

The position PD controller in vector form can be written as [1]:

$$\ddot{\mathbf{r}}^{\text{des}} = \ddot{\mathbf{r}}_T - K_d\left(\dot{\mathbf{r}} - \dot{\mathbf{r}}_T\right) - K_p\left(\mathbf{r} - \mathbf{r}_T\right) \tag{1}$$

K_p and K_d are diagonal, positive definite gain matrices. K_p represents gain for proportional part, which measures how fast a system responses to a input error. A large K_p may cause the system to be unstable. K_d is the derivative gain. The derivative of the process error is calculated by determining the slope of the error over time and multiplying with K_d. Thus, a large K_d improves the settling time and system stability. During the experiment, we set K_p and K_d as: [8.5, 8.5, 10.0] $(1/s^2)$ and [3.0, 3.0, 5.0] $(1/s)$.

During the experiment, we found that K_p and K_d should be a bit smaller compared to ones in simulation. This is probably because for realistic use, the quadrotor may not be able to fly as you wish. We know that by setting large K_p and K_d, the quadrotor will get large inputs u_1 and u_2 which intend to drive itself aggressively. But there exists some hardware limitations in CrazyFlie 2.0, such as maximum speed of the motors. To deal with this problem, it's better to tune down values of K_p and K_d for good flight performance.

After calculating the desired acceleration using equation 1, we calculated the target force using equation 2. Finally in order to get the commanded force, we project $\boldsymbol{F}^{des}$ into quadrotor z direction, thus $u_1 = \boldsymbol{b}_3^T \boldsymbol{F}^{des}$.

$$\boldsymbol{F}^{des} = m\ddot{\boldsymbol{r}} + \hat{e}_3 mg \tag{2}$$

From here we calculated the desired orientation for the quadrotor using equation 6 where ψ_T is the target yaw angle.

$$\boldsymbol{b}_3^{des} = \frac{\boldsymbol{F}^{des}}{||\boldsymbol{F}^{des}||} \tag{3}$$

$$\boldsymbol{a}_\psi = \begin{bmatrix} \cos\psi_T \\ \sin\psi_T \\ 0 \end{bmatrix} \tag{4}$$

$$\boldsymbol{b}_2^{des} = \frac{\boldsymbol{b}_3^{des} \times \boldsymbol{a}_\psi}{||\boldsymbol{b}_3^{des} \times \boldsymbol{a}_\psi||} \tag{5}$$

$$R^{des} = [\boldsymbol{b}_2^{des} \times \boldsymbol{b}_3^{des}, \boldsymbol{b}_2^{des}, \boldsymbol{b}_3^{des}] \tag{6}$$

During the experiment, the desired yaw angle was always set as 0. In this case, the quadrotor remained pointing to one direction. Additionally, we did not use our attitude controller in the hardware experiments because the CrazyFlie 2.0 was running its own internal attitude controller and thus took in as commands a target attitude and thrust.

B. Testing

For controller testing, we first chose a single point [0.5, 1.0, 2.0] (m) and let the robot "jump" to that point. The testing plots are provided in Fig. 1. This data is from simulation. The bags we collected in lab did not do a good job demonstrating the step response of the system because the trajectory generator we used in lab 1 was continuous in position and velocity with a small target acceleration.

As we can see from the plots: The system is underdamped on x and y direction ($0 < \zeta < 1$), while it's critically damped on z direction ($\zeta = 1$). In the x, y direction the rise time is about 1 sec, while the settling time is about 1.5, sec. In the z direction the rise and settling time are around 2.5-3 sec. Additionally, the system converges with zero steady state error.

In order to test the performance when controller dealing with multi-points, we chose a waypoint list as: [[0.1, 0.1, 1.0], [0.1, 1.0, 1.0], [1.0, 1.0, 1.0], [1.0, 0.0,

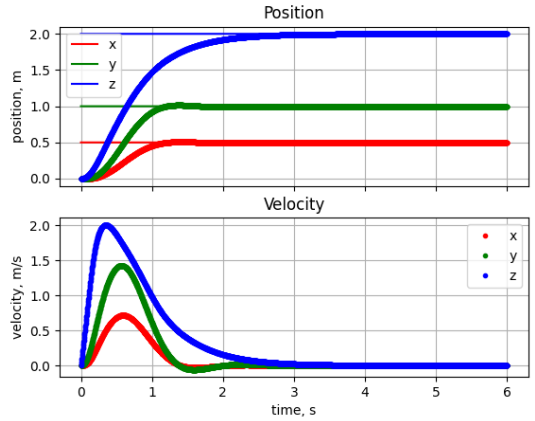

Fig. 1. Position & Velocity vs Time (single point). From simulation.

1.0], [0.1, 0.1, 1.0]] (m). We saved the bag file for this test and drew plots. The plots for multi-points testing are shown in Fig. 2.

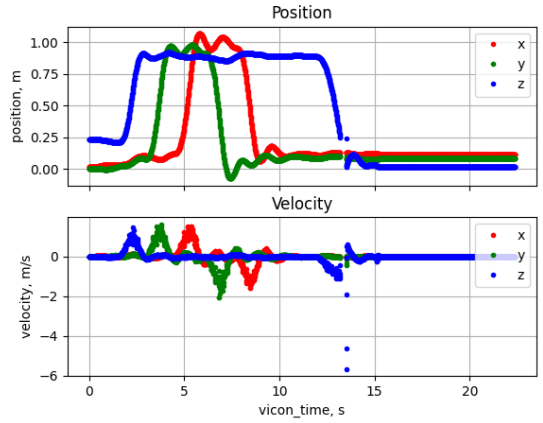

Fig. 2. Position & Velocity vs Time (multi-points) From experiment.

Given a series of waypoints, the controller successfully drived the quadrotor to desired places. Overall, the flight performance was great.

III. Trajectory Generator

After finding a path using A* through the map [2], we down-selected points using the algorithm 1. From there we used a bang bang controller with an acceleration of $0.5m/s^2$. Thus the time between points, $T = 2\sqrt{d/a}$ where d is the distance and a is the acceleration.

The equations for the trajectory are in equation 9 where $\tau := t - T/2$. Since this trajectory is a straight line through space, in order to use these equations with the quadrotor, we simply multiplied $x(t)$, $\dot{x}(t)$, $\ddot{x}(t)$ by the unit vector describing the direction from one way point to the next way point.

Algorithm 1: Down-sampling Algorithm

Result: Down sampled list of points with no collision

Let U be the dense list of points outputted by the path finding algorithm and let V be the output list of points initially empty;

i = 2, j = 0 ;

Add U_0 to V ;

while *i < length (U)* **do**

  if *is collision between* U_i *and* V_j **then**

    add U_{i-1} to V ;

    j ++; i++;

  else

    i++;

  end

  add U_{end} to V

end

$$\ddot{x}(t) = \begin{cases} a & t \leq T/2 \\ -a & T/2 \leq t \leq T \\ 0 & t \geq T \end{cases} \tag{7}$$

$$\dot{x}(t) = \begin{cases} at & t \leq T/2 \\ aT/2 - a(\tau) & T/2 \leq t \leq T \\ 0 & t \geq T \end{cases} \tag{8}$$

$$x(t) = \begin{cases} 1/2at^2 & t \leq T/2 \\ -1/2a\tau^2 + aT/2\tau + d/2 & T/2 \leq t \leq T \\ d & t \geq T \end{cases} \tag{9}$$

The resulting trajectory has continuous position, and velocity in $\mathbb{R}^3$ and has finite, though discontinuous, accelerations. This results in infinite jerk and snap commands. Fortunately with a low enough commanded acceleration, the quadrotor could handle the infinite jerk and snap. Figure 3 demonstrates the continuous nature of the position and velocity trajectories generated by bang bang control.

The infinite jerk and snap trajectory is however beneficial, in that the quadrotor always travels in straight lines and is much less likely to deviate from the A^* path and hit obstacles. In comparison, a polynomial trajectory, although smoother, requires either post processing of the trajectory or more map resolution to guarantee obstacle-free trajectories in real time (as opposed to simply changing parameters in simuation). Both options can be computationally expensive, depending on the dimensionality and size of the map. For this reason we chose bang bang control over spline fit for trajectory generation.

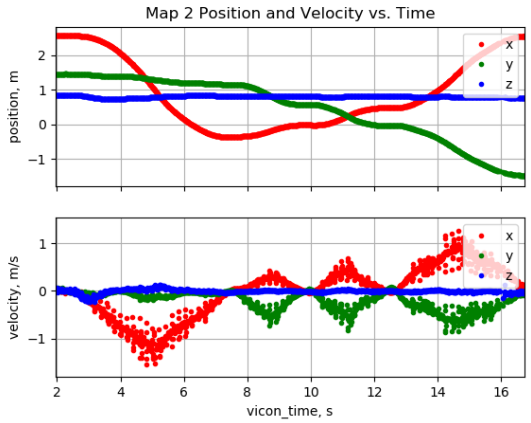

Fig. 3. Position and Velocity vs. Time for Map 2. Notice how the position and velocity are continuous, while the acceleration is discontinuous, though finite.

IV. Maze Flight Experiments

A. Test 1

In Map 1, the quadrotor starts from [-1.5, -1.5, 0.9] (m) and ends at [2.5, 1.5, 0.9] (m). The experiment plots are shown as Fig. 4 and Fig. 5.

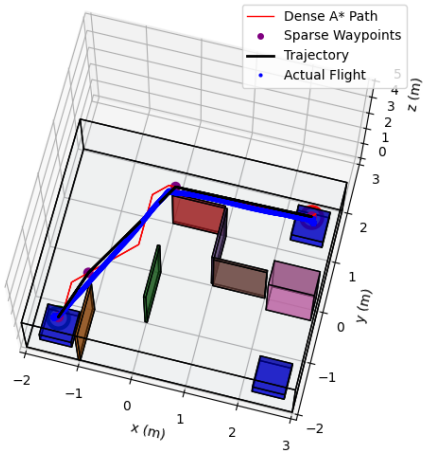

Fig. 4. 3D Path, Waypoints, Trajectory, Actual Flight for Map 1

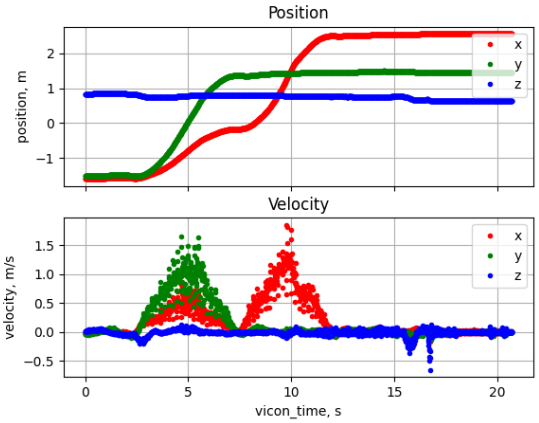

Fig. 5. Position and Velocity vs. Time for Map 1.

B. Test 2

Start: [2.5, 1.5, 0.9] (m), Goal: [2.5, -1.5, 0.9] (m). Plots are shown as Fig. 6 and Fig. 7.

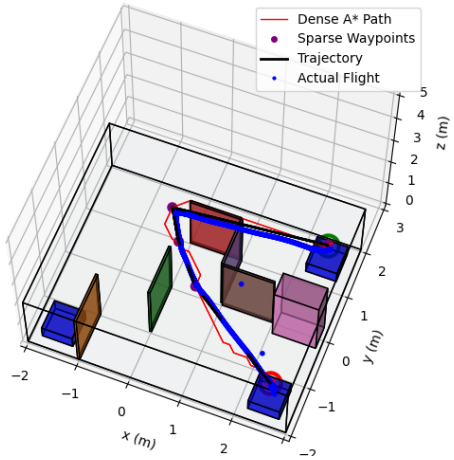

Fig. 6. 3D Path, Waypoints, Trajectory, Actual Flight for Map 2.

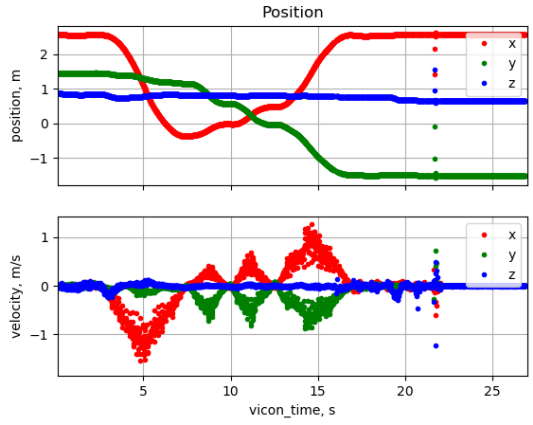

Fig. 7. Position and Velocity vs. Time for Map 2.

C. Test 3

Start: [2.5, -1.5, 0.9] (m), Goal: [-1.5, -1.5, 0.9] (m). Plots are shown as Fig. 8 and Fig. 9.

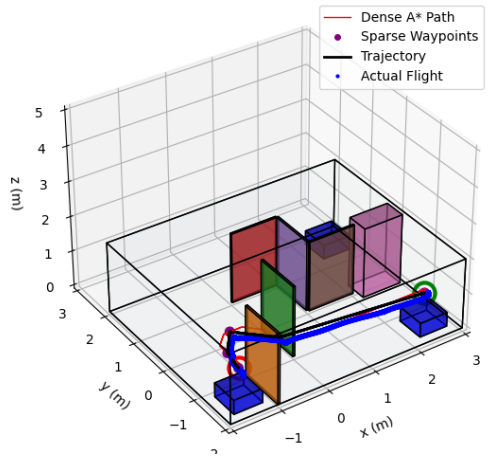

Fig. 8. 3D Path, Waypoints, Trajectory, Actual Flight for Map 3.

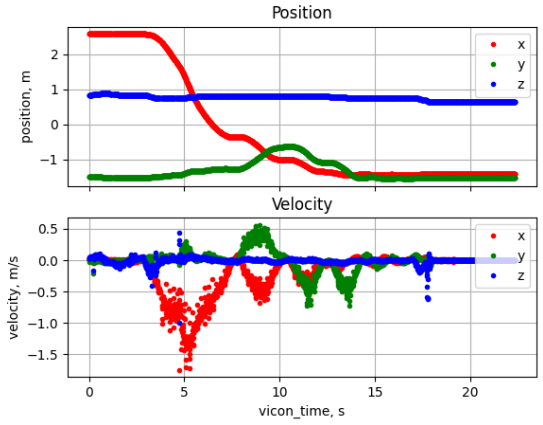

Fig. 9. Position and Velocity vs. Time for Map 3.

D. System Analysis

For clearer system performance analysis, we computed distance for desired trajectory and actual flight to get the tracking error. The tracking errors are computed in an accumulative way and can be view as: Table I.

Map	Desired Distance (m)	Actual Distance (m)	Error (m)	Percent Error
1	6.0851	6.6142	0.5291	8.0 %
2	7.7684	11.9351	4.1667	34 %
3	5.2915	5.5247	0.2332	4.2 %

TABLE I

TRACKING ERROR TABLE.

The error increases with larger flying distance and more sharp turns. The large error in distance traveled for map 2 is likely due to an issue with the Vicon system. The Vicon reported a speed of 50 m/s at t=22 sec. We believe that the quadrotor was not moving at 50 m/s and this was simply an erroneous measurment (possibly the Vicon picked up a different quadrotor at that time). Overall, our quadrotor's doing a fantastic job following the desired trajectory.

E. Future Work

Our trajectories can easily be more aggressive. In simulation the quadrotor can fly through these maps with an acceleration of $3.5m/s^2$ instead of $0.5m/s^2$.

In order to speed up the trajectory, we can use trajectory optimizaion using direct collocation. Although this may greatly increase the solve time due to having to solve a nonlinear program, direct collocation can optimize for time while allowing for curved trajectories. Direct collocation can also allow for more constraints being set such as region keepout at the obstacles and jerk limits.

We can also use a different trajectory planner. Bang bang control is a minimum time controller but in this setting where we have multiple waypoints, it may be slow because the quadrotor has to stop at each waypoint. A minimum jerk or minimum snap planner may result in smoother trajectories that the quadrotors can follow with less errors. The trajectories can also be more aggressive where the overall speed is faster. The trade-off here would be more computation devoted to guaranteeing no collision.

If we want to make the bang-bang trajectory more reliable, the easiest manner will be to add jerk limits to the trajectory while keeping the zero end velocity. There are still closed form solutions for this situation and the trajectories can still be straight lines in $\mathbb{R}^3$.

If we have one more section for this lab, it will be interesting to implement jerk limits to the trajectory and see how the finite jerk affects the tracking error with the same controller. Alternatively we can let our quadrotor fly through more interesting obstacles such as a hula-hoop.

F. Reference

[1] 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.

[2] T. Lee, M. Leok, and N. McClamroch, "Geometric tracking control of a quadrotor uav on SE(3)," in Proc. of the IEEE Conf. on Decision and Control, 2010.