# MEAM 620 Project Report: Quadrotor Planning and Control

TEAM 18: Yilan Lu, Yuqi Chen, Ye Duan, Rohit Bhikule
E-mail: elainelu@seas.upenn.edu, yuqichen@seas.upenn.edu, duanye@seas.upenn.edu, rbhikule@seas.upenn.edu,

## I. INTRODUCTION AND SYSTEM OVERVIEW

The purpose of the in-person Crazyflie labs is to give us the chance to implement the algorithms we wrote for Lab 1.1, 1.2 and 1.3 on a real robot. In project 1.1, we got the controller to allow us to order the quadrotor to fly to a certain destination. In project 1.2, we got the trajectory generator to compute the optimal paths by implementing two graph search algorithms: Dijkstra's algorithm and A*. In project 1.3, we improved the algorithm from 1.1 and 1.2 and put them together to allow the quadrotor to fly through a 3D environment with obstacles. Finally, in these in-person Crazyflie labs, we implemented all these things on a real quadrotor. We tuned the gains in the controller and parameters in the trajectory generator for the real quadrotor in the first lab. And we implemented all things to allow the quadrotor to fly through a 3D environment with obstacles in real life.

The robot capabilities we have demonstrated throughout both sessions are: plan an optical path due to the 3D environment with obstacles (which could avoid collision with a certain safe distance) and fly through the environment with the order from the lab computer. In this lab, our quadrotor could operate very smoothly and agilely to complete the flight mission. The quadrotor we used in these labs is Crazyflie 2.1. Some brief specifications are: takeoff weight is 27g, size (WxHxD): 92x92x29mm (motor-to-motor and including motor mount feet) and maximum recommended payload weight is 15g. For more detail, we could find the specifications on this website [1]. There are many cameras mounted on the roof of the lab room, they could capture the motion of the quadrotor by detecting the Vicon sensors mounted on the Crazyflie 2.1. These motion data would be sent back to the computer and help the system to control. For computation, data would be sent to the server of the lab computer and the server would do the computation. To control, the Vicon system would allow us to know the position of the quadrotor. Due to the position, the computer server would do computation and give orders to the quadrotor to control. Besides, about the lab computer, the operating system is Ubuntu Linux OS.

## II. CONTROLLER

### A. Introduction

For this project, we implemented a Geometric Non-linear controller due to its ability to perform well under aggressive maneuvers since these maneuvers have a lot of high speed and sharp turns. The basic intuition behind this controller is tilting the quadrotor's axis to point in the desired direction and then applying the thrust. The position PD controller is given by:

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

$K_p$ and $K_d$ are diagonal, positive definite gain matrices. $K_p$ is the Proportional gain. It indicates how fast a system responds to minimize the error of the dynamics. If the value of $K_p$ is too high, the system may become unstable. $K_d$ is the derivative gain. The derivative of the error is calculated by finding the slope of the error over time and multiplying it by $K_d$. $K_d$ helps in improving the settling time and stability of the system. A large value of $K_d$ may cause the system to become overdamped and hence increase the settling time. Hence, we need to find the optimal values for the gains so that the system converges to stability as fast as possible. After several iterations of trial and error, we found the optimal values for $K_p$ as $[18, 18, 22.5](1/s^2)$ and for $K_d$ as $[7.3, 7.3, 9](1/s)$. These values were tuned after reducing the original values which were used in the simulation. Higher $K_p$ and $K_d$ values would have led to higher control inputs which may cause the Crazyflie quadrotor to become unstable. Also, for high control input, there were limitations for hardware such as the motor speed. To avoid this issue, the values were reduced a bit and then tuned to get a better performance.

After calculating the desired acceleration, we calculate the thrust force $F^{des}$ using the following equation.

$$F^{des} = m\ddot{r}^{des} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

We project this thrust force in the quadrotor Z-axis direction. Hence, we get

$$u_1 = b_3^T F^{des}$$

.

After calculating the command force, we calculate the desired orientation of the quadrotor $R^{des}$.

$$b_3^{des} = \frac{F^{des}}{||F^{des}||}$$

$$a_\psi = \begin{bmatrix} \cos\psi_T \\ \sin\psi_T \\ 0 \end{bmatrix}$$

$$b_2^{des} = \frac{b_3^{des} \text{ x } a_\psi}{||b_3^{des} \text{ x } a_\psi||}$$

$$R^{des} = [b_2^{des} \text{ x } b_3^{des}, b_2^{des}, b_3^{des}]$$

In the above equations, $\psi_T$ represents the desired yaw angle which was kept as zero as to point the quadrotor in one direction. After calculating the desired orientation, the error was calculated between the desired orientation and the current orientation. This was also done for the angular velocity. The gains for these dynamics were tuned during the simulation in

sandbox.

$$e_R = \frac{1}{2}(R^{des^T}R - R^T R^{des})^\wedge$$
$$u_2 = I(-K_R e_R - K_\omega e_\omega)$$

The command quaternion $q^{des}$ can be obtained by transmitting $R^{des}$ into a quaternion.
Furthermore, we did not use our altitude controller on the Crazyflie quadrotor as it was running its own internal altitude controller.

### *B. Testing*

For testing and tuning the controller, we made the quadrotor to jump vertically to the point [0, 0, 1] first to tune the gain values in the z direction. The initial starting point was [0, 0, 0]. The Position vs time plot is shown below. As we can see from the plot, the system is underdamped $(0 < \zeta < 1)$ in z direction as there is some overshooting. Also, there is some steady-state error as we can see a small difference between the desired position and the actual position of the quadrotor. In the z-direction, the rise time and settling time is approximately between 3.5 to 4 seconds.

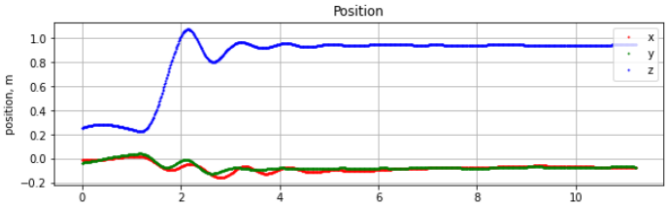


Fig 1: Position vs time plot for tuning gains in z direction

The same above process was carried out to tune the gains in the x and y direction. There is some steady-state error in the x and y direction as well. We can see that the system is underdamped as there are oscillations present. To reduce these oscillations, we increased the value of $K_d$ and decreased the value of $K_p$ a little bit to find the optimal tradeoff.

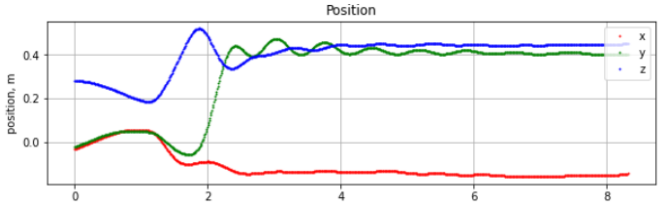


Fig 2: Position vs time plot for tuning gains in y direction

After the tuning was completed, we tested the performance of our controller with multiple points. The list of waypoints which was given was $[[0,0,0],[0,0,1],[0,1,1],[1,1,1],[1,0,1],[0,0,1]](m)$. We saved the bag files for this test and the plots are given below.

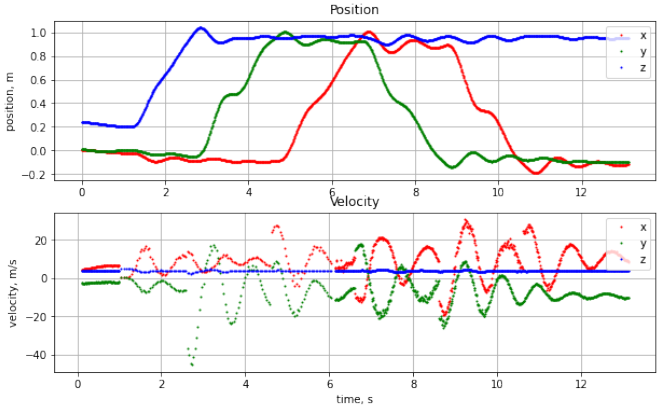


Fig 3: Position and Velocity vs time plot for multipoint trajectory

Given the waypoints, the quadrotor was succesfully able to drive through all the points. The performance was also good.

## III. TRAJECTORY GENERATION

### *A. Waypoint generation*

After generating the trajectory using $A^*$ algorithm, the points were down-sampled. The extra points on a straight line were removed. If there was no occupied block between the former and latter neighboring point of a point, it was deleted. This is done by checking the occupancy every 0.01 m on the connection line. Next, the parameters margin and resolution were decided. $margin + 2 * resolution$ should be less than the height and width of a window (0.7m for Lab 1.4) or there will be no available trajectory. It was also inferred that margins should be large for safer flight. The value of resolution is a tradeoff between generation speed and path length. The final choice for Lab 1.4 is $margin = 0.29\ m$ and $resolution = 0.1\ m$.

### *B. Time allocation*

The value of the constant acceleration was taken as $1.5\ m/s^2$. The time between each two points is $\sqrt{\frac{2d}{a}}$ where $a$ is the acceleration and $d$ is the distance. This value is only used for allocating time, it does not mean that the quadcopter is flying at constant acceleration.

### *C. Interpolation*

Cubic interpolation was used to generate functions between $x$, $y$ and $z$ depending on $t$. The constraints are given below.

$$f_{i-1}(t_i) = f_i(t_i) = x_i$$
$$f'_{i-1}(t_i) = f'_i(t_i)$$
$$f''_{i-1}(t_i) = f''_i(t_i)$$

There is no extra constraint on the starting point and end point, so the function generated will make the quadrotor unable to follow from the start and shoot over the ending point. To make the starting and ending more smooth, we multiplied the starting and second to last part of trajectory time by 2 and divide the acceleration and velocity by 2. This ensured

smoothness in the trajectory following.

$$f_x(t) = f_1(0.5t).....t < t_1$$

$$f_x(t) = f_n(0.5(t - t_{n-1}) + t_{n-1}).....t_{n-1} < t < t_n$$

And for the last point, the rest of the trajectory becomes hovering around the end point. The plots of position and velocity are shown below. From the plots, we can see that they are continuous and smooth. Although there are small jumps causing snaps, it is much smaller than using constant velocity.

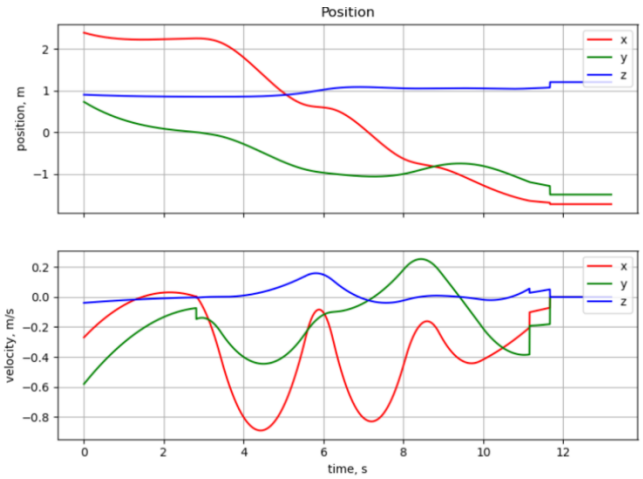


Fig 4: Position and Velocity vs time plot for polynomial trajectory

## IV. MAZE FLIGHT EXPERIMENTS

### A. Test 1

In map 1, the quadrotor starts from $[-1.73, -1.5, 1.17](m)$ and ends at $[-0.212, 1.341, 1.644](m)$ as shown in the map.

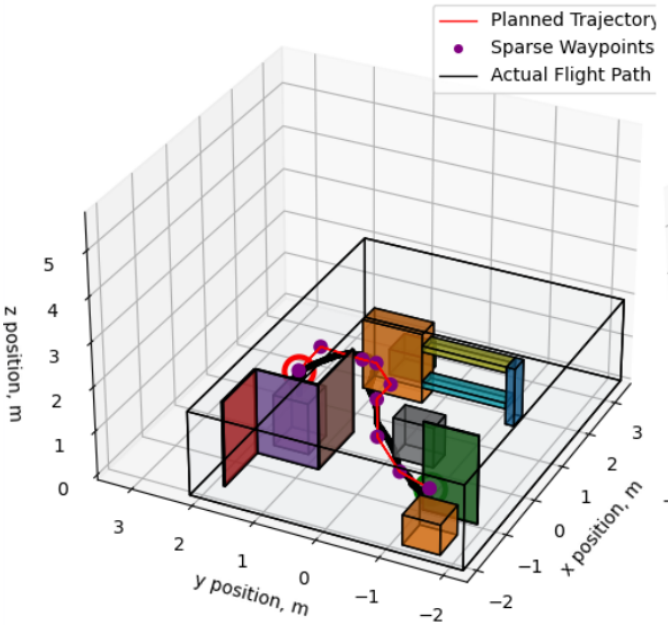


Fig 5: 3D Plot of World Obstacles, Planned Trajectory, Waypoints and Actual Flight Path for Maze Map 1

### B. Test 2

In map 2, the quadrotor starts from $[-0.212, 1.341, 1.644](m)$ and ends at $[2.386, 0.727, 0.898](m)$.

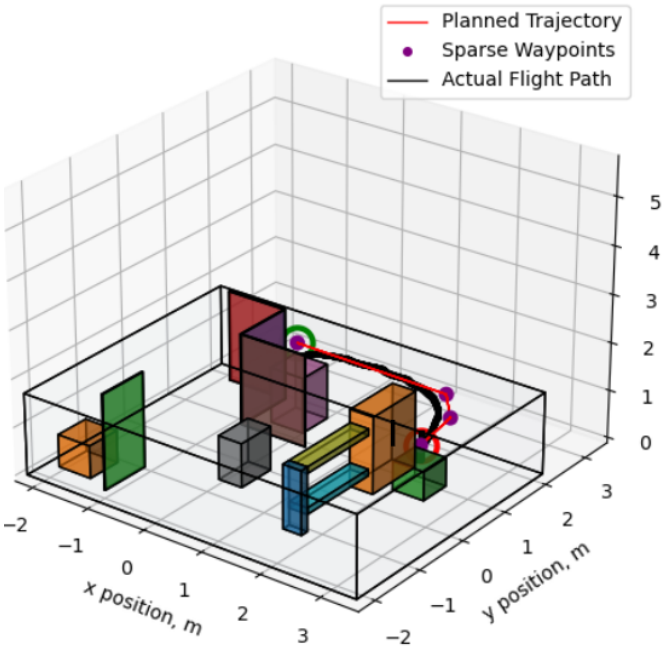


Fig 6: 3D Plot of World Obstacles, Planned Trajectory, Waypoints and Actual Flight Path for Maze Map 2

### C. Test 3

In map 3, the quadrotor starts from $[2.386, 0.727, 0.898](m)$ and ends at $[-1.73, -1.5, 1.2](m)$.

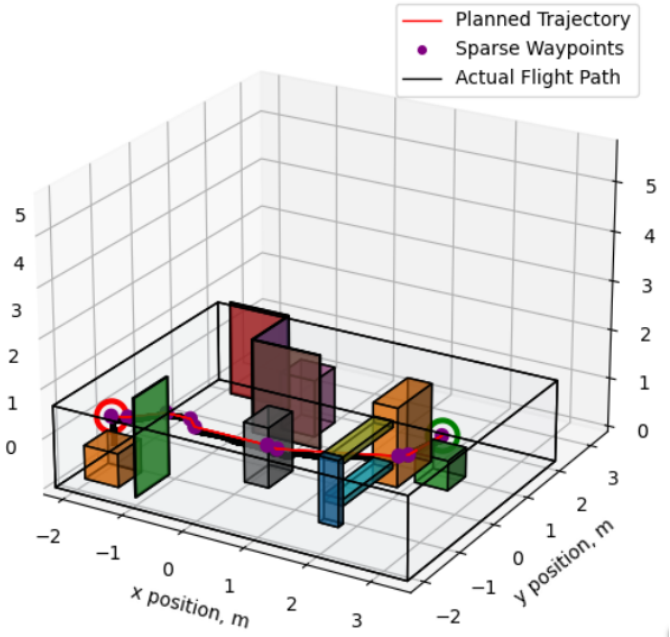


Fig 7: 3D Plot of World Obstacles, Planned Trajectory, Waypoints and Actual Flight Path for Maze Map 3

The plots for position and velocity are given below.

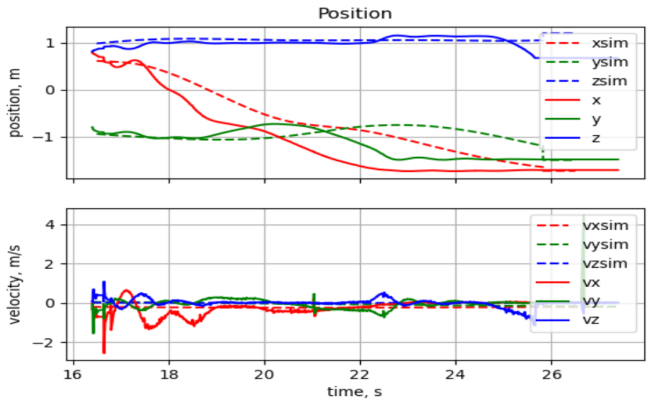


Fig 8: The plot of Position and Velocity(also include simulated position/velocity) vs Time Using Maze Map 3

Based on the observations from the above plots, one can observe that the tracking errors exist but it is relatively small, and the existing tracking errors have no obvious effects on the overall stable performance of the quadrotor. One can also observe that the x and y direction contains some small overshoots(tracking errors). Based on these observations, a more safe trajectory can be planned by decreasing the x and y direction proportional gain and increasing the x and y direction derivative gain parameters in the controller.

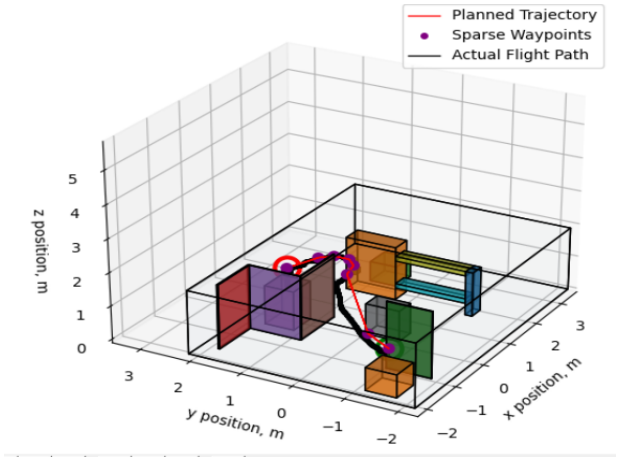


Fig 9: 3D Plot of Maze Map 1 with Acceleration to be 1.7 (Speed Up)

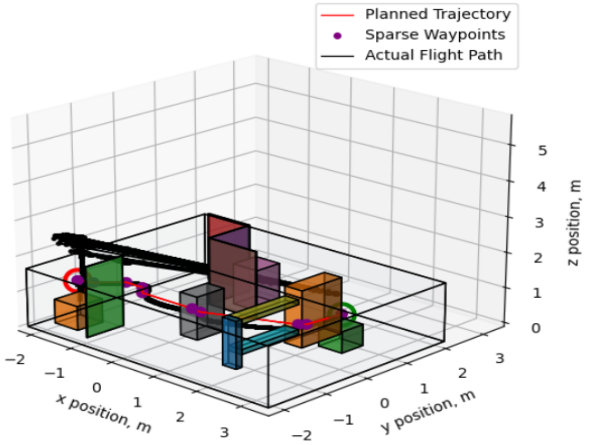


Fig 10: 3D Plot of Maze Map 3 with Margin to be 0.3 and Resolution to be 0.1(smaller resolution and larger margin)

As the above plot shows, we can actually make the trajectories to be more aggressive by increasing the quadrotor acceleration to 1.7, decreasing its resolution to 0.1, and decreasing the margin at the same time. Because the quadrotor can travel faster by increasing its acceleration, the quadrotor may also find a shorter or more optimal trajectory by using a smaller resolution and be closer to the obstacles(thus more aggressive) by implementing a smaller margin. Also, as the plot in figure 9 shows, we can actually increase the acceleration of the quadrotor to 1.7 to make the quadrotor travel from the start to the goal point more rapidly(obvious speed up). Figure 10 above has some noisy lines besides the normal flight path and I think this is not because the code to generate 3D plots is problematic but because the source .bag file contains some unnecessary data. And these data may come from incorrect capture or save procedures we conducted during the experiment. We also noticed from figure 9 that when the original margin is 0.2, the quadrotor is too close to the purple obstacle in the middle of the map. So as the above plot shows, we decrease the resolution to 0.1 to generate a potential more safe map and at the same time increase the margin to 0.3 to make the quadrotor leave more space with the purple obstacles in the middle of the map to get a more reliable path. And it is verified by observing the corresponding plot in this question. If I can have more time with this lab, I would want to have a try on tuning parameters including increased acceleration(speed up the quadrotor), manipulate with gain parameters(increase control stability of the quadrotor), decrease resolution(generate more optimal path), and decrease margin(potential shorter/faster path) to detect the maneuver limit of the quadrotor with our own designed controller, path planner, and trajectory generator codes.

## V. CONCLUSIONS

In this project, we have successfully implemented a Geometric Non-linear controller which enables the quadrotor to follow a desired trajectory. The position control gains were tuned for the Crazyflie quadrotor using several trial and error iterations. We have also successfully implemented $A^*$ algorithm which finds a path through an environment filled with obstacles. The waypoints were down-sampled and a polynomial trajectory was fitted through the reduced waypoints to obtain smooth trajectory. All these tasks were first carried out in the simulator and then later implemented practically on the Crazyflie quadrotor.

## REFERENCES

[1] https://www.bitcraze.io/products/crazyflie-2-1/

[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.

[3] 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.