MEAM620 Project1.4 Report

Group 5: Xiatao Sun, Aadith Kumar, Rithwik Udayagiri, Francis Sowande

I. System Overview

This project aims to test the control, path planning, and trajectory generation algorithms developed in Project 1 for a quadrotor to navigate through a deterministic cluttered environment. The first session of this project demonstrated the abilities of generating a trajectory based on given start and goal positions in the environment, and controlling the quadrotor to track the generated trajectory. The second session extended this to planning an optimal path in a complex maze, generating and executing a trajectory based on the path with no collision with the environment.

The quadrotor used in this project is CrazyFlie 2.0, which has an onboard IMU for angular velocity and acceleration feedback and a micro-controller for estimation and low level control such as attitude. Vicon, which is a camera-based motion capture system, is used as a sensing device to track the position of the quadrotor in the environment. The lab computer is used as the computation device for path planning, trajectory generation, and high level commands such as position control. The communication from the computer to the quad-rotor is performed through CrazyRadio, which is a 2.4GHz radio.

II. Controller

The controller used in this project is a non-linear geometric controller. This controller tries to minimize error in the quad-rotor position and orientation using PD feedback control structure.

For the position controller, it firstly calculates the commanded acceleration using Eqn. 1. In the equation, $k_{d,i}$ and $k_{p,i}$ are tuned control gains. $r_{i,T}$, $\dot{r}_{i,T}$, and $\ddot{r}_{i,T}$ are from the $flat\_output$ dictionary outputed by the trajectory generator. r_i and $\dot{r}_i$ are estimated displacement and velocity of the quadrotor respectively.

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

As shown in Eqn. 2 the total commanded force is calculated by the addition of forces from commanded acceleration and gravity according to the Newton's Second Law of Motion.

$$F^{des} = m\ddot{r}^{des} + [0 \quad 0 \quad mg]^T \quad (2)$$

This controller utilizes the intuition of b_3, which is the axis pointing upward in the body-fixed frame $\mathcal{B}$, heading towards the target direction and applying thrust. b_3 is obtained using Eqn. 3, where R is the quaternion of the present state in rotation matrix format.

$$b_3 = R\,[0 \quad 0 \quad 1]^T \quad (3)$$

The input u_1 is calculated using Eqn. 4

$$u_1 = b_3^T F^{des} \quad (4)$$

The most recently tuned control gains are:

$$K_p = [1.8, 1.8, 2.5] \qquad K_d = [4, 4, 3]$$

Each element in each vector represent the control gain on each axis sequentially. Because K_p is the coefficient of the error between the present displacement and the desired displacement with the unit as m, and the control equation for the desired acceleration outputs acceleration with the unit as m/s^2, the unit of K_p should be $1/s^2$. Also, K_d is the coefficient of the error between the present velocity and the desired velocity with the unit as m/s, the unit of k_d should be $1/s$.

Inside the control equation, K_p is the proportional term, and K_d is the derivative term. K_p has a capacitance response. By increasing it, the rise time and steady-state error will decrease, and the overshoot will increase. K_d has a resistance response. By increasing it, the overshoot and settling time will decrease.

Although the position control is transmitted to the quadrotor using a 2.4GHz radio, the attitude control is executed onboard. The attitude loops and position loops are executed using different tasks to achieve different rates. This is to ensure the attitude controller being considerably faster than the position controller. As the illustration at the beginning of this section, the thrust is applied to the desired direction that b_3 should be pointing to. To make b_3 align with the desired direction and minimize the error as much as possible, the attitude controller has to run extremely fast. Therefore, the attitude controller runs on the microcontroller of the quadrotor with an order of magnitude faster than the position controller rather than being used directly to solve the system's underactuation.

III. Trajectory Generation

The first task in generating a smooth and efficient trajectory, was ensuring we could find the shortest path between our goal, and start positions without colliding into any of the surrounding the objects, and to

accomplish this we implemented an A* graph search algorithm. When implementing the A* algorithm, we first discretized the space, by breaking it up into a finite set of voxels of uniform dimension, with a neighbor referring to any surrounding voxel. Each voxel was stored as a dictionary with its unique 3D position as the index, and its cost, parent, and a Boolean to indicate its whether it had been explored or now. A voxel could only be visited if the Boolean indicated it hadn't been visited already, if it didn't go beyond the allowed boundaries, and if it was unoccupied by any obstacles. To ensure we were always checking the smallest value, we stored each node in a heapq by cost, which automatically organized every node in increasing numerical order of costs; ensuring the node at the top was always the one with the smallest cost. With this we were able to extract a list of coordinates our drone would need to visit to reach the goal.

With a set of points obtained, the next steps was to ensure the robot visited each of them in an efficient manner, and to accomplish this we implement a minimum jerk approach after pruning the path. To prune the path from a dense set of points to sparse way points we simply choose every third point from the trajectory keeping the *start* and *goal* fixed. Through the sparse way points we generate the trajectory.

For the minimum jerk trajectory we need a polynomial of degree 5 to represent every segment between the way points, as seen in equation 5.

$$p_i(t) = t^5 c_{i,5} + t^4 c_{i,4} + t^3 c_{i,3} + t^2 c_{i,2} + t c_{i,1} + c_{i,1} \tag{5}$$

Here i is the segment number. Since we have 6 unknowns per segment we have a total of $6m$ constraints, where m is the number of segments.

The constraints are:

- Boundary constraints, 6:

$$\begin{aligned} p_1(0), &\text{ initial position} \\ \dot{p}_1(0), &\text{ initial velocity} \\ \ddot{p}_1(0), &\text{ initial acceleration} \\ p_m(t_m), &\text{ final position} \\ \dot{p}_m(t_m), &\text{ final velocity} \\ \ddot{p}_m(t_m), &\text{ final acceleration} \end{aligned}$$

- Position constraints, $2m - 2$:

$$p_i(t_i) = p_{i+1}(0), \text{ way point positions}$$

- Intermediate continuity constraints, $4m - 4$:

$$\begin{aligned} \dot{p}_i(t_i) &= \dot{p}_{i+1}(0) \\ \ddot{p}_i(t_i) &= \ddot{p}_{i+1}(0) \\ \dddot{p}_i(t_i) &= \dddot{p}_{i+1}(0) \\ \ddddot{p}_i(t_i) &= \ddddot{p}_{i+1}(0) \end{aligned}$$

$$\text{Total constraints} = 6 + 2m - 2 + 4m - 4 = 6m$$

Solving these constraints for each segment individually is a tedious task and thus we use matrices to perform these operations.

$$Ab = C \tag{6}$$

The dimension of each matrix in equation 6 is:

$$\begin{aligned} dim(A) &= 6m \times 6m, \text{ matrix of equations} \\ dim(b) &= 6m \times 1, \text{ unknown coefficients vector} \\ dim(C) &= 6m \times 3, \text{ constraints matrix for x,y and z axes} \end{aligned}$$

Once we solve for b we would have generated a smooth minimum jerk trajectory passing through each way point.

Constructing matrix A, b and C can be challenging and in equations from figure 1 we show the same for a trajectory with 3 way points - {*start, intermediate point, goal*}.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & & & & & & & & \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & & & & & & & & \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & t_2^5 & t_2^4 & t_2^3 & t_2^2 \\ t_2^1 & 1 & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 5t_2^4 & 4t_2^3 & 3t_2^2 & 2t_2^1 \\ 1 & 0 & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 20t_2^3 & 12t_2^2 & 6t_2^1 & 2 \\ 0 & 0 & & & & & & & & \\ t_1^5 & t_1^4 & t_1^3 & t_1^2 & t_1^1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & & & & & & & & \\ 5t_1^4 & 4t_1^3 & 3t_1^2 & 2t_1^1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & & & & & & & & \\ 20t_1^3 & 12t_1^2 & 6t_1^1 & 2 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & & & & & & & & \\ 60t_1^2 & 24t_1^1 & 6 & 0 & 0 & 0 & 0 & 0 & -6 & 0 \\ 0 & 0 & & & & & & & & \\ 120t_1^1 & 24 & 0 & 0 & 0 & 0 & 0 & -24 & 0 & 0 \\ 0 & 0 & & & & & & & & \end{bmatrix}$$

$$b = \begin{bmatrix} c_{1,5} & c_{1,4} & c_{1,3} & c_{1,2} & c_{1,1} & c_{1,0} & c_{2,5} & c_{2,4} & c_{2,3} & c_{2,2} \\ c_{2,1} & c_{2,0} & & & & & & & & \end{bmatrix}^T$$

$$C = \begin{bmatrix} p_1(0) & \dot{p}_1(0) & \ddot{p}_1(0) & p_2(t_2) & \dot{p}_2(t_2) & \ddot{p}_2(t_2) & p_1(t_1) & p_2(0) & 0 & 0 \\ 0 & 0 & & & & & & & & \end{bmatrix}^T$$

Fig. 1: Matrices for equation 6

To calculate the time between two way points we use a dynamic approach to set an average velocity between the two points based on the distance between them, as shown in equation 7. The average velocities we used for different distances are:

- First and last 3 segments - $\bar{v} = 0.7m/s$
- Segment distance: $d <= 0.24$ - $\bar{v} = 0.8m/s$
- Segment distance: $0.24 < d < 0.35$ - $\bar{v} = 0.9m/s$
- Segment distance: $d >= 0.35$ - $\bar{v} = 1m/s$

$$t_i = d/\bar{v}_i, \; i = \text{segment number} \tag{7}$$

IV. Results

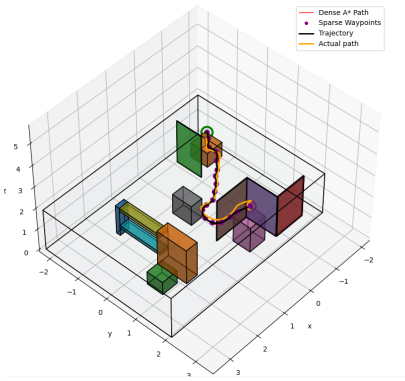

Fig. 2: Map 1

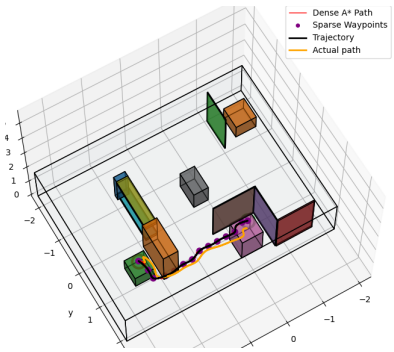

Fig. 3: Map 2

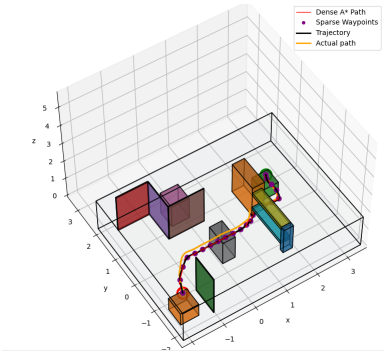

Fig. 4: Map 3

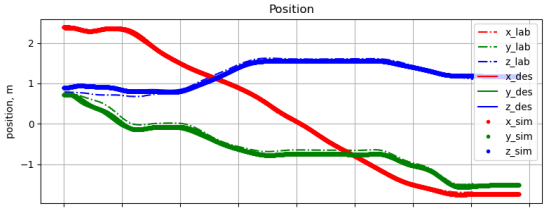

Fig. 5: Position tracking for Map 3

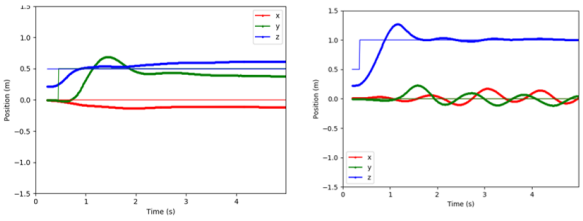

Fig. 6: Step responses

As seen in figure 5 we obtained a typical tracking error of $2cm$

V. Simulation to Lab

The first session required us to test basic motion of the drone and perform simple paths such as tracking the edges of a cube in space. We were able to observe the system behavior for basic step responses. The step responses from Lab 1 can be seen in figure 6 and estimations of relevant values in table I.

	Steady state	T_R	T_S	η	Delay
Step y	0.1	0.4	1.5	0.15	0.23s
Step z	0	0.4	1.5	0.1	0s

TABLE I: Characteristic values of step response

There were several things in our experiments that highlighted how actual testing differed vastly from simulation and we used our knowledge in control system designs to interpret these differences and address them accordingly. In the following section, we will list them with our interpretation of their causes and solutions.

1) Drone Veering out of Control on any command
   Observation : The drone quickly stumbled out of control for any commanded non-zero input to the controller.
   Interpretation : The controller was very aggressive, overcompensating for any slight error from desired position or orientation. Further, there was a time delay between commanded states and actual states because real systems have delay in their responses. In simulation the drone worked as the actuators we able to achieve rapid shifts and operate completely in saturation. But this causes trouble in real systems as time delay tends to destabilize systems (moves the poles towards the positive side of the plane).
   The time delay can be seen clearly in the step response and even in the maze navigation between commanded orientation and observed orientations. Operating at saturation often leads to situations where the drone is not able to generate some extra instantaneous thrust/moment than what the simulation accounted for. Imperfect modelling of

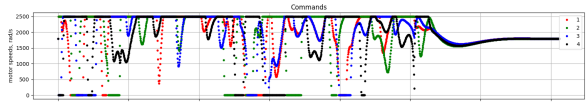

Fig. 7: Initial Gains

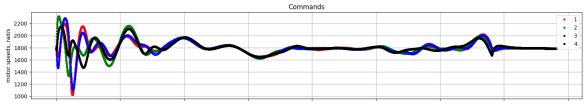

Fig. 8: Final Gains

dynamics or external disturbances could make the system demand more command value to maintain stability but as the system is already saturated it cannot provide that.

Solution : We needed to make the control less aggressive, allowing the system to operate slower, where the small-time delay has a smaller impact. Further our control gains needed to be reduced such that we could get desired behavior without saturating the controller even in simulation.

The difference in the operating ranges of the actuator can be seen for our initial and final gains (both from simulation) in figures 8 and 7.

2) Steady State error from Trajectory

Observation : The drone on hover, or simple trajectory tracking itself exhibited a steady state error. Particularly in the second lab, the error was significant. Increasing gain values reduced error but did not remove it. This error was somewhat constant across the trajectory.

Interpretation : As the controller did not have an integral component, steady state error could not be eliminated. That would also explain how Increasing proportional gains reduced the error by a little but also tended to saturate the controllers and show more aggressive behaviour. Simulation did not show significant error

Solution : Hypothetically, We could add an integral component to work on the error but that would have been more involved, and would bring a host of issues like integrator windup which we will need to address. Since the error was easy to observe and model, we used a feed forward structure where we modelled the inverse of the error based on observation and modified the control input to account for that directly. In this case, we had a larger error in the Z axis, which was constant with a value around 0.3 m. Thus, we could model our controller to always track a position with a Z offset of +0.3. We also played on the trade-off between aggressive control and error to get an acceptable error margin yet well-tuned, robust controls.

3) Dynamics changing with time and trial

Observation : With different trials, we could get slightly different results. The changes were often more significant if different drones were used. In simulation, every trial had identical responses.

Interpretation : The actuator dynamics and the drone dynamics is not constant and is subject to wear and tear, damage and other non-modellable characteristics. One example is the battery, when the drone had low battery, there was a slight difference in the power output capacity of the actuators. Similarly, when the drone was switched or the drone experienced damage/collisions, the mechanics would change by a small amount, leading to different results. In simulation these factors are modelled as constant if at all.

Solution : As these are non-certain, non-linear parameters, there is no direct solution. The best thing is to test and fine-tune existing values of gain, and error models until we get a good result in the lab.

VI. Further Work

At this point we have a system that can plan in a predefined environment and navigate safely through that. Our controller works well but is not robust to steady state error that could be caused by sensor readings and drone model inconsistencies. For further work, we would have liked to add an integral component in the controller to eliminate that error. Further, we could extend path planning in an dynamic environment if we have an ability to read the state of the environment live - using d* for example. There are other control schemes we could design and compare performances as well!

VII. Reference

[1] A. Hsieh, "MEAM 620 Project 1 Phase 1" canvas.upenn.edu, Jan. 19, 2022. [Online]. Available: https://canvas.upenn.edu/courses/1636758/files/folder/Projects/Project%201_1?preview=105918694.

[2] A. Hsieh, "06 control" canvas.upenn.edu, Feb. 1, 2022. [Online]. Available: https://canvas.upenn.edu/courses/1636758/files/folder/Lectures?preview=106389698. [Accessed Mar. 19, 2022].

[3] A. Hsieh, "10 piecewise trajectory" canvas.upenn.edu, Feb. 1, 2022. [Online]. Available: https://canvas.upenn.edu/courses/1636758/files/folder/Lectures?preview=106915045