MEAM 620 Project 1.4 - CrazyFlie Lab Report

Team 11: Ayush Goel, Irene Grace, Kausik Sivakumar, Zhijun Zhuang

I. Introduction and System Overview

The CrazyFlie labs give us a platform to test the working of our implemenation of controller and path planning on a real experimental set up. The in-person labs contain two sessions, with an object to demonstrate that:

1) We are able to use the CrazyFlie safely under the supervision of TA.
2) We can generate a dynamically feasible, obstacle-free trajectory real world maze environment.
3) We can use our controller to follow the generated trajectory stably.

In this lab, CrazyFlie 2.0 is used as the drone platform for our controller and trajectory generator. The Vicon system and one onboard IMU on the CrazyFlie are used to get the current position, attitude, velocity and acceleration of the drone.

First, a dynamically feasible trajectory are generated. Then, the position and attitude information from the Vicon system are sent to the lab computer. With the current position and attitude of the CrazyFlie and the reference trajectory, the controller will compute and output the command parameters. After that, The CrazyFlie received the command from the lab computer, including the total thrust from the motor and the moments about the body frame.

II. Controller

We used a geometric controller to determine the control commands (thrust and moment) that help it track the desired trajectory. We use a PD controller to get the accelerations and moments required to reach a particular position and orientation and hence, the performance of our system depends on how well we tune the K_p and K_d gains for the position and attitude controllers.

This non-linear geometric controller is built on intuition that we tilt the body-fixed b_3 axis along the desired direction for applying thrust. For position control, the desired acceleration vector is computed as,

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

where r_T is the positions from the way-points trajectory. The total command force F^{des} is calculated as,

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

The input u_1 is obtained by projecting F^{des} onto b_3 as

$$u_1 = \mathbf{b}_3^T \mathbf{F}^{des}$$

To compute the moments input u_2 we need to make sure that b_1 is aligned with the desired yaw angle ψ_T. To find error in the attitude, we find error in the rotation matrix space, so we first find the R^{des} as $R^{des} = [\mathbf{b}_1^{des}, \mathbf{b}_2^{des}, \mathbf{b}_3^{des}]$. We already found $\mathbf{F^{des}}$ in the direction of $\mathbf{b}_3^{des}$ so we have,

$$\mathbf{b}_3^{des} = \frac{\mathbf{F^{des}}}{||\mathbf{F^{des}}||}$$

The yaw direction vector in the global frame is given by,

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

and we need $\mathbf{b}_2^{des}$ to be perpendicular to both $\mathbf{b}_3^{des}$ and a_ψ which gives us,

$$b_2^{des} = \frac{b_3^{des} \times a_\psi}{\left\|b_3^{des} \times a_\psi\right\|}$$

By the properties of rotation matrices, we get $\mathbf{b}_1^{des}$ as the cross product as shown below,

$$R^{des} = [\mathbf{b}_2^{des} \times \mathbf{b}_3^{des}, \mathbf{b}_2^{des}, \mathbf{b}_3^{des}]$$

The error in orientation is computed as shown below where V is the vee operator,

$$e_R = \frac{1}{2}\left(R^{des^T} R - R^T R^{des}\right)^V$$

The input u_2 is computed as,

$$u_2 = I\left(-K_R e_R - K_\omega e_\omega\right)$$

where $e_\omega = \omega - \omega^{des}$. We take ω^{des} as zero for the purpose of the experiment.

The final gains we arrived at are given below and they are dimensionless gains that were obtained using rigorous tuning, first on simulation and then on experiment.

$$K_p = diag([5.0, 5.0, 8.0])$$
$$K_d = diag([4.0, 4.0, 7.0])$$
$$K_R = diag([480, 480, 240])$$
$$K_\omega = diag([70, 70, 50])$$

K_p determines how strongly the controller responds to the error, while K_d determines the strength of the response to changes in the error over time. Higher K_p leads to a faster response, but too high can cause instability. Higher K_d results in a more rapid response to changes in error, but too high can cause high-frequency oscillations and instability. The attitude controller is not actually used directly in this project. Instead, we run the attitude controller in as an inner loop of the overall controller in the Crazyfile.

Some of the adjustments that were made to account for the change from the simulation to experiment were as follows:

- Reducing position and attitude control gains by atleast 30%.
- Reduce the speed of the quadcopter and less than 1m/s.
- Adding mass offset to account for force imbalance in Z-direction when compared to simulation.

There are several plausible reasons for these differences. One reason could be that the simulation did not accurately capture all of the real-world factors that can affect the behavior of the quadcopter, such as wind or sensor noise. Another reason could be that there were differences in the hardware used in the simulation and the physical experiment, which could affect the quadcopter's behavior. Additionally, there may have been differences in the initial conditions of the simulation and the experiment, which could affect the behavior of the quadcopter as well.

We tuned our controller specifically for the experiment by explicitly giving step references. We started with the Z-step to tune the K_p and K_d for the position controller. After getting satisfactory responses, we moved on to step along x and y to further fine tune the K_p and K_d giving us the trajectory and aggressiveness desired by our controller. Here, we have shown the step-y response that was obtained in lab 1 after tuning. The 2nd order error dynamics parameters are calculated as shown in the fig 1. The steady state error is found as 0.045m. The overshoot is found to be 0.15m so the percentage overshoot is found as $PO = \frac{0.15}{0.455} \times 100 = 32\%$. The damping ratio is obtained using the PO as

$$\xi = \frac{-\ln\left(\frac{PO}{100}\right)}{\sqrt{\pi^2 + \ln\left(\frac{PO}{100}\right)^2}} = 0.3409598$$

The settling time is taken as the time it takes in seconds to be bounded inside the error band as shown in the figure. The rise time is the time the process output takes to first reach the new steady-state value. As our respose starts from $t = 1.5s$, settling time is found as 3s and rise time as 0.45s.

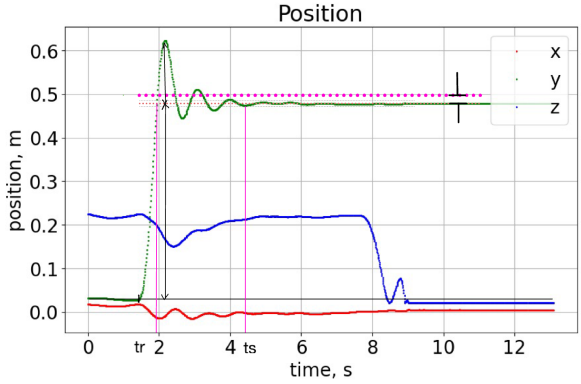

Fig. 1. Second order dynamics for step input in y direction

III. Trajectory Generator

A. A* trajectory graph search planner

Given the map as an input, we make use of A* algorithm to obtain an obstacle-free dense set of points from start to goal. A* is a variant of Dikstra's algorithm that utilizes heuristics with cost to open nodes in the graph. We use an L_2 norm from node to the goal node as the heuristic. We found this heuristic to be admissible and consistent in this situation where the quadrotor can move in 3 dimensions. We use a map resolution of $0.1m$ in all three axes and $0.2m$ for the margin.

B. Constant Velocity trajectory generation

In this section we detail the constant speed trajectory generation algorithm that we use in our experiments. We begin by simply downsampling by choosing every third point from the A* path planner. In doing this, we found that the points in the path were roughly 0.4 meters away from each other. We then set a constant velocity (v) of $1.0m/s$ which is a design choice. We set the velocity to 1m/s owing to safety considerations. We calculate waypoint time based on the following equation:

$$T_i = \begin{cases} 1.5 \times \frac{\|P_{i+1} - P_i\|}{v} & i = 0 \\ \frac{\|P_{i+1} - P_i\|}{v} & else \\ 1.5 \times \frac{\|P_{i+1} - P_i\|}{v} & i = n-1 \end{cases} \quad (1)$$

T_i is the time allocated to $i-th$ segment of the trajectory, P_i is the start waypoint of this segment and P_{i+1} is the end waypoint of this segment. n is the number of segments in the trajectory.

We know the $n-1$ unit vectors describing the direction of travel for each segment can be expressed as:

$$\hat{\mathbf{l}}_i = \frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|}$$

So, The desired velocity for a segment i is $\dot{\mathbf{r}}_T(t) = v\hat{\mathbf{l}}_i$, the desired position for time t is $\mathbf{r}_T(t) = \mathbf{p}_i + v\hat{\mathbf{l}}_i\,(t - t_{\text{start},i})$. For the first and last segment, we add an linear interpolation on the velocity to reduce the effect of rapid change in the velocity. The acceleration, jerk and snap are set to 0 for the constant velocity trajectory.

Since we assigned constant velocity to each segment, the planned velocity of the trajectory consists of piece-wise segments, corners and multiple spikes, as show in figure 2. So, it is naturally not as smooth as the min-snap trajecotry and the drone can not strictly execute the desired velocity as the dash-line demonstrated. However, we fine-tuned the controller so the actual trajectory was able to compensate for the non-smoothness, resulting in a more smooth and feasible trajectory.

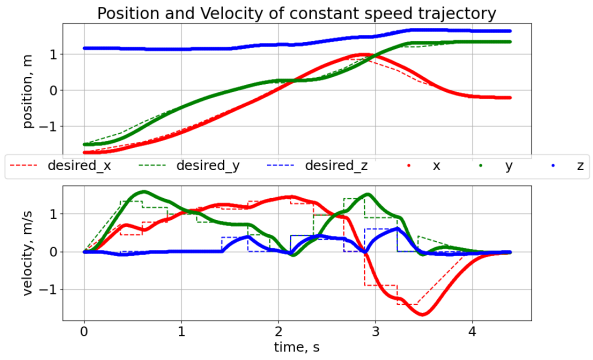

Fig. 2. Position and velocity of the constant speed trajectory

IV. Maze Flight Experiments

The maze flight experiments consisted of controlling the quadrotor to efficiently navigate from a start position to a goal position while avoiding collisions with static obstacles. We initially align the quadrotor's yaw with the world coordinate for accurate tracking with the Vicon system, the quadrotor was then brought to hover before tracking the trajectory given by the constant velocity trajectory generator. In this section, we report our results in testing our controller and trajectory generation technique for three different pairs of start and goal positons [1]. For safety considerations, the quadrotor was only planned on running with a maximum speed of 1m/s.

In Fig.3, Fig.4, and Fig.5, we could see the sparse waypoints, the static obstacles and the trajectory followed by the quadrotor in simulation. We notice that with a dense pruning of waypoints from the A* planner, the quadrotor was able to safely track the constant velocity trajectory, with slight deviations from the planned

[1]We denote the three trajectories as maze1,maze2 and maze3

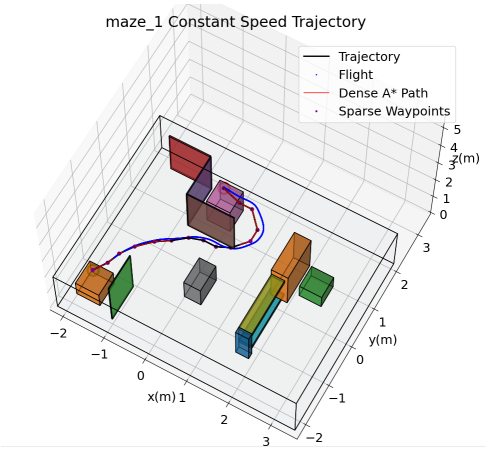

Fig. 3. Maze 1

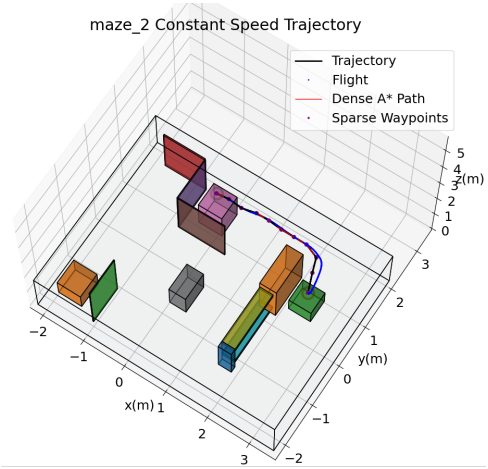

Fig. 4. Maze 2

path. We could see that the tracking error is small when the trajectory is straight, but the actual trajectory deviates from the planned trajectory in the presence of sharp turns. To alleviate this, we could reduce the planned speed at sharp corners or make use of higher order trajectory generators like one that minimizes snap.

In Fig.6 and Fig.7, we demonstrate the position-velocity curve and the trajectory followed by the quadrotor in actual experiment for Maze 1. A visual description of Maze 1's obstacle and planned trajectory is provided in Fig.3. The drop in z axis at the end in the position-velocity curve is due to the quadrotor landing after completing the trajectory. We found it interesting that different quadrotors exhibited slightly different behaviors for the same PD gains.

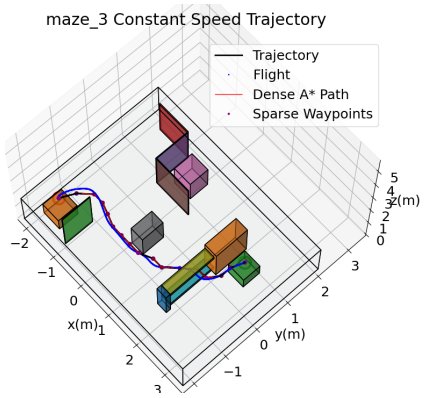

Fig. 5. Maze 3

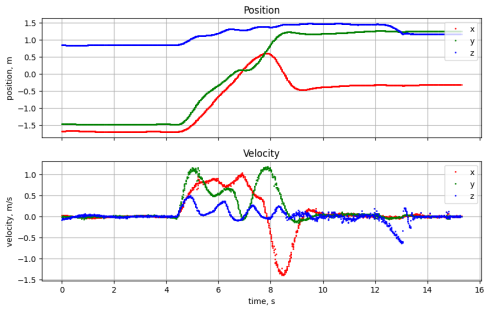

Fig. 6. Position and Velocity for the actual flight of maze 1(the last drop at z axis is because the drone is landing)

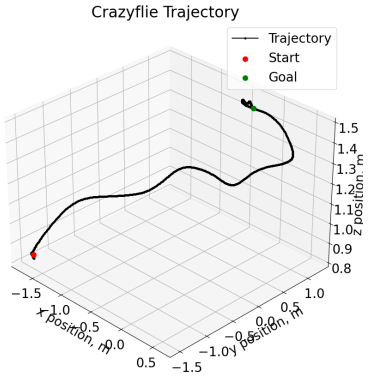

Fig. 7. Constant speed Trajectory for the actual flight of maze 1

In our experiments, we found that increasing speed or reducing the margin in the A* planning phase resulted in more aggressive trajectories. But it might not be safe for the constant speed trajectory to be more aggressive, because the tracking error is large at the sharp corners, so increasing speed or be more close to the obstacles will easily causes a hit to the obstacles or even worse, out of control. We also noticed that aggressive maneuvers drain the batters faster in the CrazyFlie. This is because of the commanded high motor speeds which demands more power from the battery.

We believe that it is possible to make more additons to improve upon the current provided approach in this report and thereby increase reliability of our system. Given more time, we would like to expand upon the following areas

- **Controller tuning:** Find better gains for the PD controller used, which could reduce overshoot and increase reliability in aggressive maneauvers
- **Graph search:** Tune the margin for our graph-search method, so shortcuts could be found in the maze that provides a better optimal trajectory (in terms of time taken). We do note that the planned trajectory might not be safe, but with the CrazyFlie's hgih basin of attraction tracking aggressive trajectories might be possible
- **Trajectory generation:** Add intelligent waypoint pruning to make the densed path from A* more sparse and feasible, this overall could decrease compute and planning wall-clock time. We would also love to explore higher order splines for trajectory generation such as minimum snap and improve upon it on the regions of time-segment optimization.