MEAM620 Project 1.4: Planning and Control, From Simulation to Reality

Team 01: Andrew Garrett, Yuntao Hu, Yunpeng Wang, Alexander Koldy

I. Introduction and System Overview

In Project 1.4, the objective is to deploy, test, and tune algorithms for planning and control onto a physical quadrotor platform. These algorithms are first developed in a simulation environment to tune their performance on a wide variety of environments. However, this lab seeks to expose the "sim-to-real" gap which is highly relevant to implementing safe and robust robotic systems in the real world.

Session Structure: The lab is divided into two working sessions, where the second session builds on progress made in the first session. Upon completing both sessions, the physical quadrotor has a tuned controller for attitude and position, a waypoint generator for optimal paths from a start position to a goal position, and a trajectory generator for dynamically feasible motion planning between waypoints. The first session focuses on tuning the controller for hovering, stepped control targets, and constant-velocity waypoint navigation. The second session focuses on tuning the waypoint and trajectory generators for more complex settings.

Lab Setting and Hardware: The lab setting is a controlled indoor environment to minimize extraneous factors that might affect accurate evaluation of algorithm performance. A VICON system of cameras is used to track the pose of the quadrotor with both high-fidelity and high-frequency. The quadrotor has retro-reflective markers arranged asymmetrically to ensure that the orientation of the tracked model is correct. The VICON system communicates with a dedicated host, which performs frame synchronization and extracts motion capture data from each camera, which is then fused for a high accuracy estimate of the true pose of the quadrotor with low latency and high accuracy. The quadrotor platform is a Crazyflie 2.1, which is a relatively cheap, small robot commonly used in education and research settings. Onboard the Crazyflie is an IMU (accelerometer and gyroscope), a radio transceiver for communication with a host, and a microcontroller with ESCs for relaying commands to the four brushless DC motors.

Communication with the Drone: A host computer performs all of the algorithmic computation and communicates with the quadrotor to send outputted control signals. As discussed in more detail in section II, the controller running on the host sends a vector of four motor-speed commands (radians per second) to the quadrotor via radio. The motor commands are each real-valued, positive floats which are clipped onto the domain $[v_{min}, v_{max}]$, which are parameters specified by the motor manufacturer. The quadrotor also has capability to relay IMU data back to the host, however this is not leveraged because the VICON system is more than sufficient for tracking the quadrotor pose.

II. Control

To control the robot's motion three-dimensional space, a nonlinear geometric controller is designed, with feedback coming in the form of position ($\mathbf{r}$), velocity ($\dot{\mathbf{r}}$), orientation $\left({}^{W}R_{B}\right)$ and angular velocity ($\boldsymbol{\omega}$). The desired force necessary to command the robot's movement in $\mathbb{R}^3$, $\mathbf{F}_{des}$, is given by the expression

$$\mathbf{F}_{des} = m\ddot{\mathbf{r}}_{des} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}. \tag{1}$$

The desired acceleration of the quadrotor is found via a proportional-derivative (PD) controller,

$$\ddot{\mathbf{r}}_{des} = \ddot{\mathbf{r}}_T + K_p(\mathbf{r}_T - \mathbf{r}) + K_d(\dot{\mathbf{r}}_T - \dot{\mathbf{r}}), \tag{2}$$

where the subscript T represents the variables associated with a generated trajectory, and the gains are given by

$$K_p = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 10 \end{bmatrix} \quad \text{and} \quad K_d = \begin{bmatrix} 4.5 & 0 & 0 \\ 0 & 4.5 & 0 \\ 0 & 0 & 5 \end{bmatrix},$$

with units of s^{-2} and s^{-1}, respectively. The acceleration from the trajectory acts as a feedforward term, where corrections are made via the proportional controllers on position and velocity. Using equations (1) and (2), the thrust input u_1 to the quadrotor is calculated via

$$u_1 = \mathbf{b}_3^T \mathbf{F}_{des}, \tag{3}$$

where $\mathbf{b}_3 = {}^{W}R_{B}\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ is the quadrotor's body z-axis expressed in the inertial frame. The input u_1 only provides the input necessary to keep the robot in the air, therefore a second input u_2 is used to change the robot's attitude. To calculate this input, a desired rotation based on a desired trajectory and input thrust is established. The input moments ($\mathbf{u}_2$) drive the robot's current orientation to this desired orientation given by the expression

$$\left({}^{W}R_{B}\right)_{des} = \begin{bmatrix} \mathbf{b}_{1_{des}} & \mathbf{b}_{2_{des}} & \mathbf{b}_{1_{des}} \end{bmatrix} \tag{4}$$

Since thrust always aligns with the body z-axis of the quadrotor, $\mathbf{b}_{3_{des}}$ is defined as the normalized desired force vector:

$$\mathbf{b}_{3_{des}} = \frac{\mathbf{F}_{des}}{||\mathbf{F}_{des}||} \tag{5}$$

The body y-axis should be perpendicular to both the body x-axis and a vector which defines the yaw direction in the plane made up by the world x and y axes. The following

yields an expression for the body y-axis, which ensures that the plane formed by the body z-axis and body x-axis contains the the vector defining yaw direction:

$$\mathbf{b}_{2_{des}} = \frac{\mathbf{b}_{3_{des}} \times \begin{bmatrix}\cos(\psi_T) & \sin(\psi_T) & 0\end{bmatrix}^T}{||\mathbf{b}_{3_{des}} \times \begin{bmatrix}\cos(\psi_T) & \sin(\psi_T) & 0\end{bmatrix}^T||}, \quad (6)$$

where ψ_T is the yaw given by a generated trajectory. The body x-axis is computed by taking the cross product between the body y-axis and body z-axis, i.e.,

$$\mathbf{b}_{1_{des}} = \mathbf{b}_{2_{des}} \times \mathbf{b}_{3_{des}}. \quad (7)$$

Equations (7), (6) and (5) are plugged into equation 4 to generate the desired attitude for the quadrotor. For the sake of simplicity, the superscripts and subscripts are dropped from both ${}^W R_B$ and $\left({}^W R_B\right)_{des}$. The orientation error is given by

$$\mathbf{e}_R = \frac{1}{2}\left(R_{des}^T R - R^T R_d es\right)^\vee, \quad (8)$$

where $\vee$ is used to vectorize a skew-symmetric matrix. Moreover, the error in angular velocity $\mathbf{e}_\omega = \boldsymbol{\omega} - \boldsymbol{\omega}_{des}$ is computed with the desired angular velocity being set to zero (though a trajectory-based desired angular velocity can be computed by exploiting differential flatness).

The second input set is now generated as follows:

$$\mathbf{u}_2 = -I(K_R \mathbf{e}_R + K_\omega \mathbf{e}_\omega), \quad (9)$$

where I is the inertia tensor of the robot and the gains are given by

$$K_R = \begin{bmatrix}875 & 0 & 0\\ 0 & 875 & 0\\ 0 & 0 & 875\end{bmatrix} \quad \text{and} \quad K_\omega = \begin{bmatrix}91 & 0 & 0\\ 0 & 91 & 0\\ 0 & 0 & 91\end{bmatrix},$$

with units of s^{-2} and s^{-1}, respectively.

The gain matrix K_p provides a proportional constant on the position error of the robot in $\mathbb{R}^3$, which produces large accelerations with high error and less acceleration with low error. The gain matrix K_d provides a proportional constant on the velocity error of the robot $\mathbb{R}^3$, which produces a damping effect on the system's acceleration. This helps to minimize the overshoot the quadrotor experiences when following trajectories or flying towards a point.

The attitude controller does not actually get used directly, as the Crazyflie features an on-board attitude controller to control q (quaternion representation of R) to q_{des} (quaternion representation of R_{des}).

For real experiments, the maximum velocity of the robot is limited to $1\frac{m}{s}$; moreover, gains (shown previously) are reduced to approximately 70% of their original value. After this, gains are re-tuned off the response of the physical system. The simulation likely does not account for highly-nonlinear aerodynamics or measurement error, so lowering speed and gains allows the quadrotor to follow a more stable flight.

According to Figure 1 the controlled system has an approximate rise time of $2.4s$, approximately a 28% overshoot (with respect to the steady-state), a settling time of about $4s$, an approximate steady-state error of $0.1m$, and a damping ratio of about 0.4. The controller performs well with a quick rise time and settling, time but can certainly be better tuned by increasing the damping gain K_d. The performance of the controller can be further improved with an integral term on the quadrotor's position to eliminate the steady-state error.

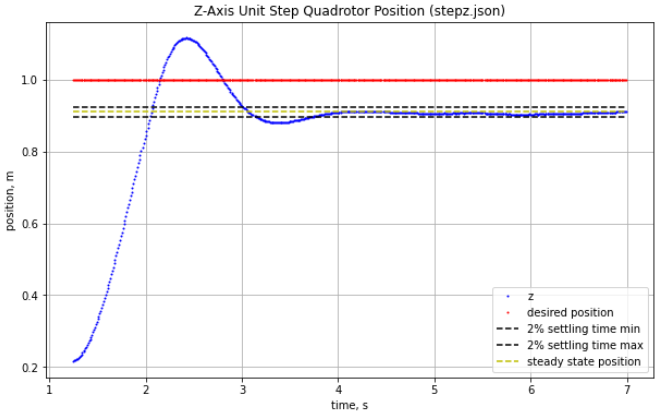

Fig. 1: Step response in z

For the second portion of the lab, separate gains were used for more complex trajectory following. These gains are as follows:

$$K_p = \begin{bmatrix}13 & 0 & 0\\ 0 & 13 & 0\\ 0 & 0 & 13.5\end{bmatrix}, \quad K_d = \begin{bmatrix}5 & 0 & 0\\ 0 & 5 & 0\\ 0 & 0 & 5.2\end{bmatrix},$$

$$K_R = \begin{bmatrix}350 & 0 & 0\\ 0 & 350 & 0\\ 0 & 0 & 100\end{bmatrix} \quad \text{and} \quad K_\omega = \begin{bmatrix}27 & 0 & 0\\ 0 & 27 & 0\\ 0 & 0 & 21\end{bmatrix}.$$

III. Motion Planning

The quadrotor's motion planning uses the A* Search algorithm for generating an optimal path from a start position to a goal position, linear downsampling for path pruning, and a constant-speed trajectory generator for tracking between waypoints on the pruned path.

World Model: For a known 3D map, a 3D occupancy grid $M \in \{0,1\}^{n \times m \times o}$ can be parameterized by voxel resolution $r \in \mathbb{R}$ and margin $p \in \mathbb{R}$ to inflate obstacles and boundaries. M can be represented with a graph $G(V, E)$, where V is the set of unoccupied voxel positions and E is the set of allowed transitions between $u, v \in V$. $\forall u, v \in V$, $\exists e(u, v) \in E$ if $u \neq v$ and $||u - v|| \leq \sqrt{3}r$, meaning that u may have at most 26 edges to neighboring v.

Path Planning: A* Search is a hallmark graph algorithm in robotics which builds upon Dijkstra's Algorithm by expanding nodes prioritized by the sum of their "cost-to-come" ($g(v) : V \rightarrow \mathbb{R}$) and an estimated "cost-to-go" ($h(v) : V \rightarrow \mathbb{R}$), $f(v) = g(v) + h(v)$. A* Search takes as input a graph $G(V, E)$, a source node $v_s \in V$, and a sink node $v_g \in V$. The "cost-to-come" term $g(v)$ defines the cost of following the optimal sub-path from v_s to $v \in V$. The "cost-to-go"

term, also known as a heuristic $h(v)$, is an estimate of the cost to follow the optimal path from $v \in V$ to v_g. A* Search is optimal and complete when $h(v)$ is both admissible and consistent. $h(v)$ is admissible if $\forall v \in V$, $h(v) \leq c^*(v, v_g)$, where $c^*(v)$ is the true cost of the optimal path from v to v_g. $h(v)$ is consistent if $\forall u, v \in V$ such that $e(u, v) \in E$, $h(u) \leq c(u, v) + h(v)$. For this lab, h is chosen as the Euclidean (L_2) distance, however other candidates include the Manhattan (L_1) distance and Diagonal or Chebyshev (L_∞) distance. As such, A* Search will return a path $p^* = (v_s, ..., v_g)$ which minimizes the euclidean distance traversing $e \in E$ from v_s to v_g.

Path Pruning: Path pruning serves to reduce the number of nodes in the path p^* planned by A* Search, which can offer immediate benefits in reduced time and space complexity for trajectory generation and tracking. In trajectory generation, dense waypoints can not only lead to inefficient runtime and storage, but may also result in failure to converge to a smooth trajectory for methods such as minimum-jerk and minimum-snap. Furthermore, high-frequency and high-magnitude changes in desired state can lead to jerky and/or inaccurate tracking.

We tried the Ramer-Douglas-Peucker algorithm, but because of its characteristic of retaining so few path points that it loses its own curve properties, we ended up using the mean point pruning method. Linear downsampling p^* by an integer factor $k > 0$, means that pruned path w^* has $|p^*|/k$ waypoints. First, create $\lfloor |p^*|/k \rfloor$-partitions of p^*, each of size k. Elements of w^* are simply the mean of each of these partitions, and the goal position is appended to the end. As such, $|w^*| = \lfloor |p^*|/k \rfloor + 1$. This greatly reduces the number of waypoints used for trajectory generation, while preserving the general shape of the path. For the experiments in section IV, $k = 5$.

Trajectory Generation: Trajectory planning aims to convert our points into the desired control states for the system. Although we tested our quadrotor using the mini-snap and mini-jerk methods, the results were not very satisfactory. In this lab, we applied the "constant speed trajectory" method with some adjustments and improvements. State outputs will be:

$$\dot{r}_{T,i} = v\bar{l}_i$$

$$r_{T,i} = P_i$$

$$\ddot{r}_{T,i} = 0;\ \dddot{r}_{T,i} = 0;\ \ddddot{r}_{T,i} = 0$$

$$\psi_{T,i} = 0; \dot{\psi}_{T,i} = 0$$

where i means the corresponding time segment for current time, P_i means the current control point, v is constant speed we set, which $v = 1m/s$, $v\bar{l}_i = \frac{P_{i+1} - P_i}{||P_{i+1} - P_i||}$. For position output, we controlled the quadrotor to follow the starting point of the current time segment during that segment instead of propagating a dynamic movement from that point. This allows the quadrotor to maintain a certain level of stability (its controller may be linearly increasing) without the need for frequent fine-tuning of the quadrotor input at all positions. Other outputs, such as acceleration, were set to 0.

IV. Maze Flight Experiments

The position and velocity vs time of three different mazes are shown in Fig. 2, Fig. 4, and Fig. 6. And the inertial obstacles, waypoints, planned trajectory, and actual flight path of three different mazes are shown in Fig. 3, Fig. 5, and Fig. 7.

The tracking errors on positions in three mazes are mostly less than 0.1 meters and less than 0.2 meters in sharp turns case. The errors are acceptable because the quadrotor still has a margin of 0.25 meters. At the same time, the velocity errors flutter obviously due to the jittery actual velocity. It may be partially due to the constant velocity trajectory generator we use. In both the position and velocity cases, the error increases obviously when suddenly changing the forward direction, i.e. the sharp turns used to bypass the obstacles.

Our trajectory could be more aggressive. If we make the resolution of our map extremely small, we will be able to find nearly optimal path and trajectory, but this will lead to explosive computing complexity which may not be acceptable with the limited resource on the quadrotor. In addition, if we do not use the Ramer-Douglas-Peucker algorithm, we can obtain finer grain trajectory, but this also leads to a higher computing complexity because of the extremely large matrix we are going to solve in the trajectory planner.

If we use the PID controller, the performance of the trajectory following will be way better. In this case, the quadrotor can fly at a higher speed and is more robust to sharp turns. At the same time, the threshold of the Ramer-Douglas-Peucker algorithm also has a huge impact on the number of waypoints the quadrotor follows. And the more waypoints the quadrotor follows, the fewer sharp turns the quadrotor are going to face. In addition, a quadratic or cubic spline generator that is used to fill the gaps between the waypoints also can mitigate the overshooting problem when facing sharp turns. However, it's also a tradeoff between performance and computing complexity.

If we have more sessions, we will test everyone's code. When we were doing the in-person labs, the device problem takes up too much time. And the numpy version on Andrew's and Yunpeng's machines is different from the one on the controlling computer in the labs. So although we have a min-jerk and min-snap trajectory generator with theoretically better performance, we don't have enough time to verify our code on the quadrotor.

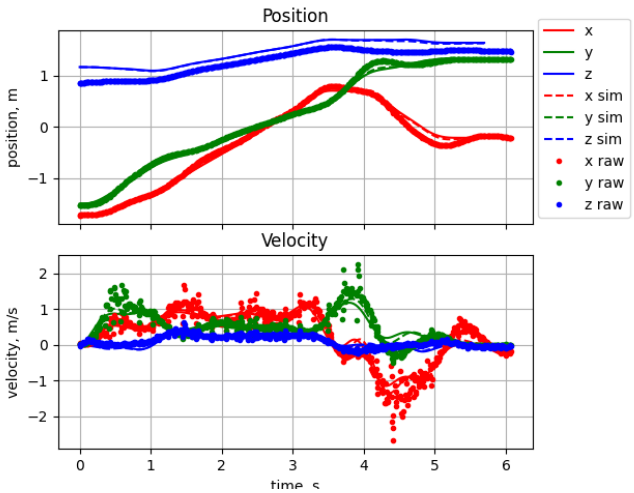

Fig. 2: Position and velocity vs time of maze 1

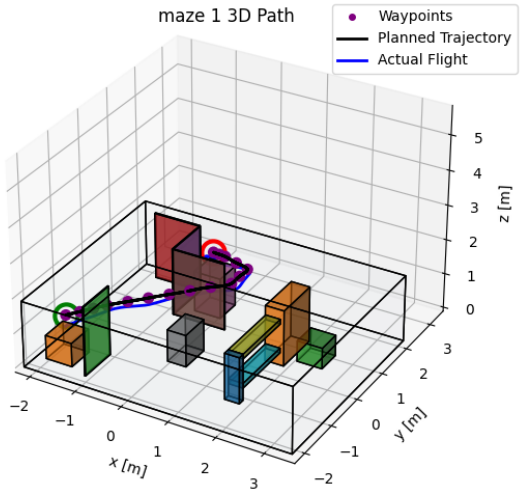

Fig. 3: Obstacles, waypoints, and planned trajectory of Maze 1

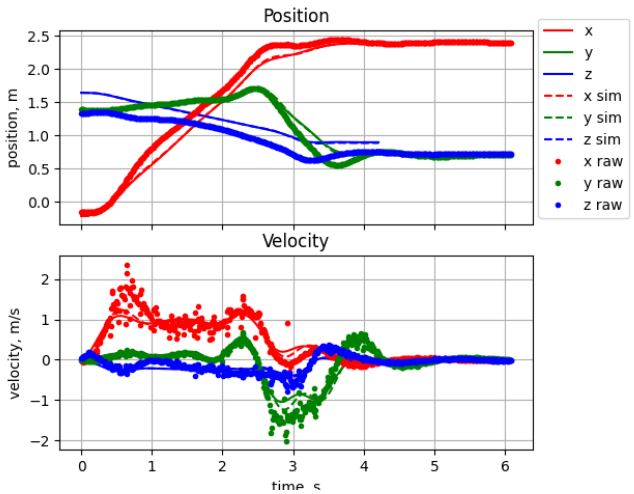

Fig. 4: Position and velocity vs time of maze 2

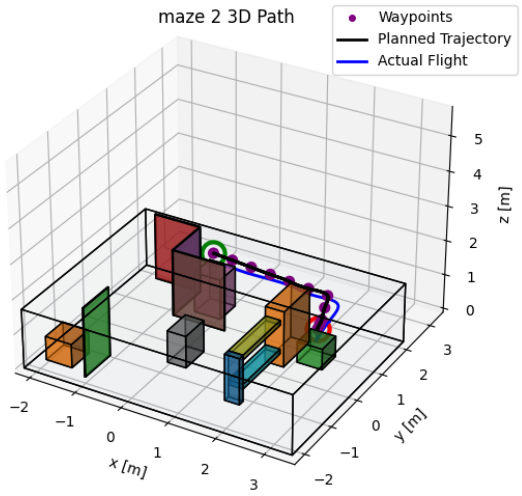

Fig. 5: Obstacles, waypoints, and planned trajectory of Maze 2

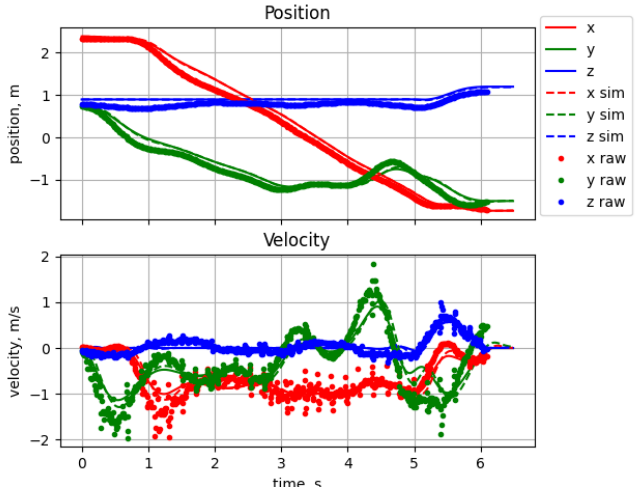

Fig. 6: Position and velocity vs time of maze 3

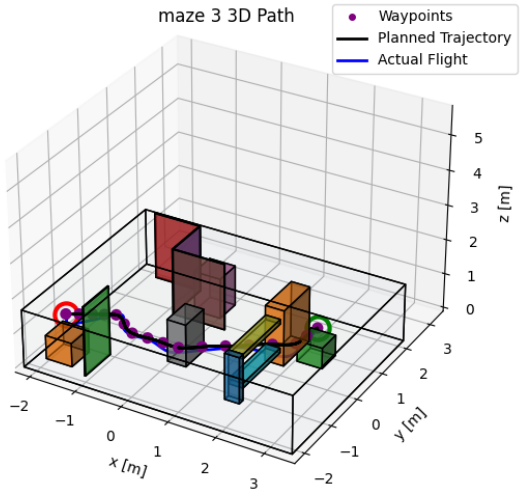

Fig. 7: Obstacles, waypoints, and planned trajectory of Maze 3