Quadrotor Planning and Control

Nikolaos Bousias, Spencer Folk, and Shaun Schaeffer
Group 5

March 31st, 2020

1 Introduction & System Overview

The main objective of this lab was to implement and verify the performance of a controller and trajectory generator on a quadrotor platform. The results show that the implemented algorithms can successfully navigate a Crazyflie 2.0 platform through a known map. The Crazyflie is an open source quadrotor platform with a characteristic length of 92-mm and a mass, including battery, of 30g. The Crazyflie has an on-board microcontroller and Inertial Measurement Unit (IMU) which enables low-level control and partial state estimation. The low-level control is handled using a geometric nonlinear controller that runs offline, but sends roll, pitch, thrust, and yaw rate commands to the on-board computer's attitude controller. The high-level trajectory commands, i.e. time-dependent position and heading, are also computed offline with a path planner, and used by the geometric controller as a feedforward term in the control algorithm. Communication between the offline host computer and the Crazyflie is accomplished via the CrazyRadio (2.4GHz) and Robot Operating System (ROS). A VICON system reports precise estimations of the 3D position of the quadrotor for both analysis and feedback control.

2 Nonlinear Geometric Controller

The controller plays a vital role in stabilizing the quadrotor under disturbances and tracking a trajectory that safely navigates the quadrotor between waypoints. At its core, the controller implements a Proportional-Derivative (PD) control of the quadrotor's position $\mathbf{r} = [x,y,z]^T$ to a desired trajectory defined by a position and heading, $\mathbf{z}_T = [\mathbf{r}_T, \psi_T]^T$. A nonlinear geometric controller similar to [1] was utilized.

The desired acceleration is defined via the error dynamics:

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

where K_p and K_d are diagonal, positive definite gain matrices. The desired force vector is then:

$$\mathbf{F}^{des} = m\ddot{\mathbf{r}}^{des} + \begin{bmatrix} 0 & 0 & mg \end{bmatrix}^T \tag{2}$$

where m is the quadrotor's mass and g is the gravitational acceleration. However, the quadrotor is only capable of generating forces along its body frame z-axis ($b_3 = R[0,0,1]^T$, where R is the rotation matrix from the world to body frame). Therefore, we define an input u_1 as the projection of the desired force onto the $\mathbf{b}_3$ axis:

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

The next step is to align the quadrotor with a desired attitude where each axis is defined by the desired force vector and heading, ψ_T. Each axis can be defined as:

$$\mathbf{b}_3^{des} = \frac{\mathbf{F}^{des}}{||\mathbf{F}^{des}||}, \quad \mathbf{b}_2^{des} = \frac{\mathbf{b}_3^{des} \times \mathbf{a}_\psi}{||\mathbf{b}_3^{des} \times \mathbf{a}_\psi||} \tag{4}$$

where the vector $\mathbf{a}_\psi = [\cos\psi_T, \sin\psi_T, 0]^T$ is aligned orthogonal with the desired heading for a given trajectory in the inertial frame. Finally:

$$\mathbf{b}_1^{des} = \mathbf{b}_2^{des} \times \mathbf{b}_3^{des} \tag{5}$$

The desired attitude can be constructed from Equations 4-5 as a rotation matrix $R^{des} = [\mathbf{b}_1^{des}, \mathbf{b}_2^{des}, \mathbf{b}_3^{des}]$.

The resulting input u_1 and rotation matrix R^{des} (converted to a quaternion) are sent to the on-board microcontroller for feedback control. The reason for this is that the attitude must run at a much faster rate (roughy 5x) than that of the position controller. Even the offline geometric controller is not capable of running at this high of a frequency. More details about the on-board feedback controller can be found in [2].

2.1 From Simulation to Experimentation

A number of adjustments had to be made when transitioning from simulation to experiment. These are a result of imprecise physical measurements (e.g. mass), unmodeled aerodynamic effects (e.g. ground effect, blade flapping), or differences between commanded thrusts and observed forces. To account for all the above, re-tuning of the gains was necessary. The final gains were:

$$K_p = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 11 \end{bmatrix} \quad [m^{-1}] \tag{6}$$

$$K_d = 2 \cdot \sqrt{K_p} \quad [s - m^{-1}] \tag{7}$$

$$K_R = 500 \cdot \mathbb{1} \quad [rad^{-1}] \tag{8}$$

$$K_\omega = 70 \cdot \mathbb{1} \quad [s - rad^{-1}] \tag{9}$$

where $\mathbb{1}$ represents a [3x3] identity matrix.

K_d and K_p are the derivative and proportional gains, respectively. K_d controls the gain given to the velocity error and acts as a damper, while K_p controls the gain given to the position error and acts as a spring. Values for K_d were determined based on K_p such that the second-order error dynamics are critically damped ($\zeta = 1$) and a desired natural frequency ω_n is obtained. The attitude gains, K_R and K_ω, were originally formulated in a similar manner, but were altered during the manual tuning process simply to achieve stable behavior. The performance of this controller can be assessed by observing a step response.

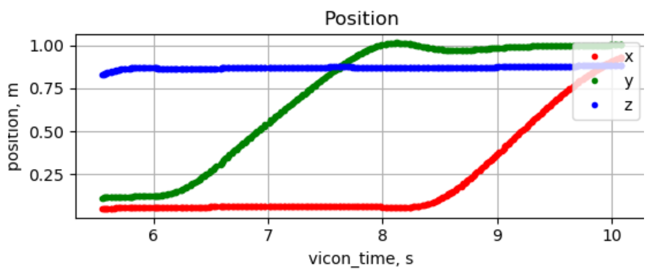

Figure 1: CrazyFlie step response taken from a cube trajectory from a prior lab session.

A step response in the y direction was identified in Figure 1. From this figure, we estimate the settling time to be approximately $2.5s$, the steady state error less than 1.5%, the rise time close to $1.6s$ and the overshoot $\approx 1\%$. The response is identical in the x direction. However, the z direction shows a much larger steady state error as shown in Figure 3 below.

3 Path Planner

The core of the path planner is an A* algorithm that runs offline upon map initialization; it returns an optimal path for a given map, resolution (assumed 0.125-m in each direction), quadrotor safety margin (assumed 0.3-m), and starting and goal 3D positions. The path generated by A* is guaranteed to be the shortest path with no obstacle collisions. The resolution and safety margin affect the running time (off-line) of A*.

3.1 Waypoint Pruning

Unfortunately, the initial path generated from A* is quite dense, and isn't well-suited for trajectory generation. A pruning algorithm reduces the dense path to the smallest number of waypoints that still avoids collisions. From a given waypoint, the pruning algorithm iterates along the dense A* path, checking for collision if an intermediate waypoint is removed. Once a possible collision is detected, prior waypoints are removed and the algorithm restarts with the next available waypoint. The result of the pruning algorithm is a sparse path resembling that generated by the A* algorithm. The difference is apparent in the 3D map views below. The sparse path is more computationally efficient for trajectory generation algorithms and is better conditioned for planning smooth and fast trajectories. However, these trajectories need to be linear as a higher order spline could cause collisions with obstacles that were avoided in the dense A* path.

3.2 Trajectory Generation

For this project, a "bang-bang" method of trajectory generation was used on the set of sparse waypoints. The name "bang-bang" comes from the simple premise of switching between positive and negative **constant** accelerations between each segment of the trajectory. The trajectory generation algorithm works as follows:

The algorithm begins by defining a nominal acceleration, a_s. In this lab, the acceleration was set to a conservative $1.5\text{-}m\text{-}s^{-1}$. This acceleration is used to determine a nominal time duration for each segment i.

$$t_s^i = t_s^{i-1} + \sqrt{\frac{4\Delta^i}{a_s}} \tag{10}$$

where Δ is the total length of the segment between two waypoints that constitute segment i. Each segment also has an associated unit vector, $\hat{n}^i$. The segment duration and unit vector are tabulated for each segment, and used in implementation to identify the correct sign and direction of the acceleration.

On runtime, we determine what segment i the vehicle is currently in based on the current time, t; in addition, we identify which half of the segment the vehicle is in based on t_s^i. The nominal acceleration is then integrated forward in time to get velocity, $v_s^i(t)$, and position, $p_s^i(t)$ along the path:

$$v_s^i(t) = \begin{cases} a_s(t - t_s^i) & \text{if } t <= \frac{t - t_s^i}{2} \\ a_s\frac{(t-t_s^i}{2} - a_s(t - t_s^i) & \text{if } t > \frac{t-t_s^i}{2} \end{cases}$$

$$p_s^i(t) = \begin{cases} \frac{1}{2}a_s(t - t_s^i)^2 & \text{if } t <= \frac{t-t_s^i}{2} \\ \frac{1}{2}a_s(\frac{t-t_s^i}{2})^2 - \frac{1}{2}a_s(t - t_s^i)^2 & \text{if } t > \frac{t-t_s^i}{2} \end{cases}$$

Finally, the unit vector associated with each segment is used to project position $p_s^i(t)$, velocity $v_s^i(t)$, and acceleration onto the x,y, and z counterparts, which are then sent to the controller.

The resulting trajectory is a collection of linear segments between waypoints with discontinuous changes in acceleration. As a result there are sharp changes in direction in velocity, which is evident in Figure 8. However, the position itself remains smooth and continuous. Since the quadrotor is an underactuated system (6DOF, 4 rotors), unable to produce arbitrary accelerations, these trajectories cannot be perfectly tracked. However, given a conservative acceleration, the tracking error can remain reasonably low. These linear segments were found to be sufficiently fast in simulation; and more importantly, they had the highest success avoiding obstacles because the segments are identical to those generated after waypoint pruning.

4 Experiments

The quadrotor was tasked with three sequential navigation missions set in a maze in the lab. The location, size, and structure of each obstacle within the space is known to the path planner prior to flight. The following graphs depict the CrazyFlie trajectory in lab (black line), the simulation (green line), the path designed with the A* (red) and the pruned one (purple dots). Figures 7,8 show the position $[m]$ and linear velocity $[m\text{-}s^{-1}]$ of the CrazyFlie with respect to the simulated ones for maze path 3.

4.1 Maze Path 1

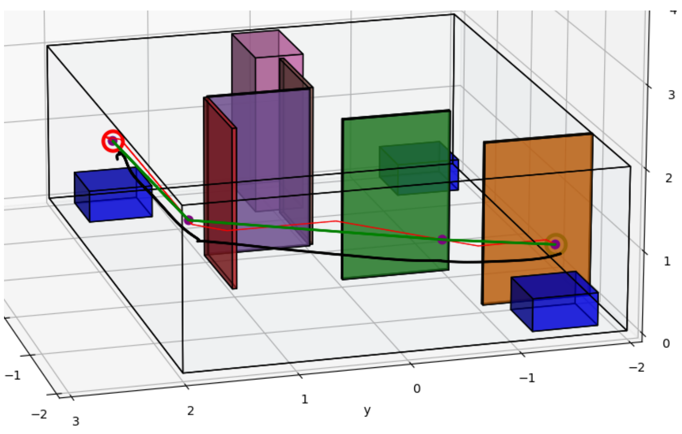

Figure 2: Crazyflie vs. simulation for Maze path 1.

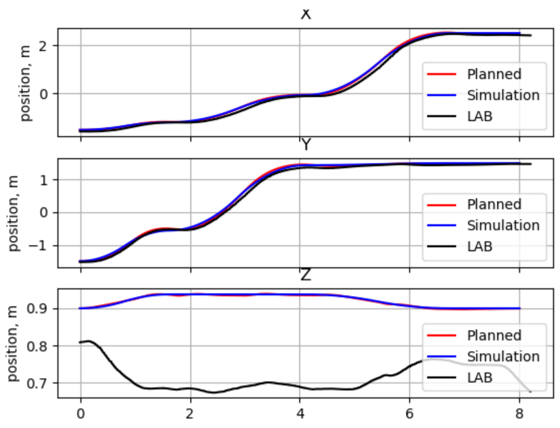

Figure 3: XYZ position for Maze path 1.

4.2 Maze Path 2

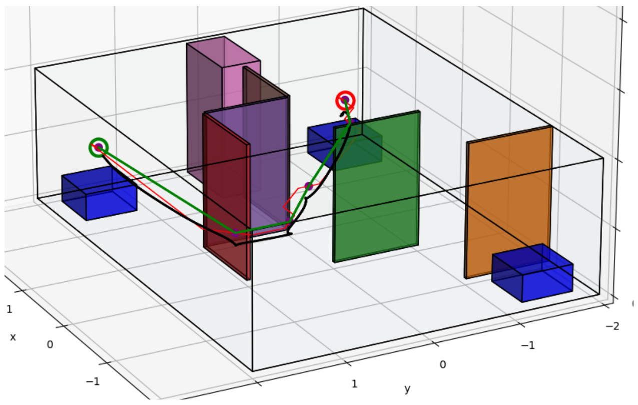

Figure 4: Crazyflie vs. simulation for Maze path 2.

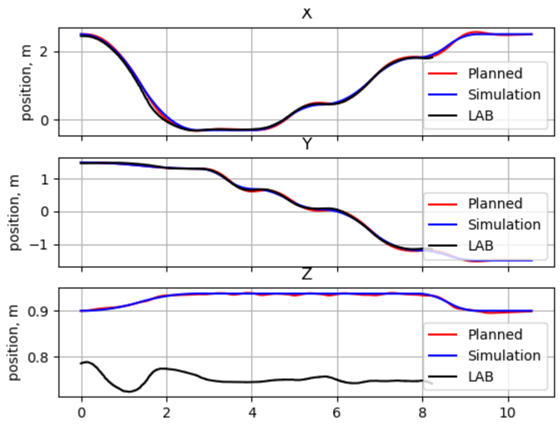

Figure 5: XYZ position for Maze path 2.

4.3 Maze Path 3

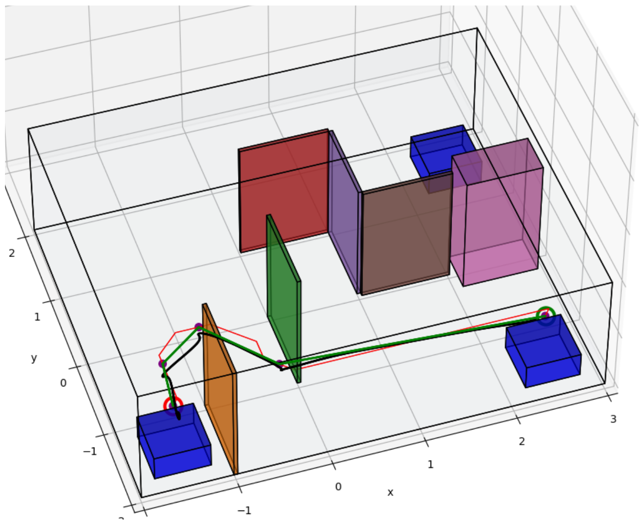

Figure 6: Crazyflie vs. simulation for Maze path 3.

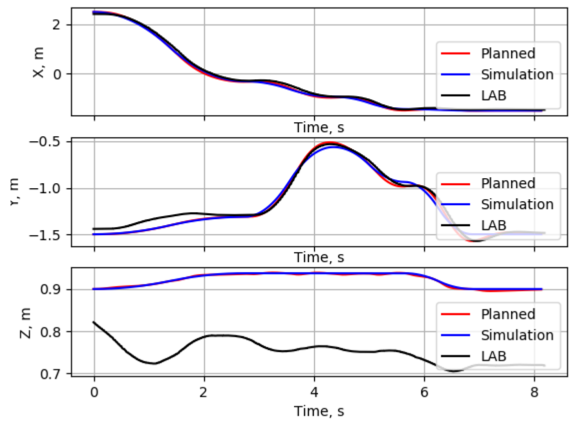

Figure 7: XYZ position for Maze path 3.

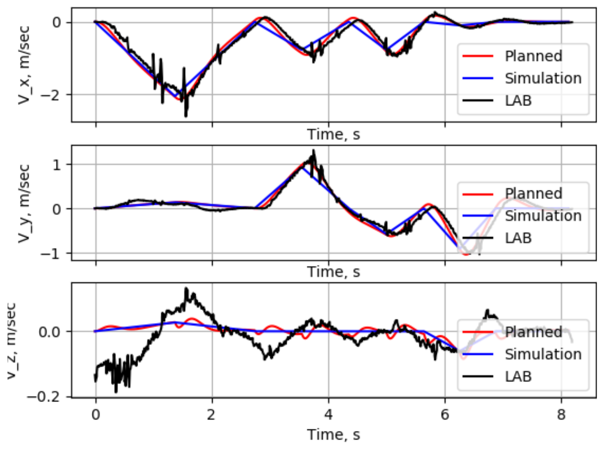

Figure 8: Linear velocity for Maze path 3.

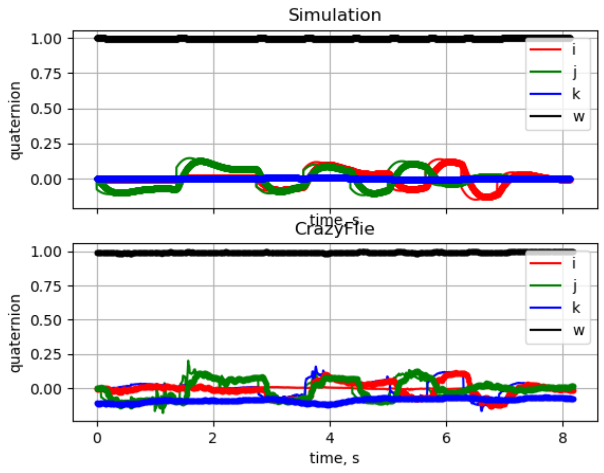

Figure 9: CrazyFlie orientation for maze path 3.

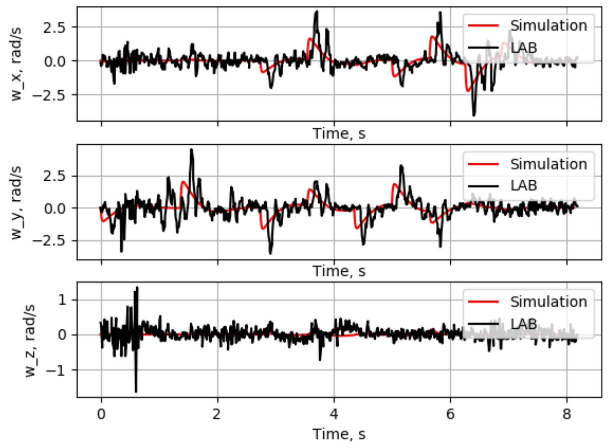

Figure 10: Angular velocities for maze path 3.

5 Discussion

As seen in the graphs above, the differences between the simulated and actual positions and velocities for all three maps are minimal. Figures **??** and 8 plot the positions and velocities, respectively, of the Crazyflie for Map 3. There is good agreement in the x and y directions. This is not the case for altitude (z), where there is a consistent steady state error of 0.10-0.15m. The same is true for the other two experiments. The mean absolute errors are summarized in the table below.

Axis	Position Errors	Velocity Errors
x	7.99%	47.75%
y	3.56%	51.40%
z	18.84%	248.63%

Table 1: A summary of the mean absolute errors (in % of the respective mean values) for position and velocity in the x, y, and z directions

Overall, the position tracking errors are within 20%, skewed by a large error in altitude, z. This suggests that a more aggressive or faster trajectory could have been used. At the speed used for the experiment the tracking in the x/y-directions was never greater than 0.10m. As such, the margin used for path planning could have been decreased from 0.3m to 0.15-0.2m. This would have allowed the Crazyflie to fly closer to the obstacles and decrease the overall travel time and distance. The limiting factor in the performance of these algorithms was the apparent steady state error in altitude, which led to much larger uncertainty in the quadrotor's ability to track a trajectory.

6 Conclusion

Group 5 successfully flew a Crazyflie through a maze full of known obstacles. A nonlinear geometric controller was implemented, along with an A* algorithm that indicated the optimal path which was later improved by a pruning algorithm. VICON data was collected both for feedback control and to assess the performance of the system. The x,y position of the quadrotor maintained an error below 10% that of the simulator. However, the z axis suffered from a $\approx 0.1m$ steady-state error, most likely due to an incorrect approximation of the quadrotor's mass.

6.1 Extensions

Given another session in the lab, we would attempt a more aggressive trajectory generation method, such as minimum jerk or snap, that would enable continuous movement between waypoints. The "bang-bang" method resulted in stopping at each waypoint, increasing the overall trajectory time and avoiding twitchy or oscillatory behavior from the quadrotor.

References

[1] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, May 2011.

[2] https://github.com/bitcraze/crazyflie-firmware.