Project Report: Quadrotor Planning and Control

Team 1: Seoungjun (Elijah) Lee, Miaoyan Sun, Maria-Elisabeth Tzes, Yufu Wang

I. Introduction and System Overview

This lab is set up for us to test our control and trajectory codes on a real quadrotor. During the process, we explored the difference between computer simulation and real robot manupulation. The first part of the experiment emphasized the tuning of control gains on the quadrotor. We made some changes on the gains and finally came up with a serial of numbers that allows safe taking off, flying, landing and gives smooth trajectories. Whereas in the second part, we are given three maps to explore feasible smooth paths to fly, during which we put more effort on controlling the velocity and flying altitude.

Speaking of hardware, the quadrotor we are using is called 'Crazyflie 2.0'. It is a star product of the company Bitcraze. Crazyflie is a small quadrotor having a size of 92x92x29mm and a weight of only 27g. The microcontroller is STM32F405, which supports transmission from iOS and Android with Bluetooth LE, and from Windows/Mac OSX/Linux with the Crazyradio or Crazyradio PA. A VICON positioning system surrounds the lab with many Anchors, which by sending short high frequency messages tracks the 3D position of Crazyflie from time to time. Other than that, the IMU senses the orientation and angular velocity of Crazyflie. After receiving these information, the computer in the lab does all necessary computations for generating trajectories. Then, with the trajectory, control input data including thrust and moment are calculated and sent to the Crazyflie to carry out the test.

II. Controller

We implemented and utilized the geometric nonlinear controller for both our lab sessions. The position controller can be described as

$$u_1 = b_3^T F^{des} \tag{1}$$

where $F^{des} = m\ddot{r}^{des} + (0, 0, mg)^T$ and $b_3 = R(0, 0, 1)^T$. That is, the direction of the controlled thrust is aligned with the z axis of the current configuration of the quadrotor, while the magnitude of the thrust is chosen such that the difference between $u_1 b_3$ and F^{des} is minimized. This design makes no assumption about the responsiveness of the altitude controller, and allows the quadrotor to maneuver more aggressively.

The position control gains we adapted are $K_p = (4, 4, 8)s^{-2}$ and $K_d = (5, 5, 5)s^{-1}$. In error dynamics, K_p governs the importance of position error and has a spring effect on the error. K_d governs the importance of the first derivative error and has a damping effect. For our system, we want K_p to be large enough for the quadrotor to be responsive, and then tune K_d to minimize overshot or overdamp.

In the lab, we only utilized the position controller above, and let the onboard altitude controller filled in the blank by simply commending the desired altitude quaternion. One reason is because the altitude controller and its gains depend on the thrust and drag coefficients, as well as the design of the quadrotor (e.g. arm length), which could be very different than that in the simulator.

We have to adjust the gains of the position controller slightly in the lab. This is because the mappings between the commend thrust and the actual force are different than those in the simulator. Additionally, as we have mentioned, we relied on the onboard altitude controller, which could be slower than simulation.

Next, we will analyze the performance of our controller with data from a z axis step response experiment, as shown in Fig 1. First of all, notice at around $t = 2s$ when we switched in our controller, the position of the drone started drifting, till at around $t = 4s$ when our controller finished initialization and took control. This is an unexpected behavior that is not part of our controller, as the onboard controller was supposed to maintain the position until our controller finish initialization (the TA could not explain that either). For the sake of our analysis, we will use $t = 4s$ as the starting position, and ignore initial velocity due to drifting.

From the figure, we can measure common controller characteristics, as presented in Table I. We see that the z response has all most no overshot and little steady state error, with $\zeta = 0.88$ signaling that the system is slightly underdamped. Rising time T_r and 2% settling time T_s, directly measured from the figure, shows that the system is responsive for our purpose.

There are a few ways to calculate ζ. One way is assuming it is a perfect second order system, with $\zeta = \frac{K_d}{2*\sqrt{K_p}} = 0.88$. It is also possible to estimate ζ from T_s and T_r without having to rely on known control gains. This gives an estimation of a similar $\zeta = 0.87$, showing that the second order control law is an accurate model for our system.

One last comment is that the position gains K_p has

lower values for x and y. From more step response analysis we conducted, which is not included here due to space limitation, we find larger steady state errors and slower responses for x and y. So it is possible that the gains for x and y should have been higher and closer to 8.

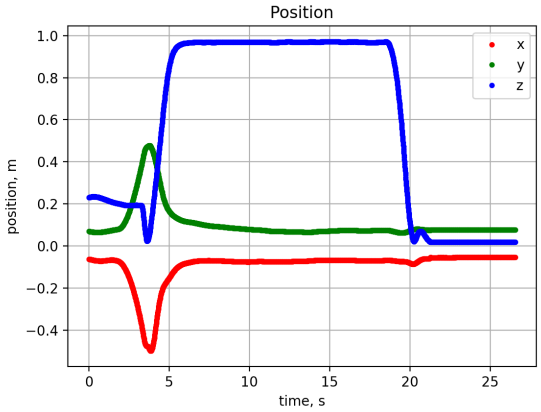

Fig. 1. Z step response

e_{ss}	T_r	$T_s(2\%)$	OO	ζ
0.04 m	1.2 s	2 s	0	0.88

TABLE I

III. Trajectory Generator

Our trajectory generator first discretizes the space into a voxel map and runs A^* search algorithm to obtain a set of waypoints as a path. From this set, we selected waypoints by removing the redundant points that lie in the middle of some segments along a path (i.e. a midpoint from a line path is considered redundant). In this case, we avoid all the delaying maneuver of a drone along the straight line path, and the drone will travel straightly going from the initial to the end points along a line path.

Algorithm 1 shows the overall state variables that are precomputed and stored at each waypoint. The algorithm takes the input of waypoints and number of the waypoints and returns set of distances, set of velocities, and set of times that include each state variable at each waypoint.

Notice that for line path with its norm greater than 4, we set higher velocity than those less than 4. This quantity can be arbitrarily altered to guarantee the efficient but safe uav flight manuever.

The time is allocated between waypoints according to this algorithm (i.e. it is basically proportional to the distance between waypoints but it will take a half time for the distance larger than 4).

For this lab, we implemented trajectory generator so that each flat output obeys the basic physics relation (i.e. velocity is the change in position divided by infinitesimal time and acceleration is the change in velocity divided by infinitesimal time). For distance, we use precomputed state values at each waypoint and calculate any value in between two waypoints by linearization.

For obtaining position $\mathbf{x}(t)$ between $\mathbf{x}_1(t_1)$ and $\mathbf{x}_2(t_2)$:

$$m = \frac{t - t_1}{t_2 - t_1} \tag{2}$$

$$\mathbf{x}(t) = (1 - m) \cdot \mathbf{x}_1(t_1) + m \cdot \mathbf{x}_2(t_2) \tag{3}$$

For obtaining velocity $\mathbf{v}(t)$:

$$\mathbf{v}(t) = \frac{\mathbf{x}(t + dt) - \mathbf{x}(t)}{dt} \tag{4}$$

Similarly, for obtaining acceleration $\mathbf{a}(t)$:

$$\mathbf{a}(t) = \frac{\mathbf{v}(t + dt) - \mathbf{v}(t)}{dt} \tag{5}$$

We can guarantee that velocity and acceleration converge to true value with small dt. Since we are using linear velocities in moving the drone (although it may dynamically change between two values according to Algorithm 1), we set other state variables as zero.

Due to the nature of our trajectory generator, it may have abruptly changing velocities along the path. However, since we take an approach of taking a derivative of velocity with small variation of time, reasonable acceleration will be calculated and contributes to the smooth transition of the flight maneuver.

Looking at the plot over time of position and velocity from Section. IV, we confirm that the designed trajectory generator is promising. Although there may exist dynamically changing value in velocities, as can be seen

Algorithm 1 $[D, V, T]$ = PreComputeDVT(P, n)

Input: P (set of points), n (number of points)
Output: D (set of distances), V (set of velocities), T (set of times)

initialize sets of D, V, T with a zero vector
for $i = 1$ to $n - 1$ **do**
  compute distance from $P[i]$ to $P[i + 1]$ and appends to D
  if $D[i] > 4$ **then**
    $V[i] = 1$
  else
    $V[i] = 0.5$
  end if
  if $i > 1$ **then**
    $T[i] = \frac{D[i]}{V[i]} + T[i - 1]$
  end if
end for

from Fig. 4, the planned trajectory seems reasonable and valid.

The only loop that may be missing is that the drone may not follow the path exactly because it will drift a bit at each waypoint. To avoid this problem, we should have a trajectory generator for higher dimensionality or have an algorithm that the drone slows down near each waypoint so that it stays closely to the desired path. However, we observed that having this algorithm was sufficient to move the drone from the start to the goal smoothly because non-slowing down behavior actually results in having a drone trajectory similar to the trajectory curve with a higher dimension. This was an interesting observation and thus we believe our planned trajectory is very feasible.

IV. Maze Flight Experiments

In the lab we were asked to test our controller and trajectory generator for 3 different cases. In all of the cases we noticed that the drone moves smootly from the initial to final position while avoiding with success all of the obstacles. In the following lines data for each of the case collected from the VICON system of the lab and IMU and a brief analysis of them are presented.

A. MAP #1

For the first map it was asked to control CrazyFlie to start from position $[-1.5, -1.5, 0.9]$ and land in position $[2.5, -1.5, 0.9]$. In Figure 2 we can see the 3D path the drone followed.

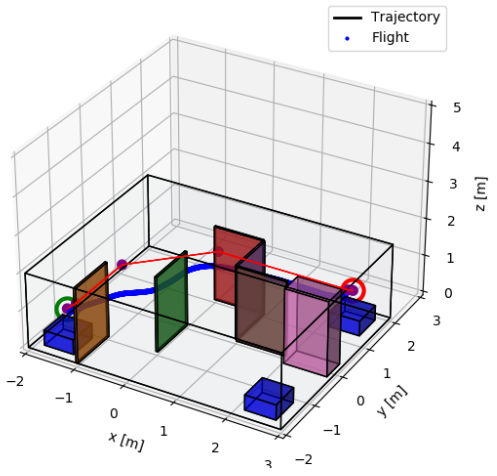

Fig. 2. 3D Path - Map 1

Comments

1) We managed to drive CrazyFlie smoothly through the obstacles from the start point towards the goal point
2) As it can be seen from Figure 2 the actual trajectory (blue) does not diverge a lot from the desired trajectory (red). It is notable to mention that most of the divergence took place when the drone had to turn. In the straight lines it managed to follow the desired path with a great accuracy.
3) Given that the flight was safe and that there was a small divergence from the desired path we could take advantage of these data and recalculate the path to follow. We could experiment by increasing the number of waypoints to follow by making the resolution larger. A small distance between the waypoints and a bad selection of time interval though may lead to even bigger divergence than the accepted ones.
4) For this map, the trajectory approaches a lot the optimal. A possible solution to make our planned trajectory more aggressive would be to omit the second waypoint if we start counting from the green point towards the red point.

B. MAP #2

For the second map it was asked to control CrazyFlie to start from position $[2.5, 1.5, 0.9]$ and land in position $[2.5, -1.5, 0.9]$. In Figure 3 we can see the 3D path the drone followed. In Figure 4 we can see the plot of Position vs Time for this map.

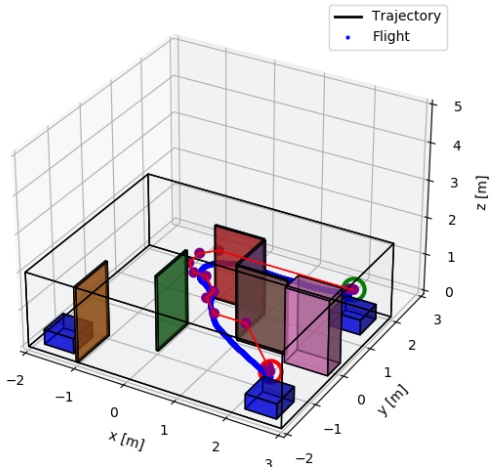

Fig. 3. 3D Path - Map 2

Comments

1) Again the drone managed to reach its desired final position and avoided all of the obstacles
2) There's not much difference between the actual and the desired trajectory. Again most of the divergence happened at the turning points which is quite reasonable since the drone already has a speed and may not be able to track precisely accumulated changes on its path.

3) An increase of the waypoints alongside with a recalculation of appropriate time intervals may lead to even better trajectories
4) Our actual trajectory is almost optimal. The environment of this map was more dangerous than the previous map so safety is crucial. At turning points where there are many obstacles the existence of a lot of points makes sense. When the robot passes through the first turn successfully all it has to do is follow a straight line approximately. So the waypoints after the first turn could probably be even less to allow the robot to increase its speed and reach faster the final goal

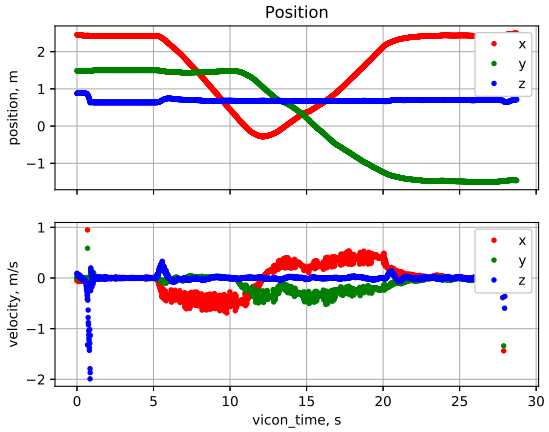

Fig. 4. Position & Velocity vs Time

- The position of the drone smoothly changes until it reaches the final goal
- During the flight, CrazyFlie had a constant speed as it can be seen by the second plot. Small distrubances that appear in the plot are due to noise captured by the VICON system and the IMU sensor.

C. Map #3

For the third map it was asked to control CrazyFlie to start from position $[2.5, -1.5,\ 0.9]$ and land in position $[-1.5, -1.5,\ 0.9]$. In Figure 5 we can see the 3D path the drone followed.

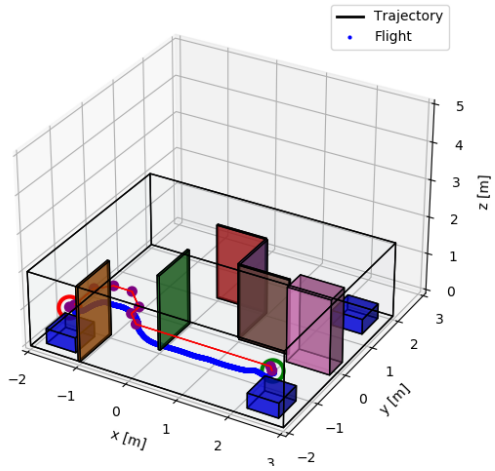

Fig. 5. 3D Path - Map 3

Comments

1) The drone executes with success its assigned task
2) The real trajectory is almost the same with small differences
3) For this case it looks like the remaining waypoints could not be improved further than this. The non existence of a waypoint between the start point and the next waypoint is reasonable allowing the drone to achieve a big acceleration
4) The designed and actual trajectory seem optimal. The only changes that someone could try is to increase the speed of the drone

D. Improvements

Experimenting with the number of waypoints, the distance between them and the time interval would lead to different responses for the drone. The more distance there is between points the more speed the drone can have keeping in mind always the importance of the time interval. A reasonable suggestion would be to eliminate in between waypoints when obstacles do not exist, i.e if there exists a straight path between waypoints A and B without obstacles then any in between waypoint could be omitted. This change would allow flexibility in terms of velocity.

In a possible extra lab experimenting with the speed of the drone would be a first step to test further the robustness of our existing trajectories. Possible changes on the waypoints would probably improve our flight. Finally, a possible lab with a camera attached on the drone would be a great addition.