Project 1 Report: Trajectory tracking with quadrotor control

Dian Chen, Omar Rizkallah, Zihang Zhang

***Abstract*— In this lab we applied the simulated geometry-based controllers and trajectory generator to a real quadrotor, Crazyflie. We had Vicon data streaming to the computer as near ground truth states of the Crazyflie. Some parameter retuning has been made to account for the mismatch between the simulation and actual physics. The three trajectories were successfully tested and some discussion is presented in this report.**

I. INTRODUCTION

In the previous phases we've completed the controllers, trajectory generator as well as the path planner. However, all those work have been done in simulation, in which we had near perfect measurements and physics of the quadrotor. When applying the codes on our real quadrotor, Crazyflie, we no longer have these advantages. The Vicon data feed is noisy even though it is already of high accuracy; the physics of the Crazyflie is also not perfectly aligned with the model in simulation. We first retuned the controller gains to make it able to steadily hover, and then tested three trajectories generated by our generator. All trajectories were closely followed with satisfactory speed which justified our implementation on real quadrotors.

II. CONTROLLERS

To command a quadrotor to reach a position or more generally, to follow a time-specified trajectory, we typically use cascade control (Figure 1). The inner loop is controlled by an attitude controller which takes care of the attitude of the quadrotor to generate desired accelerations. In the outer loop there sits a position controller, which takes in set-points from the trajectory generator and generate desired accelerations to feed into the attitude controller. In the simulation we've implemented and tuned both controllers, however in this lab the attitude controller is integrated into the onboard processor of Crazyflie so we only needed to tune the position controller. In this lab we used geometry-based controllers, pushing away the limitations imposed by linear controllers.

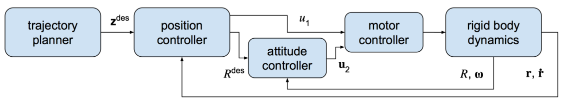

Fig. 1. Cascade control of Crazyflie

A. Position Controller

The position controller we used is a PD controller, which generates accelerations according to the following rule:

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

$$\mathbf{F}^{\text{des}} = m\ddot{r}^{\text{des}} + m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{2}$$

1) Gain tuning: Here K_d and K_p are all diagonal matrices with positive entries. Making use of the near symmetry of the quadrotor in xy plane, we chose to set the gains for x and y direction to be equal. The gains for proportional terms are $K_p = diag\{6.2, 6.2, 25\}$.

The point at which a spring-mass system is critically damped will satisfy $K_d = \frac{1}{\sqrt{2}}K_p$. We chose to set the ration to be 0.6, which would make our position controller have faster response while keeping the overshoot reasonable. So the gains for derivative terms are computed as $K_d = 0.6K_p$.

These gains were obtained from theoretical analysis and trial-and-error. For example, if we observed that the position wasn't converging fast enough, we would try increasing the proportional gains or reducing the derivative gains; if we found that the quadrotor had evident oscillation at hover point, we would try increasing the derivative gains, etc.. Finally the aforementioned values are the gains that achieved good control performance.

2) Modify the gravity: While tuning gains for z (vertical) direction, we noticed a funny behavior: the quadrotor could only converge to a height evidently lower than the actual commanded height. This is due to the mismatch of the suggested mass used in Matlab code (a postulated, ideal value) between the actual mass of the quadrotor. When the actual mass is different from the value in the code, and if we don't use integral terms in our controller, to compensate the load difference (might be negative if the actual quadrotor is heavier) there must exist a gap between the converged position and set-point, i.e.,

$$K_{p,z}\Delta h = \Delta mg \tag{3}$$

To mend the gap, we had to modify the gravity term, making the physics in our code stick to the reality as close as possible. However, modifying m would introduce some coupling to our well-tuned K_d and K_p (see the first term on the right-hand side of equation (2)). A hack to this is to modify the gravity constant instead. After some fine-tuning, we've reached a value which is $g' = 1.12g$. This also justified the observation that the original converging position is lower and the actual quadrotor is heavier.

B. *Attitude Controller*

Although the attitude control is taken care by the onboard processor, we still need to derive the attitude set-points to feed into the attitude controller. For geometric controller, the heading of the quadrotor is expected to align with yaw direction, and $\mathbf{b_3}^{\text{des}}$ should align with $\mathbf{F}^{\text{des}}$. So we have:

$$\mathbf{b_3}^{\text{des}} = \frac{\mathbf{F}^{\text{des}}}{\|\mathbf{F}^{\text{des}}\|} \tag{4}$$

$$\mathbf{a}_\psi = \begin{bmatrix} \sin\psi \\ \cos\psi \\ 0 \end{bmatrix} \tag{5}$$

To find right-handed unit vector $\mathbf{b_2}^{\text{des}}$ which is perpendicular to the plane formed by $\mathbf{b_3}^{\text{des}}$ and $\mathbf{a}_\psi$, we have:

$$\mathbf{b_2}^{\text{des}} = \frac{\mathbf{b_3}^{\text{des}} \times \mathbf{a}_\psi}{\|\mathbf{b_3}^{\text{des}} \times \mathbf{a}_\psi\|} \tag{6}$$

Now we can compute the desired rotation matrix:

$$R^{\text{des}} = \begin{bmatrix} \mathbf{b_2}^{\text{des}} \times \mathbf{b_3}^{\text{des}} & \mathbf{b_2}^{\text{des}} & \mathbf{b_3}^{\text{des}} \end{bmatrix} \tag{7}$$

Having found the desired rotation matrix, we can use the provided helper function to convert this matrix to Euler angles, which in the code reads:

$$euler = rotmat2eulzxy(R\_des); \tag{8}$$

Finally the desired Euler angles $euler$ are fed into the attitude controller and controlled onboard. Due to the rigid body dynamics of the quadrotor, the attitude loop has faster response than the position controller, so we can view the desired attitudes as having been reached before the set-point coming from the outer loop significantly changes.

III. TRAJECTORY GENERATOR

A. *Description of Trajectory Generator*

The function trajectory_generator turns the path which is composed of several waypoints into a trajectory as a function of time.

The trajectory generator works in the following way: at the first call, the function records the path in a persistent variable $path$, then based on the distance between two adjacent points it generates a list of time (also stored in a persistent variable $list\_t$) passing through each waypoint. The larger the distance, the larger the time interval between the points. Then the function tries to generate the trajectory using cubic splines. Each two points are connected by a cubic spline, with two adjacent splines having the same velocity and acceleration at their common point. In this way the trajectory becomes smooth and easy to keep track on. The function achieve this by solving $4(N-1)$ linear equations, where N is the total number of waypoint, the method is described below:

There are totally $N-1$ cubic splines, each one is expressed as follows:

$$x_1(t) = c_{13}t^3 + c_{12}t^2 + c_{11}t + c_{10} \tag{9}$$

$$x_2(t) = c_{23}t^3 + c_{22}t^2 + c_{21}t + c_{20} \tag{10}$$

$$\cdots$$

$$x_{N-1}(t) = c_{(N-1)3}t^3 + c_{(N-1)2}t^2 + c_{(N-1)1}t + c_{(N-1)0} \tag{11}$$

Then write the constraints at: first waypoint:

$$x_1(t_1) = x_1 \tag{12}$$

$$\dot{x}_1(t_1) = 0 \tag{13}$$

Second waypoint:

$$x_1(t_2) = x_2 \tag{14}$$

$$x_2(t_2) = x_2 \tag{15}$$

$$\dot{x}_1(t_2) = \dot{x}_2(t_2) \tag{16}$$

$$\ddot{x}_1(t_2) = \ddot{x}_2(t_2) \tag{17}$$

$$\cdots$$

Last waypoint:

$$x_{N-1}(t_N) = x_N \tag{18}$$

$$\dot{x}_{N-1}(t_N) = 0 \tag{19}$$

There are $4(N-1)$ unknowns of c_{ij} and 2 constraints at the first and the last waypoint, 4 constraints at each way-points in middle, therefore a total of $4(N-1)$ constraints. So *Matlab* should be able to solve this linear system by encoding the equations in matrix form. This process is repeated in y z coordinates to find the corresponding trajectory equations. And the parameters solved are also stored in a persistent variable c.

When the function is called after the first time with no path input, it will compare the time input with the list of times stored and find the corresponding cubic spline. Then it can compute the coordinates by recalling the stored parameters.

Because the trajectory is defined as combination of several cubic splines, the acceleration profile should be linear at each cubic spline, thus it is continuous but not smooth. As a result, The velocity profile should be composed of several second polynomials and is continuous and smooth.

B. *Results of Trajectory Generator*

Three trajectories were planned and run in lab. The 3d plot of each trajectories are shown in figure 2 to figure 4.

For the third trajectory, the plot of positions, velocities and accelerations with respect to time are shown in figure 5 to figure 7, where the blue lines represent planned trajectory and red lines represent actual trajectory.

The planned distance, actually flown distance, total time and average velocity are computed for each trajectories and the results are shown in Table I.

Dataset	Distance [m]	Planned distance [m]	Time [s]	Velocity [m/s]
Trajectory 1	6.6818	6.3065	22.0519	0.3022
Trajectory 2	10.4794	10.3467	21.1415	0.4960
Trajectory 3	5.5141	5.4061	20.5321	0.2658

TABLE I

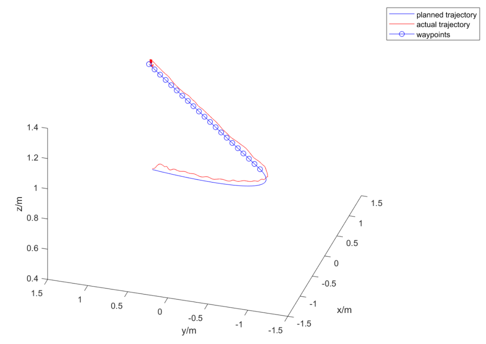

Fig. 2. First trajectory

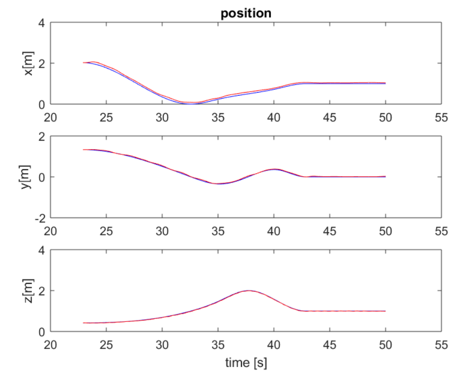

Fig. 5. Position plot of the third trajectory

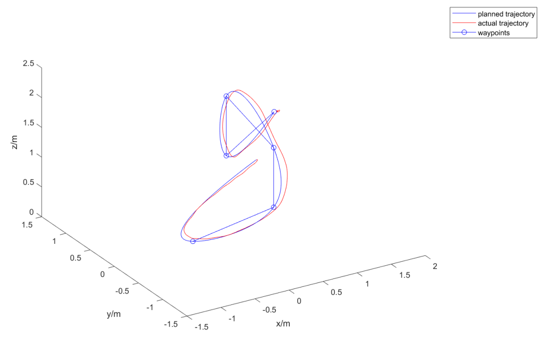

Fig. 3. Second trajectory

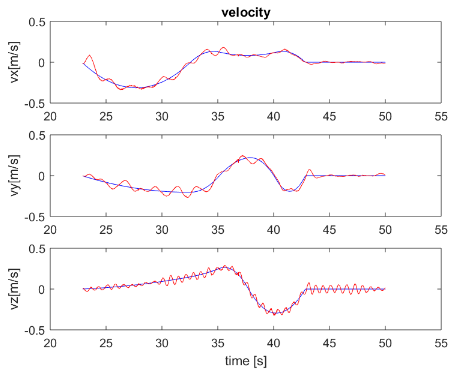

Fig. 6. Velocity plot of the third trajectory

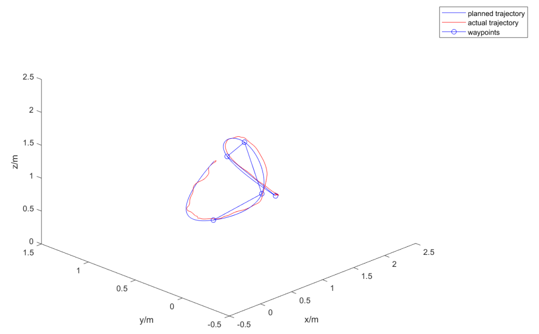

Fig. 4. Third trajectory

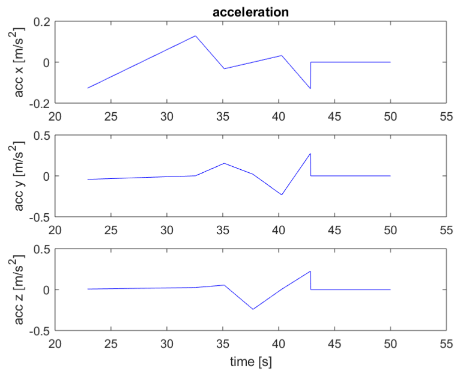

Fig. 7. Acceleration plot of the third trajectory

IV. DISCUSSIONS

A. Discussion of Controller

Our controller can be considered successful by considering the position graphs of the three flights. The flight trajectory was highly consistent with the planned trajectory in this regard. There is however a small positive x offset in each flight that could have been caused by a systematic error caused by the in-built measurement of the state. This could have been resolved by adding a correctional factor in the controller, similarly to how the gravity coefficient was manipulated. However, when considering the velocity profiles of our flights, we can find areas for improvement. In flight 2, as shown in figure 12 to figure 14, we find a highly pronounced oscillatory behavior to the velocity in the z direction, particularly after the 33rd second. This can also be seen in flight 4 and flight 3 to a lesser extent. To tackle this, we could slightly lower the response to a derivative error in the z direction in order to reach a point of damping that could diminish these oscillations.

B. Discussion of Trajectory Generator

There are two factors that influence the performance of the trajectory generator, the first is the smoothness of the trajectory. In order to have a smooth trajectory the velocity profile should be at least continuous. This is attained by constructing cubic splines between points. Another reasonable way to achieve this is by Bang-(Coast)-Bang Segments, which links the waypoints by straight lines and makes each segments start and end at 0 velocity.

The second factor is the magnitude of acceleration. If at some point the acceleration is too high, the controller will not be able to response so fast thus it will cause large overshoot and even failure in tracking the path. An example is shown in figure 8, where the trajectory generator naively divide the time equally for each line segment. However, the distance between the first and the second waypoint is very large which leads to a high acceleration at this segment, therefore, the quadrotor is not able to follow the path. There also seems to be an initial velocity requirement in 3 dimensions, which will caused an initial derivative error as the quadrotor must start from rest. This error might work against the positional error and prevent the execution of the planned trajectory. This is the reason why it is important to divide the time interval based on the distance between waypoints as well as establish smooth transitions through different phases (no initial offsets). This trajectory approach was later replaced by one that factored in distance to tackle this problem.

In our flights, we can also observe that, due to the nature of the cubic spline approach, velocity is continuous but jerk is not, meaning certain transitional situations from one curve to another can cause high jerk and result in a difficulty in following the trajectory's velocity profile. This is particular in flight 2 (Figure 13) at the 33rd second when zero velocity was commanded, as well as in the 24th second in a transition between two curves. In flight 3, there is such an occurrence in the 37th second in the x and y dimension where velocity is set to 0 in an abrupt manner. To tackle this, a fifth order curve could have been used between waypoints that could also guarantee that at points where a zero velocity is needed, a zero acceleration could also be set to guarantee stability of that velocity and no overshoot. It will also lessen the jerk on the quadrotor and allow for tighter maneuvering. This generally more accurately mimics the quadrotor's behavior since the quadrotor can only increase change acceleration through a gradient, and cannot reach any acceleration immediately. Thus, the key areas of improvement were first the magnitude of acceleration, which we tackled after our first test, and the implementation of trajectory generator generating positional curves of the fifth order with respect to time.

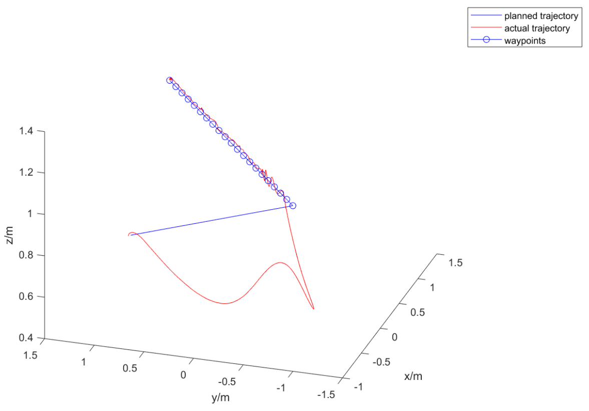

Fig. 8. Failure at first trajectory if equally dividing the time

V. CONCLUSIONS

We conclude that our trajectory generator was generally successful in the goals set forth for successfully passing through the waypoints and reaching the endpoint and ultimately coming to rest. These goals were all accomplished through our use of time and distance differentiated waypoints connected by cubic splines. However, we could have optimized for the minimization of jerk to allow the quadrotor to more smoothly alter its acceleration to prepare for phase changes from one waypoints to another. In our controller, it was able to successfully maneuver this high-jerk trajectory successfully, maintaining a close proximity to the intended path throughout the entire flight, despite a small offset error. In the future we could consider testing our system against more sudden changes, where jerk might be more of an issue, as well as demand accelerations that might push the quadrotor to its limits.

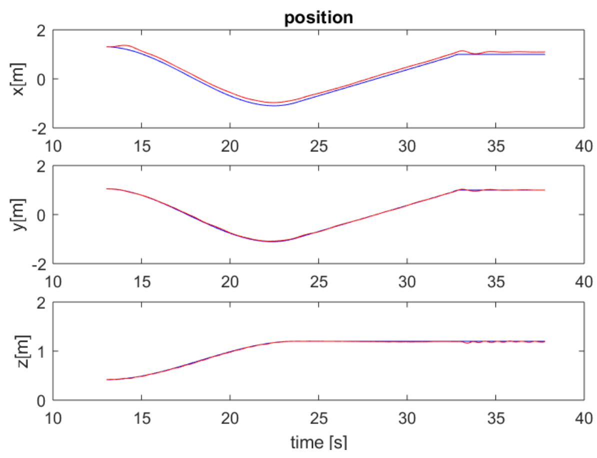

Fig. 9. Position plot of the flight 1

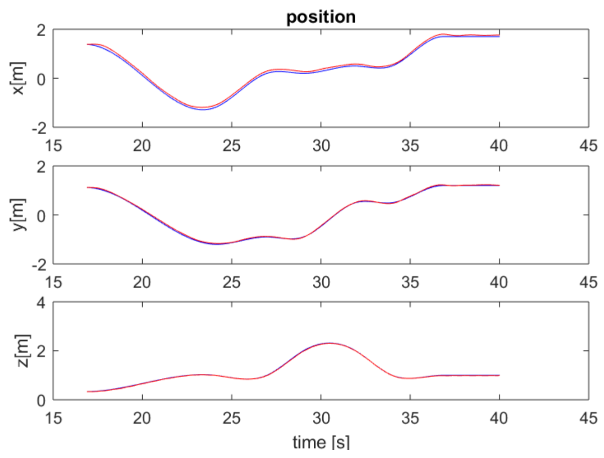

Fig. 12. Position plot of the flight 2

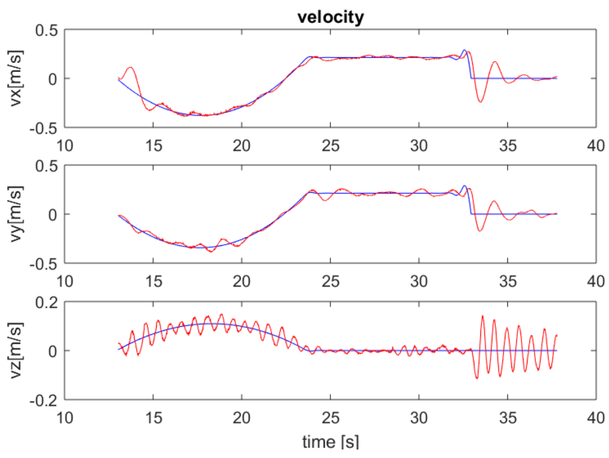

Fig. 10. Velocity plot of the flight 1

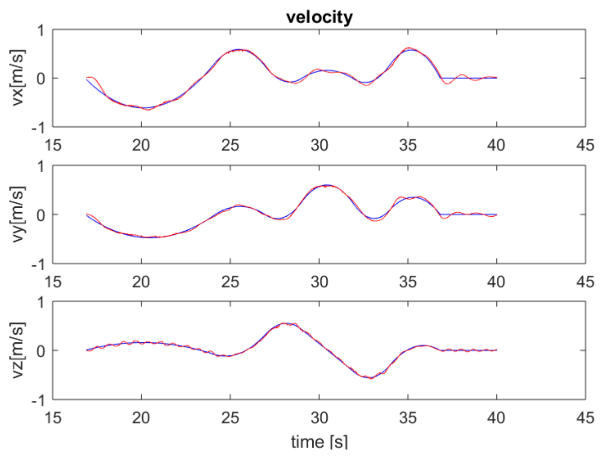

Fig. 13. Velocity plot of the flight 2

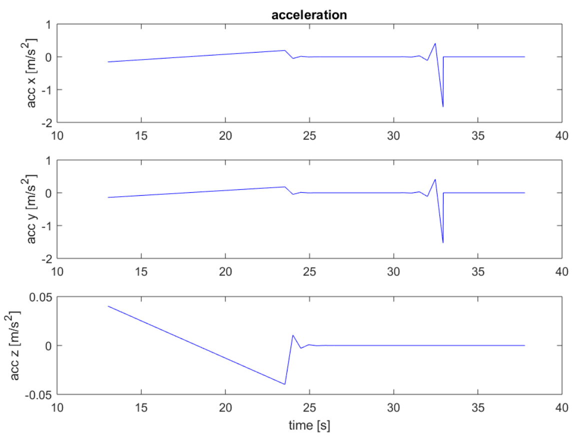

Fig. 11. Acceleration plot of the flight 1

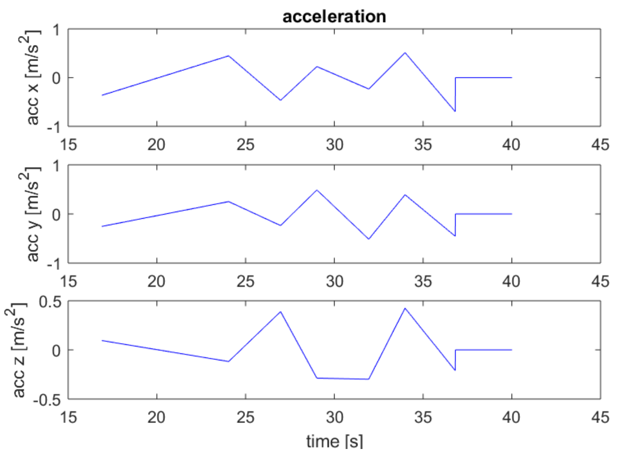

Fig. 14. Acceleration plot of the flight 2