Trajectory Tracking with Quadrotor Control

Sarah Allen, Kashish Gupta, Ken Hayashima

***Abstract*— Maneuvering a quadrotor in a lab environment is quite different from theoretical trajectory planning in simulation. After modifying trajectory generator to take in a dynamic number of waypoints, we tuned our controller to work well with our specific real life quadrotor in the lab. We found out that we also needed to change the value of gravity compensation because of the difference of mass between the actual quadrotor we used and the one provided. To run the trajectories, we used a geometric nonlinear controller and a minimum acceleration trajectory generator. At this point, our quadrotor succeeded in traversing all 3 trajectories in the lab, but we realized there was an offset between our quadrotor's position and the desired position (especially in the X dimension), which may be due to an imperfect center of mass of the quadrotor. To solve this, we changed the desired ϕ and θ in our controller. As a result, our trajectory was even closer to the desired one as measured by the sensors in lab.**

I. INTRODUCTION

In order to fly the quadrotor as we expect, we have to set up a controller with newly tuned parameters and a dynamic trajectory generator. Simulation differs from reality, so we had to retune the gains and add a coefficient to mass. This required trial and error in real world testing until the tuning was correct.

The trajectory generator worked out of the box but we had to tune the speed of the quadrotor. The trajectory generator was developed with a constant average speed for each leg of the trajectory, and we found that slower speeds were less stable than medium speeds.

Lastly, we flew the quadrotor along the trajectories given to us in lab. We found that for certain trajectories, removing co-linear points had huge performance gains. In addition, we tuned the speed of the quadrotor differently for each trajectory in order to maximize the trade-off between stability and speed.

II. THE CONTROLLER

A. How the Controller Works

Our controller was the geometric nonlinear controller. A nonlinear controller allows for more aggressive maneuvers and faster flight times over the linear approach. Ultimately, the purpose of the controller is to find thrust control (u_1) and moment control (u_2). The thrust control comes from a desired force along the quadrotors z axis (i.e. b_3), while the moment control comes from driving the quadrotor to a desired rotation matrix.

To begin the process, one must develop a PD controller for the command acceleration. This equation for the nonlinear controller is similar to the linear controller equation, just written in vector form:

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

Where r and $\dot{r}$ are the current position and velocity of the quadrotor; r_T, $\dot{r}_T$, and $\ddot{r}_T$ are the desired position, velocity, and acceleration; and K_d and K_p are diagonal, positive definite gain matrices.

Next, using this command acceleration, one can find the command force, remembering to include gravity. As discussed below in Subsection B, the gravity term was increased to obtain more accurate flight:

$$F^{des} = m\ddot{r}^{des} + 1.13 \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

Where m is the ideal mass of the quadrotor and g is the acceleration due to gravity.

By using the fact that the thrust control is along the rotated z axis ($b_3 = R \begin{bmatrix} 0, 0, 1 \end{bmatrix}^T$, where R is the current rotation), one can find the thrust control by projecting the command force onto this axis:

$$u_1 = b_3^T F^{des}$$

After finding the thrust control, one can find the moment control. The moment control is found through the development of the desired rotation matrix, $R^{des} = \begin{bmatrix} b_1^{des}, b_2^{des}, b_3^{des} \end{bmatrix}$. The easiest axis to find is b_3^{des}, as it must be in the same direction as the control thrust:

$$b_3^{des} = \frac{F^{des}}{||F^{des}||}$$

The b_2^{des} axis must be perpendicular to both b_1^{des} and the yaw direction vector, $a_\psi = \begin{bmatrix} \cos(\psi_T), \sin(\psi_T), 0 \end{bmatrix}^T$:

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

Finally, since b_2^{des} and b_3^{des} are now known, b_1^{des} is easy to evaluate, as it must be perpendicular to both b_2^{des} and b_3^{des}. Thus, the desired rotation matrix is:

$$R^{des} = \begin{bmatrix} b_2^{des} \times b_3^{des}, b_2^{des}, b_3^{des} \end{bmatrix}$$

The moment control must force the current rotation to this desired rotation. In other words, the error between the current and desired rotation matrices must go to zero. The nonlinear controller forces this by setting the moment control to be:

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

Where I is the quadrotor's moment of inertia; e_ω is the angular velocity error, defined to be the current angular velocity ω; K_r and K_w are diagonal gain matrices; and e_R is the error in orientation, defined as:

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

After careful deliberation, using the process defined in Subsection B, the following gains were selected:

Gain Name	Gain Property
K_p	8
K_d	5.6
K_R	150
K_ω	7

B. Tuning the Controller's Gains

Since the attitude control is already incorporated, we only tuned position gains. The process of tuning is as follows. First, we hovered the quadrotor using the given code to see how it is deviated from its desirable position in X,Y,Z directions. As a result, we saw that the deviation in the Z direction is the largest whereas the deviation in X and Y directions were relatively small (this makes sense because the quadrotor is told to travel only in the Z direction). This difference results from the fact that the actual mass of quadrotor we used is different from the mass provided in the handout by 13% and additionally because the thrust from the propellers may be different from what is optimal and specified by the CrazyFlie. Therefore, we increased the gravity compensation by 13% in the Z direction by multiplying the gravity term by 1.13.

After tuning the gravity coefficient for appropriate position in the Z dimension, we needed to tune the position gains (both proportional and derivative). We started with a baseline of derivative gain as 40%Though we started with XY gains set equal to each other, and with a larger Z dimension gain, eventually we simplified our tuning to have equivalent gains in the X, Y, and Z dimensions since we modified the gravity compensation already. We furthermore decided to set our derivative gain as a direct ratio to our derivative gains, which further simplified our process. Thus, we needed to decide on one value for all of the proportional gains, and a single ratio that would determine all of the derivative gains, rather than tuning six gains separately.

In order to tune proportional gains we brought the quadrotor to a stable hovering position and then physically pushed it in one plane at a time to see how easily it normalized itself. When we saw overshooting, we decreased proportional gain, and when we saw that the quadrotor was taking too long to achieve a the desired waypoint, we increased proportional gain. Next, we tuned the derivative gains. At first we set the derivative gains to be 40% of the proportional gains. However, we saw that there was still oscillation in the XY direction, so we increased the ratio to 70% which dampened the movement of the quadrotor, slowing it down, but fixing the oscillation. To finish up, we increased the proportional gain to maximize how quickly the quadrotor reaches the desired point. Our final proportional gain (K_p) was 8.0 for all XYZ, and our final derivative gain (K_d) was $0.7K_p$ for all XYZ. This took about an hour to complete, and we decided to move on, as the additional benefits would have been marginal.

C. Observation and Bias Tuning

Regardless of paths given, we found out that the quadrotor traces out trajectories with a constant offset, particularly in the x and y directions. In other words, immediately after starting, the quadrotor deviates from planned trajectory until stabilizing and then starts to fly parallel to the planned path. This can be attributed to the fact that the actual center of mass is slightly different from the theory, and therefore the IMU reading was slightly incorrect.

Therefore, we decided to compensate for this bias by changing desired ϕ and θ and see how the offset decreases. After trying multiple biases, we found that subtracting .06 from the original ϕ and adding .02 to the original θ produced the best results. Below are the graphs comparing the trajectories traced out by the controller incorporating biases (biased) and the original one (unbiased).

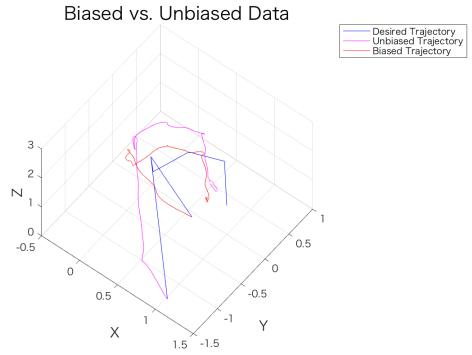

Fig. 1. Biased vs Unbiased Waypoints 3

Since we have different desired ϕ and θ between two controllers, the actual trajectories of two controllers are different, as can be seen in the graph above.

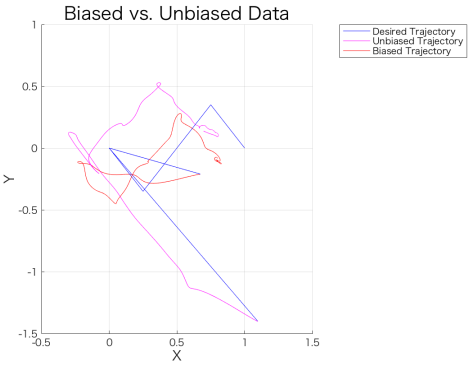

Fig. 2. Biased vs Unbiased Waypoints 3 on XY plane

When looking at the errors on XY plane, it appears that the error of the biased controller is smaller. To verify this, we decided to do error analysis in X,Y and Z directions.

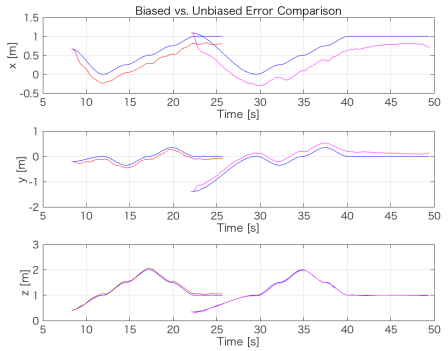

Fig. 3. error analysis between ed and unbiased

When we compare the decomposed errors in X,Y and Z directions, we can see that our biased controller (in red) provided trajectories that were closer to the desired trajectory (in blue) than the unbiased trajectory (in magenta) especially in X direction. However, error still exists. This could be due to slightly off bias values, or other unknown errors inherent to the controller. For example, in the y direction, the unbiased system overestimates the desired trajectory, while the biased systems underestimates the desired trajectory. The fact that the error flipped from one side of the desired trajectory to the other suggests that corresponding bias is too severe.

III. TRAJECTORY GENERATOR

A. How we Built the Trajectory Generator

To begin, we already already built a trajectory generator that could follow set paths such as a circle or rectangle. Additionally, we had already built an implementation of Djikstra's/A* that would generate the shortest path in a discretized coordinate system between a start and end point. The goal was to build a trajectory generator that could take in a dynamic number of waypoints each specified as XYZ positions, and give instructions to the quadrotor to go through each of those waypoints in a single path. Then we could easily combine the Djikstra's/A* implementation with the trajectory generator to guide the quadrotor along any trajectory offered to us. Before lab, we were able to build such a trajectory generator and test that it worked well in simulation.

The first step was to take the set of input points and to remove co-linear points. In order to do this, we test whether the cross product between evey consecutive pair of displacement vectors is 0. If the cross product is 0, this implies that the angle between the vectors is 0, which means that the three points defining the vectors are co-linear. In cases like these, we remove the middle point and test the next pair assuming this middle point was removed. Completing this process removes all co-linear points in our trajectory, making the trajectory significantly more smooth as the quadrotor does not need to speed up and slow down as many times.

We then implemented a multi-segment minimum acceleration trajectory generator by specifying a start position for $t = 0$ and then building the rest of the paths piecewise. In order to ensure smoothness and continuitity between these piecewise equations, we forced the quadrotor to start from rest and end at rest at each of the waypoints. To find these minimum acceleration paths we used the following equation:

$$\begin{bmatrix} a \\ b \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & T & T^2 & T^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2T & 3T^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Next, we precompute how much time we want the quadrotor to spend on each segment based on distance. To do this, we know that the distance for each leg of the trajectory is simply the L2-norm of the ending waypoint minus the starting waypoint. Then, using a constant speed that we set beforehand, we can calculate the time spend on this leg of the trajectory because $v = \delta x/t$. Using this equation we precompute all of the times for each segment of the trajectory.

Then we precompute the time at which each segment of the trajectory should begin, i.e. a starting time for each leg of the trajectory. To do this, we simply keep a counter of the aggregate starting times and then store them into an array. By looking for the first index where the current time of the quadrotor is greater than the start time in the stored array, one can determine which part of the trajectory our quadrotor is on. Using this information, one can find both the starting and ending waypoints of that leg of the trajectory as well as the time T the quadrotor will spend on that portion of the path.

We set $a = 0$ and $b = 1$ for each segment of the trajectory, so that the starting waypoint would be at relative time 0 and the ending waypoint would be relative time 1 for this segment. Then, we solve the matrix above using the inverse in matlab to solve for the coefficients c_0 to c_3. Using these coefficients, we can find the equation for the desired position of the quadrotor during this segment:

$$x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

Furthermore, by taking derivatives of the above equations:

$$v(t) = c_1 + 2c_2 t + 3c_3 t^2$$

$$a(t) = 2c_2 + 6c_3 t$$

desired velocity and acceleration of the quadrotor can be solved for, which will be fed into our controller.

This process is repeated for every leg of the quadrotor's trajectory, until the if condition satisfying end of trajectory is satisfied, at which point the quadrotor is told the maintain the final position with an acceleration of 0 and a velocity of 0.

This completes our implementation of a dynamic trajectory generator that minimizes acceleration and runs at a constant average speed. In simulation, we had success with an average speed of $0.5m/s$, and in experimentation we were able to boost the speed anywhere between $0.5 - 2.0m/s$ for trajectories with greater distance between waypoints.

IV. OUR RESULTS

Upon combining our implementation of the controller and trajectory generator as specified in the lab handout, tuning gains and mass coefficient, and testing the $hover.m$ trajectory, we began testing the 3 trajectories given to us by the instructors.

For the first set of waypoints, we ran the trajectory with a speed of $0.5\ m/s$ and found that the path was very unstable: the quadrotor was wobbling because it was moving so slow. When we increased the speed to about $1.5\ m/s$, shown below, the whobbling decreased.

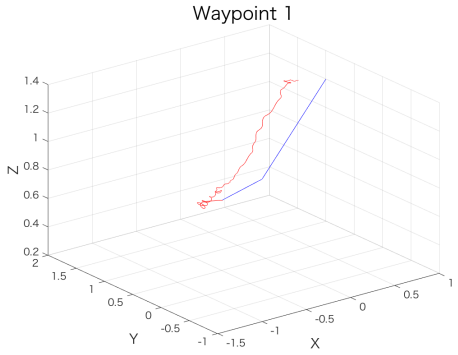

Fig. 4. Plot of Waypoints 1

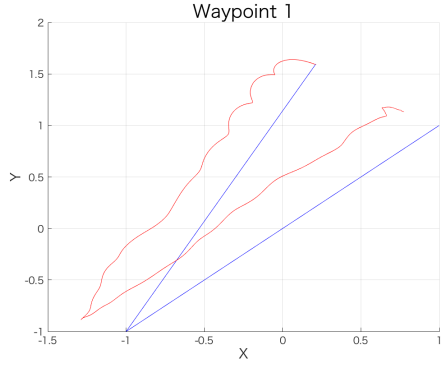

Fig. 5. Plot of Waypoints 1 on XY plane

For the first trajectory, we had very little error in the Y and Z dimensions, and as explained earlier, the X dimension is offset because of a bias.

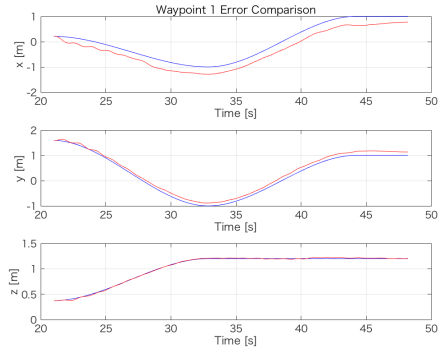

Fig. 6. Error analysis of Waypoints 1

We then repeated the same procedure for the second set of waypoints. Since there were more waypoints in this trajectory, we used a slower speed than trajectory 1.

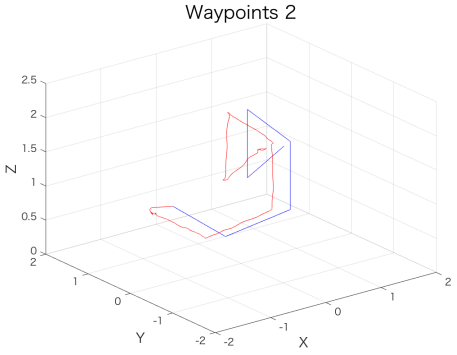

Fig. 7. Plot of Waypoints 2

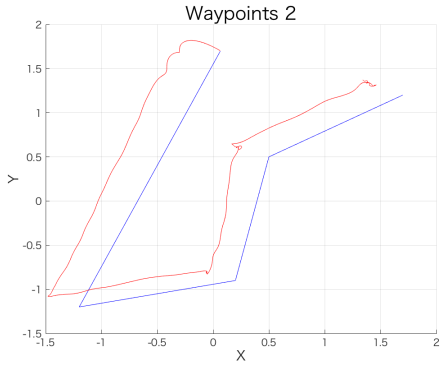

Fig. 8. Plot of Waypoints 2 on XY plane

Again we see a very low error except in the X dimension.

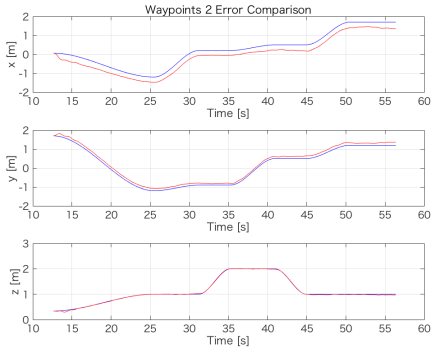

Fig. 9. Error analysis of Waypoints 2

At this point we decided to tune ϕ and θ because we wanted to fix the offset in the X dimension. The results of doing so are published in section 2C, but for the purposes of showing our results on the trajectories, we removed our gains associated with fixing the offset because we had to switch to a different quadrotor that had more battery. The ϕ and θ gains were specific to the quadrotor that we were testing them on, so in order to run the 3rd trajectory, we removed those gains and reintroduced the X dimension offset, because keeping those gains worsened the performance. Below are our results for the 3rd trajectory:

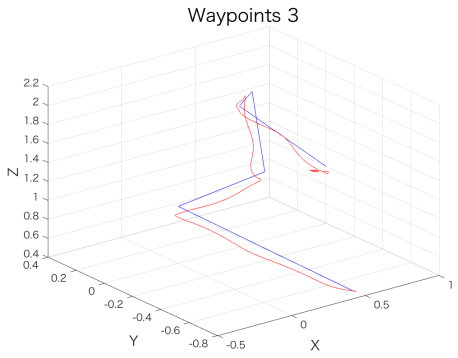

Fig. 10. Plot of Waypoints 3

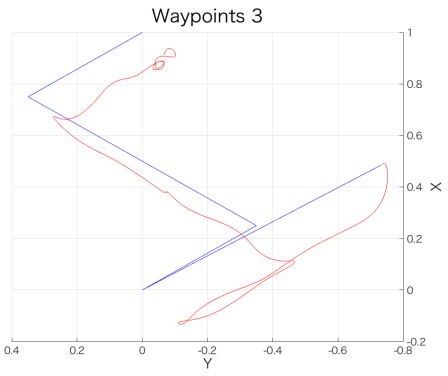

Fig. 11. Plot of Waypoints 3 on XY plane

For this new quadrotor, we see an increased offset in the Y dimension and a similar offset in the X dimension, but otherwise the trajectory follows the desired trajectory closely.

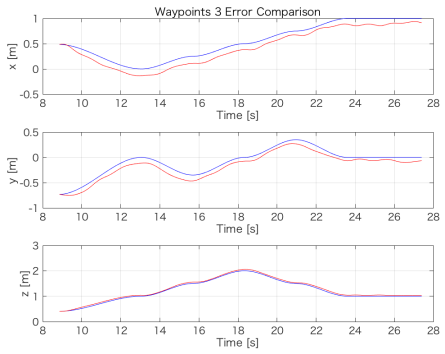

Fig. 12. Error analysis of waypoints 3

V. CONCLUSIONS

Our group decided on a nonlinear controller due to its ability to move more aggresively than the linear controller. While the equations are ultimately more complicated, we felt that this type of controller was necessary in order to minimize the error as much as possible.

This lab taught us a lot about the differences between the real world and simulation. In simulation, our gains were perfectly tuned, but to fly the quadrotor as we expected in real life, we had tune the gains again. In addition, we had to add a coefficient to the mass of the quadrotor because the real mass of the quadrotor was different from expected. After many iterations of trial and error we were able to come up with acceptable gains and a coefficient for mass that allowed us to move forward and test the trajectories.

When testing the trajectory generator in lab, we realized that it had been programmed with a constant average speed for each trajectory. This was a safe bet when choosing slow average speeds, but for trajectories that had waypoints that were spaced far apart, it was not optimal and in fact resulted in unstable traversing of the trajectory. Thus we changed the speed of each trajectory based on the number of waypoints. The trajectory generator was then able to take in any dynamic number of waypoints, remove co-linear waypoints, and give instruction to the quadrotor to traverse the trajectory while minimizing acceleration.

While testing the trajectories given to us in lab, we found that we had little to no error in the Y and Z dimensions, but had a constant offset in the X dimension. Though we were instructed that this was normal, we decided to try and improve our results by tuning ϕ and θ in our controller. We achieved moderate success doing this, but eventually removed the gains and allowed the offset. The results of our experiment were successful, and we found especially that removing co-linear points was helpful for certain trajectories, and that increasing speed was helpful for other trajectories.