Project Report

Jiahe Xu (jxu109@jhu.edu) Ray Zhang (zzhan190@jhu.edu)

1 Introduction

This project is intended to implement planning and control of a quadrotor in 3-D indoor environments. In this report, we will discuss how we design a trajectory and how we use a controller to follow the trajectory. We use parameters of CrazyFile2.0 (a real robot) in our work.

2 Modeling

The coordinate systems and free body diagram for the quadrotor are shown in Fig. 1. Since The heading (*yaw*) angle of the robot can be chosen freely without directly affecting the robot's dynamics, we use *Z-X-Y* Euler angles to describe the rotation transform.

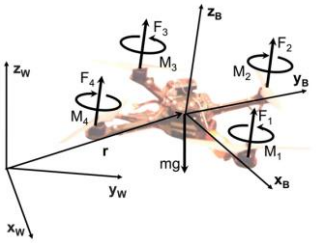


Figure 1, Coordinate systems and forces/moments acting on the quadrotor from [1].

To get from *W* (world frame) to *B* (body frame), we first rotate about z_W by the *yaw* angle ψ, then rotate about the intermediate *x*-axis by the *roll* angle *φ*, finally rotate about the y_B axis by the *pitch* angle θ. The rotation matrix for transforming coordinates from *B* to *W* is given by:

$$^{W}R_{B} = ROTZ * ROTX * ROTY \tag{1}$$

Where *ROTX, ROTY* and *ROTZ* are rotation matrix that only rotate about X-axis, Y-axis and Z-axis. The center of mass in the world frame is denoted by vector ***r***. In the system, forces are gravity, in the $-z_W$ direction, and forces come from rotors F_i, in the z_B direction. The equation involves the acceleration of the center of mass are:

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{W}R_{B} \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} F_i \end{bmatrix} \tag{2}$$

Then, we have control input $u_1 = \sum_{i=1}^{4} F_i$.

The angular velocity of the robot in the body frame is vector $[p, q, r]^{\mathrm{T}}$, it can be calculated by derivatives of the *roll, pitch,* and *yaw* angles with:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\cos\varphi\sin\theta \\ 0 & 1 & \sin\varphi \\ \sin\theta & 0 & \cos\varphi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3}$$

As for forces, each rotor generates a moment perpendicular to the x_B-y_B plane. Rotors 1 and 3 (in figure 1) rotate in the $-z_B$ direction while rotor 2 and 4 rotate in the z_B direction. We let *L* be the distance from the axis of rotation of the rotors to the center of the quadrotor. We denote *I* as the moment of inertia matrix referenced to the center of mass along the x_B, y_B and z_B axes.

Each rotor has an angular speed ω_i and produces a force F_i according to:

$$F_i = k_F \omega_i^2 \tag{4}$$

The moment produced by rotors is:

$$M_i = k_M \omega_i^2 \tag{5}$$

The angular acceleration determined by the Euler equations is:

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{6}$$

we can rewrite it as:

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix}F_1\\ F_2\\ F_3\\ F_4\end{bmatrix} - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{7}$$

where $\gamma = {}^{k_M}/_{k_F}$

Then, we can have our second control input u_2:

$$u_2 = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix}F_1\\ F_2\\ F_3\\ F_4\end{bmatrix} \tag{8}$$

Finally, we have the quadrotor's equations of motion:

$$m\ddot{r} = \begin{bmatrix}0\\ 0\\ -mg\end{bmatrix} + {}^{W}R_B \begin{bmatrix}0\\ 0\\ u_1\end{bmatrix} \tag{9}$$

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = u_2 - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{10}$$

3 Trajectory Generator

In our project, we first generate a path represented by a set of 3-D points, which minimize the distance from the start point to the goal, and then we use a flat output method (minimum jerk polynomial segments methods) to build a polynomial function to represent the trajectory.

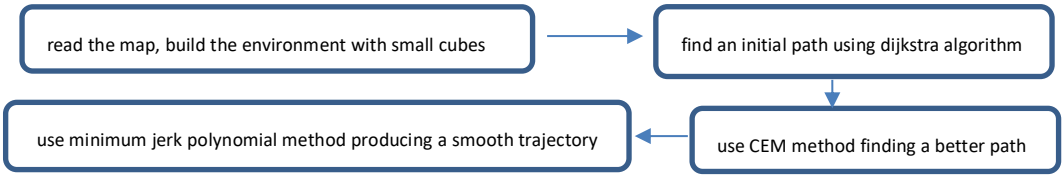


Figure 2. flowchart of path planning and trajectory generation

We first use dijkstra algorithm finding an initial path, then use cross-entropy method finding a better path which is close to the theoretical shortest path. Notice that we only need to find 'corner' knots that change the path's direction to represent a path (the path consists of straight lines). After finding those turning points in the path, we directly connect knots with straight lines by adding more knots in between.

The path generation method mentioned above does not give us a very smooth trajectory especially going through sharp corners. After doing research, we decided to use minimum jerk polynomial segments method to optimize the trajectory [2].

The minimum jerk polynomial segments methods segmented the path into numbers of segments. The quadrotor is assumed to move at a constant speed and the time at each node will be recorded based on the distance between two nodes and the speed. The jerk is the state of the third order system, and it must satisfy all the boundary conditions. The constrains of the position, velocity, acceleration, and jerk can be represented by

Position: $$pos(t) = t^5c_0 + t^4c_1 + t^3c_2 + t^2c_3 + t^1c_4 + c_5$$

Velocity: $$\dot{pos}(t) = 5t^4c_0 + t^3c_1 + t^2c_2 + t^1c_3 + c_4$$

Acceleration: $$\ddot{pos}(t) = 20t^3c_0 + t^2c_1 + t^1c_2 + c_3 \quad (11)$$

Jerk: $$\dddot{pos}(t) = 60t^2c_0 + 24t^1c_1 + c_2$$

Snap: $$pos^{(4)}(t) = 60t^2c_0 + 24t^1c_1 + c_2$$

After applying the initial and end boundary conditions, we will update the constrains for each individual segment. For each segment, it will start with representation of segment k ending at p_k at time t_k. Then it will represent segment k+1 starting at p_k at time 0 and other constrains. We can represent each segment with the following matrix equation. The fundamental of the minimum jerk method is to minimize all the derivatives of the polynomial and make them 0.

$$\begin{bmatrix} t^5 & t^4 & t^3 & t^2 & t & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 5t^4 & 4t^3 & 3t^2 & 2t & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 20t^3 & 12t^2 & 6t & 2 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 60t^2 & 24t & 6 & 0 & 0 & 0 & 0 & 0 & -6 & 0 & 0 & 0 \\ 120t & 24 & 0 & 0 & 0 & 0 & 0 & -24 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} p_k \\ p_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (12)$$

The spline will go through n-1 segments which will results in 6 (n-1) constrains and unknowns. After solving the matrix equation, we substitute the coefficients back to equation 11 to calculate position, velocity, acceleration, jerk, and snap every time we update the desired state.

We could also improve the trajectory generator by applying minimum snap polynomial segments methods by increasing the polynomial to 7th. However, for this project specifically, the minimum jerk method is adequate to generate a rather smooth trajectory.

4 Controller Description

In terms of controller design, we used linear backstepping controller with a combination of position controller and angle controller. The trajectory generator is going to pass the desired states to the controller. The linear control equation for the first part of the controller,

$$(\ddot{x}_{des} - \ddot{x}_T) + k_d(\dot{x}_{current} - \dot{x}_T) + (x_{current} - x_T) = 0 \quad (13)$$

Substituting $\ddot{x}_{des}$ back to equation 9 will conclude the position controller and calculate u1.

$$u1 = m \cdot \ddot{x}_{des,Z} + m \cdot g \quad (14)$$

As mentioned previously, u2 is the moment of the robot. The angle controller has a linear control equation

$$u_2 = I(-k_{d2} \cdot (\omega_{current} - \omega_{des}) - k_{p2} \cdot (angle_{current} - angle_{des})) \quad (15)$$

The angle controller will take the desired acceleration $\ddot{x}_{des}$ directly from position controller and use the following equations to calculate the desired roll, pitch, and yaw.

$$roll_{des} = \frac{1}{g}(\ddot{x}_{des}(1) \cdot \sin(yaw_T) - \ddot{x}_{des}(2) \cdot \cos(yaw_T)) \tag{16}$$

$$pitch_{des} = \frac{1}{g}(\ddot{x}_{des}(1) \cdot \cos(yaw_T) + \ddot{x}_{des}(2) \cdot \sin(yaw_T)) \tag{17}$$

$$yaw_{des} = yaw_T \tag{18}$$

And the desired angle can be presented with the following equation

$$angle_{des} = \begin{bmatrix} roll_{des} \\ pitch_{des} \\ yaw_{des} \end{bmatrix} \tag{19}$$

Since we assume there will be no angular acceleration in roll and pitch, the angular velocity is

$$\omega_{des} = \begin{bmatrix} 0 \\ 0 \\ \dot{yaw}_T \end{bmatrix} \tag{20}$$

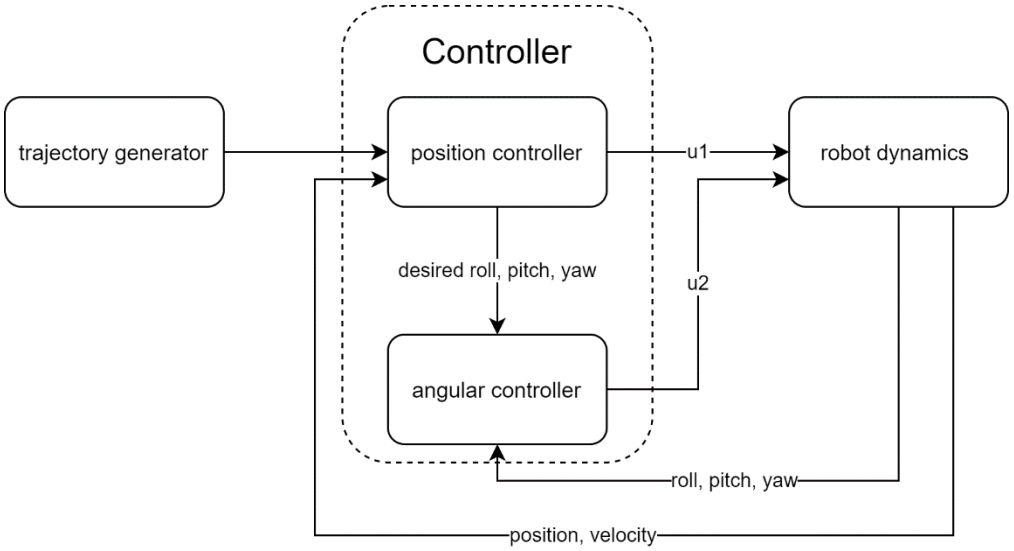


Figure 2. Linear backstepping design for the quadrotor

The robot dynamics (EOM) will then take the controller outputs and the states to calculate the $\dot{state}$, which will be used to update the trajectory using ode45.

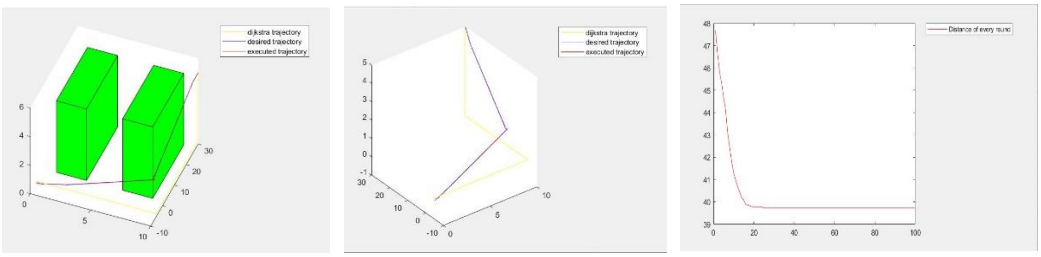
$$\ddot{pos} = \frac{1}{m}\left(\begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix} + R\begin{pmatrix} 0 \\ 0 \\ u1 \end{pmatrix}\right) \tag{21}$$

$$\dot{\omega} = \begin{bmatrix} \ddot{roll} \\ \ddot{pitch} \\ \ddot{yaw} \end{bmatrix} = I^{-1}\left(u2 - \begin{bmatrix} \dot{roll} \\ \dot{pitch} \\ \dot{yaw} \end{bmatrix} \times \left(I \cdot \begin{bmatrix} \dot{roll} \\ \dot{pitch} \\ \dot{yaw} \end{bmatrix}\right)\right) \tag{22}$$

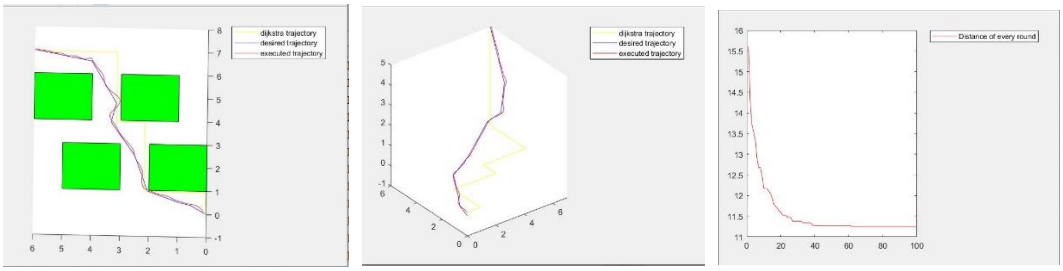
5 Experiment & Result

We have 4 test data, which will be more difficult from 1 to 4. Quadrotor's position should stay inside the boundary. Obstacles (green cubes), trajectories and the curve of distance in each CEM iteration are shown in the following pictures.

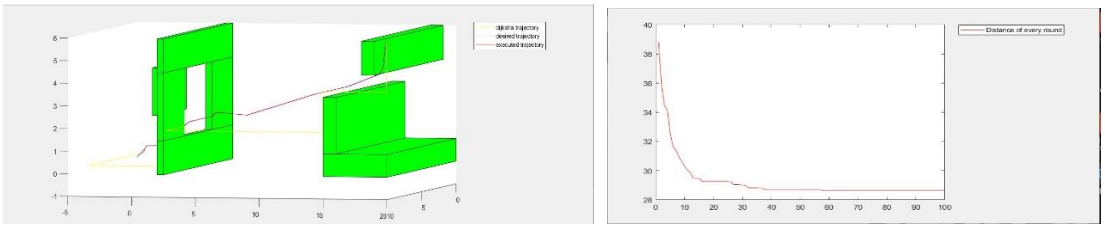
map0 (easy): theoretical shortest distance:39.74 our method: 39.77



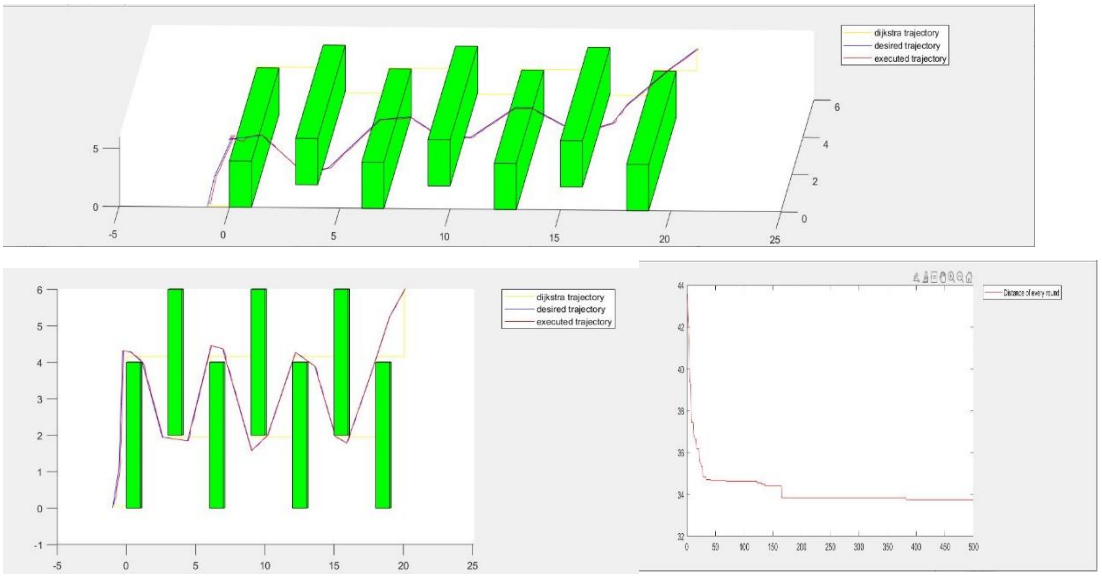
map1 (normal): theoretical shortest distance:11.05 our method: 11.24



map2 (normal)：theoretical shortest distance:27.59 our method: 28.44



map3(hard): theoretical shortest distance:29.90 our method: 33.74



During out test of the trajectory generator and path tracking, we tuned the Kds and Kps to optimize our controller to minimize the error in tracking. Initially, the tracking was not what we expected because there was huge overshoot in x and y directions. We realized that the basic PD control rule could help us improve result. To make overshoot smaller, we decreased out kps and increased our kds. Meanwhile, decreasing kp would increase the steady-state error. After testing with different sets of kps and kds, we found an optimal set of parameters for our controller.

Effects of *increasing* a parameter independently[18]

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability[14]
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

Figure 3. Effects of increasing a parameters in PID controller[3]

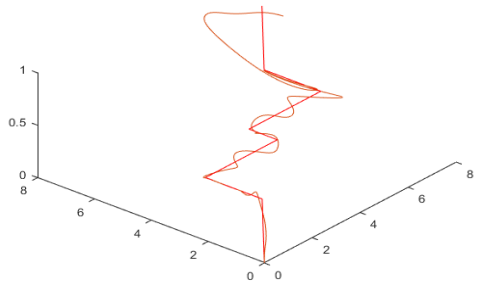


Figure 4.a Controller before tunning PD control

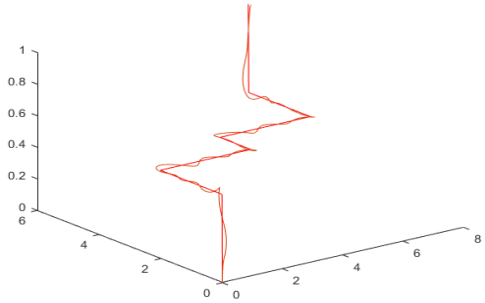


Figure 4.b Controller after tunning PD control

Discussion

As we can see, the slopes of the curves of distance change slower and slower, which is as expected. The final trajectories are quite close to the obstacle at some points, this is also expected since theoretically, the shortest path in 3D world should contain vertexes on obstacle. With more iteration rounds on CEM method the trajectory will be shorter, but closer to obstacles which makes the trajectory generated by minimum jerk polynomial segments methods *less* likely to be collision free.

For the 3rd test data (map3), the total distance of our method stops around 33.74, the theoretical shortest distance is about 29.90 This is caused by the twisted property of the trajectory, points on the trajectory cannot be too closed to the obstacles.

Based on our research[3], we found that trajectory generator is plays a significant role in planning and control since a efficient trajectory could minimize the control effort. As we increase the order of polynomials of motion and its derivative, the trajectory would be smoother and smoother. In other words, 7th order polynomial and minimum 4th order derivative would make the trajectory smoother than the 5th order polynomial and minimum 3rd order derivative, which is smoother than 3rd order polynomial and minimum 1st derivative.

We could also improve the trajectory generator by applying minimum snap polynomial segments methods by increasing the polynomial to 7th. However, for this project specifically, the minimum jerk method is adequate to generate a rather smooth trajectory.

Reference

[1] Trajectory generation and control for precise aggressive maneuvers with quadrotors

[2] MEAM 620: Robotics https://alliance.seas.upenn.edu/~meam620/wiki/index.php

[3] StackExchange

https://robotics.stackexchange.com/questions/167/what-are-good-strategies-for-tuning-pid-loops