**Project Report**

Jiahe Xu ([jxu109@jhu.edu](mailto:jxu109@jhu.edu)) Ray Zhang ([zzhan190@jhu.edu](mailto: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.

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 *zW* by the *yaw* angle ψ, then rotate about the intermediate *x*-axis by the *roll* angle *φ*, finally rotate about the *yB* axis by the *pitch* angle θ. The rotation matrix for transforming coordinates from *B* to *W* is given by:

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 *-zW* direction, and forces come from rotors *Fi*, in the *zB* direction. The equation involves the acceleration of the center of mass are:

Then, we have control input .

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

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

Each rotor has an angular speed i and produces a force *Fi*according to:

The moment produced by rotors is:

The angular acceleration determined by the Euler equations is:

we can rewrite it as:

where

Then, we can have our second control input *u2*:

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

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.

3.1 our method

find an initial path using dijkstra algorithm

read the map, build the environment with small cubes

apply minimum jerk polynomial method producing smooth trajectory

use CEM method finding a better path

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 consist 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 ----equation---.

Position:

Velocity:

Acceleration:

Jerk:

Snap:

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 at time . Then it will represent segment k+1 starting at 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.

The spline will go through n-1 segments which will results in 6\*(n-1) constrains and 6\*(n-1) unknowns. After solving the matrix equation, we substitute the coefficients back to ---equations above---- 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, position controller, is ---euqation----.

Substituting back to ----equation--- will conclude the position controller and calculate u1.

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

The angle controller will take the desired acceleration directly from position controller and use the following equations to calculate the desired roll, pitch, and yaw.

And the desired angle can be presented with the following equation

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

Diagram

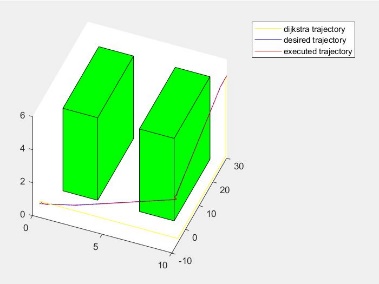
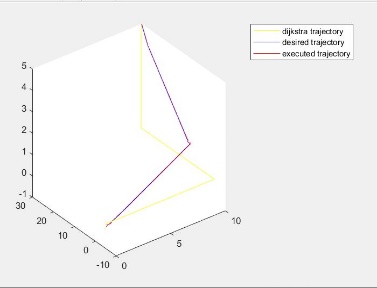
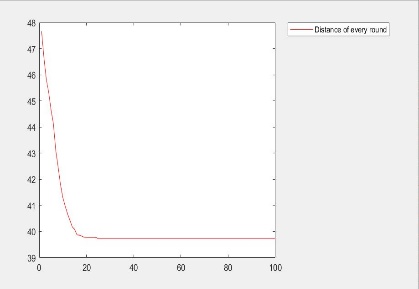
Description automatically generated

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

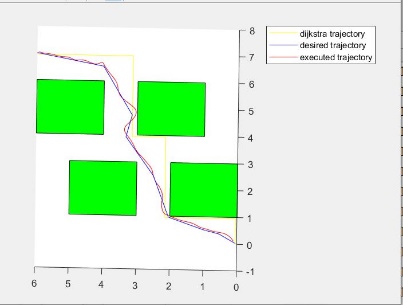
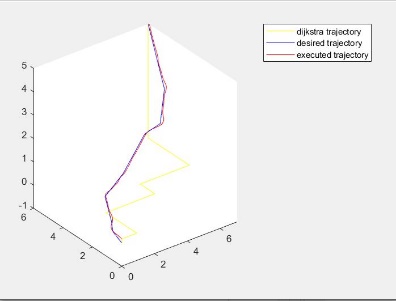
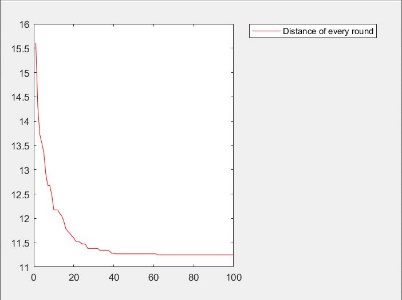
5 Experiment & Result

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.

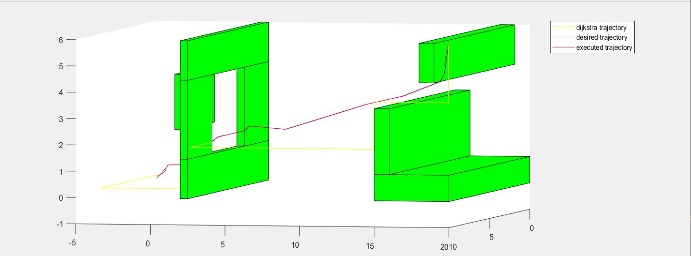
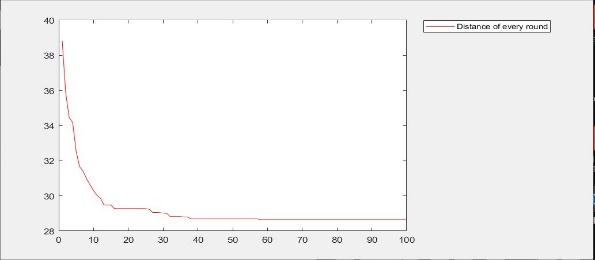
map0 (easy): theoretical shortest distance:39.74 our method: 39.77

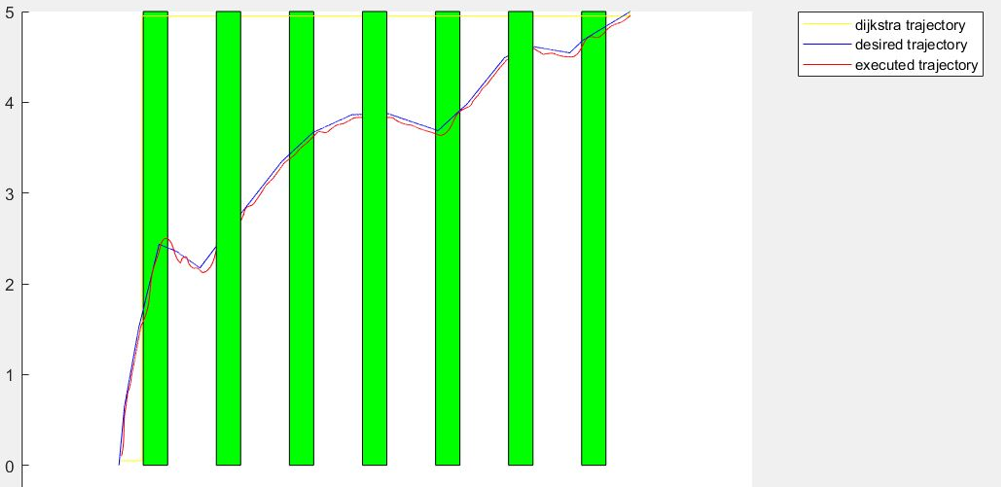
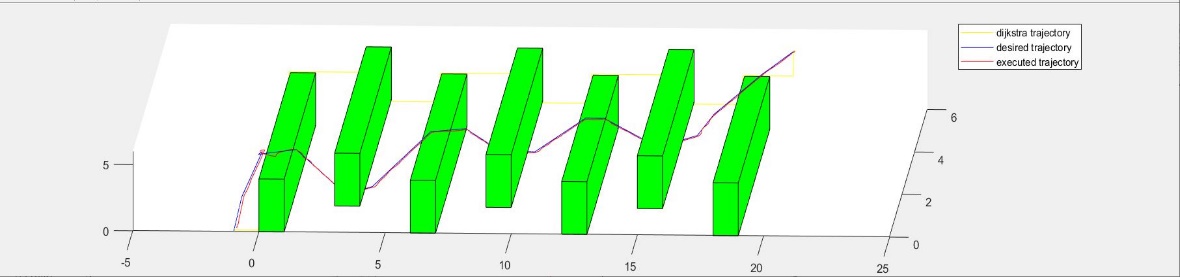
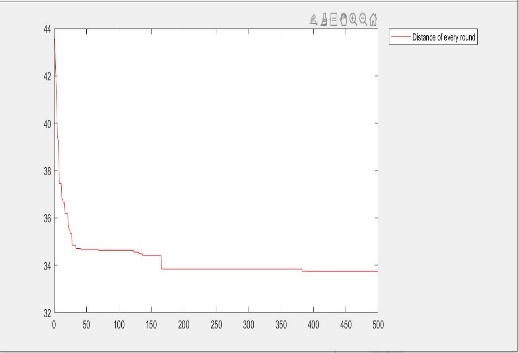
map1 (normal): theoretical shortest distance:11.05 our method: 11.64

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

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

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 (same things is vertical horizon decomposition algorithms).

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. The desired trajectory is a ‘zig-zag’ shape, which is a little counterintuitive, but actually it is correct, after rolling out the trajectory along Z-axis it should be a straight line, and after rolling back, it will be a zig-zag shape like what we have seen.

Reference

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