UNIVERSITY OF CALIFORNIA
SANTA CRUZ

TRAJECTORY PLANNING BASED ON OPTIMIZED JUMP POINT SEARCH RESULTS USING ARTIFICIAL POTENTIAL FIELD IN 3-D ENVIRONMENTS

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Shengjie Zhang

March 2021

The Thesis of Shengjie Zhang
is approved:

Professor Alex Pang, Chair

Professor Roberto Manduchi

Professor Gabriel Elkaim

Quentin Williams
Interim Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

TRAJECTORY PLANNING BASED ON OPTIMIZED JUMP POINT SEARCH RESULTS USING ARTIFICIAL POTENTIAL FIELD IN 3-D ENVIRONMENTS

Shengjie Zhang

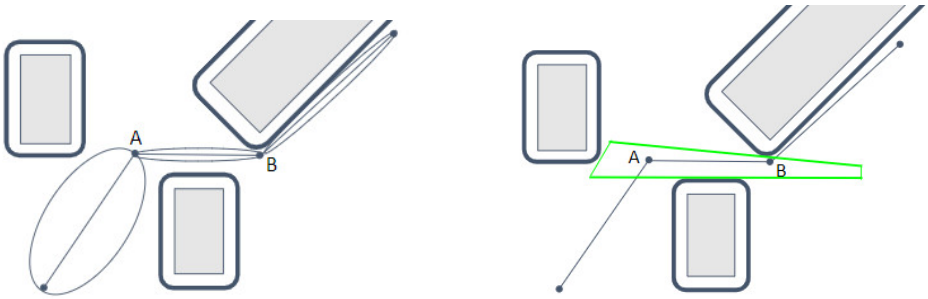
Abstract

Many of the motion planning algorithms require decomposing free space into convex regions in order to derive piece-wise polynomial trajectories. Using a path found by a fast graph search technique as a piece-wise linear skeleton to obtain convex regions is an effective method. However, this type of method suffers from narrow polyhedrons in that waypoints are in close proximity to obstacles, and in turn, narrow polyhedrons as constraints of a quadratic program (QP) influence the optimal solution of trajectory generating. This thesis proposes a method of enlarging polyhedrons through utilizing Artificial Potential Field to optimize the path found by Jump Point Search. Owing to enlarged convex polyhedrons, simulation results show that in some obstacle-cluttered map, trajectories generated by QP using enlarged polyhedrons as constraints become smoother, and the value of minimum snap cost function is smaller compared with that without optimizing.

1 Introduction

Unmanned aerial vehicles (UAVs) have received increasing interest. Navigation of a Micro Aerial Vehicle (MAV) in an obstacle-cluttered environment is a challenging problem since a MAV needs to generate a collision-free trajectory from an initial state to a final state through unknown cluttered environments. It has been shown that the trajectory generation problem can be formulated as a Quadratic Programming (QP)[4] with constraints of robots' positions. In order to avoid collision, a collection of convex connected polyhedrons can be treated as constraints in the QP. Many existing algorithms[5][22][19] for generating the collision-free convex region take a long time and require a proper selection of seeds.

To solve these problems, Safe Flight Corridors(SFC)[11] developed a novel convex decomposition method using ellipsoids computed from safe paths found by Jump Point Search(JPS)[7]. JPS is a highly fast path finding algorithm, which only expands on certain nodes in a grid map. With no intermediate nodes between certain nodes expanded, it can speed up A* by an order of magnitude and with small memory overheads in 3D grid map. However, as a result of pruning symmetrical paths, most of the turning points in the path found by JPS are close to obstacles. Consequently, several ellipsoids and corresponding polyhedrons constructed from path segments are quite narrow(Figure 1), which in turn influences the optimal solution of QP.

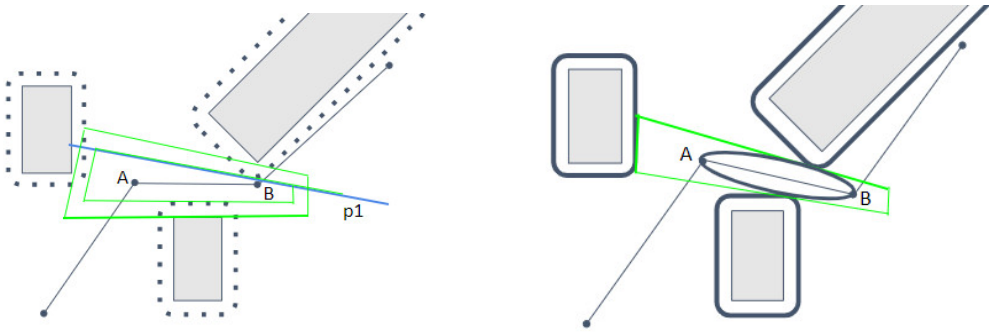


(a) Narrow ellipsoids (b) Narrow polygons

Figure 1: The problem of narrow ellipsoids and polyhedrons.

In Figure 1, gray regions are obstacles; The black bold lines outside the gray region are expanded contour of obstacles. In Figure 1(a), Waypoint B is in close proximity to obstacles causing the narrow ellipsoid with AB as major axis. In Figure 1(b), the narrow polygon is derived from a narrow ellipsoid.

To solve this problem, SFC[11] turns to use the original map instead of the expanded map, then adjusts the tilt angle of the corresponding hyperplane to derive a safe convex region(Figure 2(a)). In Figure 2(a), the dotted line represents the expanded contour of obstacles. First, SFC[11] uses original boundaries of obstacles to achieve polyhedrons(the outer contour of the green polygon); Second, SFC[11] shrinks polyhedrons to ensure they are collision-free(the inner contour of the green polygon). Note that some waypoints might be outside of polyhedrons after shrinking, to solve this, hyperplane P1 might be needed to adjust its angle.



(a) Solution in SFC[11] (b) Solution using APF

Figure 2: Abridged general view of solutions.

In this thesis, I adopt Artificial Potential Field[10] to optimize the results of JPS to solve this problem(Figure 2(b)). Artificial potential field(APF) is an obstacle avoidance scheme proposed by O.Khatib and applied on manipulator control. In Dolgov[6], they describe a practical path-planning algorithm based on A*[8] and the basic concept of APF for an autonomous vehicle operating in an unknown environment. In Zhou[23], a kinodynamic path searching originated from the hybrid-state A* search is applied for real-time quadrotor autonomous navigation.

However, although A* is completeness, it is still a challenge to apply it on 3D environments due to its limitation on computation time. JPS solves this problem by effective pruning. Inspired by[23][6], I apply APF to adjust waypoints position to enlarge ellipsoid(Figure 2(b)). In Figure 2(b), the position of Point A and Point B are adjusted using APF to enlarge polyhedrons.

Furthermore, for safety, we don't want MAVs to "hug walls"; for minimum snap, we desire to generate a trajectory that is as smooth as possible. Regarding these, optimizing a path using APF to enlarge ellipsoids is necessary. In Method Section, how to utilize JPS in a 3D grid map and to optimize a path are introduced in detail. Following path planning, finding polygon and generation trajectory based on the optimized path are described. In the Result and Discussion Section, results are shown and compared with that in SFC[11].

2 Related Work

Quadrotor motion planning covers a wide range of research in map generation, path planning, trajectory planning and trajectory following, etc. In terms of path planning, methods can be divided into two main categories: Randomized and Deterministic. With regards to randomized planners based on sampling, there are PRM[2], RRT[17], FMT[9], BIT*[15] etc. Compared with deterministic methods, randomized methods reduce searching time but they cannot guarantee to find a desired path ultimately even if a solution exists. For deterministic methods, there are Dijkstra based on breadth-first-search and querying partial solutions sorted by distance from a start point; A*[8] based on the best-first search since it adds a Heuristic function; D*[20] stands for "Dynamic A* search" and it is more efficient than A* in dynamic and complex environments with local changing; Jump Point Search(JPS)[7] performs symmetry breaking to speed up path-finding. JPS+[3] is a derivative of JPS, and its aim of searching efficiently is achieved by redesigning

a whole map with a symmetry-reduced form. Despite efficiency of JPS+, it is unsuitable for real-time motion planning of a MAV which needs to plan trajectories within a finite footprint in a local map in that JPS+ requires processing a whole map in advance.

A major problem with path planning so far is the trade-off between proximity to obstacles and path length. Take JPS used in Safe Flight Corridors(SFC)[11] for instance, a weakness of path planning is that it will choose the minimal-length path that is collision free, however, it causes a MAV to fly at the minimal collision-free distance to obstacles. A common way of penalizing proximity between turning points and obstacle is to use a potential field[6].

Trajectory planning involves how to follow a path given constraints such as position, velocity, and acceleration. According to Minimum Snap Trajectory Generation method[4], we can generate a piece-wise polynomial trajectory within multiple convex polyhedrons and turn the trajectory problem into a Quadratic Programming(QP)[16] problem. Therefore, finding polyhedrons and generating trajectories play critical roles in trajectory planning.

Concerning finding polyhedrons, Iterative Regional Inflation by semi-definite programming algorithm(IRIS)[5] allows users to select a start point in space on a terrain map and finds a maximum-volume ellipsoid inside a polyhedron combined by hyperplanes. Both the hyperplanes and the ellipsoid are refined over several iterations until the ellipsoid ceased to grow. The main disadvantage of IRIS are time-consuming due to iterations, and it makes no guarantee of finding the largest possible ellipsoid in the environment. SFC[11] uses two waypoints as major axes of an ellipsoid and the convex space consists of hyperplanes that can be computed directly through ellipsoids without iteration. On account of no iteration, it is much faster for SFC to solve the maximum ellipsoid compared with IRIS. Stereographic Projection[19] is another method that generates obstacle-free convex regions used for motion planning. The base stone of this algorithm is spherical projection and

uses convex hull generation and inverse vertex enumeration as its subprocedures. A major problem with Stereographic Projection is that the selection of initial points influences the volume of polyhedron found in solution, although this method relies on simple geometric methods. Instead of stereographic projection, Sphere Flipping[22] uses sphere flipping to transform original points to a nonlinear space, which flips interior points out. After flipping, they calculate the convex hull of these wrapped points and map vertices of the hull back to the original cartesian space inversely. In real applications, in order to enlarge the generated convex polyhedron, Sphere Flipping[22] uses initial points given by kinodynamic A*[23] to keep initial points as far as possible from obstacles.

For generating trajectory, n-th order polynomial and Bezier curves[1] have been widely used for expressing the curve of trajectory. In Minimum Snap[4] and SFC[11], the whole trajectory is composed of several polynomials and each polynomial is inside the corresponding polyhedron, thus minimum snap trajectories can be formed as a quadratic programming(QP) problem[16]. A major problem with QP is that it only guarantees two endpoints of each path segment are inside a polyhedron, accordingly some portion of a trajectory might be outside polyhedron. A common way to solve this problem is collision check, an additional specific time constrain is added halfway between two ends and the position at this time must also satisfy the constraints of polyhedrons. Following, the QP is re-solved with additional constraints and this process is repeated if necessary until the whole polynomial trajectory is collision-free, namely it is inside polyhedrons. Instead of resolving QP, in Close Form[18], the polynomial is re-optimized using an added point that splits the path segment into two. In contrast, the collision problem can be avoided by using Bezier curves due to its properties[1] that Bezier curves never pass through the intermediate control points and lie completely within the convex hull defined by the control points, but the value of minimum snap of trajectories derived from Bezier curve may not be optimal.

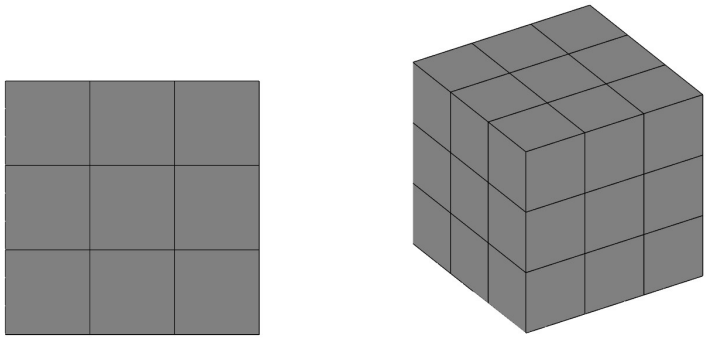
3 Methods

This section mainly discusses three parts for controlling the MAV to reach the goal: path planning, finding polyhedron and trajectory generating. The source code for these three parts can be found in https://github.com/LenaShengzhen/AerialRobotics.

3.1 Path Planning

3.1.1 JPS

In [7], details of JPS are provided in 2-D grid maps with uniform grids. This section concentrates to illustrate the differences between JPS on a 2D and a 3D map. To extend the 2D JPS proposed in [7] to 3D, the number of neighbors of a node is increased from 8 to 26(Figure 3); Definitions and illustration in 3D of natural neighbor[7] and forced neighbor[7] are also given below.



(a) In a 2D map, a center grid has eight neighbors

(b) In a 3D map, a center grid has twenty-six neighbors

Figure 3: The number of neighbors of a node is increased from 8 in 2D map (Figure 3(a)) to 26 neighbors (Figure 3(b)).

I use following definitions from JPS[7], which I include here for the convenience of explanation:

Definition 1 *A node n ∈ natural neighbour(x) if:*

1. *Assume there is no obstacles in neighbour(x)*

2. *For straight move,*

$$len(\langle p(x), \cdots, n\rangle \setminus x) > len(\langle p(x), x, n\rangle) \tag{1}$$

For diagonal moves,

$$len(\langle p(x), \cdots, n\rangle \setminus x) \geq len(\langle p(x), x, n\rangle) \tag{2}$$

In definition 1, x is a center node, $p(x)$ is the predecessor and neighbor node of x, or $p(x)$ called parent node of x. $\langle p(x), \cdots, n\rangle$ means any path from $p(x)$ to n, $\langle p(x), \cdots, n\rangle \setminus x$ indicates x does not appear on the path from $p(x)$ to n.

Definition 2 *A node $n \in$ forced neighbour(x) if:*

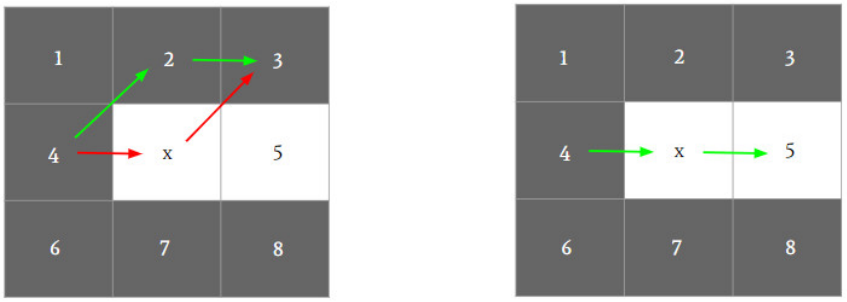
1. *n is not a natural neighbour of x and obstacles exist in neighbour of x.*

2.

$$len(\langle p(x), x, n\rangle) < len(\langle p(x), \cdots, n\rangle \setminus x) \tag{3}$$

Figure 4 explains the definition of Natural Neighbor. No obstacle exists in Figure 4 and Figure 5. In figure 4(a), x is the center node and node x is reached from its parent node No.4. For a path from No.4 to No.3, the length of path No.4 to No.2 to No.3(the green path) equals the path from No.4 to x to No.3(the red path), namely, it is not necessary for the shortest path from No.4 to No.3 to pass x. Thus, No.3 node is marked gray as a general neighbor. In Figure 4(b), the shortest path from No.4 node to No.5 node is No.4 to x to No.5, note the shortest path from No.4 to No.5 definitely goes through x. Thus, according to Equation 1, No.5 node is marked white as a natural neighbor of node x. In Figure 5 white nodes are natural neighbors of center node x. Gray nodes are general neighbors of center node x.

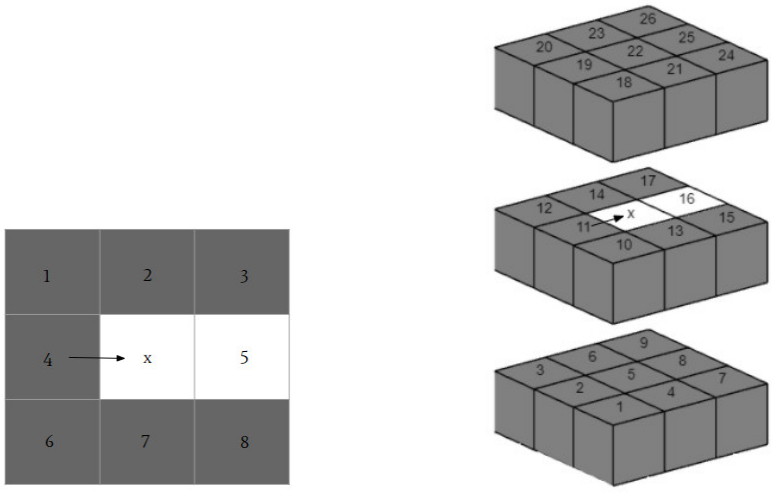
Similarly, in a 3D map, the No.16 node is a natural neighbor of center node x in that from No.11 node to No.16 node, no other shortest path could be found without passing through x. This figure only shows natural neighbors in the case of straight movement; Natural neighbors in the case of a diagonal movement can be found in the same way using Equation 2.



(a) Two shortest paths from No.4 to No.3 node.

(b) Only one shortest path from No.4 to No.5 node

Figure 4: General Neighbor of No.3 node and Natural Neighbor of No.5 node.



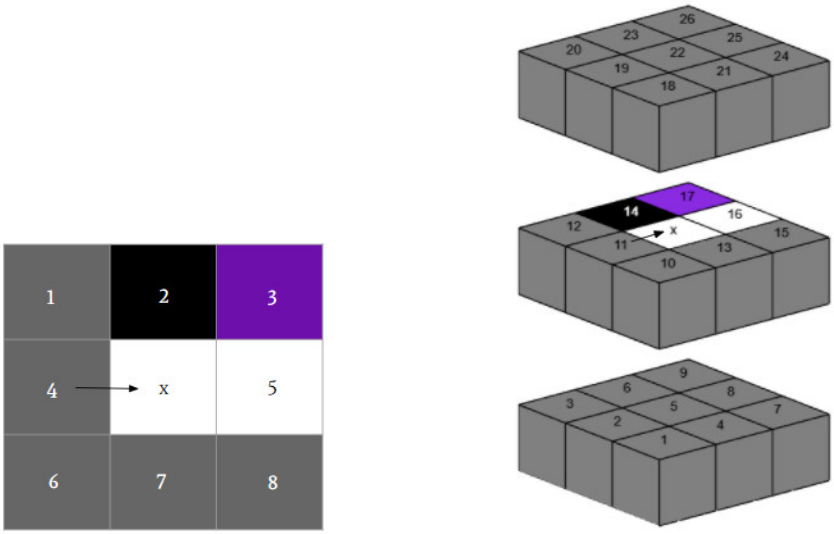
(a) White node No.5 is a natural neighbor of node x in a 2D map.

(b) White node No.16 is a natural neighbor of node x in a 3D map.

Figure 5: Natural Neighbor in a 2D and a 3D map.

Note that gaps are added between each floor grids in Figure 5(b), in order to

number each grid to explain conveniently. There should be no gaps between floors of grids in Figure 5(b) and Figure 6(b), in other words, assume Figure 5(b) and Figure 6(b) is as same as Figure 3(b).



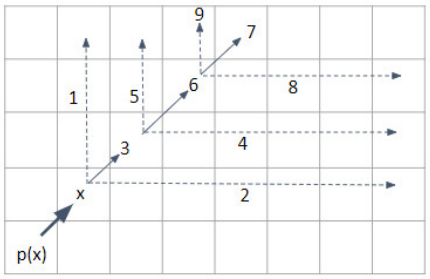
(a) Purple node No.3 is a forced neighbor of node x in a 2D map.

(b) Purple node No.17 is a forced neighbor of node x in a 3D map.

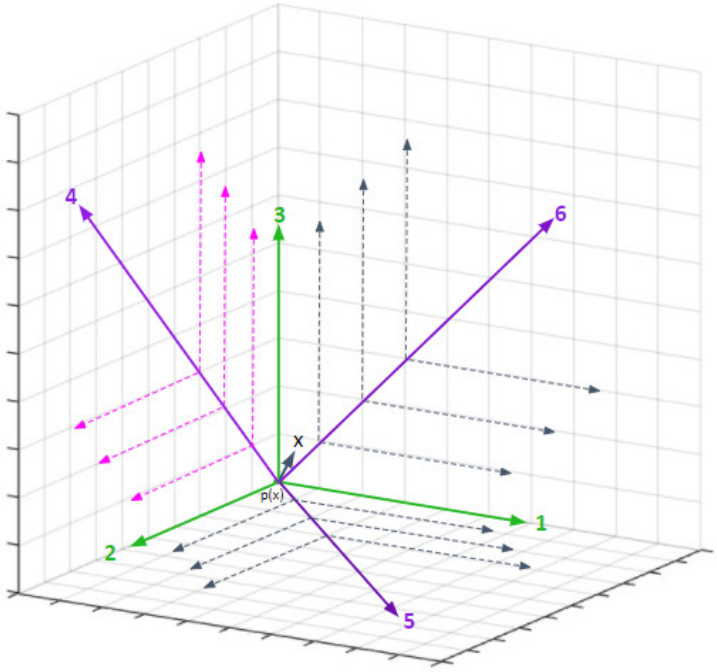
Figure 6: Forced neighbor in a 2D and a 3D map.

In Figure 6, black nodes indicates obstacles. The white No.5 node in a 2D map and the white No.16 node in a 3D map are still natural neighbors of the center node x. Gray nodes are general neighbors of center node x. For figure 6(a), in a 2D map, the No.4 node is the parent node of the node x, and the node x is reached from its parent node. The No.2 node becomes an obstacle. The shortest path from the No.4 node to the No.5 node is still No.4 to x to No.5, note that there is no change of this shortest path, although No.2 node becomes an obstacle. However, for the path from the No.4 node to the No.3 node, the shortest path of No.4 to No.2 to No.3 no longer exists due to the No.2 node becoming an obstacle. Thus, the path from No.4 to x to No.3 becomes the only shortest path from No.4 to No.3, which is definitely pass through the node x. In this situation, according to Definition 2, the No.3 node transforms from a general neighbor to a forced neighbor of node x, compared with

that in Figure 5(a). Similarly, in a 3D map, No.14 node is an obstacle. The No.16 node is still a natural neighbor of the center node x, while the No.17 node becomes a forced neighbor. When x is expanded, all nodes marked grey can be pruned.



(a) Diagonal direction search in a 2D map.



(b) Diagonal direction search in a 3D map.

Figure 7: Diagonal direction search in a 2D and a 3D map.

In addition to the number of neighbors changing from a 2D map to a 3D map, more searching directions also should be considered for diagonal searching strategy in a 3D map.

In Figure 7, a node x is reached from its parent node p(x) and the direction from p(x) to x is diagonal. In Figure 7(a), digits represent the index of the direction.

Digits 3, 6, 7 also mean the number of grids where they are located in. Here we start at the node x and need to find a jump point[7] successor of x considering this diagonal searching direction. Firstly, we travel vertically(direction 1) and horizontally(direction 2) to find whether there is a node which has a forced neighbor along with these two directions; if the result is no, we move to the next diagonal node along direction 3 and begin to travel along direction 4 and direction 5 to find a jump point successor of x. Iterating these steps until the target is found or reaching the map boundaries. For example, if a node along direction 8 has a forced neighbor, namely this node is a jump point, thus this node is a jump point successor of the No.6 node, and this in turn identifies the No.6 node as a jump point successor of the node x.

In Figure 7(b), to make the picture clear, the intersection of the gray lines represents a grid. In order to find the successor of node x, the direction of p(x) to x must be expanded to six directions: three directions(Green arrows marked 1,2,3) are parallel to the coordinate axis, the other three directions(Purple arrows marked 4,5,6) are exactly the diagonal directions on the coordinate plane. Note that when searching along these three diagonal directions, the principle is the same as diagonal searching in 2D map: searching vertically and horizontally(dashed arrow directions) firstly, and then search diagonally. For example, searching along direction 4 must be expanded to directions indicated by the light purple dotted arrows.

3.1.2 Simplify Path

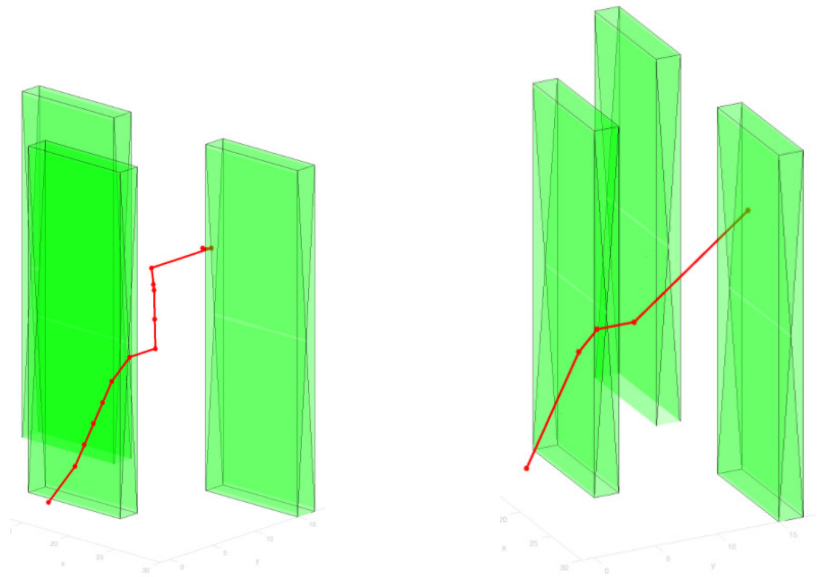
Searching directions is limited in neighbor grids(only 26 directions, Figure 3(b)) in a 3D grid map, however in practice, quadrotors can fly in any direction. Therefore, it is essential to delete redundant waypoints. How to prune redundant waypoints is described in Algorithm 1. In Algorithm 1, the index of waypoints is from 1 to n; The start point index is 1; The index of the endpoint is n. SimplePath is a variable

which records indexes of waypoints after pruning.

Algorithm 1 Simplify path

Require: path:waypoints

```
1: s = 1
2: simplePath.add(s)
3: for i from 2 to n - 1 do
4:     if line connected Point s and Point i does not collide with obstacles AND
       line connecte Point s and Point i+1 collide with obstacles then
5:         simplePath.push(i)
6:         s = i
7:     end if
8: end for
9: simplePath.add(n)
10: return simplePath
```



(a) A path without simplifying.

(b) The path with simplifying.

Figure 8: Simplifying a path

An example of simplifying a path is shown in Figure 8. In Figure 8, green cubes denote obstacles; Red lines indicate paths; red dots are waypoints. After simplifying, the number of waypoints in Figure 8(b) is less than that in Figure 8(a).

3.1.3 Path Optimization

As mentioned before, the minimal collision-free distance to obstacles might cause a narrow polyhedron, further affecting the generation of trajectories. In this thesis, Artificial Potential Field[10] is adopted to solve this problem, whose basic idea is to construct a function whose value is inversely proportional to the distance between robots and obstacles. The repulsive potential function[10] are as follows:

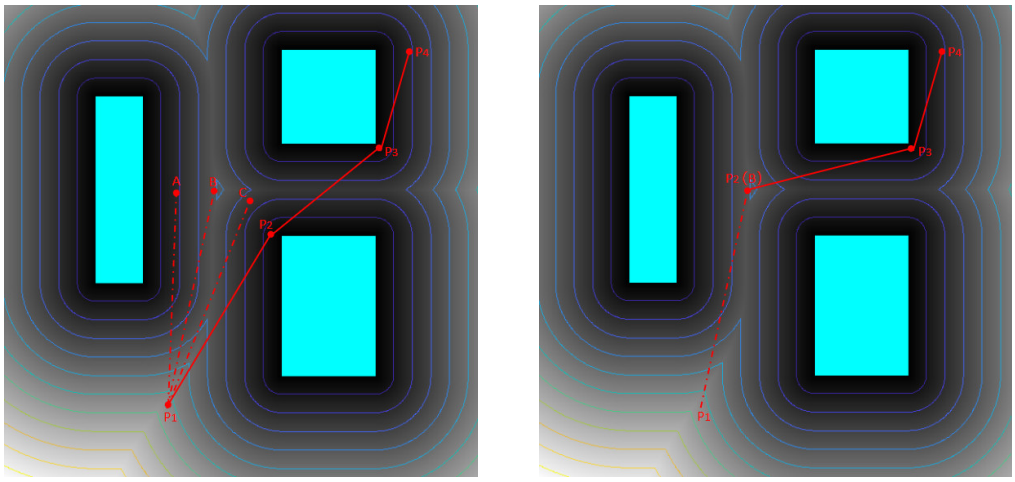
$$U_{rep,j} = \begin{cases} \frac{1}{2}\eta_j(\frac{1}{d_j(x)} - \frac{1}{Q_j^*})^2 & d_j(x) \leq Q_j^* \\ 0 & d_j(x) > Q_j^* \end{cases} \tag{4}$$

Here η_j is a constant scaling parameter of obstacle j; x is the current position of robot; $d_j(x)$ is the distance between obstacle j and the robot; Q_j^* is the limit distance of the potential field influence.

A downside of APF is that the attractive and repulsive forces might conspire to produce local minimum at locations other than a single global and desired location. Owing to this reason, I choose to utilize APF to optimize the results of JPS, instead of considering APF as another goal function during path planning using JPS.

In Figure 9, cyan regions are obstacles, and assume repulsion of grids inside cyan regions is infinite. Note that the closer the pixel to the obstacle, the darker the color, the greater the repulsion, conversely, the pixels farther away from the obstacle, the lighter the color, the smaller the repulsive force. Contour lines of the same color represent the same value of repulsive force. In Figure 9(a), red lines denote path $P = \langle P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rangle$, P_1 is the start point, P_4 is the target point, P_2 and P_3 are waypoints. Here we start from P_2, seaching neighbors of P_2 and choose a neighbor point whose repulsion is less than P_2. Assume we reach point C, check whether Line P_1C is collision-free, the maximum repulsion in Line P_1C is less than that in Line P_1P_2 and the maximum repulsion in Line CP_3 is less than that in Line P_2P_3. If these three requirements are met, continue this process

and start search from C's neighbor; if not, stop iteration.



(a) Finding a better waypoint to replace P_2.

(b) Replace P_2 with B.

Figure 9: Optimizing path using Artificial Potential Field.

Algorithm 2 Upate a waypoint

Require: P_i : current point, P_{i-1} : previous point, P_{i+1} : next point

```
pointlist = [P_i]
while P_i is not a obstacle AND P_i's repulsion > the smallest repulsion among P_i's neighbors do
    pointlist.add(the neighbor with the smallest repulsion)
    P_i = the neighbor with the smallest repulsion
end while
minRepulsion1 = MAX;
minRepulsion2 = MAX;
updatePoint = P_i
for each point P_x in pointlist do
    if the maximum repulsion along with line P_{i-1}P_i ≤ minRepulsion1 AND the maximum repulsion along with line P_iP_{i+1} ≤ minRepulsion2 then
        minRepulsion1 = the maximum repulsion along with line P_{i-1}P_i
        minRepulsion2 = the maximum repulsion along with line P_iP_{i+1}
        updatePoint = P_x
    end if
end for
return updatePoint
```

For example, if the maximum repulsion along Line P_1A is greater than that in Line P_1B and Line P_1B is collision-free, iteration need to stop at position B and update

P_2 to B(Figure 9(b)). Following, repeat this process to update next waypoint P_3. Algorithm 2 describes how to update each waypoint in a path.

3.2 Finding Polyhedron

I adopt a similar method of constructing Safe Flight Corridors(SFC)[11] to find Polyhedron. In [11], constructing SFC is divided into two main steps: (1) Find Ellipsoids, (2)Find hyperplanes that constructs polyhedron. Since in [11], the author uses 2D illustration to explain the concept and process in detail, to avoid repetition, I focus on describing the solution process with 3D illustration mainly from the perspective of formulas. The path P(Figure 10) is denoted as $P = \langle p_0 \rightarrow p_1 \rightarrow ...p_n \rangle$. The i^{th} line segement is represented as $L_i = \langle P_i \rightarrow P_{i+1} \rangle$. The purpose is to generate a convex polyhedron around each line segment in Path P.

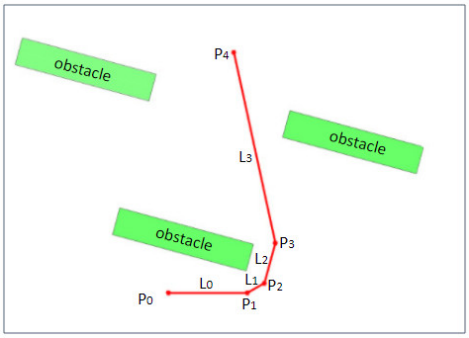
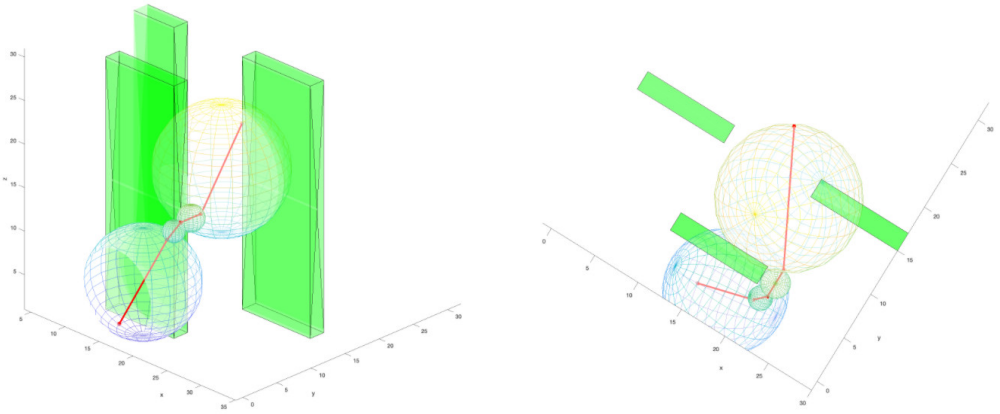


Figure 10: path segments denotation

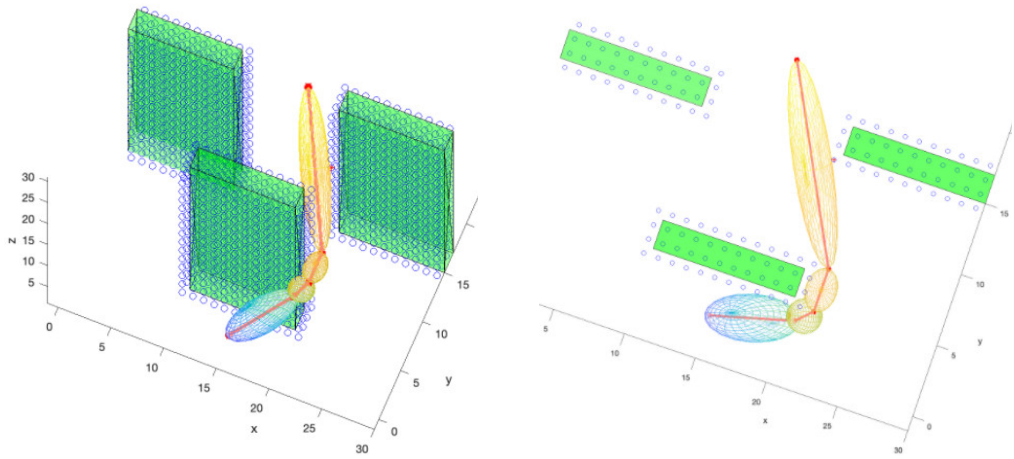
3.2.1 Finding Ellipsoid

The purpose of this step is to find an ellipsoid that includes the path segment L and contain no obstacle points. Details can be found in [11], for this step, here I only give 3D illustrations. We start from a sphere centered at the middle point of L and assume the length of ellipsoids' x-axis are fixed(Figure 11), reducing the length of the other two axes until the spheroid contains no obstacles(Figure 12). In Figure 12, hollow blue dots denote obstacle points.



(a) Initial sphere from one side view. (b) Initial sphere from a top view.

Figure 11: Observing initial sphere on each path from different angles.



(a) Initial sphere from one side view. (b) Initial sphere from a top view.

Figure 12: Observing ellipsoid contain no obstacles from different angles.

3.2.2 Finding Hyperplane

Denote the ellipsoid found in the previous step as ε, which touches an obstacle point at P^*(e.g. in Figure12, the red dot indicates P^*). The process of finding mathematical expression of ε is as follows:

The standard equation of ellipsoid in matrix representation is:

$$x^T \Lambda^{-1} x = 1 \tag{5}$$

in Equation 5, $\Lambda = \begin{bmatrix} r_1^2 & & & \\ & r_2^2 & & \\ & & \ddots & \\ & & & r_n^2 \end{bmatrix}$ and $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, here x_i indicates the i^{th} axis, r_i is half the length of the principal axes. Now, rotate this ellipsoid and let the principal semi-axes of the ellipsoid aligned with path L_i, assume the rotation matrix is A, we have:

$$(A^T x)^T \Lambda^{-1} (A^T x) = x^T (A \Lambda^{-1} A^T) x = 1 \tag{6}$$

Then move the center of this ellipsoid from the origin to the center of path L_i denoted by x_c , we have:

$$f(x) = (x - x_c)^T (A \Lambda^{-1} A^T)(x - x_c) = 1 \tag{7}$$

Equation 7 is the expression of ε, also, a normal vector of a point on the surface of this ellipsoid is:

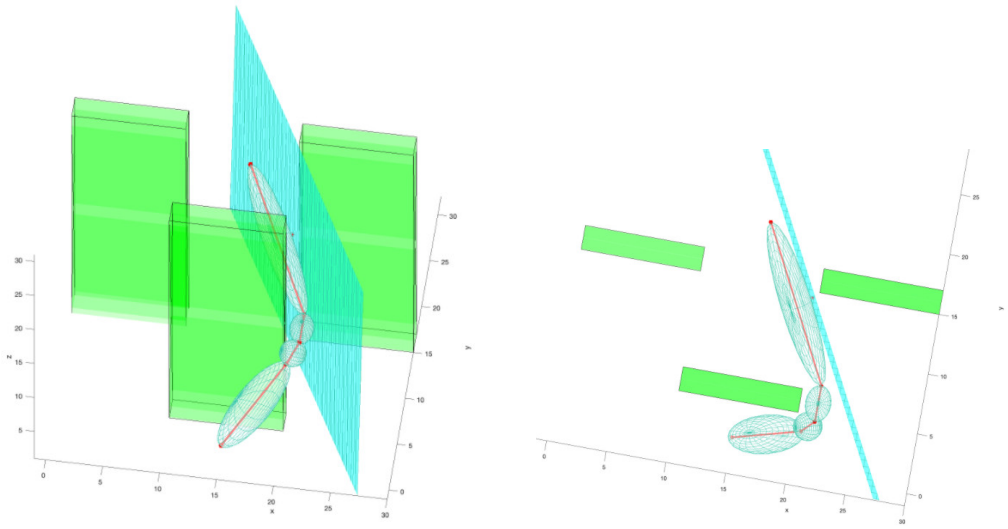
$$\nabla f = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\rangle \tag{8}$$

Assume $P^* = (x_0, y_0, z_0)$, thus the equation of the tangent hyperplane H_0(e.g. Figure 13) through P^* is as follows:

$$\nabla f \cdot (x - x_0, y - y_0, z - z_0) = 0 \tag{9}$$

Remove all the obstacles points that lie outside this tangent hyperplane H_0 (Equation 9). From the remaining obstacle points, find another point P_1^* which is nearest to the center of ε. P_1^* and the normal vector(Equation 8) create a new hyperplane H_1. This process is continued to obtain a sequence of hyperplane, $H_0, H_1, \cdots, H_m$ until these hyperplanes construct a convex polyhedron C without any obstacle points inside for L_i. Applying this method on individual line segment L_i of the path P to construct a collection of convex overlapping polyhedra that

models free space.



(a) (b)

Figure 13: Observing hyperplane from different angles.

3.3 Generating Trajectory

In this section, QP[4] of generating Minimal Snap trajectories is introduced. Given a trajectory $x(t)$, $x(t)$ denotes the position of a robot at time t, snap is the fourth derivative of $x(t)$ with respect to time, with the first, second, and third derivatives being velocity, acceleration, and jerk, respectively. In this thesis, Minimal Snap is chosen as the cost function of the trajectory.

3.3.1 Cost Function

A single trajectory segment between two waypoints can be represented as an n-order polynomial:

$$x(t) = b_0 + b_1 t + b_2 t^2 + \cdots b_n t^n \tag{10}$$

Write Equation (10) as vector multiplication form, we have:

$$x(t) = [1, t, t^2, \cdots t^n] \cdot c \tag{11}$$

In expression (11), $c = [b_0, b_1, \cdots, b_n]^T$, c is a vector of unknown parameters to be solved; t stands for time. For a trajectory with k + 1 waypoints(including the start point and the end point), namely k trajectory segments, can be composed as piece-wise polynomials[14]:

$$x(t) = \begin{cases} \left[1, t, t^2, ..., t^n\right] \cdot c_1 & t_0 \leq t < t_1 \\ \left[1, t, t^2, ..., t^n\right] \cdot c_2 & t_1 \leq t < t_2 \\ \vdots & \\ \left[1, t, t^2, ..., t^n\right] \cdot c_k & t_{k-1} \leq t < t_k \end{cases} \tag{12}$$

Here, $c_i = [c_{i_0}, c_{i_1}, ..., c_{i_n}]^T$ is the vector of coefficients of polynomial for i^{th} trajectory segment.

The cost function penalizing the squares of the derivatives of $x(t)$ can be written as Equation13, thereby, it is a Quadratic Programming(QP) problem.

$$x^*(t) = \underset{x(t)}{argmin} \int_0^T (x^{(4)})^2 dt = min \sum_{i=1}^{k} \int_{t_{i-1}}^{t_i} (x^{(4)})^2 dt = min \sum_{i=1}^{k} c_i^T Q_i c_i \tag{13}$$

Here, $Q_i = \int_{t_{i-1}}^{t_i} [0, 0, 0, 0, 24, \cdots, \frac{n!}{(n-4!)} t^{n-4}]^T [0, 0, 0, 0, 24, \cdots, \frac{n!}{(n-4!)} t^{n-4}] dt$.

According to the Euler-Lagrange Equation, a necessary condition for the optimal solution of Equation 13 is:

$$x^{(8)} = 0 \tag{14}$$

Detailed derivation about Equation 14 was provided in Flash and Hogan[21]. Thus, the minimum-snap trajectory is a 7^{th} order polynomial, namely $n = 7$ in Equation 10.

3.3.2 Constraints

For the start point and the end point in the trajectory, both of them have specific positions, velocity and acceleration constraints. For example, at the start point,

assume the position is p_0, velocity is v_0 and acceleration is a_0, we have:

Position constraint:

$$\begin{bmatrix} 1, t_0, t_0^2, ..., t_0^n, \underbrace{0...0}_{(k-1)(n+1)} \end{bmatrix} C = p_0 \tag{15}$$

Velocity constraint:

$$\begin{bmatrix} 0, 1, 2t_0, ..., nt_0^{n-1}, \underbrace{0...0}_{(k-1)(n+1)} \end{bmatrix} C = v_0 \tag{16}$$

Acceleration constraint:

$$\begin{bmatrix} 0, 0, 2, ..., n(n-1)t_0^{n-2}, \underbrace{0...0}_{(k-1)(n+1)} \end{bmatrix} C = a_0 \tag{17}$$

Here $C = [c_1, c_2, \cdots, c_k]^T$. Constraints are also imposed on the joint points of each trajectory segment. These constraints assign the same values of position, velocity, acceleration, jerk and snap, etc., namely, zero to sixth derivative of $x(t)$, to the joint point between two segments. Take the i^{th} segment and the $(i+1)^{th}$ segment for instance, assume $x_i(t)$ is the i^{th} segments and $x_{i+1}(t)$ denotes $(i+1)^{th}$ segment, constraints are as follows:

$$\frac{\mathrm{d}^k}{\mathrm{d}t^k} x_i(t) = \frac{\mathrm{d}^k}{\mathrm{d}t^k} x_{i+1}(t), \quad k = 0 \cdots 6 \tag{18}$$

e.g. at the joint point(the position of time t_i), if $k = 0$, the position constraint is as follow:

$$\begin{bmatrix} \underbrace{0, ..., 0}_{(i-1)(n+1)}, \underbrace{1, t_i, t_i^2, ..., t_i^n}_{i^{th} segment}, \underbrace{-1, -t_i, -t_i^2, ..., -t_i^n}_{(i+1)^{th} segment}, \underbrace{0...0}_{(k-i-1)(n+1)} \end{bmatrix} C = 0 \tag{19}$$

Hence, for the start point and the end point in the trajectory, there are eight constraints, if constraints are imposed on position, velocity, acceleration and jerk; For $k-1$ middle waypoints, according to Equation 18, there are $7(k-1)$ constraints. Totally, the number of constraints is $7(k-1)+8$.

Besides these constraints, polyhedrons composed of hyperplanes found in Section 3.2.2 are also constraints of the cost function(Equation 13). For the i^{th} polyhedron composed of m hyperplanes, the constraint is as follows:

$$A_i^T \begin{pmatrix} x_i(t) \\ y_i(t) \\ z_i(t) \end{pmatrix} < b_i \tag{20}$$

Here $A_i^T = \begin{pmatrix} \triangledown f_1, & \triangledown f_2, & \cdots, & \triangledown f_m \end{pmatrix}^T$, $\triangledown f_j$ is the normal vector of j^{th} hyperplane. $x_i(t)$, $y_i(t)$, $z_i(t)$ denote positions of x, y, z coordinates of the robot at time t; b_i stands for the matrix of multiplication of A_i^T and m points that create hyperplanes with the normal vector of an ellipsoid.

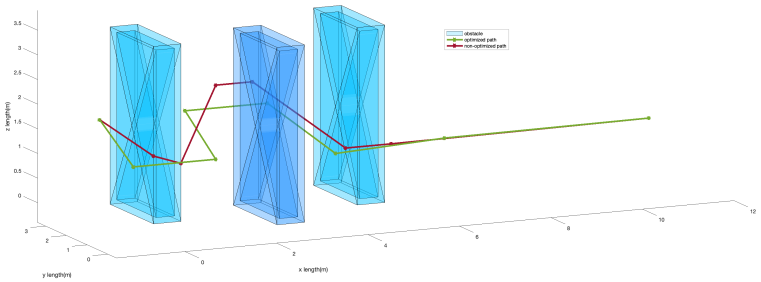
Now, cost function(Equation 13) formulate as a QP with constraints of Equation 18 and Equation 20.

4 Result and Discussion

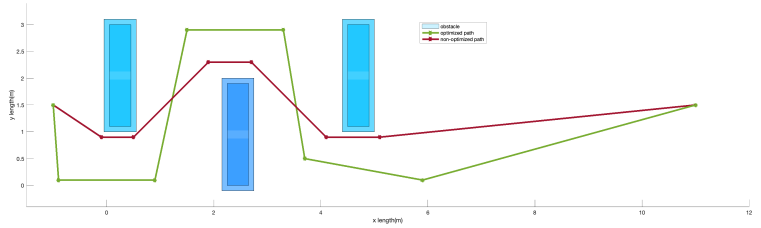
I implemented the proposed algorithm in MATLAB[13] using QUADPROG[12] to solve QP. The method of time allocation in simulation is trapezoid velocity proflile: given acceleration and maximum speed, the quadrotor accelerates from 0 to the maximum speed with a constant acceleration a, and then decelerates to 0 with the acceleration of $-a$. In this section, trajectories derived from optimized paths and non-optimized paths will be compared on different maps.

4.1 Comparison

Note that in order to avoid obstacles, the outer contour of obstacles is expanded. The expansion distance is exactly the diameter of a quadrotor, thereby, a quadrotor can be treated as a point in algorithms. In Figure14, blue cubes indicate obstacles; The red path is found by JPS; The green path is the optimized path using APF. In Figure15, it is obvious that the ellipsoid derived from the optimized path is larger than that from the original path. In Figure16, blue cubes indicate obstacles; The red path is found by JPS; The red dotted line indicates the trajectory derived from the red path with QP. In Figure17, the green path is an optimized path from the red path using APF; The green dotted line represents the trajectory derived from the green optimized path with QP. Data comparison between these two trajectories is shown in Table 1, in the situation that both speed and acceleration are zero at both the start point and the end point, and the flight time is equal, the value of snap of the trajectory derived from the optimized path is smaller than that derived from the non-optimized path.

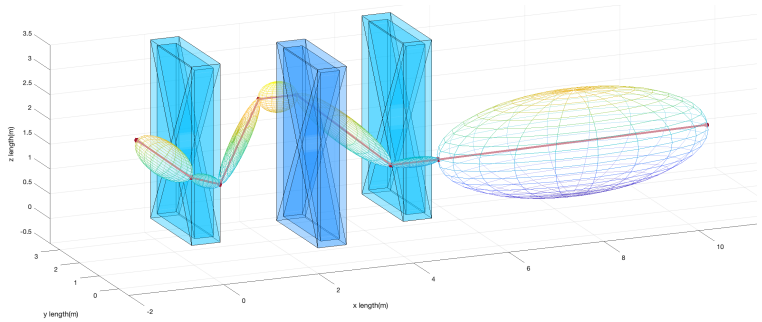


(a)

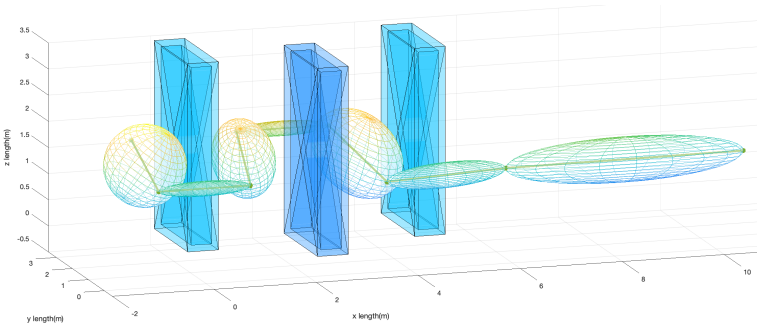


(b)

Figure 14: Observing paths from different angles.

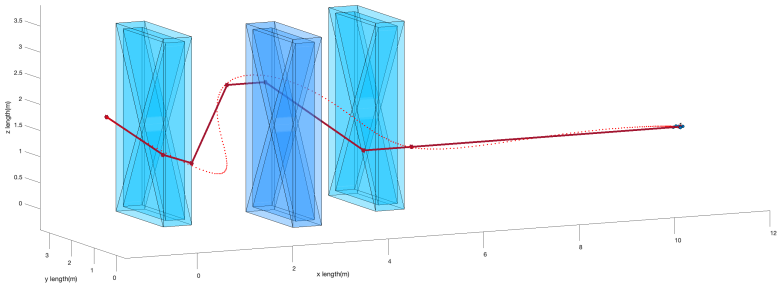


(a) Ellipsoids derived from the non-optimized path.

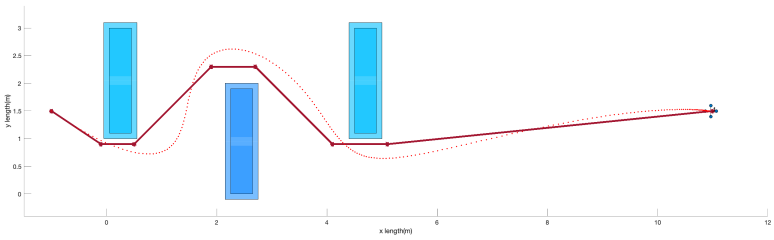


(b) Ellipsoids derived from the optimized path.

Figure 15: Initial ellipsoids derived from the path.

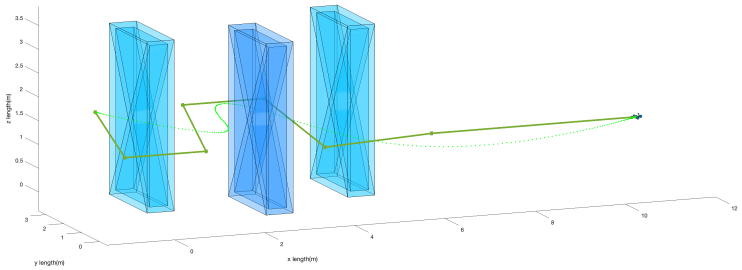


(a) Observing the trajectory from a side view.

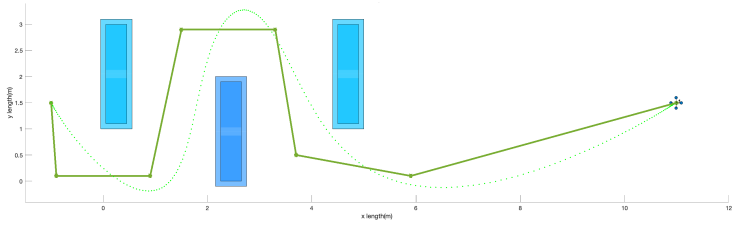


(b) Observing the trajectory from a top view.

Figure 16: Generating Trajectory using SFC.



(a) Observing the trajectory from a side view.

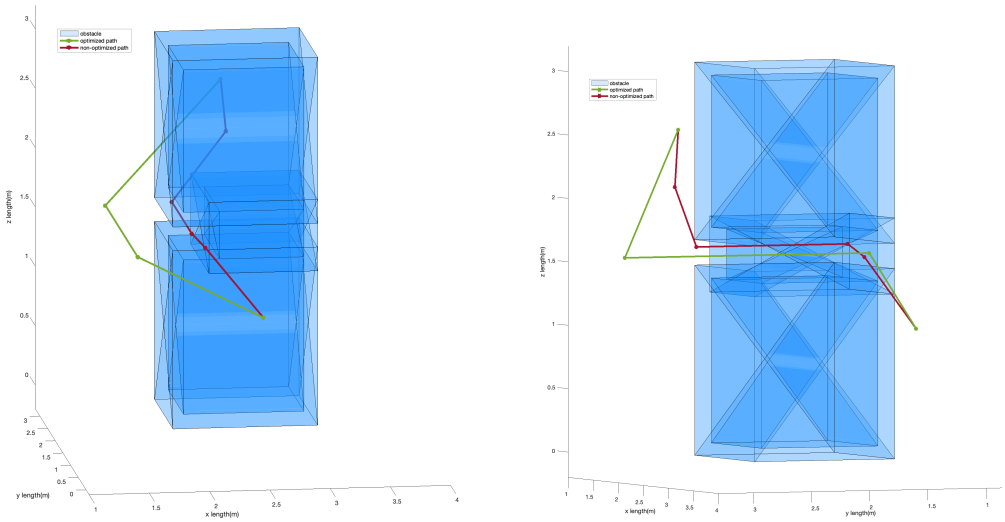


(b) Observing the trajectory from a top view.

Figure 17: Trajectory derived from the optimized path using APF.

Method	Average Speed	Time	Snap
JPS	1.10423	12.9343	342.26
JPS+APF	1.4	12.9343	99

Table 1: Data comparison between trajectories derived from different paths.



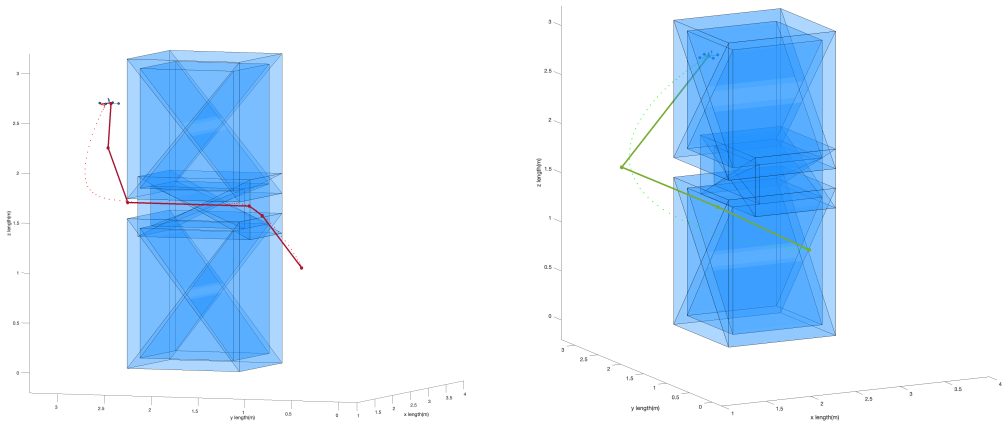
(a) (b)

Figure 18: Observing Non-optimized and optimized path in anther map from a different angle.

In Figure 18 and Figure19, paths and trajectories are compared in another map. In Figure 18, the red path is found by JPS; The green path is an optimized path based on the red path. Apparently, the red path passes through the gap between obstacles, and the optimized path bypasses the obstacles. Correspondingly, the trajectory generated from the red path also passes through the gap between the obstacles, and the trajectory generated from the green path bypasses the obstacles as well. Data comparison between these two trajectories is shown in Table 2. Since the red path has more turns than that of the green one, the value of snap of the red trajectory is much greater than that of the green one.

Method	Average Speed	Time	Snap
JPS	0.8	4.3681	6760.654
JPS+APF	1.0767	4.3681	361.1853

Table 2: Data comparison between trajectories derived from different paths.



(a) Trajectory derived from non-optimized path.

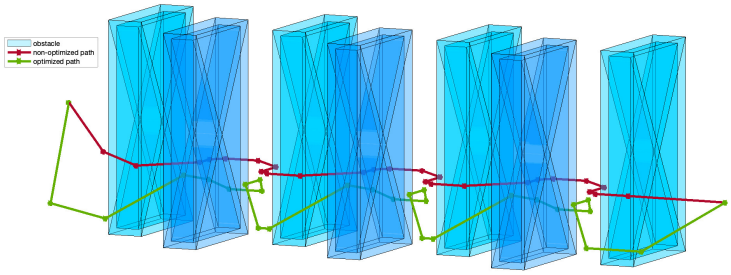
(b) Trajectory derived from optimized path using APF.

Figure 19: Trajectory derived from different paths in another map.

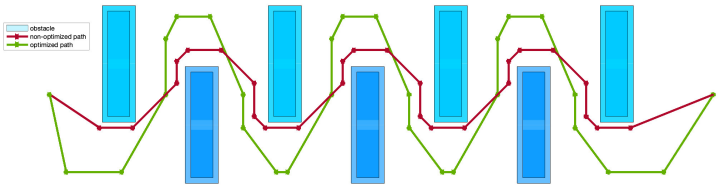
4.2 Discussion

According to the requirement of different tasks, we can choose different trajectories. Take the map in Figure19 for instance, if the task is to observe the detailed terrain

in the gap between obstacles, the red trajectory is a better choice than the green one. If the requirement is consuming as little energy as possible to reach a destination, the green trajectory is a better one compared with the red one.



(a) Observing paths from a side view.



(b) Observing paths from a top view.

Figure 20: Complex path for QP solving.

The application scenario of this thesis is navigation of the MAV in an unknown environment with local sensing in real time, thus, in order to follow a new trajectory once a robot finishes executing the current trajectory, the time for generating a trajectory must be guaranteed to be less than executing time of the current epoch. Therefore, the planning distance, which is also restricted by sensing range, is limited during each planning epoch. In this scenario, the distance range of the local map obtained from local sensing is restricted and the number of waypoints on the planned path is limited as well, from which QP could quickly generate a trajectory. However, if the robot has prior knowledge about the environment, namely, the planning distance becomes longer and more waypoints on the path, it is possible the time of solving QP is too long for a real-time control problem, or even it cannot obtain an optimal solution(e.g. Figure 20). In Figure 20, blue cuboids are obstacles; The red path is found by JPS; The green path is the optimized path using APF on

the results of JPS. In this situation, due to too many constraints, it is hard for QP to obtain an optimal solution.

5 Conclusion and Future Work

JPS is a fast grid map search technique due to effective pruning and it is suitable for real-time 3D map path planning. It is a challenge to do trajectory planning based on the result of JPS because of close proximity between waypoints and obstacles. This thesis presents a new method for optimizing the results of JPS utilizing artificial potential field. Results show that optimizing paths could enlarge convex regions, which makes the trajectory generated by QP smoother and with less value of snap.

There could be a lot of possible future work on this thesis. Time Allocation significantly affects the resulting trajectories. In addition to path segment length, considering the size of an angle between two path segments as a factor, which might improve the optimal solution of trajectory. Moreover, in terms of other finding polyhedron algorithm, e.g., Stereographic Projection[19] and Sphere Flipping[22], some methods are dependent on seed selection. Adopting Artificial Potential Field also provides a guideline to solve the problem of seed selection.

References

[1] A., L. Piecewise bezier curve trajectory generation and control for quadrotors. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 2016.

[2] Bohlin, R., and Kavraki, L. E. Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)* (2000), vol. 1, pp. 521–528 vol.1. doi:10.1109/ROBOT.2000.844107.

[3] Daniel Harabor, A. G. The jps pathfinding system. In *In Proceedings of the Fifth Annual Symposium on Combinatorial Search* (2012), AAAI, p. 207–208.

[4] Daniel Mellinger, V. K. Minimum snap trajectory generation and control for quadrotors. In *International Conference on Robotics*

and Automation (Shanghai, China, May 2011), IEEE, pp. 290–294. doi:10.1109/ICRA.2011.5980409.

[5] Deits, R., and Tedrake, R. Computing large convex regions of obstacle-free space through semidefinite programming. In *WAFR* (2014). doi:10.1007/978-3-319-16595-0_7.

[6] Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. Path planning for autonomous vehicles in unknown semi-structured environments. In *The International Journal of Robotics Research* (January 2010), vol. 29, pp. 485–501. doi:10.1177/0278364909359210.

[7] Harabor, D., and Grastien, A. Online graph pruning for pathfinding on grid maps. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence* (San Francisco, California, August 2011), AAAI'11, AAAI Press, p. 1114–1119.

[8] Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107. doi:10.1109/TSSC.1968.300136.

[9] Janson, L., Schmerling, E., Clark, A., and Pavone, M. Fast marching tree:a fast marching sampling-based method for optimal motion planning in many dimensions. *arXiv:1306.3532 [cs.RO]* (2013).

[10] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation* (1985), vol. 2, pp. 500–505. doi:10.1109/ROBOT.1985.1087247.

[11] Liu, S., Watterson, M., Mohta, K., Sun, K., Bhattacharya, S., Taylor, C. J., and Kumar, V. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. In *IEEE ROBOTICS AND AUTOMATION LETTERS* (July 2017), vol. 2. doi:10.1109/LRA.2017.2663526.

[12] MATLAB. *Matlab User's Guide*. The MathWorks Inc, 2019. Optimization toolbox:quadratic programming and cone programming https://www.mathworks.com/help/optim/ug/quadprog.html.

[13] MATLAB. *version 9.7.0.1216025(R2019b) Update 1*. The MathWorks Inc., Natick, Massachusetts, 2019.

[14] Mellinger, D. W. *Trajectory generation and control for quadrotors*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 2012.

[15] Menglu Lan, Shupeng Lai, Yingcai Bi, Hailong Qin, Jiaxin Li, Feng Lin, and Chen, B. M. Bit*-based path planning for micro aerial vehicles. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society* (2016), pp. 6079–6084. doi:10.1109/IECON.2016.7792953.

[16] Nieuwstadt, M. J. V., and Murray, R. M. Real time trajectory generation for differentially flat systems. In *Int'l Juornal of Robust and Nonlinear Control* (May 1997).

[17] Noreen, I., Khan, A., and Habib, Z. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. In *IJCSNS International Journal of Computer Science and Network Security* (October 2016), vol. 16.

[18] Richter, Charles, Bry, A., and Roy, N. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research. Ed. Masayuki Inaba and Peter Corke* (2016), vol. 114, Springer International Publishing, pp. 649–666.

[19] Savin, S. An algorithm for generating convex obstacle-free regions based on stereographic projection. In *2017 International Siberian Conference on Control and Communications (SIBCON)* (2017), vol. 4, IEEE, pp. 1–6. doi:10.1109/SIBCON.2017.7998590.

[20] Stentz, A. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation* (1994), pp. 3310–3317 vol.4. doi:10.1109/ROBOT.1994.351061.

[21] Tamar Flash, N. H. The coordination of arm movements: an experimentally confirmed mathematical model. In *Journal of Neuroscience* (July 1985), vol. 5, p. 1688–1703. doi:10.1523/JNEUROSCI.05-07-01688.1985.

[22] Zhong, X., Wu, Y., Wang, D., Wang, Q., Xu, C., and Gao, F. Generating large convex polytopes directly on point clouds. *arXiv:2010.08744v2* (2020).

[23] Zhou, B., Gao, F., Wang, L., Liu, C., and Shen, S. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters 4*, 4 (2019), 3529–3536. doi:10.1109/LRA.2019.2927938.