P2A: Tree Planning Through the Trees!

Dushyant Patil
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
dpatil1@wpi.edu

Keshubh Sharma
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
kssharma@wpi.edu

***Abstract*—This project presents an approach for path planning and trajectory tracking over a known map. We use RRT* algorithm to find obstacle avoiding path between start and goal. We use trajectory smoothing to generate a feasible path. We perform a PID controller to perform trajectory controller which maintains quadrotor dynamics**

I. Problem Statement

The aim of this project is to execute motion planning in a known forest. For this we will be using the RRT* algorithm to get a strictly linear path. The estimated path is then smoothened to allow for efficient navigation and smooth transition from one segment of RRT* path to other. This smoothed feasible trajectory is then tracked by the drone with the help of a PID controller.

II. Simulating world map in blender

The data provided to us was the textformat which consists of information about boundaries of 'world' and boundaries of obstacles. We load this data in blender as cube objects using $primitive_cube_add$ function of bpy module. The 'world' cube is made transparent with the help of mixed shader in blender. The map is displayed as shown below in blender:

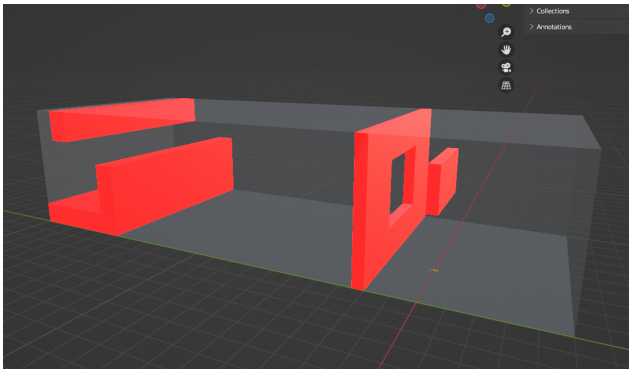

Fig. 1. World map with obstacles

III. Configuration Space

To maintain a safe distance between drone and obstacles during navigation, we inflate the size of obstacle by a certain factor. In our case with map1, we increased the size of obstacles by 10% (As shown in appendix). The image below shows obstacles inflated by 10% (the silhouettes surrounding blue and green cube in the image)

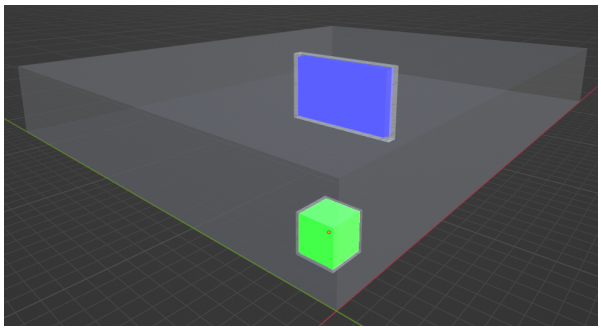

Fig. 2. Inflated obstacles to create configuration space

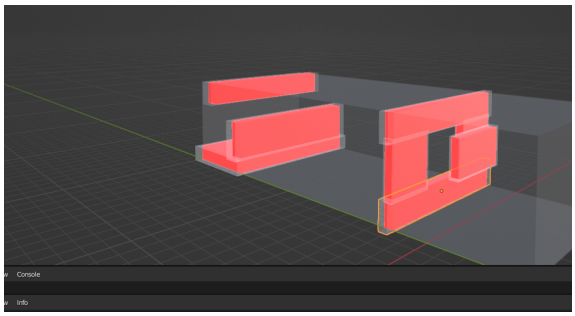

Fig. 3. Inflated obstacles to create configuration space

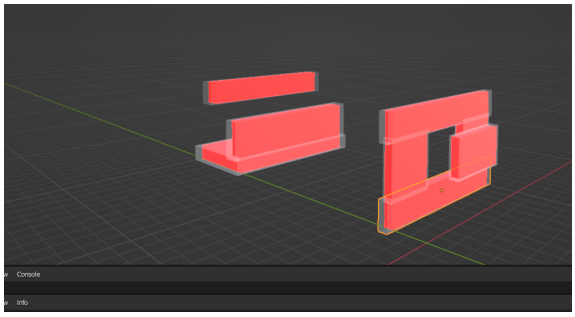

Fig. 4. Inflated obstacles to create configuration space

IV. RRT* Path finding

Rapidly-exploring Random Trees Star (RRT*) is a path planning algorithm used in robotics and AI. It iteratively builds a tree of potential paths from a starting point to a goal while minimizing the overall path cost. RRT* efficiently explores the configuration space, incrementally improving the generated paths by reconfiguring the tree structure and selecting low-cost paths. This algorithm is particularly useful for solving complex motion planning problems in dynamic environments.
The algorithm looks like follows:

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Fig. 5. RRT* Algorithm

For our application we sampled in euclidean space and used the L2 norm for cost calculation between sampled nodes. The nodes should be within 1.5 units of distance between each other and we decided to ignore all the obstacles that comes before the sampled points for optimization of collision checking. The collision checking algorithm utilizes intersection of line between the nodes with all of the 6 faces of the cuboidal obstacle defined which is faster than volumetric calculations.

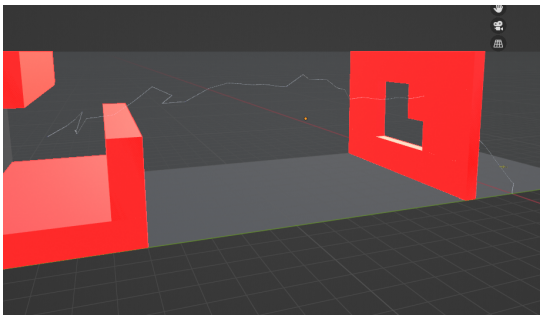

Fig. 6. RRT Path in blender

The path which we found for above mentioned start and goal positions are shown in figures 6, 7, 8.

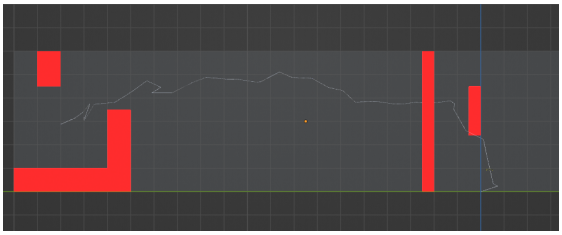

Fig. 7. RRT Path in blender

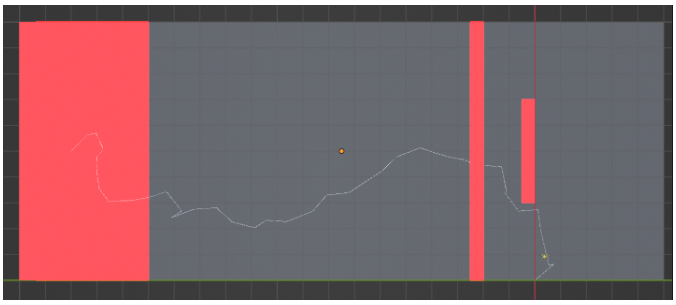

Fig. 8. RRT Path in blender

V. Trajectory Generation

The RRT path found as shown above is not a smooth trajectory and consists of a number of linear segments between waypoints which are the nodes in RRT path. This path does not allow a smooth transition from one segment to next if those 2 segments are not colinear (which they are usually not). This causes the drone to slow down before moving to next segments in the path which is very inefficient. To solve this problem, we use cubic trajectory generation. We assume the drone to follow cubic trajectory between any 2 waypoints which gives us a time parameterized trajectory as follows:

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{1}$$

Each segment of the RRT path will have their own coefficients a_0, a_1, a_2, a_3 for all 3 axes of motion. To find these coefficients we impose the initial condition the the velocity at the start and end is 0. We also impose the continuity condition during transition from one segment to next:

$$vel_i(t = t_{f,i}) = vel_{i+1}(t = t_{0,i+1}) \tag{2}$$

$$acc_i(t = t_{fi} = acc_{i+1}(t = t_{fi+1}) \tag{3}$$

Using this transition condition, we solve ffor the coefficients of all the cubic spline for all segments at once.
Let us assume that there are m segments (i.e. m+1 nodes in RRT path). Let A be a matrix of size 4mx4m. A is composed of 3 submatrices A_{pos} [2mx4m], $A_{transition}$ [2(m-1)x4m], A_{endvel}[2x4m].

$$A = \begin{bmatrix} A_{pos} \\ A_t \\ A_{endvel} \end{bmatrix} \tag{4}$$

We solve for below equation to find coefficients:

$$A * Coeffs = RHS \tag{5}$$

$$A_{pos} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & & & & \\ 1 & t_{f1} & t_{f1}^2 & t_{f1}^3 & \cdots & & & & \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & t_{f2} & t_{f2}^2 & t_{f2}^3 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & & & \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & & & \\ \cdots & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \cdots & 0 & 0 & 0 & 0 & 1 & t_{fm} & t_{fm}^2 & t_{fm}^3 \end{bmatrix}$$

$$A_t = \begin{bmatrix} 0 & 1 & 2t_{f1} & 3t_{f1}^2 & 0 & -1 & 0 & 0 & \cdots \\ 0 & 0 & 2 & 6t_{f1} & 0 & -2 & 0 & 0 \cdots & \\ \vdots & \vdots & \cdots & \vdots & \vdots & & & & \\ \vdots & \vdots & \cdots & \vdots & \vdots & & & & \\ \cdots & 0 & 1 & 2t_{fm} & 3t_{fm}^2 & 0 & -1 & 0 & 0 \\ \cdots & 0 & 0 & 2 & 6t_{f1} & 0 & -2 & 0 & 0 \end{bmatrix}$$

$$A_{endvel} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 2t_{fm} & 3t_{fm}^2 \end{bmatrix}$$

$$\text{Coeffs} = \begin{bmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \\ a_{3,1} \\ a_{0,2} \\ a_{1,2} \\ a_{2,2} \\ a_{3,2} \\ \vdots \\ \vdots \\ a_{0,m} \\ a_{1,m} \\ a_{2,m} \\ a_{3,m} \end{bmatrix}$$

$$\text{RHS} = \begin{bmatrix} x_0 \\ x_1 \\ x_1 \\ x_2 \\ \vdots \\ x_{m-1} \\ x_m \\ \vdots \\ 0 \\ 0 \\ \vdots \\ vel_{start} \\ vel_{end} \end{bmatrix}$$

Here t_{fi} is time assumed to travel i^{th} segment. We use a heuristic to allot time per segment. In out case, we find the total distance of travel and the time per segment was proportional to the segment length. Solving for equation 5 we get following smoothed trajectories (compared with RRT path).

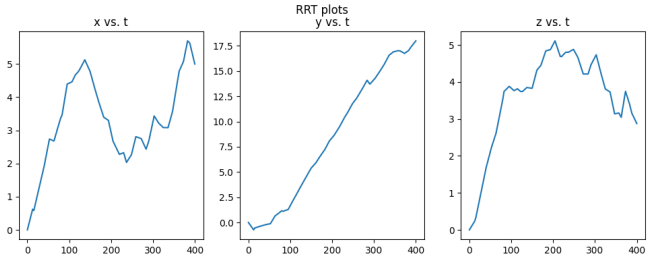

Fig. 9. RRT 2D Path

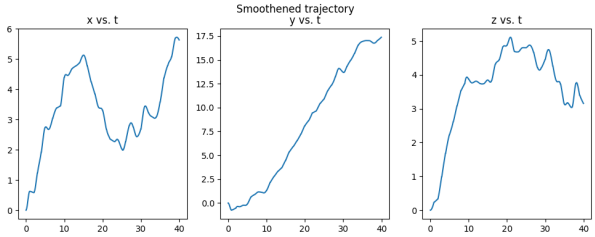

Fig. 10. Smoothened 2D Path

VI. Trajectory Following

Trajectory generated in above step is followed with the help of PID controller which was prewritten as part of the codebase. We had to tune the PID gains for inner loop of velocity control and the position control PID gains as well. We tried to tune the velocity controller first and then tried to tune the position controller gains. We first tried to tune the PID gains so that the drone could hover at one place. Then we tried to improve those gains so that the drone could move on a linear trajectory (initially with a constant 'z', then a linearly changing 'z'). After that we tried to move the drone on a trajectory with constant acceleration. In the end,we further tuned the gains on the sample helical trajectory provided to us as the part of the assignment. We used following gains for PID controller:

x_{pid} = (1, 0.0, 0.1)
y_{pid} = (1, 0, 0.1)
z_{pid} = (1.5, 0, 0.)
vx_{pid} = (2, 0, 0.0)
vy_{pid} = (2, 0, 0.0)
vz_{pid} = (2, 0, 0,)

We got the following results for few of the above mentioned cases:

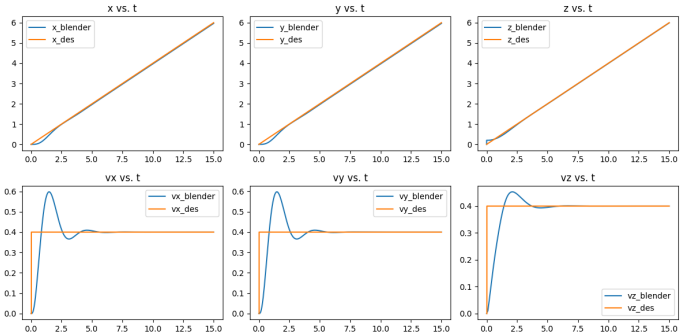

Fig. 11. Constant velocity trajectory

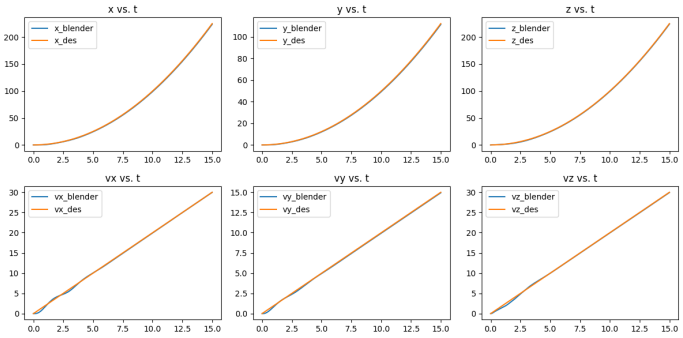

Fig. 12. Constant Acceleration

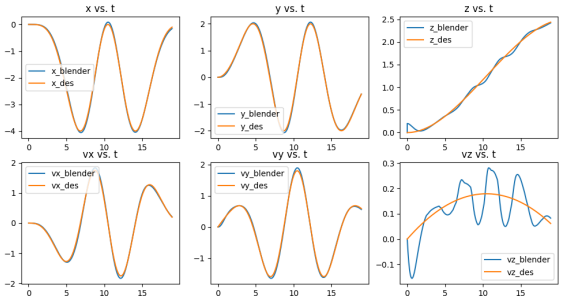

Fig. 13. Sample Trajectory

We then tried these gains with the smoothened trajectory generated using cubic spline equations. We found following results:

VII. Test Set Results

Below are 2D and 3D plots for performance of the controller on test sets. For test dataset 3, we selected start point (2,3,5) as the start point provided to us initiially was very close to first obstacle and if we inflate the obstacles, the inflated obstacle

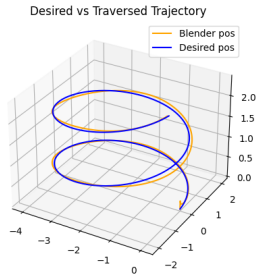

Fig. 14. Sample Trajectory

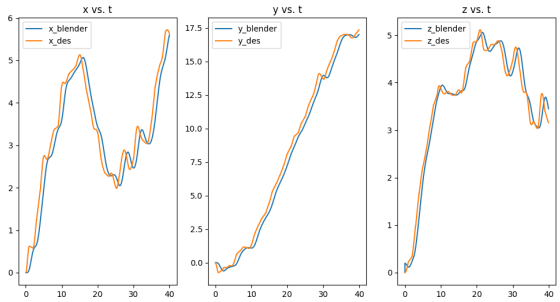

Fig. 15. Actual trajectory between goal and start

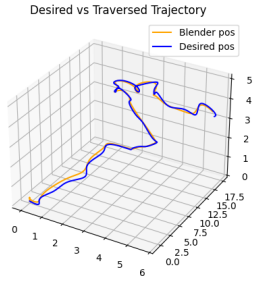

Fig. 16. 3D Desired vs Actual

and drone collide. Thus we select start point as [2,3,5] and end point as [19,3,5]

References

[1] A Quaternion-based Unscented Kalman Filter for Orientation Tracking
[2] Class Notes by Prof. Nitin Sanket

VIII. Appendix

A. Obstacle Inflation for Configuration Space

Get boundaries for all obstacles from map files provided For each obtsacle:
xmin, ymin, zmin, xmax, ymax, zmax, = boundaries of obstacle

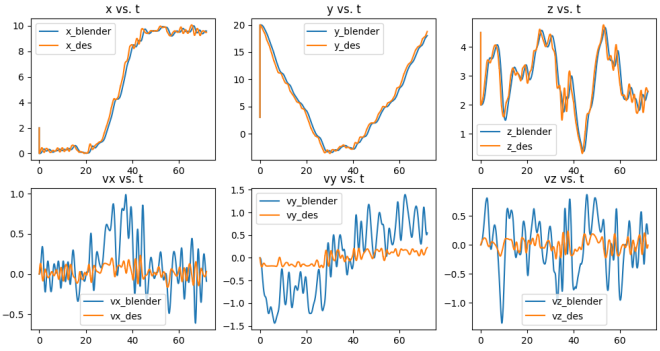

Fig. 17. Map2 Desired vs Actual

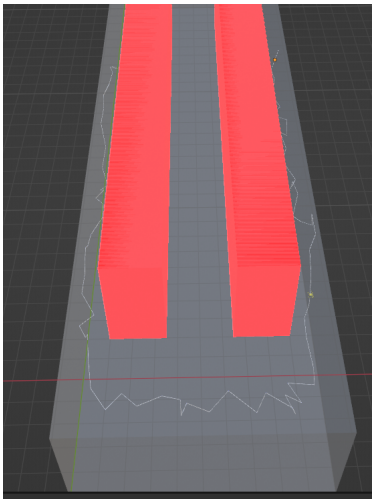

Fig. 21. Map2 Path

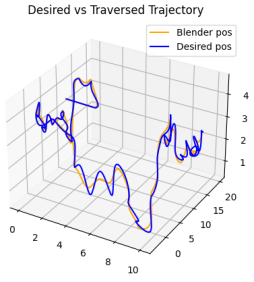

Fig. 18. 3D Map2 Desired vs Actual

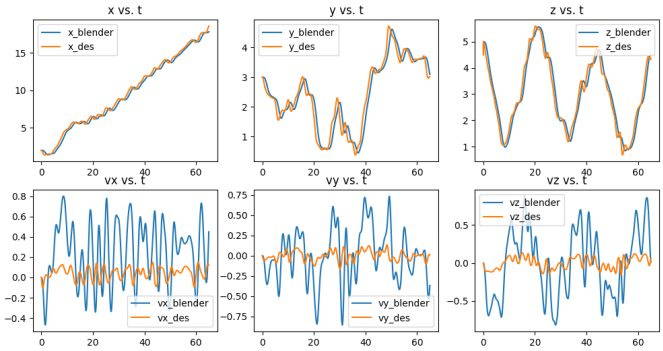

Fig. 22. Map3 Desired vs Actual

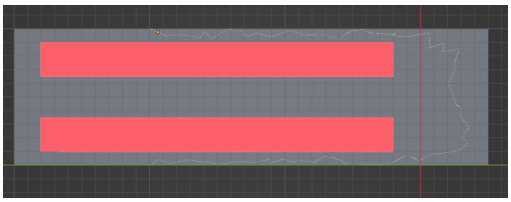

Fig. 19. Map2 RRT

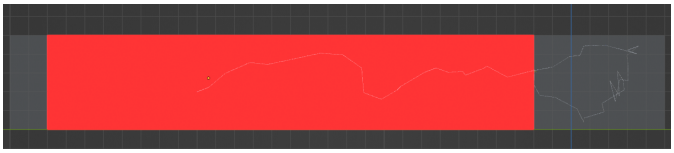

Fig. 20. Map2 RRT

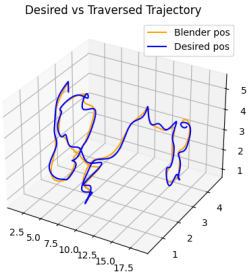

Fig. 23. Map3 3D Desired vs Actual

xlength = xmax - xmin
ylength = ymax-ymin
zlength = zmax - zmin
xmin = max($xmin_{world}$, (xmin - 0.1*xlength))
xmax = min($xmax_{world}$, (xmax + 0.1*xlength))
ymin = max($ymin_{world}$, (ymin - 0.1*ylength))
ymax = min($ymax_{world}$, (ymax + 0.1*ylength))
zmin = max($zmin_{world}$, (zmin - 0.1*zlength))
zmax = min($zmax_{world}$, (zmax + 0.1*zlength))

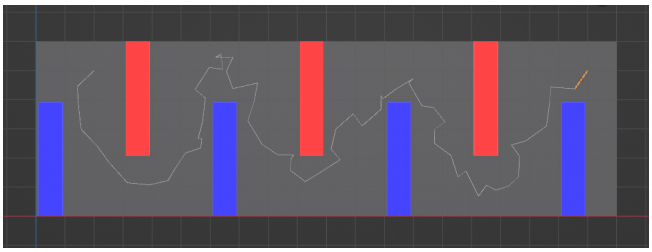

Fig. 24. Map3 RRT

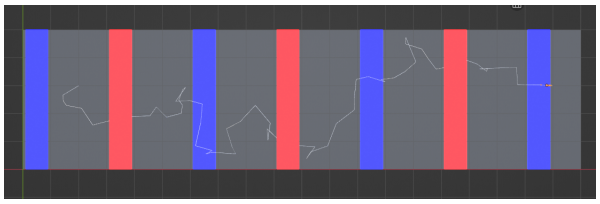

Fig. 25. Map3 RRT

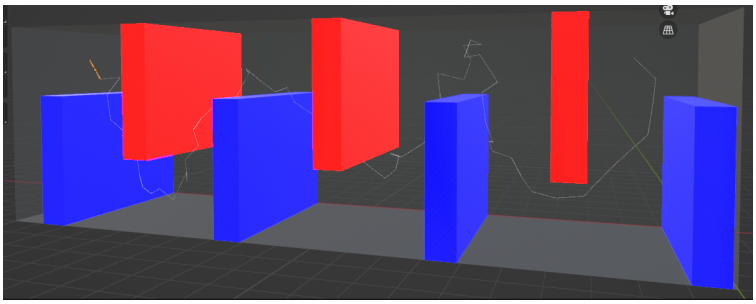

Fig. 26. Map3 Path