Path planning for Aerial Robots

Eashwar Sathyamurthy
Robotics Engineer
A. James Clark School of Engineering
University of Maryland
College Park, USA
eashwar@umd.edu

Akwasi A Obeng
Robotics Engineer
A. James Clark School of Engineering
University of Maryland
College Park, USA
obenga01@umd.edu

***Abstract*—In recent times, the increase in the usage of automobiles has caused difficulty in transporting goods between two places. As a result, the alternative use of employing aerial robots such as drones in transporting goods has gained a lot of popularity due to reduced traffic and its ability to reach goal faster. However, aerial robots face the disadvantage of navigating obstacles such as avoiding collision with tall buildings, changing path to avoid dynamic obstacles etc. Due to these difficulties, path planning of drones becomes crucial. Hence, there is a need to develop a path planning algorithm which finds the shortest path to reach the goal whilst avoiding static and dynamic obstacles. In this report, we implemented sampling based path planning algorithms to enable the drone maneuver its environment in both static and dynamic environments.**

I. Introduction

Drones in recent times are being used for surveillance in military and for transportation purposes. Recently, drones were used to transport kidneys. Hence, with the commercialization of drones, it is important to generate a path planning algorithm which can enable the drone to manuever its environment when faced with random obstacles.

Figure 1. Drone delivering kidney

This report provides detailed implementation of RRT and RRT* sampling based algorithms both in 2D and 3D and provides comparison. Furthermore, the algorithm has been developed to avoid dynamic obstacles. This report provides the simulation results of both static and dynamic obstacle avoidance in 3D.

II. Plan of Action

A. *Designing the environment*

The first step for visualing how well our code works is to generate a suitable environment that provides with it the ease of testing. The environment developed contains obstacles which should be avoided by the drone. The environment was designed both in 3D and 2D (projection of 3D). Initially, we assume the environment to be static. Dynamic obstacles-that is, other drones- are added later in 3D environment to test our implementation.The figure below shows the environment in 2D as well as in 3D.

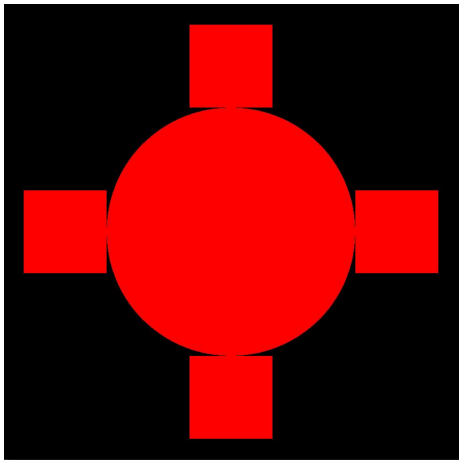

Figure 2. 2D environment created using pygame

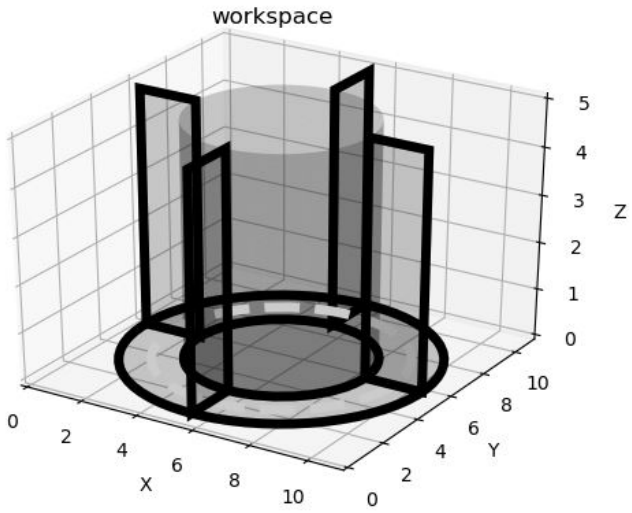

Figure 3. 3D environment created in matplotlib side view

Below is the equivalent environment created in ROS our implementation.

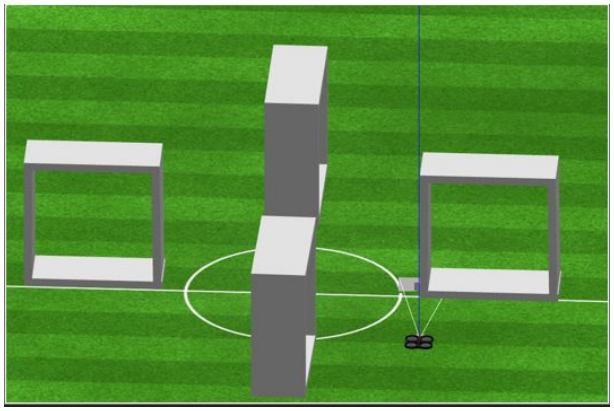

Figure 4. 3D environment created in matplotlib side view

III. Path planning method

The path planning method choosen for the drone mainly depends on the type of obstacles present in the environment. Below is a brief introduction to the types of obstacles as well as the different variants of RRT based methods used in the project.

A. *Types of Obstacles:*

There are two types of obstacles. They are

1) **Static obstacles**: These obstacles do not move with respect to time. So, the obstacle space does not change with the passage of time. In an environment consisting of static obstacles, the generation of collision free optimal path happens only once.
2) **Dynamic obstacles**: These obstacles move with respect to time. So, the obstacle space constantly keep changing with the passage of time. In an environment consisting of dynamic obstacles, the generation of a free optimal path may change continuously as dynamic obstacles can impede already generated path.

B. *Sampling Based Algorithms*

Below is a brief introduction to RRT with its variants.

1) **RRT (Rapidly Exploring Random Trees)**: In RRT, the map is generated with the help of randomly sampled points. Newly, sampled points are joined to nearest nearest node of the RRT tree and this continues until the target is reached. Each sampled point is checked against the robot workspace for validity. Refer to the pseudo code for detailed explanation of the algorithm.

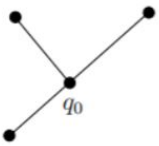

Figure 5. RRT

2) **RRT***: RRT* is just an extension of RRT but more computationally expensive. Compared to RRT, RRT* generates a more optimized path. The main difference between RRT and RRT* is that after joining a random node to a point, the nodes are **rewired** to improve the cost.

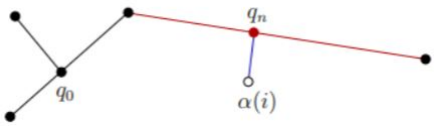

Figure 6. Connecting the tree to nearest edge point

procedure RRT(x_{init})
  $V \leftarrow \{x_{init}\}$
  $E \leftarrow \phi$
  for $i = 1, 2, ..., n$ **do**
    $x_{rand} \leftarrow SampleFree(i)$
    $x_{nearest} \leftarrow Nearest(V, x_{rand})$
    $x_{node} \leftarrow Steer(x_{nearest}, x_{rand})$
    if ObstacleFree($x_{nearest}$, x_{node}) **then**
      $V \leftarrow V \cup \{x_{node}\}$
      $E \leftarrow E \cup \{(x_{nearest}, x_{node})\}$
  return $G = (V, E)$

Figure 7. RRT pseudo code

In rewiring, all the nodes surrounding the newly added node are checked for better parent and this results in minimizing the cost-to-come. The rewiring process occurs around the neighborhood of the newly added node at some threholded distance. Thus, optimality of the path generated is achieved by finding path with minimum cost.

Although RRT* tries to find a more optimal path, the downside is that the rewiring occurs for all newly added nodes and this makes it a computationally expensive. The figure below shows the comparison of nodes exploration between RRT and RRT*. Refer below for the pseudo code for RRT*

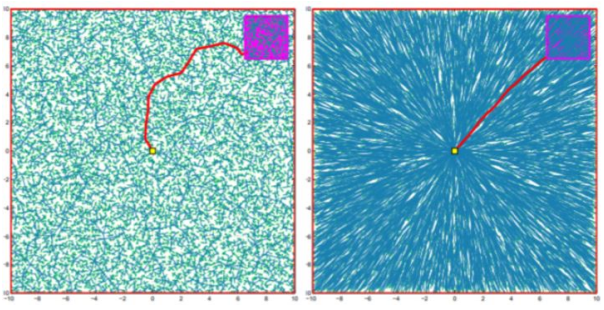

Figure 8. RRT RRT*

The pseudo code for RRT* is given below.

3) **Smoothing the Solution Path:** To avoid sharp edges in path generated, smoothing is carried out. Ideally,

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

Figure 9. RRT* pseudo code

smoothing should be carried out whilst planning. However, only the solution path was smoothed in this problem due to the exhaustive workspace used. This however has the disadvantage of not being robust since the smoothed solution path might be in an obstacle workspace. Gradient descent algorithm tries to minimize the error between the solution path and smoothened path. The equation utilized in smooth is outlined below:

$$\boldsymbol{y_i = y_i + \alpha(x_i - y_i) + \beta(y_{i+1} + y_{i-1} - 2 * y_i)} \quad (1)$$

where x_i is the i^{th} solution coordinate
y_i is the i^{th} smoothened coordinate
y_{i-1} is the $(i-1)^{th}$ smoothened coordinate
y_{i+1} is the $(i+1)^{th}$ smoothened coordinate
α is the smoothing coefficient which controls the degree of smoothing
β is the weight given to each solution node in the solution path

By tuning the α and β parameters we can control the smoothing process.

IV. Path planning for Dynamic obstacles:

For dynamic environment, other moving drones are used as dynamic obstacles. The distance of the other drones are constantly tracked and should any of them obstruct the generated drone solution path and is close to the intended drone, replanning occurs. In replanning, the RRT/RRT* algorithm replans and finds a new path using current position of the drone and the goal position with the updated environment(Obstacle blocking drone's path). Hence, the RRT/RRT* generates new path avoiding the obstacle. This process continues till the drone reaches the goal point. The pseudo code is given below[Reference 2]:

SetObsDestination($numObs$)
SetObsVelocities($numObs$)
SetRobotDestination()
SetRobotVelocity()
while $robotLocation! = GOAL$ **do**
  UpdateObsLocation($numObs$)
  UpdateRobotLocation()
  if *Replan* **then**
    DoReplan()
  end
end

Figure 10. Dynamic Obstacle Avoidance using RRT pseudo code

V. Software and Visualization tool Used

1) Python version 3
2) Matplotlib
3) Pygame

VI. Results

A. *Implementation of RRT and RRT* in 2D environment*

We first implemented RRT and RRT* in 2D using pygame. The following are the plots obtained:

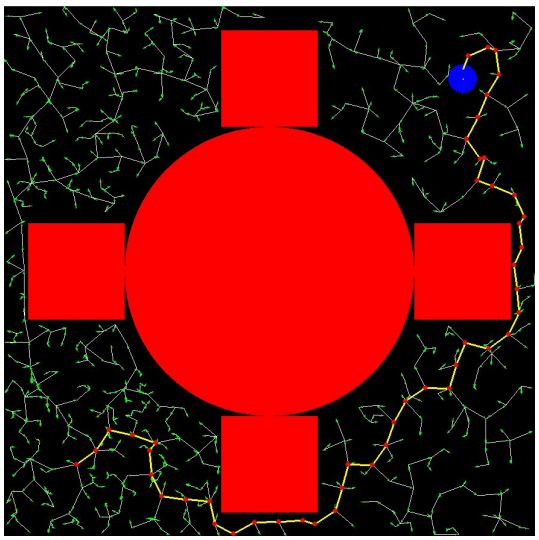

Figure 11. RRT implementation in 2D

The simulation time for RRT is about 2 seconds. We can clearly see that the path obtained is not optimal but the we get to the solution faster.

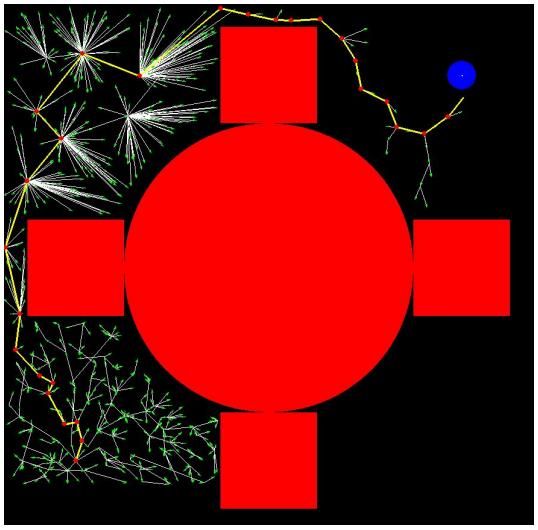

Figure 12. RRT* implementation in 2D

The simulation time for RRT* is about 113 seconds. We can clearly see that the path obtained is more optimized than RRT but the time taken to reach the solution is significantly increased. The nodes exploration is directed towards the goal position.

B. Implementation of RRT in 3D environment

As earlier mentioned, for simulation in 3D matplotlib package was used. The following are the results of RRT and RRT* for a static 3D environment.

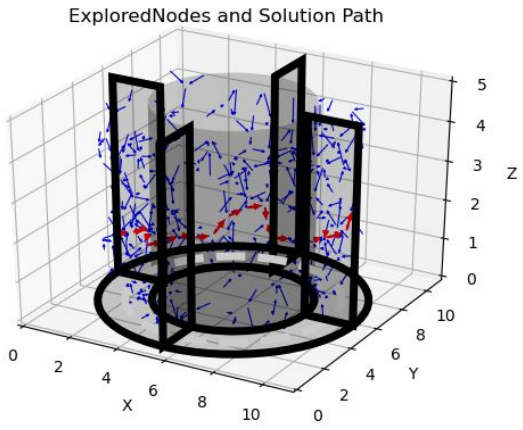

Figure 13. RRT implementation in 3D

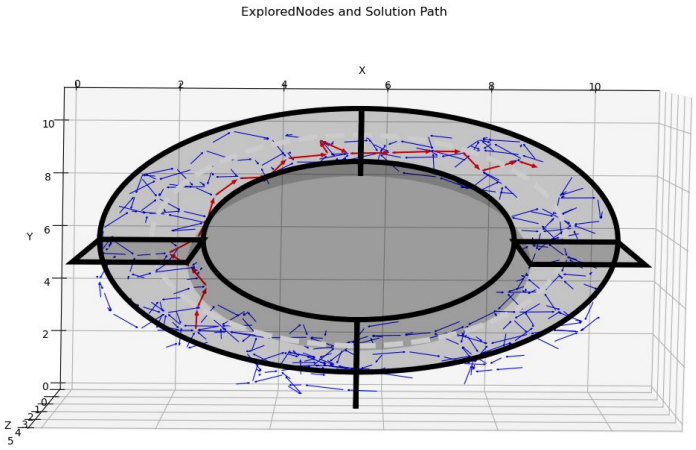

Figure 14. RRT implementation in 3D top view

In addition to this, we have created an animation of drone using matplotlib which follows the solution trajectory.

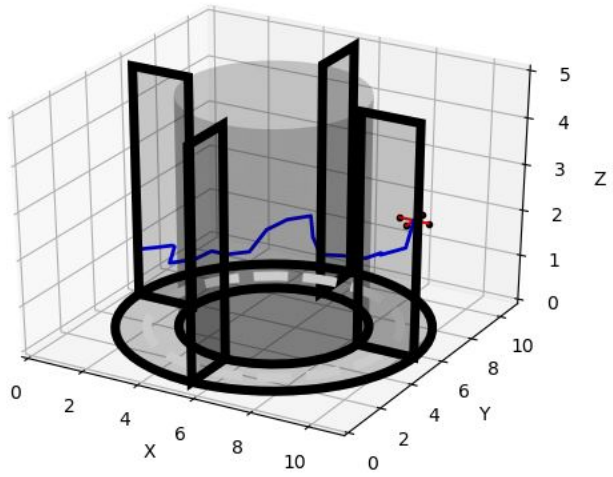

Figure 15. RRT implementation drone animation

Currently, the path generated by RRT is not smoothened. By implementing the gradient descent algorithm, the solution path is smoothened. The following is the drone trajectory after the path of the drone is smoothened. As RRT generates new path each time, the path shown in the below figures may be little different from the above graphs but difference in the smoothness of the path is clearly seen.

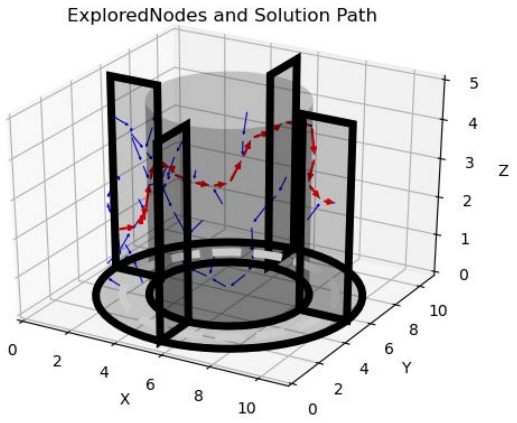

Figure 16. RRT implementation in 3D with smoothing

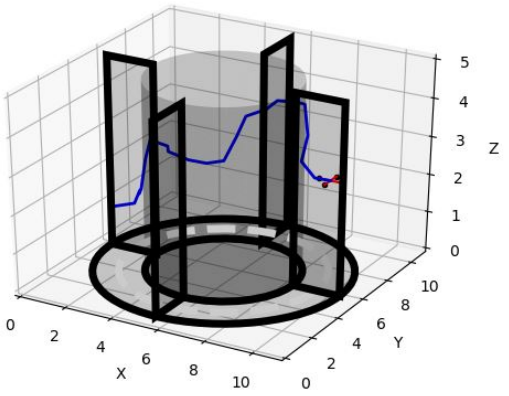

Figure 17. RRT implementation in 3D smooth drone path

C. *Implementation of RRT* in 3D environment*

Similar to RRT, we also implemented RRT* algorithm for a static 3D environment. Here are the results:

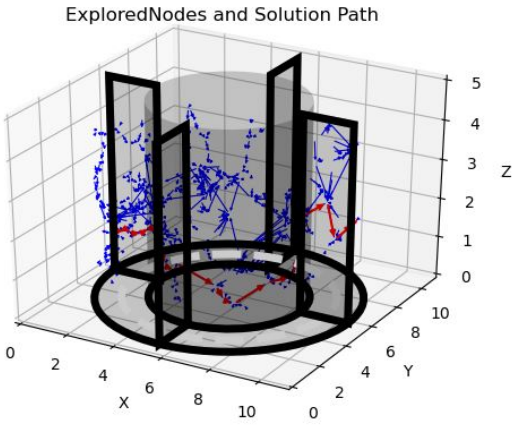

Figure 18. RRT* implementation in 3D

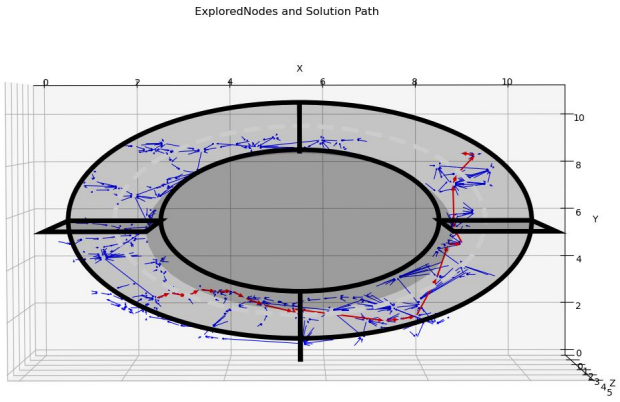

Figure 19. RRT* implementation in 3D top view

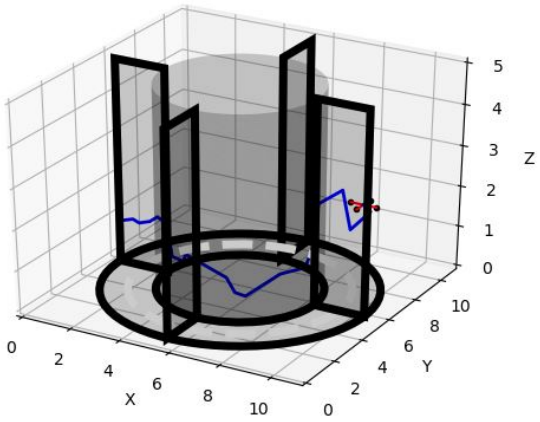

Figure 20. RRT* implementation drone animation

After performing the smoothing, the following are the results:

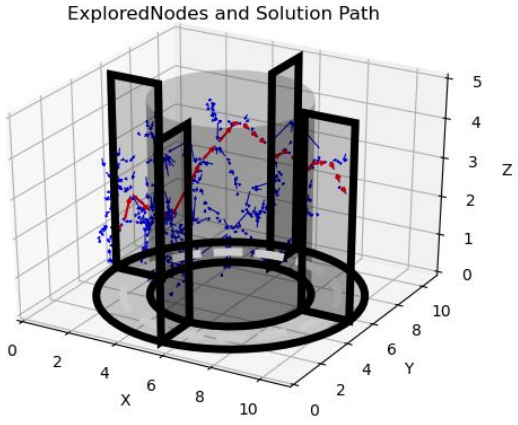

Figure 21. RRT* implementation in 3D with smoothing

D. *Dynamic Environment path planning results:*

We have developed RRT and RRT* algorithms to adapt to dynamic changes in the environment. Below sequence of images explains the pipeline with the output results. The following are the results obtained:

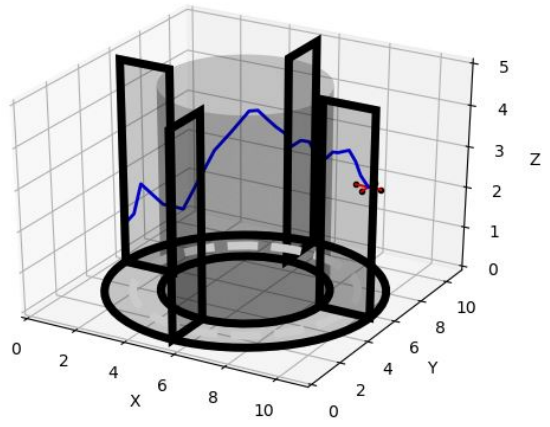

Figure 22. RRT* implementation in 3D smooth drone path

1) First RRT/RRT* generates the initial solution path considering the initial position of the obstacles(drones).

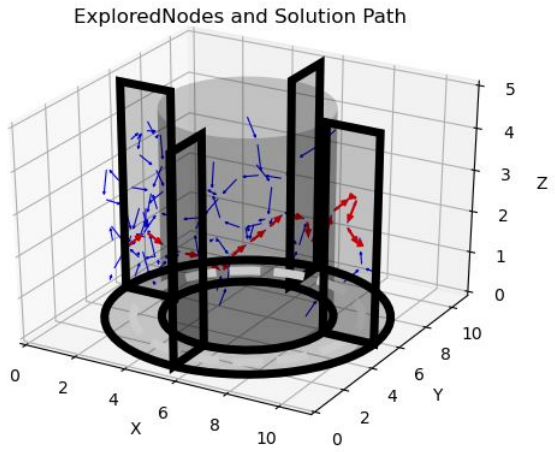

Figure 23. Generating initial solution path

2) The drone animation below shows the drone starting to the follow the initial generated solution path.

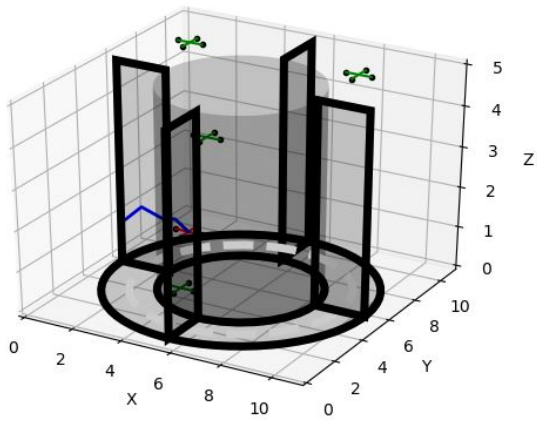

Figure 24. Drone following the initial solution path

3) In the above picture, one of the obstacle drone was obstructing as well as close to the drone. Hence, replanning of the path occurs.

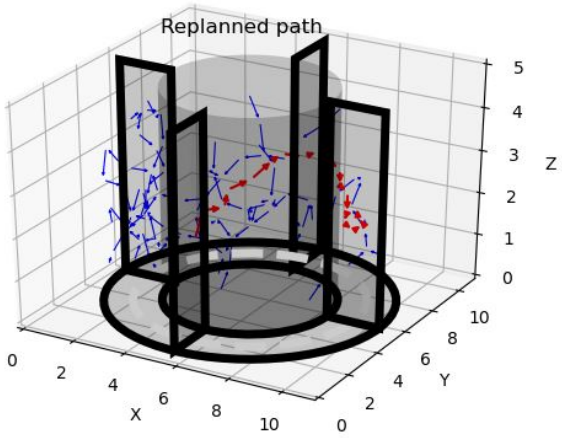

Figure 25. Replanning of drone path from current position

4) Now the figure below shows the animation of the drone following the replanned path

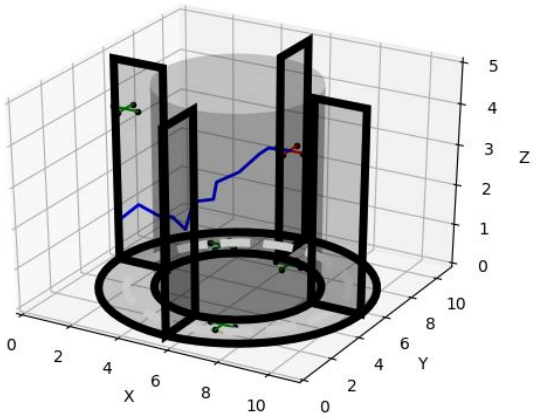

Figure 26. Drone following replanned path

VII. Goal Achieved:

1) Successfully implemented RRT and RRT* algorithms in 2D in static environment.
2) Successfully created a dynamic obstacle space in 3D.
3) Successfully implemented RRT and RRT* algorithms in static and dynamic environment in 3D.
4) Successfully smoothened the solution path using gradient descent algorithm.

VIII. Future Scope

Our future goal would be to integrate the algorithm with ROS and provide a simulation in gazebo.

IX. References:

1) The modified quadrotor.py was initially coded by Daniel-S-Ingram, Refer, https://github.com/daniel-s-ingram
2) Dynamic Obstacle Avoidance pseudo code