RRT for Quadrotor Motion Planning with Danger Zones in the Environment

Shantnu Kakkar[1], Roya Sabbagh Novin[2], Xiang He[3], Dejun Guo[4],

[1234]Department of Mechanical Engineering,
University of Utah,
Salt Lake City, Utah,
Emails: [1]shantnu.kakkar@utah.edu, [2]roya.sabbaghnovin@utah.edu, [3]xiang.he@utah.edu, [4]dejun.guo@utah.edu,

***Abstract*— This paper focuses on the path planning for quadrotors in an environment with dangerous areas. Rapidly-exploring Random Tree (RRT) with considering dynamic constraints of the quadrotor is applied to find the safest path. A Graphic User Interface (GUI) is developed to define the environment conveniently with various objects available. In order to avoid danger zones in the environment, a cost function is introduced providing a trade-off between the path length and risk of danger zones. After obtaining the path from RRT, B-Spline method is adopted to smooth the final path. Finally, the results of simulation based on Robot Operation System (ROS) and OpenGL show the 3D visualization of the process and validate the methods used above.**

Keywords*— Motion planning, quadrotor, RRT, danger, dynamic constraints

I. Introduction

Nowadays, unmanned aerial vehicles (UAVs) are applied to various tasks, such as search and rescue or surveillance. Quadrotor have been studied for use in such tasks due to the advantage in control that multiple rotors gives. Because of this, quadrotor can hover and move through complex environments that traditional UAVs cannot navigate, for instance in a damaged structure.

Many problems are needed to be dealt with to achieve the autonomous navigation for quadrotor in such tasks. How to get to the goal in a smooth path and avoid the obstacles as well as the dangerous area is the crucial issue we will focus on in this project.

First, a map will be built based on an indoor environment which the user can edit through the GUI. Then, a set of way points will be generated using a goal-biased RRT constrained algorithm. These path is collision-free and will guide the quadrotor from the starting point to the goal while avoiding the dangerous areas as much as possible.

Finally, a path smoothing algorithm based on 4th-order B-spline curves will be applied to make the path even more agile and dynamically-feasible (that is satisfy velocity and acceleration constraints). Moreover, for better visualization, a 3D environment is developed and results will be simulated using Robot Operation System (ROS) and OpenGL.

II. Related work

Various methods have been employed in literature to solve motion planning problems for different systems, such as cell decomposition [1], Voronoi Diagram [2], heuristic planning [3], and potential fields [2]. However these methods rely on an explicit representation of the geometry of free configuration space, due to which as the number of dimensions of the configuration space grow, these methods start failing considerably. Because of this shortcoming and many other [4], sampling based planners were introduced.

PRM is a popular sampling based algorithm that was used for generating a road map, and setting queries in this. But it was realized that it is not fast for single query applications, which led to introduction of RRT.

In the literature, a lot of work has been done involving RRT [2]. Many new features have been added to the basic RRT, such as putting kino-dynamic constraints [1], introducing dangerous zones [5], and shortening and smoothing the trajectory [6].

The aim of this report is to integrate ideas from various sources in literature and come up with a planning algorithm that is efficient, optimal and probabilistically complete [7][8]. The uniqueness in this work is that we have been able to implement the notion of dangerous zones, while satisfying constraints on upper limit of acceleration, ensuring continuity of velocity, acceleration and radius of curvature throughout the path.

III. Method

This section includes different parts of the proposed algorithm for the problem of this paper. The algorithm is based on the basic RRT, with additional parts to consider costs of the environment and dynamic constraints. Finally, after finding the path, a smoothing algorithm is applied in order to make it more practical.

A. *Basic RRT*

In the following the pseudocode of basic RRT is provided which will be modified later in this report with the additional sections.

Algorithm: Basic RRT

Initial position q_{init}, goal position q_{goal}, number of vertices in RRT K, incremental distance ε, bias $bias$

Output: RRT Tree $plan$

$RRT.init(q_{init}, q_{goal}, num, \varepsilon, bias)$
for k=1 to K
    $q_{sample} = sample(bias)$
    $q_{new}, q_{parent} = extend(RRT.tree, q_{sample})$
    if $in\_collision(q_{new}, q_{parent})$
        $continue$
    $add\_node(q_{new}, q_{parent})$
    $if\ is\_goal\ (q_{new})$
        $return\ get\_back\_path$

In this pseudocode, sample (bias) finds a random sample with a bias. The next step tries to extend the tree by finding the nearest neighbour in the tree to the randomly generated point, and moving ε towards it. If the new configuration is in collision, then the loop is continued else the node is added to the tree.

We have chosen basic rrt because it is fast, collision checking in RRT could be a separate module (we sample in continuous domain and do collision checking in a discrete domain by mapping our continuous sample to discrete domain), and finally RRT gives us the option of setting constraints on our path.

B. *Cost function*

As discussed in the introduction section, this paper focuses on motion planning in environments with danger zones. A danger zone is an area with hazardous condition, i.e., heating, poor wireless networking, air flow. Each of these conditions can affect movement of the quadrotor in a different way.

Considering danger zones in the motion planning means that the path is allowed to intersect with the danger zones but this should be avoided as much as possible considering the level of danger [5]. In other words, the level of danger determines how much a specific area should be avoided. This can be either set by user in scale of 0 to 10 or be called as low-medium-high levels and converted to real values in the algorithm. In this paper the later approach is used.

Now the question is how to insert the costs for the dangerous areas into the algorithm. One way, which is used in this paper, is to define a new property named as cost for each node and use it in the process of finding the best neighbor node. This means that instead of choosing the nearest neighbor, as in basic RRT, the node with the lowest cost will be chosen for each sample to connect with.

In the basic RRT, the nearest node was found using equation below:

$$d = \left| new_{sample} - n_i \right|_2, \quad \forall i = 1, \dots, n$$

Now, considering danger zones, some cost should be added to this function in order to represent the risk to go through these areas. After trying different methods, it seems that multiplying cost value of each node by this distance gives a good result on avoiding these areas reasonably. In other words:

$$new\_cost_i = cost_i \times d, \quad \forall i = 1, \dots, n$$

Then, the node with the lowest cost is chosen as the neighbor node for the new sample. Figure 1 shows the effect of adding these cost for a 2D environment with areas that have different level of dangers.

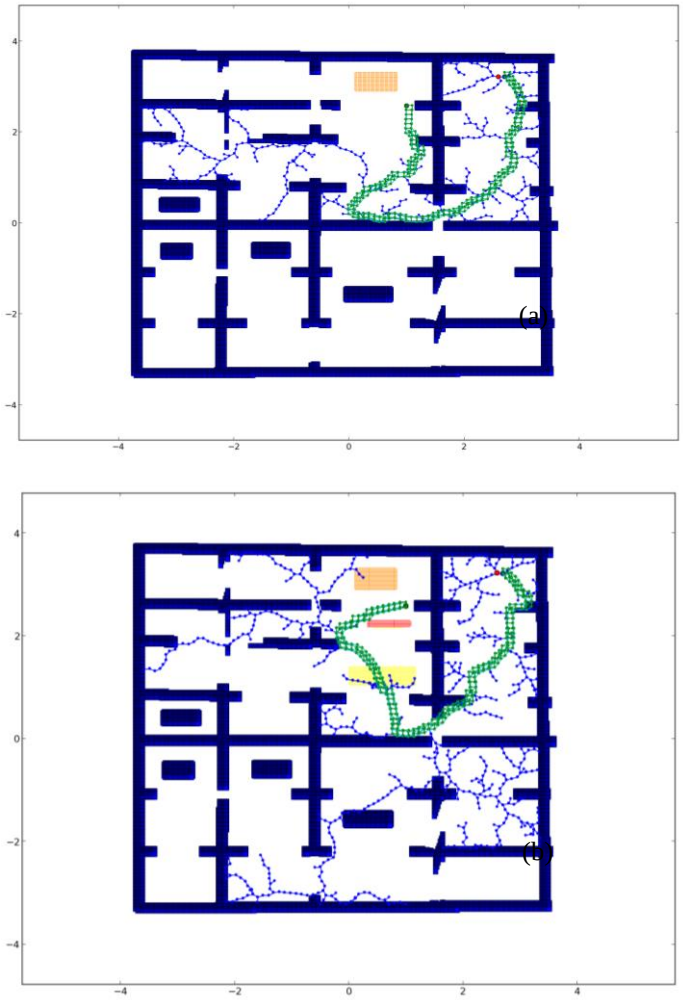

Figure 1. The difference between basic RRT (a) and having danger zones in the environment (b). The yellow, orange and red rectangles have low (cost=1.3), medium (cost=1.4) and high (cost=1.5) level of danger, respectively.

C. *Constraints*

The maximum acceleration of quadcopter is considered here. The assumption is that the norm of velocity is constant. Thus, the direction of velocity can vary according to the current random sampling points. But there exists a maximum

acceleration to turn. Such limitation leads to the maximum turning angle of the robot (Fig. 2), such that a sector region in front of the quadcopter is the only available places where it can reach at the next moment.

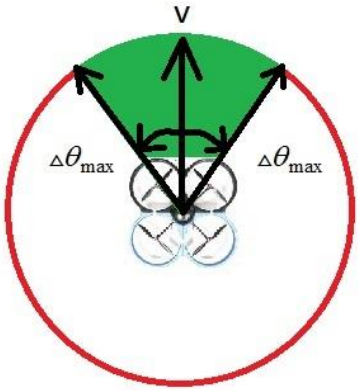

Figure 2. The difference between basic RRT (a) and having danger zones in the environment (b). The yellow, orange and red rectangles have low (cost=1.3), medium (cost=1.4) and high (cost=1.5) level of danger, respectively.

D. Path Smoothing

Until now, the danger zones have been avoided and upper bound on acceleration has been achieved. However, the path is still not smooth. To further smoothen the path, curve fitting is applied. The requirement to ensure velocity and acceleration continuity throughout the path.

There are two requirements when deciding which curve function to use for the path smoothing function. The first is Local Support. When fitting a curve to the way-points, there are cases when the generated curve intersects with the obstacles. However, it is normally observed that instead of the whole curve, only a part of it gets intersected. Hence it is logical to think that only that part of the curve, which collides with the obstacle, needs to be reshaped.

In order to achieve this, a curve that can be reshaped only at certain regions is needed. After a thorough literature survey and feasibility study, it sounds like that B-spline curve meets this requirement. As part of this project, it was initially decided to achieve this reshaping by introducing pseudo control points to the way points. There will be cases when addition of pseudo control points won't be sufficient to avoid collision. In those cases, there would be a need to re-position the way-points which are responsible for the collision.

The second requirement is the need of continuity of velocity, acceleration and radius of curvature throughout the path. Since, 4th order B-spline curves are c2 continuous, they can be used to satisfy velocity, acceleration and radius of curvature constraints.

While fitting B-spline curve, it is observed that for small incremental distanceε, the curve passed close to the waypoints. It is not required to re-position any waypoint or introduce pseudo-random points, except for the case when a very small obstacle is there that may come in way of the curve. For larger incremental distance ε, it was observed that the fitted curve collided with the obstacles. Hence, in this case some waypoints are moved to ensure collision is avoided. However, this increased the computation time.

Another advantage of choosing B-spline curve is that they can be easily implemented using De-Boors recursion algorithm, thus not adding much of any overhead. Finally, it should be mentioned that the implementation of De-Boor's algorithm for generating B-spline was done in MATLAB. Figure 3 shows the smoothing achieved after fitting the curve:

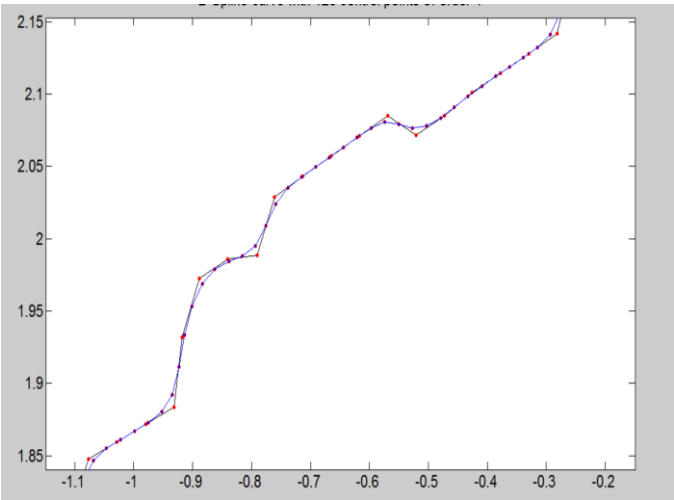

Figure 3. Part of a smoothed path after fitting B-spline. Black line is the path before and blue line is the path after smoothing.

IV. RESULTS

In the following, first, the graphical interface developed for the purpose of this paper is briefly presented. Next, the complete pseudocode of the algorithm is provided. Finally, obtained results are reported.

A. GUI

In order to make the code more user-friendly, a Graphical User Interface (GUI) is developed in MATLAB. In this GUI, the user can first set the workspace boundaries and then add different objects to the environment, for instance, walls, doors, obstacles and danger areas.

Moreover, it contains a list of different shapes for each object and a list of level of danger for danger zones. Figure 4 represents the developed GUI and a map drawn in it.

After drawing the complete map, it will be saved as an image and then by reading this image and using the RGB components produced in the image, different parts, such as free space, obstacles, start, goal will be distinguished. Finally, a txt file, based on the RGB values, will be generated and fed to the Python code.

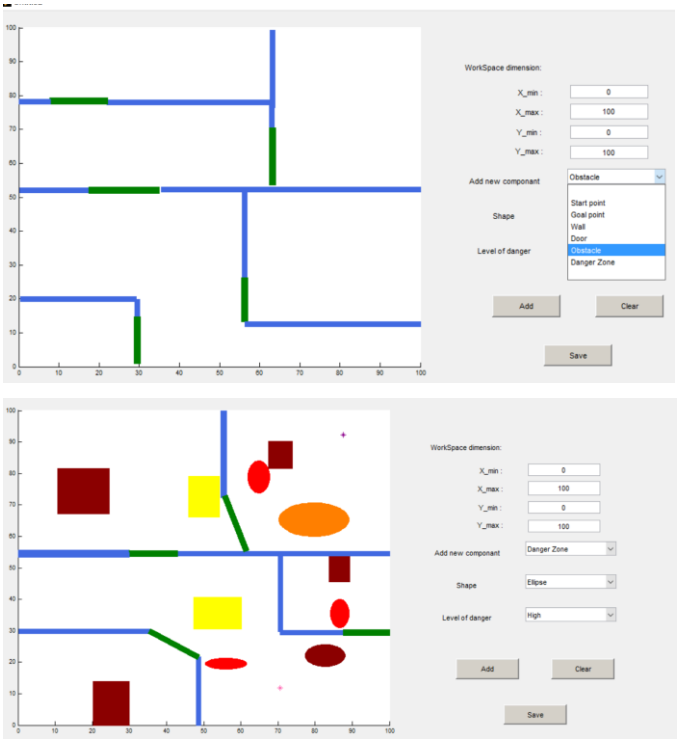

Figure 4. The developed GUI for generating the map, containing list of different objects with different shapes.

B. Algorithm Pseudo Code

The project is programmed in several different programming languages. The main part of the program where the RRT is implemented is programmed in Python while the GUI and the path smoothing is written in MATLAB. In addition, the environment visualization uses the OpenGL in C++. The algorithm is briefly described as followed:

Algorithm Danger Cost and Constraint based RRT

Input: Initial position q_{init}, goal position q_{goal}, number of vertices in RRT K, incremental distance ε, bias $bias$, acceleration limit $accel\_lim$.

Output: RRT Tree $plan$.

$RRT.init(q_{init}, q_{goal}, num, \varepsilon, bias, accel\_lim)$
for k=1 to K
  $q_{sample} = sample(bias)$
  $cost = check\_cost(q_{sample})$
  $q_{new}, q_{parent} = extend\_with\_cost(RRT.tree,\ cost)$
  if $in\_collision(q_{new}, q_{parent})$
    continue
  $q_{new}.calculate\_velocity()$
  if $calculate\_constraint(q_{new}, q_{parent}) > accel\_lim$
    continue
  $add\_node(q_{new}, q_{parent})$
  if $is\_goal(q_{new})$
    return get_back_path

The RRT will start with randomly sampling in the configuration space with a bias of $bias$, which means the sampling gets the goal as the sample with probability of $bias$. And here the $check\_cost$ function basically checks the cost scaler of the new sample point in the danger cost map and its return is based on the level of the danger. The cost is calculated in $extend\_with\_cost$ using the function in the cost function section.

The function $in\_collision$ checks the map info and if the new state is in collision or if the line from parent state to the new state goes through obstacles, RRT would discard this new sample and start a new one.

Moreover, $calculate\_velocity$ will return the velocity vector and the constraint checking will limit the angle of changing within the constraint of maximum acceleration.

One thing that can be improved is to broaden the limitation of acceleration from the changing of angle to the real changing of speed, because by now after the quadrotor achieve the maximum velocity, it won't deaccelerate until it reaches the goal. By doing so the new generated node won't be discarded and it may save more time.

As it was mentioned before, the project is broken into three main languages and five parts each of the part communicate through files. Figure 5 presents the flow chart of the program.

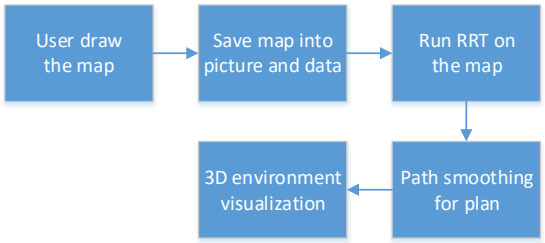

Figure 5. Project flow chart.

The first two blocks are built in MATLAB and map info is based on the RGB value of the picture and is saved for each pixel as a grid of the discrete map. The scale chosen in this project is 2cm per pixel.

C. 3D Environment Visualization

Visualization is built in ROS (Robot Operating System) with OpenGL. In order to make the quadrotor fly more real, inverse dynamics are calculated. Since the quadrotor can only achieve position change via changing its orientation, adding the orientation of the quadrotor makes it seem more real.

The external dynamic of quadrotor can be defined below:

$$\begin{cases} \ddot{x} = -cos\phi sin\theta u_1 \\ \ddot{y} = sin\phi u_1 \\ \ddot{z} = g - cos\phi cos\theta u_1 \end{cases}$$

Where u_1 denotes the input of acceleration along z_b in body frame. And to make it simple, the yaw angle is set to be always

zero as well as $\ddot{z}$. To achieve the required acceleration x, y, z in world frame, the desired roll angle ϕ and pitch angle θ can be calculated as below:

$$\begin{cases} \tan\theta = \frac{\ddot{x}_d}{\ddot{z}_d - g} \\ \tan\phi = \frac{\ddot{y}_d \cos\theta}{g - \ddot{z}_d} \\ u_1 = \frac{g - \ddot{z}_d}{\cos\theta\cos\varphi} \end{cases}$$

Since the path smoothing part provides a differentially flat trajectory based on the RRT plan, getting the smoothed attitude is not difficult. The relationship of position and required attitude is represented in Fig. 6.

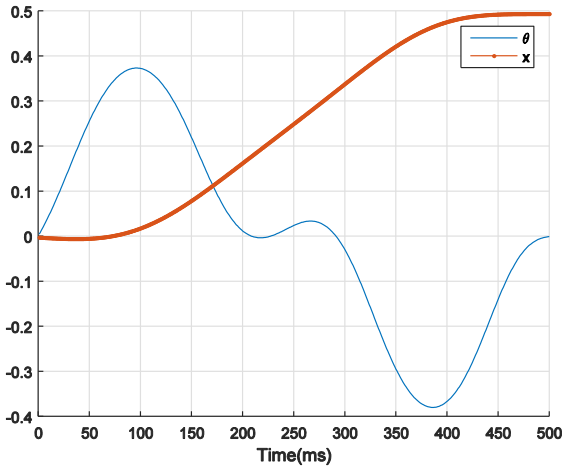

Figure 6. Relationship between position and angle.

It also should be noted that the quadrotor needs to lean forward to accelerate and lean backward to deaccelerate.

As for the visualization part, two nodes in ROS are developed to fulfill the task. The generated path and orientation is read from one node and it publishes the data at a certain frequency of 30Hz. The other node, the visualization node where the environment is established in OpenGL, subscribes the data and fresh the position and orientation of the quadrotor to follow. Figure 7 provides the node diagram from rqt_graph.

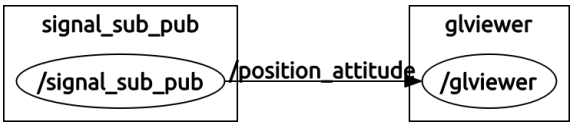

Figure 7. Node diagram from rqt_graph.

Finally, the GLviewer node reads the map file which is stored in text and each element represent 2cm^2 and is represented in cuboid. The danger zone is represented in color of yellow, orange and red, corresponding to different levels of danger. A representation of one of the final results in the 3D environment is provided in Fig. 8.

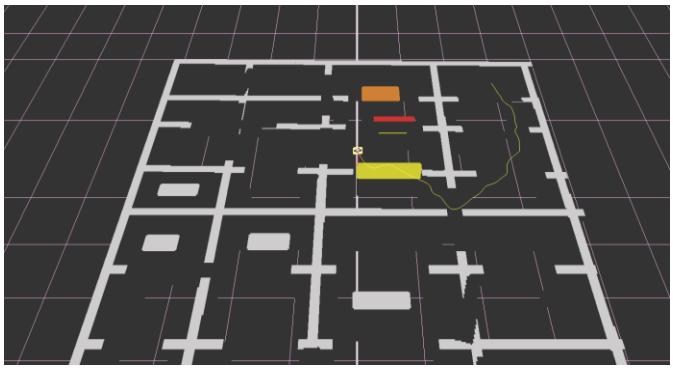

Figure 8. 3D environment in simulation.

V. Analysis

In this section, three main properties for motion planning algorithms, namely completeness, optimality and efficiency are discussed.

- Completeness: In general, randomized approaches are understood to be probabilistically complete [7], meaning that the probability that they will produce a solution approaches 1 as more time is spent. However, they cannot determine if no solution exists. In [8], a proof is presented for the probabilistic completeness of RRT-based algorithms when planning with constraints which includes the algorithm used in this paper.
- Optimality: As it is known, RRT algorithms are not optimal. Several methods have been suggested in order to make them more optimal, i.e., RRT* or shortcutting. However, due to the constraints in this problem, they could not be implemented. As a result, the path found by the proposed algorithm is not guaranteed to be the optimal path in terms of Euclidean distance.
- Efficiency: RRT is an efficient data structure and sampling scheme to quickly search high-dimensional spaces. With a very little bias, it can be even faster and grow more efficiently towards the goal. In this paper a bias of 5% was used. It should be noted that the computational time is also dependent on grid size of the map and step length as well.

VI. Conclusion

In this paper, a RRT-based method to cope with quadrotor path planning problem in damaged structures were proposed. Firstly, Graphic User Interface (GUI) was developed to define the environment conveniently. Then, RRT was used as basic path generating method. Besides, the limitation acceleration of robots was considered to meet its dynamic constraints, and cost function was proposed to avoid danger zones in the environment. Next, B-spline method was implemented to achieve the smoothing of the final path. Finally, obtained results of 3D visualized simulation based on ROS and OpenGL are reported which validate the performance of the proposed method.

REFERENCES

[1] LaValle, Steven M. *Planning algorithms*. Cambridge university press, 2006.

[2] Choset, Howie M. Principles of robot motion: theory, algorithms, and implementation. MIT press, 2005.

[3] Gupta, Kamal Kant, and Zhenping Guo. "Motion planning for many degrees of freedom: Sequential search with backtracking." *Robotics and Automation, IEEE Transactions on* 11.6 (1995): 897-906.

[4] Gilks, Walter R., Sylvia Richardson, and David J. Spiegelhalter. "Introducing markov chain monte carlo." *Markov chain Monte Carlo in practice* 1 (1996): 19.

[5] D. Sent, M. H. Overmars. "Motion planning in environments with danger zones." *IEEE International Conference on Robotics and Automation, ICRA.* Vol. 2, 2001.

[6] Koyuncu, Emre, and Gokhan Inalhan. "A probabilistic b-spline motion planning algorithm for unmanned helicopters flying in dense 3d environments." *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2008.

[7] M. Zucker, et al. "Chomp: Covariant hamiltonian optimization for motion planning." *The International Journal of Robotics Research 32.9-10 (2013): 1164-1193.*

[8] D. Berenson, and S. Srinivasa. "Probabilistically complete planning with end-effector pose constraints." *IEEE International Conference on Robotics and Automation (ICRA), 2010.*