RO47005 Planning and Decision Making Final Project

David Schep, 5643384; Dielof van Loon, 5346894; Tatsuki Fujioka, 5849837

I. INTRODUCTION

In this project, motion planning for a quadrotor in presence of static obstacles is developed using state of the art methods, namely, the rapidly exploring random tree (RRT) and the kinodynamic rapidly exploring random tree (RRT-u). During the simulations of the two motion planners, the static obstacles are randomly generated in the shape of convex hulls. The goal of this project is to compare performance of the RRT and RRT-u algorithms using metrics such as path planning time, traveling time and travelled distance. Despite both RRT and RRT-u being sampling-based incremental global planning methods, the RRT algorithm generates a holonomic global path to a goal whilst RRT-u provides a non-holonomic global path, in which velocity and acceleration bounds of an agent are respected [1]. Among other studies on RRT-u after its first debut in [1], we attempt to base our implementation of the RRT-u on the RRT-u algorithm proposed by Urban Eriksson [2], where edges and nodes of a tree are expanded based on the utility function which considers the traveling time of trajectories under two cases: the maximum velocity case and the maximum acceleration case.

In general, the RRT algorithm without a non-holonomic local planner is not well suited for the motion planning of quadrotor as it does not take non-holonomic dynamics of the quadrotor into account. For instance, when it connects the randomly sampled vertices, the resulting trajectory includes the non-smooth corners that might not be feasible to follow for the quadrotor with high velocities. By considering dynamic constraints, the RRT-u algorithm in [2] produces smooth trajectories. The author in [2] has also proposed to use dynamic resizing of the sampling region since the RRT often struggles to find a path to the goal due to the increase of the sampling space as the robot explores and covers more areas. In this project, however, the implementation of the dynamic resizing is omitted by assuming the prior knowledge of the locations of the obstacles and some bounding box of the obstacles.

For simulation environment, gym-pybullet-drones is chosen to provide accurate results on the performance of the motion planner. This simulator is written in Python and offers compatibility with gymnasium, an API standard for single-agent reinforcement learning environments. This compatibility will come in handy later in the development stage of the motion planner.

The organization of the report is as follows. First, a basic dynamics of the quadrotor robot model is introduced in II which includes some assumptions on the dynamics of the quarotor. In III, implementations of RRT and RRT-u are described. After which, the results of our implementations for the RRT and the RRT-u are presented and compared. Finally, the obtained results and comparisons are discussed in V, including recommendations of the future improvements for our implementations.

II. ROBOT MODEL

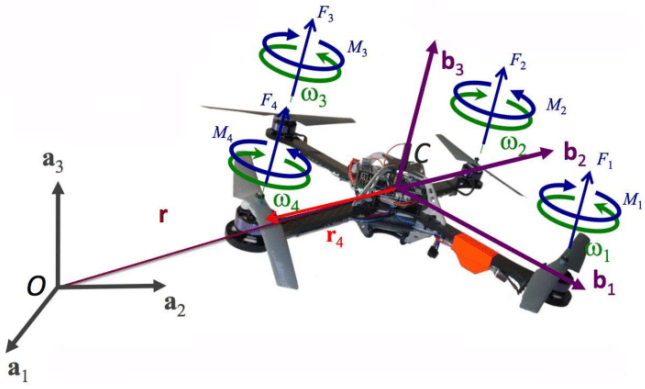


Fig. 1: Diagram of quadrotor

In this section, the simplified dynamic equations of a quadrotor (depicted in Figure 1) consisting of five rigid bodies, the main body and the four rotors, are presented. During the derivation of the dynamic equations, the following three simplifications are made. First, inertia tensor $\mathbf{I}_B$ in the body-fixed frame is assumed to be diagonal due to the symmetry around b_1- and b_2-axes [4], that is,

$$\mathbf{I}_B = \begin{bmatrix} I_{b_1 b_1} & 0 & 0 \\ 0 & I_{b_2 b_2} & 0 \\ 0 & 0 & I_{b_3 b_3} \end{bmatrix}, \tag{1}$$

where $I_{b_i b_i}$ denotes the moment of inertia around b_i-axis. Second, the moment of inertia of the propellers is assumed negligible when compared to $\mathbf{I}_B$. Lastly, the aerodynamic drag is ignored.

Using the Newton's second law of motion the translational dynamics in the inertial frame is expressed as

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}, \tag{2}$$

where m is the mass of the quadrotor, g the gravitational acceleration, F_i the thrust from ith propeller in the body-fixed frame and $\mathbf{R}$ the rotation matrix which translates the thrusts into the inertial frame, given by

$$\mathbf{R}(\psi, \phi, \theta) = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) \tag{3}$$

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$
$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

with yaw angle ψ, pitch angle θ and roll angle ϕ defined around b_3-, b_2- and b_1-axes, respectively [4]. The rotational dynamics in the body-fixed frame is formulated using Euler's rotation equation [4]:

$$\mathbf{I}_B \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ -M_1 + M_2 - M_3 + M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \mathbf{I}_B \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{5}$$

where vector $[p \;\; q \;\; r]^T$ represents the angle rate of the body-fixed frame, L the length between the center of mass (COM) and each rotor, and M_i the reaction torque of ith rotor. In Equation 5, the first term on the RHS represents the resultant torque while the second term represents the gyroscopic effect.

Having derived the two dynamic equations in equations 2 and 5, the two equations are linearized around hovering attitude in equation 6 by using the small-angle approximation and setting the collective thrust $T = \sum_{i=1}^{4} F_i = mg$.

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{6}$$

$$\mathbf{x} = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$$
$$A = \begin{bmatrix} \mathbf{0}_{6\times3} & \mathbf{0}_{6\times3} & \mathbf{I}_{6\times6} \\ \mathbf{0}_{3\times3} & \mathbf{a} & \mathbf{0}_{3\times6} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times6} \end{bmatrix}$$
$$\mathbf{a} = \begin{bmatrix} g\sin\psi_0 & g\cos\psi_0 & 0 \\ -g\cos\psi_0 & g\sin\psi_0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$B = \begin{bmatrix} \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_{b_1b_1}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{b_2b_2}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{b_3b_3}} \end{bmatrix}, \mathbf{u} = \begin{bmatrix} T \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix}$$

To sum up, the linearized dynamics of the quadrotor is expressed in the LTI system in Equation 6 with 12 states and 4 inputs consisting of the collective thrust and the torque around each axis.

Regarding workspace and configuration space of the quadrotor, a standard quadrotor has workspace $\mathcal{W} = \mathbb{R}^3$ and configuration space $\mathcal{C} = \mathbb{R}^3 \times SO(3)$. Using the differential flatness of the quadrotor, one can even reduce the configuration space to $\mathcal{C} = \mathbb{R}^3 \times SO(1)$. For the motion planning in this project, we make an assumption that the quadrotor is modelled as a point mass to reduce the complexity of the motion planner, which results in the configuration space $\mathcal{C} = \mathbb{R}^3$.

III. MOTION PLANNING

In this section, algorithms of the RRT and the RRT-u including sub-functions such as steering function and collision check function are discussed. In the algorithm of RRT, a steering function of a simple straight line is used to steer from one node to another. The algorithm of the RRT-u, on the other hand, uses a steering function which finds a smooth path based on the minimum traveling time of robot and the bounds of the velocity and acceleration. For both algorithms, the collision check of randomly sampled nodes are based on hyperplane equations of the obstacles in the shape of convex hulls.

A. RRT

The algorithm of the RRT is shown in Algorithm 1.

Algorithm 1: RRT algorithm(q_0, q_{goal}, N, r_{margin})

```
initialization: V ← {q_0}, E ← ∅
for i = 0 : N do
    q ← RandomSampling()
    if not NodeCollisionCheck(q) then
        q_neighbor ← FindNeighbor(q)
        e ← StraightLine(q, q_neighbor)
        if not EdgeCollisionCheck(e) then
            V ← {V ∪ q}
            E ← {E ∪ e}
    if ‖q − q_goal‖ ≤ r_margin then
        Break out of for-loop
Return V, E
```

In Algorithm 1, q_0 denotes the start vertex of the quadrotor; q_{goal} the goal vertex; $q_{negihbor}$ the neighboring vertex, e the edge, r_{margin} the margin between the sampled and goal vertices, V the set of vertices, E the set of edges. The details of the functions involved in the RRT algorithm are defined in the followings.

RandomSampling function returns a random point in the configuration space (i.e., a 3D position since $\mathcal{C} = \mathbb{R}^3$) with probability of 0.99, or otherwise the function returns the position of the goal vertex with probability of 0.01. Although the choice of the bias of $99:1$ may lack theoretical justification, it has been speculated in [3] to be in a range that works well experimentally and indeed, we have observed during the implementation the significant improvement on not only the planner time but also the distance of the path to the goal.

NodeCollisionCheck function detects whether the randomly sampled point is in the obstacle-free space. The detection is based on the equations of hyperplanes constituting the convex obstacles. For example, let $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ denote sets of normal vectors of triangular simplices for a convex hull in $x-$, $y-$ and $z-$directions; $\mathbf{v_0} = (x_0, y_0, z_0)$ and $\mathbf{v} = (x, y, z)$ denote the base position of the convex hull and the randomly sampled point relative to the global

frame, respectively. If the dot products of the point vector $\mathbf{v} = (x, y, z)$ and the normal vectors satisfies

$$(A, B, C)\cdot(\mathbf{v}-\mathbf{v_0}) = A(x-x_0)+B(y-y_0)+C(z-z_0) \leq 0, \quad (7)$$

then the randomly sample point is either inside or on the convex hull. In addition to this condition, we incorporate the expansion of the collision boundaries of the convex obstacles by adding the radius of the quadrotor on the RHS of Equation 7, which allows us to randomly sample points from the collision-free space instead of the obstacle-free space.

FindNeighbor function in the RRT algorithm simply returns the vertex in the expanded set V for which the Euclidean distance to the sampled vertex is the shortest.

It is important to mention that the algorithm includes a stopping criteria at line 10 in Algorithm 1. It checks whether the Euclidean distance between the goal and sampled vertices is equal/smaller than r_{margin}. Consequently, the algorithm returns an approximate solution rather than an exact one.

B. RRT-u

The algorithm of RRT-u is shown in Algorithm 2.

Algorithm 2: RRT-u algorithm(q_0, q_{goal}, N)

1 initialization: $V \leftarrow \{q_0\}$, $E \leftarrow \emptyset$
2 **for** $i = 0 : N$ **do**
3 $q \leftarrow RandomSampling()$
4 $G \leftarrow \emptyset$
5 **if not** $NodeCollisionCheck(q)$ **then**
6 **for** q' *in* V **do**
7 cost, $e' \leftarrow Steer(q, q', \sigma(q, q'))$
8 $G \leftarrow G \cup \{\text{cost} \times q' \times e'\}$
9 $q_{neighbor}$, $e \leftarrow \arg\min(\text{cost} \in G)$
10 **if not** $EdgeCollisionCheck(e)$ **then**
11 $V \leftarrow \{V \cup q\}$
12 $E \leftarrow \{E \cup e\}$
13 $e_{goal} \leftarrow Steer(q, q_{goal}, \sigma(q, g_{goal}))$
14 **if not** $EdgeCollisionCheck(e_{goal})$ **then**
15 $V \leftarrow \{V \cup q_{goal}\}$
16 $E \leftarrow \{E \cup e_{goal}\}$
17 Break out of for-loop
18 Return V, E

RRT-u [2] is in most ways identical to RRT. The main change from RRT is the way that nodes are connected to the tree. We first sample a new node q. This candidate node is compared with every other node in the tree. A cost is calculated based on dynamical constraints. This cost is simply defined as the time it takes a point mass to travel from the parent node to the new node. To be able to calculate this time, every node stores its velocity and acceleration as well as the position. After this calculation, the candidate parent node with the lowest estimated time is chosen. The quadratic path that the point mass takes is then sampled and the samples are collision checked. If the path does not collide with any obstacles, the node and the edge is added to the tree.

When a new node is added to the tree, RRT-u will attempt to connect this node with the goal node. If this connection is possible, following the dynamical constraints and the collision check, the goal node is added to the tree. This is different from RRT where the goal node is sampled randomly with a probability of 0.01. Every node stores the time it estimates the pointmass to make when traveling from the parent position to its own position. When a new node is added to the tree, it adds the estimated flight time of the parent to its own calculated time. In this way every node knows how much time it takes to get to its position from the root of the tree. When multiple branches have added the goal node to their branch, the goal path with the lowest total time is chosen. In Algorithm 2,

IV. RESULTS

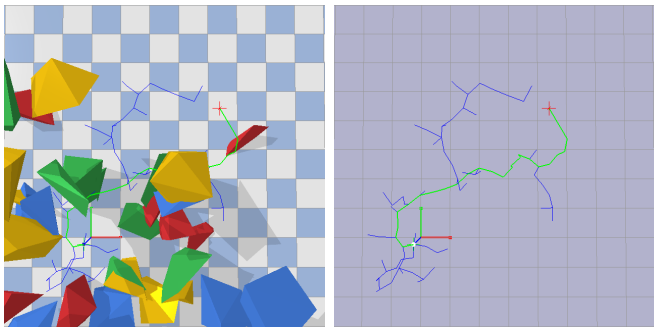
In this section, the results of the simulation of the RRT and the RRT-u algorithm are compared. The simulation results have been collected by generating a random environment with 30 randomly generated convex hulls, while making sure the start and goal node has been cleared from any obstacles. Each algorithm was ran in 100 different random environments, while tracking the performance metrics averages shown in Table I. Both planners are given 3 seconds of planner time, meaning that if the planners cannot find a solution, it is regarded as a failure.

TABLE I: Average of 100 different simulation results

Metric	RRT	RRT-u
Goal Found (%)	0.85	0.98
Goal Reached (%)	0.941	0.938
Travel time (s)	4.897	3.108
Error X (m)	70.821	84.305
Error Y (m)	70.709	84.055
Error Z (m)	10.501	9.580
Path distance (m)	7.254	6.758

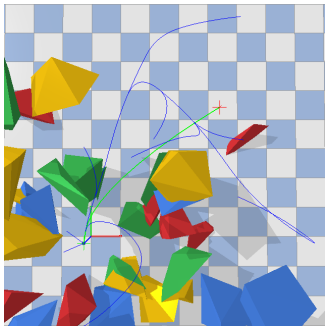
It can be seen in Table I that RRT-u often outperforms RRT, as only RRT beats RRT-u in tracking error. The reason for this is that the tracking error is calculated from the drone to the current waypoint which makes for an unfair comparison, as the two algorithms spread their waypoints differently. RRT uses a linear spread between its current and next node, while RRT-u increases its distance between waypoints when velocity increases. This means that tracking error of the quadrotor with the PID controller can be accentuated with higher velocity. Besides this, RRT-u is able to find the goal a lot more often than RRT, while creating a shorter path which can be travelled a lot quicker than the path generated by RRT. This is due to the nature of RRT-u, as it generates a smooth instead of a jagged path.

RRT and RRT-u can be visually compared in Figure 2, both with and without the environment displayed. This might accentuate the pros of RRT-u the most, as the planning tree of RRT-u can be seen to be smooth, which is related to Figure 3. The environment with the random convex hulls can also be seen in this figure.

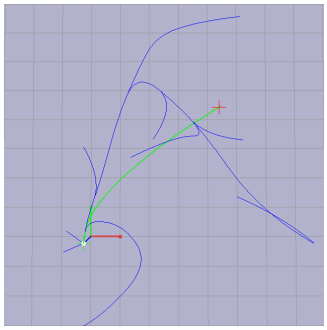


(a) Planning tree of RRT

(b) RRT without visualization of obstacles



(c) Planning tree of RRT-u



(d) RRT-u without visualization of obstacles

Fig. 2: Simulation results for one random realization of obstacles: red cross represents goal vertex, blue lines represent planned trees and green lines represent path-to-goal

In Figure 3 another pro of a smooth curve can be seen. The rate of stability is the percentage of times the drone has reached its destination successfully, while the velocity that the drone travels is plotted on the other axis. It can be seen that RRT-u offers a higher stability, especially at higher velocity. This is due to the drone being able to track the smooth trajectory much better than the jagged trajectory that RRT offers.

V. DISCUSSION

When comparing the solution rate of RRT and RRT-u, one can notice that RRT has a lower success rate than the other. This result is somewhat counter-intuitive since the RRT has more ”degrees of freedom” in the sense that its expansion of edges are less restricted as RRT does not include the kinematic bounds. A plausible reason for the difference in success rates is the difference in frequency of the attempts to directly connect to the goal vertex. For example, the RRT algorithm in Algorithm 1 tries to add a direct edge to the goal vertex with probability of 1% while the RRT-u in Algorithm 2 attempts to connect a sampled vertex to the goal at every iteration, leading to a higher chance of finding a solution for the RRT-u within the maximum allowed planner time (3s).

As for the success rates of the quadrotor reaching the goal for the RRT and RRT-u, the results reveal little to no difference between them, which could be counter-intuitive since we expect that the quadrotor with the RRT-u is more

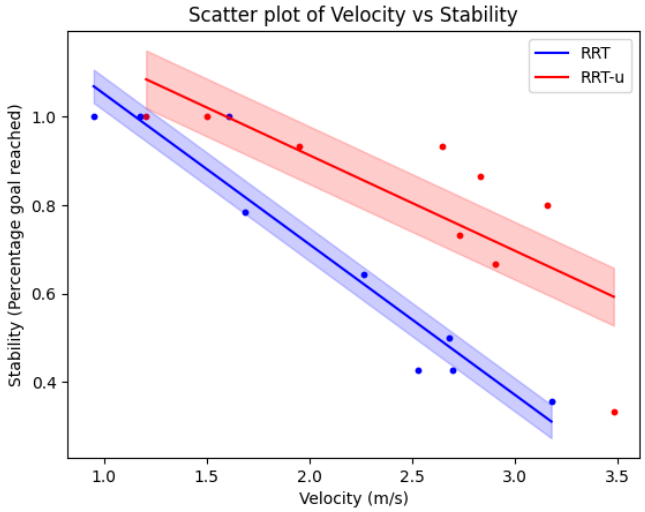


Fig. 3: Scatter plot showing the drone rate of success vs the velocity of the drone.

likely to reach the goal due to the smooth trajectory of the algorithm. This, however, could be attributed to the difference in the number of vertices in the solutions of RRT and RRT-u: as can be seen in Figures 2b and 2d. RRT-u has significantly fewer vertices than the RRT in the trajectory to the goal. Since the same number is used when dividing each edge of the paths of the two algorithms into small waypoints to generate the reference signals to the PID controller, the control over the quadrotor with the RRT-u tend to be more aggressive than the RRT-u, and vice versa. As a consequence of this, the quadrotor with RRT is more likely to move slowly along the trajectory, making the motion more holonomic-like, whereas the aggressive control on the quadrotor with RRT-u uses the advantage of the smooth trajectory.

The results show that the RRT-u tends to provide a shorter path than the RRT. Although the utility function is about the traveling time rather than the travel distance, the results accord with our expectation since avoiding obstacles in a smooth trajectory tends to be shorter than avoiding the same obstacles in a jagged trajectory. This can also be seen in Figures 2b and 2d, where the trajectory of the RRT-u to the goal is shorter than that of the RRT.

Apart from the conclusions made above, we provide the following recommendations for the future improvements of the motion plannings. First of all, as RRT as well as RRT-u are probabilistically complete but not optimal, making both planners optimal is desirable. To make the RRT algorithm an optimal algorithm, we can either incorporate Dijkstra's algorithm with the Euclidean distance as an edge cost or A* algorithm with the Euclidean distance as a heuristic function. Furthermore, in case the differential flatness is employed in the motion planners (i.e., configuration space $\mathcal{C} = \mathbb{R}^3 \times SO(1)$) and thus the edge cost is not only the Euclidean distance of two vertices but also the rotation around the yaw angle, the use of the Euclidean distance as a heuristic function in the A* algorithm could be advantageous

as it underestimates the edge cost.

Based on the average values in Table I, it is apparent that there is some room left for improvement in the success rate of the quadrotor reaching the goal position obtained. For RRT, implementing a non-holonomic local planner is necessary to achieve a success rate of 100%. A possible candidate for the local planner is a model predictive controller (MPC) with the equation of motion in Equation 6 as a dynamic equality constraint and the hyperplane equations in Equation 7 to avoid collisions. Additionally, we recommend using a quadratic cost of the states and the inputs in the MPC as the quadratic cost function, combined with the aforementioned linear constrains, forms a convex optimization problem. As for improving the success rate of the RRT-u, the choice of the hyper parameters such as the upper/lower bounds of the velocity and acceleration need to be validated based on the stability of the quadrotor with the PID controller. For example, one could investigate the maximum acceleration and deceleration at which the quadrotor becomes unstable and use them in RRT-u. Alternatively, one could possibly achieve a 100% success rate by combining the dynamic constrains of the quadrotor to the kinematic constraints used in our implementation.

Lastly, although the cause of the relatively low success rate of RRT finding a solution is not known, we deduce that the cause of low success rate is partly attributed to the over-approximation of the collision bounds of the obstacles as it can lead to smaller collision-free space. Therefore, a better implementation of the collision bounds in the gym-pybullet environment needs to be sought for.

REFERENCES

[1] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *J. ACM*, 40(5):1048–1066, nov 1993.

[2] Urban Eriksson. Dynamic path planning for autonomous unmanned aerial vehicles, 2018.

[3] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006.

[4] Caitlin Powers, Daniel Mellinger, and Vijay Kumar. *Quadrotor Kinematics and Dynamics*, pages 307–328. Springer Netherlands, Dordrecht, 2015.