Bi-directional Rapidly-exploring Random Tree (B-RRT) algorithm for a quadrotor moving in 3D-space

Georgios Apostolides (5377498) Guru Deep Singh (5312558) Markos Gkozntaris (5347955) Kevin Voogd (4682688)
Group 31

***Abstract*— This project focuses on the path planning algorithm for a quadrotor - a Bi-directional Rapidly-exploring Random Tree (B-RRT). The algorithm devised optimizes the list of vertices sampled to provide locally optimized planned trajectory. The advantage of B-RRT over traditional RRT is reduced computation time and locally optimized path to goal. We present the algorithm as a simple extension to RRT that could be further extended by more advanced path-planning algorithms. The path computed is smoothened using minimum snap and is then simulated on MATLAB, generating a collision free trajectory while complying with the dynamics of quadrotor.**

I. INTRODUCTION

The project presents the modelling of a Crazyflie 2.0 quadrotor developed by Bitcraze, which is controlled using a position and an attitude controller to follow a trajectory computed using Rapidly-exploring Random Trees (RRT) algorithm and its extensions in order to avoid collision with the obstacles present in the given map. The objective of the project is to plan a path for the quadrotor from a start to a goal position. The map for the project is chosen as a room ablaze with a human stuck inside and the quadrotor should reach the human and provide him with the necessary tools, such as a oxygen mask, or communication instruments like a mobile phone. The project considers the quadrotor moving in a 3D space with an initial position outside the room and goal position near the human. The assumption undertaken is that the fire is considered to be non-progressive, and hence assumed as a static obstacle.

The high dimensionality of the configuration space, the assumption of static obstacles, as well as, the simplicity and robustness of RRT, makes it the best choice compared to algorithms like Probabilistic Roadmap Method (PRM) or motion primitives. The RRT provides a feasible solution if one is found within the constraint of iterations. Since, RRT provides only a feasible solution and not an optimal, researchers nowadays are working on improved algorithms such as RRT*, which provide the optimal path. However, these algorithms are inefficient as sampling randomly explores the entire map, and for that reason researchers have tried to guide the sampling in a way to reduce the computation time [1]–[3]. Apart, from this many other methodologies are also being developed and researched into, like path planning using vector fields, which are in-turn improving the accuracy for path planning [4]. This project employs a bi-birectional guided RRT (B-RRT), which is compared to a traditional RRT. The software package auxiliary to the path

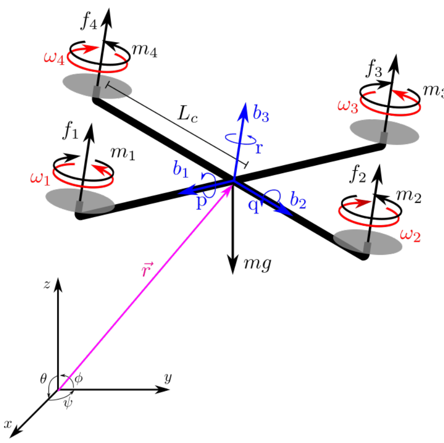

Fig. 1. Schematic drawing of the quadrotor.

planning like for control, simulation, trajectory smoothening, etc are adapted from the project by L. Yiren, C. Myles, L. Wudao and Z. Xuanyu.[1]

II. ROBOT MODEL

A. Kinematic Model

The workspace is $W = R^3$, as the quadrotor can displace to all positions in space provided that there are no obstacles at that place. The configuration space is $C = SE(3)$ [5]. The robot is able to move forward, in a side direction and upwards. In addition to that it can rotate in 3D space using the three Euler angles: Roll (ϕ), Pitch (θ), Yaw (ψ).

The position and the orientation of the quadrotor in the inertial world frame is $\xi_I = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^T$ and the vector containing the linear and angular velocities expressed in the body frame is $\dot{\xi}_B = [u \quad v \quad w \quad p \quad q \quad r]^T = [\mathbf{v}_B \quad \omega_B]^T$. The orthonormal basis vectors that represent the inertial and body-fixed frames are, on the hand hand, $\hat{\mathbf{a}}_\mathbf{1}$, $\hat{\mathbf{a}}_\mathbf{2}$, and $\hat{\mathbf{a}}_\mathbf{3}$, and on the other hand, $\hat{\mathbf{b}}_\mathbf{1}$, $\hat{\mathbf{b}}_\mathbf{2}$, and $\hat{\mathbf{b}}_\mathbf{3}$. In Fig. 1 a schematic drawing is shown, depicting the frames of reference and also the forces and moments acting on the object. The linear velocities in the body-fixed frame are related to the world velocities via a rotation matrix R, and similarly with the angular velocities with a Jacobian matrix J. [6]. In this report, the convention yaw-pitch-roll is used, resulting in the Euler-based rotation matrix $\mathbf{R}(\phi, \theta, \psi) = R_z(\psi)R_x(\phi)R_y(\theta)$ [7].

[1]https://github.com/yrlu/quadrotor

$$\mathbf{R} = \begin{bmatrix} c\theta c\psi - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix}, \quad (1)$$

where c(·) and s(·) are cos(·) and sin(·) respectively and the angular velocities are related by the inverse Jacobian matrix [8]:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (2)$$

B. Dynamic Model

In the dynamic modelling of the quadrotor it is assumed that the only forces acting on it are its own weight and the thrust force. Other forces such as drag are neglected. Then, Newton's second law is derived as:

$$m(\omega_B \times \mathbf{v}_B + \dot{\mathbf{v}}_B) = \mathbf{F}_B = (f_1 + f_2 + f_3 + f_4)\mathbf{b_3} - mg\mathbf{R}^T\mathbf{a}_3, \quad (3)$$

$$\mathbf{I}\omega_B + \omega_B \times (\mathbf{I}\omega_B) = \mathbf{M}_B = (f_2 - f_4)L_c\mathbf{b_1} + (f_3 - f_1)L_c\mathbf{b_2} \quad (4)$$
$$+ (-m_1 + m_2 - m_3 + m_4)\mathbf{b_3},$$

where m is the mass of the quadrotor and $\mathbf{F}_B$ is the resultant force acting on the quadrotor with respect to the body frame. In Eq. (4), $\mathbf{I}$ is the diagonal inertia matrix with components I_x, I_y and I_z and $\mathbf{M}_B$ is the resultant moment expressed in body-frame coordinates. The forces and the moments are related to the speed of the motors as $f_i = k_F\omega_i^2$ and $m_i = k_M\omega_i^2$, where k_F and k_M are constants and ω_i is the angular speed of the rotor. The ratio of these two constants is expressed as $\gamma = \frac{k_M}{k_F}$.

C. Linearization

The system is linearized at the hovering point: $\mathbf{r} = \mathbf{r}_0$, $\theta = \phi = 0$, $\psi = \psi_0$, $\dot{x} = \dot{y} = \dot{z} = 0$, where the roll and pitch angles are small ($c\phi \approx 1$, $s\phi \approx 0$, $c\theta \approx 1$ and $s\theta \approx 0$). The thrust force produced by each propeller is $f_{i,0} = \frac{mg}{4}$. The values for the inputs at hover are $u_{1,0} = mg$, $u_{2,0} = 0$. Linearizing Equation (3) results in:

$$\ddot{x} = g(\Delta\theta\cos\psi_0 + \Delta\phi\sin\psi_0), \quad (5)$$
$$\ddot{y} = g(\Delta\theta\sin\psi_0 - \Delta\phi\cos\psi_0), \quad (6)$$
$$\ddot{z} = \frac{u_1}{m} - g, \quad (7)$$

and Equation (4) in:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \quad (8)$$

D. Control structure

For the purpose of this application a cascade controller is used, with the inner loop being an attitude controller and the outer loop being a position controller. In the inner loop, we specify the orientation through the yaw, pitch and roll angles and feedback the actual attitude and the angular velocity. From this loop, input u_2 is calculated, which is a function of the four thrusts and moments that we get from the motor speeds and is computed based on the desired attitude. On the other hand, from the outer loop, a position feedback loop, we calculate the input u_1, which is a sum of all the thrust forces, based on the specified position and yaw angle generated form the trajectory planner and the actual position and velocity.

In this approach, we consider a linearization of the dynamics for hovering and we take into account the approximated values and assumptions of section II-C. For the design of the feedback loops PD controllers are used.

III. MOTION PLANNING

Motion planning refers to the problem of computing a collision free path from start to goal given an environment with obstacles. Single query sampling based algorithms such as RRT, randomly sample points within the configuration space to construct a collision free path. Several variants of the RRT were introduced either to ensure optimality or faster convergence.

In this work a variant of RRT is presented, upon the basis of a bi-directional function. Two RRT trees are created, one that grows from start and the other from goal. These two trees grow until two sampled points, one from each tree, can connect them without a collision. If such a connection occurs the algorithm stops and the optimization of the created path begins.

The algorithm presented in pseudo-code 1 initiated by storing the starting vertices of the two RRT trees. The two trees are maintained (in *verts*1, *edges*1 and *verts*2, *edges*2) and expanded in every iteration by sampling simultaneously collision free samples (*q_sampled*1 and *q_sampled*2). This samples are connected to their nearest neighbour,of which the index (*near_index*1 and *near_index*2) are found using the Matlab function *dsearchn*, and are checked by the *path_collision_checker* to ensure collision free connections. If both aforementioned criteria are satisfied, then *flag*1 or *flag*2 are raised for each sampled point respectively. This flags ensure that both trees will have the same amount of samples, as each tree is stuck on the iteration loop until both get collision free points and edges. If both flags are raised, than the checking of a collision free edge between the two trees happens. If such an edge exists, then the while loop breaks and the optimization of the path created starts, noted as *path_simplification* on pseudo-code starts, otherwise the expansion of the trees continues.

Once the two trees have been connected their edges and vertices are passed to the *back_tracking* function. The purpose of this function to identify the edges from the two trees that take the drone from the start to the goal. Specifically for the tree starting from the start node, it will return the path from start to the common connection between the two trees. For the tree starting from the goal it will return the path from the goal to the common connection. The two path portions are concatenated and the planned path to be followed by the draw is obtained.

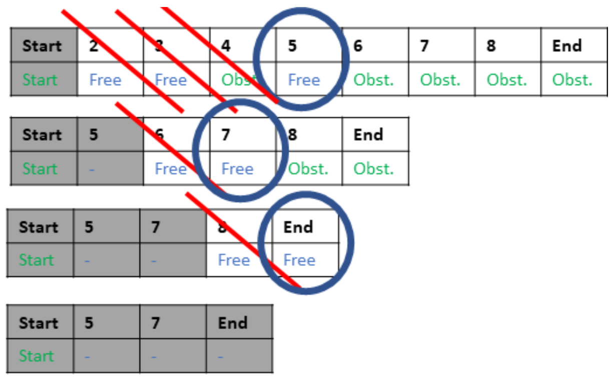

Fig. 2. Representation of the path simplification process. The final path contains does not contain unnecessary connections.

The planned returned from the back tracking algorithm is simplified using the following process: there are two nested iterations over the final path nodes. The algorithm checks whether there are collision-free straight line connections between every pair of nodes in an ordered fashion. First, it checks the starting point with the other points in the list. Then, the feasible connection with the node that has the largest index is kept and the nodes that have indices in-between the first and such node are discarded from the list. The next iteration begins from the node that was used to simplify the path until the last node is reached. The optimization is performed in the total distance traveled. The process is depicted in Figure 2. All possible connections with the right-most shaded gray node are checked. It is visible how the path is simplified.

IV. RESULTS

For the sake of this application we have prepared a map of a room, where fire is considered an obstacle. In addition to fire, there are other obstacles that may exist in a house. We assume that every obstacle is convex and is characterized by a polyhedral geometric shape. Non-convex obstacles can be described as a union of convex obstacles.

The B-RRT algorithm, Figure 4, is compared to a normal RRT algorithm Figure 3. In Figure 5, another map is provided for both algorithms, which does not necessarily imply that it is a room on fire, but it is added for the sake of comparing both algorithms in a different setup. The obstacles are marked in red and blue, and the connections of the trees can be seen clearly. The final path is shown with thick line segments. The tree grows within the boundaries of the map and those are the the maximum values of the axes. The initial and final position are located more or less on the extremes of the map.

For both algorithms, we performed a number of experimental runs on the Matlab environment and two of the basic criteria, run time and number of iterations are presented in Table I. The code with which the results had been produced can be found in the attached files. The values provided come as a result of 1000 runs for each algorithm. It needs to be mentioned that all tests were conducted on the same machine, running an Intel 8550U processor. Furthermore a video link

Algorithm 1: Proposed RRT Algorithm Pseudocode

```
Data: Start, Goal, Map
Result: RRT_path
Initialize no_iter, edges1 and edges2.
verts1 ← start;   verts2 ← goal
flag1 ← False;   flag2 ← False
for k ← 2 to num_iter do
    while True do
        if (flag1==false)
         then q_sampled1 ← sample_point(map);
        if flag2==false
         then q_sampled2 ← sample_point(map);
        point_on_obstacle1 ← collide(map, q_sampled1)
        point_on_obstacle2 ← collide(map, q_sampled2)
        near_index1 ← dsearchn(verts1,q_sampled1)
        near_index2 ← dsearchn(verts2,q_sampled2)
        path_check1 ← [verts1(near_index1), q_sampled1]
        path_check2 ← [verts2(near_index2), q_sampled2]
        path_on_obstacle1←path_collision_checker(map,path_check1)
        path_on_obstacle2←path_collision_checker(map,path_check2)
        if
         (point_on_obstacle1==False)AND(path_on_obstacle1==False)
         then
            edges1{k-1,1}←verts1(near_index1,1:3)
            edges1{k-1,2}←q_sampled1
        end
        if
         (point_on_obstacle2==False)AND(path_on_obstacle2==False)
         then
            edges2{k-1,1}←verts2(near_index2,1:3)
            edges2{k-1,2}←q_sampled2
        end
        if (flag1==True)AND(flag2==True) then
            flag1=flag2=false
            final_path_check← (q_sampled1, q_sampled2)
            path_on_obstacle_final ←
              path_collision_checker(map, final_path_check)
            if (path_on_obstacle_final==False)
             then flag_tree_connected=True ;
            break
        end
    end
    verts1← q_sampled_1;   verts2 ← q_sampled_2
    if (flag_tree_connected=True) then break;
end
if (NOT Converged) then return ERROR;
RRT_path ← back_tracking(start,goal,edges1, edges2, k-1)
RRT_path ← path_simplification(RRT_path, map)
```

can be found on the caption of both figures, with the video of RRT accelerated 500 times, a value that can be derived from how much faster B-RRT runs, in order to be watchable in a lower amount of time.

TABLE I
PERFORMANCE COMPARISON OF RRT AND B-RRT

Algorithm	run time (sec)			no. of iterations		
name	min	mean	max	min	mean	max
RRT	1.5434	31.9930	124.4538	555	11124	42589
proposed B- RRT	0.0982	1.1913	17.4709	2	55.0983	302

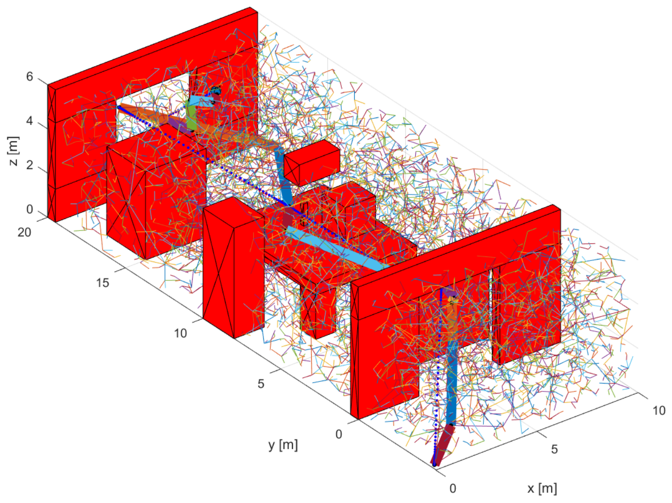

Fig. 3. RRT algorithm path: the thick segments compose the planned with the algorithm.A youtube video is provided showing the tree growth real time [9].

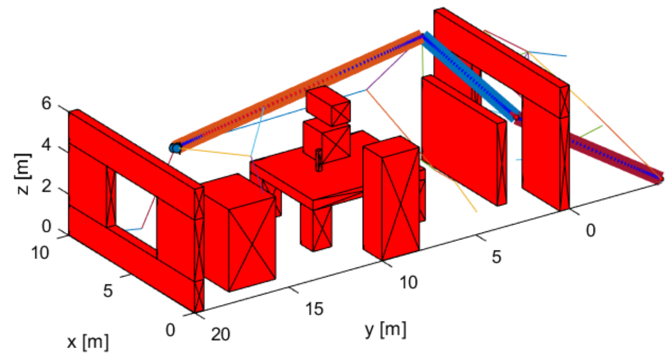

Fig. 4. Proposed RRT algorithm path: the thick segments compose the path planned with the RRT. The total path is considerably smaller. A youtube video is provided showing the tree growth real time [10].

V. DISCUSSION

From the results, it can be seen that not only the proposed B-RRT algorithm produces a better trajectory, but also converges much faster in a shorter amount of time. If the maximum number of sample iterations is reduced significantly, e.g. 300, the RRT never converges, while the proposed algorithm does it almost every time. Our algorithm reduces the total computational stress significantly due to the lower number of iterations it requires. However, it should be noted that every iteration stresses the computational system more, as two trees are being computed simultaneously, which justifies the maximum time of the proposed B-RRT compared to RRT, even though the last requires much more samples.

A bidirectional algorithm, such as the one proposed, can show significantly better results in a map such as that of Figure 5, where the number of iterations needed to converge to the goal, especially in the region near it, is noticeably smaller. That depends on the tolerance region that is set. The later, is calculated based on difference parameters like the minimum obstacle size considered, etc.

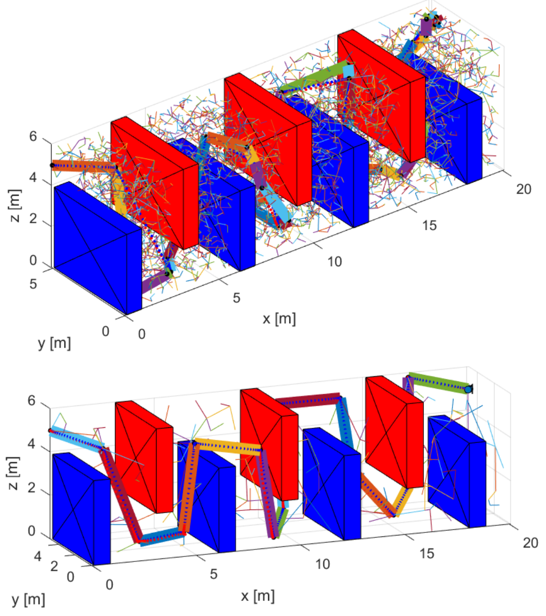

Fig. 5. a) RRT algorithm. b) Proposed RRT algorithm

The aforementioned, exploits the advantages of a bidirectional algorithm. However, the proposed algorithm, uses a collision free condition to connect the two trees. This 'heuristic' works most efficiently when we can expect relatively open spaces for the majority of the planning. For example, for a space with no obstacles in between, a connection would be immediate.

The complexity of the algorithm is: $2O(|V/2| \log(|V/2| + |E/2|)) + O(V_F^2)$. The first part of the sum refers to the time complexity of bi-directional RRT (B-RRT), and the second part relates to the optimization (or simplification) of the path obtained that has V_F nodes, which consists of two loops: one that iterates over the nodes and the second one, iterates over the paths connecting the node of the first loop with the others. Therefore, the algorithm optimizes the path obtained in the bi-directional RRT, however, this does not mean it will become globally optimal. The optimization reduces the distance to travel of the path obtained in. Just as conventional RRT, the proposed B-RRT is probabilistically complete because the probability to finding a solution, if one exists, approaches one as the number of samples taken reaches infinity. The B-RRT does not ensure optimality even on the limit, based on the fact that RRT does not ensure either asymptotic optimality [11].

The variant, RRT* performs better since it provides the global optimal path. However, it is not yet clear if the compilation time required for RRT* is greater than that of the proposed method and that can be investigated in the future. For future work, the presented B-RRT method can be improved to B-RRT* so the globally optimal solution can be found if possible. In order to do so, the RRT section needs to be adapted to RRT*.

References

[1] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints," *CoRR*, vol. abs/1205.5088, 2012. [Online]. Available: http://arxiv.org/abs/1205.5088

[2] Y. Dong, C. Fu, and E. Kayacan, "Rrt-based 3d path planning for formation landing of quadrotor uavs," in *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2016, pp. 1–6.

[3] Z. Tang, B. Chen, R. Lan, and S. Li, "Vector field guided rrt* based on motion primitives for quadrotor kinodynamic planning," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 3, pp. 1325–1339, Dec 2020. [Online]. Available: https://doi.org/10.1007/s10846-020-01231-y

[4] J. P. Wilhelm and G. Clem, "Vector field uav guidance for path following and obstacle avoidance with minimal deviation," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 8, pp. 1848–1856, 2019. [Online]. Available: https://doi.org/10.2514/1.G004053

[5] L. Quan, L. Han, B. Zhou, S. Shen, and F. Gao, "Survey of UAV motion planning," *IET Cyber-Systems and Robotics*, vol. 2, no. 1, pp. 14–21, 2020.

[6] F. Sabatino, "Quadrotor control: Modeling, nonlinear control design, and simulation," Master's thesis, KTH Royal Institute of Technology, Sweden, 2015.

[7] D. Lee, T. Burg, D. Dawson, D. Shu, B. Xian, and E. Tatlicioglu, "Robust tracking control of an underactuated quadrotor aerial-robot based on a parametric uncertain model," pp. 3187 – 3192, 11 2009.

[8] M. Triantafyllou and F. Hover, *Maneuvering and Control of Marine Vehicles*. Cambridge, Massachusetts, USA: Massachusetts of Institute of Technology, 2003.

[9] G. Apostolides, G. D. Singh, K. Voogd, and M. Gkozntaris. RRT algorithm. TU Delft. [Online]. Available: https://youtu.be/7G8JMncbQEI

[10] ——. Proposed B-RRT algorithm. TU Delft. [Online]. Available: https://youtu.be/5SEZZBKRWCk

[11] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, p. 1–11, Jun 2015. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2015.02.007

The link to the repository where the simulation can be reproduced is:
`https://gitlab.com/gurudeep1998/group-31-planning-and-decision-making-project`