Fly through ”real” trees!
Team Nimbus Navigators

Chaitanya Sriram Gaddipati
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: cgaddipati@wpi.edu

Ankit Talele
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: amtalele@wpi.edu

Shiva Surya Lolla
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetss 01609
Email: slolla@wpi.edu

Abstract—**In this project, we implement the navigation stack we developed in project p2a on DJI Tello EDU quadcopter to navigate from a start to goal position along the generated path in a pre-mapped environment.**

I. Introduction

Autonomous navigation using quadcopters demands robust path planning to determine an optimal or near-optimal route, effective trajectory planning to ensure smooth motion, and dependable control to execute the planned motion accurately. In this study, we present the execution of our comprehensive pipeline for the autonomous navigation of quadcopters within a pre-mapped environment peppered with obstacles. We visualize the RRT* tree generation, planned path, trajectory and odometry from Tello in Blender which provides an intuitive visual perspective on the navigation process. We uploaded our code on an NVIDIA Jetson Orin Nano which acts as the computer for the Tello and sends position control commands to it so that we succesfully achieve our objective of navigating from the start to goal without colliding with obstacles. Our study showcases the quadcopter's navigation capabilities, highlighting the accuracy and adaptability of our approach.

II. DJI Tello quadcopter

The DJI Tello EDU is a state-of-the-art drone designed with compactness and functionality in mind. It has a weight of approximately 80 grams, inclusive of its propellers and battery, and dimensions of 98×92.5×41 mm. The drone features 3-inch propellers and boasts an array of integrated systems such as a range finder, barometer, LED indicators, an advanced vision system. Additionally, a Micro USB port ensures power and data transfer capabilities. Regarding flight performance, the Tello EDU can traverse up to 100 meters, reach speeds of 8 meters per second, sustain a flight for up to 13 minutes, and achieve a maximum altitude of 30 meters. Its power system is uniquely designed with a detachable 1.1Ah/3.8V battery for extended use and convenience. Images and videos are stored in JPG and MP4 formats, respectively, and the inclusion of Electronic Image Stabilization (EIS) ensures high-quality and stable footage.

III. NVIDIA Jetson Orin Nano

We integrated the NVIDIA Jetson Orin module, a state-of-the-art AI computing system with our DJI Tello EDU quadcopter. This module stands out due to its energy efficiency and great performance. Specifically, the Jetson Orin delivers up to 275 trillion operations per second (TOPS), signifying an eightfold performance enhancement over its preceding generation. Such computational prowess allows for the simultaneous execution of multiple AI inference pipelines, a critical requirement for real-time processing in autonomous systems. Furthermore, it offers robust high-speed interface support, allowing a variety of sensor integrations essential for drone operations.

IV. Map Reader

Our autonomous navigation process fundamentally hinges on the precise representation and understanding of the environment. To this end, our approach incorporates an **Environment** class that serves as the foundational block in reading, processing, and graphically rendering the 3D space wherein the quadcopter operates. This class parses a descriptive file which encodes key environment parameters like spatial boundaries and obstacles. In the interest of navigation safety, each block undergoes a 'bloating' process. This introduces a safety margin around the obstacle, preventing the quadcopter from venturing too close and risking potential collisions. The 3D environment is digitally mapped within a map called as **map_array** - a strategic array where '0's signify obstacles and '1's symbolize free spaces. This array serves as the digitized terrain for the RRT* algorithm.

V. RRT* path planner

The RRT* algorithm, tailored for a 3D environment, provides a sophisticated method for pathfinding amidst obstacles. At its core, each node in the algorithm contains three spatial coordinates (x, y, z), a reference to its parent node, and an accumulated cost from the origin. The map array is initialized which is taken from the map reader. Accompanying this, a starting node and a destination node are defined, with the former being placed within a list of vertices set for expansion. Prior to each search iteration, the map is refreshed, situating the starting node appropriately.

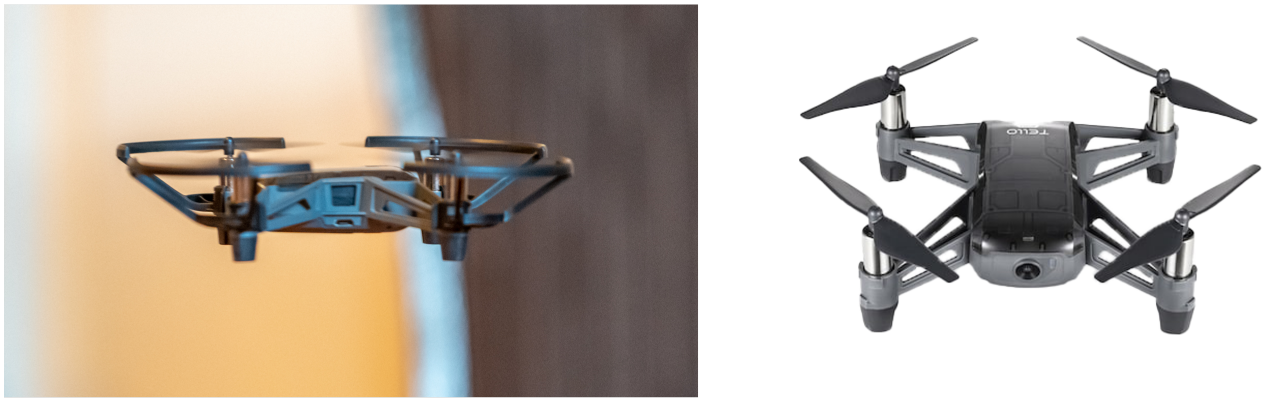
Fig. 1. DJI Tello

Fig. 2. DJI Tello

A fundamental component of the algorithm is its distance measurement technique, leveraging the Euclidean distance formula. For collision checking, the Bresenham's line algorithm, has been adapted to the 3D context. Through this, it checks for potential collisions between two points by evaluating every intermediary point on path.

As the algorithm iterates, it either selects the goal node or crafts a random point within the 3D space. The selection depends on a predefined goal bias, ensuring a delicate balance between exploration and direct pathfinding.

Once a point is decided upon, the algorithm identifies the nearest existing node within its current vertices. From this node, it then ”steers” towards the random point. If this point is distant beyond a fixed range, a new intermediate node is calculated in its direction. However, if it's closer, the point itself gets selected. A noteworthy feature of RRT* is its ability to identify neighbors of a given node within a set radius. This forms the basis of its 'rewire' function, a mechanism designed to optimize the path. Essentially, this function evaluates if a newly added node can offer a shorter route to its neighboring nodes. If a shorter, obstacle-free path is detected, it promptly ”rewires” or updates the parentage of the nodes in question, ensuring that the evolving path is not just valid, but also cost-efficient.

In essence, RRT* emerges as an innovative tree-based pathfinding algorithm. By judiciously sampling random points in a 3D landscape, expanding purposefully in their direction, and continuously refining its tree structure it returns a path. The path formed consists of waypoints leading from the start to the goal.

The tree expansion is visualized in blender where the sampled nodes are visualized as spheres connected by cylinders in blender.

VI. Trajectory generation

The waypoints generated from the path planner and taken for trajectory generation. The goal is to generate a smooth trajectory between waypoints that minimizes acceleration. We use cubic polynomials to represent the trajectory between two waypoints. The x, y, z coordinates are represented as below as a function (f) of time(t) between the waypoints.

Algorithm 1 RRT* Algorithm

```
Data: $n\_pts$, $neighbor\_size$, $goal\_sample\_probability$,
      $steer\_distance$, $neighbor\_radius$
Result: Path or empty list
1  Procedure RRT_star($n\_pts, neighbor\_size$)
2      Initialize map for $sample\_number = 1$ to $n\_pts$ do
3          $random\_sample \leftarrow$ get_new_point($goal\_sample\_probability$)
           $nearest\_node, best\_dist \leftarrow$ get_nearest_node($random\_sample$)
           $sample\_node \leftarrow$ steer($random\_sample, nearest\_node, steer\_distance$)
           $neighbors \leftarrow$ get_neighbors($sample\_node, neighbor\_radius$)
           $best\_neighbor \leftarrow$ GetBestNeighbor($neighbors$)
           if $check\_collision(sample\_node, best\_neighbor)$ then
4              $sample\_node$.parent $\leftarrow best\_neighbor$
5          rewire($sample\_node$, $neighbors$)
           if distance to goal is small then
6              found $\leftarrow$ True
7      if found then
8          $path \leftarrow$ ConstructPath($goal$) return $path$
9      else
10         print("No path found") return []
```

$$f(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

To determine the coefficients $a_0, \ldots, a_3$, we impose constraints based on the given waypoints and the desire to have a smooth trajectory.

Constraints

Given waypoints $p_0, p_1, \ldots, p_n$ at times $t_0, t_1, \ldots, t_n$:
1. The trajectory starts at p_0 at t_0:

$$f(t_0) = p_0$$

2. The trajectory goes through each waypoint:

$$f(t_i) = p_i, \quad i = 1, \ldots, n-1$$

3. The trajectory ends at p_n at t_n:

$$f(t_n) = p_n$$

4. The velocity is continuous at each waypoint:

$$f'(t_i^-) = f'(t_i^+), \quad i = 1, \ldots, n-1$$

5. The acceleration is continuous at each waypoint:

$$f''(t_i^-) = f''(t_i^+), \quad i = 1, \ldots, n-1$$

Using these constraints, we can set up a system of equations to determine the polynomial coefficients.

Matrix Formulation

We express the system of equations in matrix form $Ax = b$, where:

- A is the matrix derived from evaluating the cubic polynomial and its derivatives at the given waypoints and times.
- x is the vector of polynomial coefficients we want to find.
- b is the vector containing the waypoints and derived conditions.

To find the coefficients x, we used the pseudo inverse functionality of numpy:

$$x = A^{\dagger} b$$

Once the desired coefficients are obtained for the x, y and z coordinates as a function of time between every two waypoints, we now have the curve equations between two waypoints, which can be used to obtain the positions, velocities and accelerations at each instant of time between the, leading to a dynamically feasible trajectory between the waypoints.

POSITION CONTROLLER

Once we have the waypoints generated, we have the desired coordinates at each instance of time. We take each of their position coordinates and save them in a csv file. We then used the position control functionality (go_xyz_speed) of Tello in order to send it the saved coordinates in sequence. This enables the execution of the trajectory on the Tello. The drone follows the desired trajectory effectively in an actual environment whose map was given to us with desired start and goal positions, thereby completing the implementation of our pipeline.

CONCLUSION

In this comprehensive study, we implemented autonomous navigation on DJI Tello EDU quadcopter, showcasing safe and efficient traversal in pre-mapped environments with obstacles. The DJI Tello EDU, a cutting-edge, lightweight quadcopter, equipped with NVIDIA Jetson Orin Nano, formed the backbone of our experiments. The intricate process began with obtaining the precise map of the environment with the start and goal locations. This paved the way for the sophisticated RRT* algorithm, tailored for a 3D domain, to find a path with waypoints from start to goal amidst obstacles. With a path in place, the trajectory generation ensured a smooth journey between waypoints using cubic polynomials, with constraints ensuring continuity and smoothness. Finally, the position controller translated these waypoint locations into real-world motion on the Tello effectively enabling real-world navigation from start to the desired goal position.

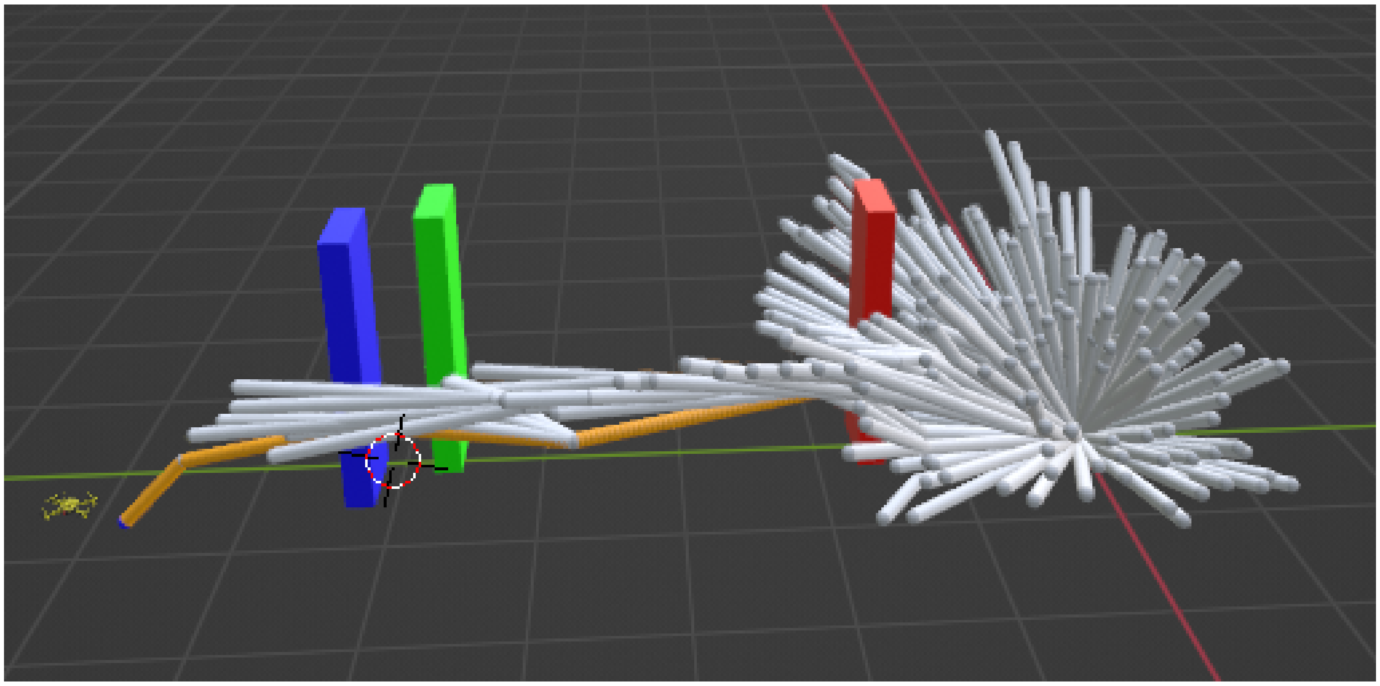

Fig. 3. Trajectory for the map given with RRT* tree

Fig. 4. Ghosted photo of quadcopter crashing while running

References

[1] https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378

[2] Nitin J. Sanket, ”Trajectory and Motion Planning,” RBE595-F02-ST: Hands-On Autonomous Aerial Robotics, Class 8, [Worcester Polytechnic Institute], [2023].

[3] https://github.com/damiafuentes/DJITelloPy/tree/master/
[4] https://djitellopy.readthedocs.io/en/latest/tello/djitellopy.tello.Tello.go_xyz_speed