P2B: Flying Through the Real Trees!

Dushyant Patil
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
dpatil1@wpi.edu

Keshubh Sharma
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
kssharma@wpi.edu

***Abstract*—This project presents an approach for drone navigation over a known map. We use RRT* algorithm to find obstacle avoiding path between start and goal. We use DJITellopy library functions to physically move thr DJI Tello drone over the generated path. We have also listed our findings as a quick start guide for using position control on a DJI Tello Edu drone for navigation on simplistic trajectories.**

I. Problem Statement

The aim of this project is to execute actual navigation (and motion planning) in a known forest. For this we will be using the RRT* algorithm to get a strictly linear path. The estimated path is then tracked by the drone with the help of a in-built PID controller.

Fig. 1. Tello in trees

II. World Map

The data provided to us was the textformat which consists of information about boundaries of 'world' and boundaries of obstacles.

1) **boundary**: This parameter is defined as $[x_{min} \quad y_{min} \quad z_{min} \quad x_{max} \quad y_{max} \quad z_{max}]$ where, $x_{min}, y_{min}, z_{min}$ defines the lower left point & $x_{max}, y_{max}, z_{max}$ defines the upper right point of the rectangular environment.
2) **block**: This parameter is defined as $[x_{min} \quad y_{min} \quad z_{min} \quad x_{max} \quad y_{max} \quad z_{max} \quad r \quad g \quad b]$ where, $x_{min}, y_{min}, z_{min}$ defines the lower left point, $x_{max}, y_{max}, z_{max}$ defines the upper right point & r, g, b defines the color of the cuboidal block.

We load this data in blender as cube objects using `primitive_cube_add` function of bpy module. The Washburn laboratory has exactly same setup which replicated the map file given:

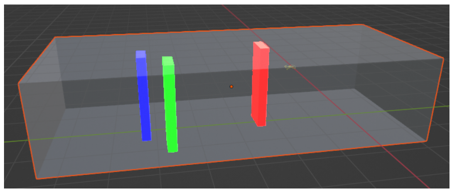


Fig. 2. World map with obstacles

III. Configuration Space

To maintain a safe distance between drone and obstacles during navigation, we inflate the size of obstacle by a certain factor. In our case with map1, we increased the size of obstacles by 10% (As shown in appendix). The DJI Tello drone uses `takeoff` command to launch itself vertically at an approximate distance of 1 m. Since the tello command is unpredictable we also check the distance it is at just after takeoff and adjust the height to ensure that quad-rotor is always at 1m before it begins tracing the given waypoints. We use this to our advantage for maintaining safety in navigation. To keep a safe ground clearance so that the drone does not touch the ground or top netting inside the lab, we bounded our Z values between 1 m and 1.5 m

IV. RRT* Path finding

Rapidly-exploring Random Trees Star (RRT*) is a path planning algorithm used in robotics and AI. It iteratively builds a tree of potential paths from a starting point to a goal while minimizing the overall path cost. RRT* efficiently explores the configuration space, incrementally improving the generated paths by reconfiguring the tree structure and selecting low-cost paths. For our application we sampled in euclidean space and used the L2 norm for cost calculation between sampled nodes. The nodes should be within 1.5 units of distance between each other and we decided to ignore all the obstacles that comes before the sampled points for optimization of collision checking. The collision checking algorithm utilizes intersection of line between the nodes with all of the 6 faces of the cuboidal obstacle defined which is faster than volumetric calculations.

The path which we found for above mentioned start and goal positions are shown in figures 3, 4.

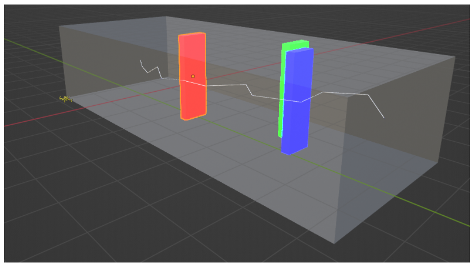


Fig. 3. RRT Path in blender

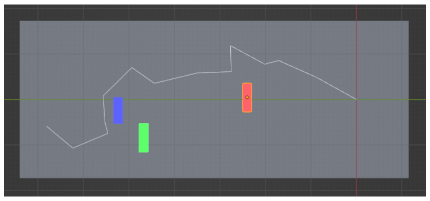


Fig. 4. RRT Path in blender

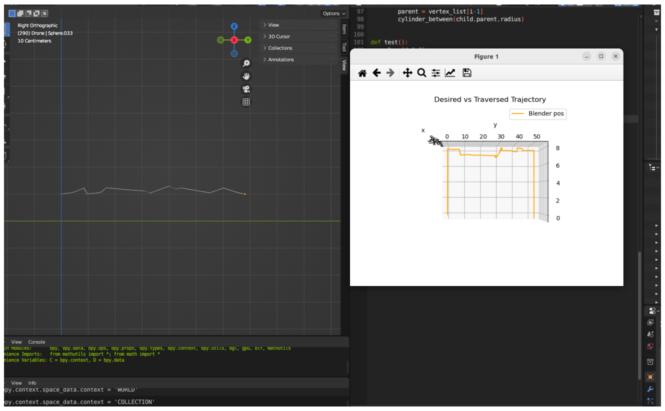
V. Physical Implementation

We implement all the path planning mentioned above on NVIDIA Jetson Orin Nano and use it to fly the drone. We faced some of the interesting problems while setting up both DJI Tello and Jetson Nano.

- The DJI Tello takes commands via UDP calls but since it works without handshake the commands gets lost in an unpredictable manner. A workaround to this was using the `send_command_with_return` function which forces tello to return the final status of execution of the sent command. This isn't a perfect solution either but it increases the reliability of the drone significantly.
- The logging on DJI Tello is imperfect as it returns the velocities in each of the principle axes but in $decimeters/sec$ and also cuts off any floating point precision from it meaning that small movements in any direction aren't logged in the Jetson Orin. This made plotting the actual path traversed by the drone will always have inaccuracies when using just the velocity. The solution to this was using the acceleration in each axes, which were given as float values, and use simple newtonian physics ($s = u.t + 0.5.a.t^2$) to get vastly better results.

VI. Data logging

We showcased during our live demo that we were able to follwo the RRT* generated trajjectory pretty closely using our DJI Tell0 with the help of `go x y z speed` command. We logged the states data from DJI Tello drone during the runtime on a parallel thread to ensure we're getting the real flight time data. Using the velocity and acceleration data, we reconstructed the trajectory travelled by the drone. We observed that we were able to get trajectory travelled by drone which follows similar trend as that of the RRT paths as shown below.



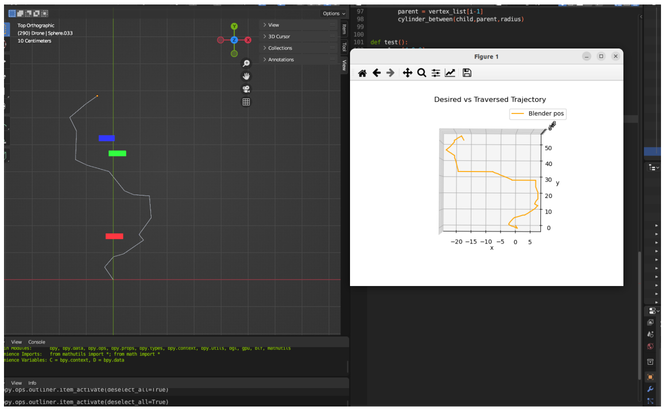
```
        parent = vertex_list[i-1]
        cylinder_between(child,parent,radius)


def test():
```

```
in Modules:      bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
nience Imports:   from mathutils import *; from math import *
nience Variables: C = bpy.context, D = bpy.data
```

```
py.context.space_data.context = 'WORLD'
py.context.space_data.context = 'COLLECTION'
```

Fig. 5. RRT Path vs logged data



```
        parent = vertex_list[i-1]
        cylinder_between(child,parent,radius)


def test():
```

```
in Modules:      bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
nience Imports:   from mathutils import *; from math import *
nience Variables: C = bpy.context, D = bpy.data
```

```
py.ops.outliner.item_activate(deselect_all=True)
py.ops.outliner.item_activate(deselect_all=True)
```

Fig. 6. RRT Path vs logged data

VII. Conclusion

After this experiment we are able to navigate a DJI Tello quad-rotor in an known and unseen space using the path generated from RRT* algorithm utilising just the on-board IMU and a simple PID controller. The quad-rotor successfully completed the flight without colliding with any obstacles or boundries and land within acceptable range of the dedicated landing point.

References

[1] A Quaternion-based Unscented Kalman Filter for Orientation Tracking
[2] Class Notes by Prof. Nitin Sanket

VIII. Appendix

A. *Obstacle Inflation for Configuration Space*

Get boundaries for all obstacles from map files provided
For each obtsacle:
$xlength = xmax - xmin$
$ylength = ymax - ymin$
$zlength = zmax - zmin$
$xmin = max(xmin_{world}, (xmin - 0.1 * xlength))$
$xmax = min(xmax_{world}, (xmax + 0.1 * xlength))$
$ymin = max(ymin_{world}, (ymin - 0.1 * ylength))$
$ymax = min(ymax_{world}, (ymax + 0.1 * ylength))$
$zmin = max(zmin_{world}, (zmin - 0.1 * zlength))$
$zmax = min(zmax_{world}, (zmax + 0.1 * zlength))$