Decoupled Kinodynamic Path Planning and Control using Informed RRT* for a Simulated and Real-World Quadrotor

Rutwik Kulkarni
Robotics Engineering
Worcester Polytechnic Institute
Email: rkulkarni1@wpi.edu

Kyle Mitchell
Robotics Engineering
Worcester Polytechnic Institute
Email: kgmitchell@wpi.edu

Ankit Mittal
Robotics Engineering
Worcester Polytechnic Institute
Email: amittal@wpi.edu

***Abstract*—This report presents a decoupled Kinodynamic Path Planning approach for quadcopters, initially employing the RRT* algorithm and then progressing to Informed RRT* for enhanced path optimization. Our method begins with RRT*, generating a kinematically feasible and collision-free path, focused on the drone's flat outputs (x, y, z, yaw). The strategy then transitions to Informed RRT*, further refining the path by focusing on promising regions of the search space. Following path generation, a series of polynomial segments are optimized to create a smooth, minimum-snap trajectory, which ensures seamless transitions and minimizes snap for mechanical viability. Utilizing the differential flatness of quadrotors, our approach achieves precise trajectory execution, considering the drone's dynamic constraints. Demonstrated through simulations(blender), and real-world hardware implementations, our implementation finds benefits for diverse drone applications.**

I. Introduction

The advancement of drone technology has opened new avenues in multiple industries, but it also presents significant challenges in motion planning within complex 3D spaces. In this project, we address these challenges with a decoupled kinodynamic approach, combining the strengths of the RRT* and Informed RRT* algorithms for initial path planning and augmenting it with advanced trajectory optimization techniques. Our methodology distinctly separates the kinematic aspects of path planning from the dynamic considerations of trajectory optimization, facilitating efficient and precise drone navigation from start to finish.

The process begins with the deployment of the RRT* algorithm, laying down the foundational kinematic path in a three-dimensional space. This initial path is crucial for outlining a feasible route through challenging environments. We then refine this path using the Informed RRT* algorithm, which narrows the focus to more promising segments of the search space, enhancing the path's efficiency and practicality.

Subsequently, we enter the dynamic phase of our approach, where we employ trajectory optimization. In this stage, polynomial trajectory segments are skillfully optimized to form a smooth, minimum-snap path. This optimization ensures that the transitions between waypoints are not only seamless but also dynamically sound, aligning with the drone's physical capabilities.

This report elaborates on each step of our approach, from the initial path planning using RRT* and Informed RRT*, through the trajectory optimization phase, to the final implementation and testing phases. Our methods are designed to push the boundaries of autonomous drone navigation in three-dimensional environments, proving to be highly effective in both simulation and real-world scenarios. The forthcoming sections will detail our methodology, its practical implementation, and the results obtained, demonstrating the robustness and

applicability of our integrated approach in diverse application scenarios.

II. Related Work

The field of kinodynamic motion planning for autonomous drones and robotics systems has predominantly been navigated through a two-step decoupled approach: geometric path planning followed by velocity planning. This section reviews the state of the art in this field, emphasizing the methods and challenges involved, and highlights our approach, which is influenced by [Richter 2016].

Geometric Path Planning: The initial stage in the decoupled approach, geometric path planning, focuses on developing a path that adheres to kinematic constraints like collision avoidance. This stage produces a quasi-static solution, suitable for slow or near-static movements. Techniques such as RRT* and Informed RRT* have significantly advanced path planning, particularly in complex 3D environments, by efficiently determining collision-free trajectories.

Velocity Planning Problem: Subsequently, the velocity planning stage involves the time parametrization of the geometric path, aligning it with the physical constraints of the specific system. The approach varies depending on the system, with manipulators often utilizing one-dimensional optimal control problems ([Bobrow 1985], [Shin 1985], [Shiller 1991]), and mobile robots employing various path smoothing techniques ([Fleury 1995], [Lamiraux 2001], [Lamiraux 1998], [Lau 2009]). In aerial robotics, our area of focus, decoupled approaches for trajectory planning have been effectively demonstrated in works like [Richter 2016], which forms the basis for our methodology.

Challenges in Decoupled Approaches: Decoupled approaches, while effective, present certain challenges. Judging the quality of paths is difficult since the optimal path may not be the shortest. Furthermore, these approaches might fail to find solutions in scenarios lacking quasi-static solutions, such as certain complex drone maneuvers.

Direct Planning and Computational Considerations: Direct planning, or the native kinodynamic approach, which entails planning directly in the control or state space, addresses scenarios where decoupled methods are insufficient. However, this approach is computationally expensive due to the high-dimensional nature of the state space and the complex dynamics involved. It requires significant computational resources, making it less practical for real-time applications.

In summary, the decoupled approach has been a mainstay in kinodynamic motion planning due to its efficiency and effectiveness in a range of scenarios. Nonetheless, its limitations, particularly in more dynamic or complex environments, necessitate continued innovation in this field. Our approach, inspired by [Richter 2016], seeks to address these challenges by merging refined path planning with advanced trajectory optimization techniques, providing a more holistic solution for autonomous drone navigation in 3D spaces.

III. Proposed Approach

A. Problem Formulation

In this study, we address drone kinodynamic motion planning in static 3D environments, aiming to navigate from a start location $\mathbf{x}_0$ to a goal location $\mathbf{x}_g$ without colliding with obstacles. The environment $\mathcal{E}$ is modeled in Blender, consisting of obstacles $\mathcal{O}_i \subset \mathcal{E}$ and free space $\mathcal{F} = \mathcal{E} \setminus \bigcup \mathcal{O}_i$. The drone's state is described by its position $\mathbf{x} = (x, y, z, \psi)$. A cascaded PID controller, implemented in Blender, controls the drone's trajectory, ensuring it follows a path from $\mathbf{x}_0$ to $\mathbf{x}_g$ while adhering to dynamic constraints. Path planning uses RRT* and Informed RRT* to generate waypoints, with trajectory optimization for position, velocity, and acceleration in x, y, z, and ψ. This approach, first validated in Blender's simulated environment, is adapted to real hardware, focusing on velocity control due to practical constraints in direct position control.

B. Environment Setup (Map Reader in Blender)

As part of the initial setup, the program needs to read environmental data from a text file. This text file should contain obstacle dimensions formatted in the following manner.

Boundary:$x_{min}\ y_{min}\ z_{min} x_{max}\ y_{max}\ z_{max}$

Block:$x_{min}\ y_{min}\ z_{min} x_{max}\ y_{max}\ z_{max}\ r\ g\ b$

Here $x_{min}\ \ y_{min}\ \ z_{min}$ represents the lower left corner coordinates of the block/boundary and $x_{max}\ \ y_{max}\ \ z_{max}$ represents the upper right coordinates of the block/ boundary. The script reads the environment and plots it in the blender. And in block $r\ g\ b$ represents the red, blue, and green values of the block for color coding.

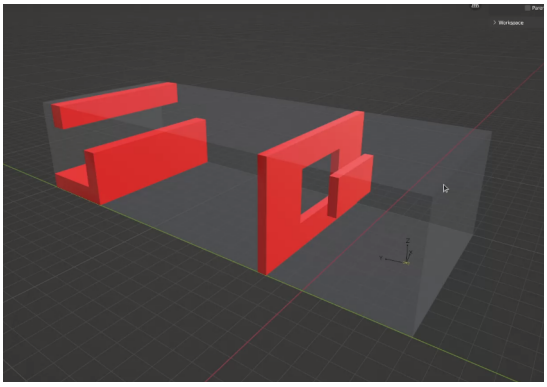

Fig. 1: Environment setup in Blender

IV. Waypoint Generation Using Geometric Path Planner

A. *RRT**

The RRT* planning algorithm is currently used for path planning from the starting position to the goal position. RRT* represents an enhanced and optimized iteration of the original RRT algorithm. In RRT, random points are generated and linked to the nearest accessible node. Before creating a vertex, a check is performed to ensure that it is positioned outside of any obstacles. Additionally, when connecting the vertex to its nearest neighbor, precautions are taken to avoid obstacles. The algorithm terminates either when a node is generated inside the desired goal region or when a predefined limit to the number of nodes is reached.
The basic principle of RRT* is the same as RRT, but two key additions to the algorithm result in significantly different results. In the RRT* algorithm, each vertex maintains a record of the distance it has covered relative to its parent vertex, which is denoted as its "cost." Once the nearest node within the graph is identified, the algorithm looks at a set of nearby vertices within a fixed radius around the newly created node. If a vertex with a lower cost than the closest proximal node is discovered, it replaces the proximal node. This feature has a noticeable impact on the tree structure, resulting in the emergence of fan-shaped branches and eliminating the cubic structure seen in the standard RRT algorithm. Another key enhancement introduced by RRT* is the concept of tree rewiring. Once a vertex has been linked to its most cost-effective neighbor, the algorithm goes on to reevaluate the neighboring vertices. It assesses whether rewiring a neighbor to the recently added vertex would result in a reduction in their cost. If such an improvement is observed, the algorithm proceeds to rewire that neighbor to the newly added vertex. This mechanism contributes to creating smoother and more efficient paths in the tree structure. Algorithm 1 is the pseudocode demonstrating RRT*. Figure 2 shows RRT*'s expanded tree, and Figure 3 shows the path generated using RRT*. The link for animation of RRT* tree expansion is given here [Link]

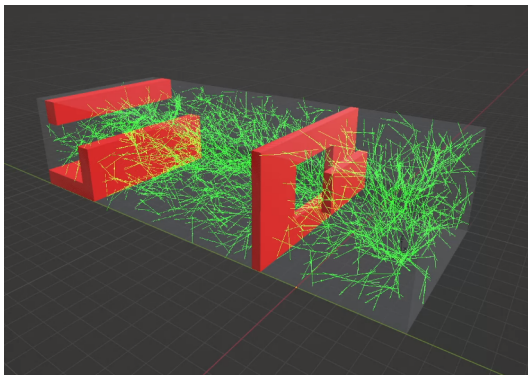

Fig. 2: Expanded RRT* Tree

B. *Informed RRT**

To improve the efficiency of our path planner, we upgraded the RRT* algorithm to Informed RRT*.

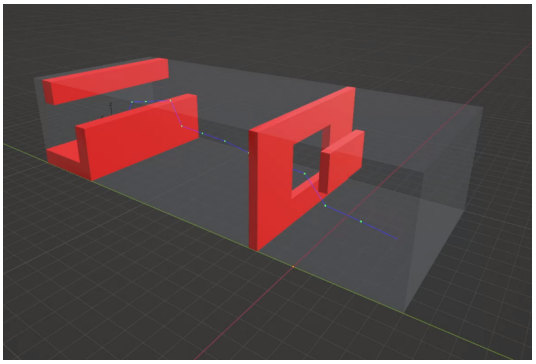

Fig. 3: Path Generated by RRT*

In Informed RRT*, once a path between the start node and goal node has been discovered, an ellipse is formed around these nodes, and the size of said ellipse is determined by the path length. The algorithm then continues to sample new nodes, but only within this ellipse, which acts as a heuristic to find a more optimal path. If node(s) that create a lesser cost are discovered, the path is rewired to use these better nodes. When this happens, the size of the ellipse shrinks, restricting the valid search area. This process continues until the maximum number of samples is reached (or a maximum valid path length is achieved). The Informed RRT* algorithm helps to better steer the sampled nodes to a region that is most likely to aid in creating a more optimal path. Algorithm 2 is the pseudocode demonstrating Informed RRT*. Informed RRT* gives us a set of waypoints from the start position to the goal position along the found path. Then, the waypoints are passed through our trajectory generator for path smoothing. Figure 4 shows Informed RRT*'s expanded tree, and Figure 5 shows the path generated using Informed RRT*. Link for the animation of tree expansion can be found here Link

Informed RRT* is either as efficient or more efficient than RRT*. In the best case scenario, it limits the search region to a small and effective region (the ellipsoid between the two points whose size is determined by the path's length) to develop a shorter and more optimal path between the start point and goal point. In the worst case scenario,

Algorithm 1 RRT* Algorithm

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1$ to n **do**
  $x_{\text{rand}} \leftarrow \text{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if ObstacleFree(x_{nearest}, x_{new}) **then**
    $X_{near} \leftarrow Near(G = (V, E), x_{\text{new}}, \min(\gamma(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta))$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
    $c_{\text{min}} \leftarrow \text{Cost}(x_{\text{nearest}}) + c(\text{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    for each $x_{\text{near}} \in X_{\text{near}}$ **do**
      if CollisionFree(x_{near}, x_{new}) $\wedge$ Cost(x_{near}) + $c(\text{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
        $c_{\text{min}} \leftarrow \text{Cost}(x_{\text{nearest}}) + c(\text{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
      end if
      $E \leftarrow E \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
      for each $x_{\text{near}} \in X_{\text{near}}$ **do** ▷ Rewire
        if CollisionFree(x_{near}, x_{new}) $\wedge$ Cost(x_{new}) + $c(\text{Line}(x_{\text{near}}, x_{\text{new}})) < \text{Cost}(x_{near})$ **then**
          $x_{parent} \leftarrow \text{Parent}(x_{near})$;
          $E \leftarrow E \setminus \{(x_{\text{parent}}, x_{\text{near}})\} \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
        end if
      end for
    end for
  end if
end for
return $G = (V, E)$

it behaves exactly like RRT*, thus making it as efficient as RRT*. Therefore, there is no downside to using Informed RRT* in terms of the final path output. However, Informed RRT* is more computationally expensive than RRT*, so by using Informed RRT* over RRT*, you are sacrificing performance. This may not matter, depending on the application. If you are performing the path planning ahead of execution, the performance hit should not affect the robot too much. However, if the robot is doing

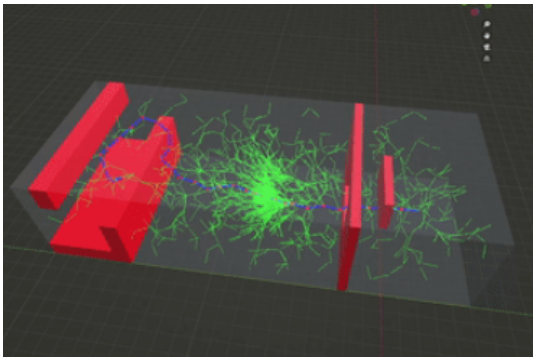

Fig. 4: Expanded Informed RRT* Tree

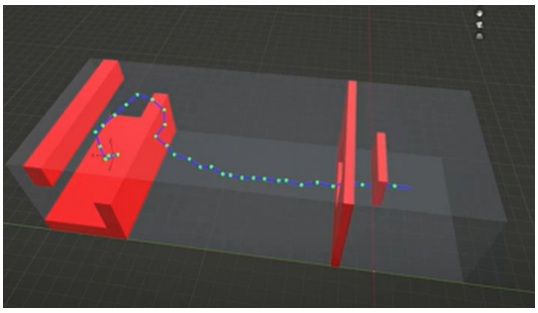

Fig. 5: Path Generated by Informed RRT*

on board path planning while it is in motion, the extra computation time may prove to be an issue. Because our robot was doing path planning ahead of time before execution, using Informed RRT* and its more expensive computation did not inhibit us.

V. Trajectory Generation and Optimization

Our trajectory generation approach centers on crafting smooth paths through a series of waypoints, balancing the need for continuous motion in all spatial dimensions $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and orientation (ψ). This balance is achieved by minimizing the snap, the fourth derivative of position, to ensure fluid and dynamically feasible movement for a drone navigating through a static environment.

*1) **Polynomial Representation**:* The trajectory between waypoints is defined using high-order polynomials, specifically 10th order, allowing precise control over the drone's motion. For each segment of the trajectory, we define the polynomial as:

$$q(t) = \sum_{i=0}^{10} a_i t^i$$

where $q(t)$ represents the position at time t, and $a_0, a_1, a_2..., a_{10}$ are the polynomial coefficients.

*2) **Optimization of Segment Times**:* The cost function for optimizing the segment times in trajectory generation is designed to minimize the snap, or the fourth derivative of position, while considering the total trajectory time. It consists of two main components: **Integral of the Squared Snap**: This part focuses on the smoothness of the trajectory and is represented as:

$$\text{Snap Cost} = \int (\frac{d^4 q}{dt^4})^2 \, dt \tag{1}$$

This integral sums the squared snap over each segment of the trajectory to quantify overall smoothness. **Total Trajectory Time**: This component accounts for the time efficiency of the trajectory. It is weighted by a factor γ to balance speed and smoothness:

$$\text{Time Cost} = \gamma \times T_{\text{total}} \tag{2}$$

Algorithm 2 Informed RRT* Algorithm

Require: M, x_{start}, x_{goal}
  $T.init(x_{start})$
  while $||x_{new} - x_{goal}||_2 \geq \epsilon$ **do**
    $x_{\text{rand}} \leftarrow$ Sample(M)
    $x_{\text{nearest}} \leftarrow$ Near(x_{rand}, T)
    $x_{\text{new}} \leftarrow$ Steer$(x_{rand}, x_{near}, StepSize)$
    if $CollisionFree(x_{new})$ **then**
      $X_{\text{near}} \leftarrow NearC(T, x_{\text{new}})$
      $x_m \leftarrow ChooseParent(X_{\text{near}}, x_{\text{near}}, x_{\text{new}})$
    end if
  end while
  $T.addNodeEdge(X_{min}, x_{new})$
  $T.rewire()$

where γ is a weighting factor emphasizing the importance of time duration in the cost function. Combining these components, the overall cost function for the optimization is:

$$\text{Cost} = \int \left(\frac{d^4q}{dt^4}\right)^2 dt + \gamma \times T_{\text{total}} \tag{3}$$

This cost function is minimized using an optimization algorithm to adjust the segment times, balancing between a smooth trajectory (minimized snap) and efficient completion (minimized total time).

*3) **Constraint Satisfaction**:* To ensure the trajectory is practical and smooth, several constraints are imposed:

- **Waypoint Constraints:** The trajectory must pass through each waypoint.
- **Continuity Constraints:** We enforce continuity in position, velocity, acceleration, and jerk at each waypoint.
- **Boundary Conditions**: The drone is expected to start and end each segment in a rest state, implying zero initial and final velocities and accelerations.

*4) **Yaw Control**:* In addition to spatial positioning, trajectory planning also involves controlling the drone's yaw. The yaw angle is adjusted to align with the direction of travel, calculated based on the velocity vector, enhancing the drone's aerodynamic efficiency and control stability.

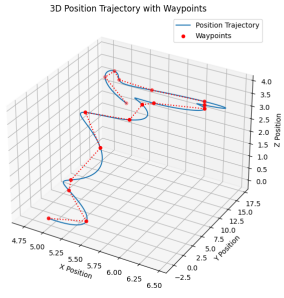

Fig. 6: Position Trajectory in 3D]

*5) **Implementation Steps**:*

- Define the initial and final conditions for each segment based on the waypoints.
- Use the boundary conditions to solve for the polynomial coefficients, ensuring that each segment of the trajectory satisfies these conditions.
- Optimize the time allocation for each segment to minimize the overall trajectory cost.
- Calculate the desired position, velocity, acceleration, and yaw at any given time using the polynomial equations.
- Ensure smooth transitions between segments by matching the end state of one segment with the start state of the next.

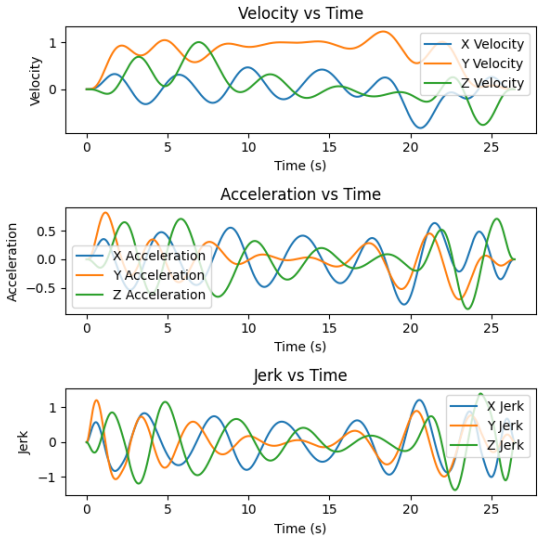

Fig. 7: Trajectory Generation and Optimization for Waypoints Obtained from Informed RRT*[Velocity, Acceleration and Jerk Trajectory]

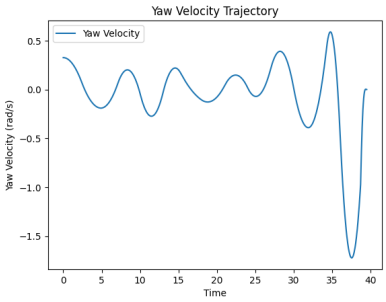

Fig. 8: Trajectory Generation and Optimization for Waypoints Obtained from Informed RRT*[Position Trajectory in 3D]

In summary, our approach provides a framework for generating smooth and dynamically feasible trajectories for drones. By employing high-order polynomials and optimizing both the trajectory shape and segment times, we achieve a high degree of control over the drone's motion, crucial for navigating complex environments.

VI. Controller Design and Tuning

A. Cascaded PID Controller

We implemented a cascaded PID (Proportional-Integral-Derivative) controller for the quadrotor, which is a popular control architecture for stabilizing and controlling the flight of drones. This cascaded design consists of multiple PID controllers stacked in a hierarchical manner.

Position Controller: The outermost layer of the cascaded controller focuses on position control in NED (North-East-Down) coordinates. It uses three PID gains to regulate the quadrotors ' positions along X, Y and Z direction. This controller takes the desired position waypoints from our trajectory and the current position state of our drone as input and computes the desired velocity setpoints to reach the desired position.

Velocity Controller: The next layer of the controller focuses on velocity control in NED coordinates. It adjusts the quadrotor's velocity based on desired velocity setpoints. Three additional PID gains are used to control the quadrotor's velocity along the X, Y, and Z axes. These controllers take the desired velocity setpoints and the current velocity as input and calculate acceleration setpoints to achieve the desired velocity.

Angular Rate Controller: The innermost layer of the controller deals with controlling the quadrotor's angular rates. It uses three PID gains to regulate the quadrotor's roll, pitch, and yaw rates. These controllers take the desired angular rates and the current angular rates as input and compute torque commands to achieve the desired rates.

B. PID Gain Tuning Methodology

The PID gain tuning for the drone's cascaded control system was conducted sequentially, starting from the innermost layers to the outer layers. Initially, the angular rate controller was tuned, providing a stable and responsive flight foundation. This was followed by the tuning of the velocity controller, beginning with the Z-axis for vertical control and then proceeding to the X and Y-axes for horizontal motion. Finally, the position controller was fine-tuned, starting again with the Z-axis and subsequently, the X and Y axes, ensuring precise waypoint navigation and position maintenance.

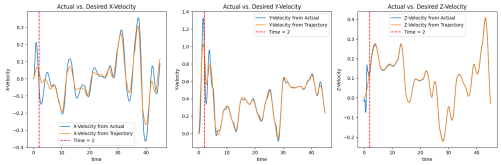

Fig. 9: Velocity Profile of Controller Executing Sample Trajectory in Environment(Blender)

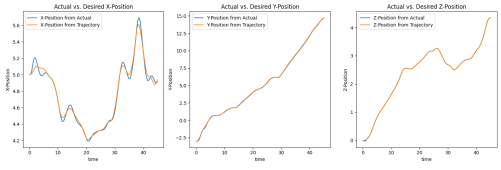

Fig. 10: Position Profile of Controller Executing Sample Trajectory in Environment(Blender)

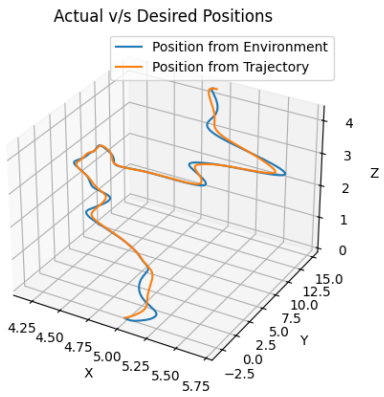

Fig. 11: Position Profile from Path Planner v/s Position Profile executed in Blender Environment

VII. Platform and Evaluation

A. Real-World Testing with DJI Tello Drone

Our path planning system was tested in a real environment using the DJI Tello Drone. The performance data, represented in the attached graphs, reflects the drone's adherence to the planned trajectory and velocity profiles. **here**.

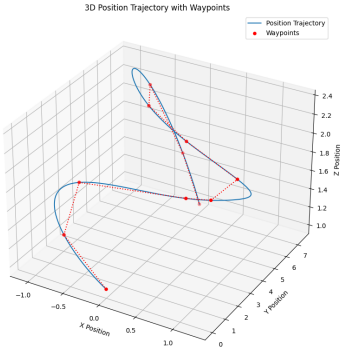

Fig. 12: Plotting 3D Position Trajectory from Real World Test

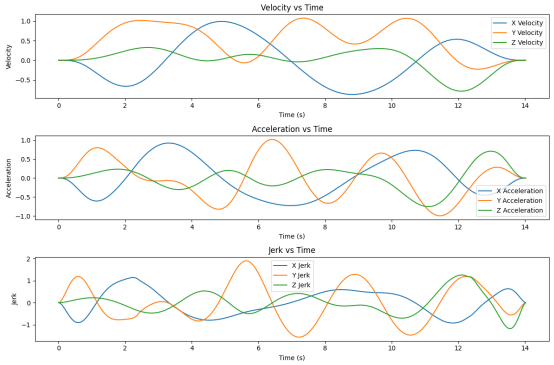

Fig. 13: Velocity, Acceleration and Jerk Plots for Real World Test

B. Video Documentation and Observations

The field testing was documented in a video **[Link to Video]**, showcasing the drone's navigation and obstacle avoidance. Despite hardware limitations, such as a maximum velocity of 1 m/s and inaccessible lower-level controls, the drone followed the path with minor deviations due to hardware latency. A notable observation was an excessive deviation to the right when avoiding the second obstacle, indicating a limitation in navigating tightly cluttered spaces. However, the drone successfully reached its goal, demonstrating the practical viability of our approach.

These real-world tests highlight the potential and limitations of our system, suggesting areas for future refinement, particularly in enhancing path optimization for cluttered environments.

VIII. Limitation of the Approach and Future Work

While the decoupled kinodynamic planning approach adopted in this project has demonstrated effectiveness in scenarios with limited computational resources and environments with sparse obstacles, it exhibits significant limitations in precision navigation, particularly in environments with intricate gaps and dynamic obstacles. These limitations highlight the necessity for a more advanced native kinodynamic planning approach, such as LQR-RRT (Linear Quadratic Regulator - Rapidly-exploring Random Tree).

A. Challenges in Precision Navigation and Dynamic Environments

1) **Limited Maneuverability in Tight Spaces:** The decoupled approach, separating geometric path planning and dynamic trajectory generation, often fails to fully account for the drone's dynamic capabilities. This is especially true in environments with narrow passageways or small gaps, where precise planning and execution are critical.
2) **Inadequate Response to Dynamic Obstacles:** In environments characterized by moving obstacles or rapidly changing conditions, the decoupled approach may not provide the required agility and responsiveness, leading to increased collision risks and suboptimal navigation.

B. The Case for LQR-RRT in Complex Environments

1) **Enhanced Precision with Integrated Dynamics:** LQR-RRT, a native kinodynamic planning approach, integrates the vehicle's dynamics directly into the path planning process. This approach is particularly advantageous in high-precision maneuvering environments, such as tightly packed indoor spaces or complex urban landscapes.
2) **Real-World Application - Indoor Flight:** Consider a drone navigating a cluttered warehouse with narrow aisles and varying heights. LQR-RRT dynamically adapts the drone's flight path in real-time, considering its maneuverability constraints, thus ensuring safer and more efficient navigation.
3) **Adaptability to Dynamic Environments:** Unlike the decoupled approach, LQR-RRT can promptly adjust the drone's trajectory in response to environmental changes, ensuring continuous adherence to dynamic constraints and operational safety.

C. Conclusion

In summary, while the decoupled kinodynamic planning approach is computationally efficient and suitable for less complex environments, its limitations in precision and adaptability are evident in more challenging scenarios. Advanced native kinodynamic planning approaches like LQR-RRT are necessary in these scenarios to enhance navigation precision and ensure safety and efficiency in dynamic and complex environments.

IX. Contributions

Rutwik Kulkarni: Trajectory Generation, Controller Implementation, Hardware Implementation, writing/editing of the report.

Kyle Mitchell: Preliminary research and testing for kinodynamic planning (project pivoted after this work), research for Informed RRT*, and work on test implementations before incorporating them into drone simulation in Blender, partial implementation of Informed RRT* into drone simulation, and writing/editing of the report.

Ankit Mittal: Implementation of RRT* algorithm in the 3D environment, visualization/simulation in Blender, writing/editing of the report

References

[1] Karaman, Sertac, and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." The International Journal of Robotics Research 30, no. 7 (2011): 846-894. **Link**

[2] Blender Plot API. **Link**

[3] Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments Charles Richter, Adam Bry, and Nicholas Roy **Link**