UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria dell'Automazione e Robotica

Field and Service Robotics

Obstacle Avoidance Algorithms for UAV

Academic Year 2024/2025

Supervisor
Prof. Fabio Ruggiero

Candidates
Fiorella Maria Romano P38000265
Claudia Panza P38000266

Contents

Chapter 1

Introduction and Objectives

1.1 Background and Motivation

Unmanned Aerial Vehicles (UAVs) have become indispensable platforms in both field and service robotics applications, ranging from environmental monitoring to infrastructure inspection. Quadrotor UAVs, in particular, offer vertical takeoff and landing, high maneuverability, and the ability to hover, making them ideal for operations in cluttered environments. However, to operate safely and autonomously, quadrotors require robust motion-planning and obstacle-avoidance strategies coupled with precise control loops. This report documents the development, implementation, and evaluation of two UAV navigation algorithms—precomputed Rapidly-exploring Random Tree (RRT) planning and real-time obstacle avoidance—in a simulated 3D environment..
The main objectives of this work are:

- To construct a realistic 3D environment with static obstacles (buildings and trees) and generate an occupancy map.
- To implement and compare two navigation approaches: RRT-based path planning with Dubins motion primitives and real-time obstacle avoidance using MATLAB's UAV Toolbox.
- To integrate a cascaded PID control architecture (position, velocity, attitude, and yaw control) for trajectory tracking.
- To quantitatively compare trajectory-following accuracy and control effort between the two approaches.

Chapter 2

Environment Setup

2.1 Scenario Creation and UAV Model

The simulation environment is instantiated in MATLAB using the `uavScenario` interface (UpdateRate=100Hz) and a North–East–Down (NED) reference frame. A quadrotor platform with mass 0.1 kg is placed at the initial pose $[0,\ 0,\ -7]$m with zero roll–pitch–yaw. For obstacle sensing, a 3D LiDAR is simulated via `uavLidarPointCloudGenerato` with the following parameters:

- `MaxRange` = 30m
- `AzimuthResolution` = 0.5°
- `ElevationResolution` = 2°
- `RangeAccuracy` = 3m
- `AzimuthLimits` = $[-179,\ 179]^\circ$
- `ElevationLimits` = $[-25,\ 25]^\circ$

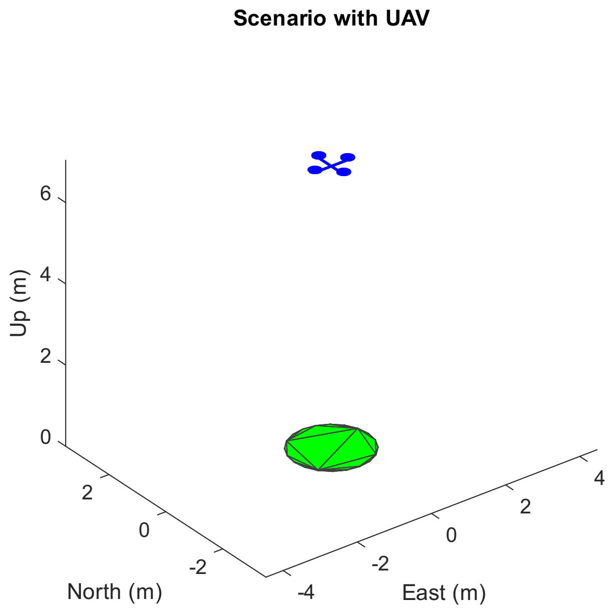

Figure 2.1: Visualization of the UAV platform within the simulated scenario.

2.2 Map Construction

Static obstacles are modeled directly into the scenario via `addMesh`:

- **Buildings:** Rectangular prisms defined by center coordinates, width, depth, and height.
- **Trees:** Vertical cylinders approximated as extruded polygons with base radius r and height h.
- **Points of Interest:** Small circular "pillars" for additional environmental clutter.

The complete 3D scene, including the UAV and all obstacles, is rendered using `show3D`.

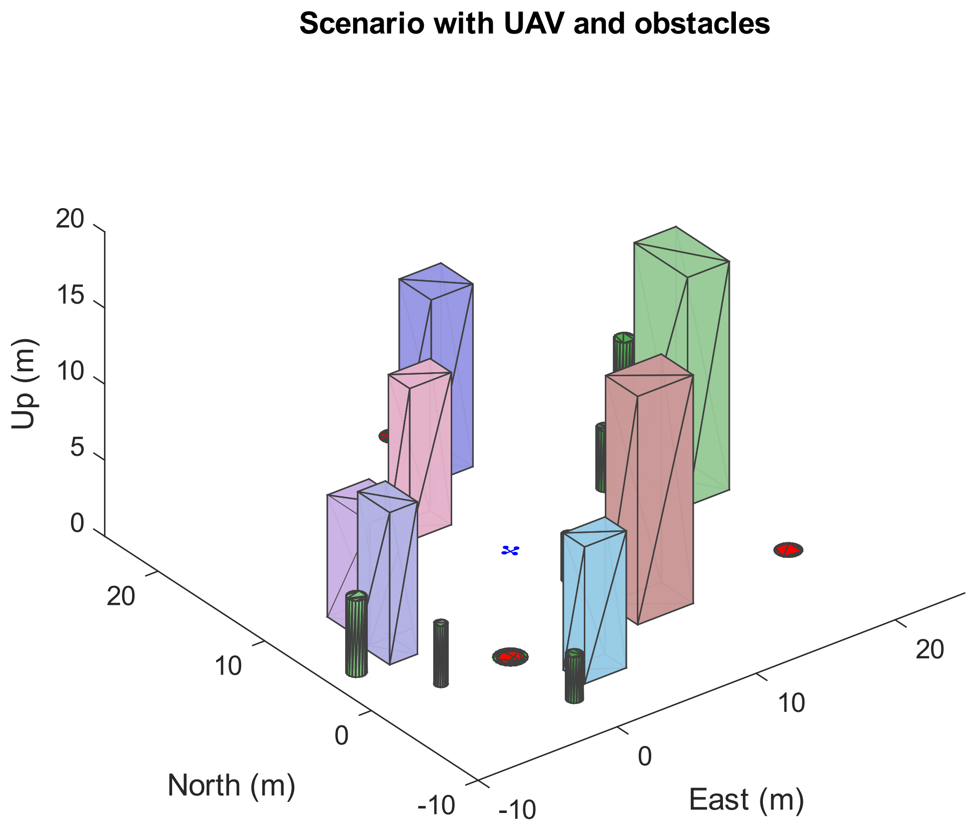

Figure 2.2: Simulated environment with buildings, trees and points of interest.

We define three mission waypoints:

$$\text{Waypoints} = \begin{bmatrix} 0 & 0 & 7 \\ 10 & 24 & 14 \\ 20 & 0 & 12 \end{bmatrix},$$

which the UAV must sequentially visit in 3D space.

2.3 Occupancy Map

An `occupancyMap3D` object is created with voxel size 0.5m, balancing spatial detail against computational load. Smaller voxels capture finer obstacle details but increase memory usage and update time. All occupied points—sampled densely within building volumes, tree trunks, and the ground plane—are marked via:

```
setOccupancy(map3D, allPoints, 1).
```

The map's free-space threshold is synchronized to the occupied threshold (`FreeThreshold` = `OccupiedThreshold` = 0.5) so that any voxel with occupancy above 0.5 is treated as an obstacle. The finalized 3D grid is displayed with `show(map3D)`.

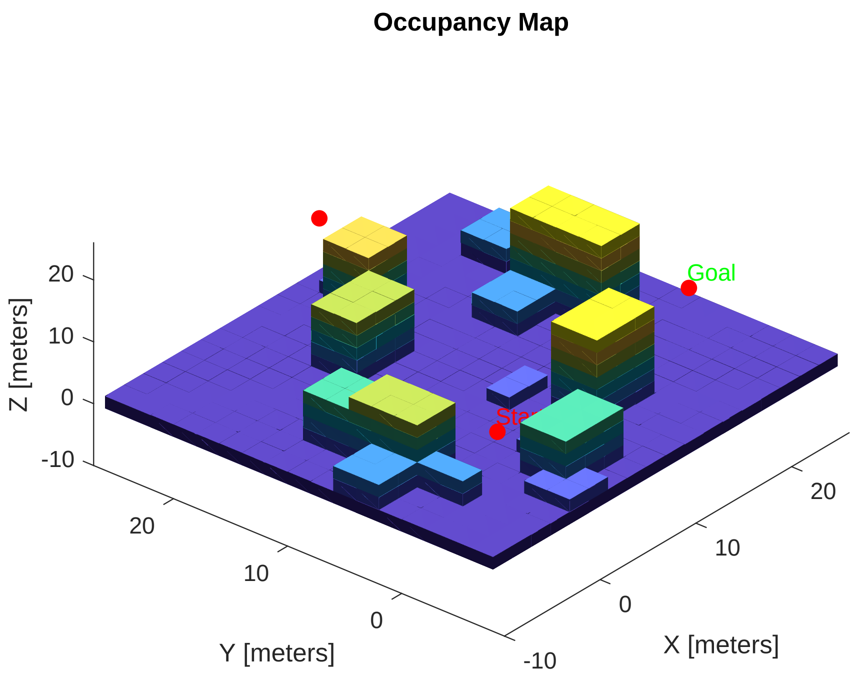

Figure 2.3: 3D occupancy grid with overlaid mission waypoints.

Chapter 3

Motion-Planning and Obstacle-Avoidance Algorithms

3.1 Motion Planning with RRT

Rapidly-exploring Random Trees (RRT) grow a collision-free search tree by sampling random states and connecting them toward the nearest node, with a goal-bias probability to improve convergence. An RRT-based motion planner is employed, using a state space with constraints resembling those of Dubins paths — namely constant forward speed, limited roll angle, and bounded flight path angle. These constraints are typical of fixed-wing UAVs, but in this project they are applied to a quadrotor, which is capable of omnidirectional flight and hovering.

However, the planned trajectories are not true Dubins curves. Instead, they are composed of discrete segments from the RRT planner, which are then smoothed and interpolated using MATLAB's navPath object, producing continuous and dynamically feasible paths for the quadrotor.

Thus, although the motion planning is inspired by forward-flight constraints, the final trajectory is fully compatible with multirotor UAV operation through interpolation and appropriate control.

Controller parameters were tuned empirically based on trial-and-error, with the goal of achieving smooth and stable flight dynamics across the entire scenario. This modular Simulink architecture facilitates the integration of real-time sensing, perception, and control logic into a unified autonomous UAV system.

The Simulink workflow is the following:

3.1.1 Scenario and Environment Subsystem

This subsystem is responsible for simulating the 3D UAV environment and visualizing its interaction with the surroundings. It includes the following blocks:

- **UAV Scenario Configuration** — Initializes and configures the UAV scenario, including the placement of static obstacles and sensor properties.
- **UAV Scenario Motion Read** — Reads the current UAV state (position and orientation) from the scenario at each simulation step.
- **UAV Scenario Lidar** — Simulates a forward-facing LiDAR sensor that generates a 3D point cloud of the environment.

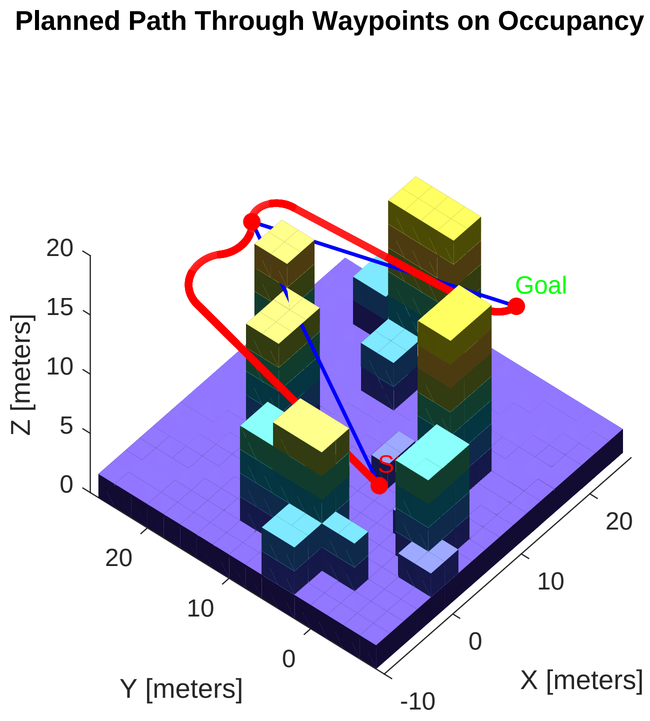

Figure 3.1: Planned RRT trajectory avoiding obstacles in the occupancy map.

- **UAV Scenario Motion Write** — Updates the UAV state in the simulation based on control inputs.
- **UAV Scenario Scope** — Visualizes the UAV trajectory and the real-time LiDAR data in a 3D scene.

3.1.2 UAV State Space and Collision Validator

To generate dynamically feasible trajectories, a reduced-order UAV state space is defined using the `ExampleHelperUAVStateSpace` object. This space captures the essential configuration variables and flight constraints of a fixed-wing UAV:

- **State Variables:** $[x, y, z, \psi]$ — position in 3D space and heading angle.
- **MaxRollAngle** $= \pi/4$ rad — limits the maximum roll angle to ensure realistic turns.
- **AirSpeed** $= 5$ m/s — assumes a constant forward velocity.
- **FlightPathAngleLimit** $= [-0.2, 0.2]$ rad — bounds the climb and descent angles.
- **Bounds:** $x, y \in [-5, 100]$ m; $z \in [0, 20]$ m; $\psi \in [-\pi, \pi]$ rad — defines the operating region.

A collision validator is created using `validatorOccupancyMap3D`, which links the UAV state space to the 3D occupancy map (`map3D`). The validator checks the validity of each path segment during planning, rejecting those that intersect occupied regions. The parameter:

- `ValidationDistance` = 0.5 m — specifies the step size used to discretize path segments for collision checking.

This ensures that planned trajectories maintain a safe clearance from obstacles present in the map.

3.1.3 Path Generation

The Rapidly-Exploring Random Tree (RRT) algorithm is a sampling-based motion planning method designed to efficiently explore high-dimensional configuration spaces. It is particularly suitable for planning paths in environments with complex dynamics or obstacles, such as UAVs navigating in 3D space.

RRT incrementally builds a search tree rooted at the initial configuration by randomly sampling the space and connecting new nodes to the nearest point in the tree. The algorithm prioritizes rapid coverage of the free space without requiring a precomputed representation of it.

The algorithm steps are the following:

1. `Initialization:` Start with the initial state as the root of the tree.
2. `Sampling:` Randomly sample a state in the configuration space.
3. `Nearest Node:` Find the nearest node in the existing tree to the sampled point.
4. `Extension:` Extend the tree from the nearest node toward the sampled point by a fixed step size, subject to dynamics constraints.
5. `Collision Checking:` Validate the new edge to ensure it lies in free space and does not intersect any obstacle.
6. `Add Node:` If the path is valid, add the new node and edge to the tree.
7. `Goal Test:` If the new node is within a given tolerance of the goal state, the path is considered found.

The following planner parameters are used in the simulation:

- `MaxConnectionDistance` — Maximum step size for each tree expansion. In the script this parameter is set to 50.
- `GoalBias` — Probability of sampling the goal state directly to guide the search. In the script this parameter is set to 0.10.
- `ValidationDistance` — Resolution used for collision checking between nodes. In the script this parameter is set to 0.5.
- `MaxIterations` — Limit on the number of iterations to prevent infinite loops. In the script this parameter is set to 1000.
- `GoalReachedFcn` — A custom stopping criterion defined as $\|x - y\| < 3$ meters.

Advantages: The RRT algorithm is efficient in high-dimensional or constrained environments, simple to implement and extend, and capable of handling non-holonomic and dynamic systems.

Limitations: The resulting path is not guaranteed to be optimal. The trajectory may be jagged or non-smooth, often requiring post-processing (e.g., interpolation). Furthermore, performance is sensitive to parameter tuning such as connection distance and goal bias.

3.1.4 Controller and Plant Subsystem

This subsystem governs the UAV's dynamic behavior by computing and applying control inputs based on the desired trajectory. It includes:

- **Controller** — Implements a cascaded PID control architecture to compute the control commands: roll, pitch, yaw, and thrust. The control structure ensures accurate position and attitude tracking.
- **Quadrotor Plant** — A dynamic model of the UAV that simulates the quadrotor's response to control inputs, including inertia, gravity, and actuation limits.
- **Conversion** — Extracts relevant state variables such as position and orientation from the UAV model, and performs data formatting and coordinate transformations for feedback and visualization.

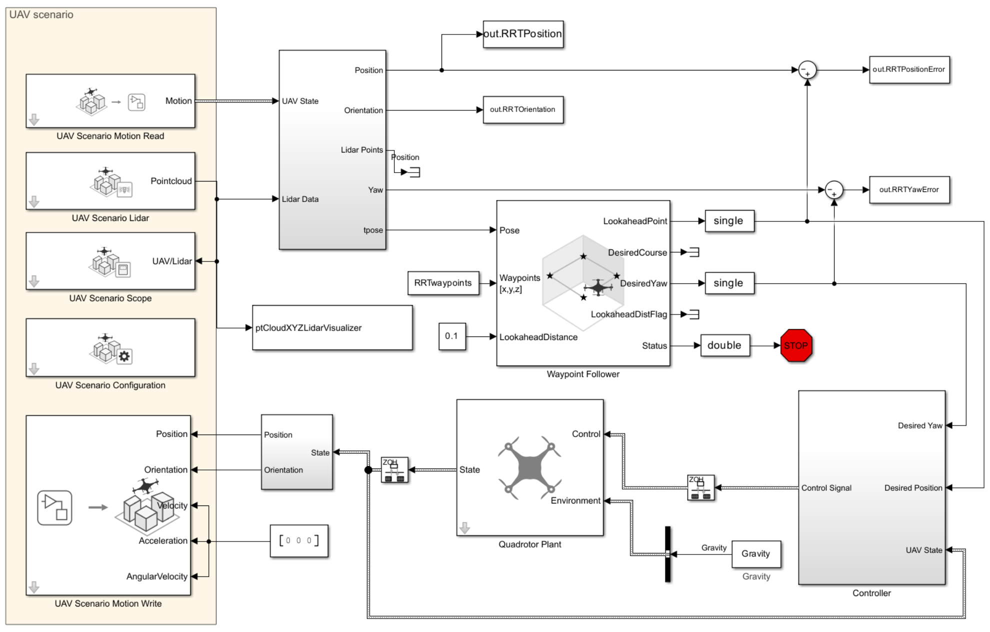

Figure 3.2: Simulink subsystem for following the precomputed RRT trajectory.

This architecture cleanly separates offline path planning from real-time trajectory tracking, facilitating modular extension to online planners or reactive obstacle avoidance.

3.2 UAV Obstacle Avoidance

The obstacle avoidance strategy is implemented in Simulink through a modular architecture composed of three main subsystems: *Scenario and Environment*, *Waypoint Following with Obstacle Avoidance*, and *Controller and Plant.*

As the first and third blocks are the same as in the previous scheme, we are going to focus on the second one.

3.2.1 Waypoint Following and Obstacle Avoidance Subsystem

This subsystem is responsible for generating a safe navigation reference by combining waypoint following with real-time obstacle avoidance. It consists of:

- **Waypoint Follower** — Computes a lookahead point in the direction of the next waypoint based on the current UAV pose.
- **Obstacle Avoidance** — Implements the 3D Vector Field Histogram Plus (VFH+) algorithm to compute a collision-free heading and yaw by analyzing the point cloud from the LiDAR. The output modifies the original lookahead point to ensure safety.
- **Conversion** — Regulates how frequently the obstacle avoidance is executed and manages necessary transformations and data conversions.
- **Lookahead Distance** — A constant block that determines the preview horizon used to compute the lookahead position. A unit vector in the desired direction is scaled by this value and added to the UAV position.
- **Waypoints** — Contains the predefined sequence of 3D waypoints that the UAV must follow throughout the mission.

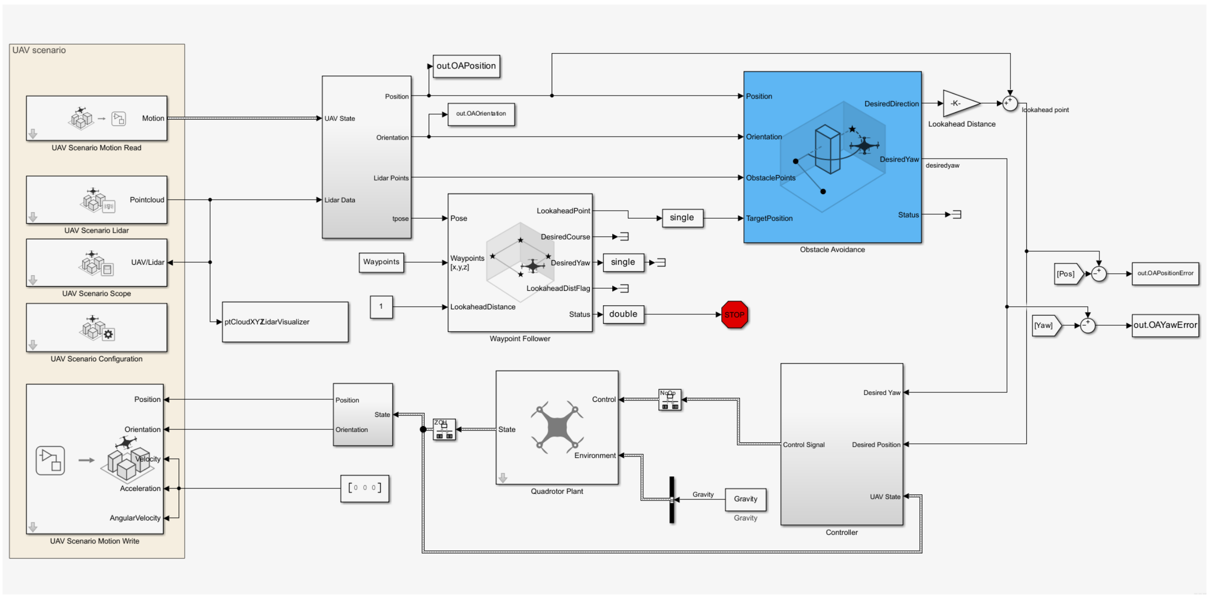

Figure 3.3: Simulink diagram for real-time obstacle avoidance using 3D VFH+.

This reactive architecture ensures that, even in the presence of unexpected obstacles, the UAV dynamically adjusts its path without the need for full repartitioning of the global plan, trading off some smoothness for robustness and computational efficiency.

Chapter 4

Control Architecture

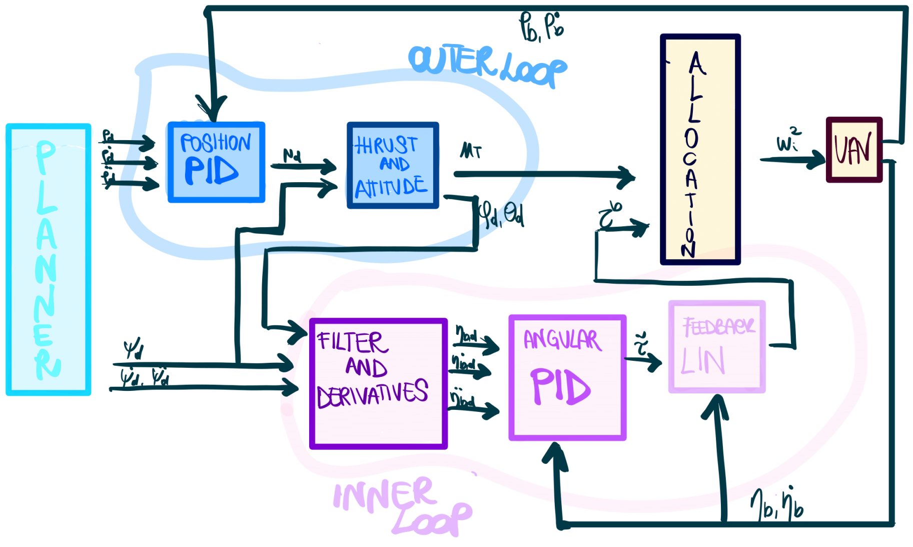

Figure 4.1: Theoretical Hierarchical Controller

4.1 Hierarchical PID Control

In this project, a multistage PID control architecture was implemented to regulate the flight of a quadrotor UAV in both trajectory-following and real-time obstacle avoidance scenarios.

Although more advanced control strategies (e.g., LQR, MPC, or geometric control on SE(3)) exist for UAVs, a cascaded PID architecture was adopted in this work for its simplicity, ease of tuning, and compatibility with MATLAB/Simulink tools. This choice is particularly effective in simulation scenarios where disturbances and model mismatches are limited.

The design follows a cascade control logic, dividing the control effort into hierarchical layers: position control, velocity control, and attitude (orientation) control. This modularity allows for better tunability, improved stability, and clearer separation of dynamic responsibilities. The goal of the controller is to ensure that the UAV follows a time-

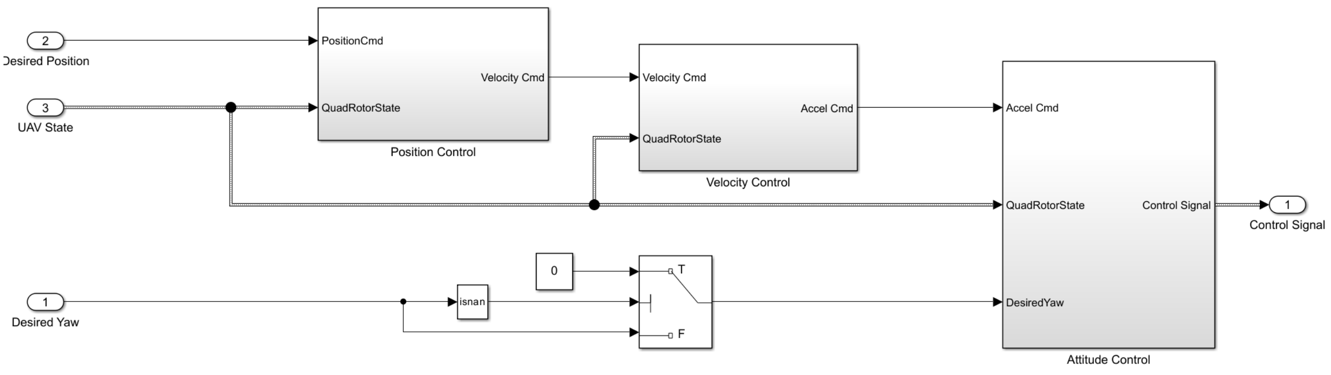

Figure 4.2: Simulink diagram for the Controller

varying reference position and yaw angle, which are generated either by a precomputed RRT path or by an online planner algorithm. At each simulation step, the UAV receives a desired 3D position $\mathbf{p}_d = [x_d, y_d, z_d]^T$ and a desired yaw angle ψ_d. The UAV's actual state is measured and compared against these targets, and the control system computes the necessary roll, pitch, yaw rate, and thrust commands.

The control structure is implemented in three main cascaded blocks:

- **Position Control:** Translates position error into desired velocity.
- **Velocity Control:** Translates velocity error into desired acceleration.
- **Attitude Control:** Converts desired accelerations into orientation commands (roll, pitch) and computes the thrust required to compensate for gravity and z-acceleration.

4.2 Position Control

The outermost control loop computes the velocity setpoints based on the error between the desired position and the current UAV position:

$$\mathbf{e}_p = \mathbf{p}_d - \mathbf{p}$$

$$\mathbf{v}_d = K_P \cdot \mathbf{e}_p + K_D \cdot \frac{d\mathbf{e}_p}{dt} + K_I \int \mathbf{e}_p \, dt$$

where:

- $K_P = [P_x, P_y, P_z]$
- $K_D = [D_x, D_y, D_z]$
- $K_I = [I_x, I_y, I_z]$

In this project, the gains were empirically tuned as:

$$P_x = P_y = 3, \quad P_z = 4$$

$$D_x = D_y = 0.001 D_z = 0.002$$
$$I_x = I_y = I_z = 0$$

for OA Algorithm, while for RRT are:

$$P_x = P_y = 5, \quad P_z = 6$$

$$D_x = D_y = 0.001 D_z = 0.002$$
$$I_x = I_y = I_z = 0$$

The filter on the derivative action is:

$$N_x = N_y = 1000, N_z = 15$$

This PID controller outputs the desired velocity vector $\mathbf{v}_d$, which feeds the next control layer.

4.3 Velocity Control Loop

The velocity control block compares the desired velocity $\mathbf{v}_d$ with the actual UAV velocity $\mathbf{v}$ and computes the desired acceleration:

$$\mathbf{e}_v = \mathbf{v}_d - \mathbf{v}$$

$$\mathbf{a}_d = K'_P \cdot \mathbf{e}_v + K'_D \cdot \frac{d\mathbf{e}_v}{dt}$$

The chosen gains are:

$$K'_P = [0.5, 0.51]$$
$$K'_D = [0, 0, 0]$$

for both OA and RRT algorithms.

In practice, the same proportional and derivative gains are reused to match the system dynamics and reduce the complexity of tuning. The resulting vector $\mathbf{a}_d$ (desired acceleration in the inertial frame) is then passed to the final block for translation into attitude commands.

4.4 Attitude Control and Thrust Computation

The attitude controller translates the desired acceleration vector $\mathbf{a}_d$ and desired yaw ψ_d into the required roll (ϕ), pitch (θ), and thrust (T) commands.

The attitude controller does not compute full inverse dynamics, but rather uses a kinematic transformation based on the assumption that the UAV's thrust vector is aligned with the body z-axis. This approach is common in cascaded PID designs, and sufficient for moderate accelerations and low attitude angles.

The following equations are used:

```
function [roll, pitch, thrust] = accControl(acc, yaw, DroneMass,
   Gravity)
    xddot = acc(1);
    yddot = acc(2);
    zddot = acc(3);

    pitch = atan2((-xddot * cos(yaw) - yddot * sin(yaw)), (-zddot +
       Gravity));
    roll = atan2(((-xddot * sin(yaw) + yddot * cos(yaw)) *
       cos(pitch)), (-zddot + Gravity));

    roll_limit = 25 * pi / 180;
    pitch_limit = 25 * pi / 180;

    roll = min(roll_limit, max(-roll_limit, roll));
    pitch = min(pitch_limit, max(-pitch_limit, pitch));

    thrust = DroneMass * (-zddot + Gravity) / (cos(pitch) *
       cos(roll));
end
```

Listing 4.1: Attitude control function `accControl`

The thrust is computed to compensate both for the desired vertical acceleration and for gravity, while taking into account the current attitude of the UAV. Since the UAV's thrust vector is aligned with its body-frame z-axis, any tilt (roll or pitch) reduces the effective vertical component of the generated force. Thus, to achieve a specific vertical acceleration in the inertial frame, the actual thrust must be scaled accordingly.

The formula used to compute the total thrust T is:

$$T = \frac{m \cdot (-\ddot{z} + g)}{\cos(\theta) \cdot \cos(\phi)}$$

where:

- T is the total thrust generated by the UAV,
- m is the mass of the UAV,
- $\ddot{z}$ is the desired vertical acceleration in the inertial frame (positive upwards),
- g is the gravitational acceleration ($9.81\,\mathrm{m/s^2}$),
- θ is the pitch angle,
- ϕ is the roll angle.

4.5 Yaw Control

In contrast to the translational control loop, yaw control is decoupled and handled separately due to the underactuated nature of quadrotors. The yaw rate command r_d is then applied through the attitude controller, allowing coordinated turns while tracking position:

$$r_d = P \cdot (\psi_d - \psi)$$

where r_d is the desired yaw rate and $P = 2$ is the proportional gain. This yaw rate is then fed into the attitude controller alongside roll, pitch, and thrust commands.

4.6 Practical Considerations and Controller Simplifications

During the implementation and simulation of the control architecture, several important practical considerations emerged, which influenced the final tuning and structure of the controllers.

Initially, the outer control loops (position and velocity) were designed with full PID control. However, during simulation, it was observed that the derivative component introduced significant high-frequency oscillations—commonly known as *chattering*—especially in the velocity and acceleration commands. This effect is due to the derivative action amplifying measurement noise and discrete-time numerical fluctuations, which are particularly pronounced in high-dynamic systems like UAVs.

To mitigate this, instead of removing the derivative term entirely, a very small derivative gain was retained, and the internal derivative filter of the PID block was configured with a low filter coefficient. This configuration ensures that only the low-frequency components of the derivative are preserved, effectively attenuating high-frequency noise without completely removing the damping contribution of the derivative term. The result is a smoother and more stable control signal with significantly reduced chattering.

Furthermore, the integral action—intended to eliminate steady-state error—caused noticeable overshoot in the position response. In a waypoint-based navigation task like the one presented here, the system is repeatedly subject to changing setpoints, meaning that the presence of an integral term often leads to *integral wind-up*, whereby the accumulated error persists even after the reference has changed. Empirically, setting the integral gain to zero resulted in faster convergence and improved waypoint compliance without sacrificing stability or accuracy.

These observations support a streamlined control structure, composed primarily of proportional control for the position and velocity loops, augmented by a lightly filtered and scaled-down derivative action. This design strikes a balance between stability, responsiveness, and noise robustness, and proved effective for tracking both RRT-generated and obstacle-avoidance trajectories in simulation.

Chapter 5

Simulation and Results

5.1 RRT Algorithm

The first simulation focuses on validating the planned trajectory generated by the RRT planner. The UAV is tasked with following the interpolated path obtained from the offline motion planner, without the aid of real-time obstacle avoidance.

Figure 5.1 shows the UAV trajectory in 3D space, where the gray path represents the desired reference trajectory and the red path indicates the actual executed path. While Figure 5.2 shows how the position evolves over time in the form of X, Y and Z coordinates.

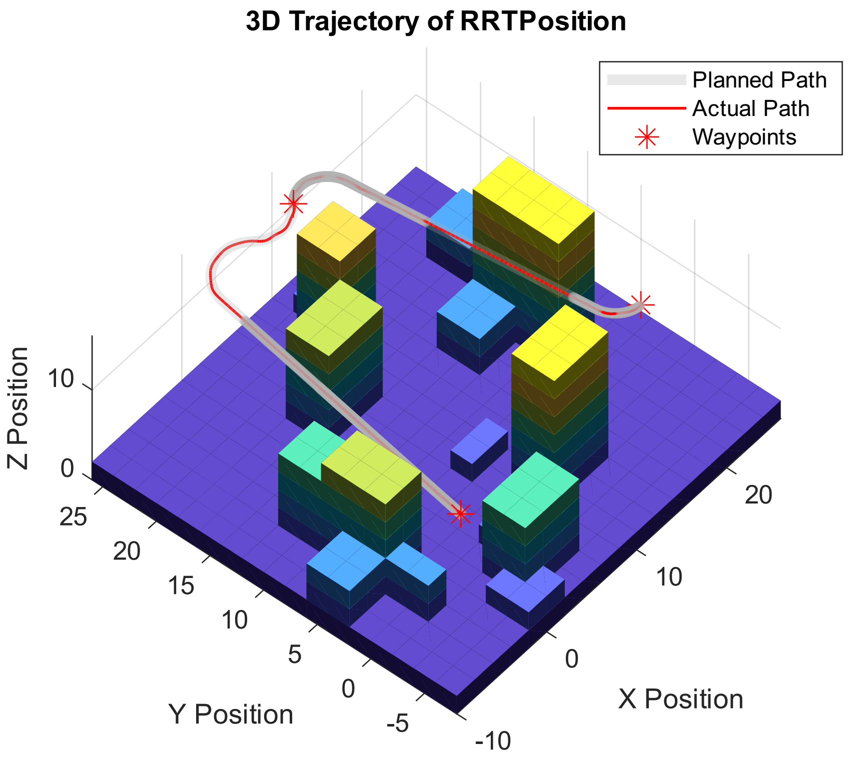

Figure 5.1: RRT: Planned vs Executed Trajectory in 3D space

The tracking error in each dimension is plotted in Figure 5.3.

The controller generates the necessary control inputs to track the RRT path. These include roll, pitch, yaw rate, and thrust. Figure 5.4 illustrates how these signals evolve

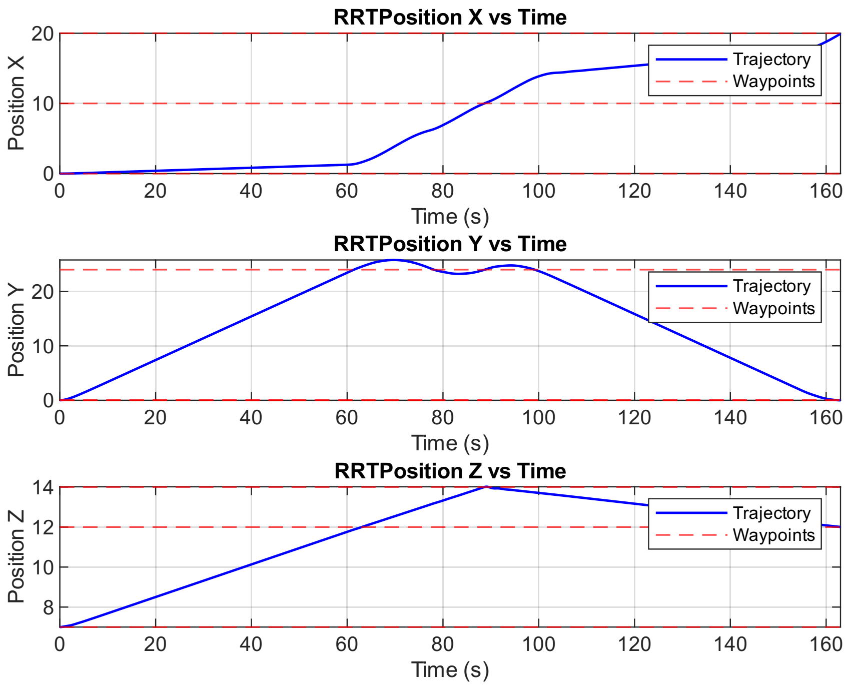

Figure 5.2: RRT: Waypoints and Executed Trajectory in x, y, z

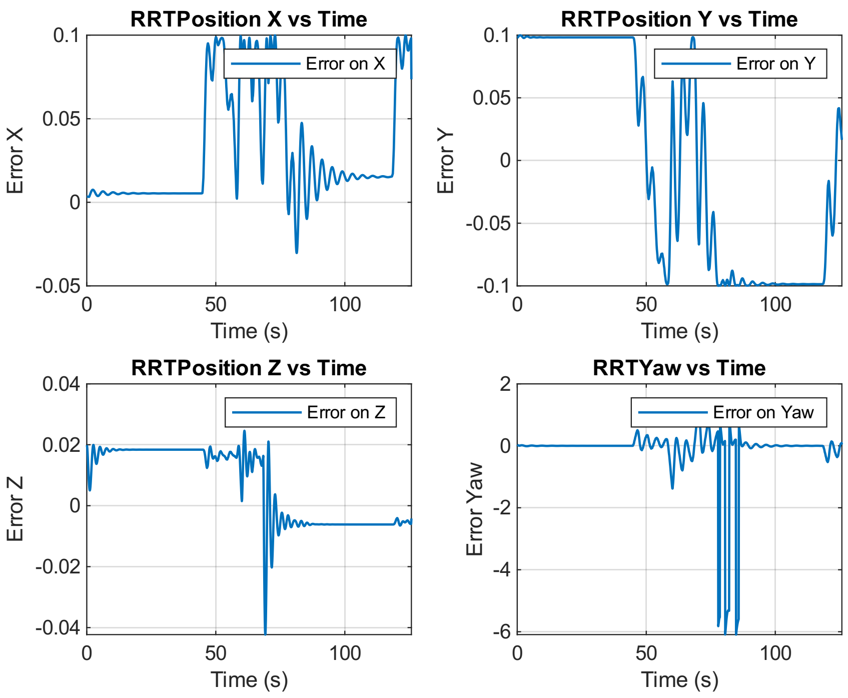

Figure 5.3: RRT: Tracking Errors in x, y, z, and yaw

during flight.

The RRT planner achieves complete waypoint compliance and accurate trajectory tracking. The position error remains bounded across all axes, with only slight increases near the final portion. However, the high control frequency (1 kHz) results in evident chattering in the control inputs.

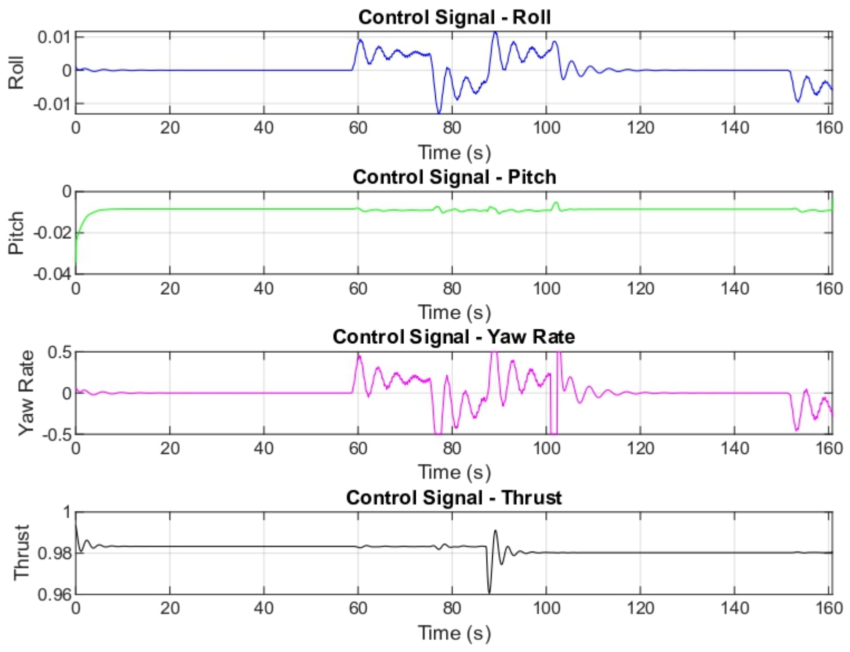

Figure 5.4: RRT: Control Inputs — Roll, Pitch, Yaw Rate, and Thrust

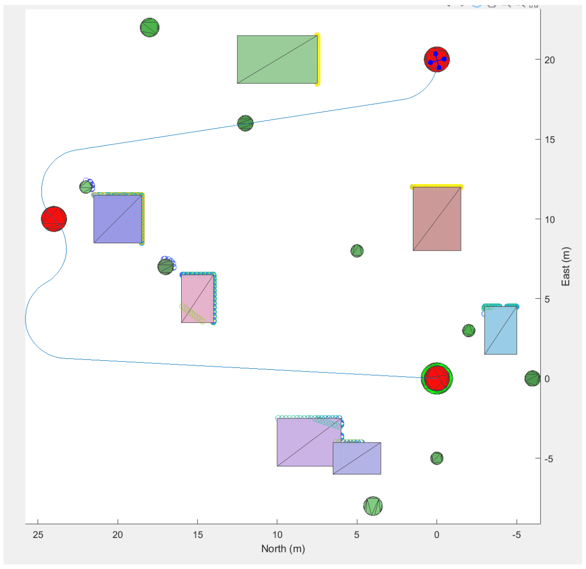

Figure 5.5: 2D Path for RRT

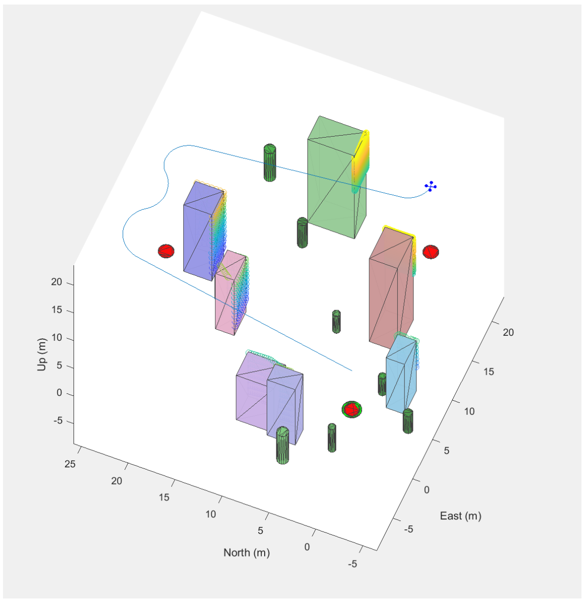

Figure 5.6: 3D Path for RRT

5.2 UAV Obstacle Avoidance

The second simulation integrates a real-time obstacle avoidance strategy based on the 3D VFH+ algorithm. The UAV follows a predefined sequence of waypoints, while dynamically reacting to obstacles using lidar perception.

The resulting UAV trajectory is shown in Figure 5.7. As the UAV navigates through the environment, deviations from the direct waypoint path are introduced by the obstacle avoidance layer. The UAV successfully avoids collisions while still progressing toward the target waypoints.

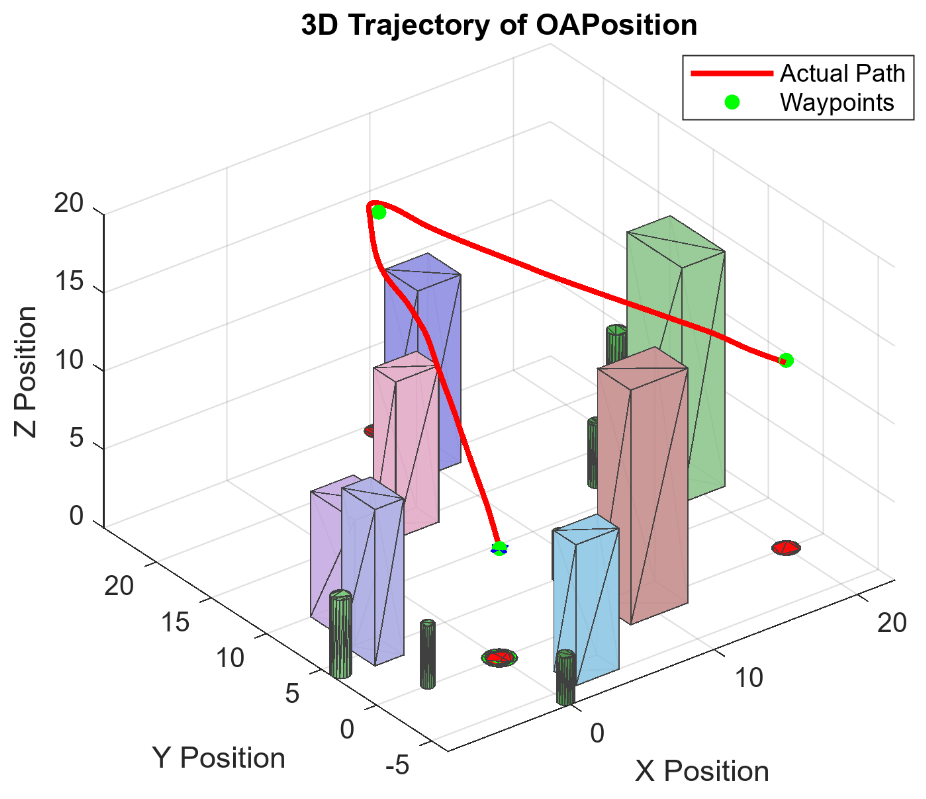

Figure 5.7: Obstacle Avoidance: Executed Trajectory in x, y, z, and yaw

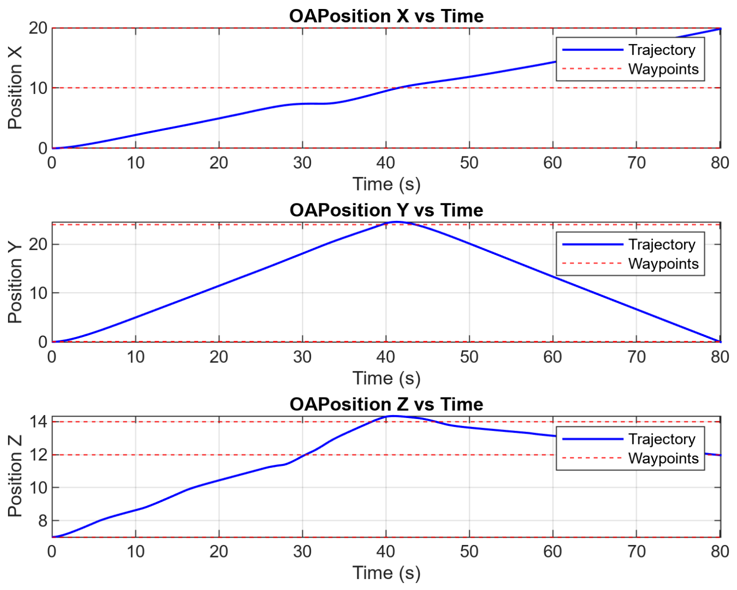

Figure 5.8: Obstacle Avoidance: Executed Trajectory in x, y, z, and yaw

Figure 5.9 shows the tracking errors with respect to the ideal path defined by direct linear waypoint connections. The error grows near obstacles due to avoidance maneuvers but remains within acceptable limits.

The control signals required to perform evasive maneuvers are shown in Figure 5.10. In comparison to the RRT case, more variation is observed in roll and yaw rate, reflecting the higher agility needed to avoid obstacles. The pitch and thrust remain within bounded ranges.

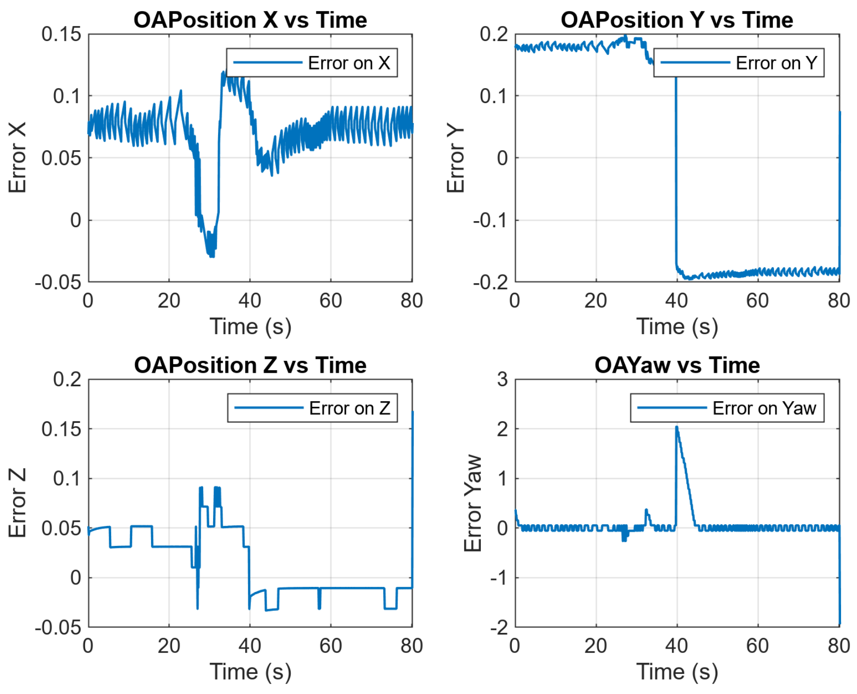

Figure 5.9: Obstacle Avoidance: Tracking Errors in x, y, z, and yaw

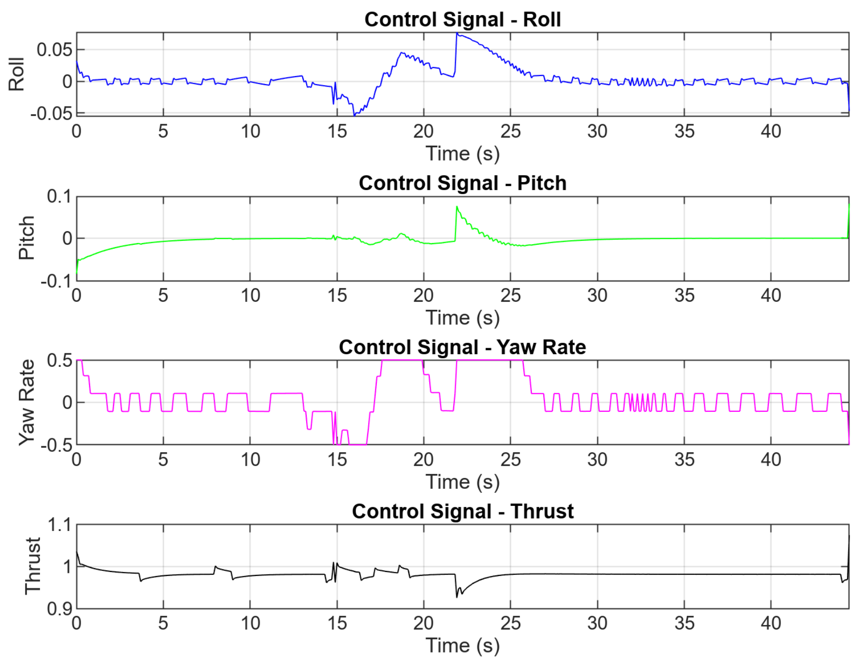

Figure 5.10: Obstacle Avoidance: Control Inputs — Roll, Pitch, Yaw Rate, and Thrust

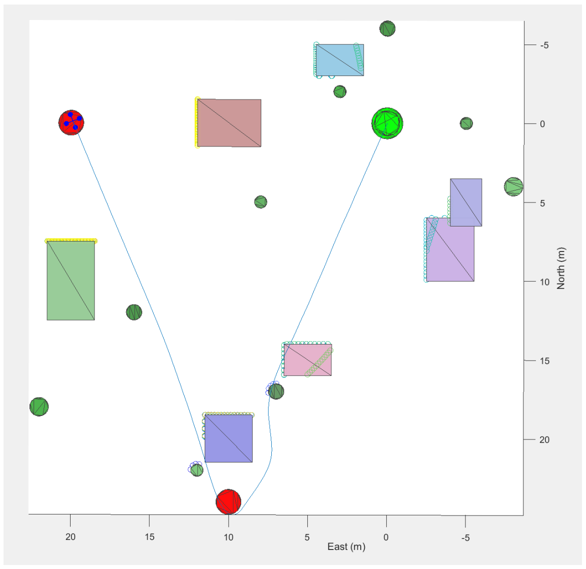

Figure 5.11: 2D Path for OA

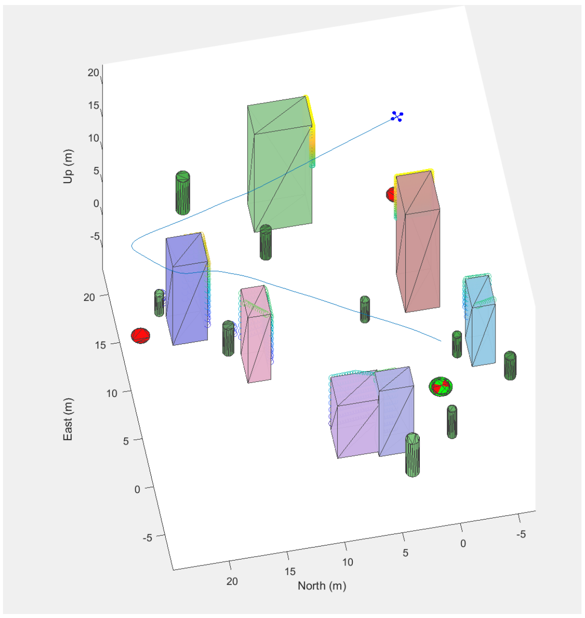

Figure 5.12: 3D Path for OA

5.3 Comparison

Trajectory Shape and 3D Path Fidelity The 3D plots confirm that the UAV path in the RRT case closely follows the reference trajectory. Both planned and actual paths are nearly indistinguishable, and the resulting motion is smooth and continuous. In the

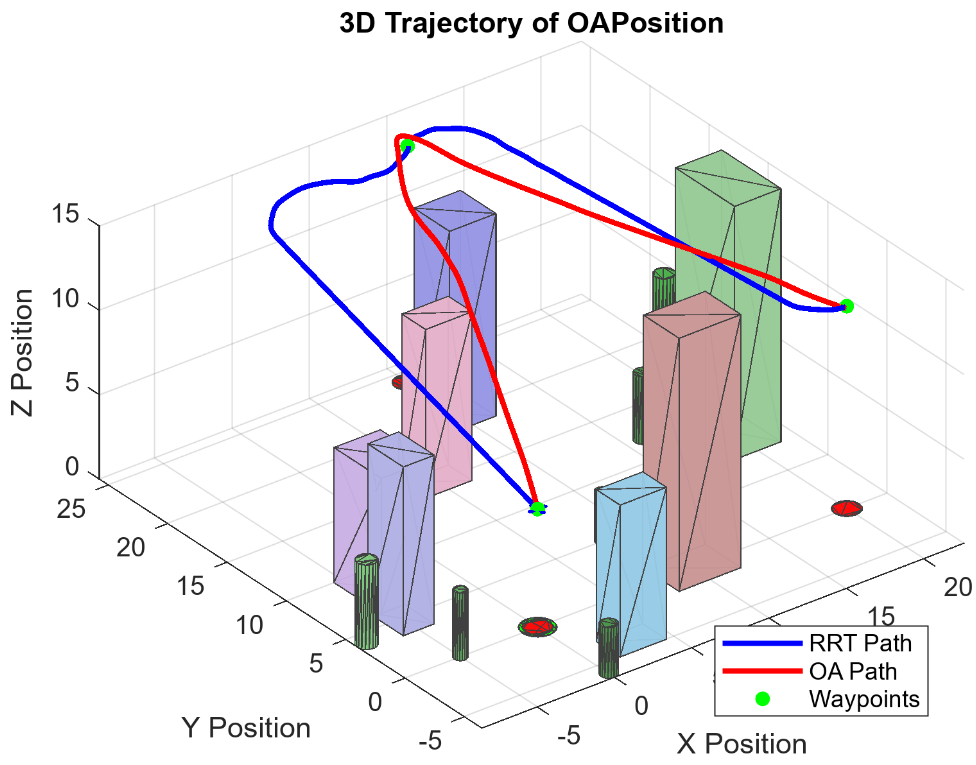

Figure 5.13: Obstacle Avoidance vs RRT paths

OA scenario, the trajectory shows clear deviations, especially during the early phases, where obstacle-induced maneuvers significantly alter the planned path. Despite this, the UAV is still able to reach its intended waypoints, validating the algorithm's effectiveness in cluttered environments.

Figure 5.13 shows the comparison between the executed paths.

Tracking Accuracy The RRT-based control exhibited excellent tracking performance (Figure 5.3), with errors consistently remaining between 0.08 m and 0.1 m over a 160-second simulation period. This indicates a highly accurate and stable path-following behavior, primarily due to the smoothness and predictability of the precomputed reference trajectory.

In contrast, the OA system experienced higher initial tracking errors (Figure 5.9), peaking at around 0.5 m before settling to approximately 0.1 m after 50 seconds. The larger initial deviation reflects the dynamic corrections made to avoid nearby obstacles, which compromise trajectory adherence in favor of safety.

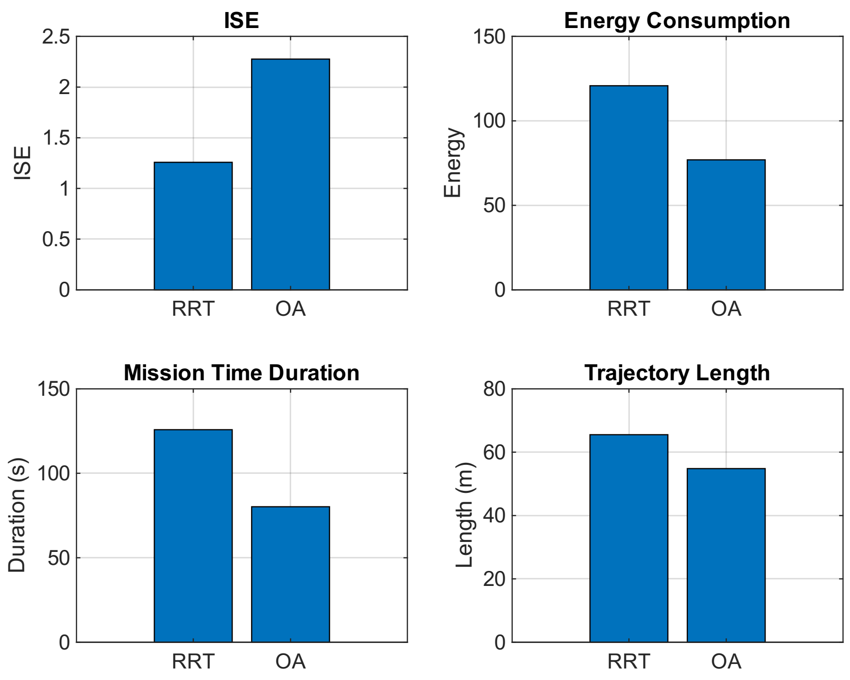

Figure 5.14: Comparison of ISE, energy consumption, mission duration, and trajectory length.

Mission Efficiency From the bar plot in Figure 5.14, it is evident that RRT achieves a substantial lower ISE, indicating better overall tracking accuracy.

In fact, the total mission time, defined as the time span from the start to the end of each trajectory, shows that OA completes the mission significantly faster, indicating more time-efficient path planning and execution.

Although OA does not precisely follow the waypoints, it achieves a shorter trajectory length compared to RRT, due to more direct paths.

Table 5.1: Comparison of performance metrics between RRT and OA

Metric	**RRT**	**OA**
Integrated Squared Error (ISE)	1.629	2.2761
Estimated Energy Consumption	156.8373	76.925
Trajectory Length [m]	63.31	80.2
Mission Duration [s]	163	56.4
Waypoint Compliance	Full	Full
Max Thrust [N]	0.9935	1.0396
Min Thrust [N]	0.9610	0.9283
Mean Thrust [N]	0.9809	0.9810

Chapter 6

Conclusion

This project focused on developing, implementing, and evaluating two UAV navigation algorithms in a simulated 3D environment populated with static obstacles such as buildings and trees. The two primary navigation strategies studied were a precomputed Rapidly-exploring Random Tree (RRT) planner using Dubins motion primitives and a real-time obstacle avoidance (OA) method based on the 3D Vector Field Histogram Plus (VFH+) algorithm. Both approaches were integrated with a cascaded PID control architecture to ensure effective trajectory tracking.

- The Obstacle Avoidance (**OA**) approach is more suited for real-time agile navigation in unpredictable or dynamic environments, where obstacle appearances and map data may be incomplete or changing. Its reactive obstacle avoidance allows the UAV to dynamically re-plan trajectories without requiring full path re-computation, resulting in gains in mission speed and energy savings at the expense of tracking precision.
- The Rapidly-exploring Random Tree (**RRT**) method is preferable for structured scenarios where a reliable occupancy map is available and mission requirements demand accurate waypoint following and stable control profiles. RRT ensures predictable and smooth trajectories, which are essential for tasks requiring strict adherence to spatial constraints.

Overall, the choice between these planning strategies should be driven by the mission context: **OA is preferable for real-time, agile navigation in unpredictable environments**, whereas **RRT is more suitable for structured scenarios where map information is available and precise path following is critical**.