Planiranje inspekcijske trajektorije bespilotnih letjelica

Čolić, Mirta

Master's thesis / Diplomski rad

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:877531

Download date / Datum preuzimanja: **2025-03-26**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 114

INSPECTION TRAJECTORY PLANNING FOR DRONES

Mirta Čolić

Zagreb, February 2025

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 114

INSPECTION TRAJECTORY PLANNING FOR DRONES

Mirta Čolić

Zagreb, February 2025

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 30 September 2024

MASTER THESIS ASSIGNMENT No. 114

Student:	**Mirta Čolić (0036519022)**
Study:	Information and Communication Technology
Profile:	Control Systems and Robotics
Mentor:	assoc. prof. Matko Orsag, PhD

Title: **Inspection trajectory planning for drones**

Description:

As part of the assigment, it is necessary to develop a software solution for trajectory planning of unmanned aerial vehicles (UAVs) for infrastructure inspection. Existing literature in the field of trajectory planning for robotic systems and UAVs should be reviewed and analyzed. The selected method must be adapted and implemented on a laboratory model of an unmanned aerial vehicle. For the implementation, the corresponding infrastructure for tracking the UAV, as well as the appropriate electromechanical and software components, must be prepared. The system should be tested in both a simulation and a laboratory environment.

Submission date: 14 February 2025

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 30. rujna 2024.

DIPLOMSKI ZADATAK br. 114

Pristupnica:	**Mirta Čolić (0036519022)**
Studij:	Informacijska i komunikacijska tehnologija
Profil:	Automatika i robotika
Mentor:	izv. prof. dr. sc. Matko Orsag

Zadatak: **Planiranje inspekcijske trajektorije bespilotnih letjelica**

Opis zadatka:

U sklopu diplomskog zadatka potrebno je razviti programsko rješenje za planiranje trajektorije bespilotnih letjelica za inspekciju infrastrukture. Potrebno je pregledati i analizirati postojeću literaturu u području planiranja trajektorije robotskih sustava i bespilotnih letjelica. Odabranu metodu potrebno je prilagodit i implementirati na laboratorijskom primjerku bespilotne letjelice. Za potrebe implementacije potrebno je pripremiti pripadajuću infrastrukturu za praćenje letjelice te odgovarajuće elektromehaničke i programske komponente. Sustav je potrebno ispitati u simulacijskom i laboratorijskom okruženju.

Rok za predaju rada: 14. veljače 2025.

I would like to express my sincere thanks to all the people who supported and guided me in the completion of this Master's thesis. First, I warmly thank my mentor, Matko Orsag, and assistant, Marijana Peti, for their guidance, insightful advice, and constant availability. My thanks also go to my family, friends, and fellow students for their moral support throughout my studies.

Contents

1 Introduction 3

2 Motivation 4

3 UAV Fundamentals 8

3.1 UAV dynamics 8

3.2 Translational Dynamics 9

3.3 Rotational Dynamics 10

3.4 Rotation Matrices 11

3.5 PID 12

3.5.1 PID Controller Components 12

3.5.2 PID controller tunning 14

4 Trajectory generation methods 15

4.1 Polynomial Method for Trajectory Generation 15

4.2 Time-Optimal Path Parametrization 17

5 Implemenation 21

5.1 System components 21

5.1.1 Crazyflie 21

5.1.2 Crazyradio 22

5.1.3 AI deck 22

5.1.4 Optitrack - Motion Capture System 23

5.2 Implementation 1 24

5.2.1 Spiral waypoint generation 24

5.2.2 Lawnmower waypoint generation 29

5.3 Implementation 2 30
5.3.1 PID cascade implementation 31

6 Results 34
6.1 Implementation 1 34
6.1.1 Spiral trajectory 34
6.1.2 Lawnmower trajectory 36
6.2 Implementation 2 37
6.2.1 Lawnmower trajectory 37
6.2.2 Analysis 39
6.2.3 Photos 40

7 Conclusion 42

References 43

Abstract 45

Sažetak 46

1 Introduction

Unmanned Aerial Vehicles (UAVs), commonly referred to as drones, have emerged as transformative and vital tools across various industries, offering innovative solutions to complex challenges. Drones equipped with high-resolution cameras and advanced sensors facilitate detailed inspections of structures such as bridges, pipelines, and power lines, significantly reducing the need for manual assessments in hard-to-reach areas. The result of performing inspections with drones is high resolution data used for detailed analysis conducted by professionals. This not only enhances safety by minimizing human exposure to dangerous environments, but also ensures cost savings through early detection of potential issues, thereby preventing repairs and reducing downtime.

The development of trajectory generation systems is an important aspect of optimizing drone operations for inspection purposes. Such systems must ensure that the drone navigates efficiently while maintaining comprehensive coverage of the target area. In this context, research focuses on implementing a trajectory generation system intended for inspection through taking images of desired area, addressing the specific requirements of industrial applications.

This thesis presents the methodology, implementation, and evaluation of a trajectory generation system for drone-based inspections. Through exploring different methods of trajectory generation, such as the polynomial and TOPP-RA method, the system aims to optimize flight paths for drones during inspection missions, ensuring that data collection is both comprehensive and efficient. By integrating the principles of robotics, optimization, and control, the proposed system enhances the precision and efficiency of inspection tasks. In the following sections, two different implementations of trajectory generation, as well as the results of these implementations, are presented.

2 Motivation

Since drone development began, it has undergone significant transformations, evolving from rudimentary unmanned aerial vehicles (UAVs) used for military training and target practice to sophisticated systems used in various applications today. Naturally, one of the questions that emerged was how drones can automate processes in industry. That led to the opening of various use cases in different industries. Drones provide safer, faster, more efficient and less expensive alternatives to inspections and data collection in hazardous environments, reducing the need for human intervention. In this chapter, various real-world applications from different industries will be covered. Industries such as infrastructure, energy, oil and gas, construction, agriculture, mining, manufacturing, public safety, maritime, and telecom benefit significantly from drone technology.

Infrastructure and Civil Engineering

Drones play a crucial role in inspecting infrastructure and ensuring the longevity and safety of vital structures. UAVs inspect bridges, roads, railways, dams, and waterways by detecting cracks, corrosion, and structural weaknesses without disrupting traffic. AI-powered drones such as the Skydio X10 can generate 3D models of bridges and tunnels, detecting defects invisible to the naked eye. Skydio drones also showed a high prevalence in the inspection and data collection of transmission line towers. [1] A study conducted by the Swedish Transport Administration explored the use of unmanned aerial vehicles (UAVs) for bridge inspections. The research involved creating digital 3D models of bridges using photogrammetry, allowing inspectors to identify damages such as cracks and spalling remotely. The findings suggest that UAVs can complement traditional inspection methods by providing detailed visual data, especially in hard-to-reach areas. [2]

Energy Sector

Drones improve safety and efficiency in power generation and distribution by inspecting power lines, substations, wind turbines, solar farms, and hydropower plants. The SkySpecs Wind Turbine Inspection Drone autonomously scans turbine blades, capturing high-resolution images to detect damage. The Flyability Elios 3 drone enables internal inspections of wind turbine blades, potentially saving more than $1 million per blade by detecting faults early. Companies such as SGS, Aerones, vHive, and Toshiba Energy Systems use drones to minimize downtime, improve safety, and automate offshore wind turbine inspections.

Oil and Gas Industry

Drones reduce risks in hazardous environments by monitoring pipelines, offshore rigs, and refineries. They detect leaks, corrosion, and methane emissions using infrared cameras. The Percepto Sparrow, an autonomous drone-in-a-box solution, is deployed for routine inspections in hazardous locations, reducing human exposure to dangerous conditions and improving response times for maintenance teams.

Construction and Real Estate

Drones facilitate site surveys, progress monitoring, safety inspections, and building maintenance. The DJI Matrice 350 RTK captures high-precision data to create 3D models, which assists in project management and documentation. Construction firms use drones to compare actual versus planned progress, ensuring efficiency and reducing costs.

Agriculture and Forestry

UAVs enhance productivity and sustainability in agriculture by analyzing crop health, monitoring irrigation, tracking livestock, and managing forests. The DJI Mavic 3 Thermal utilizes thermal imaging to detect stressed crops early, preventing large-scale losses. Drones also help in forestry by detecting illegal logging and wildfire risks, ensuring better resource management.

Mining and Geology

Drones improve safety and accuracy in mining operations by conducting surveys, environmental monitoring, and tailings dam inspections. The Microdrones md4-1000 carries LiDAR sensors for detailed terrain analysis, enabling precise volume calculations and geospatial mapping, reducing the need for human surveyors in hazardous areas.

Manufacturing and Warehousing

Autonomous drones streamline industrial operations by conducting factory inspections, inventory management, and security surveillance. The Verity Warehouse Drones autonomously navigate warehouses to perform inventory counts, while Corvus Robotics' Corvus One system operates without additional infrastructure, optimizing efficiency and labor allocation.

Public Safety and Emergency Response

Drones assist in firefighting, search and rescue, and disaster assessment. The DJI Mavic 3 Enterprise Series, equipped with thermal imaging, helps locate missing persons and assess fire damage. Public safety agencies rely on drones for real-time data to enhance emergency response strategies.

Maritime and Offshore Inspections

Drones ensure safety and efficiency in marine environments by inspecting the hulls of ships, monitoring ports, and detecting oil spills. The Voliro T hexacopter performs contact-based inspections of ship hulls and offshore structures without requiring scaffolding or divers, minimizing risks and improving maintenance processes.

Telecom Industry

Drones support network reliability and maintenance by inspecting cell towers, helping in 5G deployment, and assessing fiber optic cables. The Parrot Anafi AI, with 4G connectivity and advanced imaging, helps telecom companies assess the infrastructure remotely, reducing manual labor and operational costs.

The adoption of drones across industries highlights their transformative impact on inspection and maintenance operations. By enhancing safety, reducing costs, and improving data accuracy, UAVs have become indispensable tools in modern industrial workflows. As technology advances, further integration of AI and automation in drone inspections will continue to drive efficiency and innovation.

3 UAV Fundamentals

The drone consists of several key components: frame, motors, propellers, and controller. The frame of a drone serves as its structure and is typically constructed from lightweight materials such as carbon fiber or plastic. The motors that power the propellers are situated in the periphery of the frame. Rotation of the propellers causes a lift force that acts on the body of UAV and lifts the drone from the ground. Depending on the number of propellers, there are several types of drone configurations: tricopter, quadcopter, hexacopter, and octocopter. In this work a quadcopter is used, as it is the most common configuration. To achieve stable and controlled flight, a quadcopter relies on a combination of aerodynamic forces, gyroscopic effects, and electronic control systems that adjust rotor speeds in real time. This requires a mathematical representation of the drone's dynamics, which is typically expressed using Newton-Euler equations. These equations describe both translational and rotational motion in a 3D space, taking into account forces such as gravity, thrust, and aerodynamic drag, as well as torques responsible for rotational movement.

3.1 UAV dynamics

The quadcopter has 6 degrees of freedom (DOF), three for position translations in all three axis, x, y and z, and three for rotations around those three axis, causing roll(ϕ), pitch(θ) and yaw(ψ) rotations.

To accurately analyze and characterize the motion of a drone, it is essential to establish well-defined coordinate frames that facilitate tracking of its position, orientation, and velocity within three-dimensional space. In the context of drone modeling, two primary coordinate systems are mostly used:

1. Inertial Frame (Earth-Fixed Frame) - World or Global Frame W: This coordinate system is fixed relative to Earth and serves as a global reference for describing the drone's absolute position and motion. It is typically defined by orthogonal axes (North-East-Down or East-North-Up) and is used to represent the drone's location in a global context. The absolute position and velocity of the UAV are measured relative to this frame.

2. Body Frame B: This coordinate system is attached to the drone itself, and its origin is typically located at the drone's center of mass. The axes of this frame align with the drone's principal axes of motion. The body-fixed frame is necessary for describing the drone's orientation and relative motion, such as roll, pitch, and yaw.

Since a drone can alter its orientation relative to the inertial frame, it is necessary to establish a mathematical relationship that calculates the transformation between these two coordinate systems. This transformation is achieved through the use of rotation matrices, which provide a way to convert values from the body-fixed frame to the Earth-fixed inertial frame, and vice versa. The formulation and application of these rotation matrices will be explored in detail in a subsequent section.

3.2 Translational Dynamics

The translational motion of a drone refers to its ability to move in the X, Y, and Z directions within the inertial frame. This motion is governed by Newton's Second Law of Motion, which states that the rate of change of momentum of an object is equal to the total force acting on it. Mathematically, this is expressed as:

$$m\frac{d\mathbf{v}}{dt} = \mathbf{F} \tag{3.1}$$

where:

- m is the mass of the drone.
- $\mathbf{v} = [vx, vy, vz]^T$ is the velocity vector in the inertial frame.
- $\mathbf{F}$ represents the sum of all external forces acting on the drone.

Expanding this equation, the total force acting on the drone in the inertial frame is given by:

$$m\ddot{\mathbf{p}} = \mathbf{R}_B^I \cdot \mathbf{F}_B - m\mathbf{g} \tag{3.2}$$

where:

- $\mathbf{p} = [x, y, z]^T$ is the position of the drone in the inertial frame.
- $\mathbf{R}_B^W$ is the rotation matrix transforming forces from the body frame to the inertial frame.
- $\mathbf{g}$ is the acceleration due to gravity with direction $[0, 0, 1]^T$

For a quadcopter, the primary force acting in the body frame is the total thrust generated by all four rotors, given by the following.

$$F_T = \sum_{i=1}^{4} T_i \tag{3.3}$$

Each rotor produces a thrust force that depends on its angular velocity:

$$T_i = k_f \omega_i^2 \tag{3.4}$$

where:

- ω_i is the angular velocity of the rotor i.
- k_f is a thrust coefficient that depends on the characteristics of the propeller and the motor.

3.3 Rotational Dynamics

The rotational dynamics describe how the drone changes its orientation in space. These dynamics are governed by the Euler equations:

$$\mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{J}\boldsymbol{\omega}) = \tau \tag{3.5}$$

where:

- $\mathbf{J}$ is the inertia matrix.
- $\boldsymbol{\omega} = [p, q, r]^T$ is the angular velocity in the body frame.
- $\tau = [\tau_x, \tau_y, \tau_z]^T$ represents the external torques that act on the drone.

3.4 Rotation Matrices

The rotation matrix used to transform the coordinates in the world frame to body frame is given by:

$$\mathbf{R}_B^W = R_z(\psi)R_y(\theta)R_x(\phi) \tag{3.6}$$

where:

- ϕ (roll) is the rotation about the x_B-axis.
- θ (pitch) is the rotation about the y_B-axis.
- ψ (yaw) is rotation about the z_B-axis.

Final rotation matrix that represents rotation of body frame in world coordinate system is expressed as follows:

$$\mathbf{R}_W^B = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix} \tag{3.7}$$

3.5 PID

A Proportional-Integral-Derivative (PID) controller is a widely used control loop feedback mechanism essential for managing various industrial processes. PID control is based on a closed-loop strategy with a negative feedback scheme[3]. It works by continuously calculating an error value as the difference between a desired setpoint (SP) and a measured process variable (PV). The PID controller then applies corrective actions to minimize this error, improving system stability and performance. The error value $e(t)$ is calculated with equation 3.8, where $r(t)$ is the reference value, or as mentioned before the setpoint value, and $y(t)$ is the output value or the variability of the measured process, respectively.

$$e(t) = r(t) - y(t) \tag{3.8}$$

3.5.1 PID Controller Components

The PID controller consists of three distinct components: proportional (P), integral (I), and derivative (D) terms. Each term contributes to the control action based on the error signal e(t). The overall control action $u(t)$ is the sum of these three components, as shown in Equation 3.9. This equation shows how each component contributes to the adjustment of the control variable, allowing precise regulation of process variables such as temperature, pressure, or speed.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)dt + K_d \frac{de(t)}{dt} \tag{3.9}$$

where:

- K_p is the proportional gain
- K_i is the integral gain
- K_d is the derivative gain

Proportional component

The proportional component produces an output that is directly proportional to the current error $e(t)$. The bigger the error, the larger the correction variable, that is, the bigger

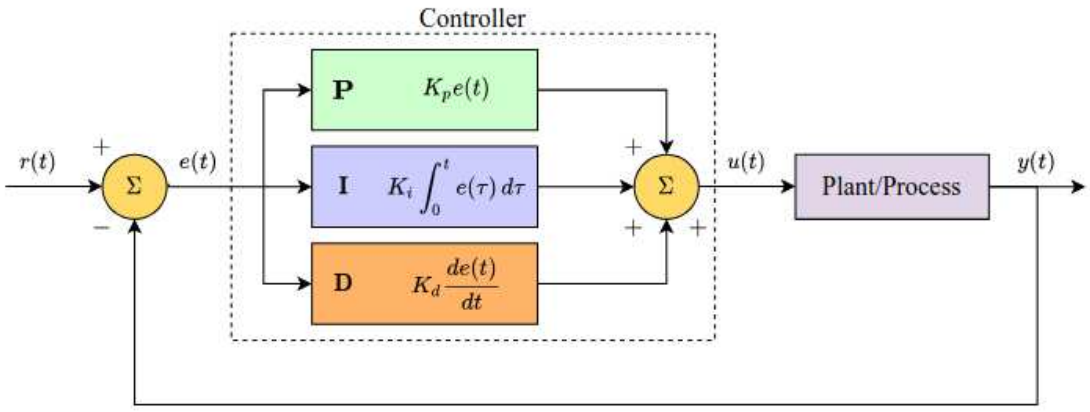


Figure 3.1: [3] Block diagram of a PID controlled process

the proportional component. The proportional gain K_p determines how aggressively the controller responds to the current error. However, using proportional control alone can lead to a steady-state error, where the output stabilizes at a value different from the set-point. Increasing K_p reduces steady-state error, but can lead to oscillations or instability if set too high.

Mathematically, the proportional component of the PID regulator can be expressed as the equation below 3.10.

$$u_P(t) = K_p e(t) \tag{3.10}$$

Integral component

The integral component accounts for the accumulation of past errors over time. Addresses accumulated past errors by integrating the error over time. This helps eliminate any residual steady-state errors that may persist after applying proportional control. The integral term ensures that even smaller errors are corrected over time. However, if the integral component is too large, it can cause overshoot and instability. The integral gain K_i adjusts the speed with which the controller reacts to accumulated errors. Impact of integral component, on the output can be calculated with equation 3.11

$$u_I(t) = K_i \int_0^t e(\tau) d\tau \tag{3.11}$$

Derivative component

The derivative term predicts future error behavior based on the rate of change of the error. It predicts future errors based on the rate of change of the current error. This anticipatory action helps dampen oscillations and improve system stability, particularly when changes occur rapidly. The derivative term dampens the system's response, reducing overshoot and improving stability, although if not chosen correctly it can amplify noise in the error signal. It is proportional to the derivative of the error signal, and is calculated as in equation 3.12

$$u_D(t) = K_d \frac{de(t)}{dt} \tag{3.12}$$

3.5.2 PID controller tunning

Process of selecting right combination of PID controller gains is called tunning. That process is extremely important, since the whole system depends on how the regulator works and whether or not the system will have big oscillations or a slow reaction time. Some of the most common methods for tuning the parameters of K_p, K_i and K_d include:

- Ziegler-Nichols method: A heuristic approach that provides initial gain values based on the system's response to step inputs.
- Trial and Error: Manually adjusting the gains while observing the system behaviour
- Model-Based Tuning: Using a mathematical model of the system to optimize the gains.

The cascade PID controller schematic used in this work, and the process of tuning PID regulators is described in the chapter 5.3

4 Trajectory generation methods

4.1 Polynomial Method for Trajectory Generation

Polynomial method for trajectory generation is a mathematical approach to planning smooth and efficient paths for drones. By using polynomials to model trajectories, drones can achieve smooth transitions in position, velocity, and higher-order derivatives such as acceleration and jerk [4]. It involves using polynomial equations to describe the position, velocity, and acceleration of the system over time. This chapter explains the mathematical principles underlying the generation of a polynomial trajectory of the n-th degree. In this work, 8th degree polynomials were used to calculate the desired trajectory. These high-order polynomials are particularly useful in scenarios that require strict adherence to boundary conditions and optimization of smoothness metrics, such as minimum jerk or snap.

Defining the polynomial

A trajectory can be represented by a polynomial of degree n. The general form of an n-th degree polynomial is:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + ... + a_n t^n \tag{4.1}$$

Where:

- $q(t)$: Position at time t
- $a_0, a_1, ..., a_n$: Coefficients to be determined
- t: time variable

Specify Boundary Conditions

To determine the coefficients (a_0 to a_n), we need at least $n + 1$ boundary conditions. These typically include:

- Initial position: $q(0) = q_0$
- Final position: $q(T) = q_f$
- Initial velocity: $\dot{q}(0) = \dot{q}_0$
- Final velocity: $\dot{q}(T) = \dot{q}_f$
- Initial acceleration: $\ddot{q}(0) = \ddot{q}_0$
- Final acceleration: $\ddot{q}(T) = \ddot{q}_f$
- Initial jerk: $\dddot{q}(0) = \dddot{j}_0$
- Final jerk: $\dddot{q}(T) = \dddot{j}_f$
- Snap (change in jerk): $\ddddot{q}(t) = s_0$

The derivatives of $q(t)$ are:

$$\dot{q}(t) = a_1 + 2a_2t + 3a_3t^2 + ... + na_nt^{n-1} \tag{4.2}$$

$$\ddot{q}(t) = 2a_2 + 6a_3t + 12a_4t^2 + ... + n(n-1)a_nt^{n-2} \tag{4.3}$$

Using these equations at both endpoints (i.e., at $t = 0$ and $t = T$), necessary equations can be formed based on the boundary conditions.

Derivatives for Velocity and Acceleration

To get velocity and acceleration expressions for UAV along the desired trajectory, mathematical expression $q(t)$ needs to be differentiated. The first derivative of position expression $q(t)$ is velocity expression for UAV 4.4. The second derivative of $q(t)$ or deriving velocity expression $\dot{q}(t)$, results in acceleration expression 4.5.

$$\dot{q}(t) = \frac{dq}{dt} = a_1 + 2a_2 t + 3a_3 t^2 + ... + n a_n t^{n-1} \tag{4.4}$$

$$\ddot{q}(t) = \frac{d^2 q}{dt^2} = 2a_2 + 6a_3 t + ... + n(n-1) a_n t^{n-2} \tag{4.5}$$

These derivatives provide information on how the drone or any other robotic system will move along the trajectory over time.

Evaluation and Implementation

After generating the trajectory, it is essential to evaluate it against any constraints or requirements specific to the desired application. This may include checking maximum speed limits, maximum acceleration limits, or collision avoidance with obstacles in the environment.

If any constraints are violated, it may be necessary to adjust the boundary conditions, consider utilizing higher-order polynomials, or employ piecewise polynomial segments to develop a more suitable trajectory.

Finally, trajectory can be implemented in robot control system, in this case UAV's. That process involves sampling points along the trajectory at discrete time intervals, sending commands to the UAV's flight controller to follow these waypoints, and monitoring real-time feedback from sensors to ensure adherence to the planned path.

4.2 Time-Optimal Path Parametrization

Time-Optimal Path Parameterization based on Reachability Analysis (TOPP-RA) is a trajectory optimization method used to compute the fastest traversal of a predefined path while respecting system constraints such as velocity, acceleration, and dynamic limits. It is particularly advantageous for real-time applications in robotics due to its convex optimization formulation, which ensures computational efficiency and robustness. Traditional approaches to TOPP rely on Numerical Integration (NI), which is fast but prone to robustness issues, or Convex Optimization (CO), which is more reliable but computationally expensive. The proposed approach, called TOPP-RA, is based on Reachability

Analysis (RA), allowing for recursive computation of controllable sets at discretized positions along a given path using small Linear Programs (LPs).

Mathematical Formulation

A given *n-DOF* robot follows a geometric path in configuration space:

$$\boldsymbol{q} = \boldsymbol{q}(s), \quad s \in [0, s_{\text{end}}] \tag{4.6}$$

The trajectory along this path is parameterized by time t as:

$$\boldsymbol{q}(t) = \boldsymbol{q}(s(t)) \tag{4.7}$$

where $s(t)$ is an increasing function defining motion along the path.

The system is subject to second-order constraints:

$$\boldsymbol{A}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T \boldsymbol{B}(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{f}(\boldsymbol{q}) \in C(\boldsymbol{q}) \tag{4.8}$$

where:

- $A(\boldsymbol{q})$: Inertia matrix,
- $B(\boldsymbol{q})$: Coriolis/centrifugal forces,
- $f(\boldsymbol{q})$: Gravitational forces,
- $C(\boldsymbol{q})$: Convex polytope of actuator limits.

By differentiating $\boldsymbol{q}(s)$:

$$\dot{\boldsymbol{q}} = \boldsymbol{q}'\dot{s}, \ddot{\boldsymbol{q}} = \boldsymbol{q}''\dot{s}^2 + \boldsymbol{q}'\ddot{s} \tag{4.9}$$

Substituting 4.8 to 4.9 results in transforminh second order constraints on the system dynamics into the constraints on $s, \dot{s}, \ddot{s}$ [5]:

$$\boldsymbol{a}(s)\ddot{s} + \boldsymbol{b}(s)\dot{s}^2 + \boldsymbol{c}(s) \in C(s) \tag{4.10}$$

where:

$$\boldsymbol{a}(s) = \boldsymbol{A}(\boldsymbol{q}(s))\boldsymbol{q}'(s) \tag{4.11}$$

$$\boldsymbol{b}(s) = \boldsymbol{A}(\boldsymbol{q}(s))\boldsymbol{q}''(s) + \boldsymbol{q}'(s)^T\boldsymbol{B}(\boldsymbol{q}(s))\boldsymbol{q}'(s) \tag{4.12}$$

$$\boldsymbol{c}(s) = \boldsymbol{f}(\boldsymbol{q}(s)) \tag{4.13}$$

$$\zeta(s) := \mathcal{C}\big(q(s)\big)\zeta \tag{4.14}$$

Instead of using numerical integration or convex optimization, this method recursively computes reachable and controllable velocity sets. The reachable set L_i at stage i (position s_i) defines all velocities that can be reached from an initial velocity $\dot{s}_0$. The controllable set K_i at stage i contains all velocities from which a valid trajectory to the goal velocity $\dot{s}_N$ exists. These sets are recursively computed using small Linear Programs (LPs).

TOPP-RA algorithm

TOPP-RA algorithm can be split in two phases. The first one is backward pass which recursively computes controllable sets K_i, and the second one is forward pass which computes optimal states and controls. TOPP-RA algorithm can be ilustrated as shown in Fig. 4.1

Algorithm 1: TOPP-RA

Input : Path $\mathcal{P}$, starting and ending velocities $\dot{s}_0, \dot{s}_N$

Output: The optimal parameterization $x_0^*, u_0^*, \ldots, u_{N-1}^*, x_N^*$

/* Backward pass: compute the controllable sets */

$\mathcal{K}_N := \{\dot{s}_N^2\}$

for $i \in [N-1 \ldots 0]$ **do**

  $\mathcal{K}_i := \mathcal{Q}_i(\mathcal{K}_{i+1})$

if $\mathcal{K}_0 = \emptyset$ *or* $\dot{s}_0 \notin \mathcal{K}_0$ **then**

  return Infeasible

/* Forward pass: greedily select the controls */

$x_0^* := \dot{s}_0^2$

for $i \in [0 \ldots N-1]$ **do**

  $u_i^* := \max u$, subject to: $x_i^* + 2\Delta_i u \in \mathcal{K}_{i+1}$ and $(x_i^*, u) \in \Omega_i$

  $x_{i+1}^* := x_i^* + 2\Delta_i u_i^*$

Figure 4.1: TOPP-RA algorithm [5]

Experimental Results and Complexity

When tested on simple 6-DOF paths, TOPP-RA is as fast as NI but more reliable. While, when tested on complex 50-DOF robots, it outperforms existing methods. Computes joint torques and contact forces directly. Computational complexity of TOPP-RA can be calulated with $O(mN)$, where as Numerical Integration and Convex Optimization method have $O(m^2N)$ and $O(KmN^3)$ complexity, respectively.

In conclusuion, TOPP-RA balances both speed and robustness, it is a scalable method for time-optimal path parameterization. TOPP-Ra is widely used in various robotic applications, including industrial robots performing high-speed pick-and-place tasks, mobile robots requiring optimal path execution, and autonomous vehicles navigating dynamic environments. TOPP-RA directly formulates the problem as a convex optimization task, making it well-suited for real-time applications and high-dimensional robotic systems.

5 Implemenation

In this chapter, two different implementations of computing drone trajectory and ensuring drone execution of desired trajectory are presented. For trajectory generation, two methods are used to make a comparison between the two, the mathematical background of these trajectories is explained in detail in previous chapter 4. The trajectories chosen for testing are spiral and *"lawnmower"* because they offer efficient coverage patterns for different scenarios. The spiral trajectory is good for base stations because of its concentric circular pattern, while the lawnmower trajectory is excellent for systematic and comprehensive coverage of rectangular or irregularly shaped areas. Crazyswarm simulator was used in the development of the system, its description and source code can be found here [6]. Once the whole system was developed and tested on the simulator, it was tested in a controlled environment with the setup described in chapter 5.1.

5.1 System components

5.1.1 Crazyflie

The Crazyflie drone is utilized in this work as the platform for investigation, experimenting and testing trajectory planning. The Crazyflie 2.1+. is a versatile open-source flying development platform that only weighs 27g and fits in the palm of your hand. Crazyflie 2.1 + is also equipped with low-latency/long-range radio as well as Bluetooth LE.[7]

Figure 5.1: Crazyflie drone

5.1.2 Crazyradio

Crazyradio 2.0, a long range open USB radio dongle based on the nRF52840 from Nordic Semiconductor, was used to establish communication between Crazyflie and computer.

Crazyradio 2.0 is not only for usage together with the Crazyflie family of devices, since it is an open project with open firmware and a Python API, it's a great building block for systems that require more predictable latency compared to WiFi and does not have the same bandwidth requirements. The hardware comes with a bootloader that enables firmware upgrades via USB without any additional hardware needed. [8]

Figure 5.2: Crazyflie drone

5.1.3 AI deck

The AI Deck 1.1 brings ultra-low-power AI computing to Crazyflie 2.x, powered by the GAP8 RISC-V processor and a Himax monochrome camera for on-board image process-

ing and classification. With an ESP32 for WiFi connectivity, it enables real-time data streaming and remote control. The initial setup requires flashing the GAP8 bootloader via JTAG, but over-the-air updates are possible afterwards. In this study, this camera is used to perform visual inspection of the desired area. [9]

Figure 5.3: AI Deck - camera used in performing inspection

5.1.4 Optitrack - Motion Capture System

Optitrack is and industry leading precision motion capture and 3D tracking systems for video game design, animation, virtual reality, robotics, and movement sciences. It utilizes infrared cameras with high-speed image processing to track reflective markers in real-time with submillimeter accuracy. The system supports multi-camera setups, allowing for flexible tracking volumes and improved accuracy through redundant data capture. OptiTrack software, such as Motive, provides tools for calibration, real-time visualization, and data streaming to external applications through APIs.

In robotics and UAV research, OptiTrack is often used for precise localization, enabling indoor positioning where GPS is unavailable. It seamlessly integrates with platforms like ROS (Robot Operating System) for autonomous navigation and control. The system can track multiple objects simultaneously, assigning unique marker patterns to differentiate between them. Wireless streaming and real-time feedback make it ideal for applications requiring high-speed motion tracking.

Figure 5.4: Optitrack infrared camera

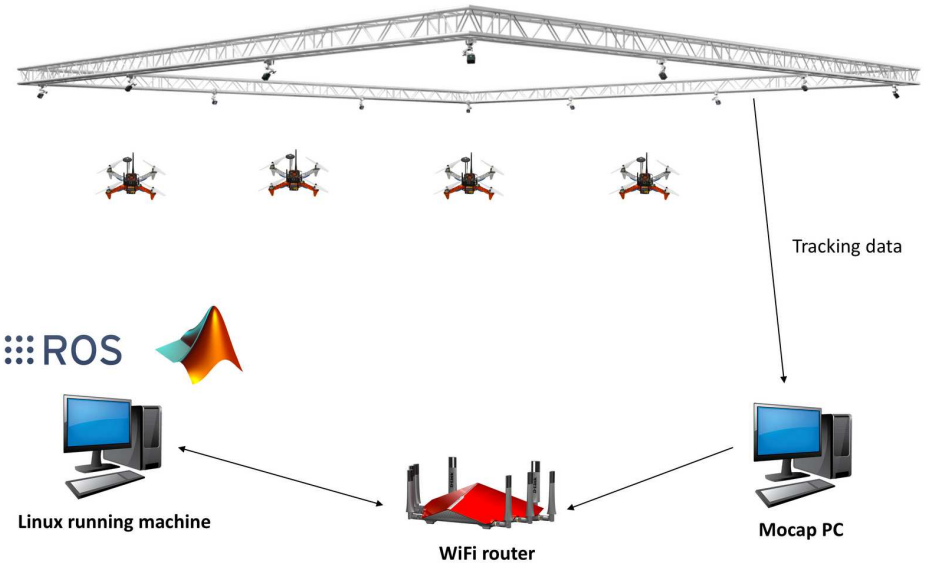


Figure 5.5: Motion tracking setup with optitrack cameras [10]

5.2 Implementation 1

5.2.1 Spiral waypoint generation

To achieve a successful visual inspection of the base station, the UAV must complete a full circular path at multiple heights. The most suitable trajectory for this task is a spiral, allowing the drone to progressively ascend or descend while maintaining full coverage of the target. Generating this spiral trajectory requires defining key parameters: waypoint radius, initial height, final height, number of circles, and total duration. These parameters can be adjusted as needed to meet specific inspection requirements. The trajectory

waypoints are generated using a parametric equation where the height z is interpolated linearly from the initial value to the final value over time, ensuring a smooth ascent or descent:

$$z = height_initial + dh \cdot t \tag{5.1}$$

The horizontal motion follows a circular path, with the radius rr determining the width of the spiral and the angular velocity ww defining the speed of rotation:

$$x = r \cdot \cos(\omega \cdot t), y = r \cdot \sin(\omega \cdot t) \tag{5.2}$$

To maintain proper orientation, the yaw angle is adjusted dynamically, ensuring that the UAV always faces inward toward the center of the spiral:

$$\psi = wt + \pi \tag{5.3}$$

This implementation consists of several nodes that are all placed in the ros package *spiral_trajectory*. In the graph 5.12 the hierarchy of *spiral_trajectory* package is represented.

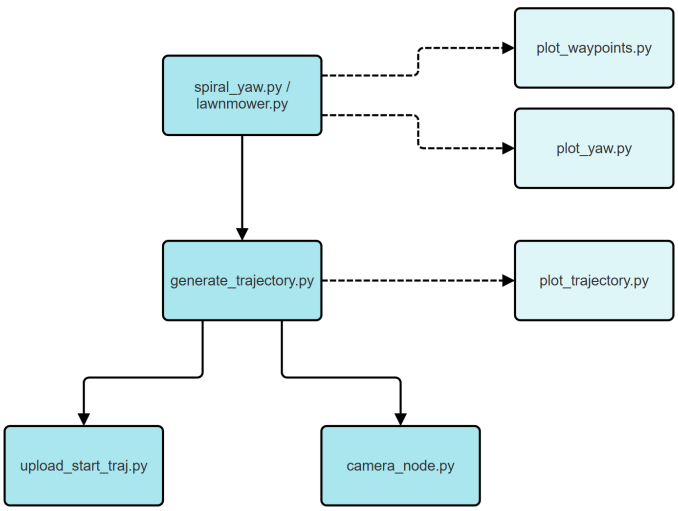


Figure 5.6: Graph representing code tree hierarchy

Firstly, there is script named *spiral_yaw.py* which generates waypoints in spiral shape and saves them in a csv file. The script allows to compute desired spirals with variable radius (r[m]), initial point height (hi[m]), final height of the spiral (hf[m]), number of circles (n), and total duration of the trajectory (t[s]). Since the purpose of this project is to perform an inspection of the base station with the drone, it is also important to implement the yaw angle calculation when generating waypoints. The mathematical equations used for the generation of the waypoints are explained in chatper **??**. After computing waypoints, with *generate_trajectory.py* script, spiral trajectory is computed and saved to a csv file. This script *generate_trajectory.py* generates a trajectory for UAVs using a piecewise polynomial approach [11]. When calling a script argument *–pieces* can be specified, the arguments define in how many pieces given waypoints are going to be split, or consequently in how many pieces trajectory is splitted. This variable is extremely important because, depending on the trajectory that needs to be computed, the success of the computation depends on it.

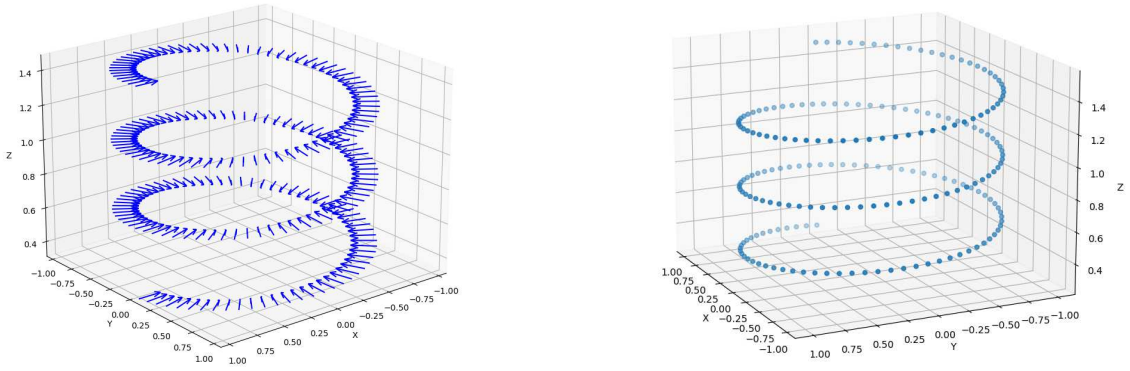
For instance, if the desired trajectory had a longer duration and/or greater path length while maintaining the same number of segments, the computation time would significantly increase, mostly exceeding 15 minutes, for some instances up to 1h, while the resulting trajectory would deviate substantially from the intended one. However, by increasing the number of segments, the computational process would be considerably more efficient, resulting in a smooth trajectory.

Each piece of the trajectory is represented with three polynomials and a time increment. The three polynomials for each piece represent polynomials of the 8th degree for the yaw, x, y and z axis. The time increment represents the duration of the execution of the trajectory piece. To start execution of the desired trajectory, script *upload_start_traj.py* has to be run. Parallel to that, *camera_node.py* should be running so that the images of an object that is being inspected are taken and saved.

Plotting

Several plotting scripts were developed to facilitate debugging by providing a visual representation of data at various stages of processing. In addition to debugging, these plotting scripts offer a clear visualization of generated output, making it easier to verify re-

sults, compare different stages of processing, and communicate findings effectively.



Yaw vector plot Waypoints plot

Figure 5.7: Spiral plots for parameters: *r=1*, *number_of_circles=3*, *minz=0.3*, *maxz=0.15*

The trajectory plot is especially useful in development because it includes plots of velocity, acceleration, omega, and yaw of the drone throughout the execution of the whole trajectory. These data are essential to assess whether the drone will be able to execute a trajectory or not due to physical restrictions. Improper selection of parameters can result in a generated trajectory in which the required velocities and accelerations exceed the physical capabilities of the drone. In such cases, the drone may not be able to perform the necessary maneuvers due to inherent limitations in its design, such as maximum speed, acceleration, and handling capabilities. When the required velocities are excessively high, it can lead to a failure to maintain stable flight, potentially causing the drone to crash. This, in turn, would prevent the successful completion of the intended inspection or task, as the drone would be unable to operate within safe and feasible performance limits.

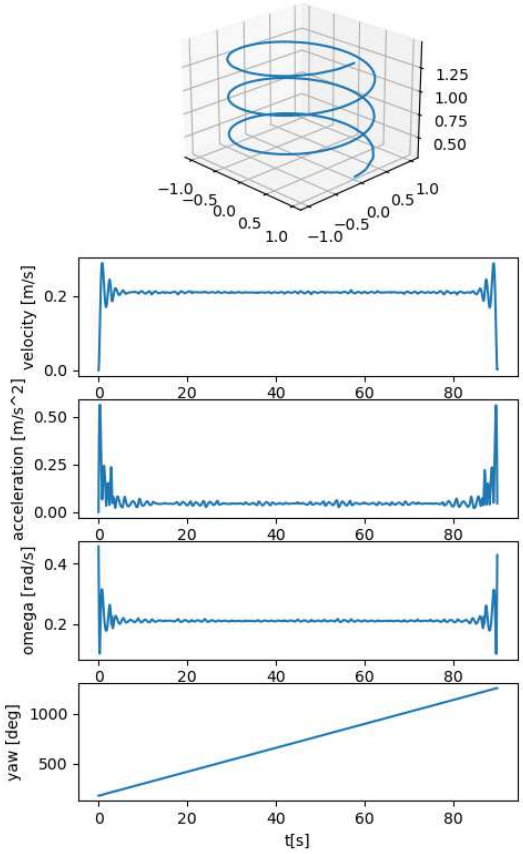


Figure 5.8: Trajectory plot for spiral with parameters: *r=1*, *hi=0.3*, *hf=1.5*, *n=3*, *t=60*

Through a trial and error process, it has been determined that drone velocity and acceleration should not exceed a threshold value of 1. This conclusion comes from the need to ensure that the drone operates within its physical limitations. It is considered optimal that these values are higher during the initial and final phases of the flight, specifically during takeoff and landing, where more dynamic movements are typically required. However, during the execution of the spiral trajectory, it is crucial that the speeds are reduced and, even more importantly, kept constant throughout this phase. This constant, lower speed ensures smoother and more controlled flight. Furthermore, it is recommended that the drone speed be kept relatively low during the image capture process. As the drone velocity increases, the time available to capture each individual image decreases, leading to a higher probability of motion blur. This blurring effect significantly impairs the quality of the images, making it difficult to extract clear and precise data from them. In the context of an inspection task, blurry images make the inspection process much harder or even impossible, which is completely opposite of the given task.

GUI

In addition to the command-line approach, a graphical interface (GUI) has been developed for easier use of the system by user. This GUI provides a way to configure and execute spiral trajectory generation without manually modifying script parameters or interacting with terminal commands. The user interface can be seen in the picture 5.10.

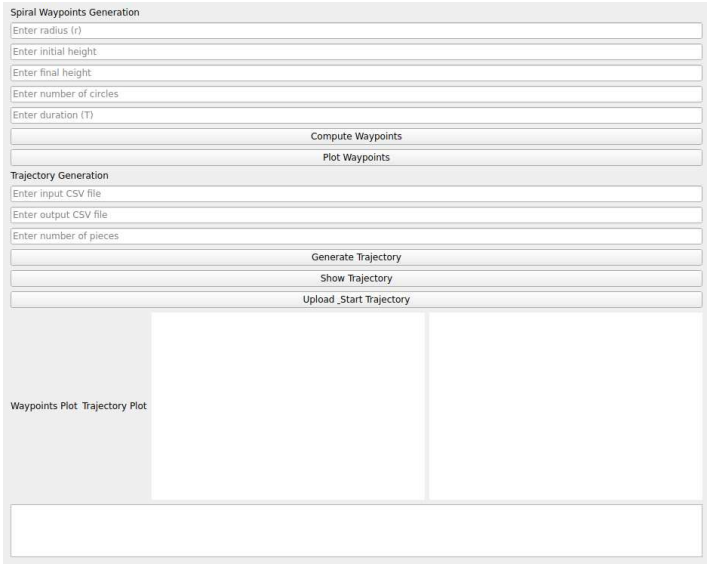


Figure 5.9: Picture of GUI

5.2.2 Lawnmower waypoint generation

To perform a visual inspection of walls, or any other flat surface, a fucntion with variable parameters that define waypoints for lawnmower trajectory had to be used. Variable parameters:

- *min1*, *max1*, *min2*, *max2* defines minimal and maximal values in a 2D plane
- *step_size* defines distance between two points
- *final_time* defines trajectory duration
- *num_turn_waypoints* define number of points per turn
- *line_spacing* define the distance between adjacent lines

It is also possible to define in which plane the trajectory is going to be generated, but

because of the placement of the camera on the UAV, in this work, the functionality is shown in the y-z plane. Algorithm and system structure for program execution are the same as described with spiral trjaectory in chapter 5.2.1.

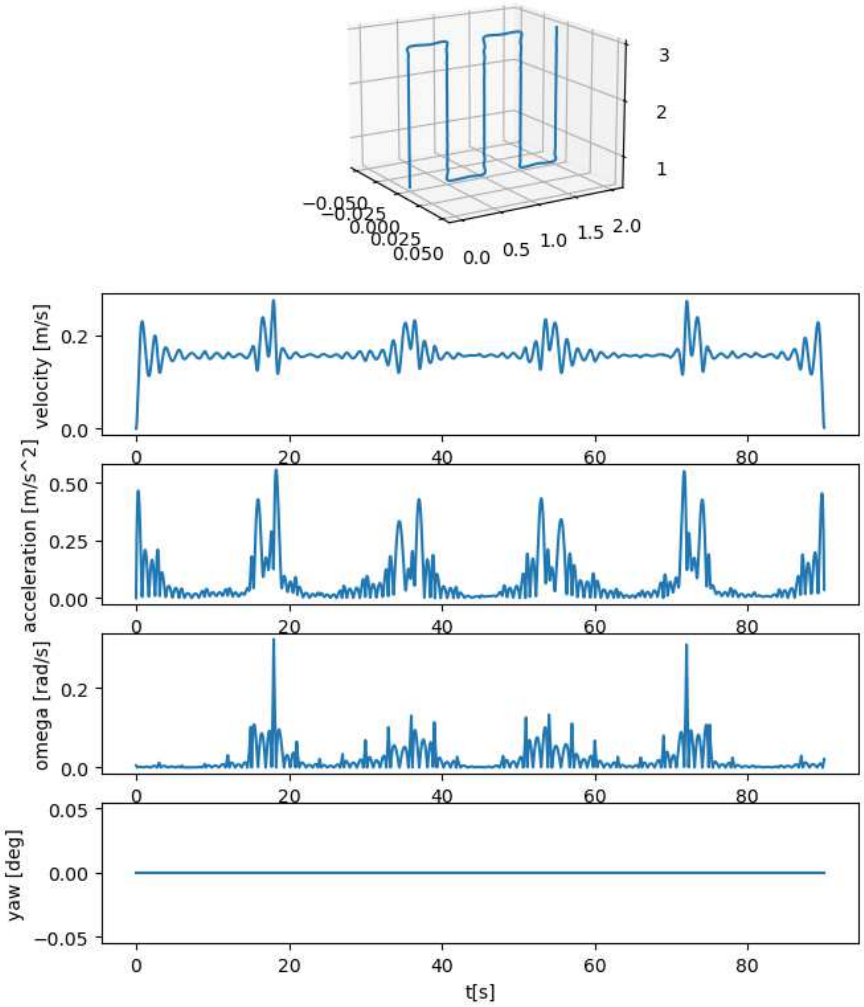


Figure 5.10: Lawnmower trajectory

5.3 Implementation 2

Aim of the second approach was to perform the inspection of a wall or any 2D plane, by using a different method of generating the trajectory and controlling the drone. To generate a suitable trajectory in this implementation, TOPP-RA algorithm, which is explained in detail in chapter 4.2, is used. TOPP-RA algorithm interface in ROS2 consists of service *GenerateTrajectory.srv*, */generate_toppra_trajectory*, request for that service is defined in *GenerateTrajectory.msg*. The request should include waypoints variable type

trajectory_msgs/JointTrajectory, samlping frequency and bool variable for plot. The waypoints variable should, beside the waypoint data, include velocities and acceleration constraints on the joints. The response of this service is a trajectory of type *trajectory_msgs/JointTrajectory*. *JointTrajectory.msg* has data on positions, velocities and accelerations in x, y, z axis for each point in the trajectory, and it also has data on time that passed from the start of the trajectory up to each particular point in trajectory. Those data are used to set reference values for cascade regulator inputs. Velocities and acceleration data are used as part of feedforward control to improve system response, compensate for dynamic effects, and reduce lag in feedback correction.

Since these data cannot be sent directly to the drone, a cascade PID controller had to be implemented. This approach implements cascade PID controller for each coordinate axis, x, y, z and *yaw*. These controllers output the final values for all three coordinate axes and yaw, which is then published to the topic */cf_1/cmd_attitude_setpoint*.

To maintain stability and follow a desired trajectory, UAVs use a control system with inner and outer loop, regulating velocity and position, respectively.

5.3.1 PID cascade implementation

Implementation of PID cascade controller requires the PID tuning process, and in this work, the pid tuning process has been done in two ways. The first approach included modeling of drone dynamics in MATLAB using SIMULINK. SIMULINK schematic for tunning PID parameteres, for both inner and outer loop, for x and y axis is the same because of the symetry of the system dynamics, meaning both axes respond similarly to control inputs. To mathematically model UAV dynamics in SIMULINK, transfer function block was used to represent PT1. A PT1 (first-order lag) system can be used to represent UAV modeling because they exhibit a delayed response to control inputs due to inertia and aerodynamic effects. The PT1 model captures this behavior by approximating how the drone's velocity or position gradually adjusts to changes in thrust or tilt, making it useful for simplified control design and stability analysis.

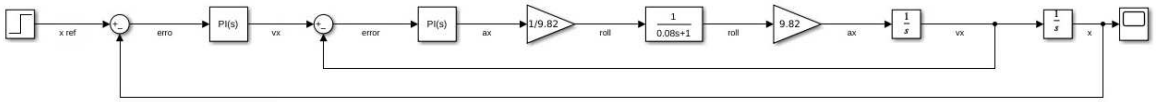


Figure 5.11: Cascade PID regulator, with MATLAB model of UAV dynamics in x-axis

To get the PID parameter values for inner loop, outer pid was removed from schematic, and step function was connected to the inner PID regulator. After running simulation, the PID tuning option can be opened in another window, where by using sliders for "Response Time" and "Transient Behavior" optimal parameters can be chosen. In this implementation, the regulators for both the x and y axes are reduced to the PI regulator, because that implementation showed the best results. After finding optimal parameters for the desired response, those parameters were set as a value for the inner PI regulator. Then another PID block was added to the simulink model, to simulate position regulator. After running simulink model, the same method used for choosing parameters for inner loop, was repeated for choosing parameters for outer loop. PI regulator was chosen for position control as well. The final SIMULINK model for tuning the PID regualtor parameters for the x and y axes is shown in 5.11.

The approach to adjust the PID parameters for the control on the z axis was different from that for the x and y axis. To live plot position and velcoity of the UAV rqt graph was used. Through running simulation, sending a certain refrence value (e.q. $z = 1$), and by observing refrence value and actual UAV position, optimal values were selected.

For yaw control, simple P regulator was used to compute yaw_rate.

After finding values for PID regulators, they were implemented, and fine tuned a little bit more. By sending the desired positions, the reference value and evaluating the response, the value of the parameters was changed based on what impact on system response each component of the ppid controllr has 3.5. Final PID parameters for UAV control used in this work can be found in table 5.1

Control Loop	Axis	Kp	Ki	Kd
position PID	X	1.2	0.02434	0.0
	Y	1.2	0.02434	0.0
	Z	0.9	0.05	0.0
velocity PID	X	5.0	0.0589	0.0
	Y	5.0	0.0589	0.0
	Z	0.07	0.0	0.0
yaw PID		1.5	0.01	0.3

Table 5.1: PID Parameters for UAV Control

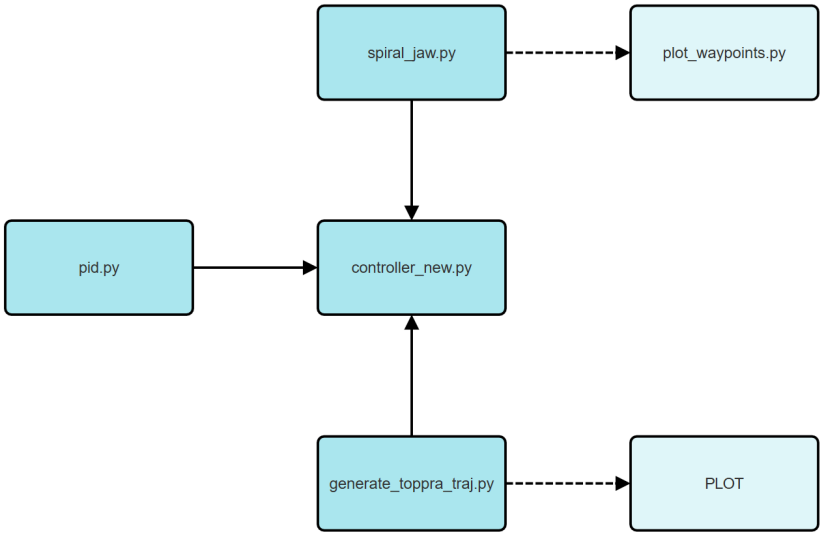


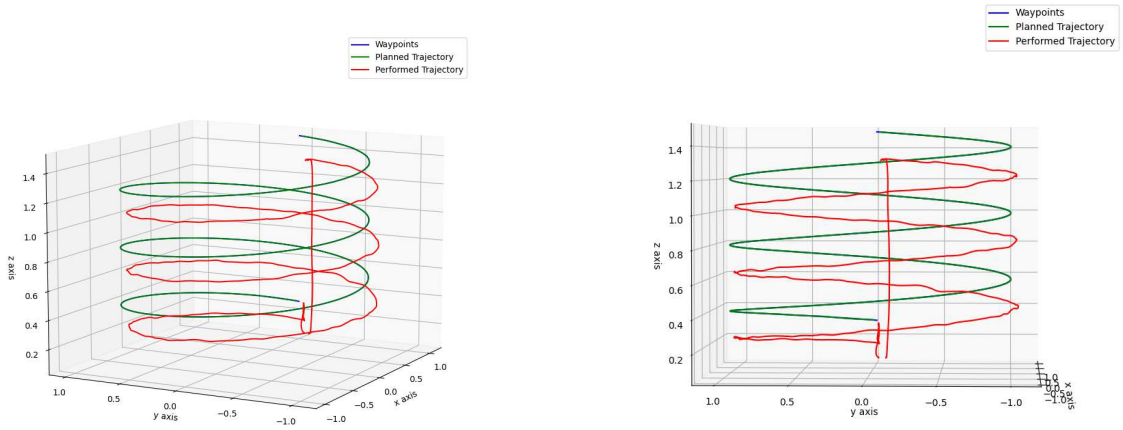
Figure 5.12: Graph representing code tree hierarchy

6 Results

6.1 Implementation 1

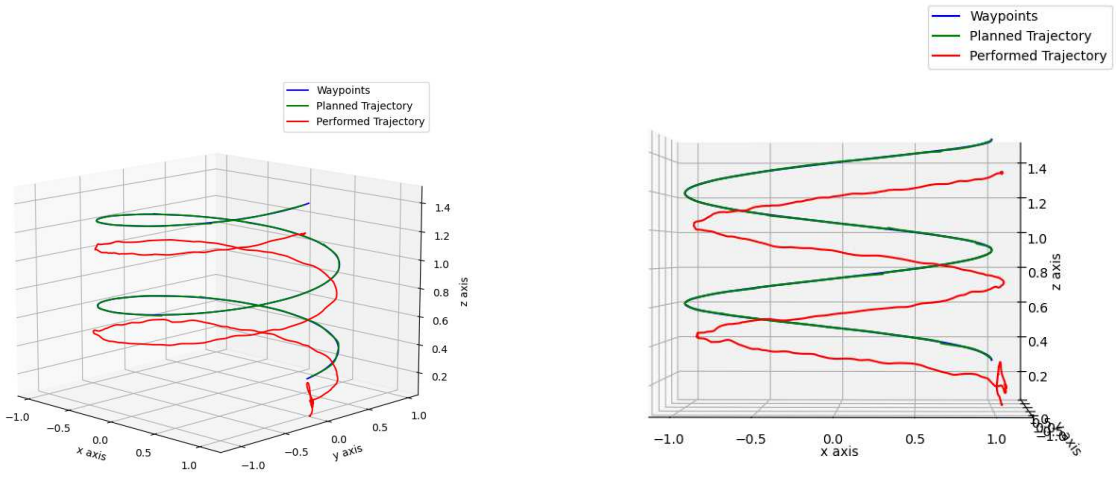
The first implementation was tested on two trajectory patterns generated with different parameters. Three plots in 2D plane, and one 3D plot is show on the images below. They contain graphs of waypoints, generated trajectory and actual position of the UAV during trajectory execution.

6.1.1 Spiral trajectory



3D Plot $y-z$ axis Plot

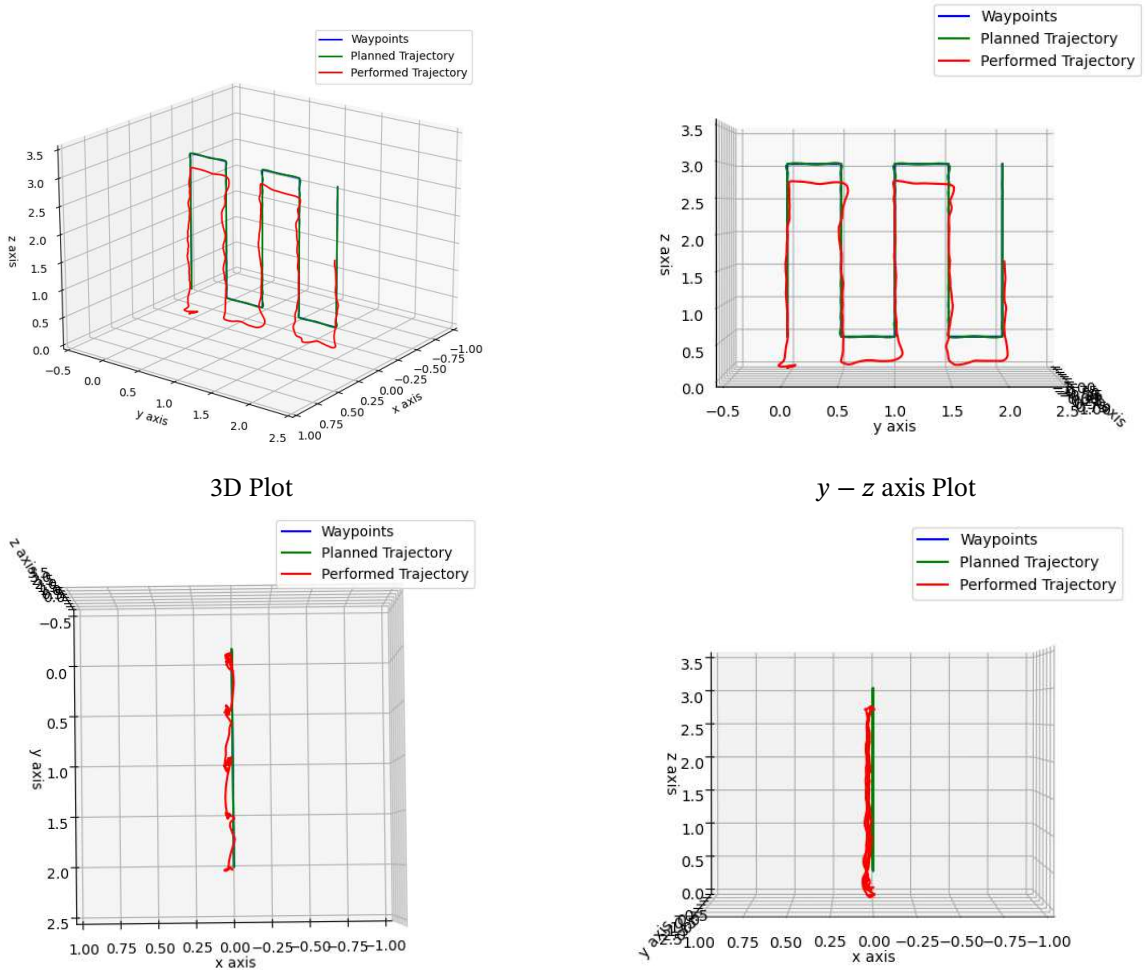
Figure 6.1: Spiral trajectory plots *r=1*, *number_of_circles=3*, *minz=0.3*, *maxz=0.15*



3D Plot $y-z$ axis Plot

Figure 6.2: Spiral trajectory plots *r=1*, *number_of_circles=2*, *minz=0.3*, *maxz=0.15*

6.1.2 Lawnmower trajectory



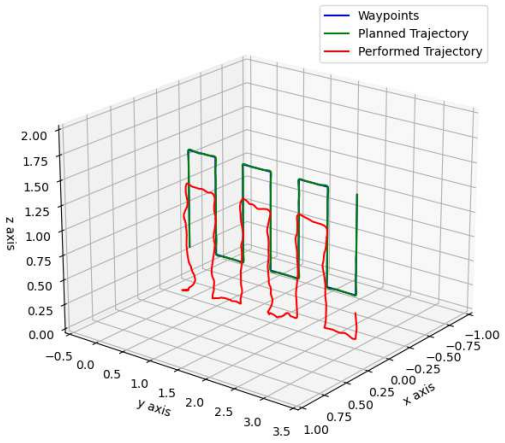
3D Plot

$y - z$ axis Plot

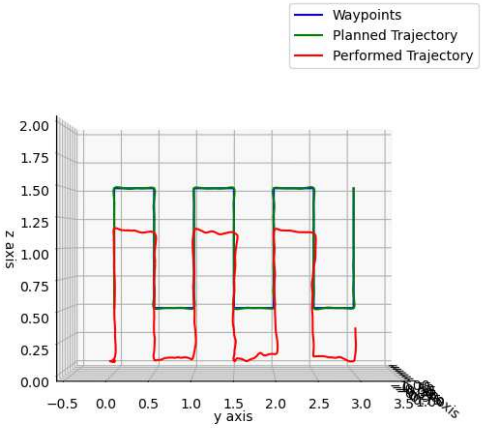
$x - y$ axis Plot

$x - z$ axis Plot

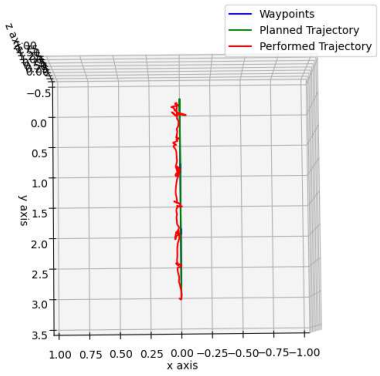
Figure 6.3: Plots of lawnmower trajectory in different planes for parameters: *miny=0*, *maxy=2*, *minz=0.5*, *maxz=3*, *step_size=0.1*, *final_time=90*, *num_turn_waypoints=5*, *line_spacing=0.5*



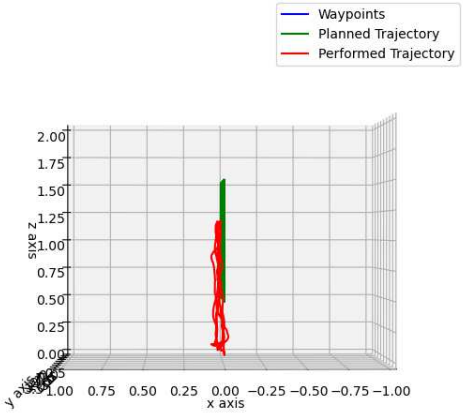
3D Plot



$y - z$ axis Plot



$x - y$ axis Plot



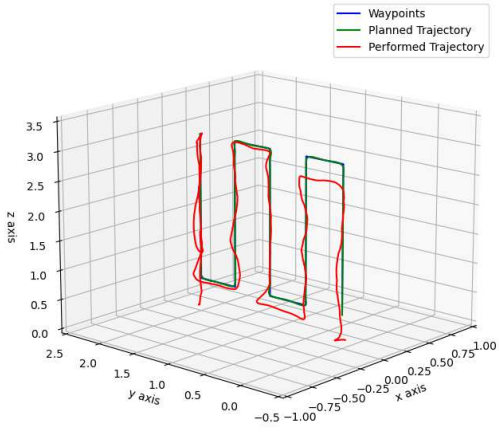
$x - z$ axis Plot

Figure 6.4: Plots of lawnmower trajectory in different planes for parameters: *miny=0*, *maxy=*, *minz=0.5*, *maxz=1.5*, *step_size=0.1*, *final_time=90*, *num_turn_waypoints=5*, *line_spacing=0.5*

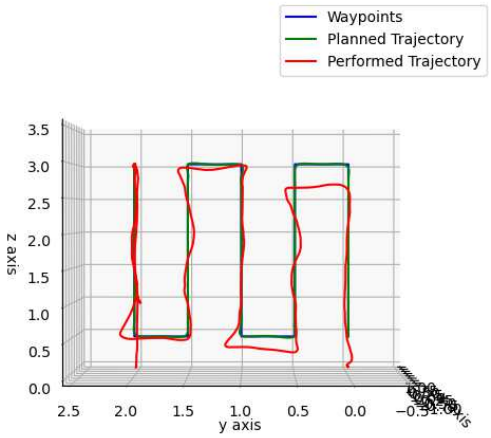
6.2 Implementation 2

6.2.1 Lawnmower trajectory

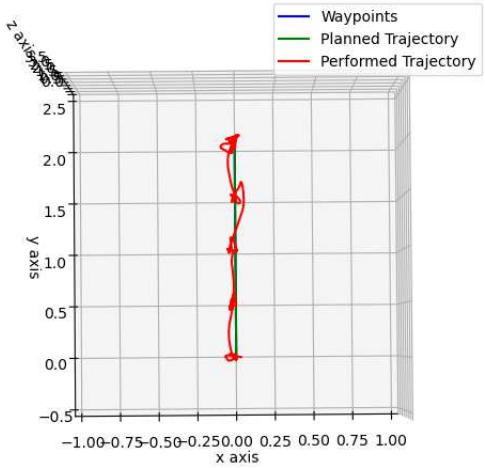
Testing cascade PID controller was conducted on lawnmower trajectory, generated from waypoints with slightly different parameters. Computed waypoints, calculated trajectory and perfomed trajectory graphs are shown in graphs below.



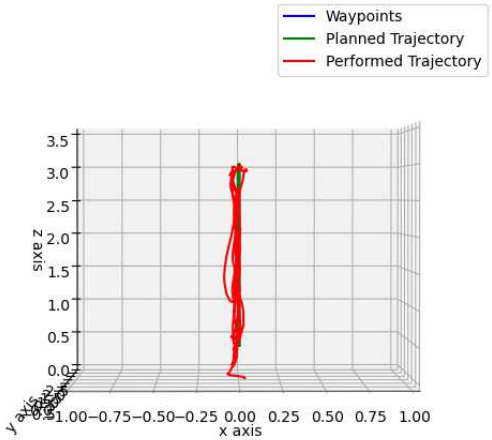
3D Plot



(a) $y - z$ axis Plot

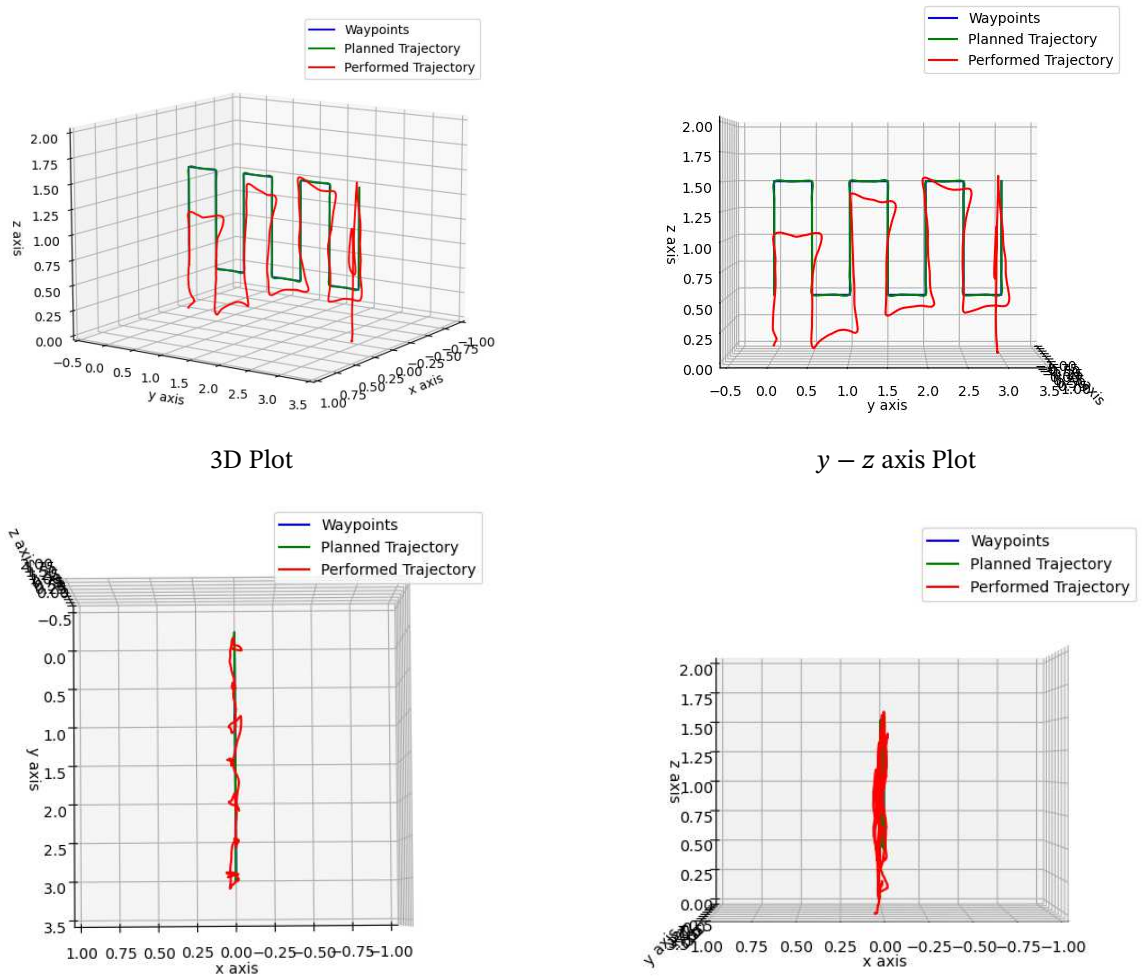


$x - y$ axis Plot



$x - z$ axis Plot

Figure 6.5: Plots of lawnmower trajectory in different planes for parameters: *miny=0*, *maxy=2*, *minz=0.5*, *maxz=3*, *step_size=0.1*, *final_time=90*, *num_turn_waypoints=5*, *line_spacing=0.5*



3D Plot

$y - z$ axis Plot

$x - y$ axis Plot

$x - z$ axis Plot

Figure 6.6: Plots of lawnmower trajectory in different planes for parameters: *miny=0*, *maxy=2*, *minz=0.5*, *maxz=3*, *step_size=0.1*, *final_time=90*, *num_turn_waypoints=5*, *line_spacing=0.5*

6.2.2 Analysis

In all implementations, generated trajectories would follow given waypoints accurately, but the actual performed trajectory differed according to the implementation. In the first implementation, the performance of the spiral trajectory was much more accurate compared to the lawnmower trajectory. In the Z-axis, there is a 20 cm downward shift, i.e., in the negative direction of the Z-axis. In addition to that, the performed trajectory path has small deviations from the generated trajectory, which are not larger than 10cm. Upon analysis of the lawnmower trajectory, the same deviations can be observed as in the spiral trajectory plot. These deviations occurred due to the physical constraints of the drone. Since this implementation is high-level control, it relies on Crazyflies low-level hardware control, and no improvments can be made. In the second implementation,

deviations in the x axis are the largest, just as was the case in the first implementation. Shifts in x axis are also around 10cm, just like in the first implementation. In this implementations, shift in z axis does not exist, but response time of regulator for this axis is clearly visible. Through additional observing, when sending UAV to exact refrence, it could be concluded that the response is slow because it takes a long time to reach the desired position. This system behavior is also reflected in the gradual increase in height in the lawnmower trajectory. With additional PID parameters tunning, there is a posibility of better system performance, and faster response time for thrust regulator.

6.2.3 Photos

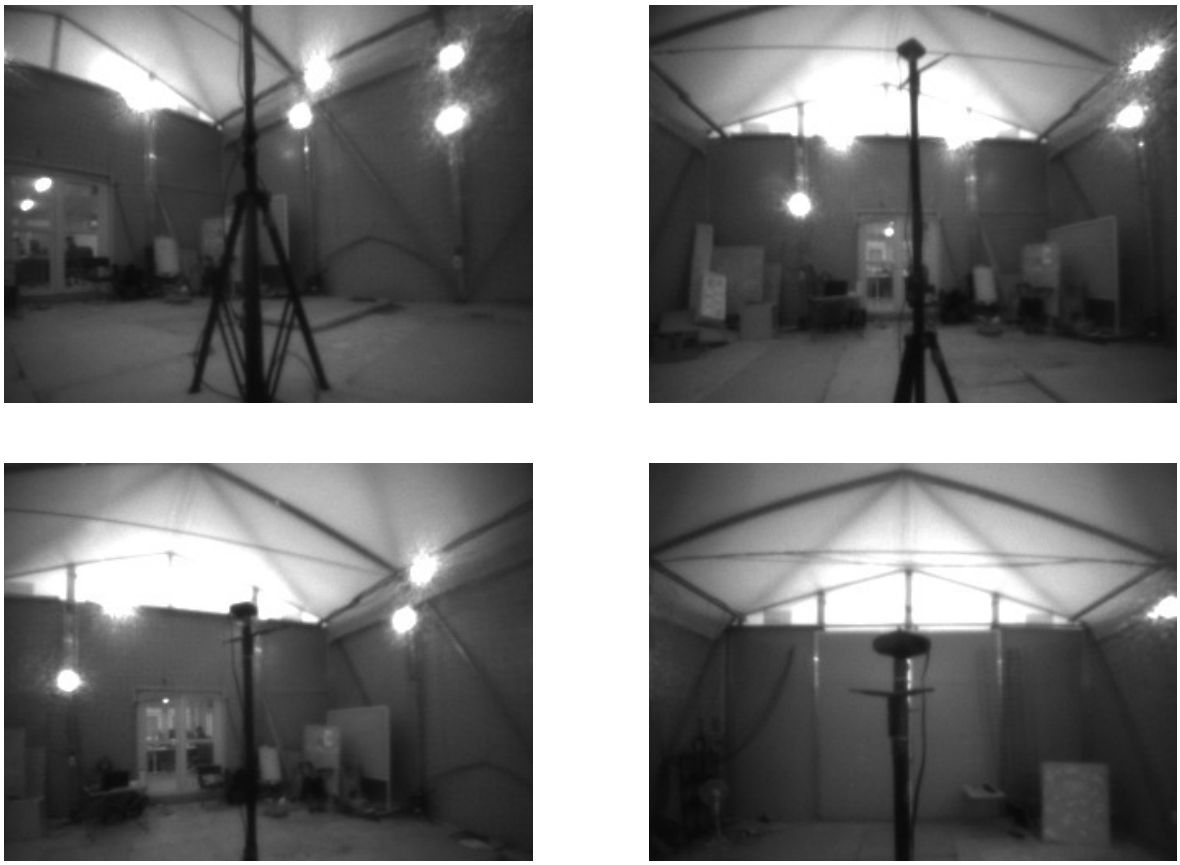


Figure 6.7: Photos taken with spiral trajectory

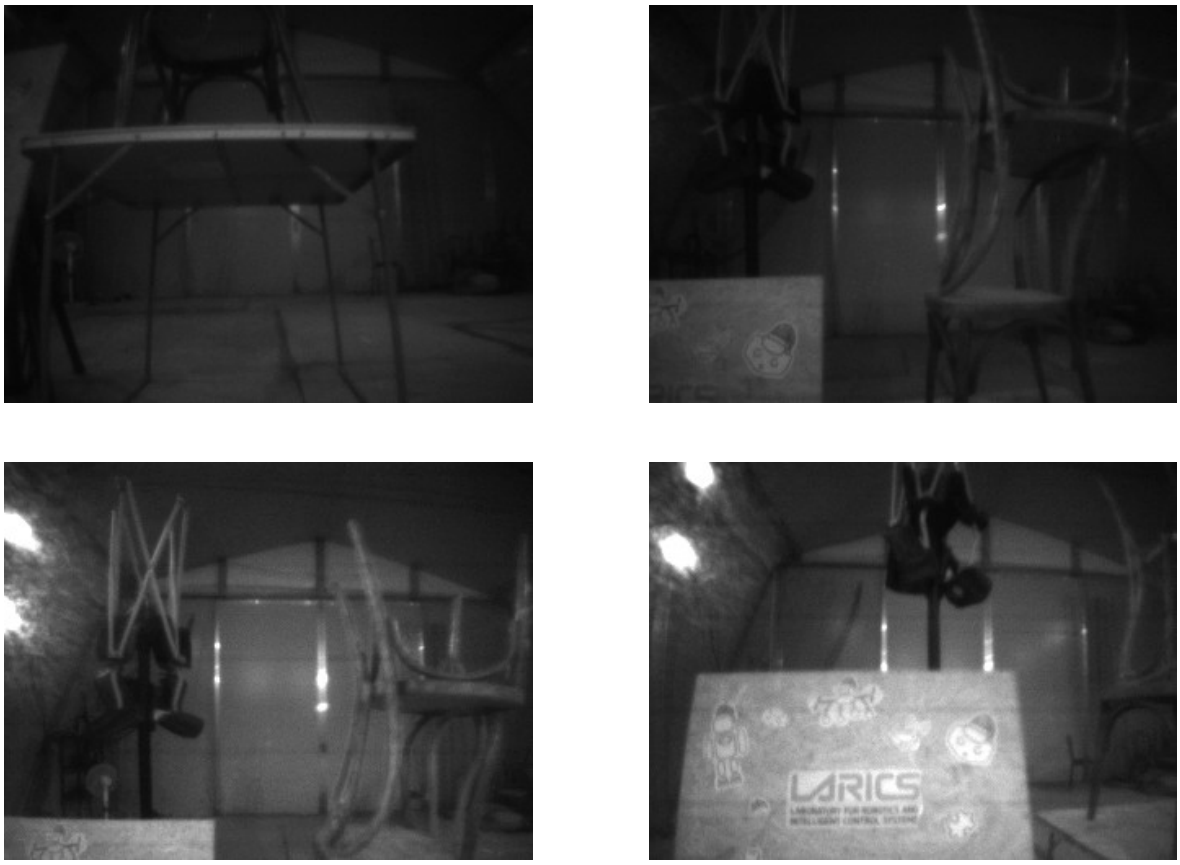


Figure 6.8: Photos taken with lawnmower trajectory

7 Conclusion

This thesis explored the development and implementation of a trajectory generation system for drone-based inspection tasks, addressing the growing need for efficient and safe data collection in various industries. This study presents two different approaches for generating trajectories for the purpose of inspection. The primary goal was to compare trajectory generation methods and evaluate their effectiveness in different scenarios. The chosen trajectory patterns provide efficient coverage patterns suited for distinct applications: the spiral trajectory is ideal for base station inspections, while the lawnmower pattern ensures systematic scanning of large, flat surfaces.

The first implementation was realized using the polynomial method to generate spiral and lawnmower trajectories. Looking at the generated trajectories, it is shown that the system is capable of generating patterns applicable to different inspection scenarios. A graphical user interface was developed, offering a user-friendly way to configure and execute trajectory generation. The second implementation explored the TOPP-RA algorithm for generating trajectories, where a cascade PID control system had to be implemented and tuned to enhance flight stability and accuracy. The results showed that both trajectory generation methods computed a significantly accurate result, whereas small differences in trajectory execution existed, due to different approaches in drone level control.

Although both implementations offer viable solutions for drone-based inspection, there are areas for future research. Further work could focus on incorporating real-time obstacle avoidance, integrating advanced sensor fusion techniques, and exploring adaptive trajectory planning based on environmental conditions. Furthermore, investigating the use of more sophisticated control algorithms, such as model predictive control, could further enhance the precision and robustness of the system.

References

[1] https://www.skydio.com/solutions/energy-and-utilities/distribution-network-inspection, [Online; Accessed: January 2025].

[2] U. O. G. S. J. H. Cosmin Popescu, Ali Mirzazade, "Bridge inspections using unmanned aerial vehicles – a case study in sweden," 2021. [Online]. Available: https://www.diva-portal.org/smash/get/diva2%3A1540949/FULLTEXT02.pdf

[3] A. R. Moreno, "Pid control as a process of active inference applied to a refrigeration system," Master's thesis, Aalborg University, Jun. 2021, master's Thesis, Control and Automation. [Online]. Available: https://projekter.aau.dk/projekter/files/415131289/1034_PID_Control_as_Active_Inference.pdf

[4] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[5] H. Pham and Q. C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, pp. 645 – 659, 06 2018. https://doi.org/10.1109/TRO.2018.2819195

[6] gtfactslab, https://github.com/gtfactslab/CrazySim, [Online; Accessed: January 2025].

[7] bitcraze, https://www.bitcraze.io/products/crazyflie-2-1-plus/, [Online; Accessed: January 2025].

[8] ——, https://www.bitcraze.io/products/crazyradio-2-0/, [Online; Accessed: January 2025].

[9] ——, https://www.bitcraze.io/products/ai-deck/, [Online; Accessed: January 2025].

[10] https://risc.readthedocs.io/1-indoor-flight.html, [Online; Accessed: January 2025].

[11] whoeing, https://github.com/whoenig/uav_trajectories?tab=readme-ov-file, [Online; Accessed: January 2025].

Abstract

Inspection trajectory planning for drones

Mirta Čolić

Unmanned Aerial Vehicles (UAVs) have revolutionized industrial inspections by enabling efficient, high-resolution data collection while minimizing human exposure to hazardous environments. This thesis focuses on developing a system for inspections and exploring different possibilities of trajectory generation. The proposed methodology incorporates mathematical models for trajectory planning, including the polynomial-based method and the TOPP-RA method to ensure smooth and continuous flight trajectory. Additionally, a cascade PID control system is implemented and tuned to enhance flight stability and accuracy. The system is validated through differently shaped trajectories, featuring a graphical user interface for intuitive control and monitoring. The results of the experiment demonstrate the effectiveness of the proposed system in optimizing inspection missions, highlighting improvements in trajectory accuracy and data collection efficiency.

Keywords: UAV; polynomial trajectory; trajectory generation; PID regulator; cascade controller; inspection; CrazyFlie

Sažetak

Planiranje inspekcijske trajektorije bespilotnih letjelica

Mirta Čolić

Bespilotne letjelice (UAV) revolucionirale su industrijske inspekcije omogućujući učinkovito prikupljanje podataka uz minimiziranje izloženosti ljudi opasnim okruženjima. Ovaj rad fokusira se na razvoj sustava za inspekcije te istraživanje različitih mogućnosti generiranja putanja. Predložena metodologija uključuje matematičke modele za planiranje putanja, uključujući metodu zasnovanu na polinomima i TOPP-RA metodu kako bi se osigurala glatka i kontinuirana putanja leta. Osim toga, implementiran je i podešen kaskadni PID sustav upravljanja radi poboljšanja stabilnosti i preciznosti leta. Sustav je testiran kroz različito oblikovane putanje, uz grafičko sučelje za intuitivnu kontrolu i nadzor. Eksperimentalni rezultati pokazuju učinkovitost predloženog sustava u optimizaciji inspekcijskih misija, ističući poboljšanja u točnosti putanja i učinkovitosti prikupljanja podataka.

Ključne riječi: UAV; polinomialna trajektorija; generiranje trajektorije; PID regulator; kaskadni sustav upravljanja; inspekcija; CrazyFlie