Quadrotor Landing on a Moving Platform

Thesis submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Bachelor of Technology
In
Electronics and Communication Engineering

by

Poojan Gandhi (BT20ECE038)
Prajyot Jadhav (BT20ECE046)

under the guidance of
Dr. Anamika Singh

Department of Electronics and Communication Engineering
Visvesvaraya National Institute of Technology, Nagpur
Nagpur 440 010 (India)
May 2024

Quadrotor Landing on a Moving Platform

Thesis submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Bachelor of Technology
In
Electronics and Communication Engineering

by

Poojan Gandhi (BT20ECE038)
Prajyot Jadhav (BT20ECE046)

under the guidance of
Dr. Anamika Singh

Department of Electronics and Communication Engineering
Visvesvaraya National Institute of Technology, Nagpur
Nagpur 440 010 (India)
May 2024

Department of Electronics and Communication Engineering
Visvesvaraya National Institute of Technology, Nagpur

Declaration

We, **Poojan Gandhi** and **Prajyot Jadhav** hereby declare that this project work titled **"Quadrotor Landing on a Moving Platform"** is carried out by us in the Department of Electronics and Communication Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution/University.

Poojan Gandhi (BT20ECE038)
Prajyot Jadhav (BT20ECE046)
Date: 09 May 2024

Certificate

This is to certify that the project titled **"Quadrotor Landing on a Moving Platform"**, submitted by **Poojan Gandhi** and **Prajyot Jadhav** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering, VNIT Nagpur. The work is comprehensive, complete, and fit for final evaluation.

Dr. Anamika Singh
Assistant Professor
VNIT Nagpur

Dr. A. Kothari
Head of Department
Electronics and Communication Engineering,
VNIT Nagpur
Date: 09 May 2024

Abstract

We propose a framework that enables UAVs to autonomously land on moving targets. It utilises real-time positional data of the target obtained from either a motion-capture system or GPS, and uses onboard sensors, including GPS and IMU, to estimate the state of both the UAV and the platform. A robust control architecture utilizing PID and linear quadratic regulator (LQR) techniques is implemented to ensure precise tracking of the moving platform during the landing maneuver. However, model-based control algorithms like the Linear Quadratic Regulator encounter challenges in stabilising quadrotors when their states deviate from equilibrium. This challenge arises from the complex nonlinear dynamics of quadrotors and the limitations of model linearization around equilibrium points. Reinforcement Learning (RL) provides a theoretical framework for continuously improving the quadcopter's landing behavior through trial and error. By iteratively exploring different control policies and observing their outcomes, the quadcopter can refine its landing strategies over time, leading to better performance and adaptability in varying scenarios. Deep RL (DRL)-based controllers are inherently robust to uncertainties and disturbances, thanks to their ability to generalize from diverse training scenarios. This robustness is particularly valuable in real-world applications where environmental conditions may be uncertain or unpredictable, ensuring reliable performance during landing maneuvers. Quadcopter landing on a moving platform involves complex dynamics and nonlinearities that may be challenging to model accurately. DRL excels in handling such complexity by learning directly from raw sensor data, bypassing the need for explicit modeling or linearization of the system dynamics. With the goal to ensure reliability and handle complex dynamics of the quadcopter for the task of landing on the moving platform, a robust DRL network using the PPO algorithm is trained. We validate the effectiveness and reliability of our methodology by conducting simulation and hardware experiments. In these experiments, our method showcases its capability to effectively land the UAV for more complex trajectories of the moving platform. Through the simulation and hardware results, we confirm its capability to achieve autonomous landings on moving platforms.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Overview

Autonomous Unmanned Aerial vehicles (UAVs) are gaining increasing prominence in various industries due to their adaptability and rapid deployment capabilities. They have proven to be highly valuable in a wide range of applications, including capturing aerial images for topographical analysis and agricultural purposes, serving as essential first responders during search and rescue missions, and assisting in comprehensive surveying and mapping tasks. UAVs can also work with delivery trucks to shorten package delivery times. However, UAVs have a limited flight time.

The majority of presently employed UAVs typically utilise rechargeable lithium polymer (LiPo) batteries, which offer superior energy density compared to alternative battery technologies. However, it is important to note that these LiPo batteries still exhibit a relatively restricted flight endurance, typically lasting around 10 to 20 minutes at their maximum capacity. To address these challenges, various solutions are available. Possibilities include integrating a tether for power transfer or enabling automatic deployment and recovery from a charging station. Tethered flights using a power transmission line, which could potentially offer extended flight endurance but their operational range is inherently constrained by the length of the power line. Enabling autonomous take-off and landing from a platform equipped with a device for charging or switching batteries presents an appealing option for scenarios requiring sustained flight operations over an extended operational range. UAVs and ground vehicles working together would therefore be very beneficial.

We present a system designed to enable the landing of a quadrotor on a moving target using real-time positional data of the target, acquired through either a motion-capture system or GPS. The control strategies implemented include PID and LQR,

considering non-aggressive manoeuvres of the quadrotor system. We validate our approach through comprehensive simulations conducted across various scenarios.

Figure 1.1: Quadrotor Landing on a Moving Platform

1.2 Literature Survey

Autonomous landing of quadcopters onto moving platforms presents a challenging problem in robotics and autonomous systems. Traditional control methods often struggle to adapt to the dynamic and uncertain nature of the task. In recent years, Deep Reinforcement Learning (DRL) has emerged as a promising approach to address this challenge, enabling quadcopters to learn adaptive landing strategies through interaction with the environment. This literature survey explores recent advancements and key works in the application of DRL methods for quadcopter landing on moving platforms. Traditional approaches to quadcopter landing typically rely on predefined control strategies and trajectory planning algorithms. These methods often require accurate models of the quadcopter dynamics and platform motion, making them less effective in dynamic and unpredictable scenarios. Furthermore, they may struggle to handle disturbances and uncertainties inherent in real-world environments.

DRL offers a data-driven alternative to traditional control methods, allowing quadcopters to learn landing policies directly from experience. By formulating the landing task as a reinforcement learning problem, researchers have developed innovative approaches to train quadcopter agents for autonomous landing on moving platforms. Following are few of the existing works for quadrotor landing on moving platform utilizing traditional control based approaches as well as Reinforcement Learning based

approaches:

- "Vision-based autonomous quadrotor landing on a moving platform"(D. Falanga et al., 2017) demonstrated drone landing on a mobile platform using only onboard sensors and computational power. An Extended Kalman Filter (EKF) is used to deal with missing visual detection and to estimate the platform's full state. Trajectories are generated by minimising the jerk and are optimal with respect to a cost function based on the energy necessary to execute it. The quadrotor is directed along the desired trajectory by a nonlinear controller which gives the position and attitude.

- Another approach, as explored in "Fully autonomous micro air vehicle flight and landing on a moving target using visual-inertial estimation and model-predictive control"(Tzoumanikas, Dimos et al., 2018) employed Model Predictive Control (MPC) for drone landing on a moving platform, with an EKF tracking the landing target. Utilizing a simplified dynamics model due to slow target rotations, it used a static reference for prediction. Penalizing deviations from both position and estimated target velocity, the approach ensured precise control during landing manoeuvres.

- Similarly, "Autonomous Landing of a UAV on a Moving Platform Using Model Predictive Control"(Paris, Aleix et al., 2019) utilized MPC, equipping the platform with GPS, IMU, and Wi-Fi for real-time data transmission. Leveraging AprilTag markers for visual tracking and a Kalman Filter for position estimation, it demonstrated another effective approach for drone landing on a mobile platform.

- "Dynamic Landing of an Autonomous Quadrotor on a Moving Platform in Turbulent Wind Conditions"(Feng, Y. et al., 2018) adopted a trajectory planning strategy focusing initially on jerk minimization for efficient tag acquisition, later adapting to minimize time spent in turbulent zones as disturbances increased. It incorporated a nonlinear ancillary sliding control to manage disturbances due to wind near the surface of the landing platform, alongside an EKF for accurate platform state estimation and visual detection for enhanced UAV-UGV pose estimation.

- "Learning to Fly by Crashing: A Deep Reinforcement Learning Approach for Autonomous Quadrotor Navigation" (Abbeel et al., 2016) This seminal work by Abbeel et al. introduces a pioneering approach to quadcopter navigation

using Deep Reinforcement Learning (DRL). The paper presents a framework where a quadcopter learns to navigate autonomously through environments by learning from trial and error, i.e., by "learning to fly by crashing." The authors formulate the navigation task as a reinforcement learning problem, where the quadcopter agent learns to map sensory inputs (such as camera images or IMU readings) to control actions (e.g., throttle, roll, pitch, yaw). The agent interacts with the environment, receiving feedback in the form of rewards or penalties based on its actions' outcomes. By iteratively exploring different control policies and observing their consequences, the quadcopter learns effective navigation strategies, enabling it to avoid obstacles and reach designated goals. While the focus of this paper is not specifically on landing, the principles and methodologies introduced lay the groundwork for subsequent research in DRL-based quadcopter control tasks.

- "Autonomous Navigation and Landing of a Quadrotor UAV on a Moving Target Using Reinforcement Learning" (Dang et al., 2020) In this paper, Dang et al. address the problem of autonomous quadcopter landing on a moving target using Deep Reinforcement Learning (DRL). The authors formulate the landing task as a Markov Decision Process (MDP), where the quadcopter agent learns a policy to land on a moving platform by maximizing cumulative rewards. They employ a Deep Q-Network (DQN) architecture to approximate the action-value function, allowing the agent to efficiently learn landing policies from experience. The training process involves simulating quadcopter-agent interactions, where the agent receives feedback based on the distance between the quadcopter and the moving platform. Through extensive experimentation in simulation environments, the authors demonstrate the effectiveness of their DRL approach in achieving accurate and robust landings on moving targets.

- "Deep Reinforcement Learning for Autonomous Quadrotor Landing: A Transfer Learning Approach" (Li et al., 2021) Li et al. propose a transfer learning approach to quadcopter landing on moving platforms using Deep Reinforcement Learning (DRL). Building upon previous works, the authors leverage pre-trained models and fine-tuning techniques to accelerate learning and improve generalization across different landing scenarios. The paper introduces a transfer learning framework where a pre-trained DRL model, initially trained on simpler landing tasks or environments, is adapted to new landing scenarios through additional training on target-specific data. By transferring knowledge from the pre-trained model and fine-tuning its parameters, the quadcopter

agent can quickly adapt to new landing conditions with minimal training data. Experimental results showcase the efficacy of the transfer learning approach in achieving fast and reliable landings in diverse environments, highlighting its potential for real-world applications where adaptability and scalability are crucial.

Despite the promising advancements in DRL-based quadcopter landing, several challenges remain to be addressed. These include the need for robustness to disturbances, scalability to real-world environments, and generalization across varying conditions. Future research efforts may focus on developing hybrid approaches that combine DRL with traditional control methods, as well as exploring novel algorithms and architectures tailored to the unique requirements of quadcopter landing tasks.
In conclusion, DRL methods offer a promising avenue for tackling the complex task of quadcopter landing on moving platforms. By leveraging reinforcement learning techniques, researchers have made significant strides in enabling quadcopters to autonomously navigate, track, and land with precision and reliability. Continued research and innovation in this area hold the potential to unlock new capabilities and applications for autonomous aerial systems in diverse real-world scenarios.

Chapter 2

Reinforcement Learning

2.1 Overview

Reinforcement Learning (RL) represents a powerful paradigm within the broader field of artificial intelligence, focusing on learning optimal decision-making strategies through interaction with an environment. This overview explores the foundational principles and methodologies that underpin Reinforcement Learning, providing insights into its theoretical underpinnings, key algorithms, applications, challenges, and future directions. RL focuses on agents learning via trial and error and receiving feedback in the form of rewards or punishments for their behavior. By repeatedly exploring and exploiting the environment, agents attempt to maximise cumulative reward over time, ultimately acquiring robust and adaptive policies. This overview serves as a gateway to understanding the multifaceted landscape of Reinforcement Learning, offering a comprehensive examination of its principles, techniques, and real-world implications.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning in which an agent learns to make consecutive decisions in an environment to achieve specific goals. RL, as opposed to supervised learning, which learns from labeled data, and unsupervised learning, which infers patterns from unlabeled data, is based on the trial and error concept. Through interactions with the environment, the agent receives feedback in the form of incentives or penalties, allowing it to develop the best decision-making strategies. This paper provides an introductory overview of RL, discussing its fundamental concepts, key components, and applications.

Reinforcement Learning (RL) studies how agents should behave in a given environment to maximize a concept known as cumulative reward. Because RL can handle sequential decision-making problems in a variety of fields, such as robotics, gaming, finance, and healthcare, it has attracted a lot of attention. The idea of learning through interaction, in which agents discover optimal actions by repeatedly investigating and taking advantage of their surroundings, is at the core of reinforcement learning.

A reinforcement learning problem has the following fundamental components: an agent, an environment, actions, states, rewards, and a policy. Formally, an RL problem can be modeled as a Markov Decision Process (MDP), defined by a tuple (S, A, P, R, γ), where: S is the set of possible states, A is the set of possible actions, P represents the transition dynamics describing the probability of transitioning from one state to another given an action, R denotes the reward function, and γ is the discount factor representing the present value of future rewards. Through the application of its policy, which relates states with actions, the agent interacts with its surroundings. The agent's learning process is guided by feedback it receives in the form of incentives from this interaction.

The value function, which calculates the predicted cumulative reward of being in a certain state or performing a specific action, is a key element of reinforcement learning algorithms. The value function is convertible into two different forms: the action-value function $(Q(s, a))$, which calculates the value of performing a specific action in a given state under a given policy, and the state-value function $(V(s))$, which calculates the value of being in a given state under a particular policy.

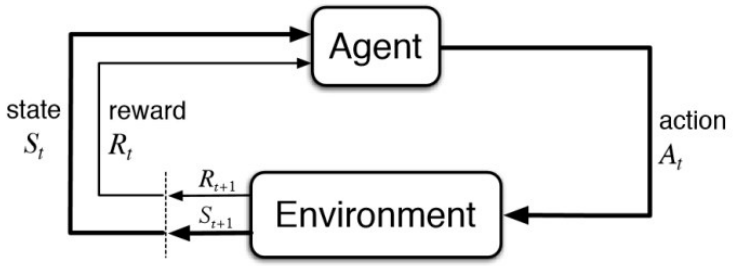

Figure 2.1: Working of Reinforcement Learning

Policy

A policy π is a mapping from states to actions, i.e., $\pi : S \to A$. The goal of RL is to find the optimal policy, denoted as π^*, which maximizes the expected cumulative

reward, often represented as the expected return

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.1}$$

where R_{t+k+1} is the reward obtained at time step $t + k + 1$.

Value Functions

For RL to assess the quality of states and state-action pairs, value functions are essential. The projected return beginning from state s and adhering to policy π is represented by the state value function, frequently referred to as $V^\pi(s)$. The expected return is represented by the action value function, $Q^\pi(s, a)$, which starts at state s, takes action a first, and then applies policy π.

2.2.1 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines reinforcement learning with deep learning techniques, leveraging deep neural networks to approximate complex functions involved in decision-making processes. With its ability to handle high-dimensional state and action spaces, DRL allows agents to learn directly from unfiltered sensory inputs like sensor data or images.

Policy Gradient Methods

A class of reinforcement learning algorithms known as policy gradient methods works by directly optimizing the policy parameters in order to maximize the expected return. Policy gradient approaches learn policies directly by calculating and updating gradients of the expected return with respect to the policy parameters, in contrast to value-based methods that estimate value functions and derive policies from them.

Proximal Policy Optimization (PPO)

State-of-the-art policy gradient algorithms like Proximal Policy Optimization (PPO) overcome the drawbacks of conventional policy gradient methods, such as instability and sample inefficiency. PPO employs a surrogate objective function and constrains policy updates to ensure more stable and efficient learning. By iteratively collecting data and optimizing the policy parameters, PPO performs at the cutting edge across a variety of reinforcement learning challenges. By optimizing the policy parameters,

PPO aims to maximize the expected return. The objective function is defined as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where:
$\mathcal{L}(\theta)$ is the surrogate objective function.
θ represents the policy parameters.
$\mathbb{E}_t$ denotes the expectation over time steps.
$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the likelihood ratio between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$.
$\hat{A}_t$ is the advantage function, representing the advantage of taking action a_t in state s_t over the baseline value function.
ϵ is a hyperparameter controlling the extent of policy update.

To avoid significant policy changes, PPO uses a clipped surrogate objective to confine the policy update. By minimizing the clipped surrogate objective function, the policy is updated:

$$\theta_{\text{new}} = \arg\min_\theta \mathcal{L}(\theta)$$

PPO offers several advantages over traditional policy gradient methods:

- Stability: By constraining the policy update, PPO ensures stable learning and prevents large policy changes that may lead to divergence.
- Sample Efficiency: PPO efficiently utilizes collected data by reusing samples for multiple policy updates, resulting in improved sample efficiency.
- Compatibility with Deep Learning: PPO is compatible with deep neural networks, allowing for effective learning in high-dimensional state and action spaces.

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

Figure 2.2: Proximal Policy Optimization

Chapter 3

Mathematical Model of the Quadrotor

3.1 Overview

We present the quadrotor's mathematical model in this chapter. This model defines the motion of a quadrotor determined by its inputs. By examining only the four motor speeds, it is possible to estimate and evaluate the positions attained by the quadrotor using these equations. We examine and simplify the quadrotor rigidbody dynamics using small angle assumptions for roll and pitch movement in order to construct the control system at equilibrium point.

3.2 Basic Concepts

In the quadrotor model, four rotors are arranged in a cross configuration. The propellers all have fixed, parallel axes of rotation. The propellers on the left and right turn in a clockwise direction, those at the front and rear spin counterclockwise.
By adjusting the four motors' speeds, the quadrotor's attitude and position can be set to the required values. With only four propellers, although the quadrotor boasts six degrees of freedom, achieving a desired set-point for all DOF remains out of reach, restricted to a maximum of four. A quadrotor is regarded as an underactuated nonlinear complicated system due to its four inputs and six outputs.
The following four fundamental movements of the quadrotor can be identified:

- **Thrust:**
  Thrust is the upward force generated by the quadrotor's rotors.By modulating the rotor speeds, the quadrotor can govern its altitude. When all rotors rotate

at the same velocity, the thrust is balanced, and the quadrotor maintains a steady altitude. Increasing the speed of all rotors simultaneously increases altitude, while decreasing the speed lowers it.

- **Roll Movement**
  Roll refers to the rotation of the quadrotor about its longitudinal axis, which runs from the front to the back of the vehicle. When the quadrotor tilts to the left or right, it rolls about this axis.To execute a roll, the quadrotor elevates the speed of rotors on one side and lowers it on the other side, creating a torque that causes it to roll in the desired direction.

- **Pitch Movement**
  Pitch is the rotation of the quadrotor about its lateral axis, which runs from one side to the other. When the quadrotor tilts forward or backward, it pitches about this axis. Pitch maneuvering necessitates elevating the speed of either the front or rear rotors while reducing the speed of the opposing set, creating a torque that causes it to pitch in the desired direction.

- **Yaw Movement**
  Yaw is the rotation of the quadrotor about its vertical axis, which runs vertically through the center of the vehicle. When the quadrotor rotates clockwise or counterclockwise, it yaws about this axis. To yaw, the quadrotor increases the speed of rotors spinning in one direction (clockwise or counterclockwise) while decreasing the speed of rotors spinning in the opposite direction, creating a torque that causes it to yaw.

3.2.1 Euler Angles

Euler angles are a fundamental concept in describing the orientation of a rigid body or frame of reference in three-dimensional space. Introduced by the mathematician Leonhard Euler, they provide a concise way to represent how an object is rotated relative to a reference frame. Specifically, we'll use ZYX Euler angles, a common convention used to describe these orientations.
In the ZYX convention, three angles—often denoted as ϕ, θ, and ψ are used to specify the orientation of the quadcopter. These angles are typically constrained within certain ranges: $\phi \in [-\pi, \pi]$, $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, and $\psi \in [-\pi, \pi]$. Following is a detailed explanation of each angle.

- ϕ (Roll): This angle represents the rotation about the body's X-axis. It corresponds to a rotation in the horizontal plane, similar to the motion of rolling a

rigid body along its longitudinal axis.

- θ (Pitch): θ represents the rotation about the body's Y-axis. It describes the tilting motion of the body, similar to how an aircraft pitches its nose up or down.

- ψ (Yaw): Yaw is the rotation about the body's Z-axis. It indicates the horizontal rotation around the vertical axis, as if the body were turning left or right.

These angles collectively describe a sequence of rotations starting from a known standard orientation. By composing these elemental rotations in a specific order, any arbitrary orientation of the rigid body can be achieved.

3.3 System Modelling

The dynamics of the quadrotor are described with reference to two frames: an inertial frame denoted $\{E\}$ and a body-fixed frame denoted $\{B\}$, as illustrated in the diagram.

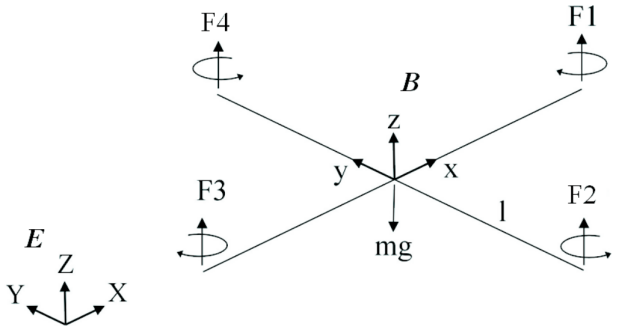

Figure 3.1: Inertial and Body-Fixed frames of reference

Using the Newton-Euler method, the dynamic model of the quadrotor is established. The following assumptions form the basis of the quadrotor modeling:

- The quadrotor's design is assumed to be rigid and symmetrical.

- The center of gravity coincides with the body-fixed frame's origin.

- The propellers are characterized as rigid, and

- The thrust and drag forces are proportional to the square of the propeller speed.

Rotation about the x, y, and z axes is referred to as roll, pitch, and yaw, respectively. The quadrotor configuration is represented by the position of the centre of mass in the inertial frame $\begin{bmatrix} x & y & z \end{bmatrix}^T$, the linear velocities in the inertial frame $\begin{bmatrix} u & v & w \end{bmatrix}^T$, the orientation in the inertial frame represented by the Euler angles $\begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ and the angular velocities in the body-fixed frame $\begin{bmatrix} p & q & r \end{bmatrix}^T$.

3.3.1 Actuator Dynamics

Even though the quadrotor possesses six degrees of freedom, only four actuators are engaged to produce control inputs: $\mathbf{F} = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 \end{bmatrix}^T$. Here, F_1 signifies the total thrust, while F_2, F_3, and F_4 correspond to the rolling, pitching, and yawing moments, respectively. The relationship between the control inputs and the rotational speeds of the propellers is expressed as follows:

$$\left.\begin{aligned} F_1 &= A(\Omega_1 + \Omega_2 + \Omega_3 + \Omega_4) \\ F_2 &= AL(-\Omega_1 - \Omega_2 + \Omega_3 + \Omega_4) \\ F_3 &= AL(\Omega_1 - \Omega_2 - \Omega_3 + \Omega_4) \\ F_4 &= K(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \end{aligned}\right\} \tag{3.1}$$

Here, L denotes the distance between a rotor and the center of the quadrotor, A and K represent the thrust and drag coefficients respectively. And, $\Omega_1, \cdots, \Omega_4$ denote the angular speeds of the propellers.

3.3.2 Quadrotor Dynamic Model

The state vector of the system is represented by $\mathbf{X} = \begin{bmatrix} x & y & z & u & v & w & \phi & \theta & \psi & p & q & r \end{bmatrix}^T$. The non-linear dynamics describing the behavior of the quadrotor are represented by the following equations:

$$
\left.
\begin{aligned}
\dot{x} &= u \\
\dot{y} &= v \\
\dot{z} &= w \\
\dot{\phi} &= p + (\sin\phi\tan\theta)q + (\cos\phi\tan\theta)r \\
\dot{\theta} &= q\cos\phi - r\sin\phi \\
\dot{\psi} &= \frac{\sin\phi}{\cos\theta}q + \frac{\cos\phi}{\cos\theta}r \\
\dot{u} &= -(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi)\frac{F_1}{m} \\
\dot{v} &= -(-\sin\phi\cos\psi + \cos\phi\sin\theta\cos\psi)\frac{F_1}{m} \\
\dot{w} &= -(\cos\phi\cos\theta)\frac{U_1}{m} + g \\
\dot{p} &= \frac{I_y - I_z}{I_x}qr + \frac{F_2}{I_x} \\
\dot{q} &= \frac{I_z - I_x}{I_y}pr + \frac{F_3}{I_y} \\
\dot{r} &= \frac{I_x - I_y}{I_z}pq + \frac{F_4}{I_z}
\end{aligned}
\right\} \tag{3.2}
$$

In these equations, m represents the mass, g denotes the acceleration due to gravity, and I_x, I_y, and I_z signify the moments of inertia around the three axes.

3.3.3 State Space Model

The quadrotor's dynamic model can be simplified to a linear form when operating near hover, provided the assumption of minimal attitude angles, with the yaw angle ψ considered as zero. The vector of the states is $\mathbf{X} = \begin{bmatrix} x & y & z & u & v & w & \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$. The state space representation is articulated as follows:

$$
\begin{aligned}
\dot{x} &= Ax + Bu \\
y &= Cx + Du
\end{aligned}
$$

In this equation, y denotes the output vector, A represents the system matrix, B stands for the input matrix, C signifies the output matrix, and D denotes the feed-forward matrix. After linearization of the model described in (3.2) we get,

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \text{-g} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \text{g} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \tag{3.4}$$

Chapter 4

Proposed Approach

4.1 Overview

In this chapter we discuss three strategies used for drone landing on the mobile platform. Two control strategies include a nested PID (Proportional-Integral-Derivative) controller and an LQR (Linear Quadratic Regulator)-based controller, whereas the third approach is Reinforcement Learning based.

4.2 PID controller

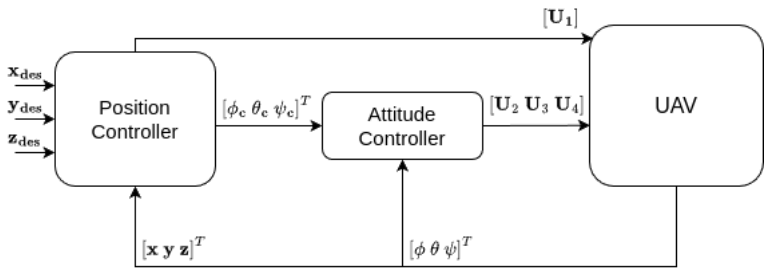

Figure 4.1: PID Control Architecture

The classical PID controller is characterized by its simple structure and ease of implementation. In our contol architecture, as shown in Fig. 4.1, we utilize nested loops, with the inner loop dedicated to attitude control and the outer loop responsible for position control. Specifically, the attitude loop is nested within the position loop. From the feedback of the current attitude and angular rates, we derive the control inputs F_2, F_3 and F_4. Particularly in the equations-of-motion outlined later, we compute the value of these control inputs based on the desired attitude. In the outer-loop, we receive the desired position and yaw. Subsequently, we compare these

with the actual position and velocity data. Through this comparison, we determine F_1. The thrusts and torques are obtained from the error values as shown in the following equations:

$$
\begin{aligned}
x_c &= K_{x,P}(e_x) + K_{x,D}\dot{e}_x + K_{x,I}\int e_x \\
y_c &= K_{y,P}(e_y) + K_{y,D}\dot{e}_y + K_{y,I}\int e_y \\
\phi_c &= \frac{1}{g}(x_c \sin\psi_{des} - y_c \cos\psi_{des}) \\
\theta_c &= \frac{1}{g}(x_c \cos\psi_{des} + y_c \sin\psi_{des}) \\
\psi_c &= \psi_{des}
\end{aligned}
$$

$$
\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} m(g + K_{z,P}(e_z) + K_{z,D}\dot{e}_z + K_{z,I}\int e_z) \\ K_{\theta,P}(\theta_c - \theta) + K_{\theta,D}(\dot{\theta}_c - \dot{\theta}) + K_{\theta,I}\int(\theta_c - \theta) \\ K_{\phi,P}(\phi_c - \phi) + K_{\phi,D}(\dot{\phi}_c - \dot{\phi}) + K_{\phi,I}\int(\phi_c - \phi) \\ K_{\psi,P}(\psi_c - \psi) + K_{\psi,D}(\dot{\psi}_c - \dot{\psi}) + K_{\psi,I}\int(\psi_c - \psi) \end{bmatrix} \tag{4.1}
$$

where $e_x = x_{des} - x$ is the error in the x-direction. Analogous expressions apply for other dimensions.

To further enhance stability and response, the twelve parameters $K_{x,P}$, $K_{x,D}$, $K_{x,I}$, $K_{y,P}$, $K_{y,D}$, $K_{y,I}$, $K_{z,P}$, $K_{z,D}$, $K_{z,I}$, $K_{\theta,P}$, $K_{\theta,D}$, $K_{\theta,I}$, $K_{\phi,P}$, $K_{\phi,D}$, $K_{\phi,I}$, $K_{\psi,P}$, $K_{\psi,D}$, and $K_{\psi,I}$ are carefully selected through rigorous testing and simulation. These parameters play a crucial role in shaping the response characteristics of the controller and ensuring optimal performance across various operating conditions.

State Machine

The system comprises of three states: takeoff, tracking and landing. We provide a more thorough explanation of each of these stages in the sections that follow.

1. *Takeoff:* The drone initially takes off and stabilizes into a hover position. Upon reaching this hover point, it is commanded to track the moving platform and the state machine is switched to *tracking* mode.

2. *Tracking:* During this phase, the quadrotor receives the current position of the moving platform as the desired state. Its objective is to track and maintain a position above the moving platform. The desired position is continuously updated at each time instant. Depending on the control method employed (such

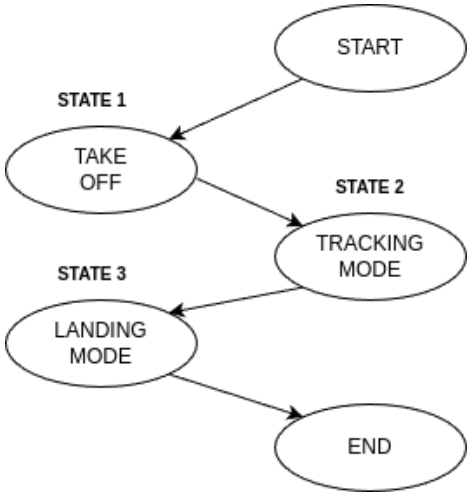

Figure 4.2: The flowchart of the state machine

as PID or LQR), the control inputs are calculated based on the error values. Subsequently, these control inputs are applied to the quadrotor. This phase concludes when the quadrotor successfully flies above the moving platform, and the error in position values falls below a predefined tolerance threshold.

3. *Landing:* Upon entering this phase, the quadrotor switches its operational mode to landing, initiating its descent toward the moving platform. Employing a vertical descent approach, the quadrotor gradually lowers itself onto the platform's surface. Upon achieving a successful landing, the quadrotor disarms itself, bringing the rotation of its motors to a halt, thereby concluding the landing operation.

4.3 Linear Quadratic Regulator Control

The objective of optimal control involves identifying the control signal that enables effective system management while adhering to physical constraints and optimizing a performance metric. In essence, solving an optimization challenge aims to reduce a specified cost while aligning the system's state $x(t)$ with the desired trajectory x_d. Additionally, it aims to optimize actuator utilization by minimizing control input usage.
The fundamental requirements for optimization control are as follows:

- A cost index J that considers the designer's goals and specifications.

- Potential boundary conditions and physical restrictions constraining the system.
- A model that can best represent the behavior of the dynamic system.

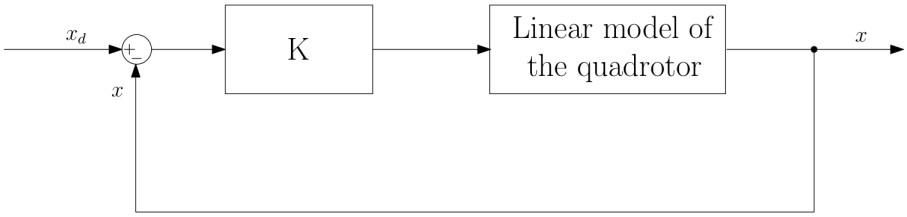

Figure 4.3: LQR controller

LQR provides optimally controlled feedback gains. It enables the design of closed-loop systems characterized by stability and high performance. For the LTI system:

$$\dot{x} = Ax + Bu$$

The cost function to be minimized and the control inputs are given by:

$$J = \int_0^\infty (\tilde{x}^T \mathbf{Q} \tilde{x} + \tilde{u}^T \mathbf{R} \tilde{u}) dt$$

$$u = u_{des} + K(x - x_{des})$$

In this formulation, $\mathbf{Q}$ represents the positive semi-definite state weighting matrix, and $\mathbf{R}$ indicates the positive-definite cost weighting matrix. The terms $\tilde{x} = x - x_{\text{des}}$ and $\tilde{u} = u - u_{\text{des}}$ represent the errors in state and input, respectively, where x_{des} and u_{des} denote the reference state and input. Additionally,

$$K = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}$$

where $\mathbf{P}$ is the solution of the Continuous ARE:

$$0 = \mathbf{PA} + \mathbf{A}^T\mathbf{P} - \mathbf{PBR}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{Q}$$

4.4 Reinforcement Learning Approach

We employed the gym-pybullet-drone environment as a foundational component for training a Deep Reinforcement Learning (DRL) model utilizing the Proximal Pol-

icy Optimization (PPO) algorithm. Our primary objective was to develop an autonomous landing system for drones onto a dynamically moving platform. Leveraging the capabilities of gym-pybullet-drone, which offers a realistic simulation environment for drone control tasks, we tailored the dynamics and reward structure to closely resemble real-world scenarios. Through iterative experimentation and policy refinement, our model learned effective control strategies, enabling the drone to autonomously navigate, track, and land on the moving platform with precision and reliability. This endeavor showcases the utility of gym-pybullet-drone as a versatile tool for research in autonomous drone navigation and control, while also demonstrating the efficacy of DRL methods, particularly PPO, in addressing complex real-world tasks within simulated environments.

4.4.1 Training Environment

Creating a custom environment in gym-pybullet-drones involves defining the dynamics, rewards, and observation spaces tailored to the specific requirements of the task at hand. To begin, we sub-classed the HoverAviary class and implement methods to specify the drone's dynamics, observation space, action space, and reward function. We included PyBullet's mobile robot, Husky into the environment, which acted as a moving platform upon which the drone aims to land on. The Husky robot was given constant velocity in x and y to fix the trajectory upon which the robot would travel. Further, to ensure that the environment's state space follows the Markov Rule, we added additional information along with the drone's kinematic information.

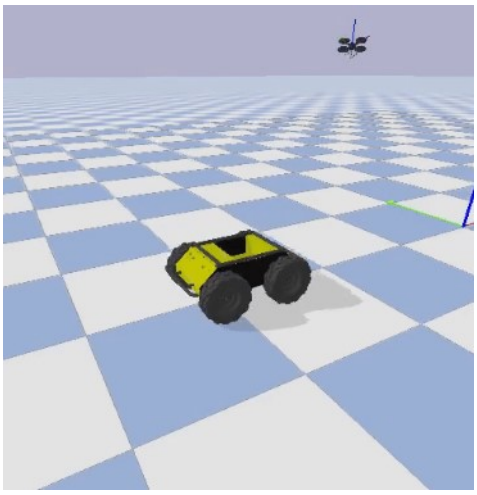

Figure 4.4: Husky and drone in PyBullet Simulation Environment

State Space

The state or the observation space would be designed to capture relevant information about the drone's state, such as its position, orientation, velocity, and sensor readings. Along with the drone readings, we also included the position and velocities of the moving platform in Cartesian coordinate system. The state space thus becomes:

- $S[0:2]$ = Drone's position in x, y, z in World Frame
- $S[3:6]$ = Drone's Quaternion Orientation in World Frame
- $S[7:9]$ = Drone's roll pitch yaw
- $S[10:12]$ = Drone's velocity in x, y, z
- $S[13:15]$ = Drone's angular velocity in x, y, z
- $S[16:19]$ = Drone's motor speed in RPMs
- $S[20:22]$ = Husky's/Moving Platform's position in x, y, z in World Frame
- $S[23:25]$ = Husky's/Moving Platform's velocity in x, y, z in World Frame

Action Space

The action space would consist of commands that control the drone's movement and behavior. The agent control's the drone through the desired torque and thrust values. The action space is

- $action[0]$ = Quadrotor's desired thrust along the z-axis
- $action[1]$ = Quadrotor's desired torque around the x-axis
- $action[2]$ = Quadrotor's desired torque around the y-axis
- $action[3]$ = Quadrotor's desired torque around the z-axis

Reward Function

Finally, the reward function would be formulated to provide feedback on the agent's performance, encouraging desired behaviors while discouraging undesirable actions. We want the desired behaviour of drone to be able to land on the husky's surface. The reward is negative of the square of the Cartesian distance between the drone and the moving platform (husky).

$$reward = -1 * (huskypos - dronepos)^2$$

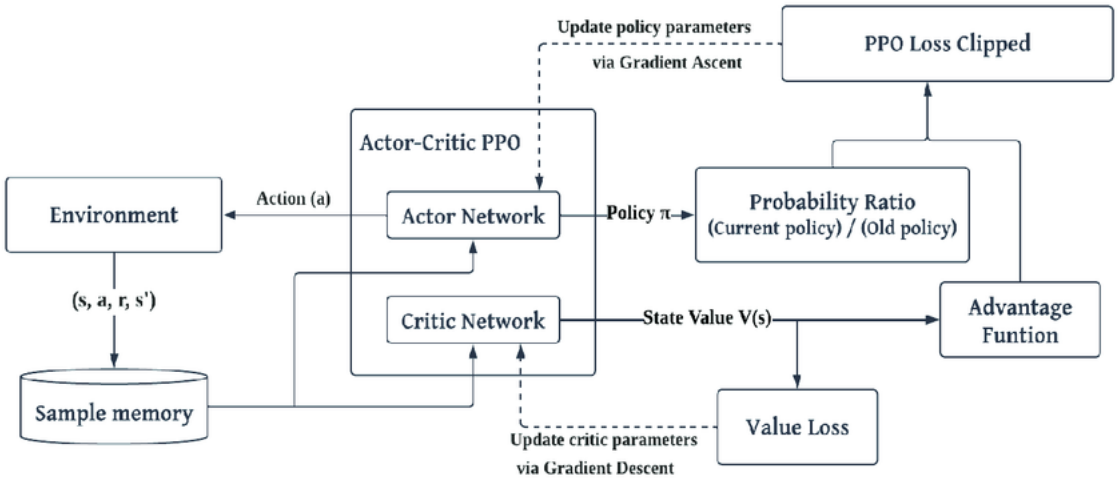

Figure 4.5: PPO Architecture

Chapter 5

Simulation Experiments and Results

5.1 Overview

In this chapter, we present the simulation experimental details and the results achieved by our proposed methods. We briefly explain the simulation softwares and flight stack used. We also provide visualizations and plots to demonstrate the effectiveness of our methodology for quadrotor landings on moving targets.

5.2 Experimental Details

5.2.1 Gazebo Classic Simulator with PX4 and ROS

PX4 Autopilot

The brain of the UAV is known as an autopilot. At its most fundamental level, it comprises of flight controller (FC) hardware running flight stack software on a real-time operating system ("RTOS"). Essential stabilization and safety elements are provided by the flight stack, which also typically offers some pilot aid for manual flight and the automation of routine operations like takeoff, landing, and mission execution. PX4 is a robust open-source autopilot flight stack that operates on the NuttX real-time operating system.

ROS and Gazebo Classic

PX4 can be used with ROS, a general-purpose robotics library, to construct drone applications. In addition to having access to other software libraries created for

Linux, ROS benefits from an active development community that works to solve common robotics difficulties. For evaluating object-avoidance and computer vision algorithms, Gazebo Classic is an excellent 3D simulator designed for autonomous robots.
PX4 transmits motor and actuator values as well as sensor data from the simulated environment with the simulator (such as Gazebo Classic). In order to give and receive commands, it interfaces with the Ground Control Station and an Offboard API (such as ROS).

Software-in-the-Loop Simulations

To validate the proposed algorithm, we performed Software-in-the-Loop (SITL) simulations using the PX4 open-source flight control software.
The ability to perform a SITL simulation to recreate your flight is one of PX4's best features. This is helpful since it allows you to test updated missions or algorithms without actually the quadrotor flight and risking damage. Software in the Loop is a software-only simulation of a system that is modeled and operated and is hardware-free. Before control software is developed and integrated into a real system, it typically needs to pass simulation verifications.

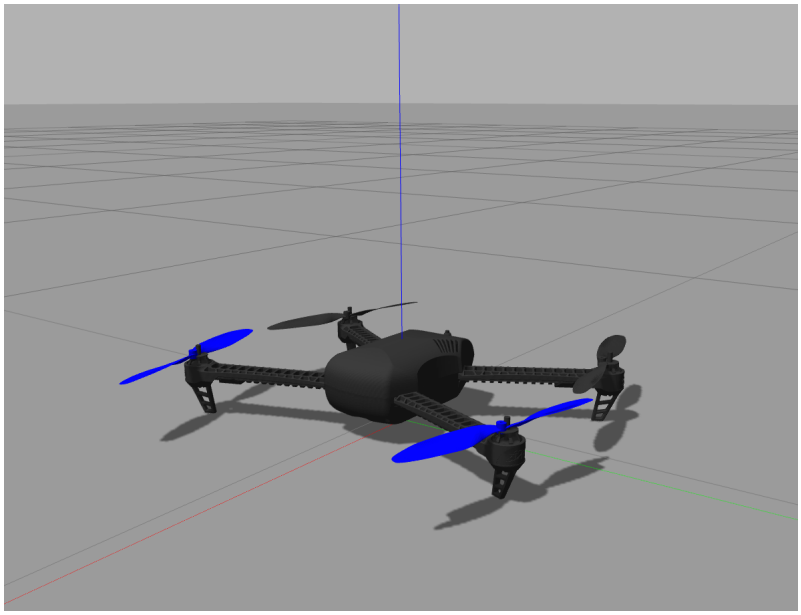

Figure 5.1: 3DR Iris Quadrotor

The simulations were performed with a 3DR Iris quadrotor model simulated in the Gazebo simulator environment. The simulated quadrotor communicates with the PX4 flight control software via the MAVLink protocol, which defines a standardized

set of messages for transmitting sensor data from the simulated environment to PX4, as well as for relaying actuator commands back to the simulated vehicle. To facilitate communication between the simulated quadrotor and the flight controller, we used MAVROS, a ROS package that allows controlling drones via the MAVLink protocol.

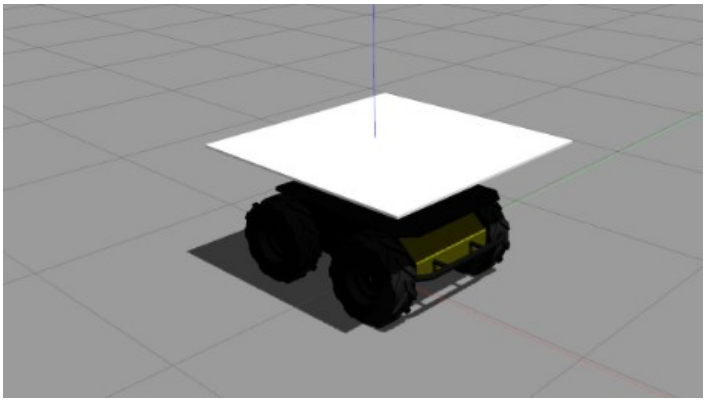

Figure 5.2: Clearpath Husky with a landing platform

Within our simulated environment, we utilized a Clearpath Husky as the ground vehicle. On top of the Husky, we mounted a landing platform measuring 100 cm $\times$ 100 cm.

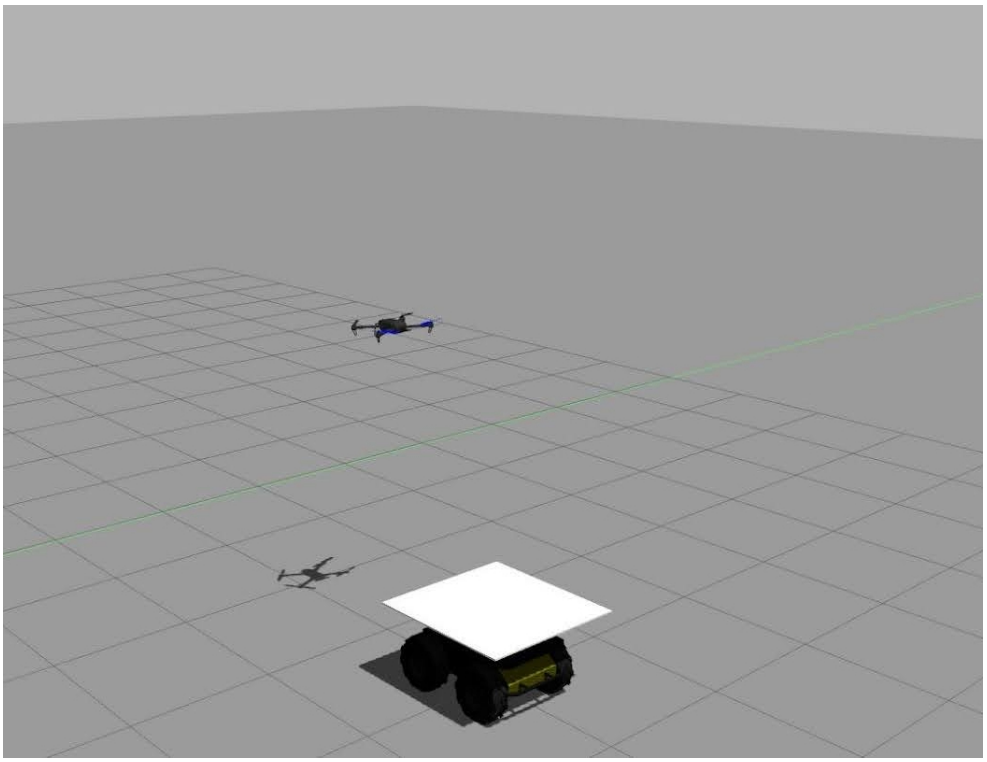

Figure 5.3: Husky and Drone in the Gazebo Classic Simulation Environment

Table 5.1: 3DR Iris quadrotor physical specifications

Parameter	**Value**
m(kg)	1.5
l(m)	0.25
I_x (kg·m^2)	0.029125
I_y (kg·m^2)	0.029125
I_z (kg·m^2)	0.055225

5.2.2 Pybullet Simulator

The gym-pybullet-drone environment provides a simulation platform for training and evaluating reinforcement learning (RL) algorithms in the context of autonomous drone navigation and control. Leveraging the PyBullet physics engine, gym-pybullet-drone offers a realistic and customizable environment for experimenting with various RL techniques aimed at developing robust drone control policies.
The gym-pybullet-drone environment is formulated as a Markov Decision Process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ represents the state space, comprising the drone's position, orientation, velocity, and other relevant physical parameters.
- $\mathcal{A}$ denotes the action space, consisting of commands for controlling the drone's motion, such as throttle, pitch, roll, and yaw.
- $P(s_{t+1}|s_t, a_t)$ is the transition probability function, modeling the dynamics of the drone's movement in response to actions.
- $R(s_t, a_t, s_{t+1})$ is the reward function, providing feedback on the quality of actions taken by the agent.
- $\gamma \in [0, 1]$ is the discount factor, determining the trade-off between immediate and future rewards.

The dynamics of the drone in gym-pybullet-drone are governed by the equations of motion for a quadcopter, including:

$$\dot{\mathbf{p}} = \mathbf{v} \quad m\dot{\mathbf{v}} = m\mathbf{g} + \mathbf{R}\mathbf{f} \quad \dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\Omega}(\boldsymbol{\omega})$$

where:

- $\mathbf{p}$ denotes the position vector.

- $\mathbf{v}$ represents the velocity vector.
- m denotes the drone's mass.
- $\mathbf{g}$ denotes the gravitational acceleration vector.
- $\mathbf{R}$ is the rotation matrix representing the orientation.
- $\mathbf{f}$ denotes the total thrust force generated by the propellers.
- $\boldsymbol{\omega}$ represents the angular velocities of the drone's motors.
- $\boldsymbol{\Omega}(\boldsymbol{\omega})$ denotes the skew-symmetric matrix associated with $\boldsymbol{\omega}$.

The reward function in gym-pybullet-drone is designed for encouraging desirable behaviors and penalize undesired actions. It typically includes components such as:

- Distance to the target position.
- Smoothness of control inputs to promote stable flight.
- Proximity to obstacles to avoid collisions.
- Task-specific objectives, such as tracking a trajectory or reaching a goal.

5.3 Results

We conducted various simulations, to validate our proposed framework. Specifically, we executed simulation experiments involving the movement of the platform along different pathways, such as straight lines and curved paths. We varied the speed of the platform from 0.5 m/s to 3.5 m/s. We conducted experiments in simulation utilizing both the PID and LQR controllers as well as the Reinforcement Learning based approach. Within the simulation environment, we evaluated the efficacy of the framework we created under different circumstances.

5.3.1 PID Controller

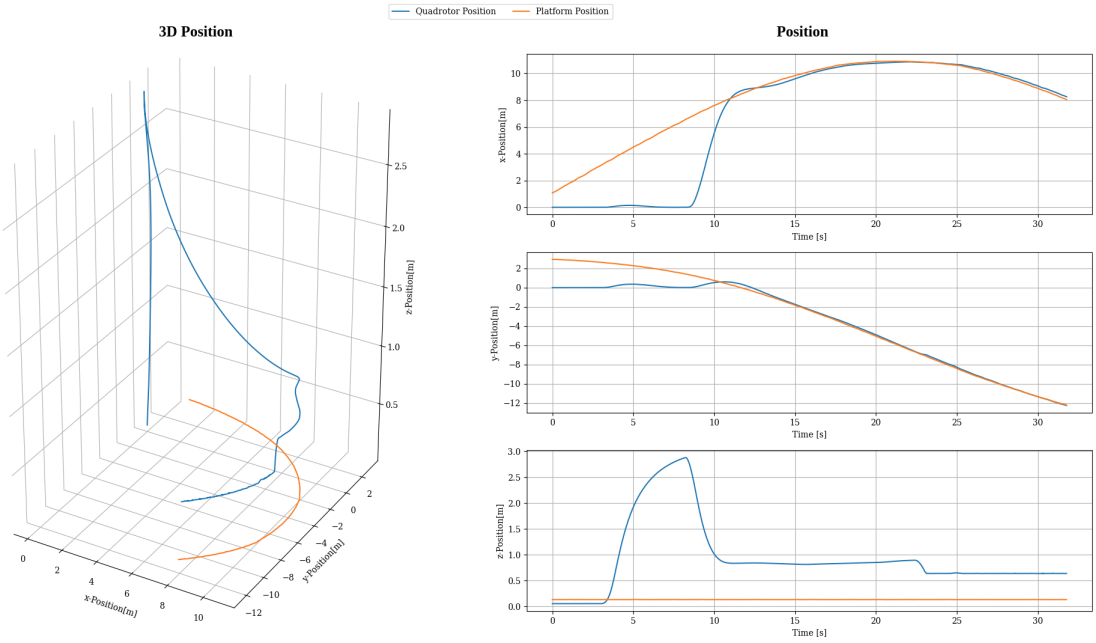

Figure 5.4: The results of one of the simulation experiments. The velocity of the ground vehicle is 1.55 m/s

Fig. 7.1 displays the outcomes of one of our simulation experiments. The quadrotor begins the takeoff phase at $t = 0$. The quadrotor tracks the moving platform until it reaches a point where the position error falls below a predefined threshold, signaling the appropriate time for landing. The discrepancies in the z-position values between the quadrotor and the moving platform post-landing are due to the landing pad's attachment at an elevation above the Husky.

5.3.2 LQR Controller

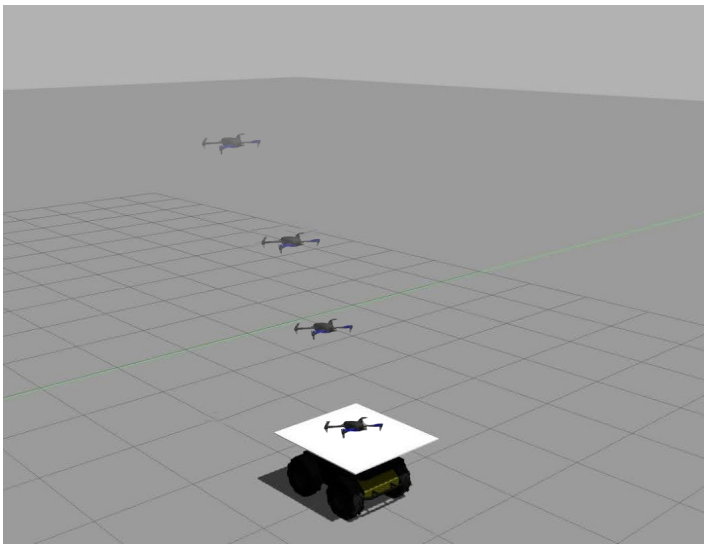

Figure 5.5: Quadrotor landing in the Gazebo Classic Simulation Environment

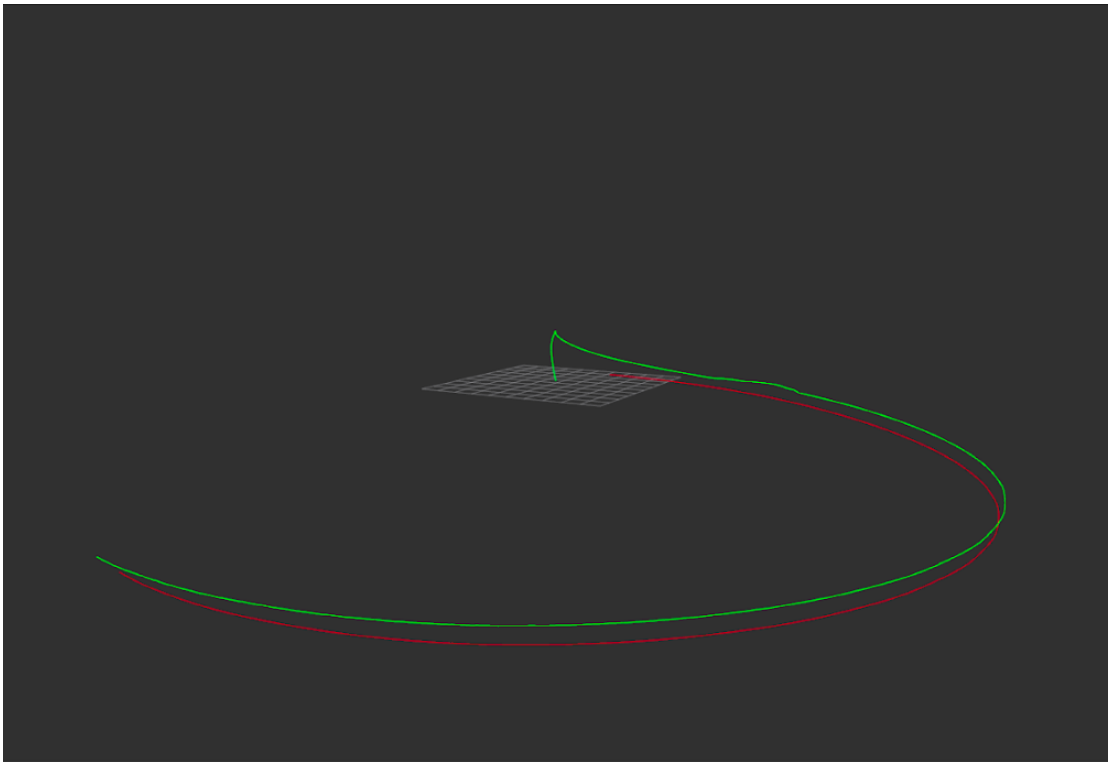

Figure 5.6: Visualisation of Platform's and Quadrotor's trajectories in RViz

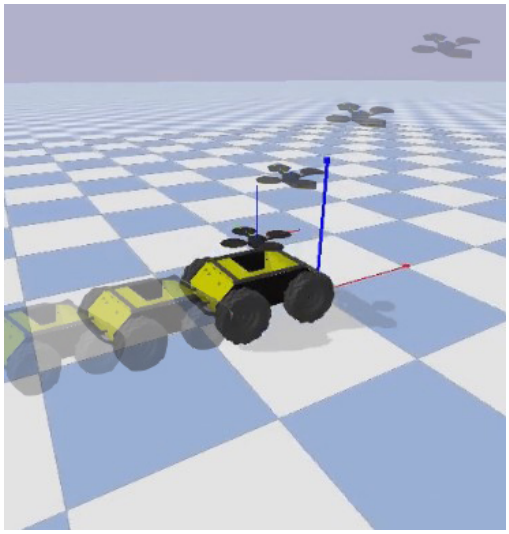

Figure 5.7: Quadrotor Landing PyBullet Simulation Environment

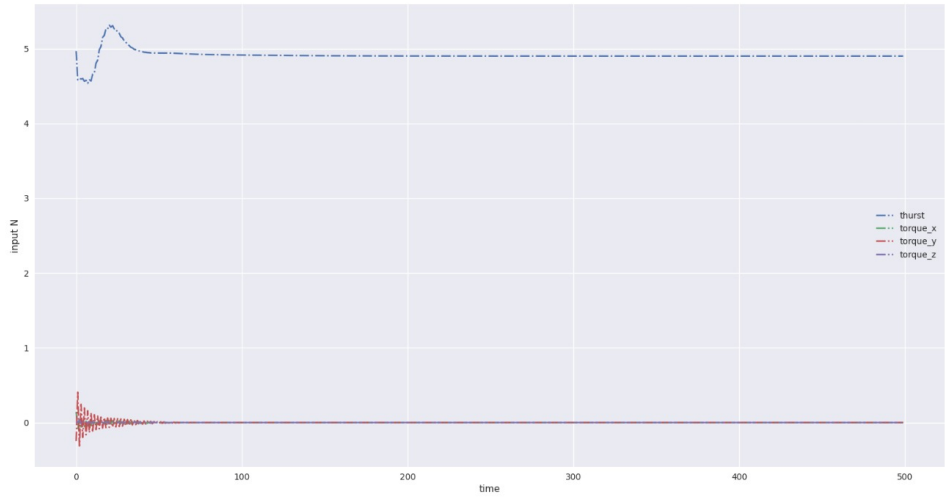

Figure 5.8: Thrust and Torques applied by the quadrotor

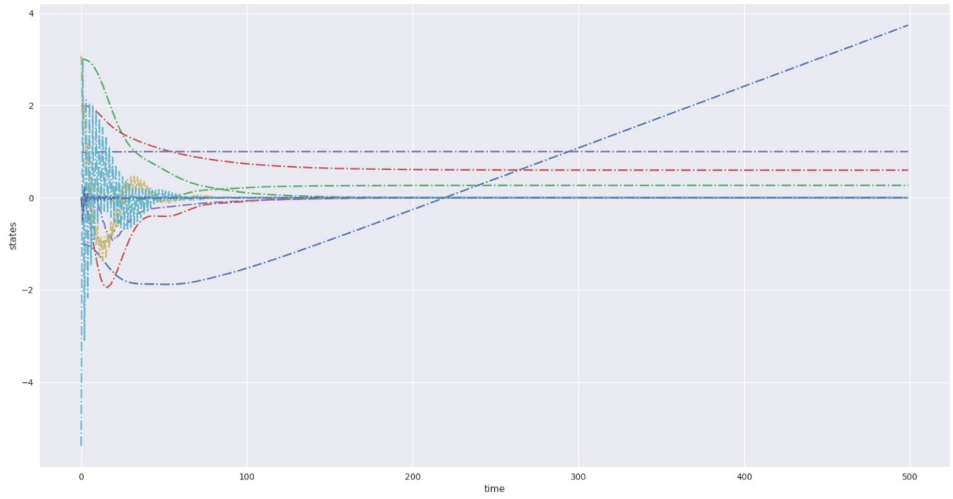

Figure 5.9: Quadrotor States

5.3.3 Reinforcement Learning

We trained a Proximal Policy Optimisation (PPO) algorithm using stable-baselines3 implementation of Deep Reinforcement Learning algorithms, which uses PyTorch and cuda to train the agent. The hyperparameters for the model are mentioned below

Hyperparameter	**Value**
Number of layers	4
Number of nodes in layer1	512
Number of nodes in layer2	512
Number of nodes in layer3	256
Number of nodes in layer4	128
Batch Size	64
gamma	0.99
learning rate	0.0003
clip rate	0.2
clip fraction	0.5

Table 5.2: Hyperparameters of the PPO Model

Loss and Reward Curves

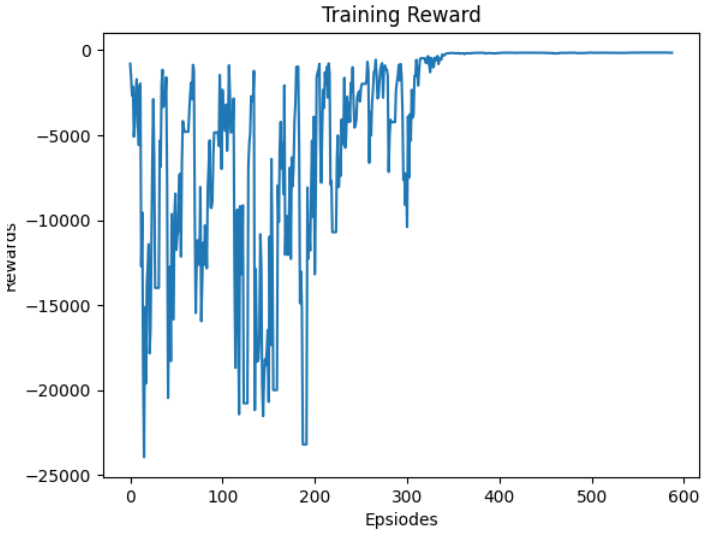

Figure 5.10: Rewards obtained during training

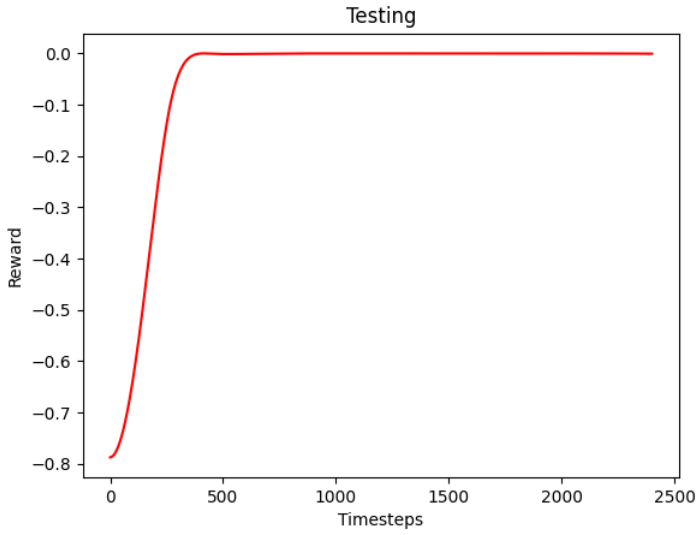

Figure 5.11: Rewards of the agent during testing

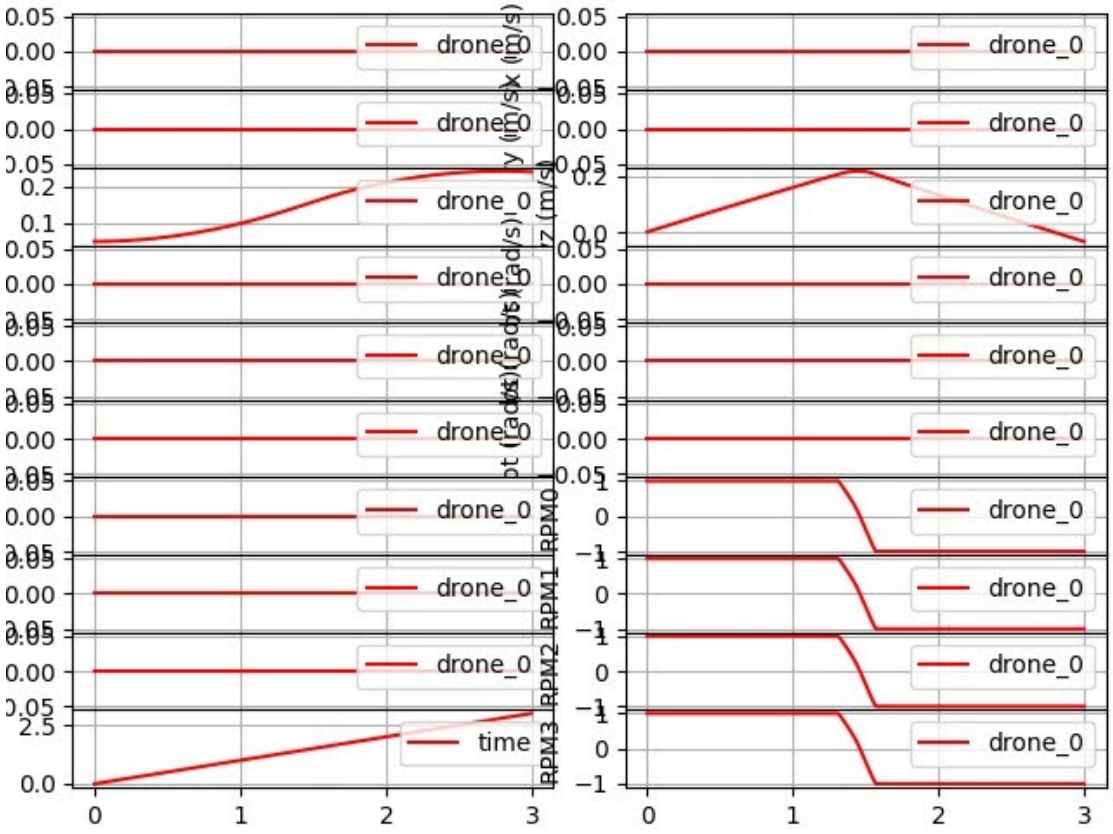

Figure 5.12: Drone's state while testing

Chapter 6

Hardware Deployment

6.1 Overview

This chapter describes all of the hardware components used in the quadrotor assembly and the construction of the moving platform. To ensure the system's dependability and functionality, every component is essential. Along with this we briefly explain the offboard mode of PX4 Autopilot and the hardware experiments we performed with the developed hardware platforms.

6.2 Components

6.2.1 Pixhawk 2.4.8 Flight Controller

The autopilot hardware that has PX4 firmware uploaded onto it is called a Flight Controller (FC).

Figure 6.1: Pixhawk Flight Controller

- **Processor**
  - 32-bit ARM Cortex M4 core with FPU
  - 168 Mhz/256 KB RAM/2 MB Flash
  - 32-bit failsafe co-processor
- **Sensors**
  - MPU6000 as main accel and gyro
  - ST Micro 16-bit gyroscope
  - ST Micro 14-bit accelerometer/compass (magnetometer)
  - MEAS barometer
- **Power**
  - Ideal diode controller with automatic failover
  - Servo rail high-power (7 V) and high-current ready
  - All peripheral outputs over-current protected, all inputs ESD protected
- **Interfaces**
  - 5x UART serial ports, 1 high-power capable, 2 with HW flow control
  - Spektrum DSM/DSM2/DSM-X Satellite input
  - Futaba S.BUS input (output not yet implemented)
  - PPM sum signal
  - RSSI (PWM or voltage) input
  - I2C, SPI, 2x CAN, USB
  - 3.3V and 6.6V ADC inputs
- **Dimensions**
  - Weight 38 g (1.3 oz)
  - Width 50 mm (2.0")
  - Height 15.5 mm (.6")
  - Length 81.5 mm (3.2")

Figure 6.2: Pixhawk 2.4.8 Specifications

The Pixhawk 2.4.8 flight controller serves as the brain of the quadrotor, controlling its flight maneuvers and stability. Its robust design and firmware support make it a popular choice among drone enthusiasts and researchers. The Pixhawk 2.4.8 integrates a wide array of sensors, including accelerometers, gyroscopes, magnetometers, barometers, and GPS modules. These sensors provide real-time data on the drone's orientation, velocity, altitude, and position, enabling precise control and navigation. The Pixhawk 2.4.8 is compatible with a wide range of drones, from small quadcopters to large fixed-wing aircraft. Its modular design allows users to customize and expand its functionality with additional peripherals and accessories, such as telemetry radios, cameras, and rangefinders. The Pixhawk 2.4.8 runs on open-source firmware, primarily PX4 and ArduPilot, which are continuously developed and improved by a global community of developers.

6.2.2 Raspberry Pi 4 Model B Compamion Computer

Separate on-board computers called "mission computers" or companion computers are linked to the flight controller to facilitate computationally demanding functions like collision avoidance and object avoidance.

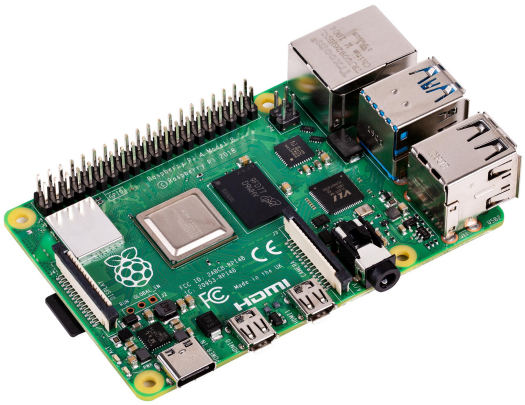

Figure 6.3: Raspberry Pi 4 Companion Computer

The flight controller includes essential flying and safety code and runs PX4. Linux is typically installed on the companion computer because it is a far superior platform for general program development. Fast serial or Ethernet links are used to connect them, and the MAVLink protocol is usually used for communication.

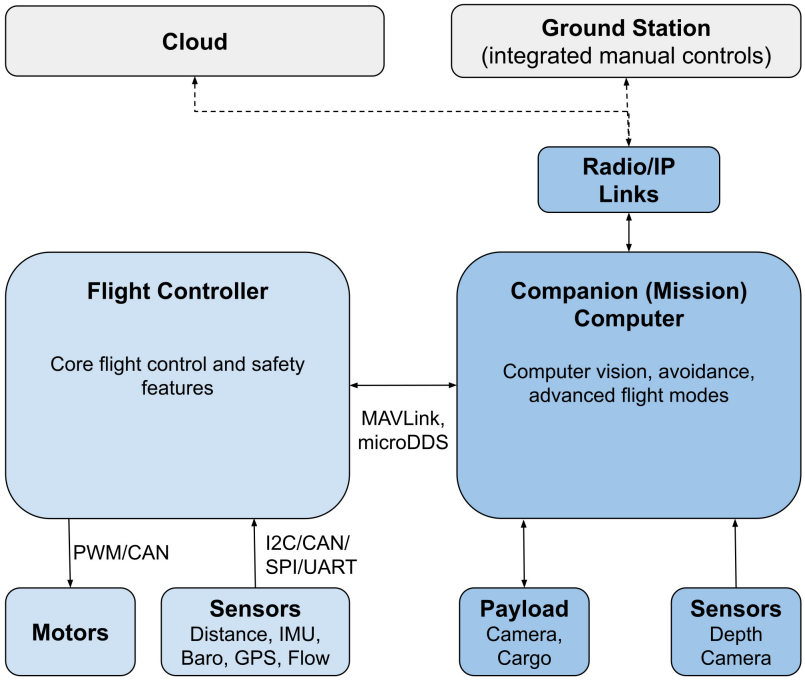

Figure 6.4: UAV architecture for flight controller and companion computer

Software for routing traffic to ground stations and the cloud, as well as for interacting with the flight controller, must be installed on the companion computer. On the companion computer, we set up ROS Noetic with the MAVROS package. For offboard control, PX4 expects companion computers to connect via TELEM2 port. By default, MAVLink is used as the interface for this port. A Raspberry Pi 4 serves as a

companion computer, running ROS 1 on the Linux Ubuntu OS. The connection between the Pixhawk flight controller and the Raspberry Pi was established through a serial interface, linking the Pixhawk TELEM2 port to the USB port of the Raspberry Pi, using an FTDI Chip USB-to-serial converter board.

6.2.3 Miscellaneous Components

In addition to the components mentioned above, our quadrotor system comprises several other essential elements:

LiPo Batteries

Lithium Polymer batteries provide the necessary power supply for the quadrotor's propulsion and onboard electronics. These high-energy-density batteries offer lightweight and rechargeable power sources, essential for extended flight durations and maneuverability.

Frame and Motors

The drone frame carries the propulsion system, consisting of brushless motors and propellers, essential for generating thrust and maneuvering. The frame's design influences the quadrotor's aerodynamics, stability, and payload capacity. High-quality materials, such as carbon fiber or aluminum alloys, ensure structural integrity and durability.

Electronic Speed Controllers (ESCs)

Based on control signals from the flight controller, ESCs adjust the drone's motor speed. ESCs play a crucial role in maintaining stable flight characteristics and responsiveness during maneuvers.

GPS (Global Positioning System) Module

GPS receivers deliver precise location data by capturing signals transmitted from Earth-orbiting satellites. Integrated into the quadrotor system, GPS enables precise navigation, waypoint following, and autonomous flight modes. By combining GPS data with other sensors, such as IMUs and barometers, the quadrotor can maintain stable flight even in challenging environments and perform complex missions with high accuracy.

6.3 Offboard Mode of PX4

For hardware deployment, we use the offboard mode of the PX4 Autopilot. In the offboard mode, the quadrotor follows position, velocity, or attitude setpoints provided via MAVLink, typically from a onboard computer. MAVROS facilitates the communication and transmission of setpoints to the vehicle.

Key Characteristics

- **Setpoint Provision**: Pose/attitude information is necessary for the mode to function.
- **Automatic Operation**: Offboard mode is automatic; RC control is disabled by default, except for mode changes. The quadcopter must be armed before engaging offboard.
- **Continuous Setpoint Reception**: The vehicle must receive a continuous stream of target setpoints ($> 2Hz$) to remain in the mode. If commands are not received at this frequency, the vehicle exits the mode.
- **Setpoint Stream Requirement**: Before activating the mode and to sustain it, the vehicle needs an uninterrupted flow of setpoint commands. To maintain its position in this mode, the vehicle relies on a continual stream of setpoints reflecting its current position.

Offboard Activation and Deactivation

- **Activation**: Prior to engaging the offboard mode, the drone must get a continuous stream of commands. Once armed and receiving setpoints at a sufficient rate, the mode can be activated.
- **Deactivation**: The vehicle will go out of offboard if the stream of target setpoints falls below 2Hz, ensuring safe operation and control continuity.

6.4 Hardware Platforms

Quadrotor

With the components listed in section 6.2, the following quadrotor has been assembled.

Figure 6.5: Quadrotor platform assembled in this thesis

Moving Platform

A moving platform of dimensions 25x25 inches was developed. It operates on a 5400 mAH LiPo battery and has two 10 rpm motors.

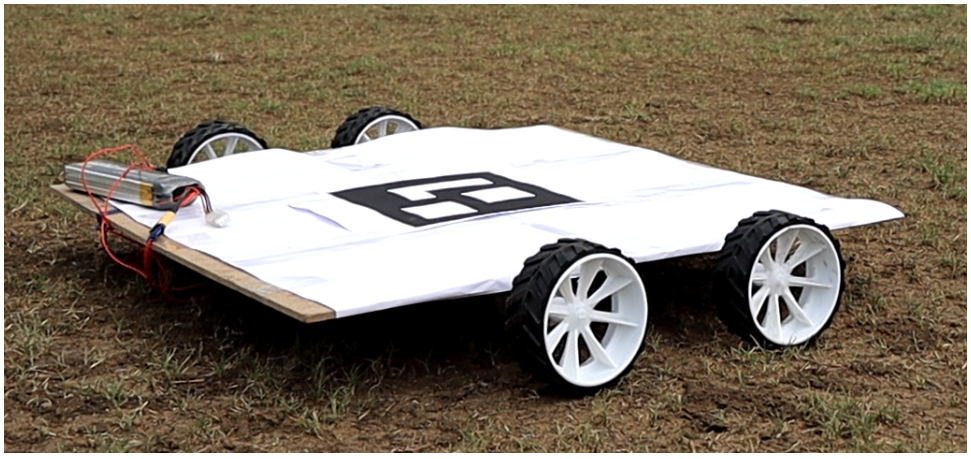

Figure 6.6: Moving Platform developed in this thesis

6.5 Hardware Experiments

In our hardware experiments, the initial position of the platform relative to the quadrotor was known. Utilizing the equations of motion for the moving platform, its position at any given moment is accurately predicted. Throughout the experiment, the moving platform maintained a steady velocity, ensuring consistent and predictable motion. Through the hardware experimentation results, we validate the effectiveness and capability of our methodology to achieve autonomous landings on moving platforms.

Figure 6.7: Drone Landing on the Moving Platform

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, various control strategies for achieving autonomous quadrotor landing on a moving platform are explored and implemented. Specifically, we implemented and compared three different control methods: PID, LQR, and Deep Reinforcement Learning (DRL). These methods were evaluated both in simulation environments and deployed on actual hardware to verify their effectiveness in real-world scenarios. The PID controller, a classic control approach, demonstrated stable performance in simulation but struggled to adapt to dynamic platform movements and disturbances in real-world experiments. The LQR controller, leveraging optimal control techniques, offered improved tracking accuracy and robustness compared to PID, particularly in scenarios with predictable platform motion. However, it still faced challenges in handling uncertainties and non-linearities inherent in practical landing tasks.

In contrast, the Deep Reinforcement Learning (DRL) approach exhibited remarkable adaptability and robustness, surpassing the performance of traditional control methods. By learning landing policies directly from experience, the DRL agent demonstrated superior tracking precision and resilience to disturbances, showcasing the potential of data-driven approaches for autonomous quadrotor landing.

In summary, this thesis provides important perspectives into the design and implementation of autonomous quadrotor landing systems, highlighting the strengths and limitations of different control strategies. By combining traditional control methods with cutting-edge deep reinforcement learning techniques and integrating vision-based sensing capabilities, more intelligent and adaptive aerial robotic systems can be developed in the future.

7.2 Future Work

This project can be further extended by incorporating the following things:

1. Looking ahead, future works should focus on further enhancing the autonomy and intelligence of quadrotor landing systems. One avenue for improvement is to initialize the DRL agent using Model Predictive Control (MPC) or LQR controllers, leveraging their established control policies as a starting point for reinforcement learning. The Learning by Cheating framework can facilitate this initialization process by providing expert demonstrations or pre-existing control policies to guide the DRL agent's initial exploration.

2. Additionally, integrating vision-based capabilities into the landing system, such as detecting ArUco markers using a camera attached to the drone, holds promise for improving landing accuracy and robustness in diverse environments. By incorporating visual feedback into the control loop, the quadrotor can localize itself relative to the landing platform more accurately, enabling precise and reliable landings even in challenging conditions.

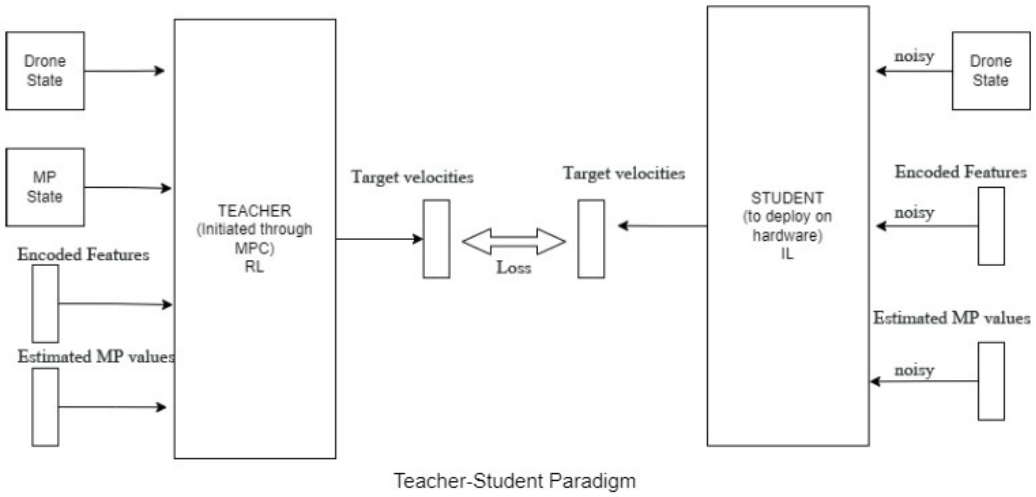

Figure 7.1: Plausible way of Initialising RL policy by using Learning-by-Cheating Framework

Chapter 8

List of conference acceptances

1. *"Reinforcement Learning Policy Initialization with Model-Based Control for Quadrotor Landing on Moving Platform"*, abstract accepted for oral presentation at the 2nd International Conference on Recent Advances in Applied Mathematics (RAAM 2024).

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Learning to fly by crashing: A deep reinforcement learning approach for autonomous quadrotor navigation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2642–2649. IEEE, 2016.

[2] Tomas Baca, Petr Stepan, and Martin Saska. Autonomous landing on a moving car with unmanned aerial vehicle. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2017.

[3] S. Bouabdallah, A. Noth, and R. Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2451–2456 vol.3, 2004.

[4] Truong Tuan Dang, Pedro Gomes Da Cruz, Vinh Hong Nguyen Le, Shahab G Abbasi, and David Michael W Powers. Autonomous navigation and landing of a quadrotor uav on a moving target using reinforcement learning. *IEEE Access*, 8:201080–201093, 2020.

[5] Davide Falanga, Alessio Zanchettin, Alessandro Simovic, Jeffrey Delmerico, and Davide Scaramuzza. Vision-based autonomous quadrotor landing on a moving platform. In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 200–207, 2017.

[6] Yi Feng, Cong Zhang, Stanley Baek, Samir Rawashdeh, and Alireza Mohammadi. Autonomous landing of a uav on a moving platform using model predictive control. *Drones*, 2(4), 2018.

[7] Philipp Foehn and Davide Scaramuzza. Onboard state dependent lqr for agile quadrotors. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6566–6572, 2018.

[8] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing, 2017.

[9] Saeed Ghaffari, Alireza Navaei, Hamidreza Khosravi, and Josep M Guerrero. Deep reinforcement learning for autonomous landing of multirotor uavs. *IEEE Transactions on Aerospace and Electronic Systems*, 54(3):1477–1490, 2018.

[10] Jawhar Ghommam and M. Saad. Autonomous landing of a quadrotor on a moving platform. *IEEE Transactions on Aerospace and Electronic Systems*, PP:1–1, 02 2017.

[11] Z. Jiang and G. Song. A deep reinforcement learning strategy for uav autonomous landing on a platform, 2022.

[12] Daewon Lee, Tyler Ryan, and H. Jin. Kim. Autonomous landing of a vtol uav on a moving platform using image-based visual servoing. In *2012 IEEE International Conference on Robotics and Automation*, pages 971–976, 2012.

[13] Xiangyu Li, Yifan Hu, Xuanzhe Luo, Chaoyue Lai, and Yang Liu. Deep reinforcement learning for autonomous quadrotor landing: A transfer learning approach. *IEEE Access*, 2021.

[14] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240, 2015.

[15] Aleix Paris, Brett Lopez, and Jonathan How. Dynamic landing of an autonomous quadrotor on a moving platform in turbulent wind conditions, 09 2019.

[16] Michalis Piponidis, Panayiotis Aristodemou, and Theocharis Theocharides. Towards a fully autonomous uav controller for moving platform detection and landing. In *2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID)*. IEEE, February 2022.

[17] Riccardo Polvara, Massimiliano Patacchiola, Sanjay Sharma, Jian Wan, Andrew Manning, Robert Sutton, and Angelo Cangelosi. Autonomous quadrotor landing using deep reinforcement learning, 2018.

[18] E. Reyes-Valeria, Rogerio Enriquez-Caldera, Sergio Camacho, and Jose Guichard. Lqr control for a quadrotor using unit quaternions: Modeling and simulation. pages 172–178, 03 2013.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.