Faculty of Power
and Aeronautical Engineering

Warsaw University of Technology

Institute of Aeronautics and Applied Mechanics

Bachelor's diploma thesis

in the field of study Aerospace Engineering
and specialisation Automation and Aerospace Systems

A Comparative Analysis of Reinforcement Learning and Robust Model Predictive Control

Leon Górecki
student record book number 327525

thesis supervisor
Professor Marcin Żugaj, DSc, Eng.
Kenan Majewski M.Sc. Eng.

WARSAW 2026

Abstract.

This thesis compares two different control approaches – Robust Model Predictive Control (RMPC) and Reinforcement Learning using Proximal Policy Optimization (PPO) – in the context of trajectory tracking when reliable positional information is not always available.

The motivation comes from real-world situations in which autonomous systems cannot rely on GNSS, such as indoor environments, underground spaces or areas affected by signal interference. In these conditions, position measurements become uncertain, which directly impacts control performance.

To study this problem, a unified simulation environment was created in which both controllers operated under the same conditions. GNSS degradation was modeled as noise affecting the robot's pose estimate, while LiDAR sensing remained available for obstacle detection. Both controllers were tasked with following a reference trajectory while avoiding static obstacles.

The RMPC controller handled uncertainty explicitly through constraint tightening and feedback correction. The PPO-based controller instead learned how to behave under uncertainty through interaction with the environment and a structured reward function encouraging tracking, progress and safety.

The results show that RMPC provides stable and predictable behavior, even when positional information degrades. PPO, on the other hand, is able to adapt to uncertainty through learned behavior, but without formal guarantees of stability or constraint satisfaction.

The comparison highlights a clear trade-off: RMPC offers safety and reliability, while PPO offers flexibility and adaptability. Neither method is universally better — their usefulness depends on the requirements of the system.

Keywords: Robust Model Predictive Control, Reinforcement Learning, Proximal Policy Optimization, GNSS-denied navigation, trajectory tracking, autonomous robotics, control under uncertainty, LiDAR-based perception

Streszczenie.

Niniejsza praca porównuje dwa podejścia do sterowania – Odporne Sterowanie Predykcyjne (RMPC) oraz Uczenie ze Wzmocnieniem w postaci algorytmu Proximal Policy Optimization (PPO) – w zadaniu śledzenia trajektorii w warunkach niepewnej informacji o położeniu.

Motywacją do podjęcia tematu są rzeczywiste sytuacje, w których autonomiczne systemy nie mogą polegać na sygnale GNSS, takie jak środowiska zamknięte, przestrzenie podziemne czy obszary z zakłóceniami sygnału. W takich warunkach pomiary położenia stają się obarczone niepewnością, co bezpośrednio wpływa na działanie układu sterowania.

W celu przeprowadzenia analizy stworzono jednolite środowisko symulacyjne, w którym oba regulatory działały w identycznych warunkach. Degradację GNSS zamodelowano jako szum wpływający na estymację położenia robota, przy jednoczesnym zachowaniu dostępu do danych z systemu LiDAR wykorzystywanego do detekcji przeszkód. Zadaniem obu regulatorów było śledzenie trajektorii referencyjnej oraz unikanie statycznych przeszkód.

Regulator RMPC uwzględniał niepewność w sposób jawny poprzez zacieśnianie ograniczeń i korekcję sprzężeniem zwrotnym. Sterownik PPO uczył się zachowania w warunkach niepewności poprzez interakcję ze środowiskiem oraz funkcję nagrody promującą dokładność śledzenia, postęp i bezpieczeństwo ruchu.

Uzyskane wyniki wskazują, że RMPC zapewnia stabilne i przewidywalne działanie nawet przy pogorszonej informacji o położeniu. PPO natomiast wykazuje zdolność adaptacji do niepewności, jednak bez formalnych gwarancji stabilności i spełnienia ograniczeń.

Porównanie uwidacznia wyraźny kompromis: RMPC oferuje bezpieczeństwo i niezawodność, natomiast PPO elastyczność i zdolność adaptacji. Żadne z podejść nie jest jednoznacznie lepsze — ich przydatność zależy od wymagań stawianych systemowi.

Słowa kluczowe: Odporne Sterowanie Predykcyjne, Uczenie ze Wzmocnieniem, Proximal Policy Optimization, nawigacja bez GNSS, śledzenie trajektorii, robotyka autonomiczna, sterowanie w warunkach niepewności, percepcja LiDAR

Contents

1. Introduction

1.1. Background and Motivation

In modern times, control systems face ever increasing scrutiny while operating in uncertain and nonlinear environments [1], [2], [3]. This scrutiny arises from growing performance, safety, and reliability requirements, particularly in safety-critical applications such as autonomous vehicles, robotics, and aerospace systems. As a result, control strategies are increasingly evaluated not only on nominal performance, but also on their ability to tolerate disturbances, modeling errors, and unanticipated operating conditions.

Historically, traditional model-based approaches, such as Model Predictive Control (MPC), have had a significant share of success when applied to systems with accurate models and predictable dynamics [2], [4], [5]. Their success is largely attributed to the explicit use of system models for prediction, optimization, and constraint handling, which enables transparent and interpretable control design. However, as system complexity and uncertainty increase, these same model-based control frameworks often struggle to remain robust and maintain high performance, particularly when modeling errors or disturbances violate underlying assumptions [2], [3], [6].

In comparison, data-reliant techniques, among which lies Reinforcement Learning (RL), discover and learn control policies through interaction with a training environment, without requiring an explicitly given system model [7], [8], [9]. This paradigm enables RL-based controllers to handle complex, nonlinear dynamics and adapt to environments that are difficult to model analytically. However, this flexibility is achieved at the cost of reduced interpretability and the absence of formal robustness and stability guarantees, which remain central concerns in control engineering.

This convergence of classical control theory and modern machine learning methods forms the primary motivation for this comparative analysis. By systematically evaluating the strengths and limitations of both approaches, this work aims to clarify their applicability to autonomous systems operating under uncertainty and imperfect modeling conditions.

1.2. Problem Statement

A systematic understanding of the relative strengths and limitations of these two vastly different control approaches in navigation under uncertainty is still incomplete. In particular, it is not well established how model-free reinforcement learning compares to robust model predictive control in stability, tracking performance, and computational efficiency for uncertain or partially known dynamical systems. More specifically, in scenarios in which absolute positional data, usually provided by the Global Navigation

Satellite System (GNSS), is unavailable. In the remaining part of this thesis, this scenario will be referred to as a *GNSS-denied scenario*, good examples of which are: underground operation (basements, parking garages), structure induced GNSS occlusion and intentional adversarial actions (spoofing, jamming). A clear resolution as to the relative performance of model-driven and data-driven methods in these GNSS-denied scenarios would serve as a basis for future development of navigation algorithms in data-scarce environments. This thesis aims to address this need through a controlled, side-by-side evaluation. What requires mention is that this work does not attempt to develop a platform which will operate without a GNSS signal, but to verify the disturbance rejection capabilities of two differing approaches, which both function nominally when provided with access to GNSS. The work tests whether model-based robustness (Tube RMPC) or model-free adaptability (PPO) better preserves trajectory tracking performance under bounded positional uncertainty.

1.3. Research Objective

The goal is to conduct a structured comparison between RL and RMPC in GNSS-denied scenarios and demonstrate the advantages and disadvantages of each method. Specifically, for a digitally simulated ROOMBA model [10] this thesis will:

1. Implement representative algorithms for both RL and RMPC, with a concrete choice being discussed in Chapter 2 and establish a performance benchmark;
2. Evaluate stability, accuracy and robustness for both approaches;
3. Analyze computational complexity and convergence characteristics and identify situations and applications where each approach is preferable.

1.4. Significance of Contributions

This work contributes an empirical and conceptual comparison between modern data-based and classical model-based control solutions within a unified evaluation framework. The results clarify trade-offs among adaptability, robustness, and computational demand, guiding practitioners selecting control strategies for systems in which absolute positional data might not be available or reliable. Examples of sectors which might benefit from such an analysis are:

- the Unmanned Aerial Vehicle (UAV) industry, especially indoor quadcopter operation [11];
- the mining industry, due to the lack of GNSS signal below the earths surface;
- the construction industry when it comes to autonomously conducted build site inspections;
- various search and rescue (S&R) applications.

1.5. Dissertation Structure

The remainder of this dissertation is organized as follows:

1. A comprehensive **Literature review** in Chapter 2 covering the existing applications of RMPC and RL in GNSS denied navigation, as well as the theoretical premises of the chosen models;
2. A detailed **Platform and environment modeling** section in Chapter 3 carefully depicting the practically implemented agent-ready platform as well as the goals and obstacles of the environment in which the platform will exist;
3. An exhaustive **Control framework theory and implementation** section in Chapter 4 discussing the practical aspects of creating RMPC and RL agents in a programming language;
4. The critical **Benchmarking and results** section in Chapter 5 documenting the two models' results when applied to environments with and without zones of temporary GNSS-denial;
5. A comparative **Conclusion and discussion** section

2. Literature review

2.1. Autonomous Mobile Robots in GNSS-Denied Environments

Autonomous mobile robots, including wheeled ground vehicles as well as low-altitude UAVs, are increasingly required to operate in environments where GNSS signals are unavailable, unreliable, or deliberately denied [11], [12]. Typical examples include indoor facilities, urban canyons, forests, tunnels, underground infrastructure, and adversarial environments of deliberate interference. In operating conditions such as these, the absence of absolute positioning information significantly complicates navigation, localization, and control tasks.

Figure 2.1. LiDAR and DRL based surveying drone by Ouster | Source: Ouster Inc. (2023) [13]

GNSS-denied operation forces autonomous robotic systems to rely on onboard sensing and state estimation techniques, such as inertial measurement units, wheel odometry, visual odometry, and LiDAR-based localization. These sensors are inherently affected by noise, bias, drift, and partial observability, leading to increasing state uncertainty over time [11], [12], [14]. Consequently, the control system must be capable of maintaining stability and performance in the presence of significant model uncertainty and external disturbances.

From a control perspective, navigation in GNSS-denied environments constitutes a constrained optimal control problem under uncertainty. This has motivated extensive research into robust control methods, including Model Predictive Control (MPC) and its robust variants, as well as data-based approaches such as Reinforcement Learning (RL), which offer an alternative paradigm for handling complex and uncertain dynamics [2], [3], [4], [7].

2.2. Model-Based Control Approaches

2.2.1. Optimal Control Foundations

The core principle of model-based control is the existence of explicitly available or estimable mathematical models of real world system dynamics, which are then used to compute control inputs aiming to optimize a predefined performance criterion. Classical optimal control theory rigorously formulates the solutions to these kinds of problems as a minimization of a cost functional being subject to difference or differential equations which in turn describe the system dynamics [15].

Due to its rigorous mathematical formulation, optimal control can offer strong theoretical guarantees. However, in practical applications in robotics we often encounter its main limitations: modeling inaccuracies, unmodeled dynamics as well as hard constraints on states and inputs. These significant limitations have motivated the development of receding-horizon control strategies, most notably Model Predictive Control [2], [3], [4], [15].

2.2.2. Model Predictive Control

Model Predictive Control is a control strategy in which an optimization problem is solved online over a finite prediction horizon at each control step.The resulting optimal control sequence is applied in a receding-horizon manner, with only the first control input executed before the optimization is repeated [2].

MPC has several properties that make it particularly attractive for robotic applications:

- explicit handling of state and input constraints,
- suitability for multi-variable systems,
- intuitive formulation of trajectory tracking and path-following objectives.

What follows, is that MPC has been widely adopted in mobile robotics applications, autonomous driving and UAV control [2], [5].

However, a limiting factor of standard MPC formulations is their assumption of accurate system modeling and the fact that they neglect the effects of disturbances. Both of these negatively impact the robustness of this control approach in uncertain environments [3], [16].

2.2.3. Robust Model Predictive Control

The natural extension of standard MPC would explicitly account for uncertainty and disturbances and correct for them. This school of control approaches has been labeled Robust Model Predictive Control and the range of uncertainties it can factor in spans from sensor noise, through unmodeled dynamics, to imperfect state estimation [3].

Several RMPC formulations have been proposed in the literature, including worst-case (min–max) MPC, tube-based MPC, and constraint-tightening approaches [16], [17]. In

particular, tube-based MPC constructs a nominal trajectory surrounded by an invariant tube that guarantees constraint satisfaction for all admissible disturbances.

MPC's robust extension retains all its formal guarantees of stability and constraint satisfaction, making it well suited for safety-critical applications. However, these guarantees come at the cost of increased computational complexity and broadly understood conservatism. An algorithm based on worst-case assumptions will often act overly cautiously - an effect which significantly grows in strength in environments in which the modeled uncertainties themselves are uncertain, such as poorly modeled or highly dynamic environments [2], [3], [16], [17].

Among the available and well documented RMPC formulations, tube-based MPC offers a favorable trade-off between robustness guarantees and computational tractability. Unlike min–max MPC, which directly optimizes against worst-case disturbances and often leads to high computational complexity, tube-based MPC separates nominal performance optimization from disturbance rejection while fully retaining formal guarantees of stability and constraint satisfaction [2], [17]. This balance makes tube-based MPC a suitable representative RMPC approach for safety-critical robotic applications and a practical baseline for comparison with learning-based methods.

2.3. Reinforcement Learning in Control

2.3.1. Fundamentals of Reinforcement Learning

Reinforcement Learning (RL) is a fairly modern data-driven control approach in which an agent learns an optimal control policy through interaction with an environment.

A common formulation of the control problem is that of a Markov Decision Process (MDP), which is defined by a discrete state space, a discrete action space, a transition model, and a reward function [7].

In comparison to the previously discussed model-based control methods, RL does not require an explicit analytical model of system dynamics be provided to it a priori. Instead the control policy (a function mapping states onto predicted best actions) can be learned by maximizing an expected cumulative reward in the process of trial-and-error interaction. These properties make RL especially appealing when applied to systems with complex, nonlinear or partially unknown dynamics [7], [8], [15].

2.3.2. Deep Reinforcement Learning and Continuous Control

The distinction between RL and Deep RL (DRL) arises from the recent increase in computational power available to the public, mostly in the form of GPU processing. DRL combines the trial-and-error approach of standard RL with function approximation based on Neural Networks (NNs). While in regular RL the policy function is discrete, the use of NNs opens the door to continuous policy functions applied to high-dimensional continuous control problems. Algorithms such as Deep Deterministic Policy Gradient,

Proximal Policy Optimization [9], and Soft Actor-Critic[18] have all demonstrated strong performance in continuous action spaces.

These methods have been successfully applied to a wide range of robotic tasks, including locomotion, manipulation, and navigation. Their ability to learn complex behaviors directly from data makes them attractive for autonomous systems operating in uncertain and difficult to model environments, including GNSS-denied scenarios.

Among contemporary deep reinforcement learning algorithms for continuous control, Proximal Policy Optimization (PPO) represents a widely adopted and well-established baseline. PPO is consistently documented to achieve a favorable balance between sample efficiency, implementation simplicity, and training stability through its clipped policy update mechanism, avoiding the need for complex second-order optimization required by earlier policy-gradient methods [9]. Its robustness to hyperparameter choices and consistent empirical performance across a wide range of robotic control tasks make PPO a suitable representative of model-free reinforcement learning for the purposes of comparative evaluation.

2.3.3. Limitations of Reinforcement Learning

Despite its flexibility, Reinforcement Learning suffers from several important limitations when applied to safety-critical robotic systems. RL algorithms typically lack formal guarantees of stability and constraint satisfaction, and their behavior is highly sensitive to reward function design and training conditions, meaning the results of the model rely greatly on the engineer performing the implementation [7], [9].

Furthermore, RL methods often require large amounts of training data and extensive exploration, which may be infeasible or unsafe in real-world applications and is therefore often conducted in simulation. These issues pose significant challenges for the deployment of RL-based controllers in autonomous navigation tasks involving obstacles and operational constraints [7], [19], [20].

2.4. Comparative and Hybrid Approaches

2.4.1. RL versus MPC: Comparative Studies

Several studies have investigated the relative strengths and weaknesses of RL and MPC for control and navigation tasks. In general, MPC-based approaches excel in predictability, constraint handling, and robustness, while RL-based approaches demonstrate superior adaptability and performance in poorly modeled environments [21].

However, direct comparisons are often hindered by differences in experimental setups, assumptions, and performance metrics. As a result, the practical trade-offs between RL and robust MPC remain insufficiently quantified, particularly in the context of GNSS-denied navigation.

2.4.2. Hybrid MPC–RL Methods

To leverage the complementary strengths of both paradigms, hybrid approaches combining MPC and RL have been proposed. Examples include using RL to tune MPC cost functions or constraints, learning system models for MPC using RL, or employing MPC as a safety filter for RL policies [22].

While these approaches show promise, they introduce additional system complexity and often blur the distinction between learning-based and model-based control. Moreover, hybrid methods do not eliminate the need for a clear understanding of the standalone capabilities and limitations of RL and RMPC.

3. Platform and Environment modeling

In this chapter, the aim is to formalize the simulation environment and the mobile robot platform on which both the RL and RMPC agents will operate. The key objective in this chapter is to provide a clear and precise description of the platform configuration, state and action representations, environment dynamics and controller interfaces used throughout the remaining chapters. The practical implementation of the environment and robot models was based on the *IR-SIM* platform [23] by Ruihua Han.

3.1. Simulation environment

The simulated world (Fig. 3.1)is defined as a square planar environment of fixed size 20 × 20 meters. The simulator operates on a fixed time-step basis and advances the robot state through discrete integration of its kinematic model. Environment geometry, robot configuration, obstacle and reference trajectory layout are defined declaratively and stored as configuration files ensuring reproducibility as well as a large enough sample size for statistical analyses.

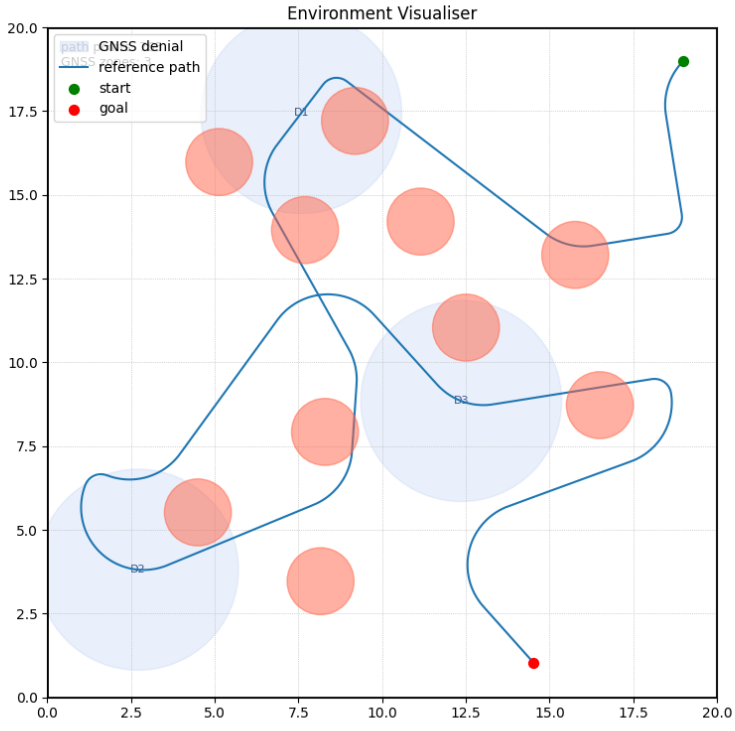

Figure 3.1. Environment example - world TR_PP_00003(3.2.1)

3.1.1. Reference trajectory definition

The reference trajectory generation relies on two stochastically alternated movement modes being numerically integrated to create a path, which is then sampled along its distance to create the intermediate goal points for both of the controllers to navigate to.

Mode 1 - straight line movement The simpler of the two modes, defines movement in a straight line inside the planar space. In this mode, the heading angle stays constant, and the change in position is dictated by the constant θ and by a predefined path length L, which can be seen in the following equations.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + L \cdot cos\theta_k \\ y_k + L \cdot sin\theta_k \\ \theta_k \end{bmatrix} \tag{1}$$

Mode 2 – Constant Radius Turn In this mode, the path follows a predefined turn radius R over a fixed arc length L. Given the current state

$$\mathbf{x}_k = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^\top$$

and the turn direction discriminant $\sigma \in \{-1, +1\}$, the center of the circular arc and the initial angular parameter are given by

$$\begin{bmatrix} c_x \\ c_y \\ \phi_0 \end{bmatrix} = \begin{bmatrix} x_k - \sigma R \sin\theta_k \\ y_k + \sigma R \cos\theta_k \\ \theta_k - \sigma \dfrac{\pi}{2} \end{bmatrix}. \tag{2}$$

For any arc-length parameter $s \in [0, L]$, the position and orientation along the trajectory are described by

$$\begin{bmatrix} x(s) \\ y(s) \\ \theta(s) \end{bmatrix} = \begin{bmatrix} c_x + R\cos\left(\phi_0 + \sigma \dfrac{s}{R}\right) \\ c_y + R\sin\left(\phi_0 + \sigma \dfrac{s}{R}\right) \\ \theta_k + \sigma \dfrac{s}{R} \end{bmatrix}. \tag{3}$$

Evaluating the above expressions at $s = L$ yields the end-of-segment state update

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} c_x + R\cos\left(\phi_0 + \sigma \dfrac{L}{R}\right) \\ c_y + R\sin\left(\phi_0 + \sigma \dfrac{L}{R}\right) \\ \theta_k + \sigma \dfrac{L}{R} \end{bmatrix}. \tag{4}$$

When represented in code, the environment generator alternates between these two modes stochastically and points are sampled at a high resolution along the final path, and subsequently saved as a *json* file to serve as configuration files.

3.1.2. Obstacle placement

While the aim of this thesis is mostly to track controller performance measured by the quality of its tracking of the reference trajectory, the author also acknowledges that an unintended collision with an object during operation should disqualify a controller from being classified as functional. Therefore, a set of randomly generated obstacles shall be placed in each world. For the simplicity of calculations they shall all have a radius of 1 meter and the location of their centers shall be sampled uniformly from the domain, with the exception that those too close to the world boundary or to the start or goal points shall be rejected and re-randomized. Obstacles shall also be rejected if they come within a radius of 1 m from a previously placed obstacle. These conditions have been put in place to simulate the correctness of the chosen reference path, while keeping the door open for controller failure due to collision. All obstacles are static and their positional information is conveyed to the robot via a simulated LIDAR signal, which is an inbuilt feature of the IR-SIM simulator. No dynamic obstacles or map uncertainty are introduced and the details of the LIDAR obstacle-detection system are described in section 3.2.2.

3.1.3. GNSS denial modeling

GNSS systems naturally exhibit uncertainty of up to 10 meters, however it can be greatly lowered using differential correction systems like Real Time Kinematic (RTK). The implementation of such systems can bring the uncertainty in position down to 1-2 cm. Since the utilized simulator base does not naturally support noise, a preprocessing step will be added between the simulator and controller, which will alter the controllers observation vector. This base level of noise will from this point on be addressed as *Mode 1 Noise*, it will be applied at all times during operation and can be expressed mathematically as the addition of isotropic zero-mean Gaussian noise.

$$\mathbf{z^{m_1}} = \mathbf{x} + \nu^{\mathbf{m_1}} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \nu_{x1} \\ \nu_{y1} \\ \nu_{\theta 1} \end{bmatrix}, \ \nu_{x1}, \nu_{y1} \sim \mathcal{N}(0, \sigma_1^2), \nu_{\theta 1} \sim \mathcal{N}(0, \sigma_{\theta 1}^2) \tag{5}$$

where $\sigma_1 = 0.02\,m$ represents standard GNSS accuracy when RTK is implemented, and where the standard deviation σ_θ can be calculated from the equation

$$\sigma_\theta = \frac{\sqrt{2}}{1.5 \cdot R} \cdot \sigma \tag{6}$$

which arises from the assumption that the heading angle is derived from a dual-antenna GNSS configuration, where vehicle orientation is inferred from the direction of a baseline

vector connecting two independent RTK-level position measurements. Each antenna is assumed to be affected by isotropic zero-mean Gaussian noise with variance σ^2, and the heading is computed via the nonlinear mapping $\theta = \text{atan2}(\Delta y, \Delta x)$. Using first-order linearization of the atan2(·) function, the subtraction of two independent position measurements results in a doubling of positional variance, while the sensitivity of the heading estimate scales inversely with the baseline length. With the virtual GNSS nodes placed at $\pm\frac{3}{4}R$ along the longitudinal axis, the effective baseline becomes $L = 1.5R$, yielding the expression for σ_θ given above.

During world creation, points on the reference path are chosen at random and designated as the center of simulated regions of GNSS-denial. These regions are modeled as circular zones with a randomly chosen radius, in which absolute positional information is additionally degraded. Upon entering these regions, the controlled robot loses access to an accurate Cartesian position, which will from this point on be titled *Mode 2 Noise* and can be expressed as

$$\mathbf{z^{m_2}} = \mathbf{x} + \nu^{\mathbf{m_2}} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \nu_{x2} \\ \nu_{y2} \\ \nu_{\theta 2} \end{bmatrix}, \ \nu_{x2}, \nu_{y2} \sim \mathcal{N}(0, \sigma_2^2), \nu_{\theta 2} \sim \mathcal{N}(0, \sigma_{\theta 2}^2) \tag{7}$$

where $\sigma_2 = 0.4\,m$ represents a degraded GNSS signal caused by low-to-moderate structural occlusion, and where σ_θ can be calculated from equation 6.

Contrary to the controller, the simulator itself retains access to the ground-truth state, which is used for collision detection, termination conditions, and performance evaluation.

This modeling choice allows the influence of GNSS-denial on control performance to be isolated, without altering the underlying system dynamics or evaluation criteria.

3.1.4. Controller–Simulator interface

At each simulation step, the controller interacts with the environment through a standardized interface. The simulator advances the robot state according to the previously applied control input and provides the current robot state together with information about nearby obstacles.

Depending on whether the robot lies within an GNSS-denial region, the controller receives the observation vector consisting of the true state modified by the appropriate noise mode. Based on this perceived state and obstacle information, the controller computes a control input, which is then applied to the simulator for the subsequent time step.

This interaction loop is identical for both evaluated controllers, ensuring that differences in observed performance arise solely from controller design rather than from discrepancies in simulator interfacing.

3.2. Robot model

The simulated agent is represented as a circular planar differential-drive mobile robot with the state being represented as

$$\mathbf{x} = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix},$$

where x, y denote the Cartesian position in the world frame and θ the heading angle (measured counter-clockwise from the *x-axis*).

The robot takes control inputs in the form of continuous linear and angular velocity commands

$$\mathbf{u} = \begin{bmatrix} v \\ \omega \end{bmatrix},$$

which are applied at discrete time steps determined by the simulator sampling rate. These control inputs affect the state by means of the following state update equations

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cdot cos\theta \\ v \cdot sin\theta \\ \omega \end{bmatrix} \tag{8}$$

A graphical depiction of the robot with its state and input parameters is presented below in Fig.3.2.

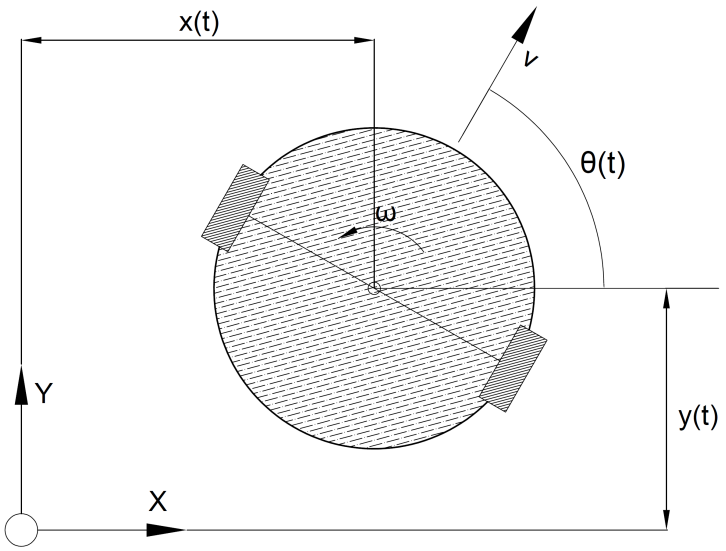

Figure 3.2. Graphical representation of a robot in a state applying control inputs

3.2.1. Dataset structure and split

With the intent of the work culminating in a large scale and statistically meaningful evaluation, the environments are generated in bulk offline and organized into disjoint datasets. These separate sets are intended to be used for controller training (applies only to PPO), validation and the final direct comparison. The environments are grouped by the aforementioned use cases and by controller type, and each of them is stored in its own directory and contains the following elements:

- an IR-SIM configuration file describing world geometry, robot placement and characteristics and obstacle placement;
- a reference trajectory file;
- an GNSS-denial file containing the positions and radii of the GNSS-denial zones

3.2.2. LiDAR system

Seeing as a strong enough GNSS disturbance leaves the robot without reliable absolute localization, it is crucial to allow it access to data from a secondary sensor system. Even though the thesis focuses on the rejection of GNSS disturbances by two algorithms, the author acknowledges that in a real world scenario, sending a robot into a GNSS-denied region equipped only in a GNSS receiver is not advisable. Therefore, the robot in simulation will be equipped with a simulated front-facing LiDAR system to ensure its awareness of obstacles despite its unawareness of its own positioning. This LiDAR is an available feature of the simulation system IR-SIM and consists of LiDAR beams spanning 180°.

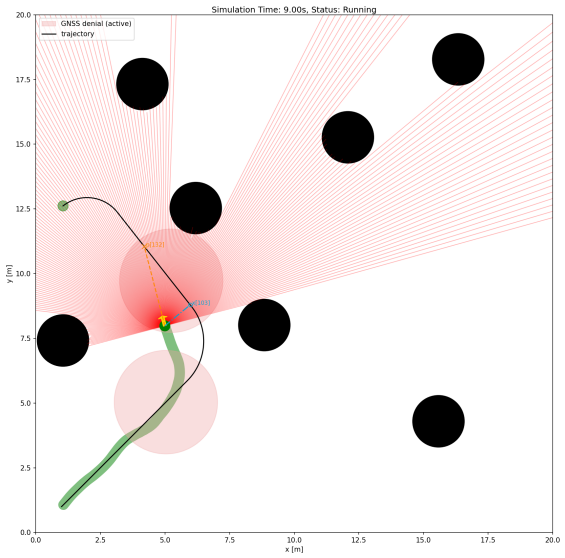

Figure 3.3. LiDAR as implemented in the IR-SIM simulation

3.2.3. Real world robot model - discussion

Due to time constraints, this thesis will not be discussing bringing the robot out of simulation and into the real world, but if that were the case, a pre-processing step would be added to ensure realism. Namely, the inputs would become the velocities of both (right and left) motors

$$\mathbf{u} = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix}$$

with the conversion between the two being described by the following equations:[24]

$$\begin{bmatrix} v_R \\ v_L \end{bmatrix} = \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} \cdot r + \begin{bmatrix} \varepsilon_{1R} \\ \varepsilon_{1L} \end{bmatrix} \tag{9}$$

$$\omega = \frac{v_R - v_L}{2 \cdot R} \tag{10}$$

$$l = R \cdot \frac{v_R + v_L}{v_R - v_L} \tag{11}$$

$$v = \omega \cdot l = (\frac{v_R - v_L}{2 \cdot R}) \cdot (R \cdot \frac{v_R + v_L}{v_R - v_L}) = \frac{v_R + v_L}{2} \tag{12}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(v_R + v_L) \\ \frac{1}{2 \cdot R}(v_R - v_L) \end{bmatrix} \tag{13}$$

where ω_R and ω_L are the motor speeds for the right and left wheel, r is the wheel radius, v_R and v_L are the groundspeeds of each wheel and ε are noise parameters originating in slippage or motor unevenness. The remaining geometric parameters have been depicted below in Figure 3.4

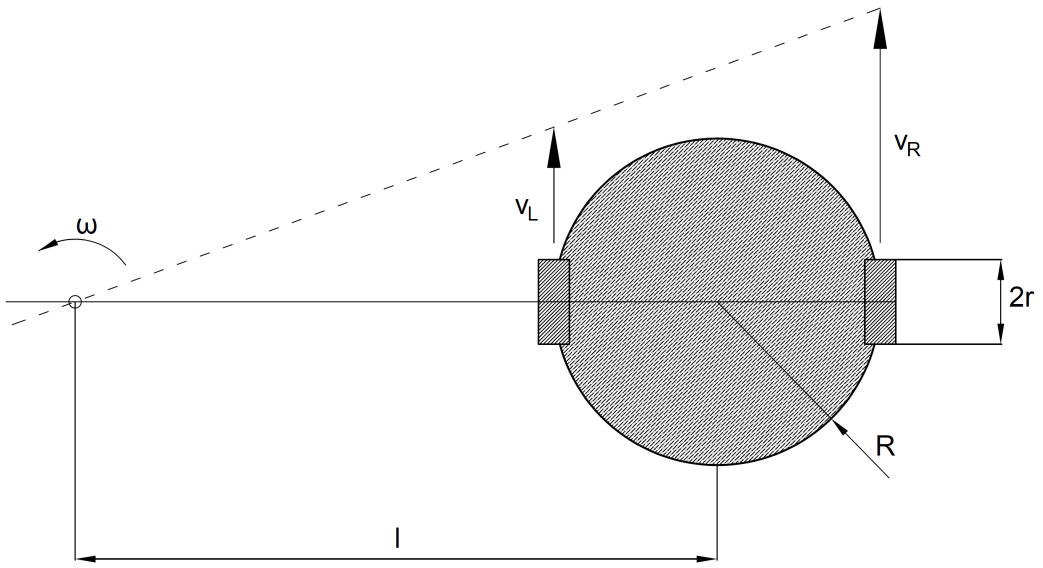

Figure 3.4. Robot model diagram

4. Control framework - Theory and implementation

4.1. Robust Model Predictive Control - Theory

4.1.1. Purpose and scope

Model Predictive Control (MPC) is selected in this thesis as the representative model-based control approach against which reinforcement learning methods are compared. MPC offers a structured framework for trajectory tracking under constraints and has been widely applied in mobile robotics. However, classical MPC relies on the assumptions of perfect state information and accurate system models, assumptions that are violated in GNSS denied scenarios.

To address this limitation, a robust variant of MPC is employed. In particular, this work adopts a control architecture inspired by *tube-based Robust Model Predictive Control (tube RMPC)*, originally introduced in [25]. Tube RMPC augments nominal MPC planning with an additional feedback mechanism that guarantees bounded deviation between the nominal and true system trajectories in the presence of bounded uncertainty.

The objective of this section is twofold. First, the theoretical foundations of tube RMPC are introduced using the canonical notation and assumptions of [25]. Second, the specific realization adopted in this thesis is described, with explicit clarification of which theoretical ingredients are implemented exactly and which are approximated to ensure computational tractability and compatibility with the considered simulation framework.

4.1.2. Disturbance-invariant tube RMPC theory

Let us consider the discrete-time linear system

$$x_{k+1} = Ax_k + Bu_k + w_k, \tag{14}$$

where $x_k \in \mathbb{R}^n$ denotes the system state, $u_k \in \mathbb{R}^m$ the control input, and w_k an additive disturbance. The disturbance is assumed to belong to a known compact, convex set

$$w_k \in \mathcal{W}. \tag{15}$$

The system is subject to hard constraints

$$x_k \in \mathcal{X}, \qquad u_k \in \mathcal{U}, \tag{16}$$

where $\mathcal{X}$ and $\mathcal{U}$ are compact, convex sets containing the origin in their interior.

The control objective is to track a reference trajectory while ensuring constraint satisfaction for all admissible disturbances. To achieve this, tube RMPC decomposes the

control action into a nominal component and an ancillary feedback term. A nominal state $x_{R,k}$ evolves according to the disturbance-free system

$$x_{R,k+1} = Ax_{R,k} + Bv_k, \tag{17}$$

where v_k denotes the nominal control input optimized by the MPC.

The applied control input is given by

$$u_k = v_k + K(x_k - x_{R,k}), \tag{18}$$

where K is a stabilizing state feedback gain chosen such that the matrix $(A + BK)$ is Schur stable[26].

Defining the error between the true and nominal states as

$$z_k = x_k - x_{R,k}, \tag{19}$$

the closed-loop error dynamics become

$$z_{k+1} = (A + BK)z_k + w_k. \tag{20}$$

If the disturbance set $\mathcal{W}$ is bounded and K is stabilizing, there exists a robust positively invariant (RPI) set $\mathcal{Z}$ such that

$$z_k \in \mathcal{Z} \quad \forall k \geq 0, \tag{21}$$

provided $z_0 \in \mathcal{Z}$. The set $\mathcal{Z}$ defines a tube around the nominal trajectory within which the true system state is guaranteed to remain.

Robust constraint satisfaction is ensured by enforcing tightened constraints on the nominal system. Since

$$x_k = x_{R,k} + z_k, \quad z_k \in \mathcal{Z}, \tag{22}$$

the nominal state and input are constrained as

$$x_{R,k} \in \mathcal{X} \ominus \mathcal{Z}, \qquad v_k \in \mathcal{U} \ominus K\mathcal{Z}, \tag{23}$$

where $\ominus$ denotes the Minkowski set difference.

This tightening guarantees that the original constraints on x_k and u_k are satisfied for all admissible disturbances.

Finite-horizon optimization problem The nominal MPC problem solved at each time step over a horizon of length N is formulated as

$$\begin{aligned} \min_{\{x_{R,k},v_k\}} \quad & \sum_{k=0}^{N-1} \ell(x_{R,k}, v_k) + \ell_f(x_{R,N}) \\ \text{s.t.} \quad & x_{R,k+1} = Ax_{R,k} + Bv_k, \\ & x_{R,k} \in \mathscr{X} \ominus \mathscr{Z}, \\ & v_k \in \mathscr{U} \ominus K\mathscr{Z}, \\ & x_{R,0} = x_R(t). \end{aligned} \tag{24}$$

Under suitable terminal conditions, this formulation guarantees the feasibility and robust stability of the closed-loop system.

The disturbance-invariant tube RMPC formulation provides formal guarantees of robust constraint satisfaction and stability under the assumptions of bounded disturbances, exact invariant set construction, and perfect knowledge of the system model. These guarantees rely critically on the availability of the RPI set $\mathscr{Z}$ and the corresponding tightened constraint sets.

4.2. Robust Model Predictive Control - Implementation

This section describes the RMPC implementation stack. The final RMPC build combines a receding-horizon optimizer with a measurement-robust augmentation layer designed for GNSS degradation.

4.2.1. State, Input, and Dynamics

The robot state and control are

$$\mathbf{x}_k = [x_k,\ y_k,\ \theta_k]^\top, \qquad \mathbf{u}_k = [v_k,\ \omega_k]^\top, \tag{25}$$

with discrete-time update

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k; \Delta t), \tag{26}$$

where Δt is the sample time and $f(\cdot)$ is the differential-drive motion model used by the simulator.

4.2.2. LiDAR-to-Obstacle Conversion

The simulated LiDAR provides range–bearing measurements of surrounding obstacles. Each scan consists of N beams, where the i-th beam returns a distance r_i at angle ϕ_i measured in the robot's local frame.

Let $\mathscr{F}_\ell$ denote the robot-fixed LiDAR frame and $\mathscr{F}_w$ the world frame. A detected point can first be expressed in Cartesian coordinates in the LiDAR frame as

$$\mathbf{p}_i^\ell = \begin{bmatrix} r_i \cos\phi_i \\ r_i \sin\phi_i \end{bmatrix}, \tag{27}$$

where r_i is the measured range and ϕ_i the beam angle relative to the robot heading.

To incorporate these points into the MPC obstacle model, they must be transformed into the world frame using the robot pose $\mathbf{x} = [x \ y \ \theta]^T$, where (x, y) is the robot position and θ its heading. The corresponding rigid-body transformation is given by

$$\mathbf{p}_i^w = \mathbf{R}(\theta)\mathbf{p}_i^\ell + \begin{bmatrix} x \\ y \end{bmatrix}, \tag{28}$$

where

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{29}$$

is the planar rotation matrix mapping points from $\mathscr{F}_\ell$ to $\mathscr{F}_w$.

The resulting point cloud in the world frame can then be clustered (using DBSCAN [27] or fallback neighbor grouping), and each cluster is approximated by a bounding rectangle (minimum-area or axis-aligned), which is subsequently converted into a polygonal obstacle for the MPC solver. Obstacles are sorted by distance and truncated to a configured maximum count by proximity to controller, yielding a compact and computationally tractable representation robust to scan noise. In GNSS-denied regions, measurement uncertainty is increased only for the robot pose estimate, while the LiDAR rays remain physically consistent. This reflects realistic conditions where exteroceptive sensing remains available despite degraded global localization.

4.2.3. Nominal MPC Problem

At each control step, the planner solves a finite-horizon problem over horizon N:

$$\min_{\mathbf{u}_{0:N-1}, \mathbf{x}_{0:N}, \mathbf{s}} \sum_{k=0}^{N-1} \left(w_s \|\mathbf{x}_k - \mathbf{x}_k^{\text{ref}}\|_2^2 + w_u \|\mathbf{u}_k - \mathbf{u}_k^{\text{ref}}\|_2^2 \right) - \lambda_s \sum s_k, \tag{30}$$

subject to dynamics, actuator bounds, acceleration bounds, and obstacle-separation constraints. The slack term s_k softens geometric constraints for feasibility in tight scenes.

Reference speed is bounded:

$$v_k^{\text{ref}} = \text{clip}\Big(k_v \, \|\mathbf{p}_k^{\text{ref}} - \mathbf{p}_k\|, \ v_{\min}, \ v_{\max} \Big), \tag{31}$$

and the optimizer runs in receding-horizon mode (apply first control, shift horizon, repeat).

4.2.4. Robust Measurement-Only Augmentation

The controller uses a measurement-only robust mode: uncertainty enters through pose measurements, while LiDAR geometry remains physically consistent.
An invariant set $\mathcal{Z}$ was not explicitly constructed; robustness is enforced via bounded correction and conservative constraint inflation.

Given a noisy pose measurement $\mathbf{z}_k$ and a predicted state $\tilde{\mathbf{x}}_k$ obtained from model propagation, a bounded measurement correction is applied to prevent excessive deviation caused by GNSS degradation.

The correction is constructed from the residual

$$\mathbf{e}_k = \mathbf{z}_k - \tilde{\mathbf{x}}_k, \tag{32}$$

and introduced as a saturated adjustment with angular wrapping applied to e_θ

$$\mathbf{x}_k = \tilde{\mathbf{x}}_k + \begin{bmatrix} \alpha_p \,\mathrm{sat}(e_x) \\ \alpha_p \,\mathrm{sat}(e_y) \\ \alpha_\theta \,\mathrm{sat}(e_\theta) \end{bmatrix}, \tag{33}$$

This operation does not constitute a stochastic state estimate, but rather enforces bounded consistency between the propagated state and the corrupted measurement. In the context of GNSS-denied operation, it acts as a robust correction mechanism limiting the influence of large measurement deviations, thereby preserving closed-loop stability of the RMPC scheme.

Next, to form the tube, the controller computes bounded state uncertainty from configured noise scales and denial-dependent multipliers:

$$|e_x| \le \bar{e}_x, \quad |e_y| \le \bar{e}_y, \quad |e_\theta| \le \bar{e}_\theta. \tag{34}$$

Obstacle constraints are tightened conservatively using

$$d_{\text{obs}}^{\text{rob}} = d_{\text{obs}} + r_{\text{tube}} + m_{\text{obs}}, \tag{35}$$

where r_{tube} is tube radius and m_{obs} is extra robust margin.

To construct the ancillary feedback, the system is linearized around the nominal trajectory

$$\delta\mathbf{x}_{k+1} = \mathbf{A}_k \delta\mathbf{x}_k + \mathbf{B}_k \delta\mathbf{u}_k. \tag{36}$$

A discrete LQR gain $\mathbf{K}_k$ (from DARE [28]) is used:

$$\delta\mathbf{u}_k = -\mathbf{K}_k \delta\mathbf{x}_k. \tag{37}$$

Feedback is clipped by configured limits, then combined with nominal MPC action:

$$\mathbf{u}_k = \mathrm{clip}\left(\mathbf{u}_k^{\mathrm{nom}} + \delta\mathbf{u}_k\right). \tag{38}$$

4.2.5. Runtime Loop

Per control cycle the final implemented RMPC controller performs the following actions

1. Read LiDAR and pose and convert scan to obstacle set.
2. Update robust estimator and uncertainty bounds.
3. Solve nominal MPC with tightened constraints.
4. Apply ancillary feedback correction and actuator clipping.
5. Execute control, log diagnostics, repeat.

4.2.6. Tuning

The performance of the implemented RMPC controller is highly dependent on a set of parameters determining controller behavior, robustness and estimator correction. Unlike stochastic learning-based methods, these parameters are not trained but selected through structured engineering tuning following four principal objectives:

1. ensure stable trajectory tracking under nominal conditions,
2. maintain constraint satisfaction under GNSS degradation,
3. avoid excessive conservatism,
4. preserve real-time computational feasibility.

The most influential tuning groups are outlined below.

Tracking weights The relative importance of trajectory tracking versus control effort is governed by the stage cost weights

$$w_s, \quad w_u.$$

Higher values of w_s improve path adherence but may induce aggressive control behavior, while higher values of w_u promote smoother motion at the expense of tracking accuracy. These were selected to prioritize collision avoidance and constraint satisfaction over exact path tracking.

Prediction horizon The horizon length N determines the controller's foresight. Increasing N improves obstacle anticipation and disturbance rejection but increases computational cost. A moderate horizon was selected to balance robustness and real-time feasibility.

Robust margins Robust obstacle separation is controlled through

$$d_{obs}^{rob} = d_{obs} + r_{tube} + m_{obs}.$$

Here:

- r_{tube} reflects bounded pose uncertainty induced by GNSS degradation,
- m_{obs} is an additional safety margin accounting for modeling and discretization errors (not useful in simulation, but necessary in real-life applications).

Larger margins improve safety but may unnecessarily restrict feasible motion, particularly in cluttered environments.

Measurement correction gains The bounded correction gains

$$\alpha_p, \quad \alpha_\theta$$

govern how strongly corrupted GNSS measurements influence the propagated state. Higher values increase responsiveness to measurement updates, while lower values prioritize model consistency. These were tuned to prevent estimator divergence while avoiding excessive sensitivity to GNSS outliers.

Tuning was performed iteratively in simulation by evaluating closed-loop behavior under nominal operation and degraded GNSS conditions. The final parameter set was selected to achieve stable tracking with low collision rates while maintaining real-time computational performance.

4.3. Proximal Policy Optimization - Theory

4.3.1. Purpose and scope

Reinforcement Learning methods aim to provide an alternative to model-based control by learning control policies directly from interaction with an environment, without requiring an explicit system model. In continuous control problems, policy gradient methods form a widely adopted class of RL algorithms due to their capability to optimize stochastic policies in high-dimensional action spaces.

Proximal Policy Optimization (PPO), introduced by Schulman et al. [9], is selected in this thesis as a representative deep reinforcement learning algorithm. PPO was designed to address the instability and poor sample efficiency of vanilla policy gradient methods while avoiding the algorithmic complexity of trust-region methods such as TRPO.This section presents the theoretical foundations of PPO, and highlights the design choices that make PPO suitable for continuous robotic control.

4.3.2. Policy gradient formulation

The control problem, as proposed in [9], is formulated as a Partially Observable Markov Decision Process (POMDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, r, \gamma)$, where the parameters making up the tuple are defined as follows. $\mathcal{S}$ denotes the state space and is a set containing all the variations of the state of the system. $\mathcal{A}$ denotes the action space – a

set containing all possible actions available to the agent in each state. $\mathcal{O}$ signifies the observation space – a set loosely connected to $\mathcal{S}$ describing the state aspects that are observable to the agent. $P = P(s_{t+1}|s_t, a_t)$ is a probabilistic function describing the transition dynamics. The remaining two elements are r and $\gamma \in (0,1)$ which represent the reward function, and the discount factor respectively.

A stochastic policy $\pi_\theta(a \mid s)$, parameterized by θ, constitutes a function mapping states to probability distributions over actions. The overall objective is the maximization of the expected discounted return

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]. \tag{39}$$

Policy gradient methods optimize $J(\theta)$ using stochastic gradient ascent, or $-J(\theta)$ using the more popular stochastic gradient descent (SGD). The policy gradient theorem introduces the gradient estimator

$$\hat{g} = \mathbb{E}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, \hat{A}_t\right], \tag{40}$$

where $\hat{A}_t$ is an estimator of the *advantage function* - a function of state s and action a returning the relative value of taking action a in state s to the estimated value of state s itself. In practice, this gradient is obtained by differentiating the objective function

$$L_{\mathrm{PG}}(\theta) = \mathbb{E}_t\left[\log \pi_\theta(a_t \mid s_t)\, \hat{A}_t\right]. \tag{41}$$

While appealing in its simplicity, repeatedly optimizing L_{PG} using the same batch of data often leads to excessively large policy updates and unstable learning.

4.3.3. Trust-region motivation

Trust region policy optimization Trust Region Policy Optimization (TRPO) [29] aims to address this instability by constraining the size of the policy update. TRPO maximizes the surrogate objective

$$L_{\mathrm{CPI}}(\theta) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)}\hat{A}_t\right], \tag{42}$$

subject to a constraint on the average Kullback–Leibler (KL) divergence [30]

$$\mathbb{E}_t\left[\mathrm{KL}\big(\pi_{\theta_{\mathrm{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)\big)\right] \le \delta. \tag{43}$$

Although this provides strong empirical stability, solving the constrained optimization problem requires second-order methods and significantly complicates practical implementation.

4.3.4. Clipped surrogate objective

PPO replaces the trust-region constraint with a modified objective that implicitly limits policy updates but remains compatible with first order stochastic optimization.

First, we define the *probability ratio* as

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{44}$$

from which it can be inferred that at the previous policy parameters $r_t(\theta_{old}) = 1$.

Building on this base, the core contribution of Proximal Policy Optimization is the *clipped surrogate objective function*, which is defined as

$$L_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t \right) \right], \tag{45}$$

where $\epsilon > 0$ is a hyperparameter controlling the maximum allowable policy change.

This objective forms a pessimistic approximation to the unconstrained surrogate, penalizing and limiting updates that would excessively increase the probability ratio and thus destabilize learning. Just like the δ parameter in TRPO, ϵ in PPO is to be determined case by case by the user and is critical to training speed and stability.

To simplify the general idea, for positive advantages, the objective discourages $r_t(\theta)$ from exceeding $1+\epsilon$, while for negative advantages it discourages $r_t(\theta)$ from falling below $1-\epsilon$. As a result, PPO achieves stable updates while remaining compatible with first-order stochastic optimization.

4.3.5. Value function and advantage estimation

In practice, the advantage function is estimated using a learned value function $V_\phi(s)$. Most commonly PPO instances employ Generalized Advantage Estimation (GAE), defined as

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{46}$$

where

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \tag{47}$$

and where $\lambda \in [0,1]$ controls the bias–variance trade-off.

When policy and value function parameters are trained jointly, PPO optimizes the combined objective

$$L(\theta) = \mathbb{E}_t \left[L_{\text{CLIP}}(\theta) - c_1 L_{\text{VF}}(\theta) + c_2 S\big(\pi_\theta(\cdot \mid s_t)\big) \right], \tag{48}$$

where L_{VF} is a squared-error value function loss and S denotes an entropy bonus

encouraging exploration of the environment. Furthermore, the c_1 and c_2 parameters can be described as *loss weighting parameters* and serve to balance three competing objectives inside the joined function.

4.3.6. Optimization procedure

PPO is initialized with a policy π parametrized by the neural network θ. It subsequently alternates between data collection using the current policy and several epochs of minibatch stochastic gradient ascent on the surrogate objective function described earlier. Unlike vanilla policy gradient methods, PPO reuses collected data for multiple optimization steps while maintaining stability through the clipped objective. The entire optimization loop can be seen depicted in Fig. 4.1

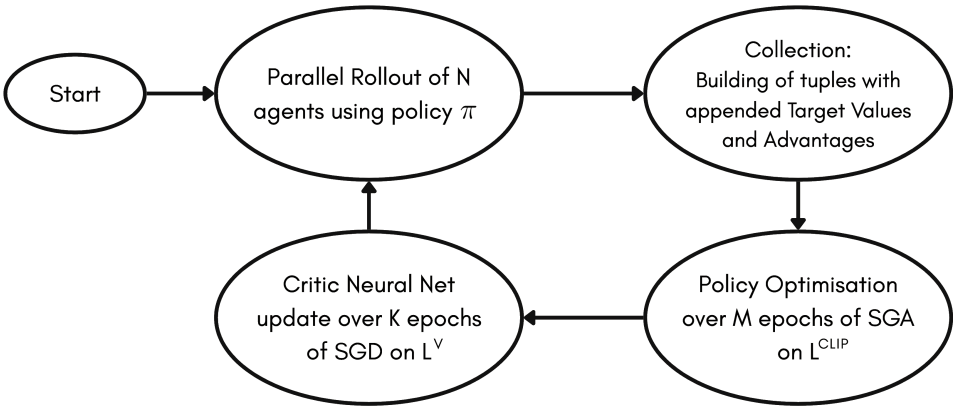

Figure 4.1. PPO training procedure

4.3.7. Remarks on guarantees

PPO does not provide formal monotonic improvement guarantees comparable to those of TRPO. However, empirical results demonstrate that the clipped surrogate objective effectively limits destructive policy updates while maintaining high sample efficiency and implementation simplicity. This trade-off makes PPO particularly attractive for continuous control tasks in robotics.

4.4. Proximal Policy Optimization - Implementation

This section documents the implementation of the PPO controller as software. It details the construction of the observation vector, reward shaping and behavior of the training loop.

4.4.1. Action Interface

The policy outputs a 2D normalized action:

$$\mathbf{a}_t = [a_t^{(v)},\ a_t^{(\omega)}], \quad a_t^{(\cdot)} \in [-1, 1].$$

At the environment boundary, this is mapped to simulator commands:

- linear velocity command v_t,
- angular velocity command ω_t,

using configured actuator limits. The environment stores both normalized action and mapped command for reward and logging.

4.4.2. Observation Construction

The observation is a fixed-length vector in $[-1, 1]$, described as:

$$\mathbf{o}_t = [\text{LiDAR bins},\ \text{tail features},\ \text{reference features}].$$

The phrase *LiDAR bins* refers to raw LiDAR ranges clipped to $[0, r_{\max}]$ and then min-pooled into n_{bins} bins and normalized by $r_{\max}$.
The *tail features* consist of:

- normalized distance-to-goal,
- previous linear command (normalized),
- previous angular command (normalized),
- cos(goal bearing error),
- sin(goal bearing error).

And the *reference trajectory features* consist of $N_{\text{ref}} = 30$ closest forward reference points from a monotonic nearest-index anchor, which are then encoded as:

$$(\text{distance to point},\ \text{heading to point}).$$

with both being clipped and normalized to fixed ranges to improve the controller's perception of the environment.

Therefore, the configured dimension of the observation vector satisfies

$$dim_{\text{obs}} = n_{\text{bins}} + 5 + 2 \cdot N_{\text{ref}}$$

4.4.3. Reference Indexing and Tracking Signals

The environment keeps a monotonic reference index (non-decreasing) within a search window. This avoids jumping backward along the trajectory and stabilizes both observation and reward terms. This reference index helps to define the term determining tracking quality for both training and the final comparison – Cross Track Error (XTE) [31], [32], which represents the distance from the robot to this indexed point on the reference trajectory.
Its implementation is described in more detail in section 5.1. Generally XTE is computed as Euclidean distance from the robot to the nearest unpassed reference point.

For reward shaping, *progress* is measured as a distance to a configurable forward point in the current horizon, named *the progress anchor*.
Letting $\mathbf{r}^\star$ be that anchor point, we define progress as:

$$\Delta d_t = \left\| \mathbf{p}_{t-1} - \mathbf{r}^\star \right\|_2 - \left\| \mathbf{p}_t - \mathbf{r}^\star \right\|_2 .$$

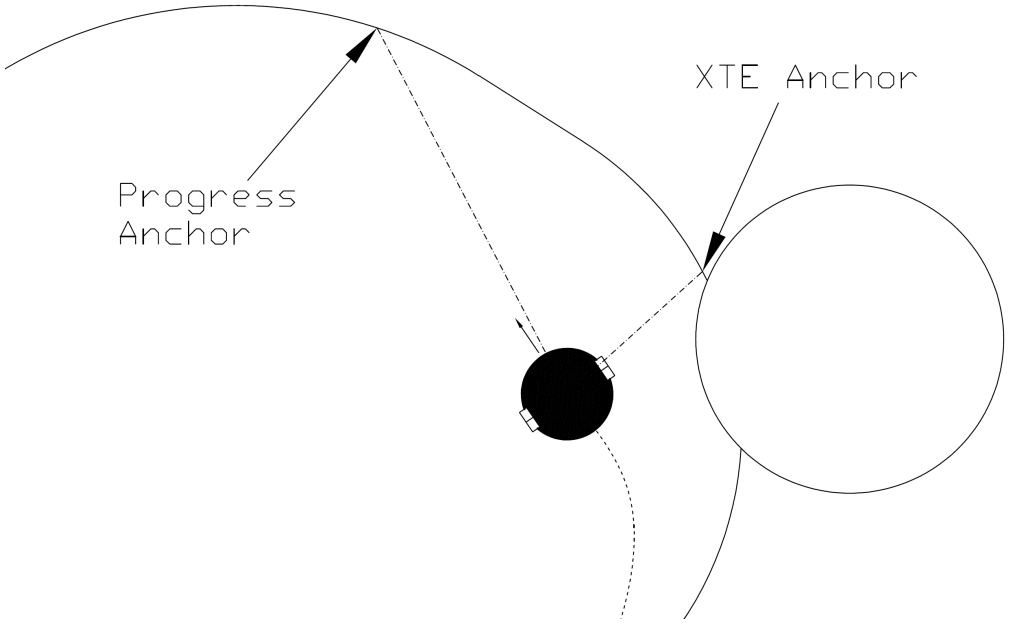

Figure 4.2. Representation of the XTE and Progress anchor points

4.4.4. Reward Implementation

At each step, reward is assembled from additive components:

$$r_t = r_t^{tracking} + r_t^{progress} + r_t^{limits} + r_t^{smoothness}$$

The *tracking* term encourages accurate following of the reference trajectory. The *progress* term ensures the agent is motivated to complete the task of reaching the final destination. The *limits* term concerns itself with obstacle proximity and the *smoothness* term penalizes jolts in inputs and attempts to impose input limits on the controller. The multiple σ parameters are scaling factors and were adjusted multiple times throughout the training process to convey the priorities of the author.

Table 4.1. Reward decomposition

Group	**Reward Term**
Tracking	
Cross-track error	$-\sigma_{XTE}\left(\frac{\min(\text{XTE}_t, XTE^{CLIP})}{XTE^{CLIP}}\right)^{\sigma_{XTEpower}}$
XTE improvement	$\sigma_{\Delta XTE}(\text{XTE}_{t-1} - \text{XTE}_t)$
Progress	
Goal reach	flat goal-reach reward
Forward progress	$\sigma_{progress} \cdot \Delta d_t$
Limits	
Obstacle proximity	$\sigma_{obstacle} \cdot \min(d_{obstacle} - d_{safe}, 0)$
Imminent collision	flat penalty if $d_{obstacle} < d_{imminent}$
Off-track condition	penalty for excessive deviation from reference path
Smoothness	
Angular velocity penalty	$-\lvert\omega_t\rvert \cdot \sigma_{omega}$
Linear velocity penalty	$-\lvert v_t\rvert \cdot \sigma_{velocity}$
Angular acceleration penalty	$-\lvert\varepsilon_t\rvert \cdot \sigma_{epsilon}$
Linear acceleration penalty	$-\lvert a_t\rvert \cdot \sigma_{accel}$

4.4.5. Training

In order to achieve a fully trained PPO agent in the shortest amount of time, and while utilizing the least amount of computational time and power, the author utilized a multi-stage training curriculum to slowly increase difficulty for the agent. This approach provides significant savings in computation time and provides the person "teaching" the agent with full control at each step. It does however create a "hands-on" training process which greatly increases the man-hours required.

The training was split into a *pretrain* and a *posttrain* stage, the former of which being further split into 10 phases. Each of these phases has its own set of training and evaluation worlds, which rank in difficulty from absolutely trivial in phase 1 (straight line - no obstacles - no denial) to medium difficulty worlds with turns, obstacles and denial zones. In order to promote a high success rate, a gate system was introduced on the phases to ensure the model is sufficiently trained to pass onto the next phase.

An additional gain from having multiple short runs at the beginning of the training process is the ability to use that time in a tuner-like fashion to find the PPO hyperparameters under which it makes the most progress. Using the pretraining, posttraining could benefit from sensible learning rate (lr) and entropy ranges as well as a baseline for what a successful run means.

Posttraining operated on much longer runs on the full training dataset with periodic evaluations and only minor tweaks to parameters like entropy and lr but with its own set of rewards which varied in scale from those of pretraining.

Presented below in fig. 4.3 is an example training run aimed at decreasing XTE while preserving a high success rate. The parameters monitored during the training process are the success and collision rates, the Cross Track Error as well as a reward estimator and a training speed estimator

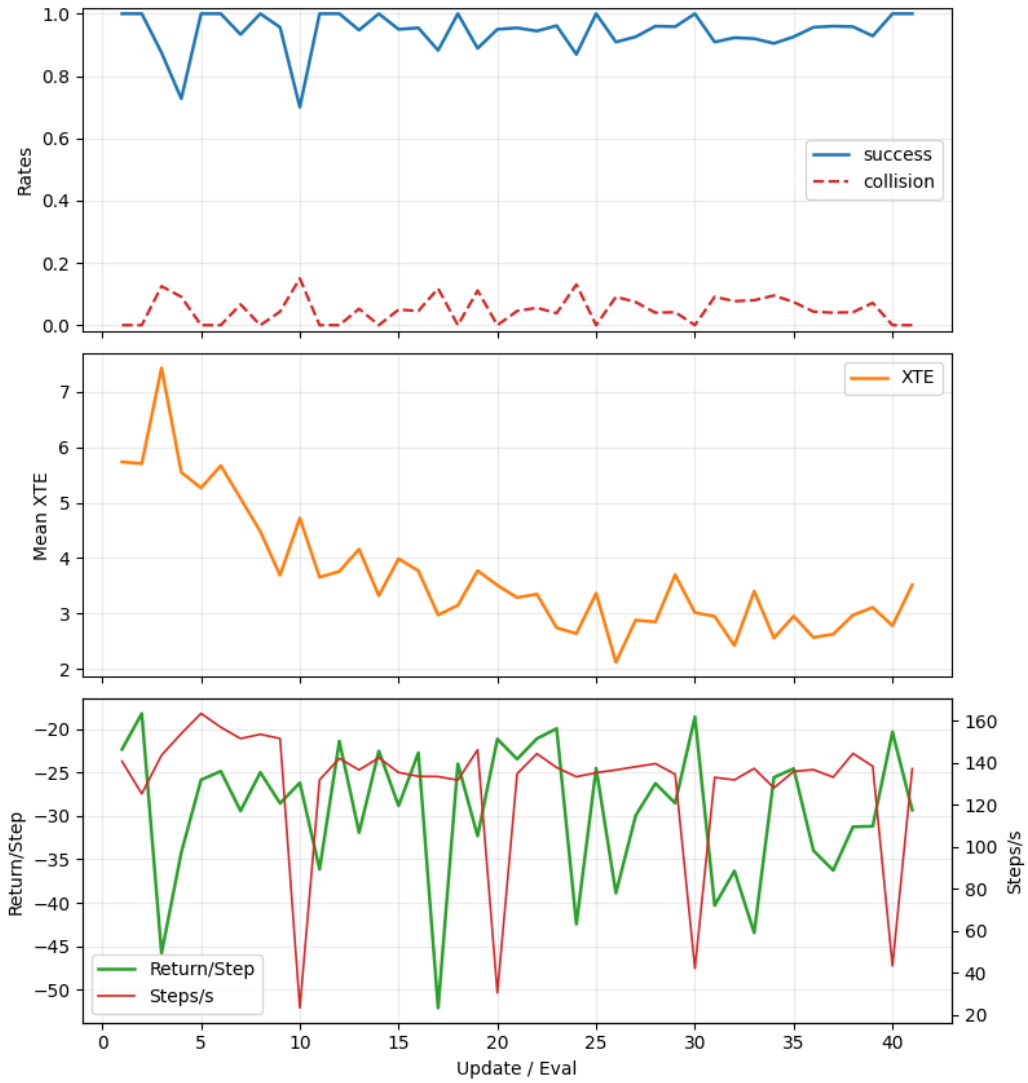

Figure 4.3. Success and collision rates, XTE and training quality estimators

and below in fig. 4.4 you can see the result of aggregate data gathering during training. These parameters weren't key indicators during training, but serve well to demonstrate the many phases of training and their varying purposes.

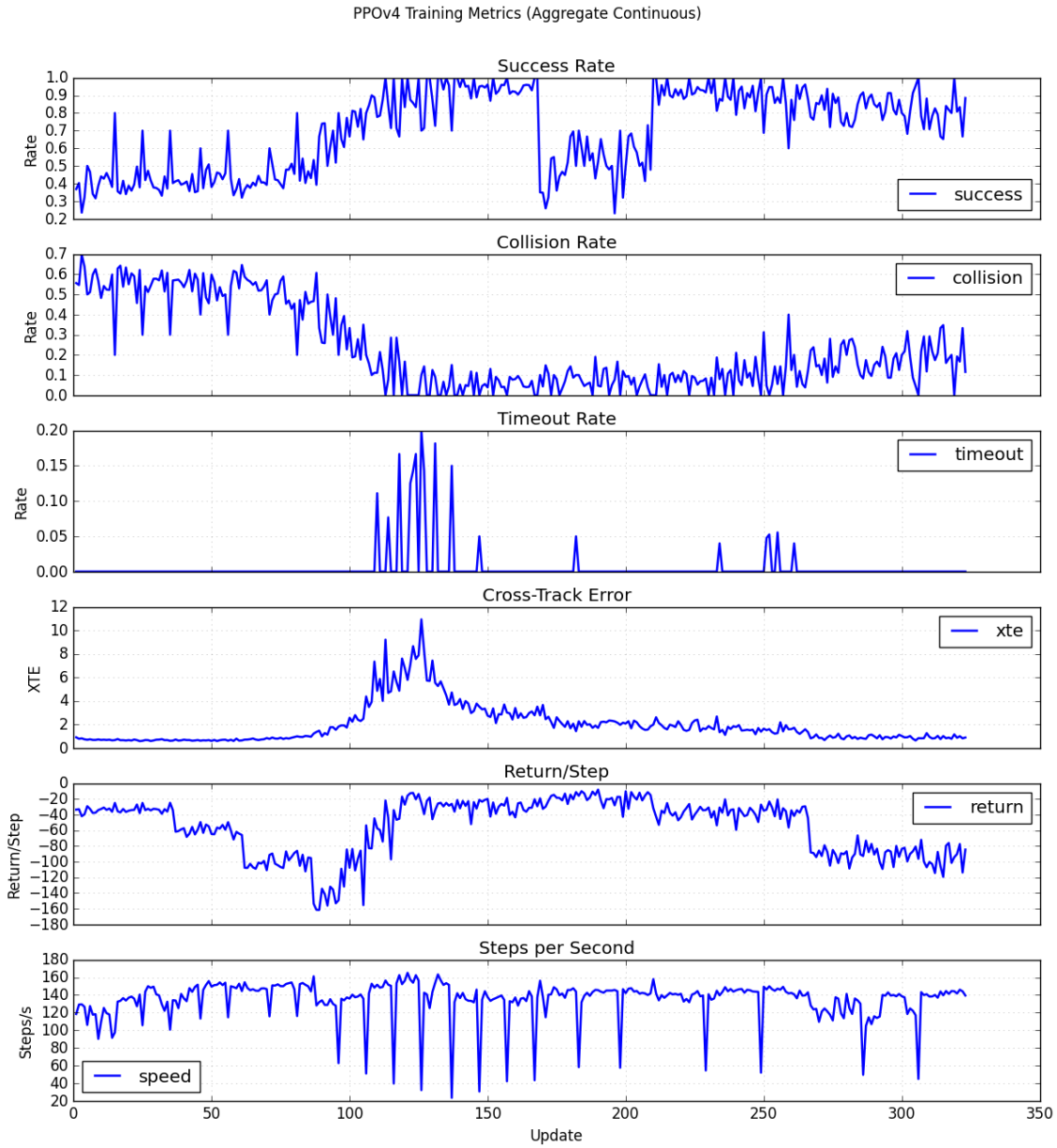

Figure 4.4. Aggregate training summary over multiple training runs

5. Benchmarking and results

5.1. Basis of analysis

The analysis of the performance of these two controllers can be performed in three distinct ways. Firstly, a pure performance comparison will consist of analyzing each of the controllers' degradations under GNSS-denial in comparison to its nominal operation. Secondly, a direct head-to-head comparison between the two controllers shall be conducted, to determine which of them experiences the sensor degradation more severely. Thirdly, a cost analysis of the two control strategies shall be conducted, aiming to compare the on-line computation cost of the RMPC with the off-line computation cost accompanying the training of the PPO agent.

In order to ensure a level playing field for comparison of the two controllers, both of them need to exhibit very low collision rates in their respective evaluation environments. Without the completion of this condition, a controller cannot be deemed operational and fit for comparison. In such a situation, the tuning parameters of the failing controller shall be iteratively modified until it is capable of reaching the desired control quality.

Having ensured the proper operation of both controllers, a need for nuance arises in the analysis of their relative quality. The basis for this more nuanced evaluation will be a measure of distance from the reference trajectory called Cross-Track Error (XTE)[31].

Since in the simulation, the reference trajectory is expressed as a discrete set of waypoints $\mathscr{R} = \{\mathbf{r}_i\}_{i=1}^{N}$, with $\mathbf{r}_i = [x_i, y_i]^\top$ and the robot position at timestep t being expressed as $\mathbf{p}_t = [x_t, y_t]^\top$, the cross-track error is the Euclidean distance to the nearest reference waypoint:

$$i_t^\star = \arg\min_{i \in \mathscr{I}_t} \|\mathbf{p}_t - \mathbf{r}_i\|_2^2, \qquad XTE_t = \|\mathbf{p}_t - \mathbf{r}_{i_t^\star}\|_2.$$

We minimize squared distance for efficiency (same minimizer as $\|\cdot\|_2$), then take the square root for the final XTE value. The set of available indexes i_t to consider is a monotonic set $\mathscr{I}_t = \{i_{t-1}^\star, \ldots, N\}$, which prevents backward index jumps and yields smoother progress estimates along the path. With dense waypoint sampling, this is a standard discrete approximation of distance-to-curve.

When comparing each controller's behavior during GNSS-denial scenarios with its unhindered operation, it is critical to analyze success, collision and timeout rates, however to provide the necessary information on tracking quality, the following parameters shall be analyzed on a statistically significant number of samples:

1. XTE_{nom} - Average XTE in non-GNSS-denied environments (nominal)
2. XTE_{mean} - Average XTE in GNSS-denied environments
3. XTE_{in} - Average XTE in GNSS-denied environments - within denial zones
4. XTE_{out} - Average XTE in GNSS-denied environments - outside of denial zones

In degradation comparison between controllers, since their absolute performance is generally a function of the user's skill in tuning the controllers' hyperparameters, the analyzed metrics will concern purely the *percentage degradation (P_{deg}[%])* in tracking. For both controllers the two types of percentage degradation will be measured as follows:

$$P_{deg-nom} = (\frac{XTE_{mean}}{XTE_{nom}} - 1) \cdot 100\% \tag{49}$$

$$P_{deg-InOut} = (\frac{XTE_{in}}{XTE_{out}} - 1) \cdot 100\% \tag{50}$$

These two metrics broadly encapsulate the significance of the degradation on the controllers performance. $P_{deg-nom}$ analyses how the introduction of GNSS denial zones negatively impacted the overall tracking ability, while $P_{deg-InOut}$ quantifies stability in the rejection of this disturbance.

To put these parameters into perspective, a $P_{deg-InOut}$ far greater than 0% signifies a quick return to tracking upon exiting of a GNSS-denial zone, while a $P_{deg-InOut}$ far lesser than 0% demonstrates that the GNSS-denial zones influence the controller to such an extent that it is not able to recover after exiting one, and in fact worsens the situation with its attempts.
Similarly, a $P_{deg-nom}$ greater than 0% is to be expected, as we can't hope for an improvement in tracking under worse conditions, however this parameter will elaborate on the extent of the tracking worsening.

Finally, the cost analysis will focus on the time and workload related with the creation of each of the controllers, as well as both of their computational needs off and on-line.

The utilized metrics will be

1. Implementation time [man-hours]
2. Training (tuning) time [hours]
3. Rollout speed [actions/second]

These metrics will answer all performance-related questions as well as those related to speed-of-implementation. The results of this comparison section will most likely turn out highly subjective since comparing a mostly off-line system with a mostly on-line one is a question of use case and not of objective truth.

5.2. Results – Raw data

Both controllers were deployed on a sample of $N = 500$ environments from the *comparison dataset* to which they had not had access during their respective training processes. The worlds for each controller and each denial configuration were identical to enable a fair and unbiased comparison. The raw measured results are as follows

Table 5.1. Monte Carlo performance benchmarks on $N = 500$ runs

Controller	RMPC (nominal)	PPO (nominal)	RMPC (denial)	PPO (denial)
Success [%]	89.8%	93.6%	85.2%	91.3%
Collision [%]	0.0%	3.1%	0.0%	4.7%
Timeout [%]	10.2%	3.3%	14.8%	4.0%
XTE_{nom}/XTE_{mean}	$0.2664[m]$	$0.7312[m]$	$0.3245[m]$	$0.8515[m]$
XTE_{in}	–	–	$0.3209[m]$	$0.2099[m]$
XTE_{out}	–	–	$0.3276[m]$	$0.8667[m]$

From these results we can see that even during GNSS denial, our accurate LiDAR measurements can bring collision rates close to zero, and that both of the controllers perform noise rejection well enough to ensure a high success rate with few timeouts.

5.3. Results – Pure performance comparison

Looking at the data in Table 5.1 we can see the differences in performance between the two controllers. Firstly, the algorithm dominating when it comes to pure success rate is Proximal Policy Optimization. Both in GNSS-denied and non-denied scenarios it achieves admirable success rates of over 90%. In this category, the slightly less successful RMPC achieves success rates of close to 90% in the nominal scenarios and 85% when experiencing GNSS denial. However it must be said that this is due to the definition of *success* as the reaching of the final goal on the reference trajectory.

When we look at the parameter potentially more important than success - collisions - we see a clear superiority of the RMPC. The formal guarantees that it provides have made it possible for the algorithm to achieve 0.0% collision rates over all testing environments. Such a low collision rate is something the PPO controller does not share. Despite this, the author is certain that a collision rate close to zero is very much possible, by means of further reward tuning and modifications to the observation vector, however this might be difficult to achieve and require a much lengthier training process.

Analyzing the third tracked parameter, the timeout rate, we can observe a side effect of RMPC's unbreakable collision constraints: a large indecisiveness and expected conservatism. Where on one hand PPO, motivated by progress and goal rewards, finds no problem with bending the rules of obstacle proximity, RMPC considers them binding and

often runs out of time in simulation. When brought into the real world, the RMPC algorithm would exhibit this property by getting into infinite loops and being unable to progress. In situations where the autonomous vehicle is not retrievable by humans (caves, heights) a timeout is basically equivalent to a collision, especially when it comes to UAVs which will collide with the earth when their battery runs out.

When it comes to tracking accuracy RMPC significantly outperforms PPO, with the latter being almost 3 times *further* off-course than the former. Similarly to the question of collisions, when implemented correctly RMPC will always perform at the height of its abilities, while a Deep Reinforcement Learning algorithm implemented correctly will still have a lot of room for improvement in most cases.

5.4. Results – Head-to-head

When applying equations 49 and 50 to the data from Table 5.1 we can see that RMPC reaches degradation percentages of

$$P_{deg-nom} = (\frac{0.3245}{0.2664} - 1) \cdot 100\% = +21.81\%$$

$$P_{deg-InOut} = (\frac{0.3245}{0.3276} - 1) \cdot 100\% = -0.95\%$$

while PPO achieves the far more surprising score of

$$P_{deg-nom} = (\frac{0.8515}{0.7312} - 1) \cdot 100\% = +16.45\%$$

$$P_{deg-InOut} = (\frac{0.2099}{0.8667} - 1) \cdot 100\% = -75.78\%$$

Firstly, to compare their degradations from the nominal state, we can see that they are fairly similar. In the case of the RMPC, which must explicitly account for any expected disturbances, a 22% worsening of tracking accuracy is an acceptable score. It possesses no inbuilt algorithm for denoising the incoming observation vector and therefore must rely on the engineer modeling the environment to know the extent of possible disturbances.

On the other hand, PPO experiences this situation from a completely different perspective. Since we were able to model the disturbances to provide their extents to the RMPC, we can also train the PPO directly on a simulation with GNSS-denial included. Therefore for the DRL agent the baseline is the denial scenario while the non-denied scenario is just a simplification. What follows is that we would expect a significantly bigger degradation percentage.

One potential explanation of this phenomenon is the size of the policy and value function NN sizes. Choosing a neural network too small for the task will most certainly result in underfitting - a complete misunderstanding of the task - which is not the case, therefore it is possible that the chosen neural network was too large, resulting in overfitting.
Overfitting can most simply be described as a scenario in which a neural network performs excellently on the task it was trained on (GNSS denial scenarios) however it significantly unperformed on tasks similar to the original one (non-denied scenario). This topic leaves space for discussion and for future work to be done in this area.

5.5. Results – workload

The workload associated with both of these controllers constitutes one of the biggest differences between them. On one hand we have the fully on-line RMPC controller requiring significantly lower maintenance during its creation and tuning but necessitating a significant amount of live compute power, and on the other we have the mixed on and off-line PPO controller which takes much greater skill and workload to initialize, but when trained runs on much less processing power. The data for this section was estimated by the author during the realization of this thesis, and is presented below in table 5.2. All simulations both training and the full Monte Carlo rollout were held on a 16 core AMD Ryzen 7 7730U CPU.

Table 5.2. Workload data | Sample size - 1 (Author)

Controller	Tube RMPC	PPO
Implementation time [man-hours]	24	120
Training/tuning duration [h]	2	250
Rollout speed [actions/s]	20.2	1270

When it comes to the difficulty and time-intensiveness of implementation, all classical control algorithms will naturally be optimal. Therefore, it is to be expected that RMPC would constitute the significantly smaller fraction of time spent on this work. Proximal Policy Optimization required seemingly endless tweaks and adjustments to its hyperparameters. Additionally, since its performance is directly proportional to the quality and quantity of data it receives, it is therefore significantly likely to require large amounts of processing power during training. This quality alone can have the capability to disqualify the use of PPO when low-cost of the implementation is a priority.

However, as it can be expected, when it comes to on-line performance we can observe PPO faring significantly better against the much slower RMPC. This is because in the PPO's mind the problem has already been solved and generalized, while RMPC treats every situation it is in as completely new. This difference results in a staggering 70-fold difference in performance speeds in actions per second in favor of the PPO.

6. Conclusion and discussion

This thesis set out to compare two vastly different control approaches – a classical control representative in the form of Robust Model Predictive Control and a Reinforcement Learning representative as a Proximal Policy Optimization agent – both applied to the same context of trajectory tracking under degraded positional information.

The motivation for this thesis originates in the increasing demand for fully autonomous systems capable of operation in scenarios where GNSS signals are unreliable or unavailable. To reiterate the problem statement, the goal of this dissertation was not to eliminate GNSS reliance entirely, but to evaluate how model-based and data-based approaches react to uncertainty stemming from one of the most crucial and trustworthy systems in navigation.

The conducted simulations demonstrated that both control methods have their place in this kind of application and both are viable under nominal conditions, but that under positional uncertainty their behaviors diverge in meaningful ways.

The RMPC controller demonstrated its stability and predictability by successfully maintaining all constraint satisfaction even when its access to the world state was degraded. Explicitly and conservatively handling uncertainty in the observations resulted in strong reliability obtained at the expense of agility and computational speed.

The PPO-based controller exhibited greatly contrasting characteristics when encountering GNSS-denial. It displayed broad adaptive responses to sensor degradation as it reduced its timeout rate to almost zero, by searching for alternative routes in challenging environments. Additionally it demonstrated how quickly it can operate on-line, which can result in a significantly lower need for on-line computing power for the same quality of navigation

From an applicability standpoint, the performed comparison highlights a clear tradeoff between the two controllers:

- RMPC provides undeniable safety and predictability
- PPO provides flexibility and adaptability

Neither method can nor should be universally regarded as superior. Instead, their suitability depends on system priorities. RMPC is better aligned with safety-critical applications where constraint satisfaction and stability are essential, while PPO may offer advantages in complex or poorly modeled environments where adaptability is more important than guarantees. An important conclusion to draw from both of the

performances of the controllers is that the world difficulty (characterized by the extent of disturbances) was most certainly towards the easy side, which greatly favored the RMPC controller. In environments with more unpredictable dynamics we are likely to see a shift in performance favoring RL methods.

Ultimately, this work supports the view that GNSS-denied navigation is not purely a localization problem, but a control robustness problem. Both model-based and learning-based strategies offer viable — but fundamentally different — responses to this challenge.

6.1. Future work

Several avenues for future work could expand on the findings of this thesis.

Firstly, the performance of both controllers should now be tested on a real-world differential drive robot equipped with LiDAR and GNSS. Such an undertaking would require addressing actuator dynamics, sensor latency as well as far more complex noise characteristics affecting all sensors. Experimental validation on physical robotic platforms is therefore a natural continuation of this work.

The implemented RMPC framework relied on a practical measurement-robust formulation rather than a fully disturbance-invariant tube MPC. Future research could investigate more exact tube constructions or adaptive tightening methods to reduce conservatism while retaining guarantees.

The PPO controller depended heavily on manually designed reward structure and training curriculum. More systematic reward design or automated tuning methods could improve robustness and reduce sensitivity to training setup.

Such developments would deepen understanding of how different control paradigms behave under degraded sensing and contribute toward more reliable autonomous navigation systems.

References

[1] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[2] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2017.

[3] D. Q. Mayne, M. M. Seron, and S. V. Raković, "Robust model predictive control: A survey", *Automatica*, vol. 41, no. 7, pp. 1137–1148, 2005.

[4] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey", *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[5] E. F. Camacho and C. Bordons, *Model Predictive Control*, 2nd ed. Springer, 2013.

[6] A. Bemporad and M. Morari, "Robust model predictive control: A survey", in *Robustness in Identification and Control*, Springer, 1999, pp. 207–226.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[8] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning", *International Conference on Learning Representations*, 2016.

[9] J. Schulman et al., "Proximal policy optimization algorithms", *arXiv preprint arXiv:1707.06347*, 2017.

[10] B. Tribelhorn and Z. Dodds, *Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education*. 2007, pp. 1393–1399. DOI: 10.1109/ROBOT.2007.363179.

[11] S. Weiss et al., "Vision-based navigation and mapping for autonomous flight in gps-denied environments", *IEEE International Conference on Robotics and Automation*, 2012.

[12] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator", *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[13] Ouster Inc. "Forest biodiversity powered by three dimensional deep learning and the ouster os0". Accessed: 2026-02-20. [Online]. Available: https://ouster.com/insights/blog/forest-biodiversity-powered-by-three-dimensional-deep-learning-and-the-ouster-os0.

[14] M. Bloesch et al., "Robust visual inertial odometry using a direct ekf-based approach", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[15] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Athena Scientific, 2012, vol. 1.

[16] M. V. Kothare, V. Balakrishnan, and M. Morari, "Robust constrained model predictive control using linear matrix inequalities", *Automatica*, vol. 32, no. 10, pp. 1361–1379, 1996.

[17] S. V. Raković, "Tube model predictive control", in *Handbook of Model Predictive Control*, Springer, 2019, pp. 327–350.

[18] T. Haarnoja et al., "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning", *International Conference on Machine Learning*, 2018.

[19] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image", *Robotics: Science and Systems*, 2017.

[20] A. Loquercio et al., "Dronet: Learning to fly by driving", *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[21] C. Tan, "Comparative study of reinforcement learning performance based on ppo and dqn algorithms", *Applied and Computational Engineering*, vol. 175, pp. 30–36, Jul. 2025. DOI: 10.54254/2755-2721/2025.AST24879.

[22] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks", *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2018.

[23] IR-SIM Development Team, *Ir-sim documentation*, https://ir-sim.readthedocs.io/en/stable/, Accessed: 2026-02-08, 2026.

[24] D. Allen, *Inverse kinematics notes*, https://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf, Accessed: 2026-02-08, 2017.

[25] W. Langson, I. Chryssochoos, S. Raković, and D. Mayne, "Robust model predictive control using tubes", *Automatica*, vol. 40, no. 1, pp. 125–133, 2004, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2003.08.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109803002838.

[26] S. Białas, "A sufficient condition for schur stability of the convex combination of the polynomials", *Opuscula Mathematica*, vol. 25, no. 1, 2005, Accessed: 2026-02-11. [Online]. Available: https://www.opuscula.agh.edu.pl/vol25/1/art/opuscula_math_2502.pdf.

[27] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", in *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 1996, pp. 226–231.

[28] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. Dover Publications, 2007.

[29] J. Schulman et al., "Trust region policy optimization", *International Conference on Machine Learning*, 2015.

[30] S. Kullback and R. A. Leibler, "On information and sufficiency", *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. DOI: 10.1214/aoms/1177729694.

[31] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011.

[32] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm", Carnegie Mellon University, Tech. Rep., 1992.

List of Figures

List of Tables