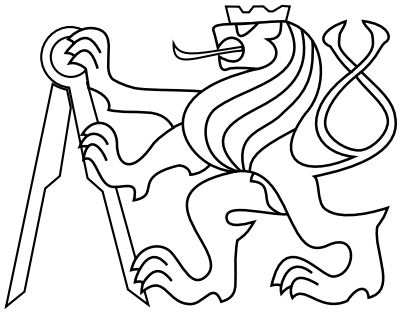
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science
Multi-robot Systems



Planning For Autonomous Aerial Interception Of UAVs

Master's Thesis

Yevhenii Kubov

Prague, May 2024

Study programme: Open Informatics
Branch of study: Artificial Intelligence

Supervisor: Ing. Matouš Vrba

Acknowledgments

Firstly, I would like to express my gratitude to my supervisor Ing. Matouš Vrba for his great support and providing all the necessary information during writing this thesis. I'm grateful to the MRS team members for guiding me during the experiments and providing consultations.

I would also like to thank my parents and friends for their support.

Finally, I would like to express my respect and deep gratitude to all those who have helped Ukraine and its citizens in any way and those who defend the country.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kubov Yevhenii** Personal ID number: **492322**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master’s thesis details

Master’s thesis title in English:

Planning for autonomous aerial interception of UAVs

Master’s thesis title in Czech:

Plánování pro autonomní odchyt bezpilotních letounů

Guidelines:

The main aim of the thesis is to compare different methods for planning agile trajectories for autonomous interception of small unmanned aerial vehicles.
• Extend the existing implementations of planning based on Model Predictive Control (MPC) developed by the MRS group.
• Research Reinforcement Learning (RL)-based planning algorithms, implement selected algorithms and integrate them with the MRS UAV system.
• Test the existing proportional navigation-based planner, both the baseline and the extended MPC planner, and the developed RL planner in realistic simulations under various conditions and compare them using relevant metrics.

Bibliography / sources:

1. R. Yanushevsky, "Modern Missile Guidance," CRC Press, 2007.
2. M. Guelman, „Proportional Navigation with a Maneuvering Target,“ in Transactions on Aerospace and Electronic Systems, vol. AES-8, no. 3, pp. 364-371, 1972.
3. Matouš Vrba, Viktor Walter, Václav Pritzl, Michal Pliska, Tomáš Báča, Vojtěch Spurný, Daniel Heřt and Martin Saska, „On Onboard LiDAR-based Flying Object Detection,“ arXiv preprint cs.RO 2303.05404, 2023.
4. T. Báča, D. Heřt, G. Loianno, M. Saska and V. Kumar, „Model Predictive Trajectory Tracking and Collision Avoidance for Reliable Outdoor Deployment of Unmanned Aerial Vehicles,“ in International Conference on Intelligent Robots and Systems, 2018.

Declaration

I declare that the presented work was developed independently, and that I have listed all sources of information used within, in accordance with the Methodical instructions for observing ethical principles in preparation of university theses. The use of generative AI tools was conducted in accordance with the official guidelines of the Czech Technical University. ChatGPT was used for grammar checking and minor test reformulation.

Date ....

Abstract

As the popularity of Unmanned Aerial Vehicles (UAVs), commonly known as drones, continues to grow, concerns regarding the potential misuse of them have become more actual. One of the emerging challenges in the area of drone security is the interception of intruder drones, especially when their presence can lead to injuries or breaks the law. Intercepting a non-cooperating drone requires a sophisticated approach, and one promising branch of this technology involves deploying an interceptor drone. For this purpose a fast and robust approach for planning the interception trajectory must be used. In this thesis an Model Predictive Control (MPC)-based trajectory planner and Reinforcement Learning (RL)-based control strategy are implemented. Their effectiveness, speed and robustness are assessed, compared and tested in simulations. The MPC-based planner is also tested in a real-world experiment.

Keywords Drone Interception, Reinforcement Learning, Model Predictive Control, Artificial Intelligence, Unmanned Aerial Vehicles

Abstrakt

Popularita bezpilotních helikoptér, běžně známých jako drony, stále roste, stejně jako obavy ohledně jejich možného zneužití. Jedním z nových bezpečnostních problémů je nedostatek efektivních prostředků ochrany před drony, zvláště v případech kdy jejich přítomnost může vést ke zranění nebo porušuje zákon. Zachycení nespolupracujícího dronu je komplikovaný problém. Jedna slibná větev výzkumu v této oblasti zahrnuje nasazení odchytového dronu. Pro tento účel je potřeba vyvinout rychlý a robustní algoritmus plánování odchytových trajektorií. V této práci je implementován plánovač trajektorií založený na prediktivním řízení a strategie řízení založená na posilovaném učení. Jejich účinnost, rychlost a robustnost jsou zhodnoceny, porovnány a testovány v simulacích. Plánovač založený na prediktivním řízení je otestován i v reálném experimentu.

Klíčová slova Odchyt Dronů, Posilované Učení, Prediktivní Řízení, Umělá Inteligence, Bezpilotní Prostředky

Abbreviations

FOV Field of View

GNSS Global Navigation Satellite System

IMU Inertial Measurement Unit

LiDAR Light Detection and Ranging

ROS Robot Operating System

RTK Real-time Kinematics

UAV Unmanned Aerial Vehicle

PN Proportional Navigation

MPC Model Predictive Control

NMPC Nonlinear Model Predictive Control

RL Reinforcement Learning

ODE Ordinary Differential Equation

MDP Markov Decision Process

PPO Proximal Policy Optimization

TRPO Trust Region Policy Optimization

DQN Deep Q-Network

LSTM Long Short-Term Memory

A2C Advantage Actor-Critic

ReLU Rectified Linear Unit

Contents

1 Introduction

The increasing popularity of small multi-rotor UAVs has undeniably revolutionized many industries. Their modern history began around 2000s, when electronic parts such as microcontrollers, IMUs and propulsion systems have become sufficiently miniaturized and inexpensive, making the final product affordable for a common person and businesses (Fig. 1.1). At that time, drones started to be seen not only as a research platform, but also as a toy, a flying camera, or a commercial tool leading to immense growth in the drone market. With the increasing amount of production UAV models of various sizes, they have been applied to simplify many tasks. Among them is cinematography: aerial shots used to be done with manned vehicles like planes or helicopters. In many cases, these costly platforms can be replaced by a drone, greatly reducing the costs [34]. Another example is industrial inspection: for instance, each photovoltaic power plant needs to be regularly inspected. Thermal imaging can unveil faults such as malfunctioning cells (Fig. 1.2), and employing the UAV to cover the rather large area of a power plant reduces the costs and speeds up the process [39]. Also, UAVs have been used for different delivery [18] or search and rescue tasks in hardly reachable areas [33] and many others.

Figure 1.1: Photo of a consumer-grade filming drone DJI Mavic 3[1].

However, the rapid development of drones has also raised notable concerns and challenges related to safety, security, privacy, and regulatory compliance. In the wrong hands, a UAV can pose a physical threat, cause material and non-material damage. Therefore, a new problem has emerged: how to limit and interrupt unwanted drone operations.

[1] "DJI Mavic 3 Classic" photo: https://store.dji.com/cz/product/dji-mavic-3-classic (Accessed: 2024-04-25)

[2] "Pv Panel Review with Thermal Drone" https://mapperx.com/en/thermal-drone-with-pv-panel-review/ (Accessed: 2024-04-25)

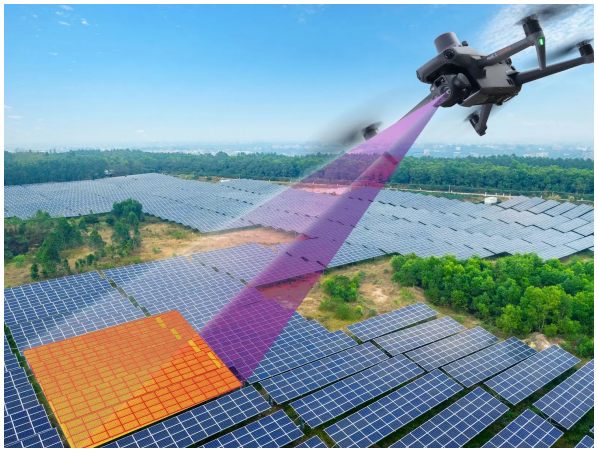


Figure 1.2: Visualization of a drone inspecting a solar power plant. Image credit[2].

1.1 Risks of unauthorized drone deployment

Unauthorized operation can happen both due to one's ignorance or deliberate bad intentions. Such operations of UAVs near restricted airspace like airports can compromise the safety of manned aircraft and passengers. It can potentially lead to collisions, accidents, and disruptions in flight operations. Adversary drones flying over critical infrastructure or military bases[3] can pose a significant threat to national security by gaining secret information or by causing physical damage on impact. The use of drones for filming or surveillance without consent can violate privacy rights by capturing sensitive personal information. Additionally, drones deployed without proper permissions can interfere with emergency services[4], reducing the effectiveness of firefighters, search and rescue missions, police operations and thus put lives at risk.

1.2 Drone mitigation techniques

Interrupting the operation of an aerial vehicle is not a new topic. Since the beginning of aviation, people have been developing different countermeasures against aerial threats. For this purpose, rifles and shotguns were mainly used. However, at the time of the First World War, there was a rise of special anti-aircraft gun development [50], followed by even more sophisticated approaches after the Second World War. Some of these methods are reviewed in this section, along with recently developed counter-drone measures.

Physical elimination by a projectile

For the purpose of physical elimination of drones, the same basic principles or even the same systems can be employed as for anti-aircraft and anti-rocket defense. Examples are anti-aircraft guns (e.g. the Flakpanzer Gepard), surface-to-air missiles (e.g. the Patriot), or short-range man-portable systems like the the Stinger.

[3] "UAV 'Attacks' German Military Base Training Ukrainian Troops On Leopard Main Battle Tanks : BILD Report" https://www.eurasiantimes.com/uav-attacks-german-military-base-training-ukrainia/ (Accessed: 2024-04-29)

[4] "Konfliktní provoz dronu v oblasti Národního parku České Švýcarsko." https://www.caa.cz/news/konfliktni-provoz-dronu-v-oblasti-narodniho-parku-ceske-svycarsko/ (Accessed: 2024-04-25)

However, this method has numerous major drawbacks and can pose an even bigger threat than the original target. The first problem is the price-per-shot, which can reach tens of thousands or even millions of dollars[5]. Another consideration is the debris caused by the drone and the missile after the impact, which renders this approach unusable over populated areas in most scenarios. Also, there is a possibility that the projectile will not perform as expected in case of a miss and therefore fall or explode in an unexpected location. Finally, destroying the intruding drone is not always desirable - for example, due to legal reasons or to allow further analysis.

Directed energy weapons

Directed energy weapons can disable the target without using a solid projectile by using a highly focused energy beam instead (laser, microwave, etc.). Especially interesting are laser systems, which have been extensively tested by militaries during the recent years[6]. These systems are seen as an alternative to the conventional anti-aircraft and anti-rocket defenses, offering greatly reduced costs (it was reported that price per shot can be as low as $ 13)[7]. Apart from manned aircraft and rockets, such systems are able to target UAVs and damage their critical components like motors, sensors, or computers [20]. As for the case of shooting down the UAV, eliminating it using a laser causes the drone to fall uncontrollably and likely also produces debris that may result in damage on the ground.

Jamming

The main idea of jamming-based drone interception is to disrupt the communication of the drone with its operator and/or GNSS [14]. That typically causes the drone to lose control and enter a fail-safe mode. Most consumer drones in the case of communication loss will perform a "return to home" maneuver or land immediately. It is a relatively safe and cheap method, but its drawback is that the drone in some cases can land or fall in an undesirable area, causing a damage. Furthermore, the exact behavior in case of jamming depends on many factors and is hard to predict.

GNSS Spoofing

Spoofing can deceive the onboard GNSS receiver by sending false signals. That causes the drone to lose its navigation capabilities or even fly in the direction desired by the defender [35]. In combination with this method, jamming of control frequencies can also be used to break the communication link with the drone operator. It is one of the safest methods to alter the intruder drone behavior. On the other hand, this method relies on the assumption that the drone's position is controlled via a GNSS feedback control loop. Also, GNSS spoofing can affect other receivers in the area.

[5] "A U.S. 'ally' fired a $3 million Patriot missile at a $200 drone. Spoiler: The missile won." https://www.washingtonpost.com/news/morning-mix/wp/2017/03/17/a-u-s-ally-fired-a-3-million-patriot-missile-at-a-200-drone-spoiler-the-missile-won/ (Accessed: 2024-04-25)

[6] "UK confirms progress on DragonFire laser weapon" https://ukdefencejournal.org.uk/uk-confirms-progress-on-dragonfire-laser-weapon/ (Accessed: 2024-05-22)

[7] "Air defense for $13 a shot? How lasers could revolutionize the way militaries counter enemy missiles and drones" https://edition.cnn.com/2024/03/13/europe/britain-air-defense-laser-dragonfire-intl-hnk-ml/index.html (Accessed: 2024-04-25)

Mid-air interception by another vehicle

Another option for intruder drone interdiction is using a specialized interceptor drone. This method is inspired by the era of dogfights, when manned aircraft were fighting each other in close air-to-air combat. Using the interceptor UAV against the intruder has several advantages over using a full-scale aircraft. First, if the intruder is small, relatively slow, yet agile (like the consumer-grade drones), it is easier to target it with something that has similar dynamics. Catching or shooting it with a manned helicopter or airplane is a more complex task, which requires more people and resources compared to the mentioned approach. Another problem is the financial efficiency. The operational costs of manned aircraft are higher than those for drones by orders of magnitude.

There are several options for utilizing drones in such scenario. The first one is using a small and cheap agile self-destructive drone[8] to directly hit the target or explode in its proximity. However, this approach suffers from the same safety issues as mentioned in sections 1.2.1 and 1.2.2. Another option is to attach a net to the interceptor drone and catch the intruder using it. This allows to prevent the drone from falling and causing damage on the ground and to recover it intact after the interception.

An example of such approach is the Eagle.One UAV (Fig. 1.5). As described by its developers, it autonomously navigates towards the estimated position of the hostile drone, continuously updating its location [5]. Utilizing onboard cameras and sensors, the drone precisely tracks the target [21], [15]. Artificial intelligence is used for drone detection, classification, and planning offensive maneuvers.

The offensive action involves using a net to capture the hostile helicopter, either autonomously or with operator confirmation. Once captured, the net remains connected via a rope, ensuring the safe removal and transportation of the target to a designated area[9].

1.3 Related works

Missile Guidance

The problem tackled in this thesis, is related to missile navigation. The same algorithms (for example Proportional Navigation) are used as a baseline solution for interception trajectory planning. Therefore, there is a need to understand the basics of homing methods.

In [24], a brief introduction into the problematic, evolution and application of guidance systems in the aerospace industry is presented. The book explains various aspects of guidance law design, frequency response analysis, the integration of classic control theory principles, and proportional navigation.

The paper [49] covers the math involved in PN method, including the proofs. The limitations and requirements for the successful pursuit are mathematically analyzed and it is demonstrated that the required missile acceleration is bounded.

Target detection

The target detection problem is not the focus of this thesis and the target position estimator is considered a blackbox providing state estimation of the target.

[8] "Anduril-Anvil" https://www.anduril.com/hardware/anvil/ (Accessed: 2024-05-22)
[9] "Eagle One" https://youtu.be/hEDGE7ofX1c (Accessed: 2024-05-22)

In [2], the authors introduce a novel approach for detecting and localizing flying objects, especially suitable for dynamic aerial interception. The method is designed for implementation onboard a UAV, leveraging a 3D LiDAR sensor for data input. It utilizes a 3D occupancy voxel mapping technique for target detection and uses a cluster-based multiple hypothesis tracker. The authors argue that in comparison to the existing methods, this approach offers enhanced localization accuracy, robustness, and an extended detection range. These attributes render the detector well-suited for applications requiring agile multi-robot interaction, such as autonomous aerial interception or formation control, where precise, reliable, and swift relative localization of other robots is essential. The practical applicability and performance of the system were demonstrated in this paper through both simulated and real-world experiments. This approach can also be used for real-world deployment of the algorithms implemented in this thesis.

In [12], another approach is used to tackle the target detection. The authors utilize only the onboard camera as a data input and employ a modified variant of the YOLO algorithm [3] to get the detection. Only ground-based targets are considered in this article, but the algorithm can be used to identify also aerial targets.

In [15], a similar method is used. The authors use an onboard camera and a neural network to identify flying UAVs. The algorithm is designed specifically to detect aerial targets, which makes it suitable for use in a real-world deployment of the methods implemented in this thesis.

Another possible solution for aerial target detection is using a ground-based radar and analyzing the radar cross-section signatures [10]. Obtained detections can then be sent to an interceptor UAV using a communication link. However, this approach is not easily applicable due to the complexity of the required infrastructure. Also, the cost and size of radar equipment can be a limiting factor.

Model Predictive Control

An explanation of the MPC problem formulation and basic underlying concepts is provided by [45].

The book "Predictive control for linear and hybrid systems"[26] provides a detailed introduction to MPC theory, applications and the optimization involved. The authors discuss techniques needed to design predictive control laws, validate and implement them.

An MPC-based approach is used as a trajectory tracker in the MRS UAV System [22]. This tracker takes desired trajectories as input and outputs commands for the controller. Authors use a linear MPC in combination with a non-linear state feedback. The approach described in [22] utilizes fast onboard simulation of UAV translational dynamics. Sampled states of the virtual UAV are used to generate control commands for non-linear feedback. Also, the system contains obstacle avoidance functionality facilitated by a long prediction horizon of the MPC. By using the mentioned approach, the system is able to perform high-precision agile maneuvers. However, the tracker is designed to track a predefined sampled trajectory and not generate one.

Reinforcement Learning

RL is another algorithm that is extensively used in this thesis. Especially interesting for this purpose is a Deep Reinforcement Learning which involves using neural networks. In [25],

an introduction into the topic is provided. The authors describe the reward-driven behavior of RL, explain the required assumptions and the challenges involved. The article provides information about both value-based and policy-based methods, covers the main algorithms in deep RL like Deep Q-Network (DQN), Trust Region Policy Optimization (TRPO) and Advantage Actor-Critic (A2C). The authors underscore the advantages of deep learning approaches and give a good overview of the field of RL in general.

RL-based approach is now widely used in robotics, and modern computers allow to create realistic world and robot dynamics models thanks to growing computational power [41]. These models are then used to train the RL algorithm in a simulation. Some of the famous applications of RL are listed next.

Games

RL was used in an AlphaGo program to beat human experts in the Go board game [36], achieving a score of 5 to 0 when playing against the European Go champion Fan Hui and 4 to 1 against the Korean professional Go player Lee Sedol (Fig. 1.3). That marked the success of the algorithm.

Figure 1.3: AlphaGo against Lee Sedol[10].

Robotics

A known task for robotic arms is reliable grasping, also called Universal Picking (UP). RL was successfully applied to manipulation tasks, where robots learn to pick objects of various shapes, sizes, and orientations. One notable example is Dex-Net, developed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR). It uses RL to train robotic arms to grasp objects and achieves high reliability [19].

Robotic applications are considered tough because the algorithm is used to interact with the physical space on a physical agent. From that fact arise numerous challenges. Due to the nature of learning algorithms, they require many interactions with the environment

[10] https://media.newyorker.com/photos/590975168b51cf59fc422f47/master/pass/House-Alpha-Go-2.jpg

during training to obtain experience. In real systems, it can be costly and require a significant amount of time. To tackle this issue, robotic problems are usually first re-created in a virtual environment where appropriate state, policy, value function, and optionally system dynamics are introduced. However, virtual models approximate the system only to some extent and do not capture all of the details, dynamics and nonlinearities. An even greater concern is that minor deviations from the real system can accumulate to a substantially different behavior for tasks with faster dynamics. Implemented RL-based control laws deployed on a real system must be robust against these factors and be able to withstand uncertainties, so the learning process often cannot be completely replaced by a simulation or requires good learning data augmentation. Designing effective rewards is often also a non-trivial task that demands a significant depth of domain expertise and is challenging in real implementations [40].

Locomotion

In locomotion tasks, such as walking, running, or navigating complex terrains, arise many challenges. The first is dynamic stability: the agent must be able not only to maintain its stable orientation in one place (which already can be non-trivial), but also to execute complex dynamic movements without losing balance. Locomotion involves coordinated movements of multiple joints and actuators and a significant part of the configuration space is unstable. Furthermore, the agent must constantly adjust its movements to respond to changes in terrain, external forces, or unexpected disturbances.

RL algorithms have been employed to teach robots (both virtual and real, and of many types) how to walk and navigate through challenging terrains. For example, Boston Dynamics' famous quadruped robot Spot depicted in Fig. 1.4 utilizes RL techniques to achieve fast and robust locomotion capabilities, enabling it to traverse various environments (which was confirmed on the Boston Dynamics official YouTube channel[11]).

Figure 1.4: Spot robot by Boston Dynamics[12].

[11] https://youtu.be/Kf9WDqYKYQQ
[12] https://bostondynamics.com/wp-content/uploads/2023/05/spot-explorer-web-2.jpg

Natural Language Processing

In the field of Natural Language Processing (NLP), RL-based algorithms were applied to a large amount of various tasks. Among them are dialogue systems and conversational agents, where RL is used to train the latter ones. It helps to generate more natural conversations that make more contextual sense and are more relevant [32].

Another example is machine translation where RL has been employed to improve the quality of translation systems by optimizing translation models based on feedback from human evaluations and other quality metrics [28][23].

Text summarization is a process of generating short summary of some larger text. The condensed version, however, must retain the main points of the original. RL has been applied for this task and achieved significant success [29].

Also, RL is used for fine-tuning or augmenting pre-trained language models like a well-known GPT to improve their performance in specific conversational tasks.

UAV control

In recent years, Reinforcement Learning was successfully applied to tackle the task of UAV control [6], [11]. In [6], the authors have conducted series of tests with different deep RL techniques and discussed their limitations. The testing scenarios were: hover, land, a random waypoint, and target following. In terms of error, PPO (which is described later in this thesis) had a better performance on the mentioned scenarios. However, the testing was conducted only in a simulation.

In [11], the authors present a learning-based approach to perform time-optimal quadrotor trajectory planning that is adaptive to environmental changes. The UAV must fly through the gates placed on a race track without perfect knowledge of the environment. The authors use the PPO algorithm to accomplish the task. The observation for the agent is composed of UAV states and relative gate position detections and the output of the network is then the thrust for individual motors. The algorithm, however, tackles a different problem than interception trajectory planning, has an incompatible formulation of the observation and the action, and therefore cannot be directly applied in this thesis.

Drone interception

In [21], the authors propose a drone detection algorithm that uses a stereo camera image as an input. It was designed specifically for the task of drone interception, and a real-life interception experiment was conducted. The interceptor used the information from the detector for real-time feedback control. The testing scenario included a non-cooperating intruder drone performing evasion maneuvers. The interceptor was able to catch the intruder using an onboard net cannon, however, the authors do not provide much explanation of the control strategy involved.

Another example of a drone used as an interceptor is presented in [5]. The authors have implemented an algorithm to catch a ball flying under another UAV using a net (a task from the MBZIRC 2020 challenge). However, the trajectory shape of the target was known (up to some parameters), so a lurking approach was used and not an active catching which is required in this thesis.

In [1], the authors propose an interception method named Fast Response Proportional Navigation (FRPN). Its purpose is to catch agile maneuvering targets while relying on onboard state estimation and tracking. The approach is based on the PN, the same as the baseline method for this thesis. The hardware that was used by the authors is similar to the one in this thesis. The proposed algorithm was evaluated in simulations and tested during real-world experiments, showing superior performance to other state-of-the-art solutions.

1.4 Problem statement

The focus of this thesis is to implement planning algorithms for capturing a non-cooperating UAV, test and compare them. The first method is using an MPC to generate a reference trajectory leading to the target. The second method is using a RL to calculate the control command which will be the reference for the interceptor UAV. The algorithms will be tested in the realistic Gazebo simulation and prepared for real-world testing. Results of the individual experiments will be discussed and compared and suggestions for future improvement will be presented.

As the interceptor UAV, the Eagle One is used. The drone has an octocopter configuration with 8 motors. The total weight is approximately 15 kilograms. It is equipped with a Pixhawk flight controller and a companion computer onboard (Fig. 1.6). The computer runs Linux and the MRS UAV System [7] based on ROS. The interceptor has a LiDAR that can detect the target as described in [2]. Positioning is performed using Global Navigation Satellite System (GNSS) and Real-time Kinematics (RTK) sensors. Low-level control of the drone is tackled by the MRS UAV System and the flight controller. As an interface between the UAV system and the algorithms implemented in this thesis, two types of commands are used: a trajectory reference[13] (used for the MPC-based trajectory planner) or the 3D acceleration and heading rate reference[14] (used for the RL-based control strategy).

The target UAV is a quadcopter also equipped with the Pixhawk flight controller and a companion computer with the MRS UAV System. Its localization is performed using RTK and GNSS and the hardware configuration is similar as in Fig. 1.6 excluding the LiDAR. During the experiments, the drone receives a predefined sampled trajectory which it then follows. The total weight of the drone is approximately 3 kilograms.

In the virtual simulation, the positions of both drones are obtained as ground truth, and an approximate Field of View (FOV) of the interceptor's LiDAR-based target detector is simulated.

To test the algorithms, close engagement scenarios between the interceptor and the target UAV are considered (tens of meters and closer).

[13] https://ctu-mrs.github.io/mrs_msgs/msg/TrajectoryReference.html (Accessed: 2024-05-22)
[14] https://ctu-mrs.github.io/mrs_msgs/msg/TrackerCommand.html (Accessed: 2024-05-22)
[15] https://mrs.fel.cvut.cz/projects/eagle-one (Accessed: 2024-04-25)

Figure 1.5: Eagle.One UAV developed by the MRS group[15].

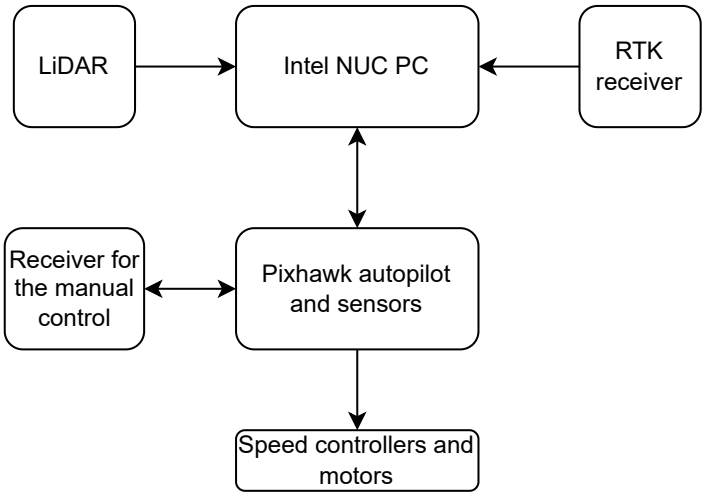


Figure 1.6: Scheme of the employed electronic hardware onboard the UAV.

2 MPC methodology

MPC is an advanced control strategy used in many engineering applications, including robotics and autonomous systems. Unlike traditional control methods, which compute control inputs based on a fixed control law, MPC works by iteratively solving an optimization problem over a finite prediction horizon. This allows MPC to deal with systems with complex dynamics, constraints and uncertainties. MPCs are also used for UAVs for example as trajectory trackers [22] or to assist learning algorithms [4] providing a reference solution.

Utilizing an internal model of the system dynamics, the controller predicts future states of the system up to a finite number of steps and computes a sequence of control inputs that minimize the desired cost function, while also fulfilling the constraints defined for the system. These constraints may include limitations on control inputs, state variable values and in some cases also the operational boundaries or other non-linear functions. To find the solution, a solver is employed.

A major drawback compared to traditional control methods is that MPC is significantly more computationally demanding. It employs a solver to find the optimal solution. However, modern computers and development of MPC frameworks allow to use this approach even in real time using commonly available hardware. The main concepts regarding MPC are explained in the following sections.

2.1 System model

MPC relies on a model of the system for predicting its future behavior, which is crucial for planning optimal control actions. The model is a mathematical representation of the dynamics of the controlled system. It describes how the system's state evolves over time in response to control inputs.

A system model can be represented in different ways and the choice depends on how expressive representation is required and which methods are used to work with it. Some common types of system models used in control engineering include a state-space matrix (Fig. 2.1), Ordinary Differential Equation (ODE) representations and transfer function representation. In this thesis, a linear continuous time-invariant formulation is used in the form:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)\,, \tag{2.1}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)\,, \tag{2.2}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the input vector, $\mathbf{y}$ is the output vector, $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are system matrices.

2.2 Prediction horizon

A prediction horizon is the future time interval, over which the system state and control actions are predicted and optimized (Fig. 2.2). For MPC, it is assumed to be finite. This sets MPC apart from regulators in the field of optimal control like LQR (Linear Quadratic Regulator) and H-infinity methods that may consider an infinite horizon.

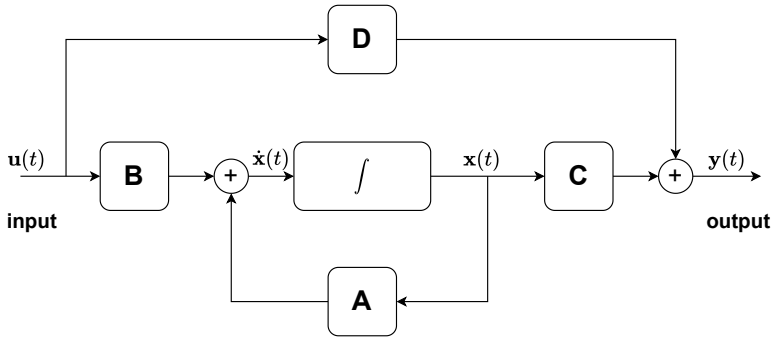


Figure 2.1: A schematic of the state-space model.

The length of the prediction horizon balances the trade-off between the computational time and long-term planning in the control strategy. A larger length means that the MPC predicts farther into the future, but is more computationally demanding and the precision of the estimate degrades over time. On the other hand, a shorter prediction horizon with a more detailed model can give estimates with better precision, but will produce an optimal solution only for the shorter time period.

The choice of the length can be limited by the dynamics of the controlled system. For systems with fast dynamics, a shorter prediction horizon is more suitable to allow real-time responsiveness. In contrast, for systems with slower dynamics and less measurement noise, a longer prediction horizon can be suitable.

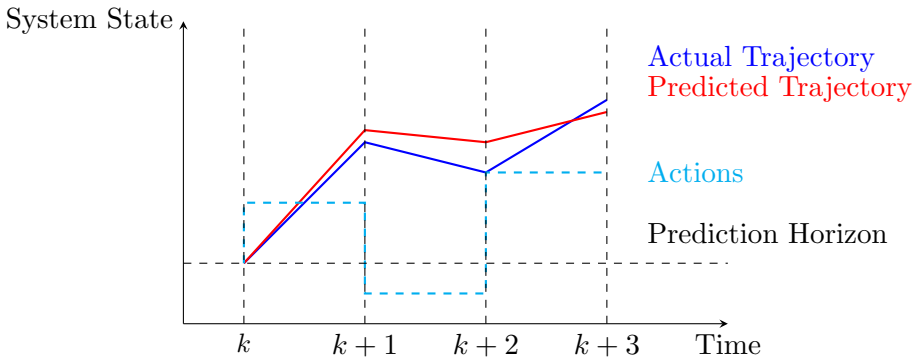


Figure 2.2: Illustration of an MPC prediction horizon starting at time k with length 3.

2.3 Cost function

In MPC, a cost function is a mandatory building block as it quantifies the performance objectives. It is a mathematical expression that evaluates the desirability of a given control state over the finite prediction horizon. The objective of MPC is to find the control trajectory that minimizes this cost function while satisfying system dynamics and constraints.

The cost function typically consists of multiple terms, each reflecting different aspects of the control system's performance and constraints. Among these terms is usually the difference between a desired state vector value and the actual value in each iteration and the magnitude of control inputs. The cost function can also have different weights or separate formulas for the so-called "stage cost" (in every step except the last) and for the "terminal cost" (in the last step). According to [45], an example of an MPC cost function is

$$J_{(N,M)}(\mathbf{x}_0, \mathbb{U}) = \min_{\mathbf{u}} \left[\mathbf{x}^T(N)\mathbf{P}_0\mathbf{x}(N) + \sum_{i=0}^{N-1} \mathbf{x}^T(i)\mathbf{Q}\mathbf{x}(i) + \sum_{i=1}^{M-1} \mathbf{u}^T(i)\mathbf{R}\mathbf{u}(i) \right], \tag{2.3}$$

where $\mathbb{U}$ is the space of possible action vectors, $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ are weighting matrices, N is the length of the prediction horizon and $M \leq N$ is the length of the control horizon (often $M = N$ holds).

The first term ($\mathbf{x}^T(N)\mathbf{P}_0\mathbf{x}(N)$) corresponds to the terminal cost. It is applied only to the last state of the system within the prediction horizon. By using it, different penalties (typically larger) are imposed on the system's state at the end of the horizon. These penalties encourage the controller to drive the system towards a desired terminal state.

The second term ($\sum_{i=0}^{N-1} \mathbf{x}^T(i)\mathbf{Q}\mathbf{x}(i)$) penalizes differences between the desired or reference state trajectory of the system and its predicted states. This term can be understood as a tracking error. Minimizing it ensures that the controlled system follows a specified trajectory or achieves a desired setpoint.

The third term ($\sum_{i=1}^{M-1} \mathbf{u}^T(i)\mathbf{R}\mathbf{u}(i)$) penalizes the control effort, which refers to the magnitude of control inputs required to drive the system towards the desired behavior. Limiting a greater control effort helps to minimize rapid changes in system behavior and in real life also the energy consumption, wear and tear on actuators, etc.

The weighting of each term in the cost function reflects the importance of different aspects relative to others. Tuning these weights allows to adjust the behavior of the controller and to prioritize specific factors mentioned above. A similar cost function is used in the MPC implementation in the following chapters.

2.4 Constraints

Constraints are another important concept in Model Predictive Control. State values of the system are often limited in range, especially in real-life applications. For instance, a point mass model representing a real-world mobile robot can have a limited position range, velocity, and acceleration given by physical limitations of the robot's actuators. These constraints can be encoded into an MPC problem and when looking for the solution, the solver will adhere to them.

Input values in an MPC controller can also be limited (and usually are). These input constraints prevent excessive or physically unrealizable control effort. Input constraints can represent limits on actuators' capacity, voltage, current, or any other physical limitations. For example, in an aircraft control system, attitude angles and throttle may be constrained to prevent aggressive maneuvers or overstressing the vehicle. Constraints that are used in most MPCs and NMPCs, can be divided into two types:

- Soft constraints, which can be violated, but this violation implies additional penalty. They enable the controller to prioritize conflicting objectives.
- Hard constraints, which must be always satisfied exactly. They represent the physical system limitations and saturations or ensure the safety and feasibility of the solution.

2.5 Control law

For practical use, sometimes only a subset of the actions from the solution is implemented. The MPC can be restarted before the end of the prediction horizon of the previous run. Therefore, if N is the length of the prediction horizon, a possible control law is to take K actions ($1 \leq K < N$), restart the MPC, and then use the fresh prediction.

2.6 Linear vs Nonlinear MPC

In linear MPC, both the constraints and model dynamics are linear and the objective is quadratic. Conversely, in Nonlinear Model Predictive Control (NMPC), some of the constraints can involve nonlinear functions, the objective can be nonquadratic [44], and also the system may contain time-varying parameters. Opting for a non-linear model changes the control problem from a convex quadratic programming to non-convex non-linear optimization. For such tasks, finding the global optimum is not guaranteed and the task is in general more

difficult compared to the linear MPC, but it allows to formulate and solve a broader class of more complex problems [46].

2.7 Solver

Numerical solvers are employed to tackle the optimization problem like MPC. Hence, at the foundation of every MPC framework lie numerical optimal control methods. Many algorithms exist and their choice greatly influences the characteristics of the controller. The main reasons for not solving such problems exactly are the computational time (as large computational times lead to greater delays) and the usage of finite precision arithmetic [30]. Designing and choosing an algorithm for numerically solving MPC problems is an actual research topic since there is no universal solution. While some solvers provide better computational speed, others are more stable and have better accuracy. As a consequence, the choice is generally based on a trade-off.

The task that is solved in MPC is the so-called initial value problem. It is described by the initial value $\mathbf{x}_0$ and the differential equations of the model [30], [42]. Since MPC is optimization-based, a definition of the optimization problem is required. Formally it can be written as

$$\begin{aligned} \min \quad & J_{N,M}(\mathbf{x}_0, \mathbb{U}) \,, \\ \text{w.r.t.} \quad & \mathbf{u} = \{\mathbf{u}(i), \mathbf{u}(i) \in \mathbb{U}\},\ 1 \leq i \leq N \,, \\ \text{s.t.} \quad & \mathbf{x}_u(0) = \mathbf{x_0} \,, \\ & \mathbf{x}_u(i+1) = f(\mathbf{x}_u(i), \mathbf{u}(i)) \,, \\ & \underline{\mathbf{x}} \leq \mathbf{x}_u(i) \leq \overline{\mathbf{x}} \,, \end{aligned} \tag{2.4}$$

where $\underline{\mathbf{x}}$ and $\overline{\mathbf{x}}$ denote the lower and upper bounds on states respectively.

2.8 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) methods are a type of solvers that work in iterations and solve quadratic programming problems obtained after approximating the original nonlinear problem by its linearization [48]. These solvers iteratively refine the solution until convergence to a local optimum is achieved. SQP solvers are widely used for MPC formulations with nonlinear cost functions and constraints. To use SQP, the objective function and the constraints must be twice continuously differentiable.

Given the optimization problem, a quadratic approximation is first obtained using this method, which gives a quadratic program (QP). This QP is then used to do one step of solution refinement, and the process is repeated.

2.9 Nonlinear Interior Point

The fundamental idea behind Interior Point Methods (IPMs) is to move iteratively through the interior of the feasible region, approaching the optimal solution without violating the constraints [43]. Unlike traditional methods that rely on the exterior of the feasible region (like the simplex method), IPMs explore the interior of the feasible region.

Both SQP and nonlinear IPMs involve evaluating problem functions and their derivatives in each iteration. SQP methods typically require fewer high-level iterations to achieve a desired level of accuracy and also excel in warm-starting [30], which is crucial in NMPC.

In practice, nonlinear IP methods are preferable for nonlinear optimization problems with inexpensive function and derivative evaluations, especially when a good initial guess is unavailable. Conversely, SQP methods are more favorable for problems with costly function evaluations, such as direct shooting methods, where good initial guesses can be provided, particularly when solving a sequence of neighboring problems [30].

2.10 The Acados MPC framework

Acados is a software package designed to provide fast and embedded solvers for nonlinear optimal control [13]. This package provides tools and main building blocks to construct a pipeline for solving optimal control problems, especially NMPC. According to the authors, the main aim was to provide SQP-type methods. It requires the problems to be formulated using the CasADi symbolic framework [16] as a nonlinear programming problem.

Acados was advised for this thesis by my supervisor and other people from the MRS group for its superior speed and ability to run on a computer with limited resources, which is also shown in [13]. It has interfaces for the C, Matlab, and Python programming languages, with the latter being used in this thesis.

3 MPC-based trajectory planner

Knowing the dynamic system constraints, MPC can be utilized as a feed-forward control action generator. Each prediction run of the MPC outputs two sequences: the first one is the sequence of control actions $\mathbf{u}(i)$ and the second one is the sequence of states $\mathbf{x}(i)$ for each time step of the prediction horizon. One possible option is to define the system model, design the objective function, periodically run the prediction, and apply the obtained control actions. In the case of the multi-rotor UAVs, designing a system model controlled directly by the motor thrust would require a low-level control over the vehicle and a very fast computational speed to ensure the stability of flight (low-level control loops for such vehicles usually run at frequencies of 200-1000 Hz). Because of that reason, a different approach is chosen which was also used before at the MRS group and which was employed in the previous implementation of the MPC trajectory planner. The details are described in the following sections.

3.1 Implementation and integration with the MRS system

The MRS UAV system interface allows for high-level control by sending a trajectory reference command. That means that actions obtained from the prediction will not be directly used. Instead, the idea is to use a sequence of predicted states as the input to the UAV controller. To create a trajectory reference command, the state predictions are transformed into a stamped trajectory reference by taking only the 3D position from each state (for the purpose of trajectory generation there is no need to model the drone as a 6 DoF body with translations and rotations). Also, a time step of the trajectory reference must be set (corresponding to the time step of the MPC). Such abstraction reduces the problem to creating a plausible sequence of future interceptor positions that will lead to a successful interception.

Although the MRS team had an existing MPC trajectory planner implementation for the interceptor, it was implemented using the doMPC framework, which has a different interface and approach to modeling the system (system matrices and discrete formulation) to those in the Acados (Ordinary Differential Equations using symbolic variables and continuous formulation). It can not be easily ported and therefore, there is a need to implement one completely from scratch while keeping the main idea similar. My contribution in this chapter of the thesis is the implementation of a new MPC planner that will replace the existing one due to its superior computational speed and therefore performance.

3.2 Model of the drone

Advanced low-level control is tackled by the existing trajectory tracker and an SE(3) controller running onboard the drone [7], [22]. The tracker receives a sampled trajectory as an input and the drone follows it. Therefore, multirotor aerial vehicle dynamics can be simplified to a 3 DoF point mass model, and the list of the predicted UAV positions through the whole prediction horizon is sampled and converted to a trajectory. This also allows for a faster computation and comprehensible decomposition of the control pipeline.

The MPC has the interceptor's acceleration 3D vector as a control input. The state

vector $\mathbf{x}$ is defined as

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \\ \mathbf{r}^{target} \\ \mathbf{v}^{target} \end{bmatrix} \in \mathbb{R}^{12}, \tag{3.1}$$

where $\mathbf{r}$ is interceptor's position, $\mathbf{v} \in \mathbb{R}^3$ is the interceptor's velocity, $\mathbf{r}^{target} \in \mathbb{R}^3$ is the target's position, $\mathbf{v}^{target} \in \mathbb{R}^3$ is the target's velocity. The control input $\mathbf{u} \in \mathbb{R}^3$ is defined as

$$\mathbf{u} = \mathbf{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \in \mathbb{R}^3, \tag{3.2}$$

where $\mathbf{A}$ is the acceleration of the interceptor. The heading is not present in the control input for the MPC since it can be explicitly computed so that the drone always faces the target.

The state derivative is then

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ \mathbf{a} \\ \mathbf{v}^{target} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{12}, \tag{3.3}$$

which is the set of ODEs describing the model. All variables can be limited by constraints. That is the form used in my implementation. In a state-space representation, the model is:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \mathbf{u}, \tag{3.4}$$

$$\mathbf{y} = \mathbf{I}\,\mathbf{x}, \tag{3.5}$$

where $\mathbf{I}$ is a 3x3 identity matrix and $\mathbf{0}$ is a 3x3 zero matrix.

3.3 Cost function

The cost function is formulated in a similar manner as in equation (2.3). In every time step $i \in \langle 0,\ N-1 \rangle$ of the prediction horizon, a value of

$$c(i) = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} \tag{3.6}$$

is calculated and added to the total cost. In the very last step N, the terminal cost is calculated as

$$c(N) = \mathbf{x}^T\mathbf{P}\mathbf{x} \tag{3.7}$$

and also added to the total cost as

$$J_N(\mathbf{x}_0, \mathbb{U}) = \sum_{i=0}^{N} c(i)\,. \tag{3.8}$$

However, to increase the readability of the practical implementation, the cost function was slightly transformed. Since the only two terms to be penalized are the distance to the

target and the velocity difference between the target and the interceptor, an error vector $\mathbf{e}$ is introduced in the following way:

$$\mathbf{e} = \begin{bmatrix} \mathbf{r}^{target} - \mathbf{r} \\ \mathbf{v}^{target} - \mathbf{v} \end{bmatrix} . \tag{3.9}$$

This vector is used instead of the state vector $\mathbf{x}$ together with simplified $\mathbf{Q}$ and $\mathbf{P}$ matrices that are denoted as $\hat{\mathbf{Q}}$ and $\hat{\mathbf{P}}$. Matrices are defined as

$$\hat{\mathbf{Q}} = \begin{bmatrix} Q_{xy} & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_{xy} & 0 & 0 & 0 & 0 \\ 0 & 0 & Q_z & 0 & 0 & 0 \\ 0 & 0 & 0 & Q_{vxy} & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_{vxy} & 0 \\ 0 & 0 & 0 & 0 & 0 & Q_{vz} \end{bmatrix}, \quad \hat{\mathbf{P}} = \begin{bmatrix} P_{xy} & 0 & 0 & 0 & 0 & 0 \\ 0 & P_{xy} & 0 & 0 & 0 & 0 \\ 0 & 0 & P_z & 0 & 0 & 0 \\ 0 & 0 & 0 & P_{vxy} & 0 & 0 \\ 0 & 0 & 0 & 0 & P_{vxy} & 0 \\ 0 & 0 & 0 & 0 & 0 & P_{vz} \end{bmatrix}, \tag{3.10}$$

where Q_{xy} is the coefficient for distance in the x and y axes, Q_z is the coefficient for distance in the z axis, Q_{vxy} is the coefficient for velocity difference in the x and y axes, Q_{vz} is the coefficient for the velocity difference in the z axis, and the notation for $\hat{\mathbf{P}}$ is similar.

The input penalization matrix $\mathbf{R}$ has the form

$$\mathbf{R} = \begin{bmatrix} R_{xy} & 0 & 0 \\ 0 & R_{xy} & 0 \\ 0 & 0 & R_z \end{bmatrix}, \tag{3.11}$$

where R_{xy} is the term penalizing the acceleration (control input) in the xy plane and R_z in the z axis. The resulting equation for the stage cost is

$$c(i) = \mathbf{e}^T \hat{\mathbf{Q}} \mathbf{e} + \mathbf{u}^T \mathbf{R} \mathbf{u} , \tag{3.12}$$

and the terminal cost is

$$c(N) = \mathbf{e}^T \hat{\mathbf{P}} \mathbf{e} . \tag{3.13}$$

3.4 Constraints

In this implementation, only linear constraints on the state and the input vector are used. There is a possibility to add also nonlinear constraints (for example a minimum distance to the target for obstacle avoidance) since a *Nonlinear* MPC framework is utilized. However, this negatively affects the computational time. Furthermore, obstacle avoidance is already available in the lower-level parts of the MRS system. For the purpose of maintaining comparability with the previous MPC planner available at MRS, the following linear constraints are used:

$$\begin{aligned} \underline{v_{xy}} \leq v_x \leq \overline{v_{xy}} , \\ \underline{v_{xy}} \leq v_y \leq \overline{v_{xy}} , \\ \underline{v_z} \leq v_z \leq \overline{v_z} , \\ \underline{a_{xy}} \leq a_x \leq \overline{a_{xy}} , \\ \underline{a_{xy}} \leq a_y \leq \overline{a_{xy}} , \\ \underline{a_z} \leq a_z \leq \overline{a_z} , \end{aligned} \tag{3.14}$$

where the underlined variables denote the corresponding lower bound and the overlined variables denote the upper bound.

4 RL methodology

RL is a concept where an agent learns its behavior by interacting with the environment (Fig. 4.1). The agent receives rewards as a result of its action, and attempts to correct its own behavior to maximise the received reward [47]. A world model can be created within a computer simulation in which way the agent can be trained faster than real-time.

4.1 RL formulation and assumptions

In RL, the state of the agent is often considered a sufficient statistic of the environment, which contains all the information needed to plan the best next action. The state vector can include the position of the agent in space, as well as the positions of its actuators [25].

The same mathematical notation as in the chapter 2 is used in this chapter to denote the inputs and states. In RL, the Markov assumption is typically used to interpret the dynamics of an environment. It states that the future state of a system depends only on its current state and the action taken, and it is independent of the past states and actions leading up to the current state. The mathematical formulation of this assumption is

$$P(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t-1}, \mathbf{u}_{t-1}, ..., \mathbf{x}_0, \mathbf{u}_0) = P(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t), \tag{4.1}$$

where $P(\mathbf{x}|...)$ is a conditional probability of getting to the state $\mathbf{x}$.

This assumption reduces the complexity of the problem by eliminating the need to consider the entire history of states and actions. It is an accurate enough approximation for many problems and allows using dynamic programming methods such as value iteration and policy iteration [27]. However, there exist different techniques where this assumption is violated. A good example are recurrent or Long Short-Term Memory (LSTM) neural networks, which can be used to capture longer-term dependencies and improve the RL algorithm performance.

General reinforcement learning problem formulation is typically modeled as an Markov Decision Process (MDP). It consists of states: $\mathbf{x} \in \mathbb{R}^n$, actions: $\mathbf{u} \in \mathbb{R}^m$, transition model: $P(\mathbf{x}'|\mathbf{x}, \mathbf{u})$, rewards: $R(\mathbf{x}, \mathbf{u}, \mathbf{x}') \in \mathbb{R}$, policy: $\pi(\mathbf{u}|\mathbf{x})$. As was mentioned, the best action $\mathbf{u}_t$ in the state $\mathbf{x}_t$ is determined with regard to R. This reward is provided by the environment on each state transition, after executing some action. For example in state $\mathbf{x}_t$, the agent executes an action $\mathbf{u}_t$. Then, it receives reward r_{t+1} (reward value is determined by the reward function R) and its state changes to $\mathbf{x}_{t+1}$. The cumulative reward is defined as

$$R_{tot} = \sum_{\forall t \in \text{episode}} R(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}), \tag{4.2}$$

where episode is the sequence of steps from the beginning of the interaction with the environment till the termination. The goal of the agent is to obtain the highest value of the cumulative reward which can be achieved by optimizing the policy. A policy π is a mapping from states to a probability distribution over actions:

$$\pi : \mathbb{X} \to p(\mathbb{U} = \mathbf{u}|\mathbb{X}), \tag{4.3}$$

where $\mathbb{X}$ is the space of possible states, $\mathbb{U}$ is the space of possible actions and p denotes the conditional probability. Therefore, the problem can be formulated more generally as

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[R \,|\, \pi] \tag{4.4}$$

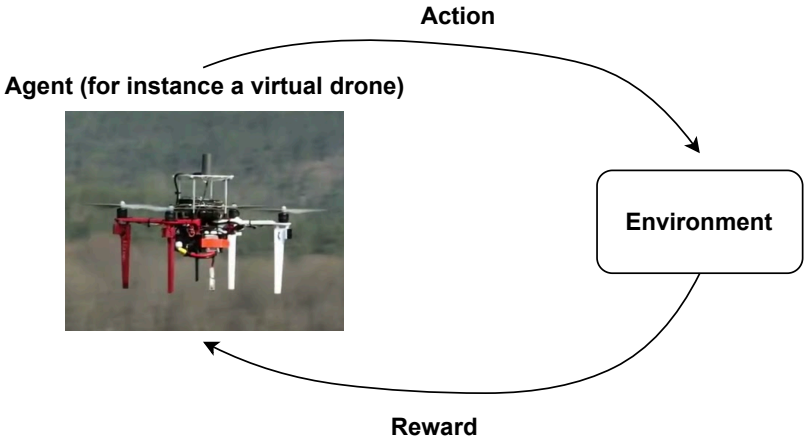


Figure 4.1: An illustration of the basic idea of reinforcement learning.[1]

where the expected value $\mathbb{E}$ of the reward R is maximized.

4.2 Brief overview of frequently used RL algorithms

Numerous RL algorithms exist. They are distinguished based on the action or state space of the environment (discrete/continuous), being off-policy or on-policy (whether the agent uses the current policy to update its values), and other details.

Temporal difference (TD) learning

TD learning is an off-policy and model-free algorithm used for discrete action and state spaces. It can estimate value functions or policy functions directly from experience without requiring a model of the environment dynamics. It combines Monte-Carlo estimation and dynamic programming ideas. In TD learning, value functions are updated iteratively based on the observed transitions from one state to another. In this way, there is no need to wait till the end of the episode. The update rule is

$$V^{\pi}(\mathbf{x}_t) := V^{\pi}(\mathbf{x}_t) + \alpha(r_{i,t} + \gamma V^{\pi}(\mathbf{x}_{t+1}) - V^{\pi}(\mathbf{x}_t))\,, \tag{4.5}$$

where V^{π} is an estimated value of a state, $r_{i,t}$ is a reward in episode i and time step t, α is a learning rate and γ is a discount factor.

Q-learning

Q-learning is another off-policy model-free algorithm for discrete action and state spaces. However, the logic behind is different. Unlike the previous algorithm, it doesn't just estimate the value function, but the state-action value function $Q^{\pi}(\mathbf{x}, \mathbf{u})$. Update rule is

$$Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t) = Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t) + \alpha\big(r_t + \gamma \max_{\mathbf{u}\in\mathbb{U}} Q^{\pi}(\mathbf{x}_{t+1}, \mathbf{u}) - Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t)\big)\,. \tag{4.6}$$

[1] UAV photo source: https://mrs.felk.cvut.cz/research/micro-aerial-vehicles

SARSA

SARSA is an on-policy model-free algorithm for discrete action and state spaces. Its update rule is very similar to Q-learning, however, it uses action $\mathbf{u}_{t+1}$ instead of max over actions:

$$Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t) = Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t) + \alpha\big(r_t + \gamma\, Q^{\pi}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1}) - Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t)\big)\,. \tag{4.7}$$

DQN

DQN is an algorithm that combines ideas of Q-learning with deep neural networks to approximate the Q-value function. It is used for continuous or discrete state-spaces and only discrete action-spaces. This algorithm is especially suitable for environments with high-dimensional state-space, such as images or raw sensor data. DQN was introduced by researchers at DeepMind [37] and gained significant attention for its ability to achieve human-level performance on a variety of Atari 2600 games. Large and high-dimensional environments when used with regular Q-learning, in order to store the Q-value table, may require amounts of memory that way exceed the capabilities of a normal personal computer (terabytes, petabytes, and more). Q-value function $Q^{\pi}(\mathbf{x}, \mathbf{u})$ is a mapping $(\mathbf{x}, \mathbf{u}) \to \mathbb{R}$ and its tabular representation requires an entry for each combination of state and action. So, the table size is itself $|\mathbb{X}| \cdot |\mathbb{U}|$ assuming a finite state-space. But with the so-called "curse of dimensionality", the table size often explodes. DQN tackles this problem by approximating the table with its neural network which maps inputs to outputs (Fig. 4.2). It most commonly includes convolutional, fully connected layers and activation functions in its so-called hidden layer.

DQN offers good sample efficiency because it uses a so-called replay buffer. Also, the state space doesn't need to be explicitly designed.

Even though the algorithm is worthy of mention, for this thesis DQN still can't be used. The action space of the drone agent representation is not discrete.

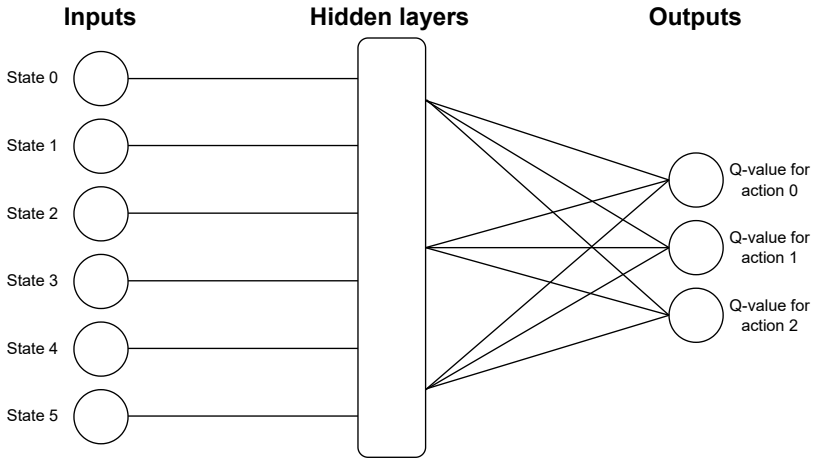


Figure 4.2: Example of a DQN architecture.

PPO

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that is used for training agents in environments with both discrete and continuous action spaces. It be-

longs to the family of policy gradient methods, which estimate the policy gradient and use it for a stochastic gradient ascent. It combines ideas from the Advantage Actor-Critic Method and Trust Region Policy Optimization (TRPO). Unlike other policy gradient methods which perform single gradient update per data sample, PPO uses objective function to perform multiple epochs of minibatch updates which improves sample efficiency [31]. Also, PPO trains reasonably fast and is good for multiprocessing. Another core concept of PPO is limiting the policy change between epochs. Large updates can, in practice, destabilize the training process, thereby reducing the likelihood of policies converging to the optimal solution. In some environments, a big change in policy leads to an entirely different behavior of the agent (incomparable to the one before the policy update) which severely affects the training.

A major difference compared to the DQN is that in policy gradient methods the policy is learned directly (which basically means action probabilities). In DQN, the action value function is learned.

A more detailed explanation of the algorithm's core principles is presented in the next section.

4.3 PPO Algorithm

As was mentioned, PPO combines ideas of A2C, TRPO and is a policy-gradient method. Some of the crucial concepts that inspired and shaped the PPO are presented next.

Policy gradient

The commonly used form of the policy gradient estimator is:

$$\hat{g} = \hat{\mathbb{E}}_t \Big[\nabla_\theta \log \pi_\theta(\mathbf{u}_t, \mathbf{x}_t) \hat{A}_t \Big] , \tag{4.8}$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimate of the advantage function at time t [31]. Advantage function describes how much better is a specific action in some state over randomly selected action in the same state according to π, under assumption that π is followed in all future steps. Mathematically speaking, the advantage function is defined as

$$A^\pi(\mathbf{x}, \mathbf{u}) = Q^\pi(\mathbf{x}, \mathbf{u}) - V^\pi(\mathbf{x}) . \tag{4.9}$$

Doing multiple steps of gradient ascent directly using formula 4.8 leads to large policy updates and therefore to poor performance and the undesired behavior mentioned above [31].

TRPO

For stability reasons, it is often good to limit the change of parameters to some extent between the learning episodes. This is the idea which was implemented in 2015 at UC Berkeley [38].

TRPO methods introduce a trust region constraint to ensure that policy updates are conservative and do not deviate too far from the current policy. An objective function is formulated as the optimization problem with a constraint formalized as limiting the KL-divergence between the new policy and the old policy:

$$\begin{aligned} \max_\theta \quad & \hat{\mathbb{E}}_t \big[\frac{\pi_\theta(u_t|x_t)}{\pi_{\theta_{old}}(u_t|x_t)} \hat{A}_t \big] \\ \text{s.t.} \quad & \hat{\mathbb{E}}_t \big[\mathrm{KL}[\pi_\theta(\,\cdot\,|x_t), \pi_{\theta_{old}}(\,\cdot\,|x_t)] \big] \leq \delta . \end{aligned} \tag{4.10}$$

Ratio $\frac{\pi_\theta(u_t|x_t)}{\pi_{\theta_{old}}(u_t|x_t)}$ is later denoted as $r_t(\theta)$.

Kullback–Leibler (KL) divergence, also called relative entropy, is defined for continuous probability distributions P and Q as

$$\mathrm{KL}(P,Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx\,. \tag{4.11}$$

Informally speaking, it means "how far are two probability distributions apart from each other".

It was reported that pure TRPO is computationally demanding compared to other methods and is hard to implement in practice because it uses the KL-divergence constraint outside of the objective function[2]. In PPO, this issue is solved.

Clipped Surrogate Objective function

Authors of the PPO have proposed to use a so-called clipped surrogate objective function

$$L^{\mathrm{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right], \tag{4.12}$$

where ϵ is a hyperparameter.

It combines two terms: the unclipped surrogate objective $r_t(\theta)\hat{A}_t$ and its clipped counterpart. By using the minimum of them, PPO ensures that the policy updates are limited to a certain range determined by ϵ. If the update is already small, the original function is used, if the update is too large, then the clipped one is selected. Concrete examples of the function output are presented in the table 4.1 taken from [8]. Using this idea, PPO clips the probability ratio directly in the objective function unlike TRPO.

$p_t(\theta)\ > 0$	A_t	**Return Value of *min***	**Objective is Clipped**	**Sign of Objective**	**Gradient**
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓

Table 4.1: Table summarizing the behavior of PPO's objective function [8].

4.4 Value and policy networks

A common implementation of the PPO algorithm requires two networks: the actor network and the critic network. While the first one determines the actions that should be taken by maximizing the expected return of the policy, the second network learns to minimize the difference between the estimated return and the actual return.

An example of the PPO architecture is shown on Fig. 4.3.

[2] https://huggingface.co/blog/deep-rl-ppo

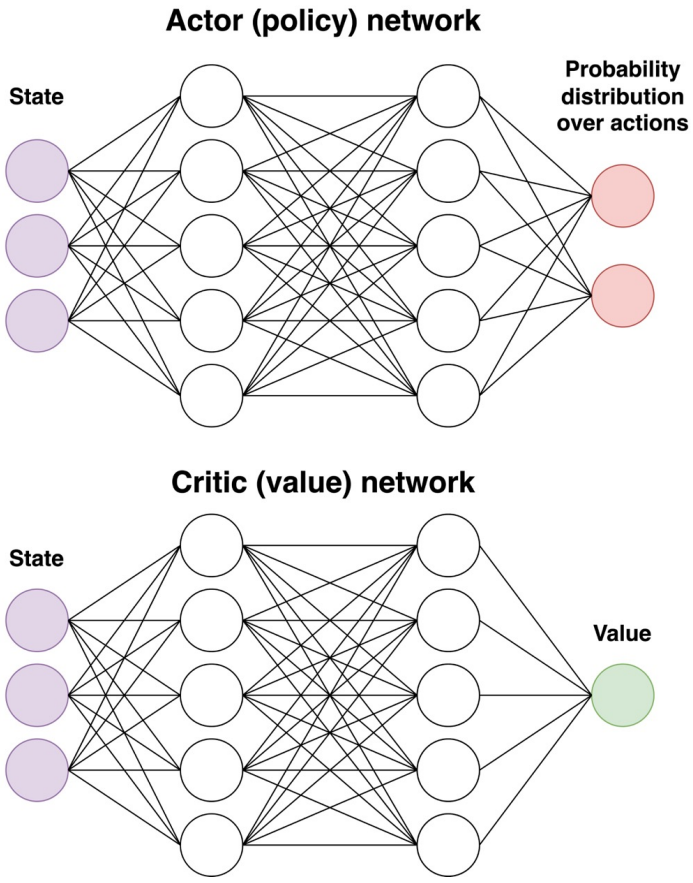


Figure 4.3: Example of a PPO architecture.

4.5 Stable baselines

Stable baselines is a library of open-source Python implementations of deep RL algorithms such as DQN, PPO, SAC, TD3, and more. It is a fork from OpenAI baselines, developed by a trusted and well-known research organization. These implementations are rigorously tested and benchmarked to ensure stability, efficiency, and scalability. Authors of this library wanted to tackle the issue of RL results in reproducibility which is often a non-trivial task [17]. The library offers a consistent interface and modularity, allowing to swap the algorithm without significant code changes, so they can be easily compared [9]. It is well integrated with the OpenAI Gym, a toolkit for developing, testing, and comparing RL algorithms. This integration allows to train and evaluate algorithms on different standard environments provided by Gym. There is also a possibility to implement a custom environment for usage with the library. Additionally, it has repositories with experimental features and community contributions.

Stable baselines' convenient modular design, high-quality tested implementations, efficiency, compatibility with OpenAI Gym, tensorboard support, and large community make it a popular choice for researchers and people curious about the field of Reinforcement Learning.

5 RL-based control strategy

The control strategy is realized by a model trained using the PPO algorithm in a simulation environment provided by the MRS group. As an input, it receives the state of the UAV and the target. Output is a control action. In the following sections, it is covered in detail.

5.1 Training environment

The agent is trained in a custom simulation environment developed by the MRS. It simulates full UAV dynamics and has an acceleration-based interface to control the drone. The environment can be used with the Stable Baselines RL framework. This simulator provides the reward and observation on each step and receives the control command as an input to control the virtual interceptor.

5.2 Observation space and action space

A state vector $\mathbf{x}_{rl}(i)$ is obtained using the following data:

$$\mathbf{x}_{rl}(i) = \begin{bmatrix} \mathbf{r}(i) \\ \mathbf{v}_{rel}(i) \\ \mathbf{a}_t(i) \\ z(i) \\ \mathbf{v}(i) \\ \mathbf{R}(i) \\ \dot{\theta}(i) \\ \mathbf{u}_{rl}(i-1) \end{bmatrix} \in \mathbb{R}^{27}, \tag{5.1}$$

where $\mathbf{r}(i) \in \mathbb{R}^3$ is the range vector to the target, $\mathbf{v}_{rel}(i) \in \mathbb{R}^3$ is the relative velocity to the target, $\mathbf{a}_t(i) \in \mathbb{R}^3$ is the target acceleration, $z(i) \in \mathbb{R}$ is the interceptor altitude, $\mathbf{v}(i) \in \mathbb{R}^3$ is the interceptor velocity, $\mathbf{R}(i) \in \mathbb{R}^{3\times3}$ is the interceptor rotation matrix, $\dot{\theta}(i) \in \mathbb{R}$ is the interceptor heading rate, $\mathbf{u}_{rl}(i-1) \in \mathbb{R}^4$ is the previous action of the interceptor. These vectors are concatenated and the rotation matrix is flattened and concatenated.

The input to the neural network consists of two state vectors stacked, where in a step i the first is the current state $\mathbf{x}_{rl}(i)$ and the second is the previous state $\mathbf{x}_{rl}(i-1)$. It helps to capture longer-time dependencies and gives information about how the state evolved. The time step between the states is 0.02 s which also corresponds to the time step of the training environment.

The action vector $\mathbf{u}_{rl}(i)$ is composed of the acceleration in 3 dimensions and the heading rate:

$$\mathbf{u}_{rl}(i) = \begin{bmatrix} a_x \\ a_y \\ a_z \\ \dot{\theta} \end{bmatrix} \in \mathbb{R}^4. \tag{5.2}$$

5.3 Networks

When researching the network architectures, it was observed that for instances with similar complexity, examples on SB3 Zoo[1] employ networks with two fully connected layers and ReLU activation function. The size differed from 64 to 256 nodes per layer. After checking the learning speed and the training results, a network with 2 fully connected layers and 128 nodes per layer creating a hidden layer was chosen as the one having the best tradeoff. ReLU activation is used. Two similar hidden layers are used for the value and for the policy network.

5.4 Reward function

To formalize the reward function, the following variables and constants are also required:

- $\mathbf{r}_{net}(i)$ range vector from the net position (below the interceptor) to the target,
- $\mathbf{r}_{body}(i)$ range vector in the interceptor coordinate frame,
- $\alpha(i)$ angle between the x-axis of the interceptor and the target,
- $\boldsymbol{\omega}(i)$ vector of angular rates,
- $d_{interception} = 0.5\,(m)$ minimal distance for interception,

where i is the step number.

The reward in step i is then defined as

$$\begin{aligned} R(i) = {} & c_{distance} \cdot ||\mathbf{r}_{net}(i)|| + \\ & c_{angle} \cdot \alpha^2(i) + \\ & c_{change\_acc} \cdot ||\mathbf{a}(i) - \mathbf{a}(i-1)|| + \\ & c_{change\_hdg\_rate} \cdot |\dot{\theta}(i) - \dot{\theta}(i-1)| + \\ & c_{\omega} \cdot ||\boldsymbol{\omega}(i)|| + \\ & c_{\omega z} \cdot |\omega_z(i)| + \\ & c_{v} \cdot ||\mathbf{v}(i)|| + \\ & c_{terminated} + \\ & c_{out} + \\ & c_{interception} \cdot \text{clamp}(1 - \frac{||\mathbf{r}_{net}||}{d_{interception}}, 0, 1)\,, \end{aligned} \tag{5.3}$$

where $c_{distance} \cdot ||\mathbf{r}_{net}(i)||$ is a distance penalty, $c_{angle} \cdot \alpha^2(i)$ heading difference penalty, $c_{change\_acc} \cdot ||\mathbf{a}(i) - \mathbf{a}(i-1)||$ acceleration change penalty, $c_{change\_hdg\_rate} \cdot |\dot{\theta}(i) - \dot{\theta}(i-1)|$ heading rate change penalty, $c_{\omega} \cdot ||\boldsymbol{\omega}(i)||$ angular rate penalty, $c_{\omega z} \cdot |\omega_z(i)|$ angular in z-axis penalty, $c_{v} \cdot ||\mathbf{v}(i)||$ velocity penalty, $c_{terminated}$ termination penalty (applied only if terminated), c_{out} out of world penalty (applied only if out of the defined space), $c_{interception} \cdot \text{clamp}(1 - \frac{||\mathbf{r}_{net}||}{d_{interception}}, 0, 1)$ interception reward (linear between zero distance and interception distance). The terms are multiplied by corresponding constants shown in Table 5.1.

The interception reward is counted only once until the interceptor gets farther from the target than $3 \cdot d_{interception}$. After that, the interception reward can be obtained again.

[1] https://stable-baselines3.readthedocs.io/en/master/guide/rl_zoo.html

$c_{distance}$	-0.09
c_{angle}	-0.0001
c_{change_acc}	-0.04
$c_{change\_hdg\_rate}$	-0.02
c_{ω}	-0.0001
$c_{\omega z}$	-0.1
c_{v}	0
$c_{terminated}$	-5000
c_{out}	-5000
$c_{interception}$	1000

Table 5.1: Constants corresponding to different terms of the reward function used in this thesis.

5.5 Training process

Apart from the network architecture, PPO requires several hyperparameters. Since they can't be analytically computed, there is a need to identify them first by trial and error or by taking the parameters that have been previously applied for a similar instance. Then they must be tuned to achieve the best performance. The initial guess of the parameters was determined after looking at the examples of PPO usage on SB3 Zoo. After doing multiple iterations of hyperparameter tuning, the following values were empirically selected based on observed performance:

- Learning rate: a function of the remaining training progress. It starts at 3e-4 in the beginning and linearly reduces with time to 1.5e-6 at the end of the training (the total amount of steps is predefined),
- Number of steps to run each environment per update: 300,
- Minibatch size: 3000,
- Number of parallel environments: 100,

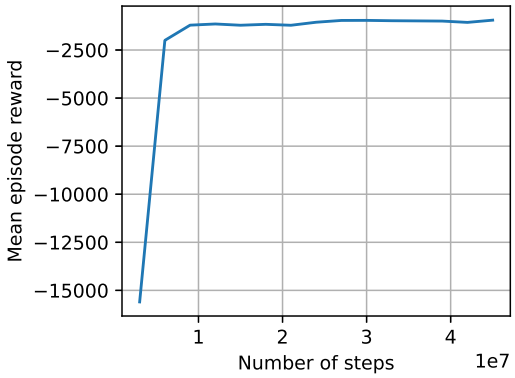
During the training, it was observed that the algorithm is prone to getting stuck in local minima. A comprehensible metric of the learning process success is the mean reward per episode. Most of the runs ended up converging to a local minimum with poor performance and a low mean reward. An example is presented on Fig. 5.1a, where the agent after 50 million steps of training receives a reward of approximately -1000 and has not improved since hitting the 10 million steps mark.

An example of not getting stuck is on the Fig. 5.1b. The value rises gradually up to approximately 2000 and the trend shows that further improvement is possible. After this experiment, the total number of steps was doubled to 100 M.

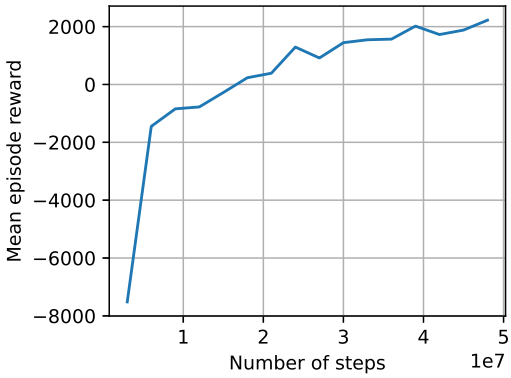
The final model was trained for a total of 100 000 000 steps until the mean episode reward reached a reasonably stable and high value of 4000 for the last 10 M steps (Fig. 5.1c).

5.6 Examples of the learned behavior

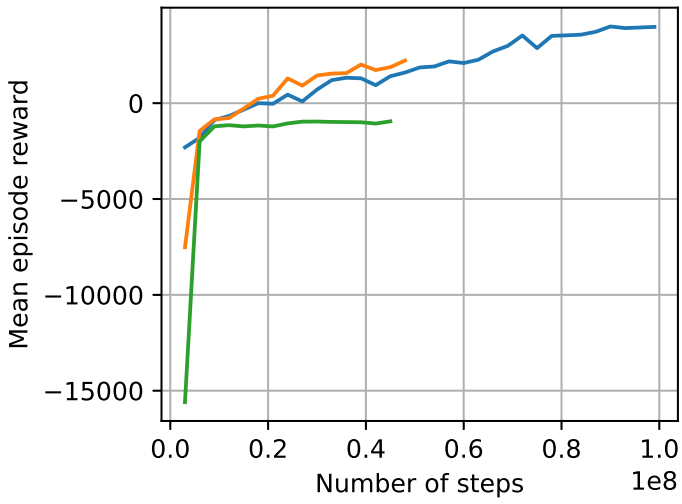
The trained model was first validated using the training environment. Some selected results are shown in the Fig. 5.2a and Fig. 5.2b. As can be observed, the agent has learned to perform some zig-zag maneuvers over the target. The behavior is logical considering the



(a) Learning stuck in a local minimum.



(b) Learns successfully for 50 million steps.



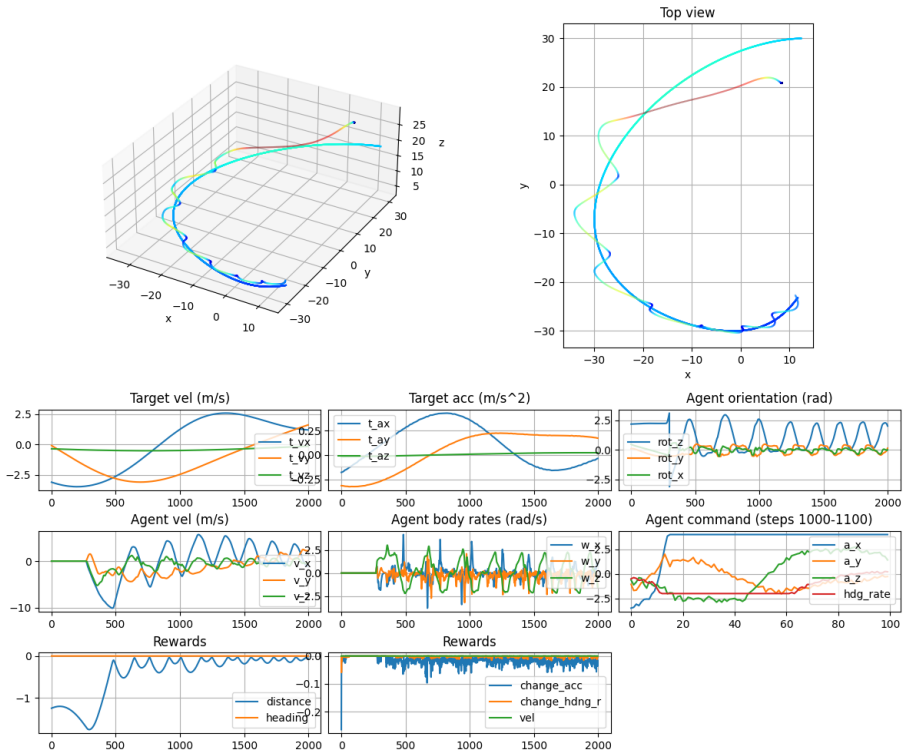
(c) Learns successfully for 100 million steps (blue line). Previous runs are shown for comparison (green and orange lines).

Figure 5.1: Mean reward per episode during the training process.

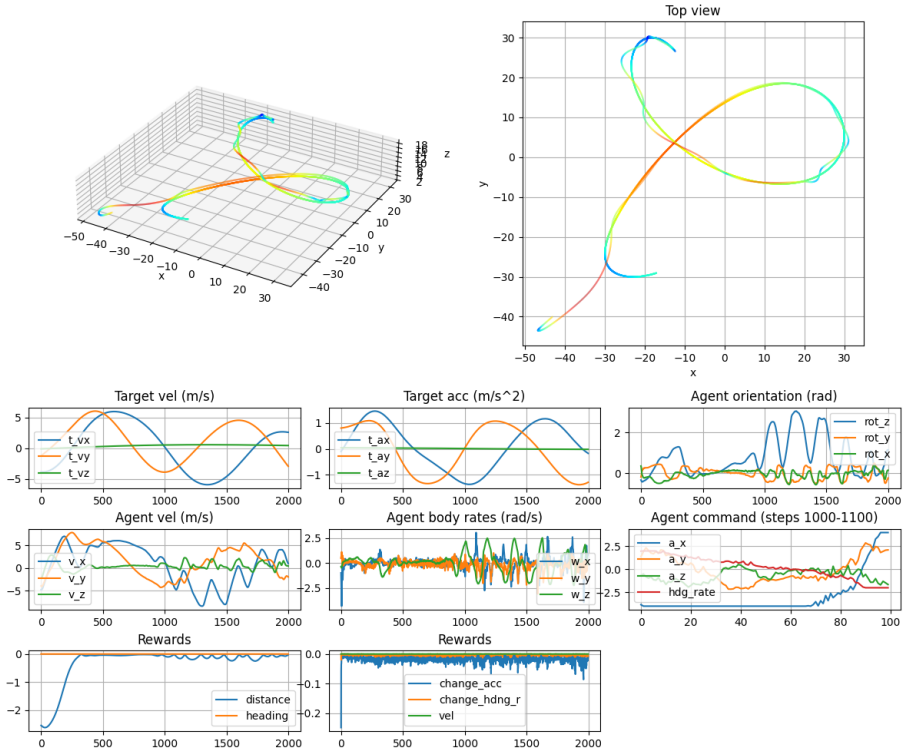
designed reward function and also makes practical sense. When doing this maneuver, the UAV will catch the target with its net while not staying too close to the target for long periods since that leads to the loss of it from the sensor's FOV. The control command is reasonable and does not suffer from oscillations.

5.7 ROS node

To be able to test the trained RL model with the MRS UAV System, it is required to implement a specialized ROS node. First, it processes the data like vehicle state and the target state into a format defined in the equation 5.1. Then, this data is used to perform a forward pass through the previously trained actor neural network. The action vector that was obtained as a result of the forward pass is processed into a ROS message and sent back to the MRS UAV System to control the drone.



(a)



(b)

Figure 5.2: Testing in the training environment. UAV positions and selected data are visualized. The control command plot is zoomed to show only 100 steps to visualize the lack of rapid oscillations.

6 Simulation

To safely evaluate the efficiency and performance of the algorithms proposed for autonomous aerial interception of UAVs, a realistic simulation environment must be used in the first place. Direct testing on real robots can be dangerous and in case of unstable or incorrect algorithm implementation, such deployment will lead to unpredictable behavior. Therefore, accurate virtual testing is required before proceeding with any real experiments. The Gazebo simulator was used for this task because it provides an accurate representation of real-world scenarios and robot dynamics and offers the flexibility to control the virtual world. It is a popular environment for performing mobile and aerial robotics experiments. Its seamless integration with the rviz tool, ROS, and a long history of its extensive usage among the MRS group members, make it a perfect choice for testing my implementation.

6.1 Testing scenarios

A set of 4 interception scenarios was used within the Gazebo environment to represent a variety of potential encounters between the interceptor UAV and the target UAV. These scenarios differ in factors such as the trajectory shape and the speed of its execution. During the test, the target follows a predefined complex trajectory in a 3D space. The trajectories are shown in Fig. 6.1.

Trajectory 1 in Fig. 6.1a is the longest out of all. It contains several parts corresponding to different maneuvers. First, the drone starts executing a relatively narrow zig-zag pattern. Then it flies forward, turns left, and executes 1.5 circles with small height changes. After that, it performs several "squeezed eight"-shaped maneuvers also with changes in height and finishes.

Trajectory 2 in Fig. 6.1b is a simple "eight" with a constant height.

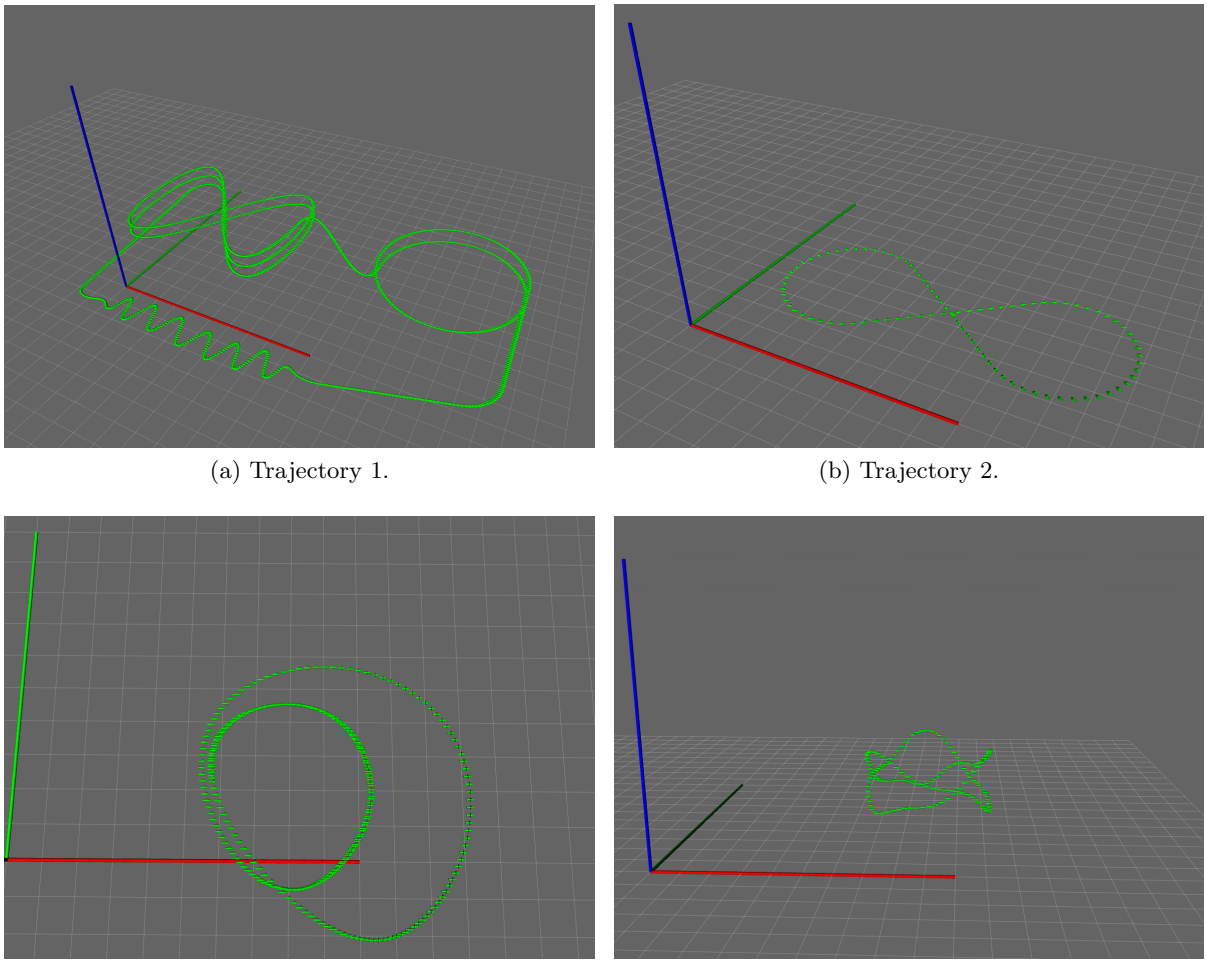
Trajectory 3 in Fig. 6.1c consists of two deformed circles of different size, which form a closed line. The drone flies through it twice.

Trajectory 4 in Fig. 6.1d when looking top-down resembles a deformed circle. In fact, it consists of two circles with large variance on the z axis, also forming a closed line. The drone flies through it twice.

6.2 Evaluation

Each testing trajectory was executed by the intruder UAV within the simulation environment. During the execution, as the interceptor attempted to catch it, data was recorded to a logging file and afterwards various metrics were computed using the logged data. These metrics are:

- Interception success: determining whether the interceptor UAV successfully intercepted the target UAV or the number of interceptions throughout the trajectory. The number of successful interceptions is denoted as n.
- Miss distance on interception: evaluating how closely the executed trajectory approached the target on interception. The mean miss distance is denoted as $\overline{D}$ and its standard deviation σ_D.



(a) Trajectory 1.

(b) Trajectory 2.

(c) Trajectory 3.

(d) Trajectory 4.

Figure 6.1: Visualization of trajectories used for testing in Gazebo. The grid size is 5 m.

- Distance to the target: evaluating how closely on average the interceptor followed the target during the flight. The mean distance throughout the experiment is denoted as $\overline{d}$ and its standard deviation σ_d.
- Computational performance: assessing the computational time required for a single execution of the algorithm. The mean computational time is denoted as $\bar{t}$ and its standard deviation σ_t.

The analysis is based on computing and comparing the mentioned metrics across the scenarios to identify strengths, weaknesses, and trade-offs of each approach. The following rules were chosen to compute the metrics:

- the target is intercepted if the distance to it in the xy-plane is less than $d_{interception}$,
- the interception attempt is counted if the mutual distance in the xy-plane is smaller than the interception distance for at least 5 timesteps (odometry rate during the tests is 125 Hz),
- only neighboring samples fully lying in the interception radius are counted as a single attempt.

The z-axis is ignored in these experiments since due to the configuration of the used MRS UAV system version, the vehicle can't be controlled by the acceleration command in the z-axis (needed for RL method and PN method). Instead, the altitude must be commanded explicitly in meters and during the experiments it was set to always be 2 meters above the target.

To test the robustness, the trajectory 1 is used, and the speed of the target is increased gradually by reducing the trajectory time step. The physical limit for the target, however, is performing the flight 2 times faster than shown in Fig. 6.2.

6.3 Baseline proportional navigation

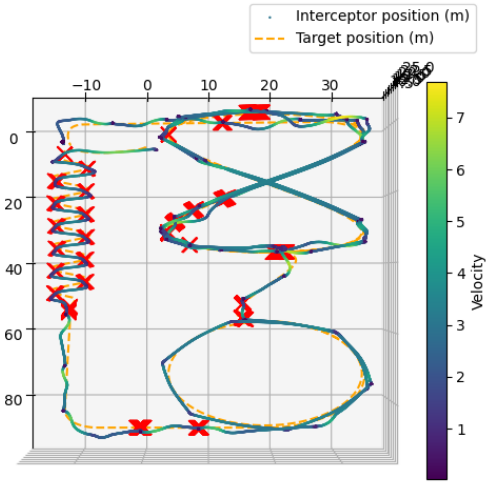
From the testing data, it is visible that PN is good for intercepting the target which performs "zig-zag" evasion maneuvers (successfully caught the target on every turn of the "zig-zag" in Fig. 6.2a).

On a straight part of the trajectory, PN showed a worse result, but still managed to catch, as visible in the lower part of the graph and on relatively straight parts of the "eight" in Fig. 6.2a.

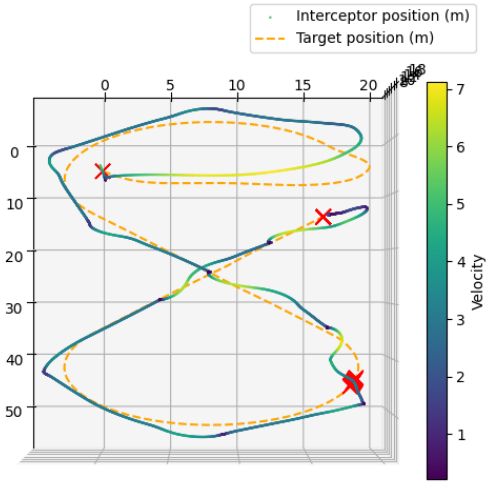
However, when the target was executing trajectories close to circular, PN was observed to lag slightly behind, often not getting close enough to score the interception as seen on the circular part of Fig. 6.2a, Fig. 6.2b, Fig. 6.2c. In the Fig. 6.2d the result is better because the target was slowing down a bit in the xy plane when going up/down, therefore giving the opportunity for the interceptor.

Overall, during the experiment, the PN method tracked the target with good precision staying close to it and only small deviations. It has successfully intercepted the target several times on every trajectory.

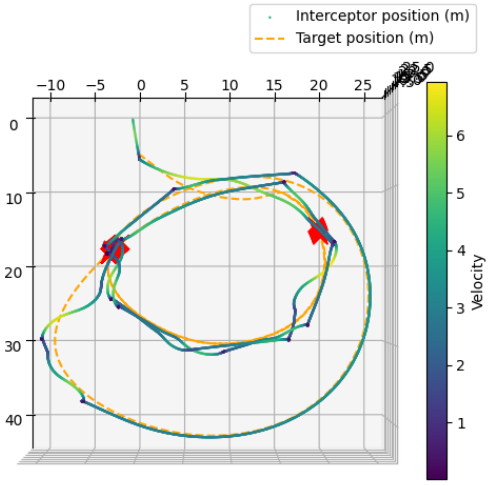
The computational speed of the algorithm was not measured since PN only involves evaluating several simple arithmetic operations and is therefore not computationally demanding.



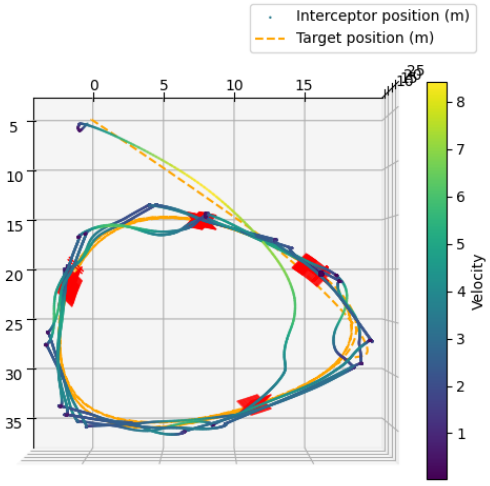
(a) Trajectory 1: $n = 27$, $\overline{D} = 0.29\,\text{m}$, $\sigma_D = 0.11\,\text{m}$.



(b) Trajectory 2: $n = 3$, $\overline{D} = 0.16\,\text{m}$, $\sigma_D = 0.12\,\text{m}$.

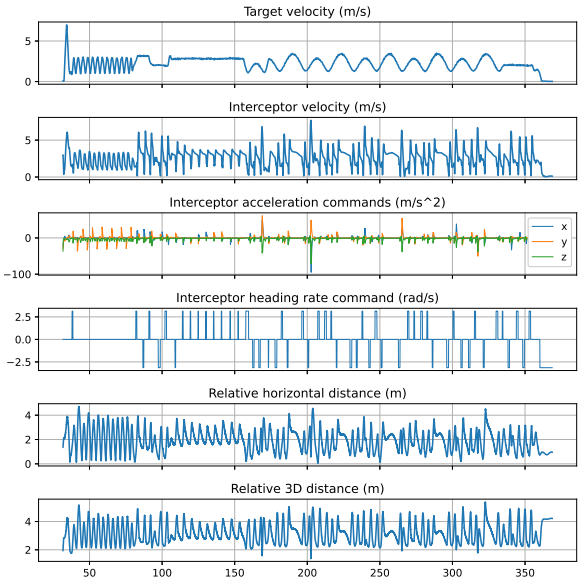


(c) Trajectory 3: $n = 4$, $\overline{D} = 0.24\,\text{m}$, $\sigma_D = 0.09\,\text{m}$.

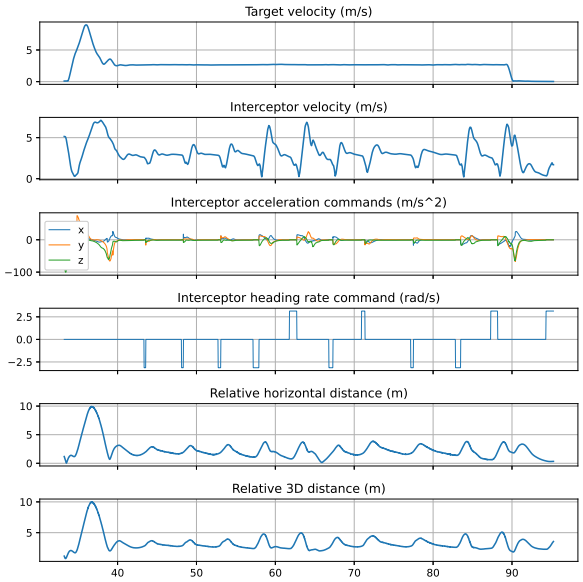


(d) Trajectory 4: $n = 7$, $\overline{D} = 0.22\,\text{m}$, $\sigma_D = 0.11\,\text{m}$.

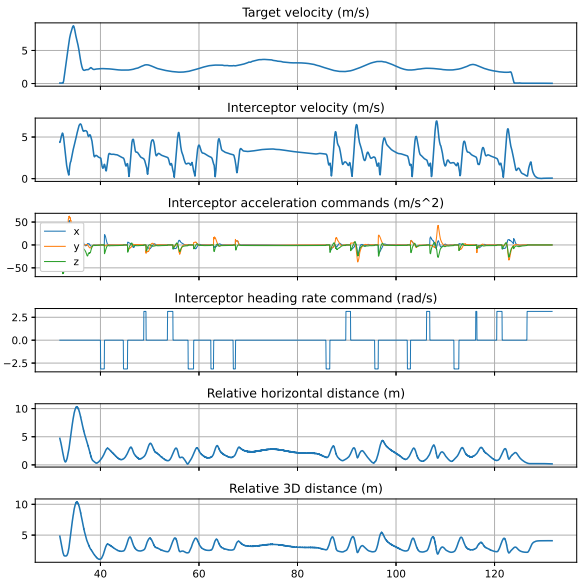
Figure 6.2: PN method, results of testing in Gazebo. Top view. Red crosses denote the interceptions.



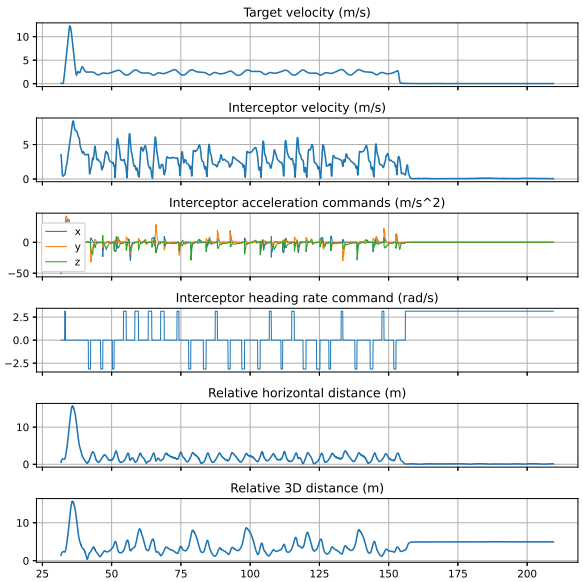
(a) Trajectory 1.



(b) Trajectory 2.



(c) Trajectory 3.



(d) Trajectory 4.

Figure 6.3: PN method, results of testing in Gazebo, additional metrics.

Limitations

When the target performed the flight two times faster than in the original experiment on Fig. 6.3a, the UAV scored 4 interceptions. Despite the target performing fast maneuvers, the PN method showed good robustness and tracking capabilities, being able to catch the intruder. The target speed is already maximal, its further increase is not possible.

6.4 MPC-based trajectory planner

As can be seen from the testing data (Fig. 6.4), the MPC-based planner showed outstanding results, tracking the target precisely on all trajectories during almost the whole flight. It has not suffered from any distinguishable problems on any specific part of the testing trajectories. On every graph, there is a repeating pattern of places where the interception is detected (red crosses) followed by a few meters with no interception. The reason is that when flying directly above, the interceptor loses the target from its simulated field of view (which is expected to happen) but then quickly recovers.

Limitations

To test the limitations, the target performed the flight two times faster than in the original experiment on Fig. 6.5a. In that scenario, the MPC still managed to score 23 interception attempts, making it a robust choice. The target speed is already maximal, its further increase is not possible.

Computational time comparison with the existing planner

Both the previous version of the MPC trajectory planner implemented using doMPC and my Acados implementation were tested under the same scenario and with the same problem formulation. An important point was to test the computational efficiency since both planners do the same task and use a similar model. Therefore, only computational demands are compared. To measure the computational times, a complex trajectory on Fig. 6.1a was selected. Data was collected from 2000 calls of each planner. Metrics are the mean time and the standard deviation presented in Table 6.1.

Testing was performed using the following hardware configuration: CPU: Intel core i7 8700, GPU: Nvidia GTX 1060, RAM: 32 GB.

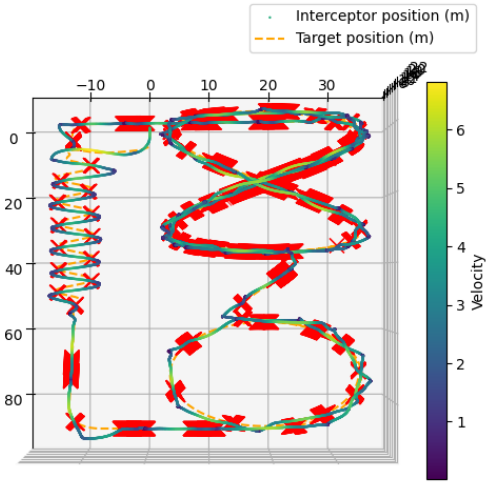
Settings of the interceptor implemented in doMPC, provided by the MRS group:

- Solver: IPOPT MUMPS,
- default settings as of doMPC version 4.6.4.

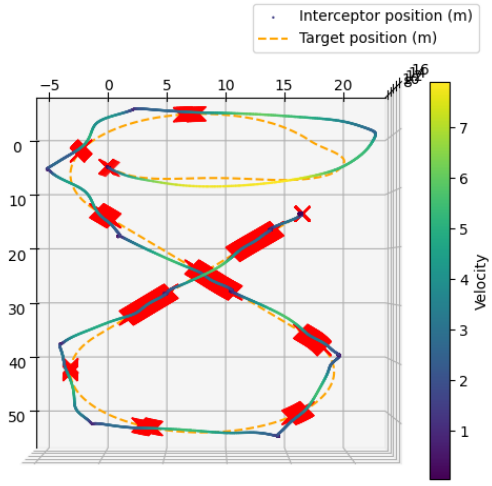
Settings of the interceptor implemented in Acados:

- nlp_solver_type: SQP,
- qp_solver: PARTIAL_CONDENSING_HPIPM,
- hessian_approx: GAUSS_NEWTON,
- integrator_type: IRK.

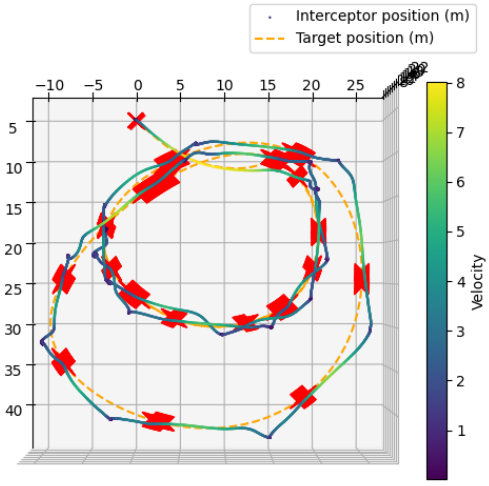
As can be observed from the results, the planner implemented in Acados offers a significant performance boost compared to the provided doMPC planner. Under the same scenario, the average computational time and its standard deviation are lower when using the planner implemented in this thesis.



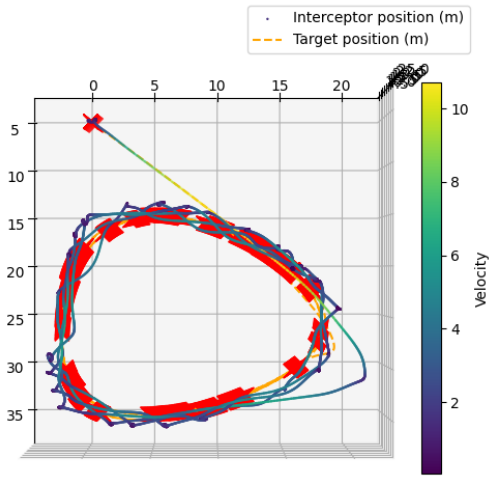
(a) Trajectory 1: $n = 71$, $\overline{D} = 0.15\,\text{m}$, $\sigma_D = 0.13\,\text{m}$.



(b) Trajectory 2: $n = 12$, $\overline{D} = 0.16\,\text{m}$, $\sigma_D = 0.11\,\text{m}$.

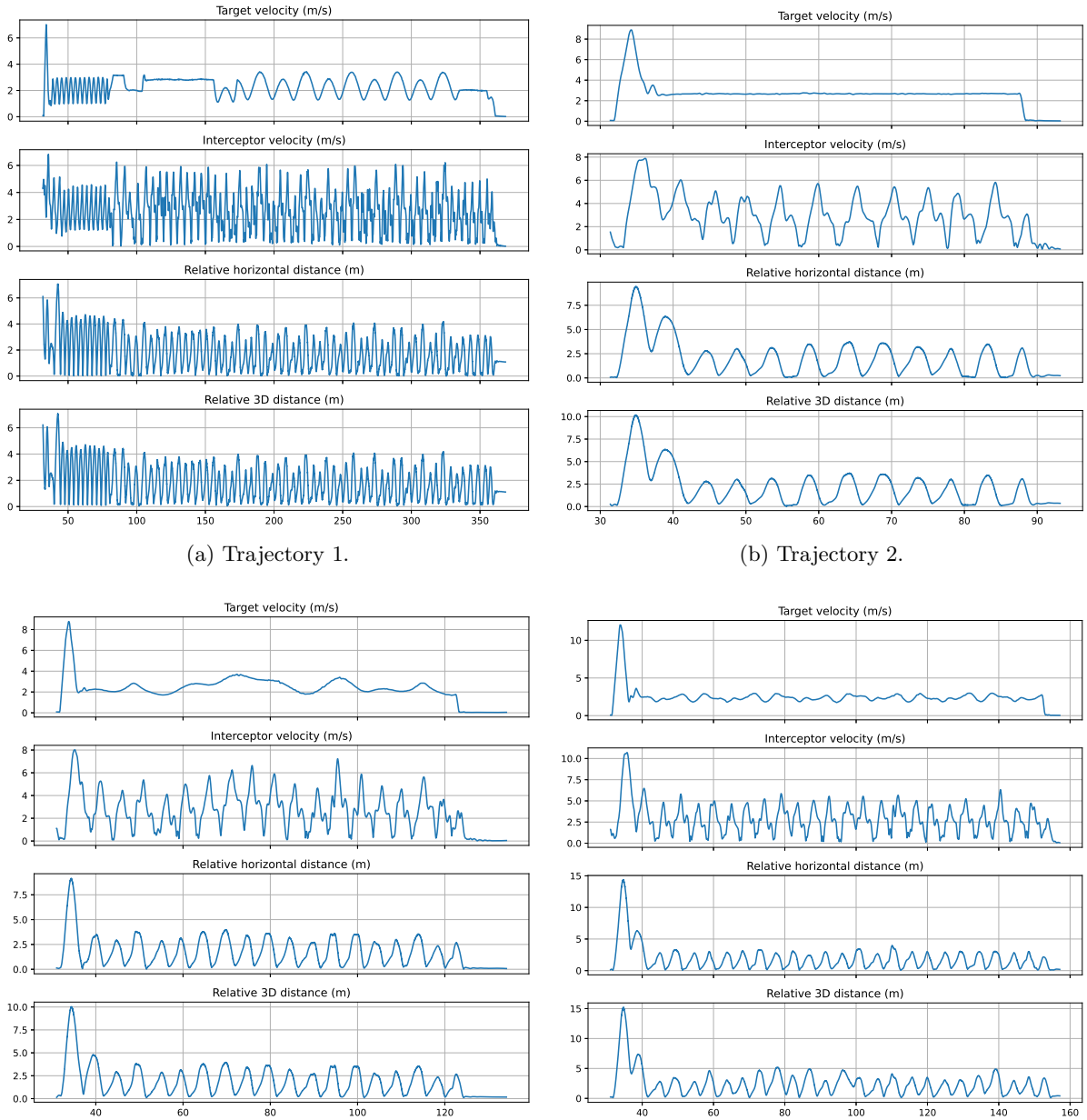


(c) Trajectory 3: $n = 20$, $\overline{D} = 0.19\,\text{m}$, $\sigma_D = 0.10\,\text{m}$.



(d) Trajectory 4: $n = 26$, $\overline{D} = 0.19\,\text{m}$, $\sigma_D = 0.08\,\text{m}$.

Figure 6.4: MPC-based method, results of testing in Gazebo. Top view. Red crosses denote the interceptions.



(a) Trajectory 1.

(b) Trajectory 2.

(c) Trajectory 3.

(d) Trajectory 4.

Figure 6.5: MPC-based method, results of testing in Gazebo, additional metrics.

	Number of calls	Mean computational time	Standard deviation
doMPC	2000	0.0580 s	0.0182 s
Acados	2000	0.0136 s	0.0034 s

Table 6.1: Comparison of the computational time between the doMPC-based and the Acados-based planners.

6.5 RL-based control strategy

The algorithm based on RL has also shown better results than the baseline method (Fig. 6.6). In the graphs, it is visible that the interceptor trajectory differs more from the target trajectory. Unlike when using previous methods, the UAV was attempting to not only track the target but perform more aggressive side-to-side maneuvers to catch it. It helps to do the interception while also maintaining the target in the FOV. A worse performance was observed when the target was executing a zig-zag maneuver on Fig. 6.6a, but it can be solved by training the model more on such trajectories.

The computational speed is fast compared to the MPC because to compute the action, only a forward pass through the network is performed. Measured values are $\bar{t} = 0.0049\,\text{s}$, $\sigma_t = 0.0031\,\text{s}$ calculated over 2000 calls.

Limitations

A known issue found during the experiments was a situation when the target is too far away from the interceptor. In such cases, the control command is usually saturated for long periods of time in all axes, possibly leading to flyaways or crashes. A suggested solution is using another guidance algorithm when the target is farther than a predefined distance and then switching to RL for closer engagement.

The RL-based algorithm performed poorly in terms of the average distance metric, being often far from the target. A possible solution is to create more training scenarios with a faster target for the RL model.

When the target performed the flight two times faster than in the original experiment on Fig. 6.7a, it led to scoring only 5 interceptions. The behavior mentioned in the previous paragraph was also observed when the target occasionally became too far. However, the interceptor still managed to recover and not fly away. The target speed is already maximal, its further increase is not possible.

6.6 Analysis and comparison

While all methods have successfully caught the target on the presented trajectories, there is a significant performance difference between them, which was observed during the testing.

The baseline PN method has shown the worst performance, scoring the least interceptions. It tracked the target closely but lagged slightly behind for most of the flight.

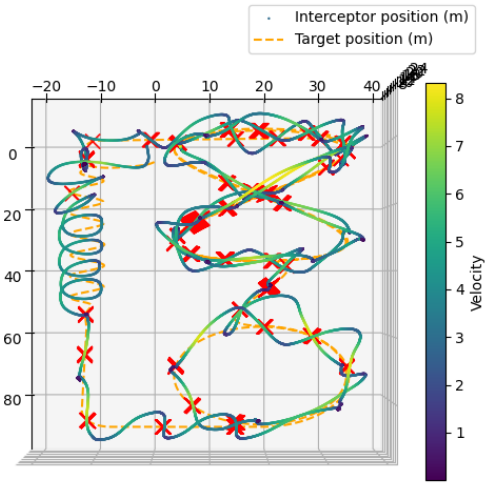
The MPC-based planner, on the other hand, was able to precisely track and intercept the target, except for moments when it was lost from the sensor's FOV. According to the previously selected metrics, it scored the most interceptions out of all algorithms. In real-world

deployments, however, the performance can be worse, given the uncertainties and noises of the target's detection.

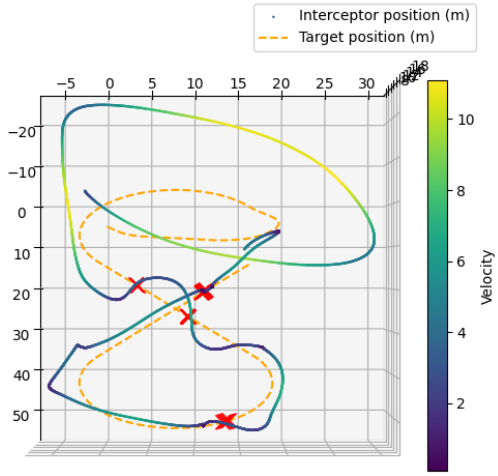
The RL-based control strategy had a different behavior than previous algorithms. Instead of attempting to track the target, it performed more active maneuvers which can be better for the purpose of aerial interception. RL does not require real-time optimization like MPC which makes it less computationally demanding. The control action is calculated by performing a forward pass through the network, which is relatively fast. Also, a highly complex non-linear reward function can be defined for training purposes, containing rewards for many aspects. The interception rate is much higher than using PN but lower than using the MPC. Although according to the metrics, it performed worse than the MPC (see Table 6.2), it is arguable whether the same holds in practice. The complex behavior of the RL-controlled UAV can be more desirable. A drawback, however, is that this method showed worse stability with a faster-flying target, which may be solved by including such scenarios in the training data.

	Algorithm	n	$\overline{D}\,(m)$	$\sigma_D\,(m)$	$\overline{d}\,(m)$	$\sigma_d\,(m)$
Trajectory 1	PN	27	0.29	0.11	1.90	0.90
	MPC	**71**	**0.15**	0.13	**1.82**	1.30
	RL	42	0.18	0.13	3.23	1.82
Trajectory 2	PN	3	0.16	0.12	2.29	1.53
	MPC	**12**	0.16	0.11	**2.02**	1.90
	RL	4	**0.15**	0.14	5.96	5.39
Trajectory 3	PN	4	0.24	0.09	2.03	1.42
	MPC	**20**	**0.19**	0.10	**1.71**	1.49
	RL	11	0.20	0.14	3.60	2.25
Trajectory 4	PN	7	0.22	0.11	**1.47**	1.94
	MPC	**26**	**0.19**	0.08	1.87	1.94
	RL	12	0.28	0.12	3.72	2.69
Trajectory 1, 2x speed	PN	3	0.29	0.13	**4.09**	2.35
	MPC	**24**	**0.19**	0.14	5.20	3.16
	RL	4	0.29	0.14	20.20	14.45

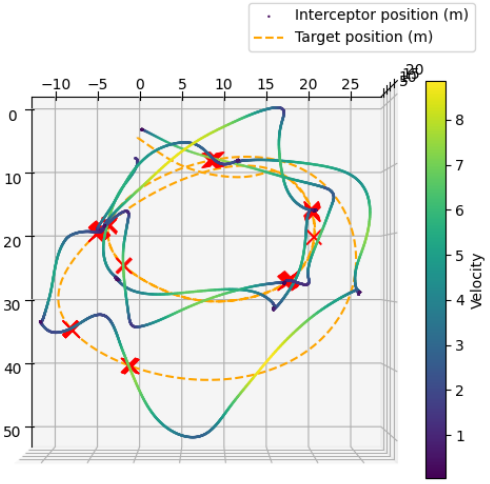
Table 6.2: Results of the Gazebo experiments. The best result for each trajectory is highlighted in bold.



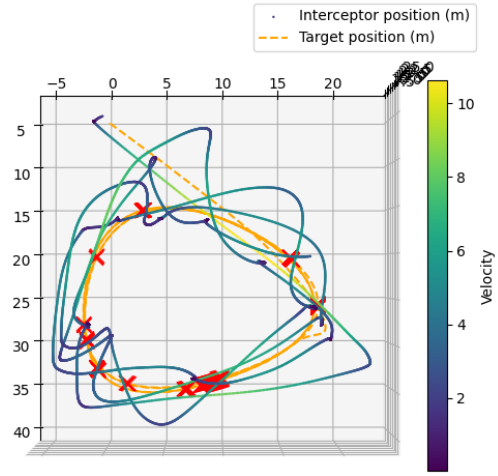
(a) Trajectory 1: $n = 42$, $\overline{D} = 0.18\,\text{m}$, $\sigma_D = 0.13\,\text{m}$.



(b) Trajectory 2: $n = 4$, $\overline{D} = 0.15\,\text{m}$, $\sigma_D = 0.14\,\text{m}$.

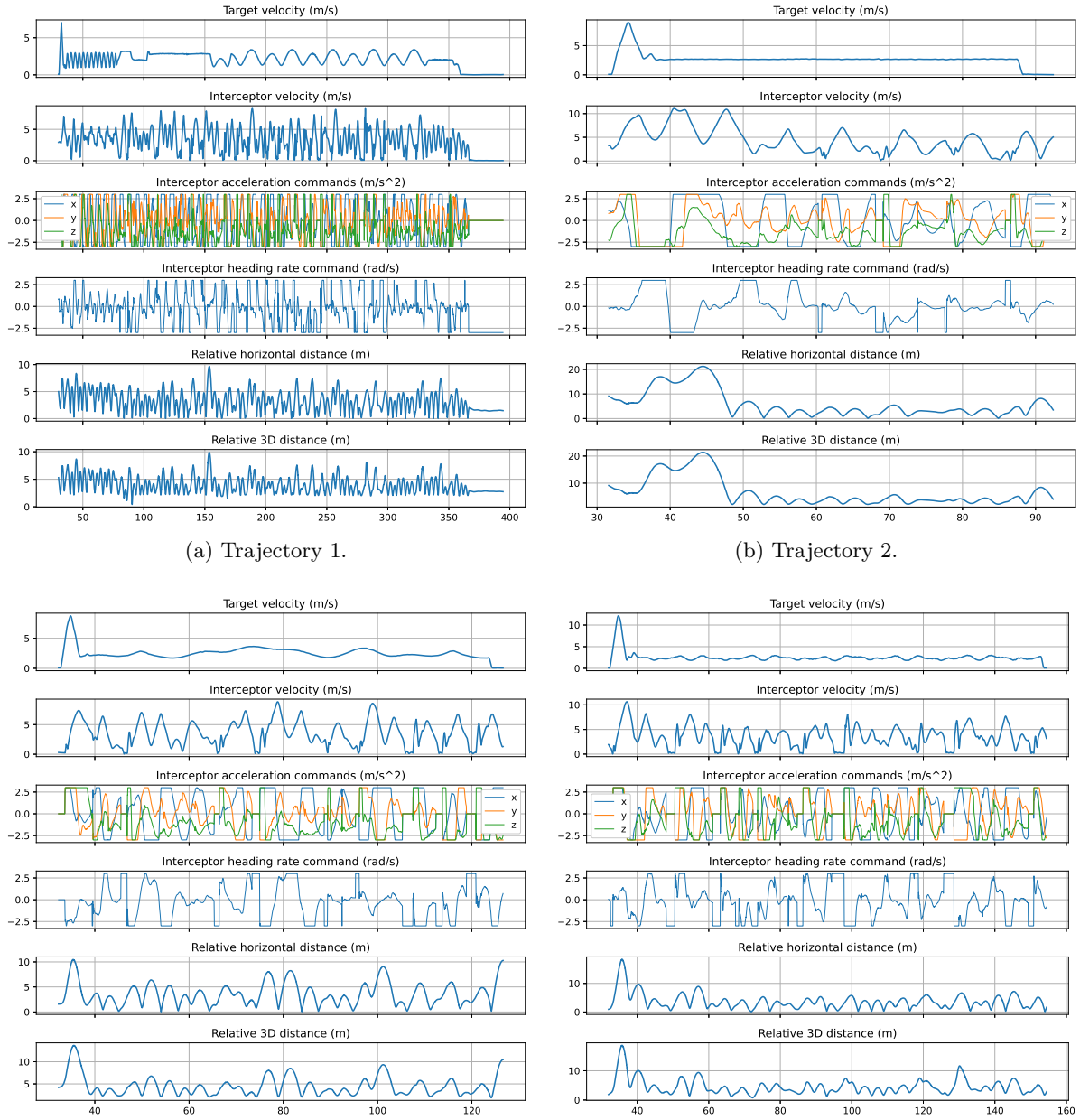


(c) Trajectory 3: $n = 11$, $\overline{D} = 0.20\,\text{m}$, $\sigma_D = 0.14\,\text{m}$.



(d) Trajectory 4: $n = 12$, $\overline{D} = 0.28\,\text{m}$, $\sigma_D = 0.12\,\text{m}$.

Figure 6.6: RL-based method, results of testing in Gazebo. Top view. Red crosses denote the interceptions.



(a) Trajectory 1.

(b) Trajectory 2.

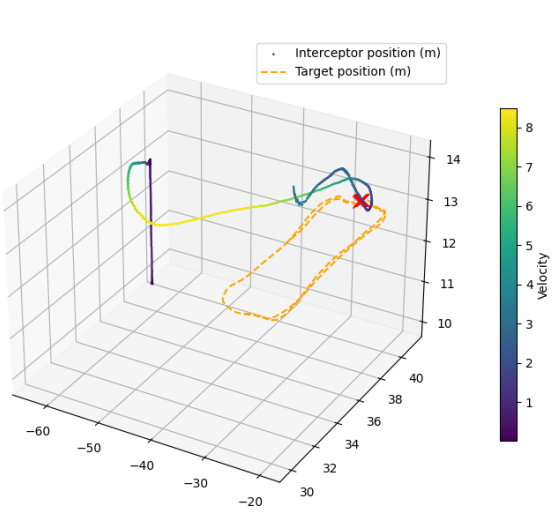
(c) Trajectory 3.

(d) Trajectory 4.

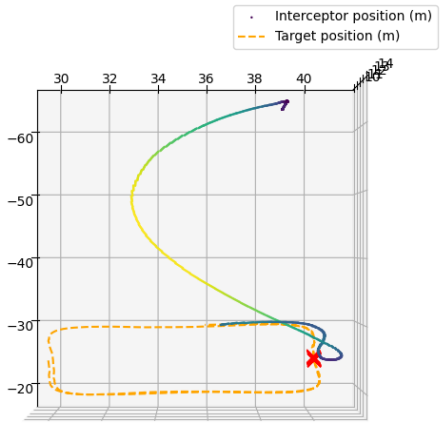
Figure 6.7: RL-based method, results of testing in Gazebo, additional metrics.

7 Real-world evaluation of the MPC-based trajectory planner

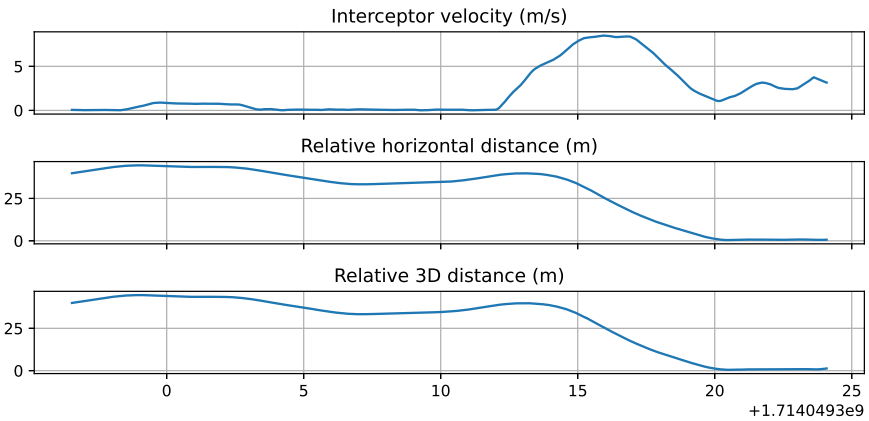
The performance of the MPC-based trajectory planner was tested in a real-world experiment. At the beginning of the experiment, the target UAV was executing a periodic rectangular trajectory, and the interceptor was approximately 40 meters far. Then, the planner was enabled and the interceptor started flying towards the target. The target was "caught" on the first attempt with a miss distance of 0.42 m. The plot finishes shortly afterward, when the manual control over the interceptor is enabled for safety reasons. This experiment shows that the algorithm performs well not only in the simulation, but also in a real-world scenario. However, further experiments are required to gather more data about the performance of the planner given various target speeds and maneuvers.



(a) 3D plot of trajectories of both UAVs.



(b) Top view of trajectories of both UAVs.



(c) Interceptor's velocity and distance to the target. Horizontal axis denotes the time in seconds.

Figure 7.1: Results of a real-world experiment using the MPC-based method.

8 Conclusion

In this thesis, a task of aerial interception of intruder drones was tackled. The aim was to design a system for navigating an interceptor UAV to catch a flying target. A Model Predictive Control (MPC)-based trajectory planner and a Reinforcement Learning (RL)-based control strategy were successfully implemented after performing a thorough research of the related literature and understanding the underlying concepts. Their effectiveness, speed and robustness were assessed, compared and tested in a simulation. A comparison with a baseline Proportional Navigation (PN) method was conducted. The MPC-based planner was also compared with the previous implementation of the MPC planner provided by the MRS group and tested in a real-world experiment, scoring a successful virtual interception.

The implemented algorithms demonstrated good interception performance, being more effective than the provided baseline PN-based method. Especially the MPC-based planner has shown outstanding interception effectiveness while also providing superior computational time to the previous version. To further improve the MPC planner, nonlinear constraints can be added to take collision avoidance into consideration directly in the produced trajectory. Another possible improvement is implementing penalization terms for the sensor Field of View so that configurations not preserving target visibility will have a worse value of the objective function.

The RL method on the other hand performed more complicated maneuvers than just tracking the target. It resulted in a greater mean distance from the target during the flight, but distinct interception attempts. The interceptor always scored more interceptions when using this method instead of the PN baseline method. However, according to the defined metrics, it has not shown as good interception rate as the MPC. Worse results were also observed with an increased target's speed. These issues can be tackled in future work by adding more training scenarios, containing the problematic configurations. Redesigning or tuning of the objective function may further improve the overall result. Another improvement can potentially be achieved by fine-tuning (training) directly in the Gazebo environment (currently it was trained only using a custom MRS simulator).

The implemented algorithms showed their viability as demonstrated during the experiments. Based on the simulation results, the algorithms work sufficiently well and are prepared for deployment. There is also room for further improvements of the two implemented methods presumably leading to better results as discussed in the thesis.

9 References

[1] M. Pliska, M. Vrba, T. Báča, and M. Saska, *Towards safe mid-air drone interception: Strategies for tracking & capture*, 2024. arXiv: 2405.13542 [cs.RO].

[2] M. Vrba, V. Walter, V. Pritzl, M. Pliska, T. Báča, V. Spurný, D. Heřt, and M. Saska, *On onboard lidar-based flying object detection*, 2023. arXiv: 2303.05404 [cs.RO].

[3] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of Yolo algorithm developments," *Procedia computer science*, vol. 199, pp. 1066–1073, 2022.

[4] Y. Song and D. Scaramuzza, "Policy search for model predictive control with application to agile drone flight," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2114–2130, 2022.

[5] M. Vrba, Y. Stasinchuk, T. Báča, V. Spurný, M. Petrlík, D. Heřt, D. Žaitlík, and M. Saska, "Autonomous capture of agile flying objects using UAVs: The MBZIRC 2020 challenge," *Robotics and Autonomous Systems*, vol. 149, p. 103 970, 2022, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2021.103970. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889021002396.

[6] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed, *et al.*, "Drone deep reinforcement learning: A review," *Electronics*, vol. 10, no. 9, p. 999, 2021.

[7] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, p. 26, 2021.

[8] D. Bick, "Towards delivering a coherent self-contained explanation of proximal policy optimization," Ph.D. dissertation, 2021.

[9] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.

[10] V. Semkin, M. Yin, Y. Hu, M. Mezzavilla, and S. Rangan, "Drone detection and classification based on radar cross section signatures," in *2020 International Symposium on Antennas and Propagation (ISAP)*, IEEE, 2021, pp. 223–224.

[11] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 1205–1212.

[12] L. Tan, X. Lv, X. Lian, and G. Wang, "Yolov4_drone: UAV image target detection based on an improved yolov4 algorithm," *Computers & Electrical Engineering*, vol. 93, p. 107 261, 2021.

[13] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "Acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021, ISSN: 1867-2957. DOI: 10.1007/s12532-021-00208-8. [Online]. Available: https://doi.org/10.1007/s12532-021-00208-8.

[14] V. Matić, V. Kosjer, A. Lebl, B. Pavić, and J. Radivojević, "Methods for drone detection and jamming," in *Proceedings of the 10th International Conference on Information Society and Technology (ICIST)*, 2020, pp. 16–21.

[15] M. Vrba and M. Saska, "Marker-less micro aerial vehicle detection and localization using convolutional neural networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2459–2466, 2020, ISSN: 2377-3766. DOI: 10.1109/LRA.2020.2972819.

[16] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, pp. 1–36, 2019.

[17] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, *Deep reinforcement learning that matters*, 2019. arXiv: 1709.06560 [cs.LG].

[18] G. Ling and N. Draghic, "Aerial drones for blood delivery," *Transfusion*, vol. 59, no. S2, pp. 1608–1611, 2019.

[19] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, no. 26, eaau4984, 2019. DOI: 10.1126/scirobotics.aau4984. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aau4984. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.aau4984.

[20] H. T. Obering, "Directed energy weapons are real... and disruptive," *Prism*, vol. 8, no. 3, pp. 36–47, 2019.

[21] M. Vrba, D. Heřt, and M. Saska, "Onboard marker-less detection and localization of non-cooperating drones for their safe interception by an autonomous aerial system," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3402–3409, 2019.

[22] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 6753–6760.

[23] L. Wu, F. Tian, T. Qin, J. Lai, and T.-Y. Liu, "A study of reinforcement learning for neural machine translation," *arXiv preprint arXiv:1808.08866*, 2018.

[24] R. Yanushevsky, *Modern missile guidance*. CRC Press, 2018.

[25] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[26] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[27] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[28] K. Nguyen, H. Daumé III, and J. Boyd-Graber, "Reinforcement learning for bandit neural machine translation with simulated human feedback," *arXiv preprint arXiv:1707.07402*, 2017.

[29] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *arXiv preprint arXiv:1705.04304*, 2017.

[30] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[32] A. R. Sharma and P. Kaushik, "Literature survey of statistical, deep and reinforcement learning in natural language processing," in *2017 International conference on computing, communication and automation (ICCCA)*, IEEE, 2017, pp. 350–354.

[33] C. Van Tilburg, "First report of using portable unmanned aircraft systems (drones) for search and rescue," *Wilderness & environmental medicine*, vol. 28, no. 2, pp. 116–118, 2017.

[34] H. Perritt Jr and E. Sprague, *Domesticating Drones: The technology, law, and economics of unmanned aircraft*. Routledge, 2016.

[35] M. L. Psiaki and T. E. Humphreys, "GNSS spoofing and detection," *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1258–1270, 2016.

[36] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[38] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[39] P. B. Quater, F. Grimaccia, S. Leva, M. Mussetta, and M. Aghaei, "Light unmanned aerial vehicles (UAVs) for cooperative inspection of PV plants," *IEEE Journal of Photovoltaics*, vol. 4, no. 4, pp. 1107–1113, 2014.

[40] J. Kober, J. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, Sep. 2013. DOI: 10.1177/0278364913495721.

[41] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[42] L. Grüne and J. Pannek, "Nonlinear model predictive control," in Apr. 2011, pp. 43–66, ISBN: 978-0-85729-500-2. DOI: 10.1007/978-0-85729-501-9_3.

[43] I. M. Bomze, V. F. Demyanov, R. Fletcher, T. Terlaky, I. Pólik, and T. Terlaky, "Interior point methods for nonlinear optimization," *Nonlinear Optimization: Lectures given at the CIME Summer School held in Cetraro, Italy, July 1-7, 2007*, pp. 215–276, 2010.

[44] F. Allgower, R. Findeisen, Z. K. Nagy, *et al.*, "Nonlinear model predictive control: From theory to application," *Journal-Chinese Institute Of Chemical Engineers*, vol. 35, no. 3, pp. 299–316, 2004.

[45] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," *Computers & chemical engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[46] S. J. Qin and T. A. Badgwell, "An overview of industrial model predictive control technology," in *AIche symposium series*, New York, NY: American Institute of Chemical Engineers, 1971-c2002., vol. 93, 1997, pp. 232–256.

[47] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[48] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[49] M. Guelman, "Proportional navigation with a maneuvering target," *IEEE Transactions on Aerospace and Electronic Systems*, no. 3, pp. 364–371, 1972.

[50] W. Dickinson, *History of Anti-aircraft Guns*, 1941. GP 0., 1920.