Universiteit Leiden

Master Computer Science

[Sim-to-Real Reinforcement Learning for Obstacle Avoidance and Goal Navigation in Micro Drones]

Name: [Yan Shen]
Student ID: [S3798801]

Date: [04/08/2025]

Specialisation: [Artificial Intelligence]

1st supervisor: [Mike Preuss]
2nd supervisor: [Aske Plaat]

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

With the growing interest in autonomous micro-UAVs, reinforcement learning has emerged as a popular approach. Traditional autonomous navigation systems for UAVs that avoid obstacles typically rely on GPS, external position systems, or cameras—devices with high hardware requirements. However, during emergency rescue missions such as earthquakes or fires, these systems may malfunction. To tackle these challenges while keeping costs low, this study introduces a reinforcement learning approach that relies solely on basic optical flow and ranging sensors. By carefully designing the observation space and reward mechanisms, and employing the Proximal Policy Optimization (PPO), a policy gradient method, we trained the UAV to avoid obstacles and navigate through doors in the custom-built simulation environment. The optimal policy obtained in the simulation environment is then deployed on the Crazyflie platform in physical-world validation scenarios. The evaluation demonstrates that the model achieves a 90% success rate in obstacle avoidance and door navigation within the simulation environment. In sim-to-real transfer experiments, even in realistic situations where the model has never been trained, it still achieves a 40% success rate in door navigation, validating the feasibility of zero-shot transfer and demonstrating the model's generalization capability and robustness.

Keywords: autonomous UAV, PPO, sim-to-real, zero-shot transfer, obstacle avoidance

Contents

1 Introduction

As technology continues to advance, the use of drones has become increasingly widespread in modern society. Whether they are large aircraft or palm-sized quadcopters, and whether used for aerial monitoring or autonomous navigation in rescue missions[1]. Due to effectively reduce labor costs, unmanned aerial systems have become essential tools in our smart society. Traditionally, drones have been manually controlled, yet as computing and AI have advanced, turning flight control over to learned algorithms has become a crucial stride toward full autonomy. Among the various machine learning approaches, deep reinforcement learning (DRL) shows great potential in robotic and industrial applications, which is emerging as a promising research direction[2].

In recent years, the application of small-scale drones in daily navigation and exploration tasks has become a research hotspot, leading to numerous successful results[3]. However, in extreme conditions such as natural disasters, earthquakes, or fires, where GPS signals or external systems (e.g., Vicon-based localization)[4] may be unavailable, there remain significant challenges for these small drones to complete fundamental indoor navigation and rescue tasks. Micro-drones like the Crazyflie[5] offer notable advantages such as flexibility and portability, but their limited size and payload capacity restrict their ability to carry advanced sensing or positioning modules. Moreover, relying on high-end sensors greatly increases the deployment cost. Therefore, it is of practical significance to explore how autonomous indoor navigation and obstacle avoidance can be achieved using only basic onboard hardware, particularly to bridge this capability shortfall in real-world scenarios.

The objective of this research is to investigate a system that enables a Crazyflie drone, which is equipped only with its basic hardware (Flow Deck[6] and Multi-Ranger Deck[7]), to perform indoor obstacle avoidance and door-passing tasks using a reinforcement learning-based approach, without relying on external systems. On one hand, the study evaluates the feasibility of deploying reinforcement learning algorithms on low-computation micro aerial vehicles; on the other, it provides a low-cost and scalable solution for micro-drones with limited payloads to perform patrol, exploration, or rescue missions in real-world environments.

To achieve these goals, this thesis proposes and implements a complete indoor navigation framework for micro drones. This work employs the Proximal Policy Optimization (PPO) algorithm [8], a method in reinforcement learning designed to restrict policy updates for stable training and reliable convergence, which is well suited for onboard deployment on the Crazyflie platform, and is trained in a simulation environment constructed by PyBullet[9]. The simulated environment consists of a 10×10 meter square room containing four rectangular obstacles and a 1.5-meter-wide door. A spherical agent is used to simulate the Crazyflie during training. The model receives sensory observations such as obstacle distances from four directions and local velocity data. The agent learns to avoid obstacles and identify the door by continuously exploring the environment. The best-performing policy during training is saved as the baseline model, which is later evaluated through both controlled experiments and sim-to-real[10][11] transfer experiments. As the final step, the trained policy is run offboard on a PC and sends control commands to the Crazyflie drone via Crazyradio 2.0[12]. The onboard sensors provide real-time data to the model, which then outputs control actions for autonomous flight. This allows us to assess the feasibility and generalization capability of the sim-to-real transfer.

This thesis is structured into seven chapters. Chapter 1 provides the introduction, motivation of the project, and primary contributions of the study. Chapter 2 reviews related work in reinforcement learning applications for UAVs. Chapter 3 outlines the methodology, covering

how both the simulation and real-world systems were designed and implemented. Chapter 4 describes the experiments and provides an analysis of the results obtained from both simulated and real-world evaluations. Chapter 5 offers a discussion of the observed challenges and failure cases. Chapter 6 summarizes the main findings and conclusions. Chapter 7 presents possible future directions for algorithmic improvements and experimental design.

2 Related Work

2.1 Vision-Based DRL for Autonomous Drone Flight

Indoor autonomous drone navigation increasingly leans on reinforcement learning techniques, especially for vision-based obstacle avoidance. In 2017, Sadeghi and Levine proposed a method called CAD2RL[13], which uses the Monte Carlo policy evaluation algorithm to train the model using RGB images rendered from a simulated environment constructed by randomizing CAD models. Finally, the model was tested on real UAVs without any real-world training, verifying the success of simulation-to-reality generalization. Anwar et al. introduced NAVREN-RL, a method for indoor drone navigation using end-to-end RL[14]. This technology uses the onboard camera to capture images and employs Double DQN to train the model to navigate autonomously while avoiding obstacles. Andreas Seel [15] proposed a reinforcement learning architecture that uses cameras and LiDAR sensors to achieve indoor navigation without relying on GPS. To evaluate the strategy, the group assembled a full-featured simulation suite with different arenas for training and validation. In the work by Xie et al. (2023) [16], a quadcopter equipped with an RGB-D camera was trained using the Soft Actor-Critic (SAC) algorithm [17], enabling it to autonomously perceive and navigate through narrow, unknown, and inclined environments. An RGB-D camera was employed to process visual data and infer the gap's location and orientation. The authors applied curriculum learning and domain randomization to gradually guide policy training and enhance the drone's robustness in real-world scenarios, which also inspired the sim-to-real component of our experiments. In their 2023 work, Kalidas et al. [18] explored the use of reinforcement learning to achieve visual navigation in drones, enabling them to avoid both static and dynamic obstacles. Their study considered discrete and continuous action spaces and compared three algorithms, DQN, SAC, and PPO, in a variety of training and testing scenarios in the AirSim virtual environment. The results indicate that the off-policy algorithms, SAC and DQN, outperform the on-policy PPO. However, the work is restricted to simulation and does not address sim-to-real deployment.

2.2 PPO and Alternative Algorithms for UAVs

McGuire et al. proposed the method called minimalist Swarm Gradient Bug Algorithm (SGBA)[19] that enables micro-robots with limited computing resources, such as Crazyflie, to navigate by sensing obstacles in real time and avoiding them. They proposed the concept of 'wall-following' to avoid static obstacles. The drones communicate with each other and return to the starting point based on the received signal strength indication (RSSI) at the home position. This paper demonstrates that Crazyflie can rely on real-time sensors for obstacle avoidance. This finding provides theoretical support for the possibility of our experiment relying on Multi-range deck sensor to measure obstacle distances for obstacle detection. In addition to training with the standard PPO algorithm, Molchanov et al.[20] applied domain randomization (DR) technology, i.e., randomizing parameters during training to enhance policy robustness and adaptabil-

ity, facilitating a smoother transfer from simulation to real-world deployment. In our research, we similarly enhance policy generalization by randomizing the agent's take-off position. A PPO-based approach integrated with the Asymmetric Actor-Critic method was introduced by Kangtong et al. [21], to train multiple drones to transport packages of unknown weight indoors. The paper uses a single camera for navigation and target localisation. They validated the algorithm's effectiveness through experiments conducted in the Gazebo simulation environment. Results indicated that the proposed approach outperformed alternative methods in simulation; however, it was not validated through real-world experimentation. Of course, beyond PPO, lightweight CNN-based deep-learning architectures have also achieved promising results in enabling autonomous obstacle avoidance for UAVs.[22]. This paper proposes a CNN architecture called NanoFlowNet, which primarily improves the success rate of obstacle avoidance using monocular vision through real-time optical flow calculation, thereby confirming the effectiveness of optical flow information for obstacle avoidance. Ali et al. established a two-dimensional to three-dimensional simulation environment, where agents applied different reinforcement learning algorithms to learn obstacle avoidance navigation policy, thereby evaluating and comparing the performance of each algorithm[23]. They reported that DDPG achieved superior results in obstacle avoidance, whereas PPO was prone to being trapped in local optima and demanded a substantial amount of samples to reach convergence.

2.3 Research Advances in Sim-to-Real Policy Transfer

An on-policy reinforcement learning strategy was adopted by Hwangbo et al. [24] to enable precise trajectory tracking in drone control tasks. After simulation training, it was transplanted to the Hummingbird quadcopter for flight testing. The experimental results show that the neural network policy trained by reinforcement learning can effectively enable autonomous flight of drones. The absence of real-world disturbances during training significantly constrains the transfer of policies from simulation to reality. Numerous studies have explored deploying reinforcement learning strategies on UAVs without further fine-tuning in physical settings. Kang et al. studied a deep reinforcement learning algorithm with mixed training [25], which not only utilises visual features learned from images in a simulated environment but also incorporates real-world data into the training as a control subsystem. Experiments demonstrated that even in real-world, when encountering complex environments never trained before, the drone was able to fly successfully, and the collision-free flight distance was increased by four times. Consequently, Loquercio et al. (2021) [26] incorporated realistic sensor noise, such as missing depth cues, into the model training to achieve zero-shot transfer, thereby enabling the policy to handle noise that exists in reality and enhancing its robustness. Their experimental results demonstrated that even during high-speed flights performed without any fine-tuning, the sim-to-real transition cut the failure rate by a factor of ten. A benchmark evaluation of advanced learning-based drone control strategies was carried out by Kaufmann et al. [27]. They proposed a Collective Thrust and Bodyrates (CTBR) control strategy, which was trained using the PPO algorithm. The research results demonstrated that the CTBR strategy exhibits excellent transferability, responsive performance, and suitability for high-speed flight in real-world scenarios. Additionally, Kaufmann et al. (2023) proposed the Swift system, which defeated human pilots in a drone racing competition [28]. This system is also trained using the PPO algorithm. For the sim-to-real component, the system injects non-parametric real-world noise into training data, uses Gaussian processes to simulate drift in real-world flight, and employs k-nearest-neighbour regression to refine simulated dynamics. These techniques

enabled Swift to achieve a 60% win rate and set the fastest race time record. Joshi et al.[29] constructed a random simulation environment using PyBullet, added different types of noise for comparison, trained the agent to avoid obstacles using the PPO algorithm, and evaluated the performance of policies under different noise conditions. In real-world tests, motion capture markers were added to precisely locate the drone and control perturbations in the observed data. The experiments demonstrated that adding a small amount of noise during training can improve training performance, providing a reference for our own use of Gaussian noise in training. Chen et al. (2015) introduced a reinforcement learning system named SimpleFlight [30]. The method leverages the PPO algorithm as its core, incorporating rotation matrices and time vectors as inputs in the Actor and Critic networks, encouraging smoother actions, and increasing batch sizes during training to enhance the generalization capabilities of the policy. The system uses the OptiTrack motion capture system on Crazyflie to capture position and velocity data, enabling it to track arbitrary trajectories. This demonstrated that the algorithm reduces trajectory error in the sim-to-real transfer phase by over 50% compared to baseline reinforcement learning methods.

2.4 Study on Crazyflie

Due to its lightweight and portability, Crazyflie is an ideal platform for drone research. Dunkley et al.(2014) used Crazyflie equipped with a micro PAL camera to achieve autonomous hovering and flight through real-time positioning and SLAM systems [31]. This further demonstrated that Crazyflie is very suitable for autonomous flight research as a miniature drone. Preiss et al.(2017) developed Crazyswarm, a system architecture consisting of 49 micro quadcopter(Crazyflie), enabling them to perform coordinated formation flight indoors and interact with humans as dynamic obstacles [32]. This research demonstrates that Crazyflie is highly suitable for academic research, performing exceptionally well in both obstacle avoidance and cooperative flight. Duisterhof et al. [33] proposed using Deep Reinforcement Learning and tinyRL to train the Crazyflie micro-robot to perform autonomous indoor flight and navigate around obstacles to find information sources. They employed the DQN algorithm. Due to the indoor light source and the custom light sensor equipped on the Crazyflie, the observation space included not only basic ranging data but also two light intensity readings.

2.5 Research on UAV Flight Path Planning

Trajectory optimization for autonomous UAV flight has also been a popular area of research. Since trajectory optimization often relies on Vicon motion capture systems, our experiments did not involve trajectory optimization. However, t these studies still provide valuable insights to help us understand potential flight control challenges during autonomous navigation. Mellinger et al. demonstrated that quadrotor dynamic systems have differential flatness, which allows for the generation of minimum snap trajectories.[4]. Their experiments have proven that drones can complete pre-set trajectories even through narrow gaps or with steep pitch angles. Liu et al.[34] also proposed a novel drone route planning framework. It first constructs a basic path in a simulation environment using the Visibility Path Searching algorithm, and then optimizes the motion trajectory based on the previously generated path using the PPO algorithm. However, this study focuses more on the simulation aspect, and there is no further discussion or verification of the subsequent real-world deployment. In reinforcement learning, beyond enabling a single UAV to accomplish tasks, Coordination and cooperation between multiple

drones have also become an important research direction. Jungwon Park et al.[35] primarily discuss a trajectory planning algorithm suitable for multi-agent obstacle avoidance. The paper first uses a grid-based method to calculate the initial trajectory, then combines aerodynamics to optimize the trajectory. They formulate trajectory generation as a convex optimization task, which helps to avoid possible impacts with surrounding objects. The quality and safety of the trajectories were verified through simulations and real-world flight experiments. Similarly, regarding multi-agent trajectory planning, Yunwoo Lee et al.[36] also introduced a DMVC-Tracker online distributed trajectory planning algorithm to generate trajectories suitable for multiple agents to avoid obstacles while maintaining appropriate safety distances. This method was validated in static and dynamic obstacle environments using Crazyflie quadcopter drones in both simulations and hardware experiments.

3 Methodology

3.1 Reinforcement Learning Background

As highlighted by Sutton and Barto [37], reinforcement learning has become a widely adopted framework within the field of artificial intelligence. Agents learn to adapt their behavior by engaging with the environment, relying on well-designed observations and feedback signals to refine their decisions. Over time, agents learn to choose optimal actions that maximize cumulative rewards[38]. Similarly to how students gradually master problem-solving skills through feedback from teachers, agents rely on reward and punishment signals from the environment to continually adjust their behavior, ultimately learning to make effective decisions in specific scenarios. A typical criterion for categorizing reinforcement learning approaches is the use of an environment model. In model-based methods, the dynamics are either predefined or learned, whereas model-free methods function without this information. That is, given a state, the agent can take an action and the model can predict the environment's response, including the next state and the reward received. Dynamic Programming (DP) is a typical model-based method[39], which derives the optimal policy by simulating state transitions and reward processes. Rather than utilizing an internal model of environmental dynamics, model-free approaches learn solely through interaction. In model-free learning, the agent derives behavioral strategies solely from experience, without any reliance on a predefined model of the environment. In highly dynamic or poorly modeled environments, methods that learn directly from experience tend to demonstrate better adaptability. Nevertheless, these methods often demand extensive experience collected through interactions before achieving stable learning performance.

Policy-based reinforcement learning is a type of model-free method. In contrast to value-based strategies—which determine the expected return of states or actions—policy-based approaches learn a function $\pi(a|s)$ that assigns probabilities to actions based on the current state. These algorithms aim to directly optimize the policy to maximize long-term rewards. A key strength of such methods lies in their effectiveness in managing continuous action spaces, which makes them particularly suitable for complex scenarios requiring smooth action selection.

Apart from this classification, reinforcement learning can also distinguish between two types of backup strategies: off-policy and on-policy[40]. In the off-policy method, the agent doesn't necessarily learn the policy that it's currently following. Instead, it is able to acquire knowledge from various policies and even from prior experiences. For example, using the Experience Replay method, the agent stores previous experiences and then updates its policy by randomly

sampling from these stored experiences. This helps the algorithm avoid getting stuck in local optima. In contrast, algorithms that follow an on-policy strategy often refine their behavior using data generated from their current decision-making process. This approach helps ensure consistency and reduces variability across training, leading to a more stable training process. Our work employs PPO—an on-policy method that updates using only data gathered by its latest policy, ensuring steady improvement while maintaining exploration.

3.2 Overview of PPO Algorithm

3.2.1 Structure of the Actor-Critic Method

The actor-critic architecture integrates policy learning $(actor)$ with value function approximation $(critic)$, leveraging the benefits of both approaches [41]. By integrating these two components, it achieves a trade-off between bias and variance, thereby promoting consistency and robustness during training. Thus, the Actor learns a policy aimed at directing the agent to maximize the cumulative rewards obtained through its interactions with the environment. Updates network parameters via policy gradient methods, based on the value assessments provided by the Critic, thus increasing the probability of taking actions that maximize value. Meanwhile, the critic estimates how effective the current policy is by approximating its expected returns. For example, it may employ the advantage function $A_n(s_t, a_t)$, commonly expressed in Equation 1. The advantage function measures the relative quality of an action against a reference baseline, thereby promoting choices with higher expected returns.

$$A_n(s_t, a_t) = Q_n(s_t, a_t) - V(s_t) \tag{1}$$

3.2.2 PPO Algorithm

Proximal Policy Optimization (PPO), developed by Schulman and collaborators in 2017[8], is a reinforcement learning technique that operates without relying on an explicit environment model and utilizes the most recent policy during training. As a policy gradient approach, PPO focuses on maintaining both effective learning progress and robust training outcomes. PPO also adopts the basic structure of the Actor-Critic framework, but it introduces a Clipping mechanism on this basis. By limiting how much the policy can change in a single update, it prevents the risk of adjustments being excessively large or too minor, thereby enhancing the stability of training. As mentioned earlier, PPO also employs the advantage function, which makes the policy update process more efficient. Moreover, PPO introduces the clipped probability ratio ϵ, which can employ a clipping strategy that compares action probabilities under new and old policies [8]:

$$r_t = \frac{\pi_{\mathsf{new}}(a_t \mid s_t)}{\pi_{\mathsf{old}}(a_t \mid s_t)} \tag{2}$$

This formulation helps further control the range of policy updates. PPO seeks to improve the policy through the optimization of a clipped surrogate objective function [8]:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t,\ \mathsf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t \right) \right] \tag{3}$$

In PPO, Generalized Advantage Estimation (GAE)[42] is often adopted to compute the advantage values. This is because directly and precisely calculating $Q(s_t, a_t)$ and $V(s_t)$ can

be quite challenging in reality. By integrating temporal-difference(TD) and Monte Carlo techniques, GAE approximates the advantage function in a way that enhances stability and learning efficiency. The estimation is defined as follows [42]:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{4}$$

where: - γ denotes the discount rate - The parameter λ adjusts the balance between bootstrapped TD learning and Monte Carlo returns.

As a result, using this approach can reduce the issues of high variance and high bias, and allows for balancing the mixture of the two by adjusting the λ parameter.

As shown in Equation 3, the most important improvement over the traditional actor-critic framework is the introduction of the clipping function in PPO. The clipped function[8] is defined as follows:

$$\text{clip}\left(r_t(\theta),\ 1-\epsilon,\ 1+\epsilon\right) \tag{5}$$

To maintain stable policy updates, PPO restricts the probability ratio $r_t(\theta)$ within the interval $[1-\epsilon, 1+\epsilon]$ by applying clipping when it deviates beyond this range. Through this clipping mechanism, the algorithm maintains relatively stable and improves the convergence of the training process. Moreover, keeping an appropriate ratio of strategy exploration to exploitation is essential for preventing local optima and policy collapse.

The explained pseudocode is shown as 1 outlines a general implementation of the PPO method[8]. This algorithm has found broad application in reinforcement learning, especially within robotic domains. The algorithm demonstrates excellent training stability and is capable of converging within a relatively short period of time. As a result, it is especially well-suited for autonomous learning and accomplishing complex tasks in continuous or high-dimensional action spaces. In this project, we utilize PPO as the primary training algorithm (see Section 4 for training details) to develop a navigation policy for a quadrotor drone within a simulated indoor environment, to eventually deploy the learned behaviors in real-world scenarios.

3.3 Implementation Details

To achieve the goal of enabling the Crazyflie drone to autonomously avoid obstacles and successfully locate the exit, we have divided the overall design and development process into two phases: The first phase involves the construction of a simulated environment and the training of navigation policies, while the subsequent stage is deploying the trained model to effectively guide the drone's flight behavior in real-world scenarios. The development structure is shown as Figure 1 below.

In the entire simulation environment, the agent has no knowledge of the locations of obstacles or the door. It relies solely on two basic onboard sensors and learns to avoid obstacles and locate and pass through the door by interacting with the environment through a designed reward function. Due to the limited information available and the inherent difficulty of the task, it was not possible to design an effective reward function from the outset. Therefore, we adopted a curriculum learning approach[43]. Initially, we focused on constructing the simulation environment and simulating the sensor inputs, allowing the agent to navigate freely in four directions based on its observation and action spaces.

After integrating the PPO algorithm, we observed that the initial reward design—comprising only a reward for passing through the door and a penalty for collisions—was insufficient. The

Algorithm 1 The Proximal Policy Optimization (PPO)[8]

Initialize policy network π_θ, value function V_ϕ
Set clipping threshold ϵ, discount factor γ, GAE parameter λ
while not converged **do**
    Collect trajectories $\{(s_t, a_t, r_t, s_{t+1})\}$ using current policy π_θ
    for each timestep t **do**
        $\delta_t \leftarrow r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
        $A_t \leftarrow \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$
        $r_t(\theta) \leftarrow \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
    end for
    for epoch $= 1$ to N **do**
        Compute clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t \right) \right]$$

        Update policy parameters θ via gradient ascent on $L^{\text{CLIP}}(\theta)$
        Compute value function loss:

$$L^V(\phi) = \mathbb{E}_t \left[(V_\phi(s_t) - R_t)^2 \right]$$

        Update value function parameters ϕ via gradient descent
    end for
end while

agent would often crash into obstacles shortly after takeoff. To simplify the problem, we fixed the drone's hover height and removed the yaw angle from the action space, restricting movement to the horizontal plane. We also set a fixed starting position in each episode.

At this stage, we introduced a basic distance-based reward scheme that encouraged movement towards the door, penalized deviation from the door center, and discouraged lingering near the doorway. Over time, the agent gradually learned to pass through the door, although the success rate remained low. To further guide the agent, we added penalties for approaching obstacles and rewards for forward progress. Initially, we designed an additional reward when the agent's distance to the door was closer than its previously best distance, but this proved ineffective. We then modified the scheme to reward the agent whenever its current distance to the door was shorter than in the previous step, and penalize it otherwise.

Throughout the experiments, we continued to fine-tune parameters and reward mechanisms—for instance, by removing the penalty for lateral deviation and only rewarding proximity to the door center, thereby encouraging exploration. Once the agent reliably achieved 100% success in obstacle avoidance and door traversal from the fixed starting point, we extended the training to include random initial positions using the same reward strategy and made further adjustments accordingly.

During the real-world trials, the control loop was initially capped at 5 steps, and we printed the Crazyflie's proximity measurements in all four horizontal directions, along with the model's predicted actions, the velocity commands sent to the Crazyflie, and the inferred movement direction. Through this process, we discovered an inconsistency between the coordinate systems of the two systems. We then modified the control script accordingly to resolve this issue. After confirming correctness, we gradually extended the control loop to 500 steps for full

deployment. There are some screenshots showing the model inference and Crazyflie flight information printed in the terminal during our real-world experiments. These screenshots are included in the Appendix 272829.

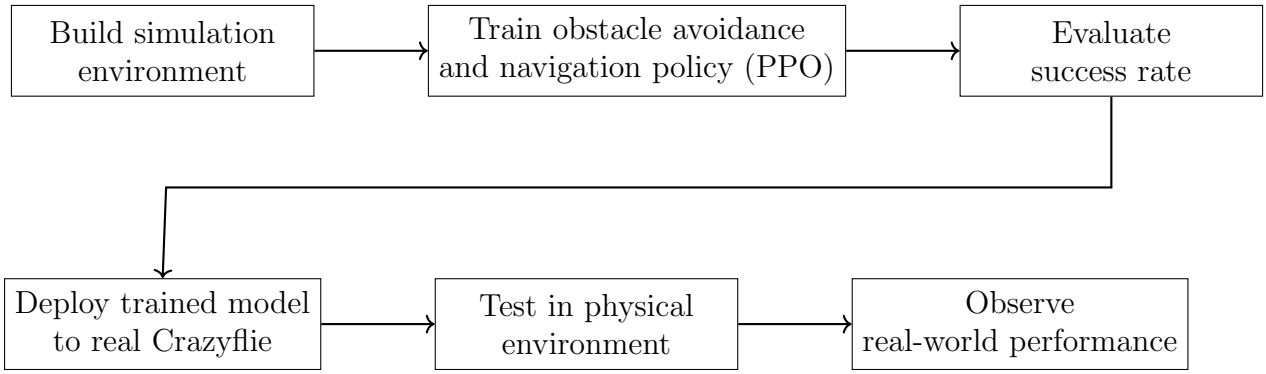


Figure 1: Sim-to-real process for obstacle avoidance task with Crazyflie

3.3.1 Configuration of the Simulated Environment

Considering that reinforcement learning models typically require a large number of training iterations, especially since real-world quadrotors are prone to collisions and hardware damage. Therefore, it is essential to first build a simulation environment before conducting any real-world flight tests. While several simulation platforms are available, and Bitcraze officially recommends using Webots for Crazyflie simulations, given that PyBullet provides native compatibility with popular RL libraries such as Stable-Baselines3[44], making it easier to define observation/action spaces, design reward functions, and integrate with algorithms like PPO. This flexibility and ease of integration make PyBullet a more suitable choice for our learning-based navigation task.

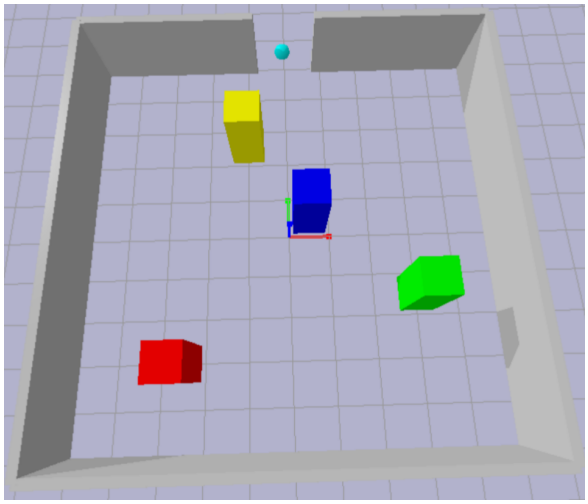
The agent was trained using a PyBullet-based simulation, which featured random initial positions and a fixed door location. The room was designed as a 10-meter by 10-meter square, with an open door width of 1.5 meters, to simulate a typical room environment. The height of all four walls is uniformly set to 2 meters. A three-axis frame (x, y, z) is defined with its origin at the room's center: the x-axis spans the left–right direction, the y-axis spans the forward–backward direction, and the z-axis is oriented vertically. The origin is defined as $(0, 0, 0)$. Around the origin, four obstacles are placed, each with a height equal to the wall height, i.e., 2 meters. Their precise coordinates are listed in Table 1.

Object	**X (m)**	**Y (m)**	**Z (m)**
Obstacle 1	-2.5	-3.0	2.0
Obstacle 2	3.0	-1.5	2.0
Obstacle 3	0.5	0.5	2.0
Obstacle 4	-1.0	2.5	2.0
Exit (Door Center)	0.0	5.0	1.0

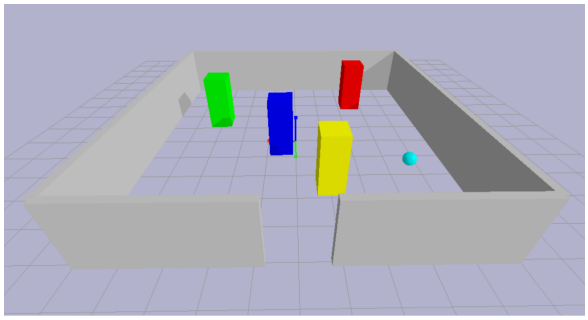
Table 1: Coordinates of Obstacles and Exit in the Environment

To simulate the Crazyflie drone, we represent the agent as a spherical rigid body with a radius of 0.2 meters and a mass of 0.5 kg. This proxy is equipped with basic collision and visual

properties to facilitate position tracking and obstacle interaction during training. To mimic the flight process of a real drone, the sphere starts from an initial height of $z = 0$ at each reset and ascends to the designated hovering height—half the wall height, which is 1.0 meter during environment initialization. Additionally, we capped the maximum horizontal velocity (x, y) of the sphere at 1.5 m/s and updated its state at fixed time steps of 20 ms. The sphere executes actions by translating its position and perceives the environment through simulated optical flow and range sensors. During the simulation, it can detect obstacles in four directions, and it learns navigation strategies based on this information. The simulation environment, as shown in Figure 2, consists of four enclosing walls, a designated exit (door), four obstacles, as well as an agent (blue ball).



(a) Top-down view of the simulated environment, showing the Crazyflie agent's final position, obstacle layout, and the door as the exit.



(b) Perspective view from inside the room facing the exit, illustrating the navigation challenge of passing through the door while avoiding obstacles.

Figure 2: Two perspectives of the simulation environment: top-down view (left) and door-facing view (right).

3.3.2 State-Action Representation

The experimental platform in the project is Crazyflie 2.1+ quadrotor[5]. Adjusting the position and orientation of aerial robots is challenging due to their extensive motion flexibility. Quadrotors are typically modeled and controlled in a continuous action space to accurately capture their dynamic behavior, which is crucial for real-world flight tasks.

Compared to discrete action spaces, the flight trajectory of a drone is more suitable for continuous and complex control. Therefore, in our simulation environment, we adopt a continuous action space to better reflect the physical behavior under realistic conditions. In order to replicate the perception abilities of the actual onboard hardware, the observation space is designed with two primary components.

Optical flow velocities (v_x, v_y) are contained in the first part of the observation, simulating the data provided by the FlowDeck on the Crazyflie. Here, v_x measures motion horizontally along x (left–right), while v_y measures motion along y (forward–backward). To improve the

feasibility of sim-to-real transfer, the values are perturbed using stochastic noise approximating real sensor errors, with variability bounded by a limited deviation of 0.01.

The second part emulates Multi-Ranger readings by capturing proximity data toward the front, rear, and both lateral sides. This is implemented using PyBullet's `rayTest` function, which simulates laser ranging with a maximum measurable distance of 4 meters. To replicate realistic sensor behavior, noise with a small spread (std. dev. $= 0.02$) is introduced.

Actions are represented by a pair of continuous variables, each lying in the interval $[-1, 1]$, and the resulting output is multiplied by 1.5 meters per second. In our coordinate system, motion toward the right corresponds to a positive x value, while positive y values represent forward motion and negative values backward. All flights are constrained to a fixed altitude of $z = 1.0$ meters, simulating the Crazyflie hovering at a constant height. Vertical movement is not considered in this project.

3.3.3 Reward Function Design

Since the Crazyflie used in our experiments is only equipped with two basic sensors—the Flow Deck and the Multi-Ranger Deck—it lacks any global positioning capability. As a result, the agent cannot perceive its global coordinates or the actual distance to the door during decision-making. To ensure alignment between simulated settings and actual hardware, the Euclidean distance to the door, used during training, is only employed for internal reward calculation and is not exposed to the agent. Instead, the agent relies solely on local sensor data for state perception and action selection.

Therefore, the effectiveness of the reward structure directly impacts the agent's ability to learn. Different mechanisms can lead to significantly different learning outcomes. To encourage the agent to effectively avoid obstacles and ultimately locate and pass through the door, this study proposes a well-structured, progressive reward function that incorporates both reward progress and reward shaping strategies[45], as shown in Table 2:

Reward Progress The reward signals are formulated to motivate the agent to advance effectively toward the target:

- **Terminal Reward:** When the agent successfully passes through the door—satisfying the condition that its x-position is within the door frame width and its y-position has crossed the door threshold, it receives a one-time terminal reward of $+5000$.
- **Stepwise Distance Reward:** The agent evaluates how its distance to the door changes by comparing the current and previous measurements at every time step. If the agent moves closer to the door, a positive reward is given; otherwise, a penalty is applied. This reinforces the agent's movement in the correct direction.

Reward Shaping Additional rewards are introduced to guide the agent more effectively and accelerate training convergence:

- **Staged Distance Rewards:** Multiple distance thresholds to the door are set. When the agent reaches these thresholds, corresponding rewards are triggered, providing incremental incentives and alleviating the sparse reward problem.

- **Forward Openness Reward:** When the front sensor detects a distance greater than 0.5 meters, the path ahead is considered clear. A normalized distance-based reward is then added:
$$\text{reward} \mathrel{+}= 1.0 \times \frac{\text{front}}{\text{self.max\_range}},$$
encouraging the agent to move forward more quickly.

- **Center Alignment Reward:** To encourage the agent to pass through the center of the door, an additional reward is given based on proximity to the center line ($x = 0$). Since the door is located at the center, relying solely on y-direction rewards might cause the agent to collide with the door frame after obstacle avoidance. Therefore, when the agent is near the door area, the closer it is to the center line, the higher the reward it receives.

- **Stagnation Penalty And Time Penalty:** If the agent remains stationary or gets stuck, a stagnation penalty is applied. Additionally, a small time penalty (e.g., -0.01) is imposed at each time step to push the agent toward a prompt completion rather than wasting time.

Obstacle Avoidance Logic Two types of strategies are designed to guide the agent in learning to avoid obstacles and fly safely:

- **Collision Penalty:** By detecting whether the agent has contact points with obstacles (contact points > 0), the agent immediately receives a -100 penalty, and done is set to True to terminate the episode.

- **Obstacle Avoidance Guidance:**
  - When any of the front, back, left, or right sensors returns a distance less than 0.3 meters, a penalty is applied.
  - If the dangerous direction is the front (front sensor), after evaluating lateral distances, the side with more open space receives a positive reward, guiding the agent to bypass the obstacle.

In summary, the reward function is carefully crafted to reflect the hardware limitations and task objectives. It provides the agent with a structured and interpretable learning signal to accomplish the obstacle avoidance and door-passing task even without global position awareness.

3.4 Simulation-to-Reality Adaptation

Sim-to-real adaptation involves applying a policy trained in a simulated setting to real-world scenarios. After the agent has successfully learned obstacle avoidance and door-crossing capabilities in simulation, we deploy the trained policy onto a Crazyflie 2.1+ platform[5], which is equipped with Flow and Multi-Ranger decks[6][7] for velocity sensing and obstacle detection. During real-world flight, the drone utilizes this policy model to infer its next flight action commands based on perceived sensor information.

#	Name	Condition (trigger)	Value in code
1	Door passed	$\lvert x\rvert < 0.8 \cdot \text{door\_half\_width} \wedge y \geq \text{door\_y} + 0.2$	+5000 & episode done
2	Step progress	$\Delta d = (\text{prev} - \text{curr})$ toward door	$\pm 3.5 \cdot \Delta d$
3	Distance milestones	First time current_dist $< t_i$	+threshold_rewards$[i]$
4	Forward-open bonus	$y \geq \text{door\_y} - 0.8 \wedge \text{front} > 0.5$ m	$+1 \cdot (\text{front}/\text{max\_range})$
5	Center alignment	$4.5 < y \leq 6.0 \wedge \lvert x\rvert < 0.8 \cdot$ door_half_width	$+10 \cdot \left(1 - \frac{\lvert x\rvert}{0.8w}\right)$
6	Collision	any ToF < 0.3 m	$-20 \cdot (0.3 - d)$
7	Steering hint	front < 0.3 m $\rightarrow$ choose clearer side	$+0.8 \cdot (\text{clearer}/\text{max})$
8	Stagnation	planar move < 0.01 m	-10
9	Time step cost	every step	-0.01

Table 2: Reward Function Components and Implementation

3.4.1 Coordinate Alignment And Sensor Mapping

According to the coordinate system defined in the simulation environment, the x-axis represents movement along the horizontal plane, whereas the y-axis denotes motion in the forward and backward directions. In the Crazyflie's own frame of reference[5], the x-axis governs forward–backward motion and the y-axis governs lateral motion, which creates a notable mismatch with the simulation coordinate setup.

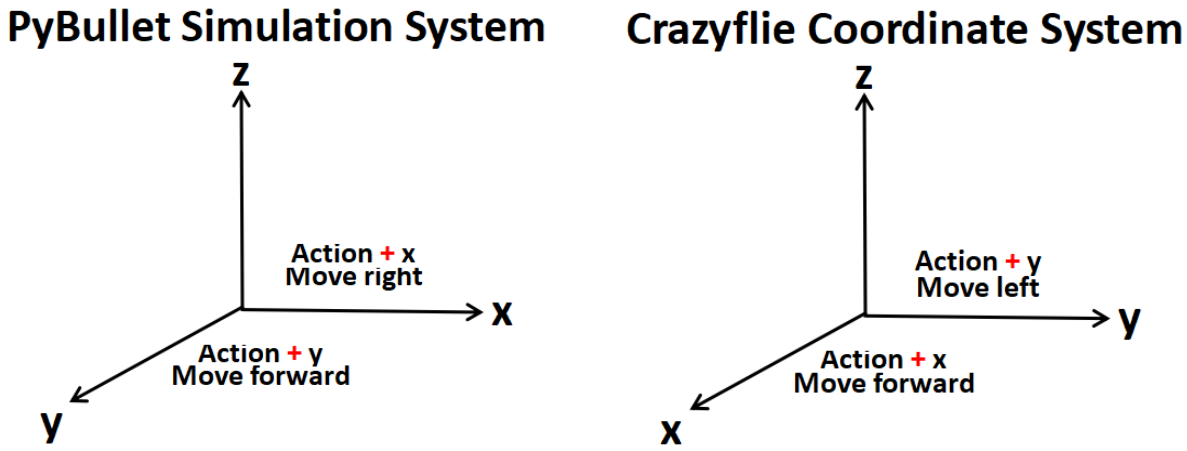
Therefore, during the actual deployment of the control script, it is necessary to transform the actions predicted by the model by swapping the x and y values. This ensures that the velocity commands sent to the Crazyflie align with its coordinate definitions. Rightward movement in the simulation is defined by the positive x-axis and leftward by the negative x-axis. In the Crazyflie's frame, the positive y-axis is associated with flight to the left, while the negative y-axis indicates rightward flight.

The coordinate mapping between simulation and real hardware can be intuitively understood through the coordinate comparison diagram shown in Figure 3.

3.4.2 Inference and Control Pipeline

This section describes the process of deploying the trained Proximal Policy Optimization (PPO) model onto a real-world quadrotor. The flowchart for the sim-to-real is shown:4. Initially, the pre-trained model file (`.zip`) from the simulation environment is loaded using `PPO.load()` and executed on a CPU. State observations are obtained from the Crazyflie's Flow Deck and Multi-Ranger Deck[6][7], including flow velocity and distance measurements from four directions. We stack the observations into a 6-dimensional vector, which serves as the input to the policy network.

The network outputs proportional velocity commands in two directions, denoted as v_x and v_y. These values, originally within the range $[-1, 1]$, are scaled by a maximum speed of $0.2\,\text{m/s}$ to linearly map them to actual velocity values, thereby determining the drone's flight



(a) For the simulation environment, the coordinate system where the axes are oriented with x running left–right and y running front–back.

(b) Coordinate system of the Crazyflie drone, where the x as the longitudinal (fore–aft) axis and y as the lateral (side-to-side) axis.

Figure 3: Coordinate system comparison between (a) the simulation environment and (b) the Crazyflie drone body frame, demonstrating the axis transformation required for sim-to-real deployment.

direction and speed. Given the mismatch between the coordinate frames of the simulator and the physical drone, as previously discussed, the output actions from the model need to be swapped accordingly. The velocity commands are then transmitted to the Crazyflie controller at fixed intervals, with each action lasting 0.2 seconds. To minimize flight jitter during real-world operation, a dead-zone threshold is applied. Action commands with magnitudes less than 0.05 are filtered out, and the remaining values are re-normalized to ensure smoother and more stable flight behavior.

3.4.3 Zero-Shot Transfer

A notable disparity exists between simulation and real-world conditions, primarily because simulations lack sensor noise, communication latency, and flight drift. To bridge this gap, we employed domain randomization by introducing Gaussian noise to both the optical flow and range sensor data during training. This simulates the uncertainties of the real world and improves the generalization ability of the trained policy. Furthermore, as previously discussed, we applied randomization to the drone's initial takeoff position during training within the simulated room. This initial state randomization increases the variability of flight trajectories, further enhancing the reliability of transferring policies from simulation to actual operation.

To enable direct deployment from simulation, the PPO policy was applied in the physical environment without further learning or parameter adjustment. Although the trained model has acquired basic obstacle avoidance capabilities, issues such as sensor perception errors or signal delays may occur in the real world, which may lead to occasional failures in obstacle detection and avoidance.

To ensure flight safety and prevent collisions during real-world deployment, an emergency obstacle avoidance mechanism is integrated into the system. When the distance to an obstacle ahead is detected, the system automatically switches to an emergency avoidance mode. In this

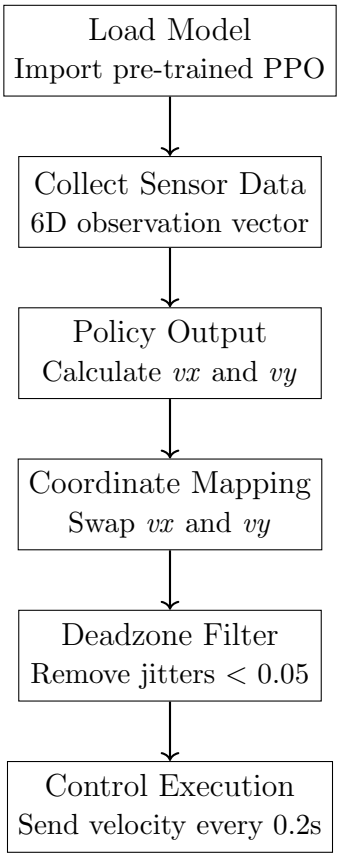


Figure 4: Sim-to-real PPO inference and control flow

mode, the system evaluates the available space to either side of the obstacle and guides the drone to maneuver towards the more open direction, thereby reducing the risk of collision.

Additionally, to prevent potential command interference caused by residual data from previous experimental runs, a "warm-start" mechanism is implemented. Before each takeoff, the system clears any remaining data that may still reside in the command queue, ensuring the accuracy and stability of flight control commands.

4 Experimental Analysis

4.1 Experiment Setup

4.1.1 Design of Simulation Parameters

The simulation was conducted using a custom environment based on PyBullet and wrapped with OpenAI Gym[46]. The agent's training employed the Proximal Policy Optimization (PPO)[8] algorithm implemented in the Stable-Baselines3[44] framework [44]. A DummyVecEnv wrapper was used to create a vectorized environment compatible with Stable-Baselines3. We set the environment seed to 42, which makes it possible to repeat the results. The PPO agent was configured with the following main hyperparameters:

- Policy network: We adopt a Multi-Layer Perceptron (`MlpPolicy`)[47], a commonly used feedforward architecture, to generate the action probability distribution.

- Number of steps per rollout (`n_steps`): 1024 means that the agent collects 1024 time steps of experience and then performs another learning process.
- Batch size (`batch_size`): 256 samples are randomly selected from the rollout data for each gradient descent step during training.
- Learning rate (`learning_rate`): 3×10^{-4} is set. A higher value leads to faster updates but may cause instability. In the experiment, the learning rate was increased to 4×10^{-4} and 5×10^{-4}. However, this adjustment did not lead to better performance.
- Entropy coefficient (`ent_coef`): 0.05 is a hyperparameter that balances how much the agent tries new actions versus relying on what it has already learned. We experimented with different values including 0.01, 0.03, and 0.05. The results showed that higher values encouraged more exploration and generally led to better performance.
- Clipping range (`clip_range`): A value of 0.4 was used to control the magnitude of policy updates in PPO. We also experimented with a smaller value of 0.2, but it resulted in poorer performance compared to 0.4.
- Discount factor (`gamma`): This parameter balances the weight of future versus immediate rewards. Although 0.99 was used initially, a lower value of 0.95 provided better results.
- GAE lambda (`gae_lambda`): 0.95 is used in GAE to compute the advantage function.
- Total training timesteps: 1,500,000 is the total steps the agent takes during the whole training process. We also conducted control experiments with 1,000,000 and 2,000,000 timesteps.

An evaluation mechanism was also integrated into the training process. The same environment was used for evaluation, and the agent was evaluated every 20,000 steps using 5 deterministic episodes. The best-performing model was automatically saved using `EvalCallback`, enabling reliable model selection for later testing and deployment.

Furthermore, to enhance the drone's generalization ability across various real-world environments in the future, the initial placement of the agent is randomized at the start of every episode. Although this setting increases the complexity of training, it significantly improves the robustness and adaptability of the learned policy across diverse scenarios.

4.1.2 Real-World Platform

Due to its compact design and flexibility, the Crazyflie 2.1+[5], a lightweight quadrotor small enough to fit in the palm of a hand, has become popular in research applications. Measuring 92 $\times$ 92 $\times$ 29 mm, it is powered by an STM32F405 MCU. As we can see in Figure 5, we equipped a flow deck underneath the Crazyflie to estimate relative velocity through optical flow and provide the drone with a fixed hovering. On the top, we also equipped a multi-ranger deck for detecting obstacles positioned ahead, behind, above, and on both sides of the drone, using this range data to avoid obstacles, as shown in Figure 6. In our experiments, we used an upgraded 350 mAh 30C battery, which increased the hover time from approximately 7 minutes to over 10 minutes without payload. Additionally, the upgraded 7$\times$20 mm motors paired with 51 mm propellers provided greater thrust, enabling the drone to carry extra sensors and perform high-speed maneuvers more effectively. The whole framework of Crazyflie 2.1+ in our experiments is shown as Figure 7.

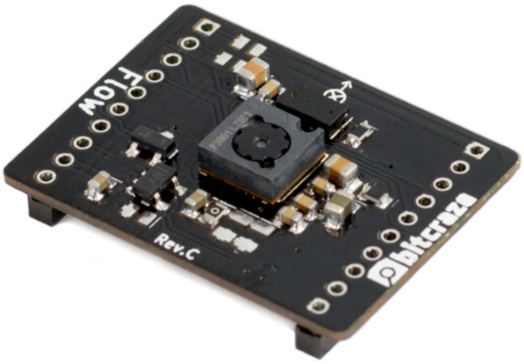


Figure 5: The Flow Deck (optical flow sensor) for Crazyflie

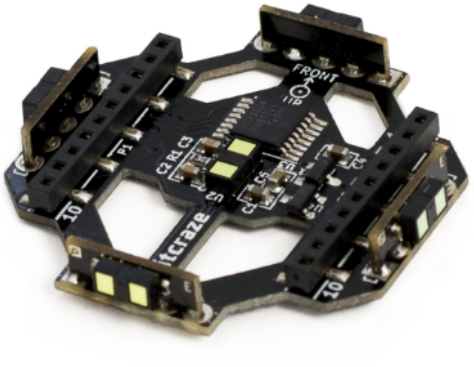


Figure 6: The Multi-Ranger Deck (range sensor) for Crazyflie

Figure 7: The Crazyflie 2.1+ is equipped with a Flow Deck and Multi-Ranger Deck for onboard sensing.

4.2 Training Performance in Simulation

Ten independent test episodes were conducted using the trained PPO agent in the simulated Crazyflie environment. The agent was initialized at random positions within the room, and its objective was to pass through the doorway without collisions. The trajectory, starting positions, rewards curve, and success status were recorded.

4.2.1 Simulation with fixed starting position

Under the condition of a fixed takeoff position, the experimental environment was configured with four obstacles to examine whether the agent could effectively avoid collisions and successfully locate the exit to complete the door-passing task, while keeping all other parameters unchanged.

To evaluate the agent's performance in a deterministic scenario, we fixed the start position at coordinates $[0.0, -4.0, 1.0]$, which was visualized in Figure 8, located far from the door and

behind multiple obstacles. The goal was to assess whether the trained policy could consistently navigate the same complex path and successfully complete the door-passing task.

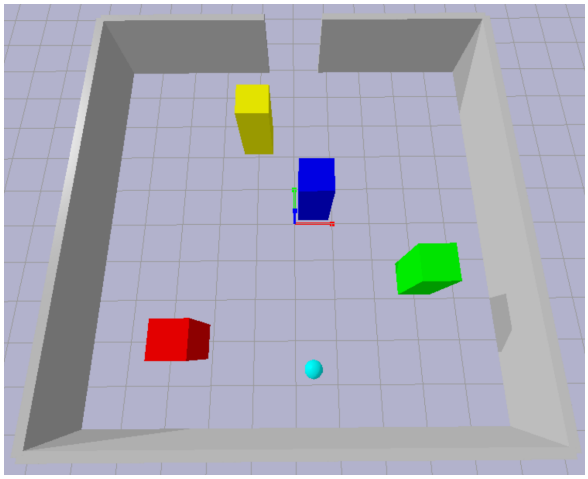


Figure 8: Simulation environment with takeoff from the fixed starting point $[0.0, -4.0, 1.0]$

Episode	Starting Position	Success
1	[0.0, -4.0, 1.0]	Yes
2	[0.0, -4.0, 1.0]	Yes
3	[0.0, -4.0, 1.0]	Yes
4	[0.0, -4.0, 1.0]	Yes
5	[0.0, -4.0, 1.0]	Yes
6	[0.0, -4.0, 1.0]	Yes
7	[0.0, -4.0, 1.0]	Yes
8	[0.0, -4.0, 1.0]	Yes
9	[0.0, -4.0, 1.0]	Yes
10	[0.0, -4.0, 1.0]	Yes
Success Rate		**100%**
Average Reward		**5409.77**

Table 3: Evaluation Results with Fixed Starting Position

As shown in Table 3, the agent achieved a 100% success rate across 10 episodes, with an average reward of 5386.09. These results demonstrate that the learned policy is highly reliable and effective when starting from a fixed position, even in the presence of multiple obstacles. This also reflects strong trajectory consistency and obstacle avoidance capability in a fixed-path scenario.

The flight paths of the Crazyflie drone, starting from a fixed position in a room with four obstacles, are illustrated in Figure 9. Each colored line represents a separate trial, and the drone consistently navigates around the obstacles and successfully reaches the door area. Despite slight variations, most trajectories follow a smooth and curved path, demonstrating stable and repeatable behavior under the same initial conditions.

4.2.2 Circular Sampling Around Fixed Start Position

In real-world scenarios, even if the Crazyflie takes off from a predefined position, there is always some drift around that point. To account for this, we conducted an experiment in which the starting positions were sampled within a circular area centered around the fixed point. The radius was set to 0.2 meters, reflecting the typical range of positional drift observed in practice.

As shown in the Figure 11, the small sphere represents the agent, whose initial position remains at $[0.0, -4.0, 1.0]$. Taking this point as the center, we randomly sampled 10 positions along a circular boundary with a radius of 0.2 meters to examine whether the agent could still avoid obstacles and successfully pass through the doorway.

It can be observed from the Figure 10 that although the agent started from slightly different initial positions across 10 trials, the final flight trajectories consistently overlapped, successfully avoiding obstacles and passing through the doorway in all cases. The success rate was 100% across all runs, which can be illustrated in Table 4. This indicates that minor initial drift does not negatively impact the navigation outcome and offers solid support for the policy's ability to generalize during the transition from simulation to real-world deployment.

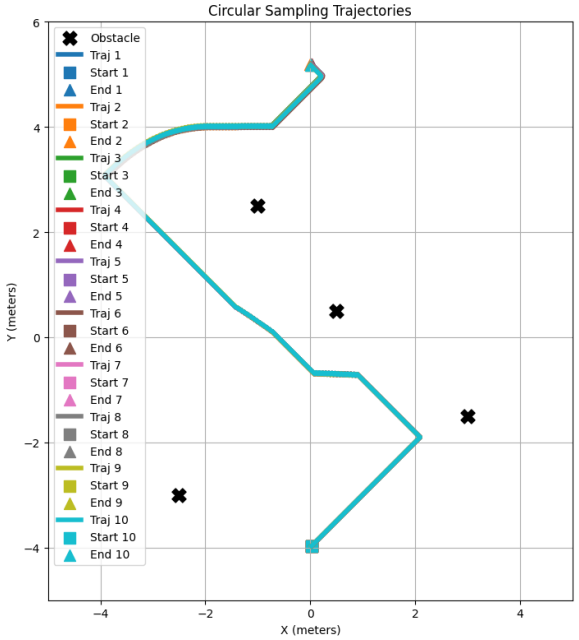


Figure 9: The trajectories of the agent with fixed starting positions in the presence of obstacles. All ten trajectories overlapped with each other.

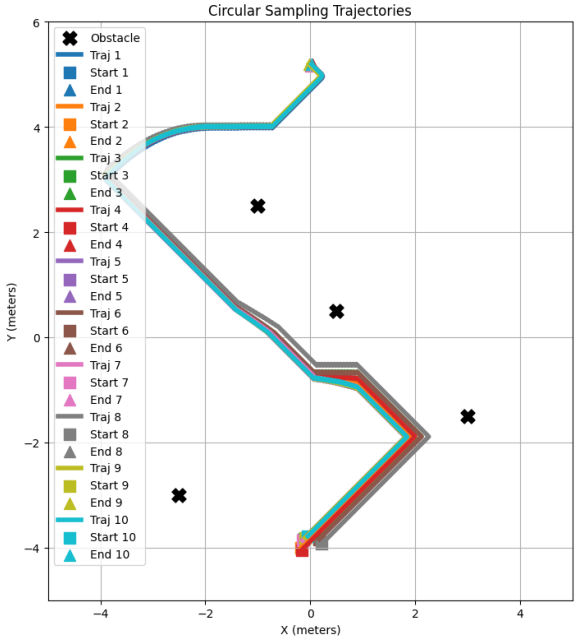


Figure 10: Ten trajectories are randomly selected from ten points centred on a fixed point as the starting point. The flight trajectories in the final stage also overlap.

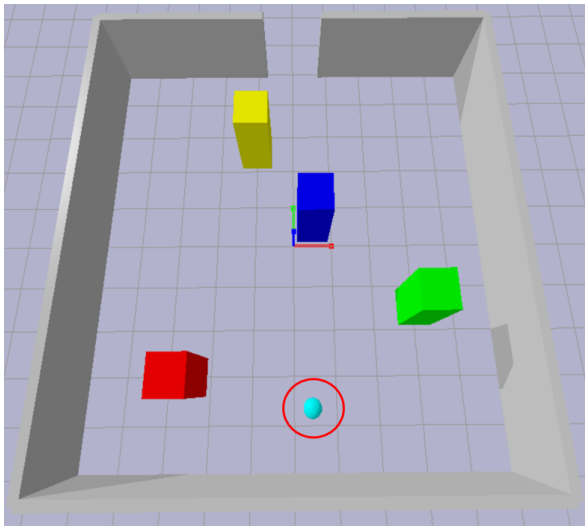


Figure 11: Circular Sampling Around $[0.0, -4.0, 1.0]$ and the radius is 0.2 meters

Episode	Starting Position	Success
1	[-0.14, -3.86, 1.0]	Yes
2	[-0.20, -4.03, 1.0]	Yes
3	[0.15, -3.86, 1.0]	Yes
4	[-0.19, -4.07, 1.0]	Yes
5	[-0.16, -3.88, 1.0]	Yes
6	[0.15, -3.87, 1.0]	Yes
7	[-0.17, -3.89, 1.0]	Yes
8	[0.20, -3.96, 1.0]	Yes
9	[-0.12, -3.84, 1.0]	Yes
10	[-0.07, -3.81, 1.0]	Yes
Success Rate		**100%**
Average Reward		**5409.26**

Table 4: Circular Sampling Starting Positions and Success Rate

4.2.3 Simulation with 10 Random Starting Positions

In addition to fixed-position takeoff tests, we conducted ten trials with randomized starting positions to evaluate real-world generalization. All other parameters remained unchanged, the starting positions of the agent in each episode are shown in the Figure 12, where each blue dot represents a takeoff point, and the corresponding flight trajectories are also plotted. As shown in the Figure 13, 8 out of 10 trajectories successfully avoided obstacles and passed through the doorway, resulting in a success rate of 80% (see the Table 5). The two failures were due

to one starting position being too close to an obstacle, and the other involving a collision with the doorframe just before exiting..

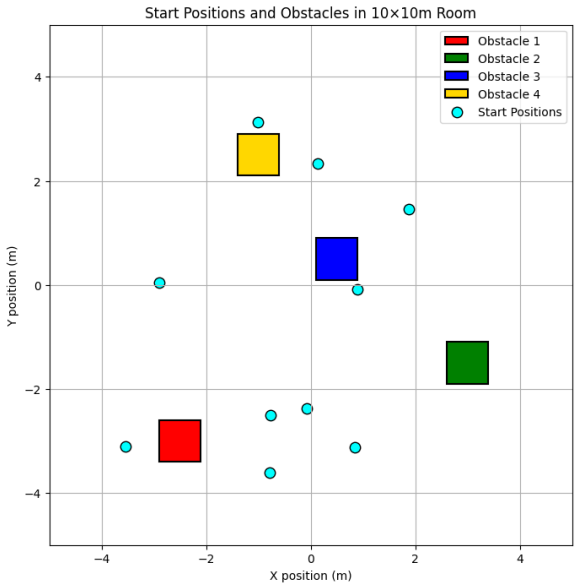


Figure 12: The same baseline model as in the previous experiment, regarding the distribution of starting points randomized in each of the ten episodes

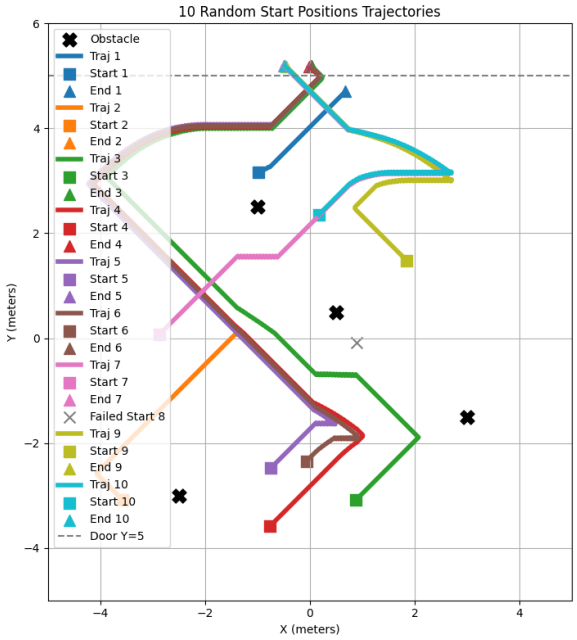


Figure 13: The trajectories of ten episodes corresponding to the randomized starting positions in Figure 10

In response to this issue, we adjusted the success criterion in the reward function for passing through the doorway, serving as a control experiment. Originally, the agent was rewarded as long as it passed within the doorway's full width; we revised this to require passage within 80% of the doorway width, effectively narrowing the valid corridor and increasing the difficulty. The starting positions are illustrated in Figure 14 and the updated trajectory Figure 15, the success rate improved to 90% (see Table 6). Trajectory 1 still failed due to a collision with the doorframe.

The experimental results demonstrate that even with randomized starting positions, the agent achieves a high success rate in the simulation environment, indicating strong generalization capability of the trained policy and providing a solid foundation for subsequent sim-to-real transfer. By tightening the reward function's criteria for passing through the doorway, we increased the task difficulty and observed an improvement in overall success rate. However, occasional collisions with the doorframe remain unresolved and will be further analyzed in the following sections.

4.2.4 Simulation with no obstacles

To perform a comparative analysis, we designed an additional experiment to evaluate how the trained obstacle avoidance policy performs in a simulation environment without obstacles. In this setup, all four obstacles were removed while keeping other parameters unchanged. The agent was tested across 10 episodes, each starting from a randomized initial position, to see if it could locate and exit through the doorway. As shown in the Figure 16 and Figure 17,

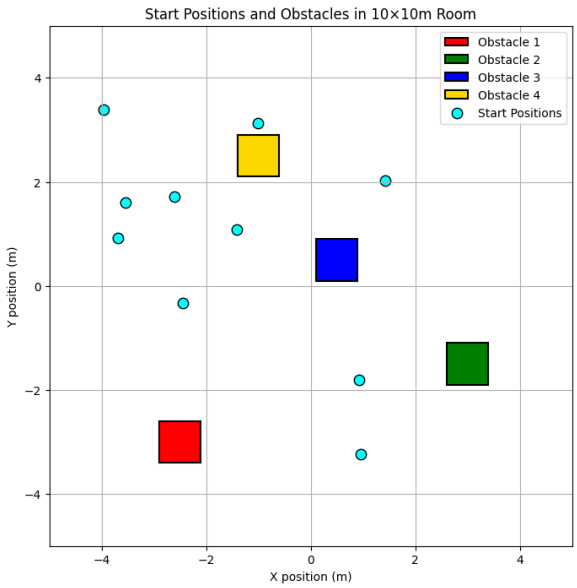


Figure 14: A new model trained after tightening the door conditions was added, and the distribution of 10 randomly generated starting positions was tested for 10 episodes

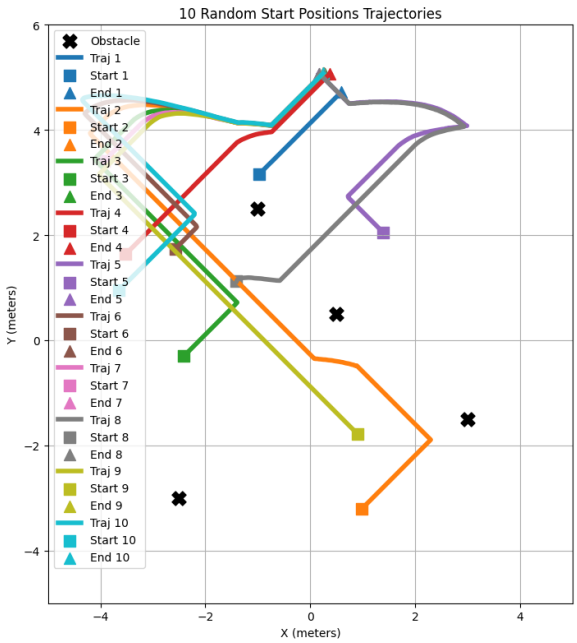


Figure 15: The trajectories of ten episodes in the new trained model corresponding to the randomized starting positions in Figure 12

Episode	Starting Position	Success
1	[-1.00, 3.13, 1.0]	No
2	[-3.55, -3.11, 1.0]	Yes
3	[0.85, -3.12, 1.0]	Yes
4	[-0.78, -3.62, 1.0]	Yes
5	[-0.76, -2.51, 1.0]	Yes
6	[-0.08, -2.37, 1.0]	Yes
7	[-2.90, 0.04, 1.0]	Yes
8	[0.90, -0.08, 1.0]	No
9	[1.87, 1.45, 1.0]	Yes
10	[0.14, 2.33, 1.0]	Yes
Success Rate		**80%**
Average Reward		**4301.93**

Table 5: Results of 10 randomly generated different starting positions in each flight scenario in baseline model

Episode	Starting Position	Success
1	[-1.00, 3.13, 1.0]	No
2	[0.95, -3.24, 1.0]	Yes
3	[-2.44, -0.33, 1.0]	Yes
4	[-3.55, 1.61, 1.0]	Yes
5	[1.42, 2.02, 1.0]	Yes
6	[-2.61, 1.71, 1.0]	Yes
7	[-3.96, 3.39, 1.0]	Yes
8	[-1.41, 1.09, 1.0]	Yes
9	[0.93, -1.81, 1.0]	Yes
10	[-3.68, 0.92, 1.0]	Yes
Success Rate		**90%**
Average Reward		**4801.49**

Table 6: Results of 10 randomly generated different starting positions in each flight scenario in the new trained model

compared to the previous experiment with obstacles, the flight trajectories are noticeably more direct and efficient, with fewer detours and turns. Notably, trajectory 5 passes through the area where an obstacle used to be, demonstrating the strong generalization ability of policy. This suggests that the policy is not overfitted to the specific obstacle configuration seen during training, but rather adaptable to various map layouts.

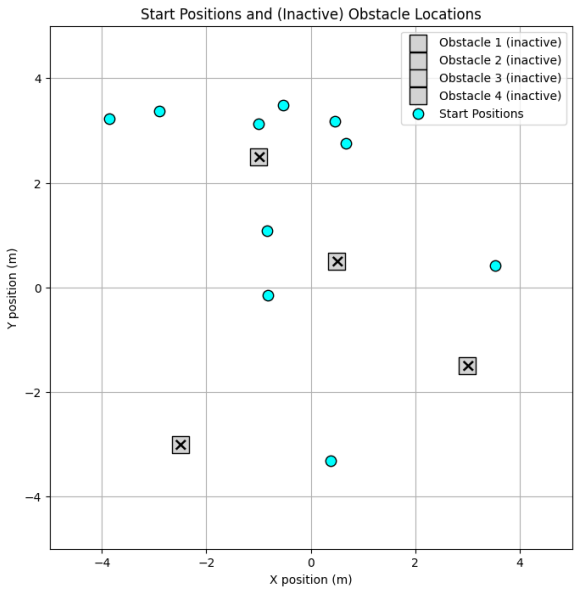


Figure 16: Random distribution of drone take-off positions in an obstacle-free simulated environment

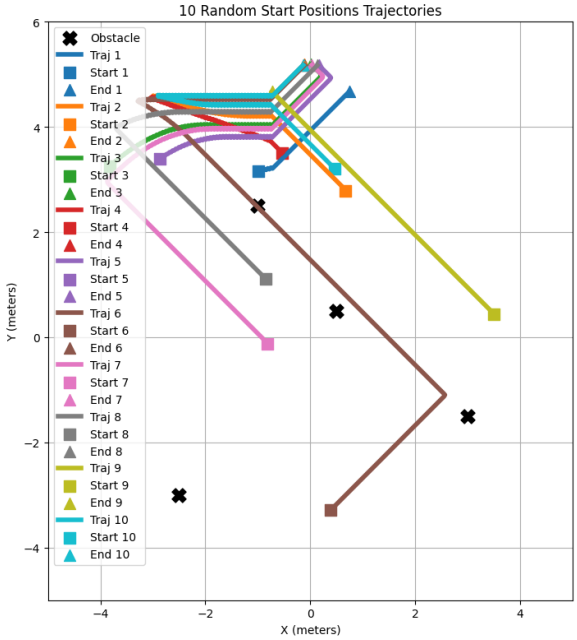


Figure 17: Trajectory plot of random take-off points for drones in an obstacle-free simulated environment. The sixth trajectory clearly shows that it crossed a position where an obstacle was originally present.

The Table 7 shows that the success rate remains at 80%, consistent with the earlier results. The two failures were again due to collisions with the doorframe, a recurring issue that had already been observed in the experiments with obstacles.

4.2.5 Reward Curve Comparison over Timesteps

When using the PPO algorithm for training, a large number of training steps are usually required to achieve good learning results. A longer training duration allows the agent to progressively refine its behavior in complex tasks and to better balance exploration and exploitation. As a result, the model becomes more capable of generalizing to a broad range of environments.

To analyze the impact of training duration, we started with 1,000,000 timesteps and increased the training length in increments of 500,000. The reward curves associated with the three different training durations are shown in the Figure 181920, illustrating how the policy evolves and converges over time. In the evaluation process, every 20,000 training steps, the agent is evaluated over 5 episodes, and the mean reward across these episodes is computed. As shown in the figure, PPO exhibits rapid learning in the early stages, with the reward quickly improving from negative values to positive ones. The agent even begins to pass through the doorway, receiving the final reward of 5000. During the mid-training phase, the reward continues to increase steadily and stabilizes in the positive range. However, in the later stages, the training becomes less stable, and the model may start to overfit, resulting in less optimal rewards. In fact, increasing the number of timesteps does not necessarily lead to better performance. To avoid relying on probabilistic outcomes—since we cannot determine in advance at which timestep the model will perform best—we choose to save the model with the highest

Episode	Starting Position	Success
1	[-1.00, 3.13, 1.0]	No
2	[0.67, 2.76, 1.0]	Yes
3	[-3.86, 3.22, 1.0]	Yes
4	[-0.53, 3.48, 1.0]	Yes
5	[-2.89, 3.37, 1.0]	Yes
6	[0.37, -3.31, 1.0]	Yes
7	[-0.82, -0.15, 1.0]	Yes
8	[-0.85, 1.08, 1.0]	Yes
9	[3.53, 0.42, 1.0]	No
10	[0.46, 3.18, 1.0]	Yes
Success Rate		**80%**
Average Reward		**4261.40**

Table 7: Evaluation results of 10 episodes without obstacle situations

average evaluation reward during training as the final best model. This model is then used for experimental analysis and the subsequent sim-to-real transfer.

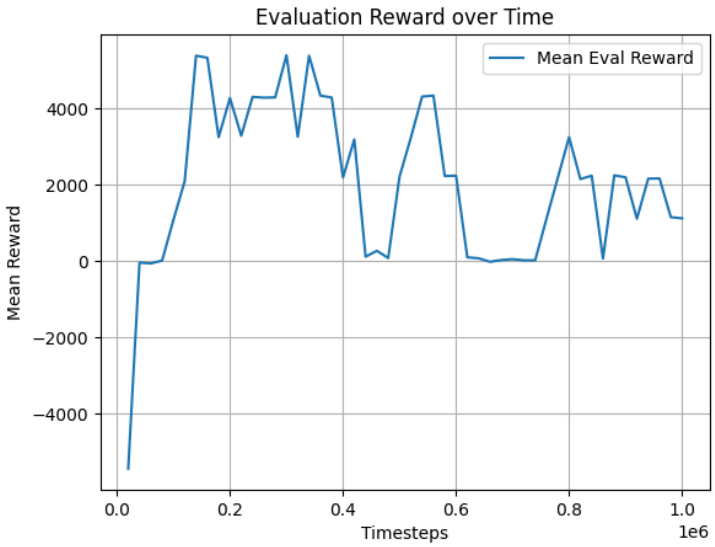


Figure 18: Mean reward curve of the PPO-trained agent over 1.0M training timesteps in a simulated environment with four obstacles and randomly generated takeoff points.

4.3 Training Performance in Reality

After training in the simulation environment, we selected a model with an 80% success rate and tested it by transferring it from simulation to reality. During the experiments, we utilized the Crazyradio PA (Power Amplifier), a USB radio transceiver, to establish wireless communication with the Crazyflie fitted with a Flow Deck for optical flow sensing and a Multi-Ranger Deck for distance measurement. The Crazyflie continuously sends back motion data (optical flow) and range measurements from four directions at 50 Hz. These sensor readings are received by the control script on the PC, processed by the pre-trained PPO model to infer appropriate

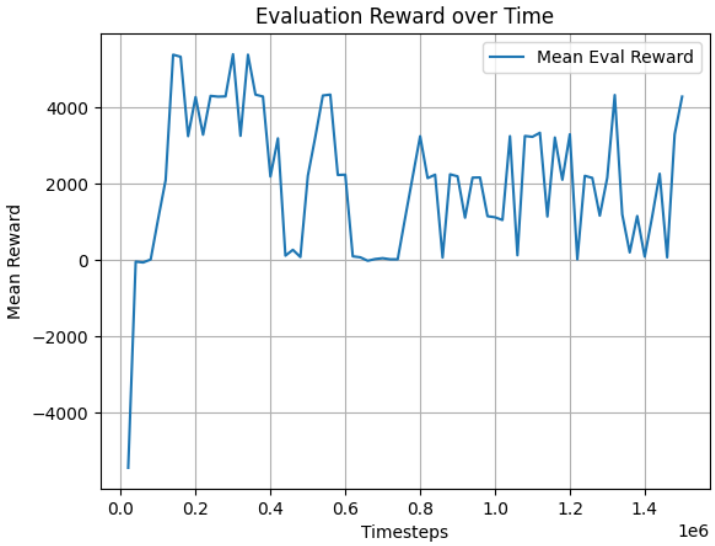


Figure 19: Mean reward curve of the PPO-trained agent over 1.5M training timesteps in a simulated environment with four obstacles and randomly generated takeoff points.

velocity commands, which are then transmitted back to the Crazyflie through Crazyradio PA. Table 8 summarizes the key configuration parameters applied in this process.

All real-world experiments were conducted in our apartment. The width of our room's door is 92 cm. Trash bins were used as obstacles, each measuring 23.5 cm in length, 16 cm in width, and 45.5 cm in height. For each trial, we marked a consistent takeoff point using transparent tape. Although slight deviations may occur after the Crazyflie takes off, our previous experiments have shown that a certain degree of drift is acceptable and does not affect the overall performance. Throughout all experiments, it was essential to ensure that the Crazyflie's front was always facing the doorway. This is because, during training in the simulation environment, the drone was consistently oriented in the same direction toward the door.

In addition, as previously mentioned, we also incorporated an emergency obstacle avoidance mechanism into the transfer script to prevent unexpected incidents such as communication delays or sensor measurement errors. When the front distance becomes too small, the drone determines the more open direction based on the left and right range measurements and executes a lateral evasive maneuver accordingly.

4.3.1 Crazyflie without Obstacles

Firstly, to assess how effectively the sim-to-real transfer performed and to establish a corresponding real-world baseline for the obstacle-free simulation, we first conducted a test in which the Crazyflie flew in our apartment without any obstacles, as shown in Figure 21 and Figure 22. A total of five trials were conducted, all resulting in a 100% success rate of passing through the doorway. In one of the trials, the Crazyflie collided slightly with the doorframe but still managed to complete the task. Table 9 summarizes the results obtained from the five trials.

Therefore, experimental evidence indicates that the trained PPO model achieves successful door-passing on the Crazyflie. Notably, the real-world environment differs significantly from the simulated one, highlighting the model's strong adaptability. This supports many existing studies

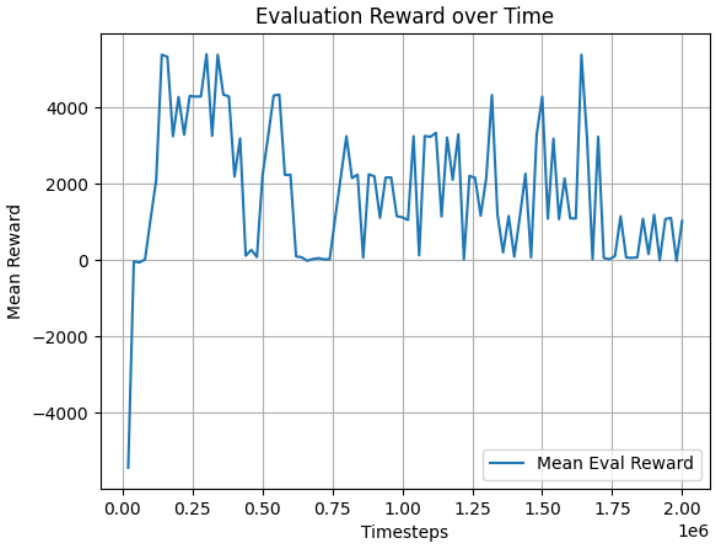


Figure 20: Mean reward curve of the PPO-trained agent over 2.0M training timesteps in a simulated environment with four obstacles and randomly generated takeoff points.

on zero-shot transfer using PPO. The high success rate further confirms the effectiveness of our sim-to-real transfer.

4.3.2 Crazyflie with Obstacles

As the model's training occurred in a simulated environment incorporating four obstacles and randomized starting positions, we designed a new real-world experiment to evaluate its performance under more complex conditions. Specifically, we introduced a trash bin as an additional obstacle into the previously obstacle-free room, as shown in Figure 23 and Figure 24, and observed the Crazyflie's ability to avoid the obstacle and pass through the door. The experimental setup remained consistent with previous tests: we conducted five flight trials using the same starting positions.Table 10 illustrates that the rate of successful door-passing decreased to 40%., with collisions occurring in 4 out of 5 attempts—either with the trash bin, the wall, or the doorframe. The average completion time increased to 36.6 seconds, and the average number of steps rose to 243, compared to 28.8 seconds and 141.2 steps in the obstacle-free scenario. It is evident that with the addition of obstacles, both the task duration and trajectory length increased, making the flight process more complex.

It is worth noting that in these experiments, the Crazyflie was facing directly toward the obstacle at takeoff, which increased the difficulty of navigation. In contrast, when the drone was not aligned with the obstacle, the success rate improved significantly. Overall, despite the added complexity and increased challenge, the trained model still demonstrated strong generalization and adaptability.

Parameter	Value	Description
URI	'radio://0/80/2M'	Communication address of Crazyflie, used to establish the connection between the PC and the drone.
LOG_FREQ_HZ	50	Logging frequency of sensor data transmission.
DT	1/LOG_FREQ_HZ	Minimum time interval for each control loop cycle (i.e., data read/send period), in seconds.
MAX_SPEED	0.2	Maximum flight speed of the Crazyflie, in meters per second.
HOVER_Z	0.2	Default hovering altitude after takeoff, in meters.
DEADZONE	0.05	Velocity deadband threshold to filter out minimal motion commands and reduce jitter.
EMERGENCY_SPEED	0.2	Speed applied when an emergency obstacle avoidance maneuver is triggered, in meters per second.
EMERGENCY_TIME	DT	Duration of each emergency avoidance action, aligned with DT, in seconds.
ACTION_DURATION	0.2	Duration for which each predicted action is maintained on the Crazyflie, in seconds.
SWAY_THRESHOLD	0.3	Threshold for deciding which side to swerve toward when an obstacle is detected in front, based on the difference between left and right range measurements.

Table 8: Key parameters used in the sim2real script and their descriptions

Exp No.	Obstacles	Start Position	Collision	Pass	Time (s)	Steps
1	No	Direct to the door	No	Yes	31	156
2	No	Direct to the door	No	Yes	32	170
3	No	Direct to the door	Yes	Yes	30	143
4	No	Direct to the door	No	Yes	29	127
5	No	Direct to the door	No	Yes	22	110

Table 9: Sim-to-Real Experiment Results (No Obstacles)

5 Discussion

5.1 Failure cases analysis

Initially, the success rate of the simulation experiment was only 40%. This occurred because the original reward function determined progress rewards using the distance from the starting point to the agent's current location. Later, we modified it to compare the distance between the agent and the center of the door at the current step versus the previous step. This new approach encouraged the agent to make consistent progress toward the goal in every step, resulting in a clearer objective and more directed behavior. As a result, the success rate significantly improved to 80%.

In the previous experimental chapter, we noted that almost all simulation failures were due to the agent colliding with the doorframe. The crash situation in the simulation environment is shown in Figure 25. Similar collisions were also observed in the real-world experiments with the Crazyflie. To further investigate the cause, we conducted an ablation study by testing different reward designs, aiming to understand how each reward component affects the learned policy. Despite modifying the reward function—such as tightening the condition for successful door

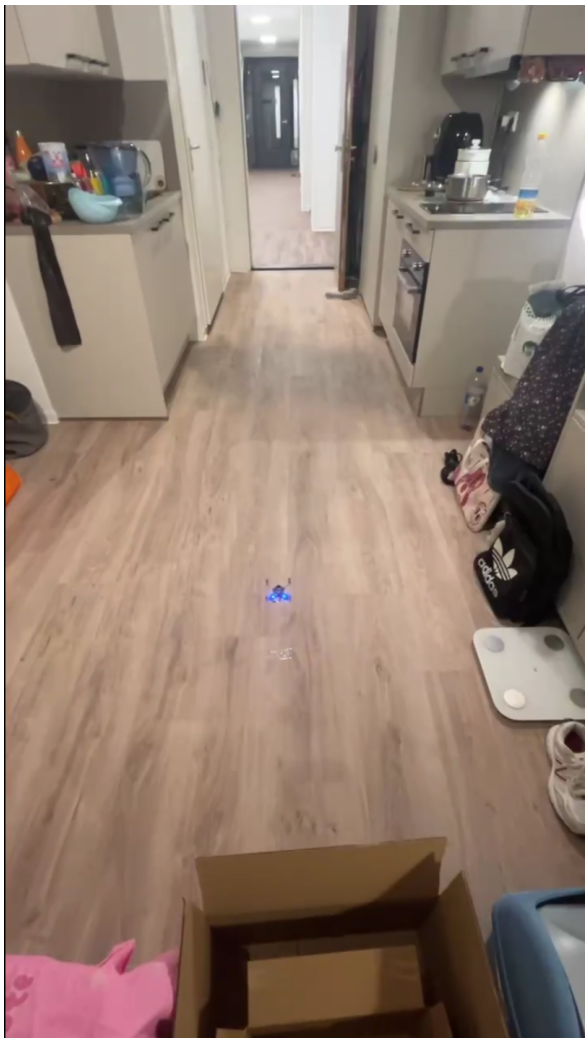


Figure 21: Real-world experimental scene with the Crazyflie. The environment contains no deliberately placed obstacles (e.g., trash bins). The image is a frame captured from the experiment video.

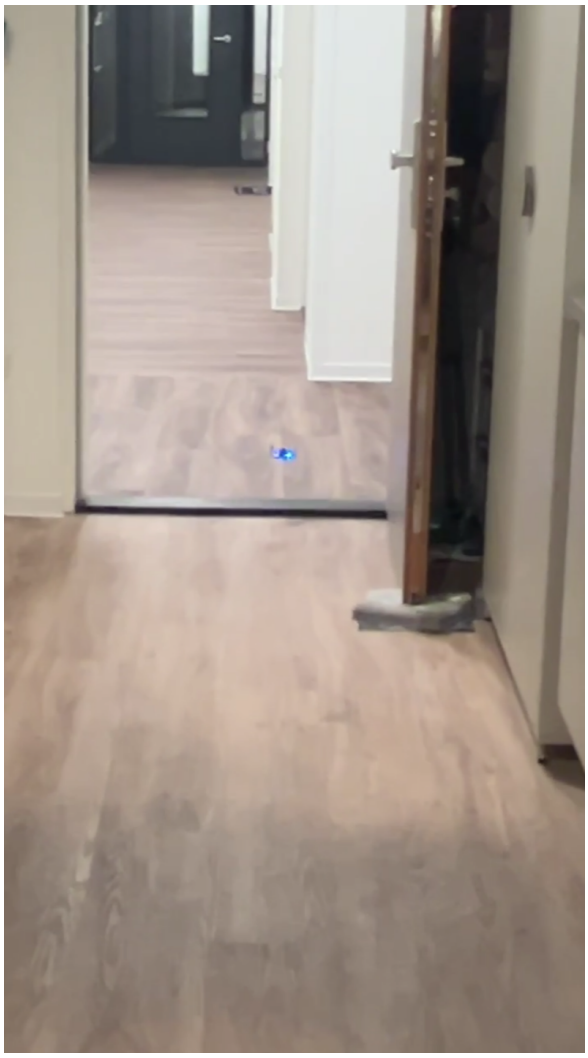


Figure 22: The Crazyflie successfully performed autonomous flight through a doorway. The image is captured from the experiment video.

passage to discourage close proximity to the doorframe—the results remained consistent. As shown in the previous chapter, these modifications did not reduce the occurrence of collisions with the doorframe.

We therefore hypothesize that the root cause is not due to reward design flaws, but rather the hardware limitations of the Crazyflie's onboard sensors. Specifically, the drone obtains range data along four predefined orientations — forward, rearward, leftward, and rightward. When an object is positioned diagonally, for instance at roughly 45 degrees toward the front-right (such as a door frame), the drone cannot perceive it and thus may inadvertently crash into it. To test this hypothesis and rule out the possibility of poor reward design, we examined one successful episode from the random-start experiments. As depicted in Figure 26, the agent, after identifying the doorframe within its detection range, it performed a clear turning maneuver and successfully passed through. This supports our conclusion: the agent is capable

Exp No.	Obstacles	Start Position	Collision	Pass	Time (s)	Steps
1	Yes	Direct to the Obs	Yes	No	49	364
2	Yes	Direct to the Obs	Yes	Yes	25	134
3	Yes	Direct to the Obs	No	Yes	69	433
4	Yes	Direct to the Obs	Yes	No	14	99
5	Yes	Direct to the Obs	Yes	No	26	185

Table 10: Sim-to-Real Experiment Results (With Obstacles)

of avoiding obstacles when they appear in the directions it can sense, but becomes vulnerable when obstacles lie outside those detection zones.

5.2 Simulation-to-Real Challenge

In the sim-to-real transfer phase, the trained policy proved it could generalize to the real world and repeatedly thread the doorway, yet its success rate with obstacles was noticeably lower than in simulation. This discrepancy mainly stems from the fact that sensor data in the simulation is idealized. Although we introduced Gaussian noise during training to mimic real-world uncertainty, it remains insufficient to fully replicate the complexity of real environments.

In reality, the Crazyflie drone may experience various sources of error during flight, such as drifting, communication delays, and especially instability in distance measurements. Through multiple experiments, we observed that although the Crazyflie prints the distances to obstacles in all four directions at each timestep, the sensor does not always reflect the actual obstacle distance in real time. Often, one additional timestep is needed before the measurement becomes accurate. This lag in perception introduces deviations in the observation data received by the model, which can lead to suboptimal or incorrect action predictions.

Apart from sensor uncertainty, the structural differences between the real and simulated environments also significantly impact transfer performance. The training was conducted in a 10×10 meter square simulated room with only four cylindrical obstacles, whereas the real-world experiments took place in a much more complex apartment environment. Although we only introduced a trash bin as the primary obstacle, other elements like cabinets, walls, and doorframes also served as unintended obstructions. The Crazyflie had to perceive and avoid all of these while still identifying and flying through the doorway. However, this difference between training and deployment environments aligns with the goal of our study: we ultimately aim for our model to address the problems in reality, where environments are inherently unstructured and unpredictable.

Interestingly, in some experiments, we observed that when approaching the door, the Crazyflie would momentarily retreat after detecting an obstacle, but then re-approach and successfully navigate around the trash bin to pass through the door. According to the sensor data logs, when an obstacle is detected ahead, the drone tends to back off to a safer position. However, since the training reward design encourages forward progression, the drone eventually resumes its forward path after avoiding the hazard. This behavior reflects an exploratory pattern of "retreat and advance," leading to task success. It also demonstrates that our model retains a degree of robustness and adaptability in real-world deployment, despite being affected by perception delays and increased environmental complexity.

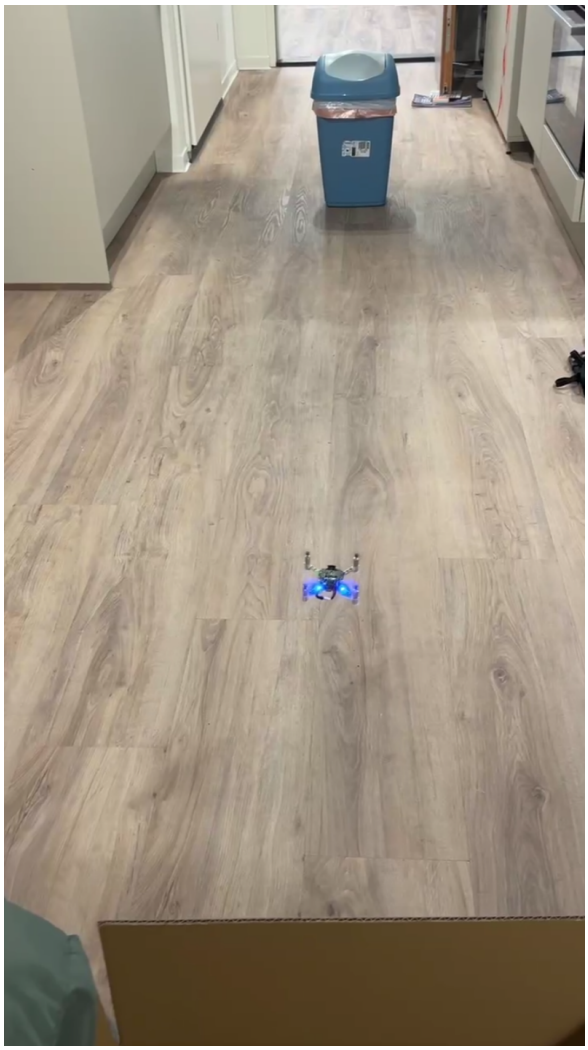


Figure 23: A trash bin was placed in the room as an obstacle, and the Crazyflie was positioned to take off directly facing it. The image is a screenshot from the experiment video.

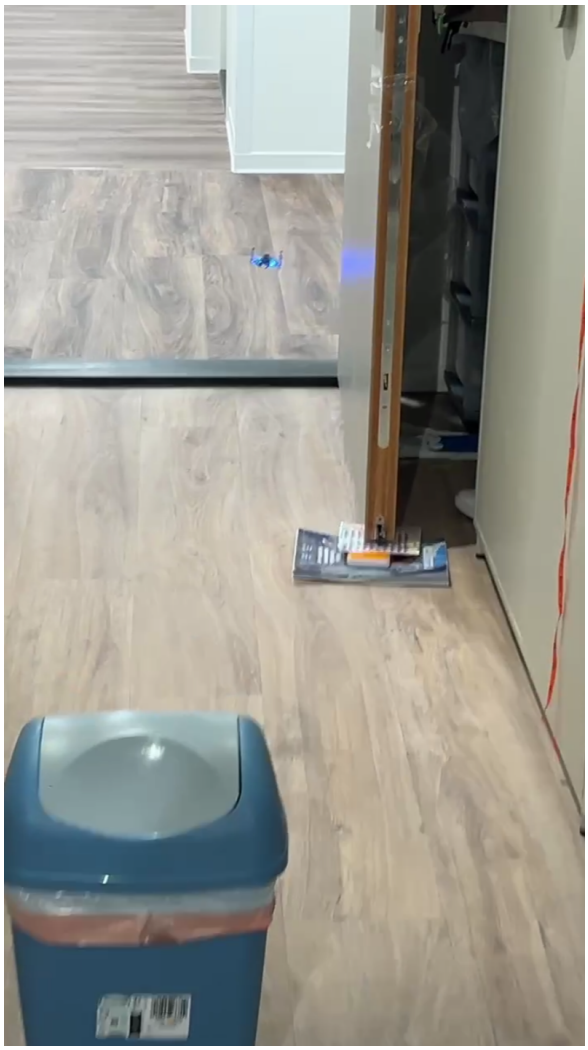


Figure 24: The Crazyflie successfully performed autonomous flight, avoiding the obstacle and passing through the room's doorway. This image is a screenshot from the experiment video.

6 Conclusion

Our experiments demonstrate that, in the simulation environment, our model can accomplish obstacle avoidance and door-passing tasks relying solely on the basic Flow Deck and Multi-Ranger Deck of the Crazyflie, achieving a success rate of up to 90%. This confirms the effectiveness of the PPO policy trained through reinforcement learning. In the sim-to-real experiments, the results further validate that our model not only performs well in simulation but is also capable of completing tasks even when deployed in real-world environments that differ entirely from the training setup. This result validates that the proposed simulation-to-reality approach can be successfully transferred and applied in practice. The study demonstrates that, by employing the PPO algorithm within a reinforcement learning framework, a Crazyflie equipped solely with basic distance sensors—without any external GPS or vision system—can effectively perform obstacle avoidance and navigate through doorways. This reduces the differ-

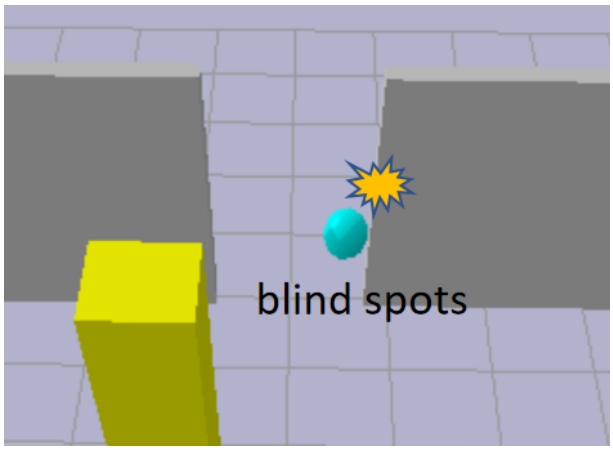


Figure 25: This is an example of a failed episode in the simulation experiment. The screenshot was taken while running the simulation in the graphical user interface of PyBullet. The sphere represents our drone, which collided with the door frame during this episode

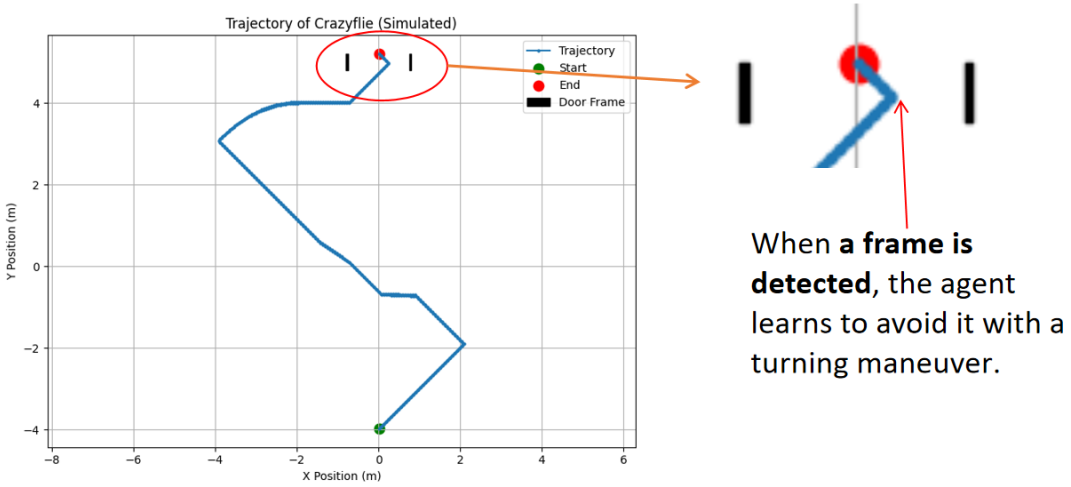


Figure 26: The image shows a trajectory from a simulation episode in which the drone successfully detected the door frame, performed a turning maneuver to avoid obstacles, and ultimately passed through the door

ence between the simulated setup and actual conditions, while also confirming the zero-shot transferability of PPO. It also highlights the effectiveness of reinforcement learning for autonomous drone navigation, which is an important research area for future intelligent systems. This study shows that even with training conducted in a single simulated environment, the model can still exhibit strong generalization capabilities when facing completely unknown real-world scenarios. Furthermore, this work demonstrates that, even under hardware limitations or restricted external resources, autonomous intelligent behaviors of small UAV systems can be achieved through well-designed training strategies and transfer mechanisms. Such capability provides strong support for future applications of small drones in complex or unknown real-world scenarios, such as autonomous exploration or disaster response, significantly reducing both labor and financial costs.

7 Future Work

Due to time constraints, we only introduced randomized starting positions to enhance the generalization ability of the trained policy. However, introducing additional randomization—such as varying the number and positions of obstacles—would likely further improve the sim-to-real transferability. Our entire experiment actually tested a wide variety of reward function combinations, gradually increasing the success rate from 40% to 80%. However, the process did not specifically record the experimental data. In the future, we can add more ablation experiments and then quantify the experimental results and compare them with the final version.

Moreover, in this study, we only tested the PPO algorithm. In future work, other reinforcement learning strategies could be explored. For instance, Soft Actor-Critic (SAC), an off-policy method with entropy regularization, allows for efficient experience reuse and typically achieves faster convergence. Its suitability for continuous control tasks makes it a strong candidate for onboard drone navigation. Similarly, Twin Delayed DDPG (TD3) is also well-suited for tasks involving small aerial robots in continuous action spaces. By using a dual-Q network structure and delayed policy updates, TD3 offers higher training stability, which is particularly beneficial for real-world deployment. A comparative study of different policy architectures for onboard execution would help identify the most robust and reliable approach.

Beyond single-agent control, future research could also extend to multi-agent reinforcement learning. For example, coordinating two or more micro-drones for collaborative navigation, obstacle avoidance, or exit-finding tasks opens the door to more complex behavior learning. Under these circumstances, incorporating swarm-intelligence-based methods, for example the Artificial Bee Colony (ABC) algorithm[48], could be a promising approach for decentralized path planning, followed by Multi-Agent PPO (MAPPO) to learn cooperative behaviors along those optimized trajectories. The combination of classical swarm intelligence methods and deep reinforcement learning could provide a powerful framework for real-time cooperation in dynamic environments.

Certainly, we can enhance the Crazyflie's experimental and perception capabilities by adding additional hardware modules. For instance, integrating the Loco Positioning System (LPS) allows the drone to obtain indoor global position information, enabling more advanced tasks such as global path planning and indoor SLAM. On the other hand, equipping the system with visual cameras or other high-precision sensors can significantly improve its obstacle avoidance and environmental awareness. However, while these upgrades can enhance performance, they inevitably increase hardware costs and system complexity, and often require higher demands on the physical environment. Therefore, practical applications require a careful balance between performance, cost, and system design.

Acknowledgements

As my two-year journey as a master's student draws to a close, I am deeply thankful to my supervisor, Dr. Mike Preuss, for his unfailing guidance and support during the course of this thesis. In our discussions, he consistently provided valuable academic insights and patiently assisted me in refining my research. His recognition and encouragement became an important source of motivation for me to complete this project. I would also deeply grateful to Prof. Aske Plaat. It was through his reinforcement learning course that sparked my keen interest in the field, which provided the essential background knowledge for me to pursue this reinforcement learning-based project.

Additionally, I am also grateful to my family and friends, with particular appreciation for my parents in China, for their unconditional trust and support. Whenever I felt anxious due to experimental setbacks, they were always there to comfort and encourage me through video calls. Their unwavering support has lifted me throughout this journey — from China to the Netherlands, from a bachelor's degree to a master's.

Looking ahead, I will keep moving forward with determination, living a life worthy of all the support I've received. Finally, I'd like to end with one of my favorite quotes:'Give me a challenge, and I will meet it with joy.'

References

[1] Hazim Shakhatreh et al. "Unmanned Aerial Vehicles: A Survey on Civil Applications and Key Research Challenges". In: *IEEE Access* 7 (2019), pp. 48572–48634. DOI: 10.1109/ACCESS.2019.2909530.

[2] Fadi Almahamid and Katarina Grolinger. "Autonomous Unmanned Aerial Vehicle Navigation using Reinforcement Learning: A Systematic Review". In: *Engineering Applications of Artificial Intelligence* 115 (2022), p. 105321. DOI: 10.1016/j.engappai.2022.105321.

[3] José Amendola et al. "Drone Landing and Reinforcement Learning: State-of-Art, Challenges and Opportunities". In: vol. 5. IEEE, 2024, pp. 520–539. DOI: 10.1109/OJITS.2024.3444487.

[4] Daniel Mellinger and Vijay Kumar. "Minimum Snap Trajectory Generation and Control for Quadrotors". In: *IEEE International Conference on Robotics & Automation(ICRA)*. 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[5] Bitcraze AB. *Crazyflie 2.X—Open-Source Nano Quadcopter*. https://www.bitcraze.io/products/crazyflie-2-1-plus/.

[6] Bitcraze AB. *Flow Deck v2—Optical Flow and Time-of-Flight Sensor Board*. https://www.bitcraze.io/products/old-products/flow-deck/.

[7] Bitcraze AB. *Multi-ranger Deck—Multi-Sensor Ranging Module*. https://www.bitcraze.io/documentation/hardware/decks/multi-ranger/.

[8] John Schulman et al. "Proximal Policy Optimization Algorithms". In: (2017). arXiv: 1707.06347.

[9] Jacopo Panerati et al. "Learning to Fly—a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021. DOI: 10.1109/IROS51168.2021.9635857.

[10] Noel E. Jakobi, Phil Husbands, and Inman Harvey. "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics". In: *Advances in Artificial Life*. 1995, pp. 704–720. DOI: https://doi.org/10.1007/3-540-59496-5_337.

[11] Andrei A. Rusu et al. "Sim-to-Real Robot Learning from Pixels with Progressive Nets". In: *Proceedings of the 1st Annual Conference on Robot Learning*. arXiv:1610.04286. 2017, pp. 262–270.

[12] Bitcraze AB. *Crazyradio 2.0*. https://www.bitcraze.io/products/crazyradio-2-0/.

[13] Fereshteh Sadeghi and Sergey Levine. "CAD2RL: Real Single-Image Flight without a Single Real Image". In: *Proceedings of Robotics: Science and Systems (R:SS)*. 2017. URL: https://www.roboticsproceedings.org/rss13/index.html.

[14] Malik Aqeel Anwar and Arijit Raychowdhury. "NavREn-Rl: Learning to fly in real environment via end-to-end deep reinforcement learning using monocular images". In: IEEE, 2018. DOI: 10.1109/M2VIP.2018.8600838.

[15] Andreas Seel et al. “Deep Reinforcement Learning Based UAV for Indoor Navigation and Exploration in Unknown Environments”. In: *2022 8th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2022. DOI: 10.1109/ICCAR55106.2022.9782602.

[16] Yuhan Xie et al. “Learning Agile Flights Through Narrow Gaps with Varying Angles Using Onboard Sensing”. In: vol. 8. 9. IEEE, 2023, pp. 5424–5431. DOI: 10.1109/LRA.2023.3295655.

[17] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018. URL: https://doi.org/10.48550/arXiv.1801.01290.

[18] Amudhini P. Kalidas et al. “Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles”. In: *Drones* 7.4 (2023), p. 245. DOI: 10.3390/drones7040245. URL: https://doi.org/10.3390/drones7040245.

[19] K. N. McGuire et al. “Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment”. In: 2019. DOI: 10.1126/scirobotics.aaw9710.

[20] Artem Molchanov et al. “Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019). DOI: 10.1109/IROS40897.2019.8967695.

[21] Kangtong Mo et al. “DRAL: Deep Reinforcement Adaptive Learning for Multi-UAVs Navigation in Unknown Indoor Environment”. In: *arXiv preprint arXiv:2409.03930* (2024). URL: https://arxiv.org/abs/2409.03930.

[22] Rik J. Bouwmeester et al. “NanoFlowNet: Real-time Dense Optical Flow on a Nano Quadcopter”. In: *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023. DOI: 10.1109/ICRA48891.2023.10161258.

[23] Mirza Aqib Ali et al. “Comparative Evaluation of Reinforcement Learning Algorithms for Multi-Agent Unmanned Aerial Vehicle Path Planning in 2D and 3D Environments”. In: *Drones* 9.6 (June 16, 2025), p. 438. DOI: 10.3390/drones9060438.

[24] Jemin Hwangbo et al. “Control of a Quadrotor with Reinforcement Learning”. In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103. DOI: 10.1109/LRA.2017.2720851.

[25] Katie Kang et al. “Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019. DOI: 10.1109/ICRA.2019.8793735.

[26] Antonio Loquercio et al. “Learning High-Speed Flight in the Wild”. In: *Science Robotics* 6.59 (2021). DOI: 10.1126/scirobotics.abg5810.

[27] Elia Kaufmann et al. “A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight”. In: *Proceedings of the 2022 IEEE International Conference on Robotics and Automation (ICRA)*. 2022, pp. 10504–10510. DOI: 10.1109/ICRA46639.2022.9811564.

[28] Elia Kaufmann et al. “Champion-level Drone Racing Using Deep Reinforcement Learning”. In: *Nature* 620.7976 (2023), pp. 982–987. DOI: 10.1038/s41586-023-06419-4. URL: https://doi.org/10.1038/s41586-023-06419-4.

[29] Bhaskar Joshi et al. “Sim-to-Real Deep Reinforcement Learning Based Obstacle Avoidance for UAVs Under Measurement Uncertainty”. In: IEEE, 2024. DOI: 10.1109/ICARA60736.2024.10553074.

[30] Jiayu Chen et al. “What Matters in Learning a Zero-Shot Sim-to-Real RL Policy for Quadrotor Control? A Comprehensive Study”. In: vol. 10. 7. IEEE, 2025, pp. 7134–7141. DOI: 10.1109/LRA.2025.3575011.

[31] Oliver Dunkley et al. “Visual-Inertial Navigation for a Camera-Equipped 25 g Nano-Quadrotor”. In: *IROS2014 Aerial Open Source Robotics Workshop*. Workshop extended abstract. 2014.

[32] James A. Preiss et al. “Crazyswarm: A Large Nano-Quadcopter Swarm”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3299–3304. DOI: 10.1109/ICRA.2017.7989376.

[33] Bardienus P. Duisterhof et al. “Learning to Seek: Autonomous Source Seeking with Deep Reinforcement Learning Onboard a Nano Drone Microcontroller”. In: *arXiv preprint arXiv:1909.11236* (2021). URL: https://arxiv.org/abs/1909.11236.

[34] Zhaohong Liu et al. “RESC: A Reinforcement Learning Based Search-to-Control Framework for Quadrotor Local Planning in Dense Environments”. In: *IEEE Robotics and Automation Letters* 10 (2025), pp. 9032–9039. DOI: 10.1109/LRA.2025.3592101.

[35] Jungwon Park et al. “Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee Using Relative Bernstein Polynomial”. In: *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*. 2020, pp. 434–440. DOI: 10.1109/ICRA40945.2020.9197162.

[36] Yunwoo Lee et al. “DMVC-Tracker: Distributed Multi-Agent Trajectory Planning for Target Tracking Using Dynamic Buffered Voronoi and Inter-Visibility Cells”. In: vol. 10. 5. IEEE, 2025, pp. 4842–4849. DOI: 10.1109/LRA.2025.3551255.

[37] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[38] Aske Plaat. *Deep Reinforcement Learning*. Springer Nature, 2022. ISBN: 978-981-19-0637-4. DOI: 10.1007/978-981-19-0638-1.

[39] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957. ISBN: 978-0691079516.

[40] Volodymyr Mnih et al. “Human-level Control through Deep Reinforcement Learning”. In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[41] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2018.

[42] John Schulman et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. arXiv:1506.02438. 2016.

[43] Yoshua Bengio et al. “Curriculum Learning”. In: *Proceedings of the 26th International Conference on Machine Learning (ICML 2009)*. ACM, 2009, pp. 41–48. DOI: 10.1145/1553374.1553380.

[44] Antonin Raffin et al. “Stable Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.164 (2021), pp. 1–8. URL: https://jmlr.org/papers/v22/20-1364.html.

[45] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. “Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping”. In: *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*. 1999, pp. 278–287.

[46] Greg Brockman et al. *OpenAI Gym*. 2016. URL: https://arxiv.org/abs/1606.01540.

[47] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536.

[48] Dervis Karaboga and Bahriye Basturk. “A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm”. In: *Journal of Global Optimization* 39 (2007), pp. 459–471. DOI: 10.1007/s10898-007-9149-x.

Appendix Terminal Output of Real-World Crazyflie Experiment

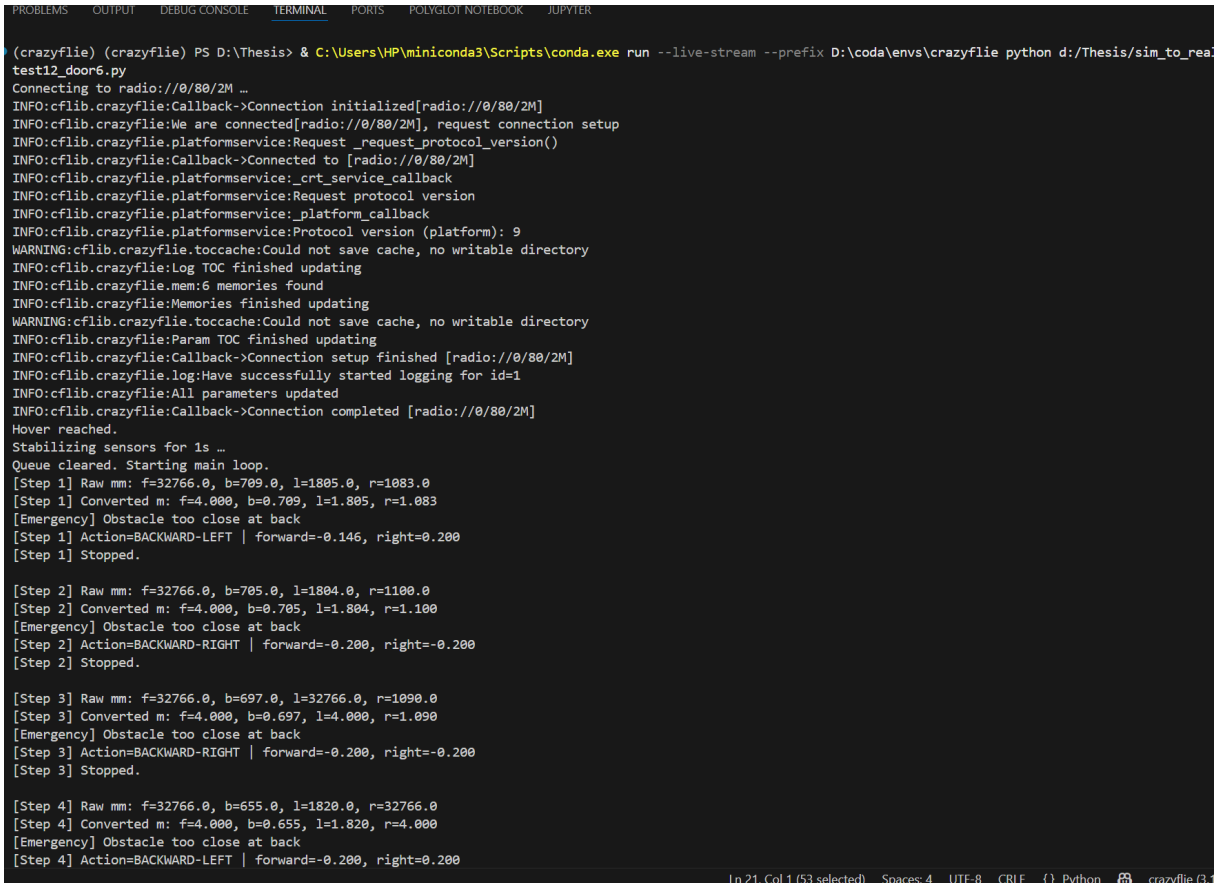


Figure 27: Terminal Log - Inference and Obstacle Avoidance

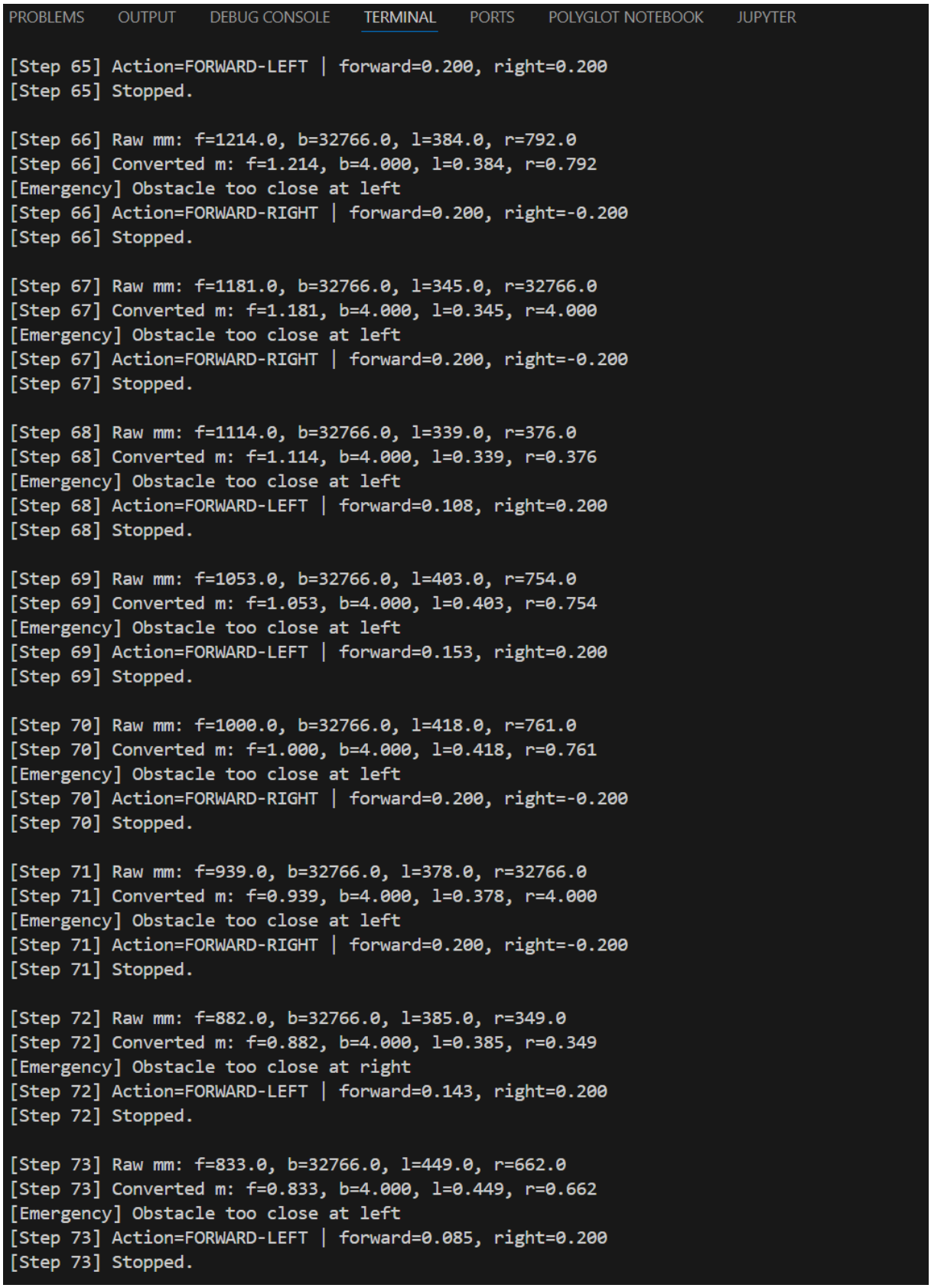


Figure 28: Terminal Log - Inference and Obstacle Avoidance

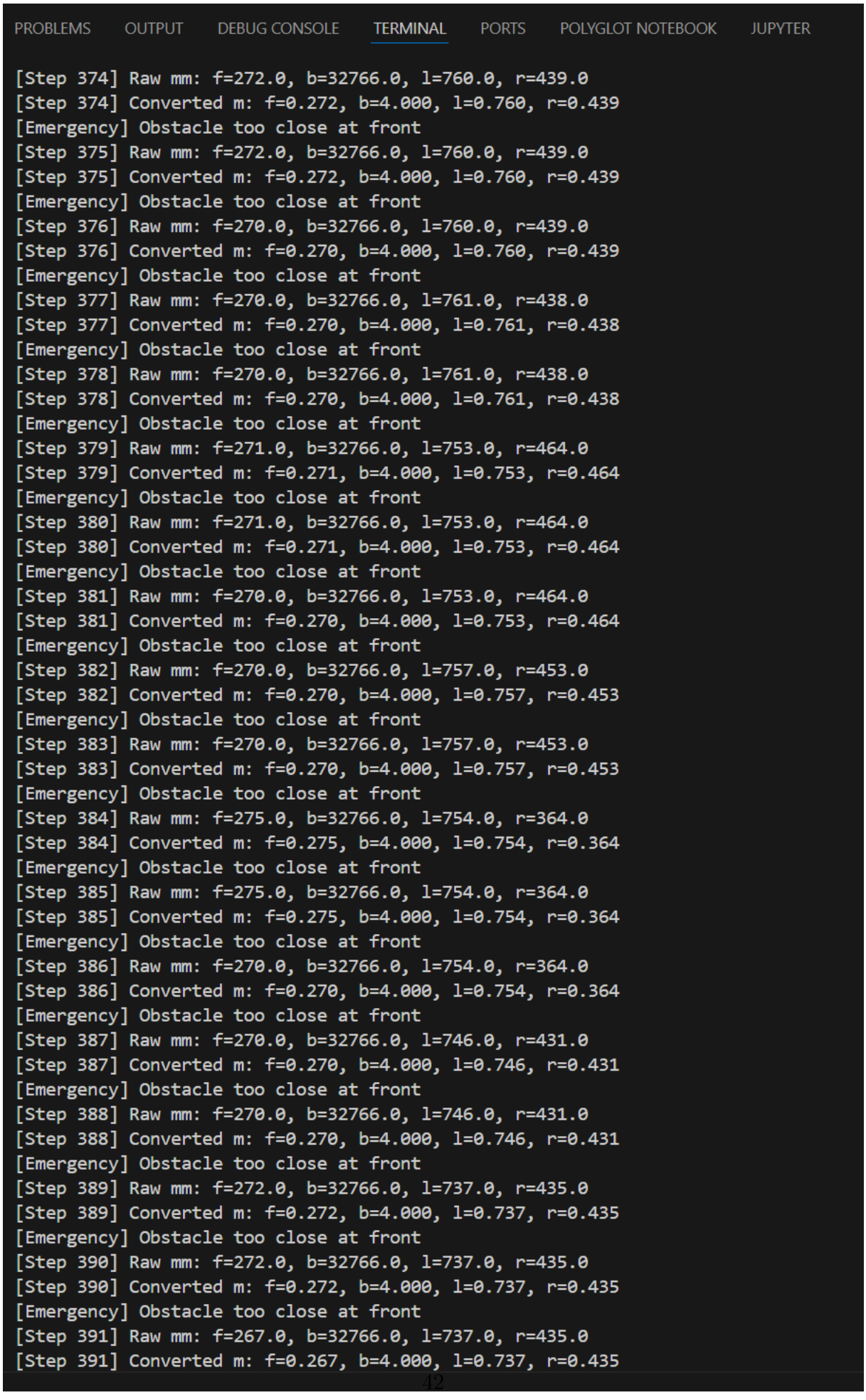


Figure 29: Terminal Log - After Send the Ctrl C command