TU WIEN Informatics

Sichere Positionskontrolle eines haptischen Geräts mit Hilfe von Reinforcement Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Victor Mittermair, BSc.
Matrikelnummer 11809916

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Hannes Kaufmann
Mitwirkung: Projektass. MSc. Mohammad Ghazanfari
Univ.Ass. Dipl.-Ing. Soroosh Dini-Mortezapoor

Wien, 27. November 2025

Victor Mittermair	Hannes Kaufmann

TU WIEN Informatics

Safe Position Control of an Encountered-Type Haptic Display through Reinforcement Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Victor Mittermair, BSc.
Registration Number 11809916

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Hannes Kaufmann
Assistance: Projektass. MSc. Mohammad Ghazanfari
Univ.Ass. Dipl.-Ing. Soroosh Dini-Mortezapoor

Vienna, November 27, 2025

Victor Mittermair | Hannes Kaufmann

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Victor Mittermair, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 27. November 2025

Victor Mittermair

Danksagung

Besonders möchte ich mich bei Mohammad Ghazanfari und Soroosh Mortezapoor für alle hilfreichen Ratschläge und Ideen bedanken. Die Ergebnisse dieser Arbeit sind das Resultat zahlreicher Meetings, selbst über Kontinente und verschiedene Zeitzonen hinweg. Außerdem danke ich Hannes Kaufmann und der VR/AR-Gruppe der TU Wien dafür, dass sie mir über mehrere Monate hinweg die Nutzung ihrer PCs aus Tokio ermöglicht haben.

Acknowledgements

I especially want to thank Mohammad Ghazanfari and Soroosh Mortezapoor for their insightful suggestions and ideas. The outcomes of this thesis are a result of numerous back-and-forth meetings we had, even across continents and different time zones. I also thank Hannes Kaufmann and the VR/AR group at TU Wien for letting me use their equipment from Tokyo over several months.

Kurzfassung

Haptisches Feedback ist ein zentraler Bestandteil immersiver Virtual-Reality-Erlebnisse. Herkömmliche Handcontroller können jedoch nur eingeschränkt realistische Berührungen vermitteln. Mobile Encountered-Type Haptic Displays (mETHDs) setzen hier an: Sie positionieren physische Objekte autonom und „just-in-time“, sodass Nutzer reale Objekte genau dann ertasten können, wenn sie diese in der virtuellen Umgebung erwarten. Die sichere Navigation dieser Roboter in unmittelbarer Nähe von VR-Nutzern, die ein Head-Mounted Display (HMD) tragen, stellt jedoch eine anspruchsvolle Herausforderung dar.

In dieser Arbeit wird ein End-to-End-Framework auf Basis von Deep Reinforcement Learning (DRL) vorgestellt, das die sichere und reaktionsschnelle Positionierung eines mETHD sowohl in Einzel- als auch in Mehrbenutzerszenarien ermöglicht. Das entwickelte System verarbeitet 2D-LiDAR-Daten mithilfe eines 1D Convolutional Neural Networks, um robuste Hindernisvermeidung sicherzustellen. Um die erhöhte Komplexität durch mehrere Benutzer zu bewältigen, wird das Steuerungsproblem in hierarchische Teilprobleme zerlegt: Eine strategische Policy schätzt die Nutzungsabsicht der VR-Nutzer ab, während eine Navigationspolicy die sichere Bewegung des Roboters ausführt. Quantitative Evaluationen in Simulationen zeigen, dass der DRL-Ansatz klassischen statischen oder heuristischen Baselines sowohl im Einzel- als auch im Mehrbenutzerfall deutlich überlegen ist. Die erlernten Policies reduzierten sicherheitskritische Eingriffe um über 88% und verbesserten die Positionierungszeiten für haptische Interaktionen bis um das Sechsfache. Diese Leistungsgewinne werden durch hochfrequente, proaktive Steuerung ermöglicht, die einen sicheren Betrieb des mETHD in unmittelbarer Nähe von Menschen erlaubt. Über VR-Anwendungen hinaus schaffen die in dieser Arbeit entwickelten Methoden eine Grundlage für intelligente, proaktive Robotersysteme, die sicher und zuverlässig in Mensch-Roboter-Interaktionsszenarien eingesetzt werden können.

Abstract

Haptic feedback is essential for immersive Virtual Reality (VR) experiences, yet traditional handheld controllers offer limited tactile realism. Mobile Encountered-Type Haptic Displays (mETHDs) address this by autonomously positioning physical props to enable realistic touch interactions "just-in-time". However, safely navigating these robots around users wearing Head-Mounted Displays presents a significant control challenge. This thesis proposes an end-to-end Deep Reinforcement Learning (DRL) framework for the safe and responsive positioning of a mETHD in both single and multi-user scenarios.

The proposed architecture processes 2D LiDAR data via a 1D Convolutional Neural Network to ensure robust obstacle avoidance. To manage the complexity of multi-user environments, the control problem is decomposed into hierarchical policies: a strategic policy that predicts user intent and a navigation policy that executes safe movement. Quantitative evaluation in simulation demonstrates that the DRL approach significantly outperforms traditional static and heuristic baselines in both single and multi-user scenarios. The learned policies reduced safety-critical interventions by over 88% and improved haptic positioning times by up to 6x. These gains are achieved through high-frequency proactive control, which allows the mETHD to operate safely in close proximity to users. Beyond immersive environments, the methodologies presented here lay the groundwork for intelligent, proactive robotic systems capable of operating safely alongside humans in broader Human-Robot Interaction contexts.

Contents

CHAPTER 1

Introduction

Haptics, a method of conveying the sense of touch to users, plays an important role in elevating the feeling of presence in immersive environments [1]. Haptic devices are often used in conjunction with Virtual Reality (VR) environments to enable tactile feedback of virtual objects. Usually, users experience a virtual world through visual and audio feedback provided by a Head-Mounted Display (HMD) coupled with handheld controllers for tactile feedback via vibrations [2]. This limits the possibility for a variety of haptic interactions that can occur in the real world. Consider architectural design, for example. An architect is designing an interior in a virtual environment. In a traditional VR setup, the visuals are the guiding factor for the architect to design the room. However, the ability to physically feel the textures of walls or the shape and flexibility of a newly placed couch could significantly increase the architect's immersion and spatial understanding, thereby improving both creativity and task performance. In general, the integration of haptic feedback in collaborative virtual environments can enhance user performance and realism [1, 3].

Several technologies have emerged in recent years, tackling the challenge of conveying physical touch to users in an immersive environment. Generally, you can divide them into passive [4] and active haptic devices. Active haptics are usually worn by the user either as a handheld [5] or on-body [6]. In contrast, passive haptic devices employ physical objects as haptic props. Passive haptic devices offer more realistic haptics compared to active haptics but are less scalable since the physical object or environment is usually static and has to be prepared beforehand [7].

To combine the best of both worlds, Encountered-Type Haptic Displays (ETHDs) [8, 9] have been developed. The idea of ETHDs is to move the haptic prop near a potential haptic interaction position, making a flexible and realistic haptic interaction possible. Mortezapoor et al. [10] developed CoboDeck, an autonomous ETHD that is able to serve interactions in a large-scale environment. This system consists of an omnidirectional mobile robot base and a collaborative robot (cobot) arm. The VR user walks around freely

in a defined space, and once the user wants to touch a virtual object, the robot moves in front of the user and extends its arm with the haptic prop, allowing them to feel the virtual object. CoboDeck makes it possible to have haptic interactions with theoretically any objects, as the end-effector of the arm can be fitted with a variety of objects. Since the robotic arm is attached to a mobile robot, large-scale haptic environments are possible. The goal of this thesis is to extend the capabilities of CoboDeck.

A significant challenge in this Human-Robot Interaction (HRI) scenario is to safely manoeuvre the robot, especially given that the user wearing an HMD is not able to see the robot. The autonomous robot, in our case, weighs around 130kg and is able to reach speeds of 3.0m/s, making it very crucial to have a robust and safe positioning system, especially in multi-user environments. In addition to safety, the delay for haptic interactions is of importance as well, since the users' immersion in the virtual environment relies on having an uninterrupted and seamless haptic experience. Thus, the primary challenge of this HRI system is to minimise haptic interaction delay while ensuring user safety.

Currently, CoboDeck uses a hand-crafted solution for positioning its mobile base to serve a single user during a VR session. This system operates on a rule-based algorithm, moving to predefined zones based on the user's state. While functional for basic interactions, this approach lacks adaptability. It cannot dynamically adjust to a user's unique movement patterns or predict their immediate intent, leading to potentially inefficient or non-intuitive robot positioning even in the single-user case.

This problem becomes even more prevalent in a multi-user environment. The introduction of multiple, independently moving users creates a large number of scenarios. A hand-crafted, rule-based algorithm cannot feasibly account for the complex decision-making required, such as path planning around other users, prioritising which user to serve, and ensuring safety for everyone in the shared space.

This thesis aims to address these limitations by first improving the pre-haptic interaction robot positioning for single users and then developing a robust multi-user system. Instead of expanding the hand-crafted solution, a data-driven approach through Deep Reinforcement Learning (DRL) is proposed. The reason is twofold. First, for the single-user case, it can learn a policy that adapts to individual user behaviours, moving beyond fixed rules to optimise positioning based on experience. Second, and more critically, DRL thrives in the high-dimensional, dynamic environments that multi-user scenarios create. By combining Deep Learning (DL) to interpret the complex state (e.g., positions and poses of all users) with Reinforcement Learning (RL) to discover an optimal action policy (where to move), the system can develop sophisticated strategies for navigation and interaction. This aligns with recent advances that have shown promising results for DRL in mobile robot control [11, 12, 13].

The following research questions will guide the scientific approach in the proposed thesis:

1. *What kind of data is necessary to successfully train a policy that can position a*

mobile ETHD safely?

2. *Can a data-driven approach outperform existing hand-crafted solutions in terms of safety and robustness?*

1.1 Contribution

- Proposing a novel DRL-based positioning solution for safe and responsive mobile robot control. The new solution enables single or multiple users to safely move around in a large-scale room during a haptic-enabled VR session.
- Design and implementation of a simulation environment using NVIDIA Isaac Sim and Isaac Lab platform that is used for training the RL agent. This will be discussed and explained in detail to facilitate future research.
- Recording a dataset of human motion trajectories recorded with a sub-millimetre motion tracking system.
- Evaluation of the proposed solution by comparing its performance against baseline methods.

1.2 Structure of the Thesis

The remainder of this thesis is structured to guide the reader from theoretical foundations to experimental validation. Chapter 2 and Chapter 3 establish the necessary context for the work. It begins by introducing the fundamental concepts of Reinforcement Learning and the technical specifications of the CoboDeck platform, including its underlying software architecture based on ROS and the FlexBE behaviour engine. This is followed by a review of the State of the Art in ETHDs and human-aware robot positioning, identifying the specific gaps in multi-user safety that this thesis aims to address.

The core contribution is detailed in Chapter 4, which presents the methodological approach. This chapter defines the design choices of the RL problem and details the system implementation. It also describes the creation of the human motion dataset, recorded with a sub-millimetre tracking system in the Arsenal lab, and the design of the simulation environment using NVIDIA Isaac Sim [14] and the Isaac Lab framework [15] to facilitate training.

Finally, this thesis assesses the performance of the proposed system. Chapter 5 presents the experimental setup, evaluation metrics, and quantitative results, comparing the DRL agent against baselines, including the existing hand-crafted CoboDeck solution and static methods. The results and their interpretation are discussed in Chapter 6.

The thesis concludes by outlining limitations and potential future research directions in Chapter 7, followed by a summary of the main findings in Chapter 8.

CHAPTER 2

Background

This chapter establishes the theoretical and practical foundations necessary for the proposed method. First, the principles of Reinforcement Learning are reviewed, moving from basic Markov Decision Processes to advanced Deep Reinforcement Learning algorithms like Proximal Policy Optimization [16]. Subsequently, the CoboDeck platform is introduced, detailing its hardware specifications, existing behaviour engine, and the supporting software infrastructure based on the middleware Robot Operating System (ROS).

2.1 Reinforcement Learning (RL)

This section presents a comprehensive overview of Reinforcement Learning (RL). Grounded in the foundational work of Sutton and Barto [17], the fundamental concepts are first established by formalizing the problem as a Markov Decision Process (MDP) and defining the central objective of return maximization. The discussion then advances to Deep Reinforcement Learning (DRL), detailing how neural networks are utilized to approximate value functions and policies. Finally, the Proximal Policy Optimization (PPO) algorithm is examined in depth.

2.1.1 Overview

Reinforcement Learning is a branch of machine learning concerned with how an agent learns to make decisions by interacting with an environment. At each step, the agent observes the current state of the environment, selects an action, and receives feedback in the form of a reward Figure 2.1. The agent's goal is to learn a strategy, or policy, that maximizes the total amount of reward it collects over time.

At the beginning of learning, the agent is unaware of what actions lead to the maximal reward. Instead, the agent must try to *explore* the environment by trying out different

actions. Over time, some actions lead to more reward than others, which the agent can *exploit* to receive the most reward. However, the exploitation of some actions could lead to a local optimum, since the agent might be missing unexplored areas that lead to better rewards. To optimize RL problems, exploiting the best known actions while also exploring the environment is necessary. This exploration-exploitation trade-off is often crucial in finding the optimal actions to take. [17]

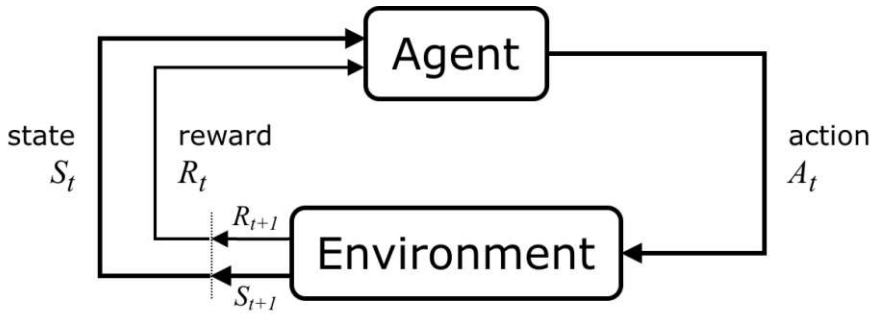


Figure 2.1: Agent-environment interaction cycle in an MDP. [17]

Sutton and Barto [17] formalize RL problems as a Markov Decision Process (MDP). At each timestep t, the agent receives a state $S_t \in \mathcal{S}$, performs an action $A_t \in \mathcal{A}$, and gets a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ from function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and new state $S_{t+1} \in \mathcal{S}$. This transition from one state to the next is governed by the transfer function $P : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times A \to [0, 1]$. This transfer function describes the dynamics of the environment and can be formalized with:

$$P(s', r \mid s, a) = Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2.1}$$

$P(s', r \mid s, a)$ outputs the probability of arriving in state s' and receiving reward r given state s and action a. Capilized letters with t subscribts such as A_t are random variables, while lowercase letters such as a are the actual values it can take, for example $a = \text{left}$ if $\mathcal{A} = \{\text{left}, \text{right}\}$

A sequence of state-action-reward tuples forms a trajectory. When the agent reaches a terminal state, the trajectory is over, and an episode ends. The goal of an agent is to maximize the (discounted) reward accumulated over an episode, called the return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

where $\gamma \in [0, 1]$ is called the discount rate, which determines how much future rewards influence current decisions. If $\gamma \approx 0$, then the agent is shortsighted, as the expected return only depends on the next few rewards. The decision-making function that maps states to probabilities of selecting an action is called the policy π. The agent learns the policy $\pi(a|s)$, which outputs the probability of selecting action a given the current

state s. The central optimization problem of RL is then to find the optimal policy that maximizes the expected return:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[G_t] \tag{2.3}$$

where $\mathbb{E}[\cdot]$ is the expected value of a random variable given that the agent follows policy π. To quantify how good a policy is, the state-value function $V_{\pi}(s)$ can be defined as:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s] \tag{2.4}$$

The state-value function $V_{\pi}(s)$ computes the expected return when starting in state s and following the policy π until termination. While there could be more than one optimal policy, all optimal policies share the same optimal state-value function V_*.

$$V_*(s) = \max_{\pi} V_{\pi}(s) \tag{2.5}$$

There are important properties of the state-value function that are used for learning. The Bellmann equations formalize these properties. For the value function $V_{\pi}(s)$ the Bellmann equation is:

$$V_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r|s, a)[r + \gamma V_{\pi}(s')], \text{for all } s \in \mathcal{S} \tag{2.6}$$

This equation shows an important relationship between the value of the current state and the values of its subsequent states. The value of the current state is equal to the sum of the expected reward received after acting based on π given s and the value of the expected next state. The Bellman optimality equation for V_*:

$$V_*(s) = \max_{a} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r|s, a)[r + \gamma V_*(s')], \tag{2.7}$$

states that the value of a state under an optimal policy is equal to the expected return of the best action starting in that state. So, if V_* is known, getting to the optimal policy is relatively straightforward. A policy that only considers the maximal next reward is optimal. This greedy approach is only possible due to the knowledge of V_*.

Practically all RL methods use some sort of generalized policy iteration (GPI), first coined by Sutton and Barto [17]. GPI consists of two processes, policy-evaluation and policy-improvement. In policy-evaluation, the value function gets updated to correspond to the current policy, then the policy-improvement process updates the policy to act greedily with respect to the value function. After the policy-improvement, usually the value function is no longer in accordance with the policy, so the policy-evaluation starts again. Once the value function is updated, the policy is usually not greedy anymore with

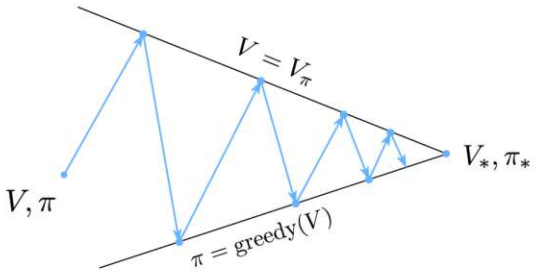


Figure 2.2: Illustration of generalized policy iteration (GPI) proposed by Sutton and Barto [17].

respect to the new value function, thus requiring another update. In theory, this process converges to the optimal value function and policy if both processes have stabilized. See Figure 2.2 for an illustration of that process.

Since it is often infeasible to know the exact value of every possible state (especially in continuous state spaces), approximations of those functions are necessary. This is where deep learning comes into play.

2.1.2 Deep Reinforcement Learning (DRL)

Deep Neural Networks (DNNs) have been proven to be capable approximators of nonlinear functions. This enables us to model state-value functions and policies in continuous state and action spaces. DNNs are usually parameterized by parameter vectors, which I denote as $\theta \in \mathbb{R}^d$ for the parameterized policy π_θ and $\phi \in \mathbb{R}^{d'}$ for the parameterized state-value function V_ϕ. For brevity, in the following, π and V refer to their parameterized versions, respectively.

An important class of algorithms that are commonly used in DRL are Actor-Critic methods. The actor (stochastic policy) takes actions in the environment, while the critic (value function) evaluates how good the taken actions were. To update the parameters of both the policy and the value function, some kind of loss or error is needed. The Temporal Difference error (TD-error) measures how good the prediction of the current value function is:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.8}$$

This is the formula of TD(0), or one-step TD, which only looks one step ahead. Note that this error can also be used to calculate a measure of how good the chosen action at timestep t was. Generalized advantage estimation (GAE) [18] is a way of estimating the advantage of an action, which tells the agent how much better a taken action was. Its formula is:

$$\hat{Adv}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{2.9}$$

The variable $\lambda \in [0, 1]$ is used to weigh future TD errors. If $\lambda = 0$, then GAE is equal to TD(0). How GAE and the TD error are tied together to update the policy and subsequently the value function will be explained by the example of Proximal Policy Optimization (PPO) [16], the algorithm used in this thesis.

Proximal Policy Optimization (PPO)

PPO is considered to be the State-of-the-Art Actor-Critic algorithm. It's a Trust Region Policy Optimization (TRPO) algorithm [19] which mitigates unstable and big changes to the policy through a clipped surrogate objective. The general idea of Policy Gradient (PG) methods [17] is to make actions that produce better outcomes more likely to appear in the future, while actions that produce worse outcomes are less likely. PPO is making sure that these updates are stable.

At the start of each PPO step, data has to be collected. This data is collected by letting the agent act in the environment for T timesteps and storing the states $s_{t:T}$, actions $a_{t:T}$, and rewards $r_{t:T}$ into a buffer. The start of a trajectory looks like this:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, s_3, r_3 \cdots$$

Note that this time the letters are lowercase, as these are the actual values the random variables take. In the PPO algorithm, the value function error term is a simple squared error loss, since we already have ground truth data in the buffer. The formula is defined as:

$$L^V = (V(s_t) - \hat{G}_t)^2 \tag{2.10}$$

T is usually much less than the episode length for efficiency reasons, which requires an estimator for G_t and Adv_t. The advantage $\hat{Adv}_t$ and return estimates $\hat{G}_t$ are calculated for each timestep in the trajectory.

$$L^{\text{CLIP}} = \hat{\mathbb{E}}_t \left[\min \left(k_t \hat{Adv}_t, \text{clip} \left(k_t, 1 - \epsilon, 1 + \epsilon \right) \hat{Adv}_t \right) \right], \tag{2.11}$$

where $k_t = \frac{\pi_{new}(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ is the probability ratio between taking action a_t in the new and old policies, and ϵ is a hyperparameter controlling the clipping range. Tested empirically, the ideal value of ϵ is 0.2. $\hat{\mathbb{E}}_t [\cdot]$ corresponds to the empirical average over a batch of samples (trajectories). To understand this clipped surrogate objective and why the min function is needed, we can take a look at four cases when the clipped and unclipped objective differ:

- $\hat{Adv}_t > 0$. The optimizer needs to update the parameters of the policy such that $k_t > 1$.

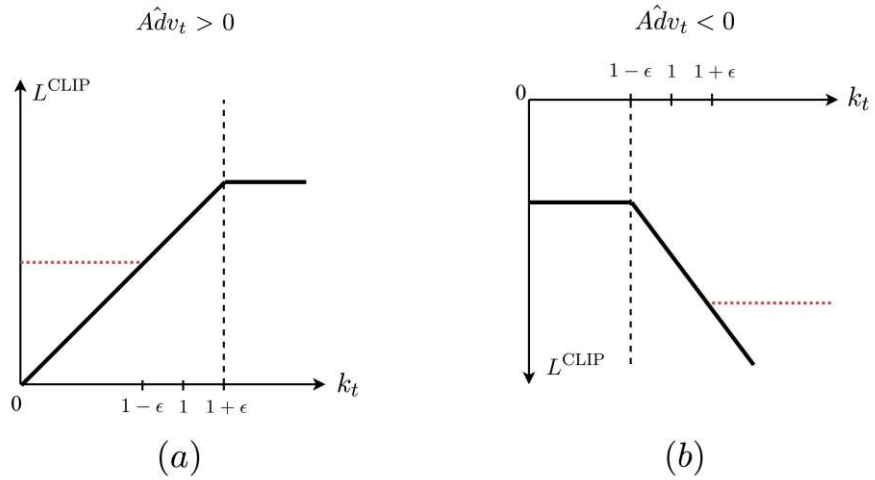


Figure 2.3: Plots showing the surrogate function L^{CLIP} as function of the probability ratio k_t. [16]. (a) The loss if the advantage is positive. (b) The loss if the advantage is negative. The red dotted lines correspond to the value if the surrogate function were not to have the min function and is only using the clipped objective.

  - Case 1: $k_t > 1 + \epsilon$. The new policy is overshooting the trust region. Clip the objective so the update is not overly confident, making the update unstable.
  - Case 2: $k_t < 1 - \epsilon$. In this case, the unclipped objective is the minimum, which is needed to correct the mistake. Otherwise, every $k_t < 1 - \epsilon$ would have a constant objective value, resulting in a zero gradient.

- $\hat{Adv}_t < 0$. The optimizer needs to update the parameters of the policy such that $k_t < 1$.
  - Case 3: $k_t > 1 + \epsilon$. In this case, the unclipped objective is the minimum. Otherwise, every $k_t > 1 + \epsilon$ would have a constant objective value, resulting in a zero gradient.
  - Case 4: $k_t < 1 - \epsilon$. The new policy is undershooting the trust region. Clip the objective so the update is not over-correcting.

To see an illustration of how L^{CLIP} is calculated, see Figure 2.3.

Algorithm 2.1 shows a high-level pseudocode of the PPO algorithm. In this case, the parameters θ and ϕ corresponding to policy π_θ and state-value function V_ϕ, are not omitted to make it clear which parameters are getting updated.

PPO has been shown to work in a variety of different tasks from robotics [20, 21] to Large Language Models (LLMs) [22].

Algorithm 2.1: Proximal Policy Optimization (PPO) — High-Level

Require: Initial policy parameters θ_0, state-value function parameters ϕ_0, learning rates α_π and α_V

for iteration $k = 0, 1, 2, \ldots$ **do**

Collect trajectory using policy π_{θ_k} in the environment for T timesteps.

Compute returns $\hat{G}_t$ and advantages $\hat{Adv}_t$

for epoch $= 1$ to K **do**

Compute clipped surrogate objective: L^{CLIP} and L^V

Update policy parameters:

$$\theta_{k+1} \leftarrow \theta_k + \alpha_\pi \nabla_\theta L^{CLIP}(\theta)$$

Update value function parameters:

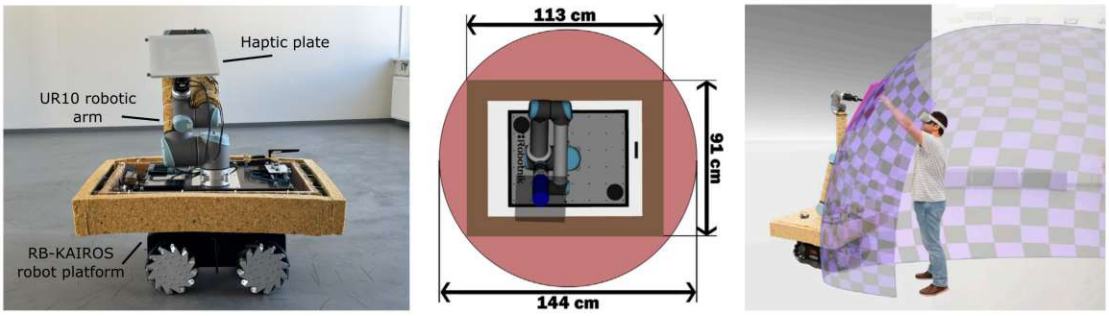
$$\phi_{k+1} \leftarrow \phi_k - \alpha_V \nabla_\phi L^V(\phi)$$

end for

end for

2.2 CoboDeck

Mortezapoor et al. [10] developed a large-scale Haptic VR System using a collaborative mobile robot. This thesis extends the capabilities of the existing system. Thus, a comprehensive understanding of the system is a prerequisite for the following chapters.



(a) (b) (c)

Figure 2.4: (a) An overview of the CoboDeck platform in real life. (b) Dimensions of the robot platform. (c) Visualization of the CoboDeck platform where a VR user is touching a virtual object during a VR session. [10, 23]

CoboDeck is originally intended to support architects in their design process. During a VR design session, the mobile ETHD provides haptic feedback of virtual objects such as walls, doors, and furniture. This is realized by an omnidirectional mobile robot platform

which carries a cobot arm that can hold physical props at its end-effector. With this, the mobile ETHD can simulate virtual objects in the real world in a large-scale environment. An overview of the system and its potential use case can be seen in Figure 2.4

Because the VR user and CoboDeck are colocated in the physical workspace, safely manoeuvring the autonomous robot is a major challenge in this HRI system. Given that the robot weighs 130kg and can reach 3.0 m/s, a robust and safe positioning system is crucial, particularly since the HMD-wearing user cannot visually track the robot. Beyond safety, minimizing haptic interaction delay is critical to ensure a seamless and immersive user experience. Ultimately, the goal is to guarantee user safety while achieving ultra-low haptic latency.

2.2.1 Behaviour Engine

To have a safe, responsive, and comprehensible system, Mortezapoor et al. utilize a behaviour engine that makes it easy to follow in what state the current system is. There are four different high-level behaviours the system can execute: *Chase*, *Haptic*, *Escape*, and *Inform*. See Figure 2.5 for the high-level state machine.

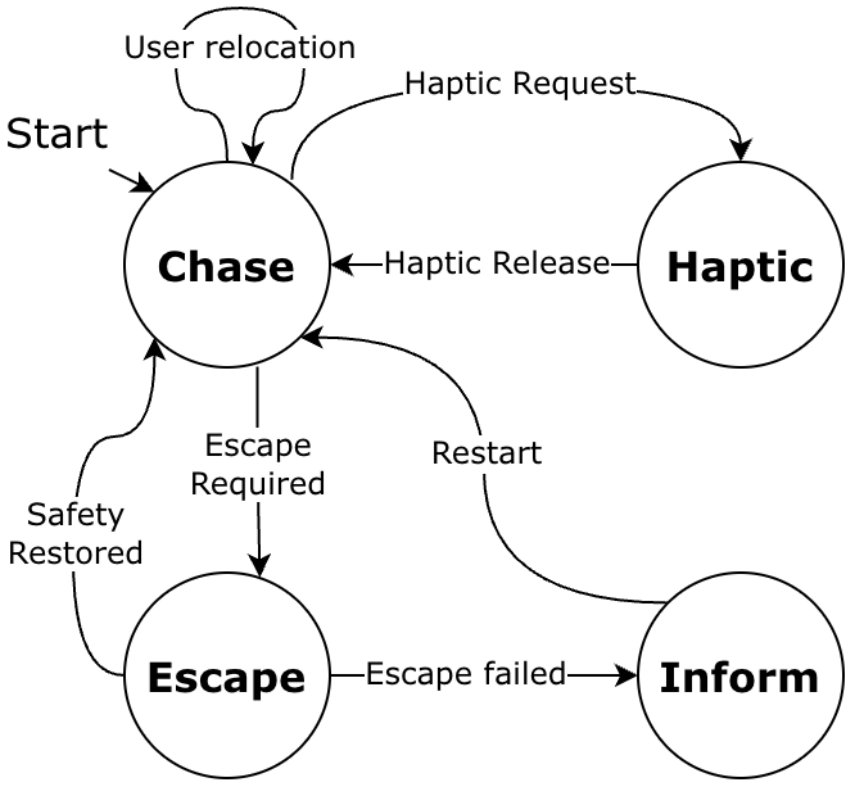


Figure 2.5: High-level diagram of the state machine used to define the robot's behaviour in CoboDeck [10]. Implemented with the Flexible Behaviour Engine FlexBE [24, 25].

Chase

This is the first behaviour that is triggered after start-up. It tells the robot to follow the user while no haptic requests are active. The goal is to be in a safe place but close to the user before a haptic request is issued, such that the haptic interaction latency is minimized. Depending on the user's position, view direction, and obstacles, potential chase poses are calculated. Details on how these chase poses are calculated is explained in Section 5.3. Once a chase point is chosen, the robot navigates to that chase point. This repeats until a haptic request is issued.

Haptic

Once a haptic request is issued, the haptic behaviour starts. Its purpose is to navigate the robot to the haptic interaction point, which is approximately 2m in front of the user. Once the robot has reached the haptic interaction point, the cobot arm moves to the desired end-effector position (DEEP). Once the cobot arm reaches the DEEP, the haptic interaction begins. After finishing the haptic interaction, the haptic behaviour ends, and chase is triggered.

Escape

Escape is triggered if the distance between the user and robot is below a predefined safety distance. This can happen due to sudden movements of the user, which the robot could not have evaded or through poor navigation caused by the chase behaviour. During escape, the robot quickly navigates to an escape point. This point is selected from a precalculated list of potential escape points. In general, these points are behind the robot in the direction of the user-robot vector, so the robot is farther away from the user after the escape behaviour finishes. After the successful navigation to the escape point, the chase is triggered.

Inform

If, for some reason, escape is unable to successfully navigate the robot to the escape point, the inform behaviour is triggered. This could happen if the path to the escape point is blocked by an obstacle. In inform, the user gets notified that there could be a possible collision with the robot, and they should stop moving. The robot model gets visualized in the VR environment to make the user aware of the robot's location. After the user stops, a restart signal is sent, and the behaviour engine transitions back to chase. If the robot is still stuck and can't find a suitable escape point or chase point, then it will be stuck in an endless loop until the user moves away from the current position.

This state-machine is implemented with the flexible behaviour engine FlexBE developed by Schillinger et al. [24, 25]. Data flow between components in the system is handled with the middleware ROS.

2.2.2 Robot Operating System (ROS)

The middleware ROS [26, 27] has been a popular way of unifying different robot systems or any device that comes with its different data definitions. ROS manages this exchange of data between different processes called *nodes*. Nodes communicate with each other through

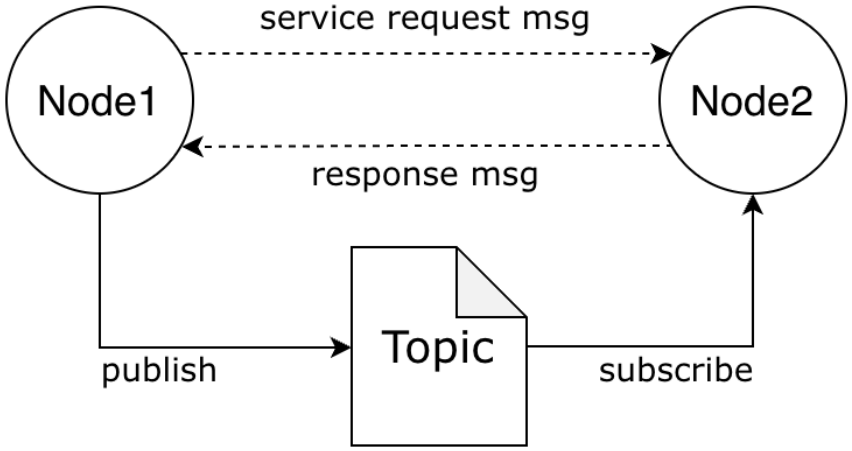


Figure 2.6: Communication between nodes in ROS. The synchronous messaging system between nodes is handled by services, while asynchronous messaging is done via topics.

messages which are strictly typed data structures. Messages can hold multiple data types (integer, floating point, float array, string, etc.). These messages have to be defined beforehand, which makes it theoretically possible to connect any device to this network. This peer-to-peer network, called the ROS computation graph, handles both synchronous and asynchronous communication between nodes. Synchronous communication is realized by so-called *services*. Services can be invoked by sending a request message to a node that is currently offering said service. Once the request message is processed by the receiving node, a response message is sent back. Asynchronous communication is implemented in a publish/subscribe model. A node can publish a *topic* which other nodes can subscribe to. Topics are defined by a topic handle as a string and the message type. Multiple nodes can publish or subscribe to a topic simultaneously. An overview of the ROS computation graph can be seen in Figure 2.6.

ROS has become the de facto standard in robotics research and development due largely to its modularity, extensive ecosystem of tools (like rviz for visualization and rosbag for data logging), and a large, active community. The original version, now known as ROS 1, established the foundational concepts of nodes, topics, and services described above. However, as robotics applications grew in complexity, particularly for industrial and commercial systems, their limitations (e.g., in real-time performance and multi-robot networking) became apparent. This led to the development of ROS 2, a significant redesign built to address these challenges. ROS 2 offers improved support for real-time

systems, enhanced security features, and utilizes the industry-standard Data Distribution Service (DDS) for its underlying communication, providing more robust and efficient data exchange.

In the context of this thesis, ROS serves as the critical software backbone connecting our various components, such as the simulation environment, sensor processing, policy inference, and actuator controllers. The current CoboDeck system uses ROS 1 Melodic due to the fact that development started a few years back. However, the policy inference is running on ROS 2 Humble, since the relevant packages required a newer ROS version. To exchange data between the two versions, ROS provides the capability of bridging topics through a package called ros1_bridge. This made it possible to use both ROS versions in our system.

CHAPTER 3

Related Work

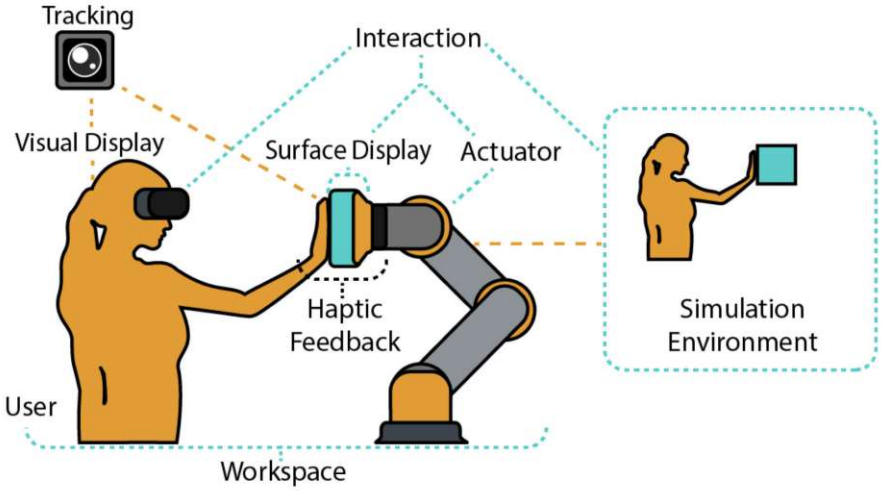
This chapter surveys the state of the art across three distinct domains essential to the proposed system. First, Encountered-Type Haptic Displays (ETHDs) are examined. Then, existing methodologies for human-aware position control are reviewed, ranging from social motion models that respect personal space to active human-following strategies. Finally, the application of Deep Reinforcement Learning in mobile robotics is analyzed, highlighting its efficacy in dynamic environments and identifying current limitations regarding multi-user VR contexts.

3.1 ETHDs

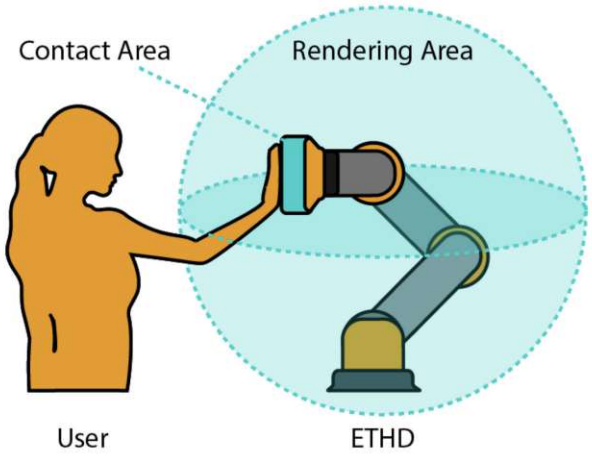
Inspired by the works of McNeely [28] and Hirota et al. [8], Yokokohji et al. [9] first coined the term Encountered-Type Haptic Display (ETHD) in the late 90s. ETHDs are characterized by the ability to serve as a haptic device in a "just-in-time" manner [29]. Unlike active haptic devices [5, 6], which generally restrict the movement of the user by being uncomfortable and bulky, ETHDs only provide haptic feedback once the user wants to touch a virtual object. This is facilitated by tracking the position of the user and moving the ETHD accordingly.

There are several ways ETHD systems are configured to make an encounter possible. Mercado et al. [29] defined grounded and ungrounded ETHD systems. The former has a limited space for the haptic rendering area, since the ETHD is attached to the ground and positioned by a robotic actuator; see Figure 3.1 for an illustration. Common configurations of grounded ETHDs are robotic arms [30, 31, 32] and fixed platforms [33, 34].

Ungrounded ETHDs, on the other hand, can freely move in the given workspace, enabling a large-scale haptic rendering area. Wheeled mobile robots [7, 10, 35] and UAVs [36, 37] are promising in enabling ungrounded ETHDs.



(a) The haptic surface display, the tracking system and the simulation environment are the main components of such systems. The area where the user and the ETHD module perform the task is called the workspace. [29]



(b) The rendering area encapsulates the possible locations of the end-effector in an encountered position to make a haptic interaction possible. The contact area is the space which can be interacted with during a specific time during the haptic rendering. [29]

Figure 3.1: Illustrations of typical ETHD systems and definitions according to Mercado et al. [29].

3.1.1 Ungrounded ETHDs

Since this thesis focuses on positioning a mobile haptic device, several ungrounded ETHDs with wheeled mobile robots have been covered in this section.

With the use of omnidirectional robots, Gomi et al. realized such an ungrounded ETHD system called InflatableBots [7]. They attached a shape-changing device to the mobile robot, which can be adapted during the haptic interaction. The idea is to inflate and deflate a vinyl tube with air to simulate different shapes, hence even continuous objects can be rendered accordingly (see Figure 3.2).

A user study was conducted to evaluate the effectiveness of InflatableBots. Soft and smooth objects can be rendered realistically, while objects with other materials feel unrealistic. The continuous haptic rendering proved to be somewhat possible. However, some users noticed a noticeable delay in the robot's movement and inflation or deflation.

Figure 3.2: Minimum and maximum height of the InflatableBots. [7]

Figure 3.3: Interaction examples of InflatableBots. (a) Stationary trees, (b) Multiple moving dogs, (c) Continuous stationary car, (d) Interactive windows (shape-changing), and (e) Drumming with drumsticks. [7]

Ikeda et al. [35] proposed a user-following ETHD called LoopBot, see Figure 3.4 and Figure 3.5. LoopBot can render continuous grounded objects such as walls or handrails. An omnidirectional robot equipped with a looped haptic prop makes this possible. To follow the user, a PID controller is used to drive the robot according to the user's position. The conducted user study found that users felt that the modelled handrail was realistic enough.
The positioning system of this work is simple and does not take unplanned user movement into consideration. The robot is moving either forward or backwards, depending on the HMD's location. Since the VR simulation is set up as a long and narrow corridor, this might be enough, but for more complex environments, a more sophisticated positioning system has to be developed.

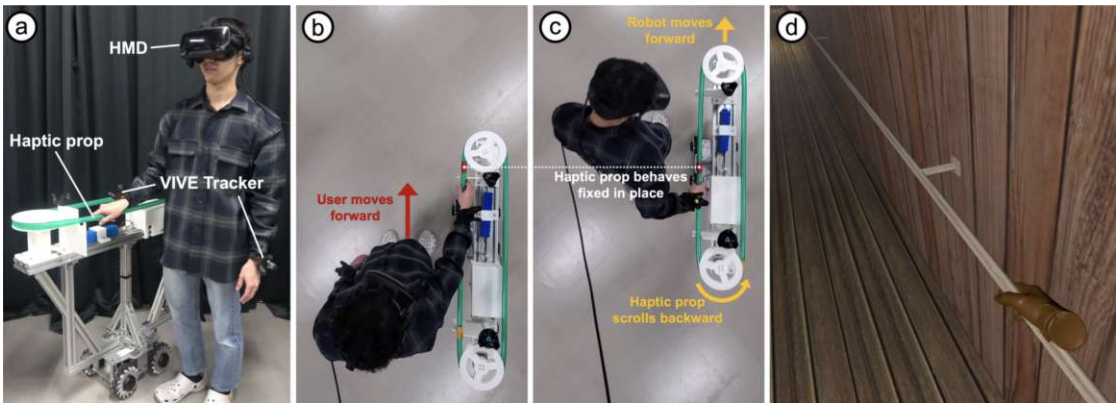


Figure 3.4: Overview of LoopBot. Loopbot acts as a handrail in a VR setting. [35]

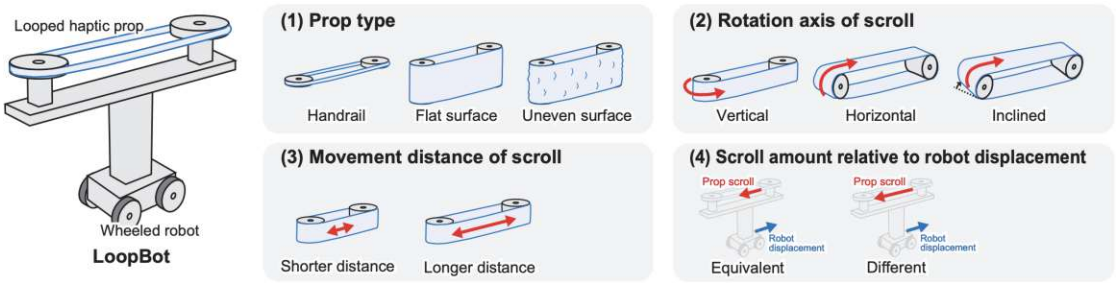


Figure 3.5: Various design ideas to render different objects for LoopBot. [35]

Commercial vacuum cleaner robots are often used to enable ungrounded ETHDs. ZoomWalls [38], RoomShift [39], and MoveVR [40] all use Roombas to move haptic props in a room-scale VR environment. MoveVR employs a single Roomba attached with a cardboard box to render multiple objects, or it can be attached with a line to simulate a pulling force. Similar to LoopBot, the positioning control is kept to a very simple, predefined move-to-goal method.
Both RoomShift and ZoomWalls use multiple Roombas to render the haptic props. In RoomShift, the robots are used to carry props to match the location of the virtual world.

Since the robots have to carry heavy props (desks, chairs), the speed of the robots is very slow (20 cm/s), which can result in a long haptic delay. The position control is done by a simple path planning algorithm. ZoomWalls uses a prediction algorithm combined with a path planning algorithm to optimally position multiple Roombas. With a slightly higher speed of 0.5 m/s, the system achieves short delays (sub-second) in simple environments. MoveVR and ZoomWalls would not be able to render strong force feedback to render a realistic wall, since the physical resistance needed requires heavier and stronger robots.

3.2 Human-Aware Position Control

3.2.1 Human-Aware Point-to-Point Navigation

Safe and socially acceptable position control in a dynamic HRI setting requires approaches that go beyond simple collision avoidance, taking human behaviour and preferences into account. A key example is the Proactive Social Motion Model proposed by Truong et al. [41]. This work explicitly models the implicit "social spaces" that humans naturally create and respect during interactions. Their model defines several types of zones that guide robot motion, as illustrated in Figure 3.6. These include personal spaces around individuals, group spaces that envelop a group of interacting people, and interaction spaces that form between a person and an object they are engaging with.

These dynamically-sized spaces are not treated as rigid obstacles but rather as "social cost" areas. During navigation, the robot's planning algorithm is penalized for creating paths that enter these zones. This "social cost" guides the robot to adhere to social customs—for example, by navigating around a conversation group rather than through it, or by avoiding cutting between a person and an object they are observing. The primary goal is to generate robot motion that feels more natural and less intrusive, enhancing both safety and user acceptance by avoiding physical and "social" collisions.

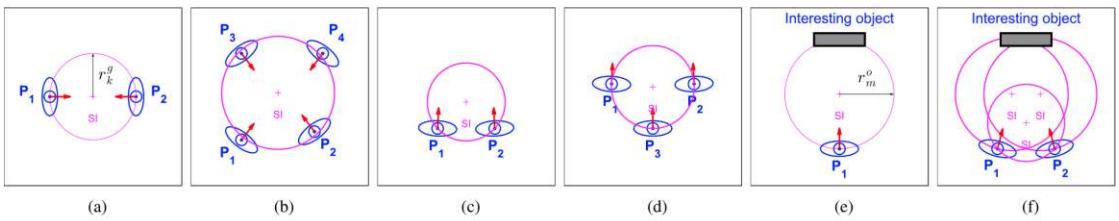


Figure 3.6: Illustrations of different social interaction spaces introduced by Truong et al. [41]. The "social cost" zone is in magenta, while users are illustrated as blue ellipses. (a)-(b) Group of standing people, (c)-(d) Group of moving people. (e)-(f) Human-object interaction spaces.

Visibility, social norms, and safety constraints guide these human-aware motion controllers to navigate robots safely around humans [42]. Sisbot et al. even consider not surprising humans in their motion algorithm. If a robot moves behind an obstacle in the field of view of the human, it moves farther behind the obstacle [43]. Human behaviour anticipation can also be used to plan and navigate a robot in a collaborative HRI setting, as shown

by [44]. Through sensor fusion of multiple RGB-D cameras, a 3D semantic scene model of the workspace is constructed. Then, safe and anticipatory controls are issued to the robot. See Figure 3.7 for more details.

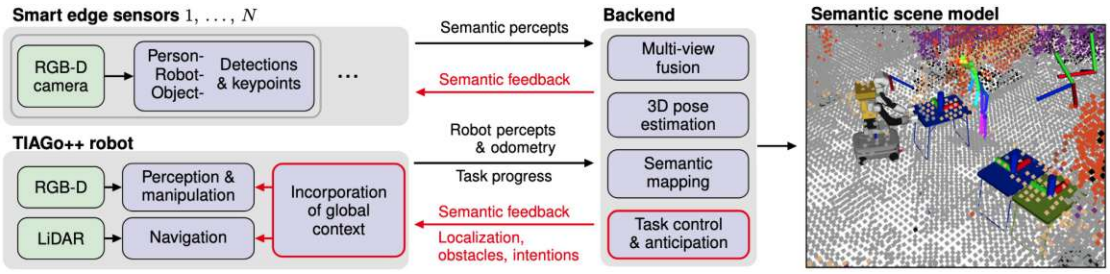


Figure 3.7: Architecture of the human behaviour anticipation system introduced by Bultmann et al. [44].

3.2.2 Human Following and Guidance

A more challenging problem is that of Human Following and Guidance [45]. Not only do you have to keep a safe distance, but you also have to be in proximity to the human during navigation. Ahn et al. [46] developed such a system by introducing an arc-shaped zone where the robot is allowed to position itself during the HRI. There are multiple potential goals in this zone where the robot can move to. In the ideal case, the robot is positioned to the front right of the user. Once obstacles block this position, the robot retracts to the back of the user in the arc-shaped zone.

Eirale et al. [47] developed a human-following system for assistance and monitoring in an indoor setting. They combine sensor data from RGB-D cameras and LiDAR to safely follow a human. The robot aims to orient itself facing the person while navigating, such that the camera can track the position of the user. This is made possible due to the omnidirectional robot base they employ (Figure 3.8).

3.3 DRL Position Control

DRL in mobile robotics has been proven to be a successful alternative to traditional methods in various use cases [11]. Human-following and guiding, as well as navigation in crowded environments, have seen improvements due to the data-driven approach of DRL in several works.

A hybrid system that uses both traditional planners and a DRL-based agent has been proposed by Eirale et al. [48], which builds upon previous work [47] of a human-following and monitoring task. The improved method uses a DRL agent to optimize the angular velocities commanded to the mobile robot, while the linear velocities are computed by traditional planners.

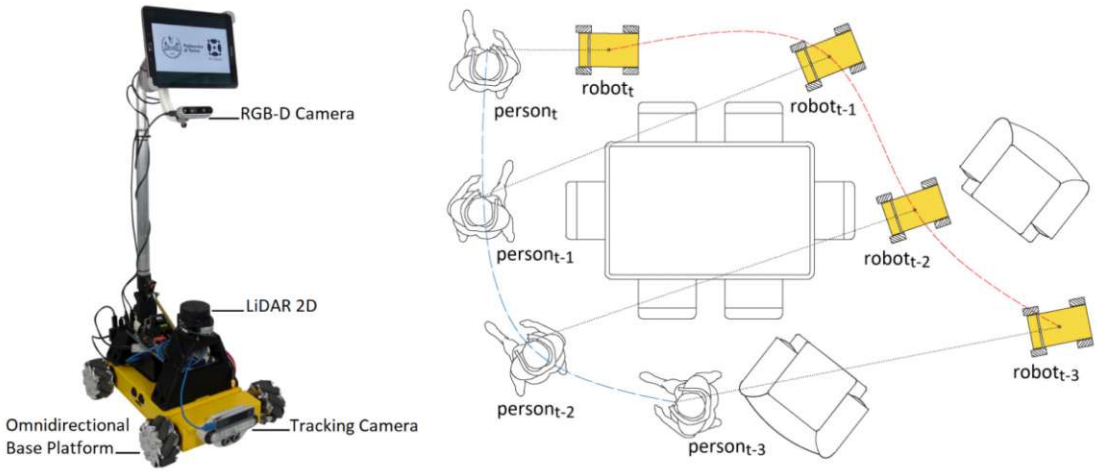


Figure 3.8: The omnidirectional robot platform Eirale et al. [47] proposed and visualization of the human-following task. The robot follows the person while avoiding obstacles.

Highly dynamic, crowded environments require safe and robust navigation. Point-to-point navigation in those crowded environments has seen recent improvements by utilizing DRL. Collision avoidance with DRL (CADRL), first proposed by Chen et al. [49], generates collision-free velocity commands in crowded environments. The follow-up works by Chen et al. [50] and [51] incorporate social norms (passing on the right) in their DRL frameworks to make the navigation conform to human motion. Xue et al. [52] model danger zones around humans that the DRL framework learns during training, resulting in crowd-aware navigation. Kästner et al. [20] take this a step further by incorporating

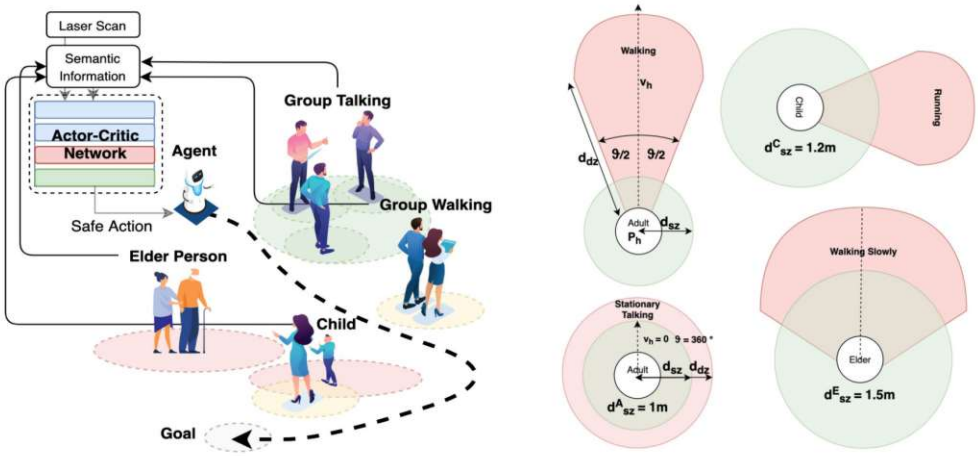


Figure 3.9: Overview of the semantic DRL framework by Kästner et al [20]. Dynamic danger zones around humans depend on the semantic data and the state of humans. These zones are based on the work of Samsani et al. [53].

semantic data of the environment into the DRL framework, such as social states of the humans (Figure 3.9). With this, they extend the capability to also follow and guide humans [54]. However, the maximum allowed velocity of the robot is restricted to 0.22 m/s, which raises questions about the feasibility of following a human at walking speed.

Previous works have shown that DRL is a proven solution to address various problems in human-aware position control. However, despite significant research in that area, important gaps remain in HRI contexts. In particular, human-aware position control scenarios involving more than one user immersed in VR and wearing HMDs are largely overlooked. These situations require special considerations: the mobile ETHD has to be in proximity to the users for responsiveness while also maintaining a safe distance to avoid collisions. To tackle this challenge, we propose a DRL-based approach that can handle responsive and safe position control of a mobile ETHD for up to two users.

CHAPTER 4

Methodological Approach

This chapter details the proposed Deep Reinforcement Learning framework for the autonomous positioning of the CoboDeck platform. The problem is first formalized as a Markov Decision Process, defining the state space, action space, and reward functions necessary for safe and responsive navigation. Then, the neural network architecture is presented, featuring a specialized collision avoidance module for processing high-dimensional LiDAR data. The implementation is then instantiated for two distinct operational contexts: a single-user (SU) scenario and a multi-user (MU) scenario. Following this, the training methodology is described, including the recording of a human motion dataset and the use of a procedurally generated simulation environment. Finally, the iterative development of the method is reviewed, justifying the design choices that led to the final architecture.

4.1 Problem formulation

The primary challenge is to develop a safe and responsive positioning system for CoboDeck, which has an omnidirectional mobile robot base. This thesis proposes a data-driven approach using DRL. We formulate this positioning task as an MDP, where an agent learns an optimal policy to control the robot's mobile base. The agent's goal is to minimize haptic interaction delays while ensuring user safety by avoiding collisions with dynamic users and static obstacles.

State space S The design of the state representation is a critical component for successful policy learning. The state space must encapsulate all necessary information for the agent to navigate safely and responsively. We hypothesize that a good representation is achieved by combining information about the robot's internal state, the relative state of the user(s), and the proximity to obstacles.

A robot-centric coordinate system is adopted for this representation (Figure 4.1). This approach ensures that the learned policy is generalizable and invariant to the robot's absolute position in the world coordinates. This invariance promotes robust behaviour, as the agent's actions are dependent on its immediate surroundings rather than its global pose.

The complete state s is defined as the concatenation of the robot state s_r, the state of N users $s_{u^{1:N}}$, and the goal state s_g:

$$s = [s_r, s_{u^{1:N}}, s_g], \tag{4.1}$$

The robot state s_r is defined as:

$$s_r = [v_r, q_r, L, a_{t-1:-2}], \tag{4.2}$$

where v_r and q_r represent the robot's 2D planar velocity (linear v_x, v_y; angular v_ω) and acceleration (linear q_x, q_y; angular q_ω), respectively. L is the vector of 2D LiDAR range measurements, and $a_{t-1:-2}$ incorporates the two previous actions to provide temporal context and aid in action smoothness.

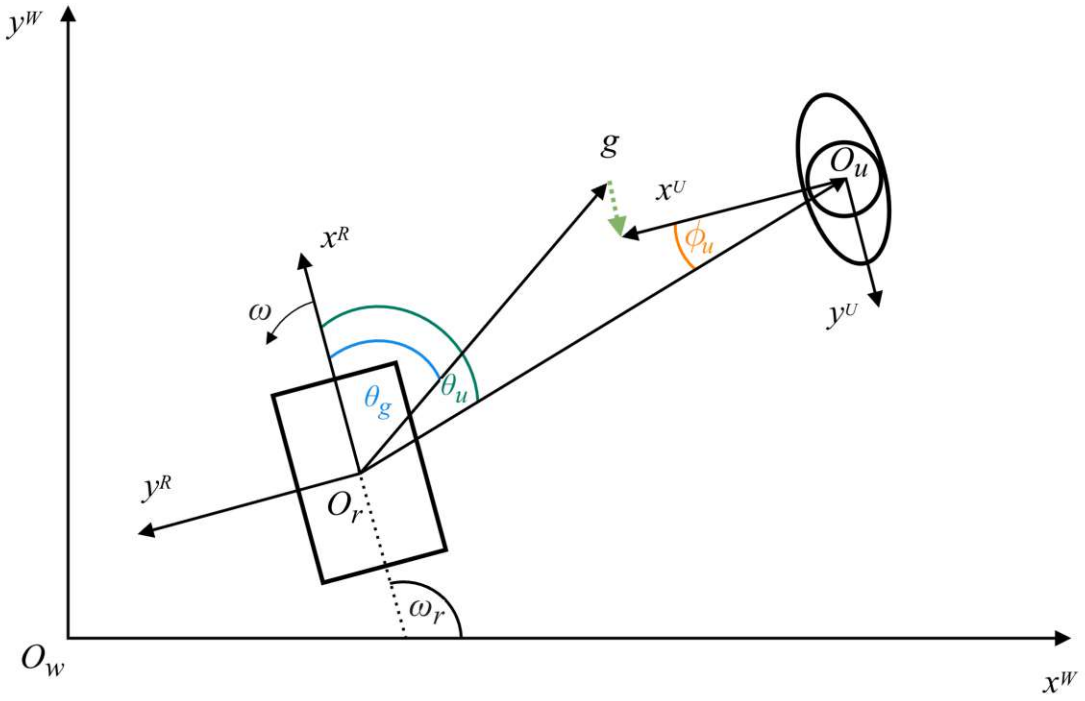


Figure 4.1: Coordinate system definitions of robot and one user. Coordinate axes are marked with superscript uppercase letters (R for robot, U for User, and W for world).

The state for each user n, s_{u^n}, is defined relative to the robot's coordinate frame:

$$s_{u^n} = [p_{u^n}, v_{u^n}, \phi_{u^n}, \theta_{u^n}, d_{u^n}] \tag{4.3}$$

This vector includes the relative position $p_{u^n} = O_{u^n} - O_r$, the velocities v_{u^n} (linear v_x, v_y; angular v_ω) relative to the robot coordinate system, the heading from the robot to the user θ_{u^n}, the heading from the user to the robot ϕ_{u^n}, and the 2D Euclidean distance between user and robot d_{u^n}.

Finally, the goal state s_g provides the agent with its objective:

$$s_g = [p_g, d_g, \theta_g, \Delta\omega_g] \tag{4.4}$$

This includes the relative position to the goal $p_g = g - O_r$, the Euclidean distance between goal and robot d_g, the heading to the goal θ_g, and the yaw error $\Delta\omega_g$ (the difference between the goal yaw and robot yaw).

Action space A The action space A is defined as a continuous vector of velocity commands in the robot's local coordinate system. Specifically, the action a_t at is defined as:

$$a_t = [v_x, v_y, v_\omega], \text{where } v_x, v_y, v_\omega \in [-1.0, 1.0] \tag{4.5}$$

Reward function R A carefully engineered reward function is essential to guide the agent toward the desired complex behaviour. Relying solely on a sparse terminal reward (e.g., upon reaching the goal) presents a significant credit assignment problem [17], which can lead to suboptimal or failed convergence. We therefore implement a combination of a sparse and dense reward shaping strategy, providing intermediate feedback at each timestep and a sparse reward upon reaching the goal.
Responsiveness: The primary objective, responsiveness, is encouraged by rewarding the agent for progress toward the goal pose. We define dense positive reward components r_d and r_ω as functions of the decreasing Euclidean distance d_g and the decreasing absolute yaw error $|\Delta\omega_g|$. This incentivizes the agent to continuously minimize its spatial and rotational error relative to the goal. The rotational error should only be a factor if the robot is close enough to the goal, not to limit the rotational movements of the robot far away from the goal. The sparse positive reward r_g incentivizes the agent to be precise near the goal location. The agent only receives this reward if it stays at the goal location for at least 0.1 s within a distance error $\epsilon_d = 0.01$ m and a rotational error $\epsilon_\omega = 0.0087$ rad (≈ 0.5 deg). This incentivizes precise and stable movement near the goal pose. If the state is currently unsafe s_{unsafe} (robot is in critical distance or obstacle inflation layer), then no positive reward is returned. This hinders unsafe movement near obstacles or users if the goal pose is inside the obstacle inflation layer or the critical distance of a user.

$$r_d = \begin{cases} 0, & \text{if } s_{\text{unsafe}} \text{ and } 3 - d_g > 0 \\ 3 - d_g, & \text{otherwise} \end{cases} \tag{4.6}$$

$$r_g = \begin{cases} 0, & \text{if } s_{\text{unsafe}} \\ 1, & \text{if } d_g < \epsilon_d \text{ and } |\Delta\omega_g| < \epsilon_\omega \text{ and } t_g \geq 0.1 \end{cases} \tag{4.7}$$

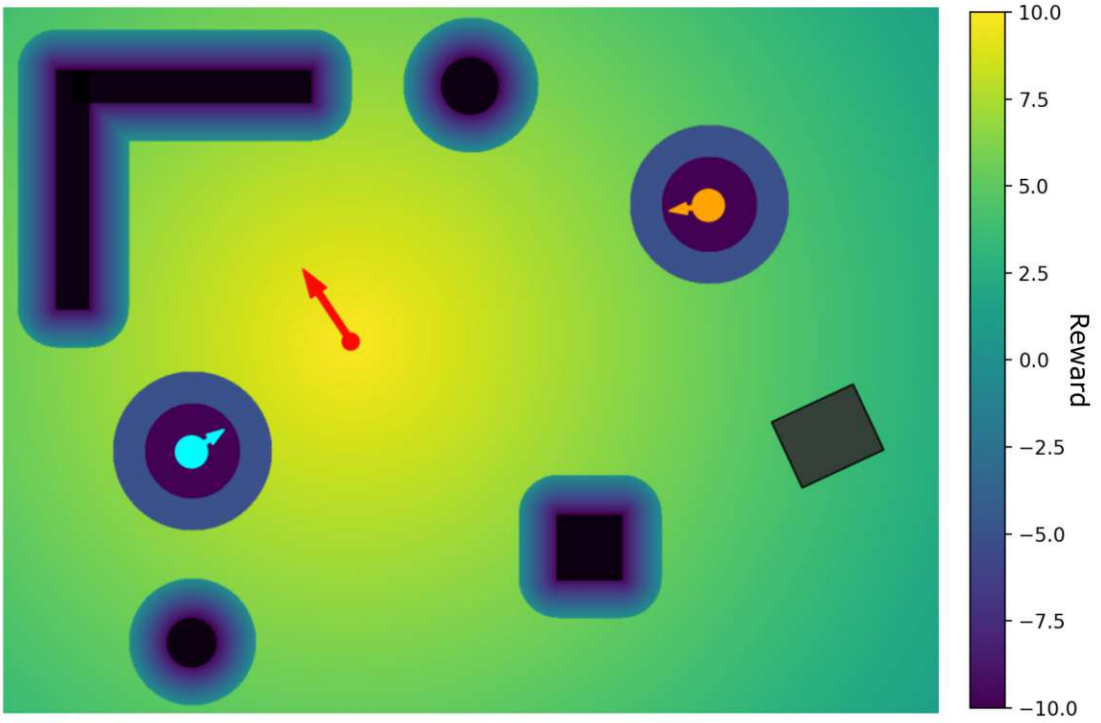


Figure 4.2: Visualization of the reward function. The reward is the highest near the goal pose (red arrow). The inflation layer around static obstacles and the critical and collision distance around the users (circles with view vector) have negative rewards. The robot is represented by the grey rectangle. Reward magnitudes are scaled for visual clarity and do not reflect the actual values.

$$r_\omega = \begin{cases} 0, & \text{if } s_{\text{unsafe}} \\ (\exp(-3|\Delta\omega_g|) \cdot \max(1 - d_g, 0)), & \text{otherwise} \end{cases} \tag{4.8}$$

Safety: The safety constraint is enforced through significant negative penalties, as visualized in Figure 4.2. These penalties are structured to discourage unsafe states. A high-cost penalty p_{u^n} is applied for collisions with a user or static obstacles p_o. User collisions are defined as a breach of a collision distance d_{coll} threshold. Static obstacle collisions are registered if any of the LiDAR ranges L is smaller than its corresponding robot boundary distance b_r, see Figure 4.3. Furthermore, to encourage anticipatory avoidance, a smaller critical distance penalty p_{c^n} is applied in a buffer zone around users, and an inflation layer penalty p_i is applied near static obstacles. This penalty structure teaches the agent to maintain safe operating distances.

Obstacle information is sourced from two modalities. The 2D LiDAR ranges L provide a general-purpose representation of static and dynamic obstacles. This is supplemented by high-precision user tracking data s_{u^n} inherent to the VR system. Utilizing both sources

creates a robust and redundant safety system, where the user-specific tracking data allows for a more precise and reliable safety margin around dynamic users.

$$p_o = \mathbb{1}\{\text{collision}\} \tag{4.9}$$

$$p_{u^n} = \mathbb{1}\{d_{u^n} < d_{\text{coll}}\} \tag{4.10}$$

$$p_{c^n} = \mathbb{1}\{d_{u^n} < d_{\text{crit}}\} \tag{4.11}$$

$$p_i = \sum_{l \in L} \max\left(\frac{d_i - f(l)}{d_i}, 0\right) \tag{4.12}$$

$$f(l) = \begin{cases} (l - b_r) \cdot |\sin(\text{angle}(l))|, & \text{if } |\sin(\text{angle}(l))| \cdot \text{r} \geq \frac{\text{w}}{2} \\ (l - b_r) \cdot |\cos(\text{angle}(l))|, & \text{otherwise} \end{cases}, \tag{4.13}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function which is 1 if the condition inside the curly brackets is true and 0 otherwise. r is the robot radius and w is the width of the robot.

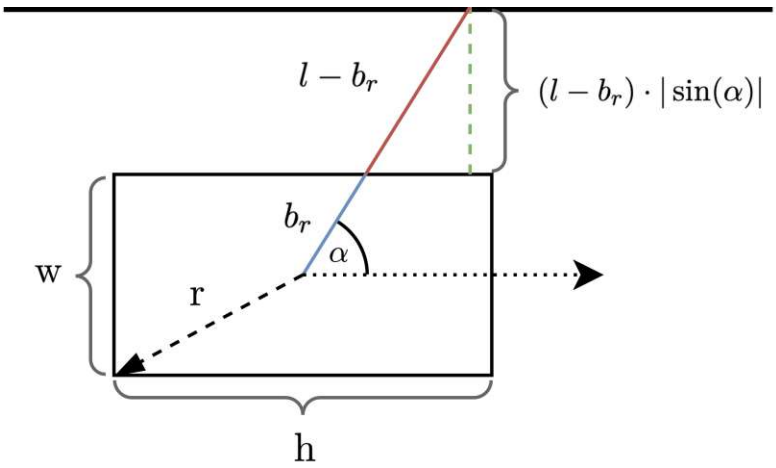


Figure 4.3: Illustration of the robot boundary aware inflation layer calculation used in Equation 4.13. It shows one LiDAR ray l hitting an obstacle and the calculation of the perpendicular distance to the robot boundary. The width w, the length h, and the radius r of the robot have to be defined beforehand for a correct calculation.

The inflation layer penalty p_i is implemented to maintain a safe standoff distance from static obstacles by penalizing proximity relative to the robot's rectangular boundary rather than its center. This calculation must be derived solely from the 2D LiDAR ranges L. A naive approach, such as taking the direct difference $l - b_r$ between a LiDAR range l and its corresponding robot boundary distance b_r, would inaccurately represent the true

perpendicular distance for a non-circular shape, as illustrated in Figure 4.3. To address this geometric complexity, an approximation of the true distance is calculated using the function $f(l)$ detailed in Equation 4.13. This function scales the excess range $l - b_r$ by the absolute sine or cosine of the ray's angle, corresponding to the robot's sides (sine) or its front/back (cosine). This methodology effectively creates a dynamic, boundary-aware inflation layer around obstacles. A significant advantage of this method is its scalability and generality: it provides robust obstacle avoidance without requiring prior knowledge of the environment's layout, relying only on the robot's known physical dimensions.

Motion Quality and Efficiency: Finally, secondary objectives are included to regularize the policy. To promote motion smoothness, the penalty p_a penalizes high acceleration and jerk (changes in acceleration) between subsequent actions. This is critical for user comfort and perceived safety, and it reduces mechanical strain on the robot hardware. Additionally, a small penalty p_v on the magnitude of the robot velocity is included to optimize for energy efficiency. A final regularization term, p_q, penalizes the acceleration of the robot, further encouraging smooth motion.

$$p_a = ||a_t - a_{t-1}||^2 + ||a_t - 2a_{t-1} + a_{t-2}||^2 \tag{4.14}$$

$$p_v = ||v_r||^2 \tag{4.15}$$

$$p_q = ||q_r||^2 \tag{4.16}$$

$|| \cdot ||$ is the L2 norm.

The final reward function for N users is:

$$R_t = r_d + r_g + r_\omega + p_o + p_{u^{1:N}} + p_{c^{1:N}} + p_i + p_a + p_v + p_q \tag{4.17}$$

Each term has an associated weight factor to it, which is omitted for better readability. The rewards for each scenario are discussed in detail in the following subsections, and the exact weights can be seen in Table 4.1 and Table 4.3.

4.1.1 Network Architecture

To solve the MDP formulated in Section 4.1, we design an Actor-Critic network architecture, which is trained using the PPO algorithm. The Actor-Critic framework consists of two distinct networks that share a common feature-extraction body to improve training efficiency and representation learning Figure 4.4.

The actor network (policy) is responsible for action selection given state s. PPO learns a stochastic policy $\pi(a|s)$ from which we sample our action $a_t = [v_x, v_y, v_\omega]$. The critic network (state-value function) is responsible for evaluation. It learns the state-value function $V(s)$, which estimates the return from state s. This value is then used to guide the actor into states that produce better returns. The shared network body has two components that fulfil different roles:

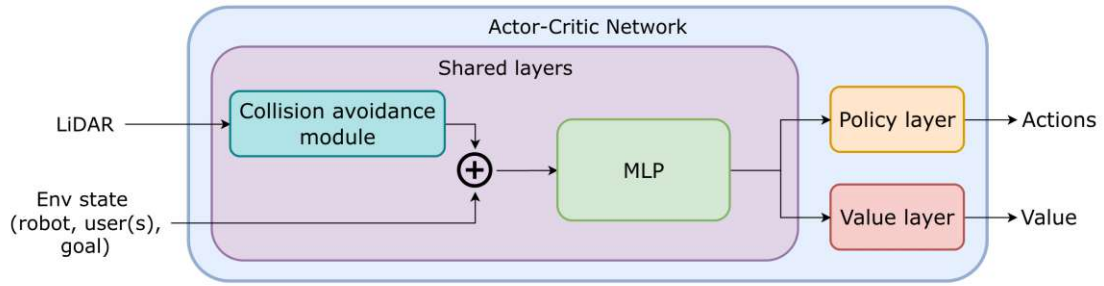


Figure 4.4: The shared Actor-Critic network architecture we use to train both single-user (SU) and multi-user (MU) navigation policies. First, the state gets preprocessed in the shared layers (same parameters), which consists of a collision avoidance module, responsible for LiDAR data processing, and a fully-connected MLP. The final policy layer and value layer have separate parameters to learn to output actions and values.

1. **Collision avoidance module:** A primary challenge is processing the high-dimensional (720-ray) LiDAR vector L. A simple MLP would struggle with this high-dimensional, spatially-correlated data and would be parameter-inefficient. Therefore, we implement a specialized collision avoidance module. This module uses a 1-Dimensional Convolutional Neural Network (1D CNN). The 1D CNN is highly effective as its filters can slide across the LiDAR scan to detect local spatial patterns. This module efficiently compresses the 720-ray input into a low-dimensional latent feature vector that encapsulates the robot's immediate surroundings.

2. **MLP**: The latent feature vector of the collision avoidance module gets concatenated with the remaining low-dimensional state vector, including the robot state s_r excluding L, the user states $s_{u^{1:N}}$, and the goal state s_g. This vector is then fed into a standard MLP. This network serves as a sensor fusion mechanism, integrating the condensed spatial features with precise kinematic states.

After processing the state in the shared layers, the final policy and value layers are then responsible for outputting action and value, respectively.

4.2 Scenario-Specific Implementations

This section details the concrete application of the proposed DRL framework across two distinct operational environments. We first address the single-user (SU) scenario, defining the specific MDP components and relevant adaptations to them. Subsequently, we extend this formulation to the more complex multi-user (MU) scenario. To manage the increased state space and the stochasticity of multiple user intents, we introduce a hierarchical policy decomposition consisting of GuideRL (for high-level strategic positioning) and NavRL (for low-level collision-free navigation).

4.2.1 Single User (SU) Scenario

In the single-user (SU) scenario, a user navigates freely within the virtual environment. Haptic requests are issued at random. Consequently, the system's responsiveness is directly determined by the robot's proximity to the Haptic Interaction Pose (HIP) g at the moment of the request (see Figure 4.5). To minimize this latency, a DRL agent is trained to maintain a safe and optimal position for the robot during the pre-haptic phase.

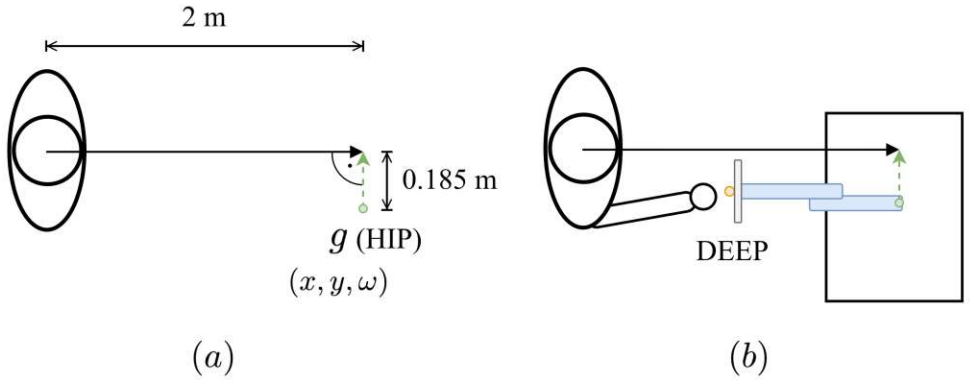


Figure 4.5: Visual definition of the target configurations for haptic interaction. (a) The Haptic Interaction Pose (HIP) g is defined with a lateral offset to facilitate natural reaching. (b) Upon reaching the HIP, the robot arm extends to the Desired End-Effector Pose (DEEP).

The MDP for the SU scenario is a direct instantiation of the general N-user formulation from Section 4.1, specified for $N = 1$.

State space S The state $s = [s_r, s_u^1, s_g]$ includes the robot state, the state of the user, and the state of the goal pose, which is the HIP of the user.

To enhance policy generalization and robustness against sensor imperfections, we introduced noise to the 2D LiDAR readings. The agent was trained using these noisy LiDAR ranges $\tilde{L}$ instead of the ground-truth ranges L. This noise model consists of additive Gaussian noise $\mathcal{N}(0, 0.03)$ and a 0.5% dropout probability per ray, where a dropout reverts the reading to the sensor's maximum range. Furthermore, the noisy LiDAR ranges $\tilde{L}$ are clamped to a max range of 3 m, reducing unnecessary information about obstacles far away. Using the clamped noisy LiDAR ranges $\tilde{L}$ in the state vector during training prevents the policy from overfitting to the perfect, noise-free information available only in simulation. To stabilize the robot's behaviour near the goal, we introduced a "tolerance zone". We found that when the goal distance d_g was less than 0.2m, the heading θ_g could "wrap around" from π to $-\pi$, causing erratic angular velocities. We resolved this by setting $\theta_g = 0$ inside this 0.2m radius, effectively telling the policy that the heading is no longer a relevant objective.

Action space A The action space in the SU scenario is the continuous velocity command $\tilde{a_t} = [\tilde{v_x}, \tilde{v_y}, \tilde{v_\omega}]$. This interpolated action is defined as follows:

$$\tilde{a_t} = w_1 a_t + w_2 a_{t-1} + w_3 a_{t-2}, \text{where } w_1, w_2, w_3 \in [0, 1] \text{ and } \sum_{k=1}^{3} w_k = 1 \tag{4.18}$$

This proved to be a necessary step in drastically smoothing the movement of the robot. The interpolation weights are $w_1 = 0.5, w_2 = 0.3, w_3 = 0.2$. To enhance stability and respect hardware limitations, the action $\tilde{a_t}$ is not commanded directly to the robot. Instead, it is passed through a dead-zone filter to produce the final commanded action $\hat{a_t} = [\hat{v_x}, \hat{v_y}, \hat{v_\omega}]$. This filter zeroes out near-zero velocities that are likely policy noise or below the motor's effective threshold. We define a linear velocity threshold $\tau_v = 0.5$ cm/s (0.005 m/s) and an angular velocity threshold $\tau_\omega = 0.25$ deg/s (≈ 0.00436 rad/s).

The final action components are determined as follows:

$$\hat{v_x} = \begin{cases} 0, & \text{if } |\tilde{v_x}| < \tau_v \\ \tilde{v_x}, & \text{otherwise} \end{cases} \tag{4.19}$$

$$\hat{v_y} = \begin{cases} 0, & \text{if } |\tilde{v_y}| < \tau_v \\ \tilde{v_y}, & \text{otherwise} \end{cases} \tag{4.20}$$

$$\hat{v_\omega} = \begin{cases} 0, & \text{if } |\tilde{v_\omega}| < \tau_\omega \\ \tilde{v_\omega}, & \text{otherwise} \end{cases} \tag{4.21}$$

Since we applied a dead-zone to the actions, we also apply it to the goal state itself: if the relative position components p_g^x, p_g^y were smaller than the linear velocity threshold τ_v, or if the yaw error $|\Delta\omega_g|$ was smaller than the angular threshold τ_ω, they were respectively set to zero.

Reward function R The reward function for the SU case $N = 1$ is defined by

$$R_t = r_d + r_g + r_\omega + p_o + p_{u^1} + p_{c^1} + p_i + p_a + p_v + p_q \tag{4.22}$$

The associated weight factors can be seen in Table 4.1. The safety distances aimed to establish a 1 m safety buffer between the robot and the middle of the user. The robot radius is 0.77 m, so we defined the safety distances $d_{\text{coll}} = 0.77 + 0.6$, $d_{\text{crit}} = d_{\text{coll}} + 0.4$. The inflation layer distance $d_i = 0.5$ m gives enough space between the robot boundary and obstacles.

Table 4.1: Reward weights for the single-user policy.

Term	Explanation	Weight
r_d	distance reward	1.0
r_g	at goal reward	50.0
r_ω	yaw reward	8.0
p_w	obstacle collision penalty	-900
p_u	user collision penalty	-1000
p_c	critical distance penalty	-20.0
p_i	inflation layer penalty	-0.5
p_a	action smoothness penalty	-0.25
p_v	velocity penalty	-2.0
p_q	acceleration smoothness penalty	-0.0015

Network architecture

The collision avoidance module is responsible for encoding the high-dimensional 720-ray LiDAR input. This module is a 1D CNN with four convolutional layers (with kernels 8x1, 5x1, 3x1, and 3x1 and strides 4, 3, 1, and 1, respectively) that progressively process the spatial information from the scan. After a flattening operation, the final fully-connected

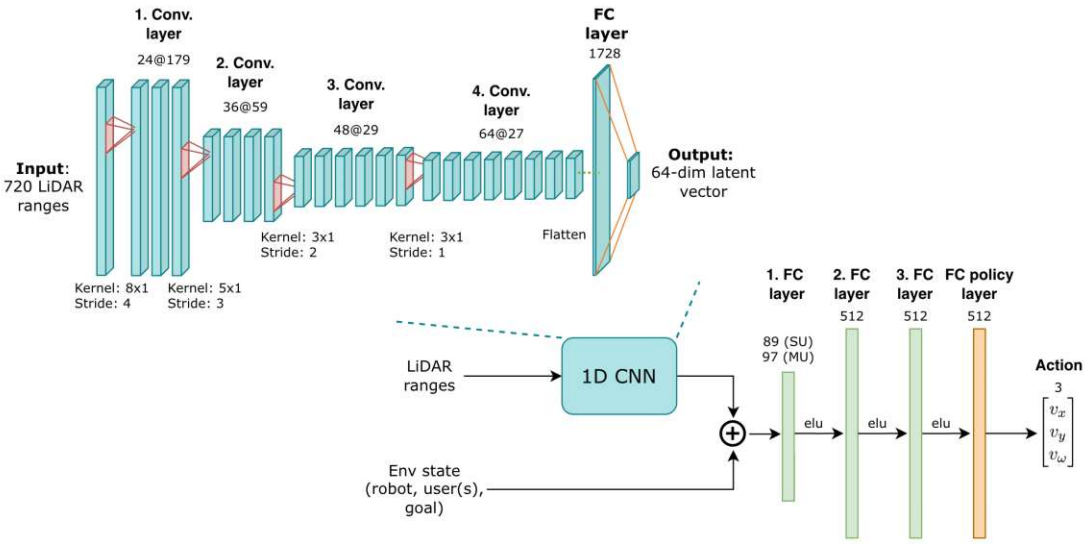


Figure 4.6: The policy network architecture with a detailed view of the collision avoidance module. The 1D CNN processes the LiDAR ranges and outputs a latent embedding for further processing in the Fully-Connected (FC) layers. Finally, the policy layer outputs the linear velocities in the x,y directions and the angular velocity.

layer embeds the LiDAR data into a 64-dimensional latent vector. This LiDAR embedding is then combined with the environment state vector (25-dimensional). This unified feature

vector is passed through an MLP with three hidden layers. Each of these FC layers consists of 512 neurons and uses the ELU activation function. Finally, an FC policy layer outputs the 3-dimensional continuous action vector, see Figure 4.6. The total learnable paramters for the policy network are 702674. Andrychowicz et al. investigated what kind of design decisions matter in training RL agents with PPO [55]. This guided us in designing our policy and value network.

4.2.2 Multi User (MU) Scenario

The MU scenario introduces a significant challenge: the mETHD must simultaneously accommodate two VR users, substantially increasing the state space complexity compared to the SU case. The robot must continuously predict which user is more likely to issue a haptic request and position itself accordingly. Initial attempts to train a single, end-to-end policy for this task resulted in unstable learning behaviour and non-convergence. To mitigate this complexity, we decomposed the task into two independent, hierarchical policies: GuideRL and NavRL. This decomposition, illustrated in Figure 4.7, separates the strategic decision-making from the navigation. GuideRL determines a dynamic target pose g based on the user states, which is then passed as the objective to the NavRL policy.

GuideRL

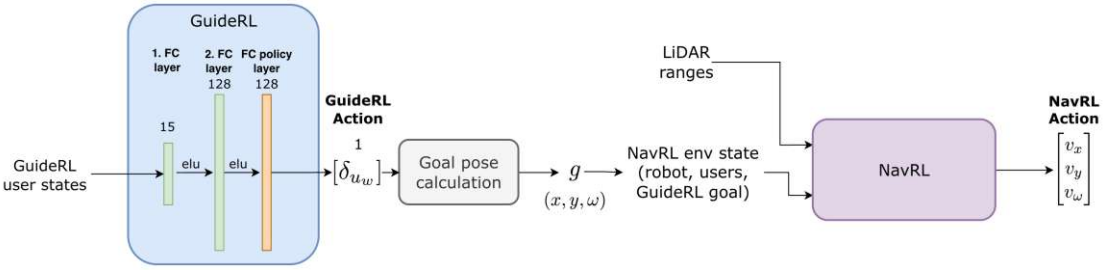


Figure 4.7: The two policies of GuideRL and NavRL working together to solve the pre-haptic positioning problem. GuideRL first decides the goal pose g, which is then fed into the NavRL policy. The NavRL policy has the network structure specified in Figure 4.6.

GuideRL is responsible for the high-level strategic task: predicting which user is more likely to issue the next haptic request. It accomplishes this by learning a policy that adjusts a user weight $u_w \in [-1, 1]$. This weight parameterizes the robot's target position $g_p = (x, y)$ on the line segment connecting the HIPs of the two users, as depicted in Figure 4.8. A value of $u_w = -1$ corresponds to the HIP of user 1, $u_w = 1$ to the HIP of user 2, and $u_w = 0$ to the midpoint. The complete goal pose $g = (g_p, \omega)$ required by NavRL also includes a target yaw ω. This yaw is determined as a function of u_w,

interpolating between the ideal yaw ω_1 (for user 1) and ω_2 (for user 2):

$$\omega = \begin{cases} \omega_1, \text{if } u_w < -0.25 \\ \omega_2, \text{if } u_w > 0.25 \\ \Psi(\omega_1, \omega_2), \text{otherwise} \end{cases} \qquad \Psi(\theta_1, \theta_2) = \arctan\left(\frac{\sin(\theta_1) + \cos(\theta_2)}{\cos(\theta_1) + \sin(\theta_2)}\right), \tag{4.23}$$

where ω_1 and ω_2 are the predefined target yaws for User 1 and User 2, respectively (as defined in Figure 4.5). The function Ψ provides a stable interpolation between the two angles when the goal is in the central region.

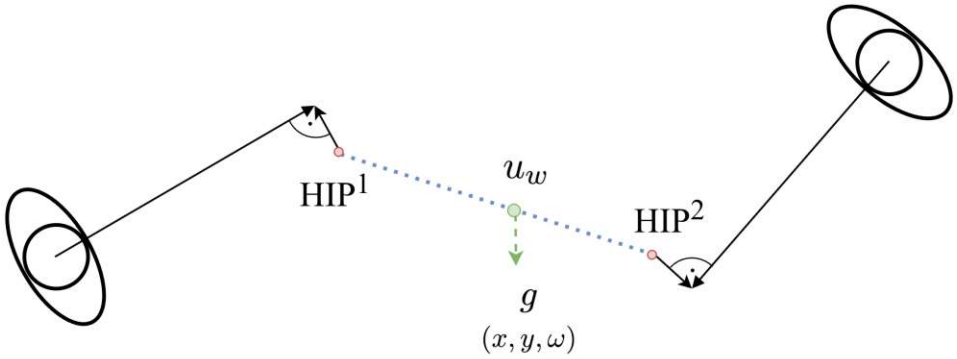


Figure 4.8: Illustration of the goal pose (green dot, and arrow visualizing rotation) calculation based on the user weight u_w and the HIPs (red dots, and arrow).

State space S^G The state space S^G of GuideRL is defined as:

$$S^G = [v_{u^1}, v_{u^2}, q_{u^1}, q_{u^2}, u_w, a_{t-1:-2}] \tag{4.24}$$

This vector includes the velocities v_{u^n} (linear v_x, v_y; angular v_ω) and accelerations q_{u^n} (linear q_x, q_y; angular q_ω) for each user $n \in \{1, 2\}$ in their respective coordinates systems, the current user weight u_w, and the two previous actions $a_{t-1:-2}$ to provide temporal context for better action smoothness.

Action space A^G The policy outputs a continuous action $a_t \in \mathbb{R}^1$, representing a small change δ_{u_w} to the user weight. This action is clipped to ensure smooth transitions of the goal pose:

$$a_t = [\delta_{u_w}] \in [-0.05, 0.05] \tag{4.25}$$

The user weight is then updated at each timestep as $u_w \leftarrow \text{clip}(u_w + \delta_{u_w}, -1, 1)$.

Reward function R^G The reward function R^G is designed to train GuideRL to predict the user most likely to issue the next haptic request. This is primarily achieved with a sparse terminal reward r_c, which is given only at the start of a haptic request. To provide denser feedback and stabilize learning, a continuous reward r_m is included.

$$r_c = \mathbb{1}\{\text{user 1 first}\} \cdot -u_w + \mathbb{1}\{\text{user 2 first}\} \cdot u_w \tag{4.26}$$

$$r_m = \max(1 - 4 \cdot |u_w|, 0) \tag{4.27}$$

The terminal reward r_c is applied when a haptic request is made. This formulation directly rewards the policy based on its final u_w. For example, if user 1 issues the request, $r_c = -u_w$. A u_w value close to -1 (correctly predicting user 1) yields a high positive reward, while a value close to 1 (incorrectly predicting user 2) results in a significant penalty. The mid-point reward r_m encourages a conservative, central position ($u_w \approx 0$) when no user has shown clear intent. This component was found to be critical to prevent the policy from oscillating between the two users, an issue observed in early training iterations.

The reward function of GuideRL is defined by:

$$R_t^G = r_c + r_m + p_a \tag{4.28}$$

Similar to the SU scenario, p_a is an action smoothness penalty defined the same as in Equation 4.14. The associated weight factors can be seen in Table 4.2.

Table 4.2: Reward weights for the GuideRL policy.

Term	Explanation	Weight
r_c	chose the correct user reward	20.0
r_m	mid-point reward	5.0
p_a	action smoothness penalty	-0.05

Network architecture

The network architecture for GuideRL is relatively straightforward. As the state space is a low-dimensional vector (15-dimensional), the network is designed as an MLP with two hidden layers. Each of these FC layers consists of 128 neurons and utilizes the ELU activation function. Finally, an FC policy layer outputs the 1-dimensional continuous action vector, see Figure 4.7. The total number of learnable parameters for the policy network is 18690.

NavRL

NavRL is the navigation policy, trained subsequently to GuideRL. It is responsible for navigating the robot safely to the goal pose g that is supplied by the GuideRL policy.

The MDP for NavRL is a direct instantiation of the general N-user formulation from Section 4.1, specified for $N = 2$.

State space S The state $s = [s_r, s_{u^1}, s_{u^2}, s_g]$ includes the robot state, the state of both users, and the state of the goal pose, which is defined by GuideRL. All state processing and augmentations detailed in the SU scenario, such as the use of clamped noisy LiDAR readings $\tilde{L}$ and the goal-state tolerance zone, are applied here as well.

Action space A The action space for NavRL is identical to the SU scenario defined in Subsection 4.2.1.

Reward function R The reward function for the SU case $N = 2$ is defined by

$$R_t = r_d + r_g + r_\omega + p_o + p_{u^1} + p_{c^1} + p_{u^2} + p_{c^2} + p_i + p_a + p_v + p_q \tag{4.29}$$

The associated weight factors can be seen in Table 4.3. Similarly to the SU scenario, the safety distance variables are $d_{\text{coll}} = 1.37$, $d_{\text{crit}} = 1.77$, and the inflation layer distance is $d_i = 0.5$ m.

Table 4.3: Reward weights for the NavRL policy.

Term	Explanation	Weight
r_d	distance reward	1.0
r_g	at goal reward	50.0
r_ω	yaw reward	8.0
p_w	obstacle collision penalty	-900
p_u	user collision penalty	-1000
p_c	critical distance penalty	-20.0
p_i	inflation layer penalty	-0.5
p_a	action smoothness penalty	-0.25
p_v	velocity penalty	-2.0
p_q	acceleration smoothness penalty	-0.0000075

4.2.3 Network Architecture

The network architecture of NavRL is identical to the SU scenario network. Only the environment state vector has a different dimension (33-dimensional instead of 25-dimensional), see Figure 4.6. The total learnable parameters for the policy network are 706770. Including GuidRL, the total number of learnable parameters of the MU scenario network architecture is 725460.

4.3 Implementation and Training

4.3.1 User Motion Dataset

To train policies that are robust to realistic human behaviour, we captured a human motion dataset. The dataset was recorded using a Qualisys optical motion capture system, which provides sub-millimeter tracking accuracy [56]. As illustrated in Figure 4.9, reflective markers were placed on the participant's head (to simulate HMD tracking) and wrist (to simulate hand-held controller tracking). A total of 442 trajectories were recorded with three expert participants. To enhance the robustness and generalization of the resulting policies, the dataset includes a diverse range of motions. Participants performed recordings both with and without wearing an HMD. Movements included fast, slow, and backwards walking, as well as side-stepping. Each trajectory was designed to simulate a complete "walk-and-interact" sequence. A participant would walk freely within the lab, stop at a random location, and then raise their arm to simulate touching a virtual object. The motion data was captured as 6-DOF poses (position and orientation)

(a) (b)

Figure 4.9: Recording of the human motion dataset with an optical tracking system. (*a*) Markers were placed on top of the head to simulate HMD tracking and on the wrist to simulate controller tracking. (*b*) Optical tracking camera setup in the lab.

for both the head and wrist constellations at a rate of 100 Hz. A boolean value was simultaneously recorded to provide a precise timestamp for the haptic interaction trigger. All data was logged and stored in ROS bag files.

The average trajectory duration is 21.9 s (SD=8.57) and covered 11.5 m (SD=6.63), while the average user speed is 0.5 m/s (SD=0.38). The median is 0.6 m/s, while a maximum speed of 1.9 m/s, excluding outliers likely due to tracking errors (> 2 m/s) was recorded Figure 4.10. The raw 6-DOF poses were used to derive the state variables required by

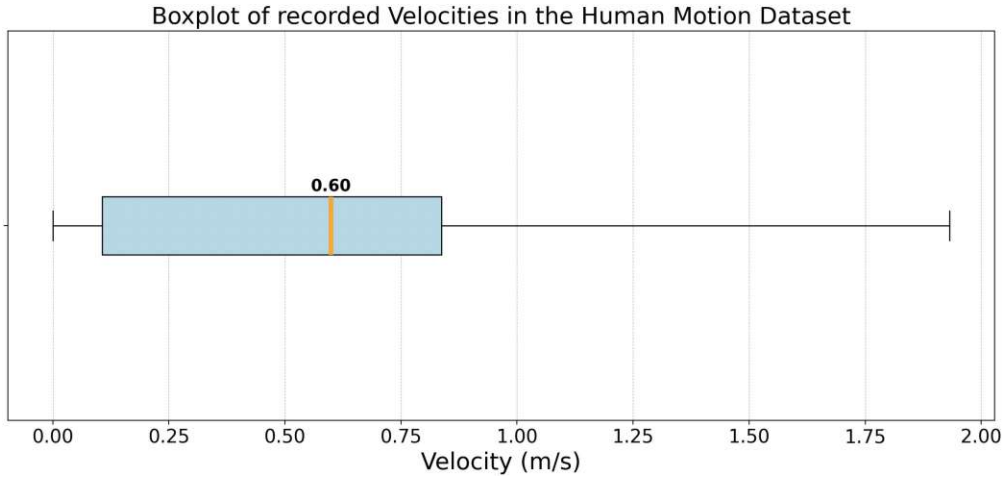


Figure 4.10: Boxplot of the recorded velocities in the human motion dataset. Outliers that were likely due to tracking errors (> 2 m/s) are not included in this data. The orange line represents the median.

the policies, specifically the 2D poses x, y, ω and subsequently the linear velocities v_x, v_y, angular velocity v_ω, and their corresponding accelerations q_x, q_y, q_ω.

The complete dataset was segmented into a training set (342 trajectories $\approx 80\%$)and a test set (100 trajectories $\approx 20\%$). To ensure a robust evaluation of generalization, the test set consists of consecutive trajectories that were never seen by the agent during training.

4.3.2 Training Environment

To train the DRL agents, we developed a simulation environment using NVIDIA Isaac Sim (v4.5) [14], a photorealistic, physics-based simulator. The training itself was managed by Isaac Lab (v2.1) [15], a modular RL framework. The primary advantage of this stack is its GPU-accelerated physics engine, which allows massive parallelization. This capability allowed us to train policies across thousands of concurrent environments, dramatically accelerating data collection and policy convergence.

We created a digital twin of our 11m x 13m laboratory. This environment allows the agent to train in a precise replica of the real-world deployment space, learning to navigate its specific layout and static obstacles. However, to prevent overfitting to a single layout and to promote generalization, a procedurally generated environment was created using Blender [57]. This environment, shown in Figure 4.12, consists of 32 unique floors. Each floor features a different, randomly generated configuration of obstacles (cylinders, triangles, and quads) of varying scales. A technical constraint of Isaac Lab is that its LiDAR simulation can only perform ray-casting against a single static mesh. To work around this while still providing diverse training scenarios, the 32 tower floors were combined into a single, large mesh. At the start of each episode, the robot and users are spawned on a randomly selected floor, forcing the agent to learn a robust and

generalizable navigation policy. A 3D model of the robot, provided by the manufacturer,

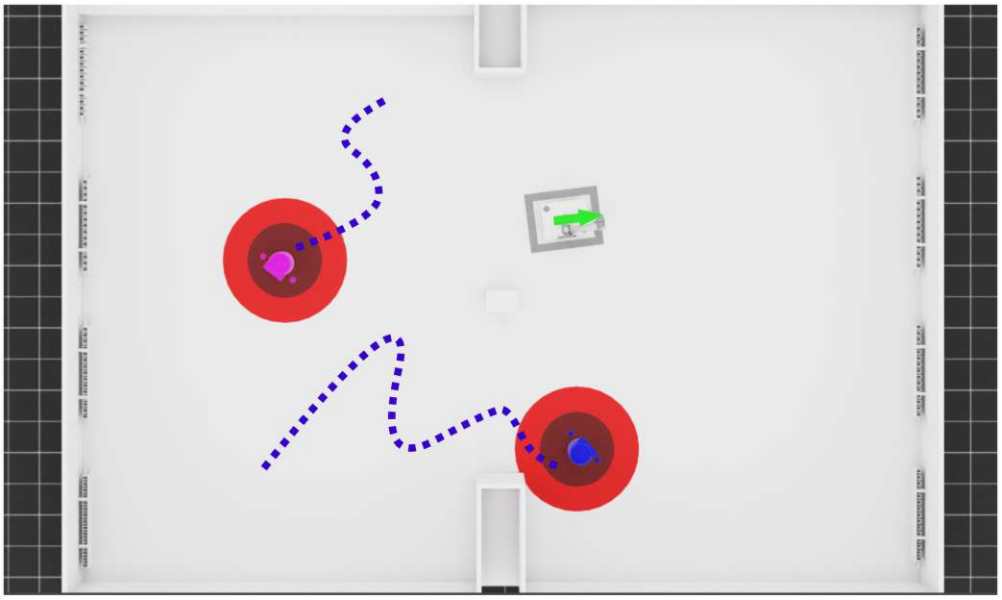


Figure 4.11: Rendering of 3D models of VR users, the robot and the lab in Isaac Sim. Dotted lines represent the replayed trajectory. The red and dark red circles visualize the critical and collision zone. The green arrow above the robot marks its orientation.

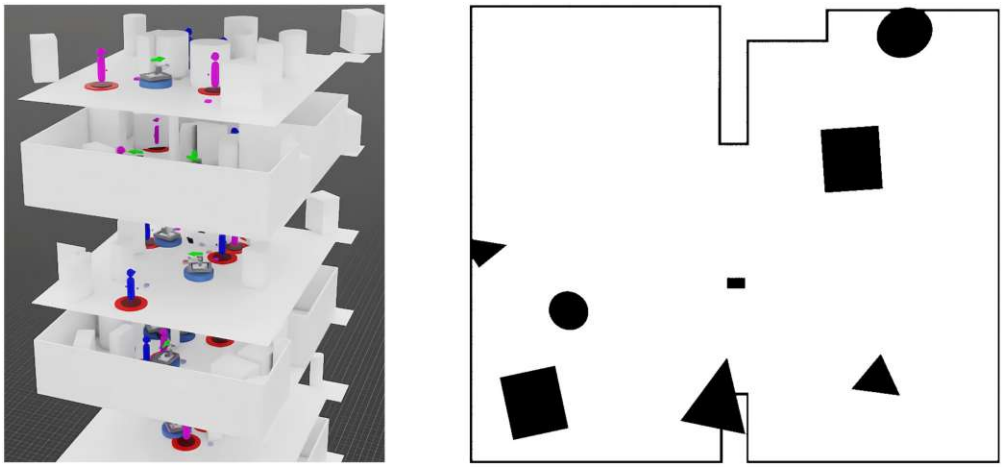
was imported into Isaac Sim. The robot is actuated via a 3-DOF holonomic controller, accepting continuous velocity commands (v_x, v_y, v_ω) from the policy. It is equipped with a simulated 2D LiDAR sensor, with specifications (e.g., 720 rays, 360° field of view) configured to match the real-world hardware.

VR users are represented by 3D models for visualization. Their motion is not dynamically simulated. Instead, at each simulation step, their poses are updated according to the trajectories from our pre-recorded human motion dataset. This provides realistic, data-driven human movement for the agent to react to.

Episode Definition

NavRL and SU Policies The training loop for the navigation policies (NavRL and the SU policy) is defined as follows:

1. **Initialization**: One (for SU) or two (for NavRL) users are spawned on a randomly selected tower floor, each beginning at the start of a randomly sampled trajectory from the training dataset. The robot is spawned at a random, collision-free pose (position and yaw) on the same floor.
2. **Execution**: The simulation proceeds, with the agent controlling the robot and the users' poses being replayed from their respective trajectories.



(a) (b)

Figure 4.12: Environment layouts used for training the navigation agents. (*a*) Parallel training in the procedurally generated tower in Isaac Sim. (*b*) Top-down view of one floor in the tower with random obstacles.

3. **Termination**: An episode ends when one of the following occurs:
   - Haptic Request (Success): The user trajectory replay completes. A 10-second cooldown period begins, giving the robot a final window to reach the HIP. The episode ends successfully after this period.
   - Collision (Failure): The robot collides with a static obstacle (detected by LiDAR) or a user (breaching the d_{coll} threshold). The episode terminates prematurely, and the relevant collision penalties (p_o or p_u) are applied.

To ensure valid, collision-free starting positions, we implemented an adaptive occupancy grid for each floor. If the robot spawns inside an obstacle, that grid cell is marked as non-spawnable. Over time, the grid converges to include only valid spawn locations. Similarly, to avoid colliding with a user at the start position, the robot gets spawned at a random cell that is not occupied by a user.

GuideRL Policy The episode definition for the GuideRL policy is simpler, as it does not involve physical navigation or collision detection:

1. **Initialization**: Two random user trajectories are sampled from the training set.
2. **Execution**: The kinematic data from the user movements is fed to the GuideRL policy, which updates its user weight u_w.
3. **Termination**: After the trajectories finish, a 5-second cooldown period occurs. The episode ends when the haptic request is issued, at which point the terminal reward r_c is calculated.

While Isaac Lab supports multiple RL libraries, we selected skrl (v1.4.3) [58] for its high-performance and well-documented PPO implementation. All training was conducted on a workstation equipped with an Intel Core i9-14700KF CPU, 128 GB of RAM, and an NVIDIA RTX 4090 GPU, running Ubuntu 22.04 LTS. The high-level structure of this training loop is detailed in Algorithm 4.1 with hyperparameters defined in Table 4.4. The training is divided into repeating iterations, each consisting of two main phases: data collection and network updates.

1. **Data Collection**: In the first phase, experience is gathered from n parallel environments. The current policy is executed in each environment for *rollouts* timesteps, collecting a large buffer of state transitions, actions, and rewards. This parallel collection process generates a batch with a total of $n \times rollouts$ data samples per iteration.
2. **Network Update**: In the second phase, the agent learns from the collected data. To improve sample efficiency, the entire batch is reused *learning epochs* times. Within each epoch, the batch is divided into b distinct mini-batches. First, the PPO losses are calculated, then the optimizer performs a full optimization step, updating the network parameters of the actor and critic networks for each mini-batch.

This two-level loop structure means that for each single data collection phase, the network performs a total of *learning epochs* $\times\, b$ gradient updates. This parallelized data gathering allows for stable and highly efficient policy convergence. We chose 50 environment steps per second as our rate in the simulation. The policy inference in both SU and MU scenarios runs at this same rate (50 Hz) in the test environment. This proved to be agile enough in both scenarios for safe positioning.

Algorithm 4.1: Parallel PPO with skrl — High-Level

$n \leftarrow$ *number of envs*
$b \leftarrow$ *mini batches*
for iteration = 1 to $\frac{total\ timesteps}{rollouts}$ **do**
  Collect data $(states, actions, rewards)$ from n envs for *rollouts* steps
  mini-batches $\leftarrow$ Sample b mini-batches with size $\lfloor \frac{rollouts \cdot n}{b} \rfloor$ from collected data
  for epoch = 1 to *learning epochs* **do**
    for mini-batch in *mini-batches* **do**
      Compute PPO losses of minibatch
      Optimizer step, update actor and critic network
    end for
  end for
end for

We chose a batch size of $\approx 100k$ and $\approx 200k$ (GuideRL) as this was the ideal spot for training time and learning performance. The empirical results of Rudin et al. [21] and Andrychowicz et al. [55] guided us in choosing these hyperparameters.

Table 4.4: Hyperparameters for the PPO implementation of skrl per policy. Not mentioned hyperparameters have the default values of the skrl implementation integrated in Isaac Lab (v2.1).

Parameter	**SU**	**NavRL (MU)**	**GuideRL(MU)**
rollouts	48	48	48
learning epochs	5	5	5
mini batches (b)	8	8	8
discount factor (γ)	0.99	0.99	0.99
GAE factor (λ)	0.95	0.95	0.95
learning rate	0.0005	0.0005	0.0005
entropy loss scale	0.005	0.005	0.005
number of envs (n)	2048	2048	4096
total timesteps	960000	960000	240000

4.3.3 Training Results

All trained agents converged to a stable mean episode reward, visualized in Figure 4.13, Figure 4.14, and Figure 4.15. The training times, however, varied significantly based on the complexity of the policy and its network architecture. The GuideRL policy, which has a simpler task and a lightweight MLP-based network, converged the fastest. On our workstation (Intel Core i9-14700KF, 128 GB of RAM, RTX 4090), it reached a stable policy after approximately 50k environment steps, which required only ≈ 5 min of training. This policy was trained on the set of all possible trajectory pairs ($\binom{342}{2} = 58311$), and the resulting agent was able to predict the correct user with 99% accuracy.

In contrast, the agent in the SU scenario and NavRL required substantially more training, which is attributed to the inclusion of LiDAR simulation and their larger network architectures. Specifically, the inclusion of the computationally intensive 1D CNN module for processing the 720-ray LiDAR input slowed training compared to a pure MLP. The SU agent converged after approximately 200k environment steps (≈ 1.5 h), while the NavRL agent took the longest, converging after 600k environment steps (≈ 4 h).

Reward Curves

The training progression for all three policies is visualized through the mean episode reward accumulated over environment steps. Across all plots, a symmetric logarithmic scale is employed on the y-axis to effectively visualize the transition from large initial penalties (resulting from collisions and unsafe states) to stable positive rewards.

In the SU scenario (Figure 4.13), a rapid initial increase in reward is observed. The agent quickly overcomes early failures, transitioning from negative to positive rewards within the first 50000 steps. Convergence to a stable maximum is achieved after approximately 200000 steps, indicating that the policy effectively learned to balance the safety constraints

with the proximity objectives without significant oscillations.

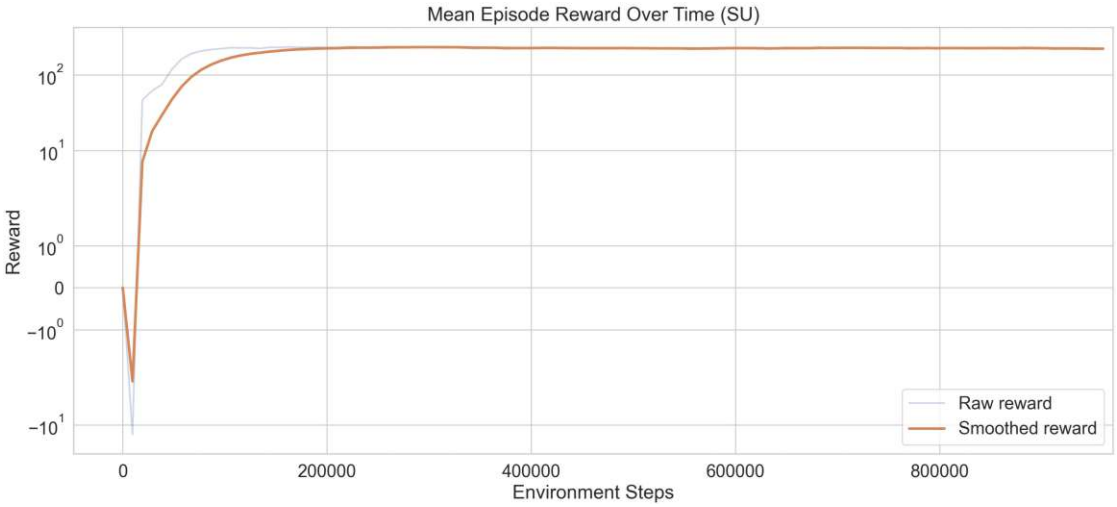


Figure 4.13: Plot of mean episode reward over time for the SU agent. The reward axis is scaled by a symmetric logarithm for a better visualization.

The GuideRL policy (Figure 4.14) exhibits the fastest convergence rate. As this task is limited to a low-dimensional strategic prediction without complex navigation dynamics, the optimal policy is identified almost immediately. The reward curve flattens significantly before 50000 steps, reflecting the high accuracy attained in predicting the target user.

Figure 4.14: Plot of mean episode reward over time for the GuideRL agent. The reward axis is scaled by a symmetric logarithm for a better visualization.

On the other hand, the NavRL agent (Figure 4.15) demonstrates a more gradual learning curve. Due to the increased complexity of the state space, which requires avoiding two

dynamic users while tracking a moving goal, the initial exploration phase is longer. Slower convergence is visible in the raw reward signal during the first 200k steps. However, a stable, high-reward policy is ultimately converged upon after approximately 600k steps.

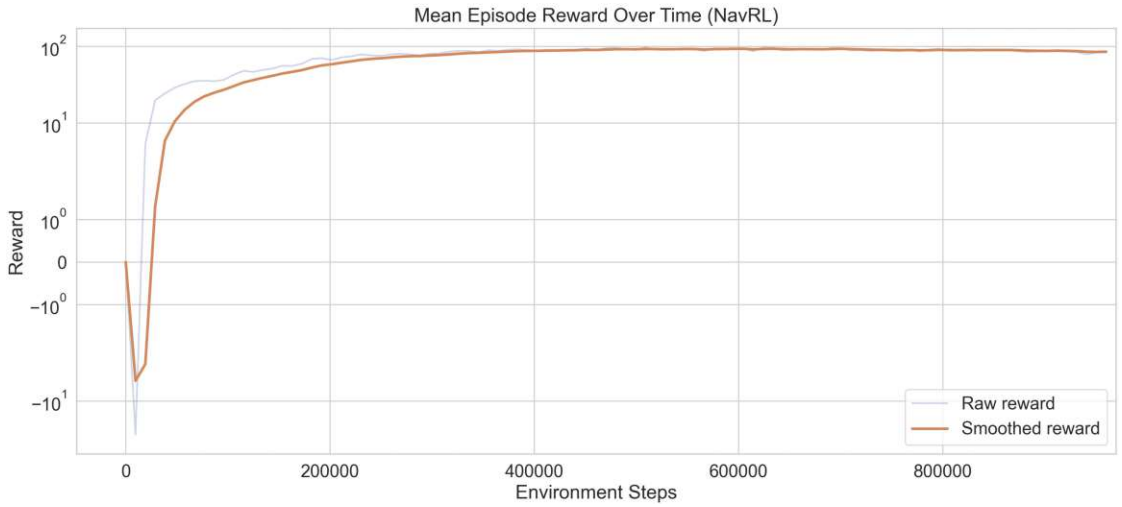


Figure 4.15: Plot of mean episode reward over time for the NavRL agent. The reward axis is scaled by a symmetric logarithm for a better visualization.

Emergent Behavirous

The trained agents have acquired robust safety and positioning strategies in both scenarios. To illustrate these capabilities, example episodes of learned behaviors within the Isaac Sim simulation are presented in Figure 4.16 and Figure 4.17.

SU Scenario Figure 4.16 highlights the agent's ability to balance goal tracking with environmental constraints. In row (a), the agent demonstrates efficient path planning. At $T = 0$, recognizing that the direct path to the HIP is blocked by a central pillar, it selects the shortest collision-free route around the obstacle. Row (b) illustrates proactive collision avoidance in confined spaces. Between $T = 0$ and $T = 6$, the agent deviates from the user's HIP to avoid a collision with the upper wall. Furthermore, at $T = 9$, the system handles an edge case where the user's HIP is located inside a static obstacle. Rather than attempting to enter the obstacle, the agent intelligently parks at the closest valid position outside the inflation layer and danger zones of the user.

MU Scenario The complexity of the MU scenario requires the agent to anticipate the movements of two independent users, as shown in Figure 4.17. Row (a) demonstrates the collision avoidance capabilities of NavRL. At $T = 3$, the agent detects that maintaining its position would lead to being trapped between two users. It proactively retreats to maintain an escape route. Row (b) showcases the interaction between the GuideRL and

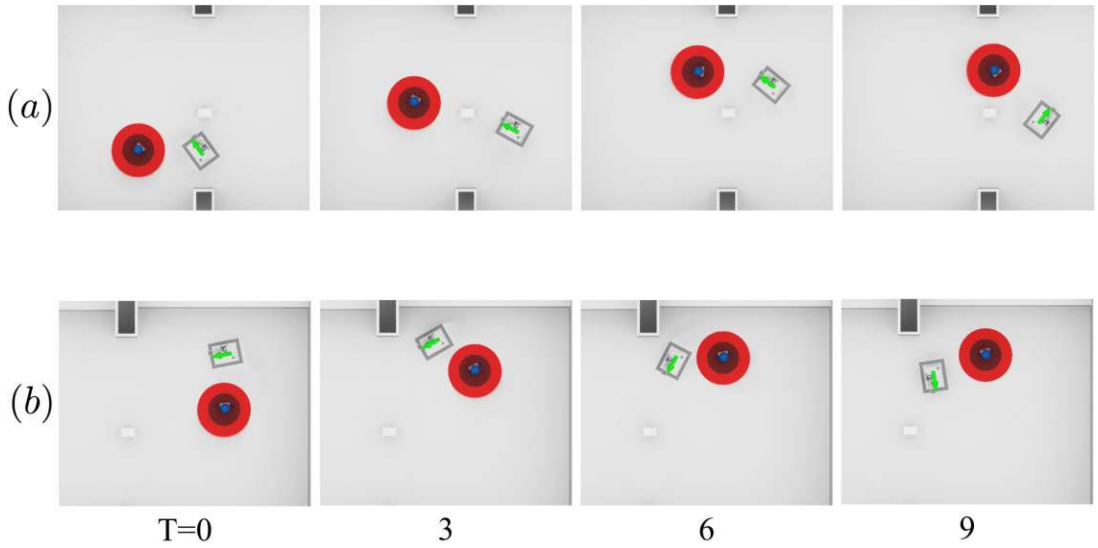


Figure 4.16: Visualization of learned behaviours in the SU scenario. The robot's orientation is indicated by the green arrow, while the dark red and red circles represent collision and critical safety zones, respectively. (*a*) The agent navigates around a static obstacle (pillar) to maintain proximity to the user. (*b*) The agent performs proactive collision avoidance ($T = 3$) and handles an invalid goal pose located inside an obstacle ($T = 9$).

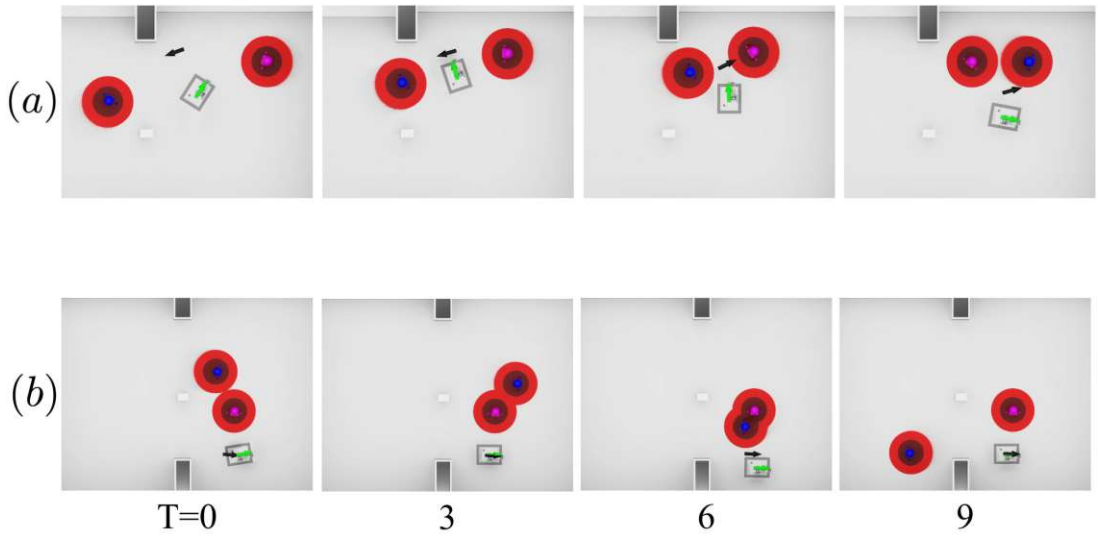


Figure 4.17: Visualization of learned behaviours in the MU scenario. The black arrow represents the dynamic goal pose generated by GuideRL. (*a*) The agent retreats from a potential encirclement by two users ($T = 6$). (*b*) The NavRL policy temporarily abandons the target HIP ($T = 6$) to avoid a collision with the second (blue) user passing through the interaction zone.

NavRL policies. While GuideRL correctly identifies the pink user as the target, the blue user moves to intercept that location at $T = 6$. Consequently, the NavRL policy overrides the goal tracking to yield space, preventing a collision with the other user before repositioning back to the HIP at $T = 9$.

4.4 Iterative Development Process

The design of the agents for both the SU and MU scenarios was an iterative process. The final architecture was the result of several major design changes. This section highlights the most significant iterations and the rationale for abandoning them in favour of our final approach.

4.4.1 Hybrid method with ROS

Our development began with a "start simple" guiding principle, focusing first on the SU scenario. The initial iteration was a hybrid approach, attempting to combine the strengths of DRL with traditional robotics methods. In this model, the agent's role was high-level. It was tasked with outputting a discrete action with k possible values, which corresponded to a predefined goal position in a discretized grid of the environment's free space, see Figure 4.18. This chosen goal was then passed to the ROS navigation

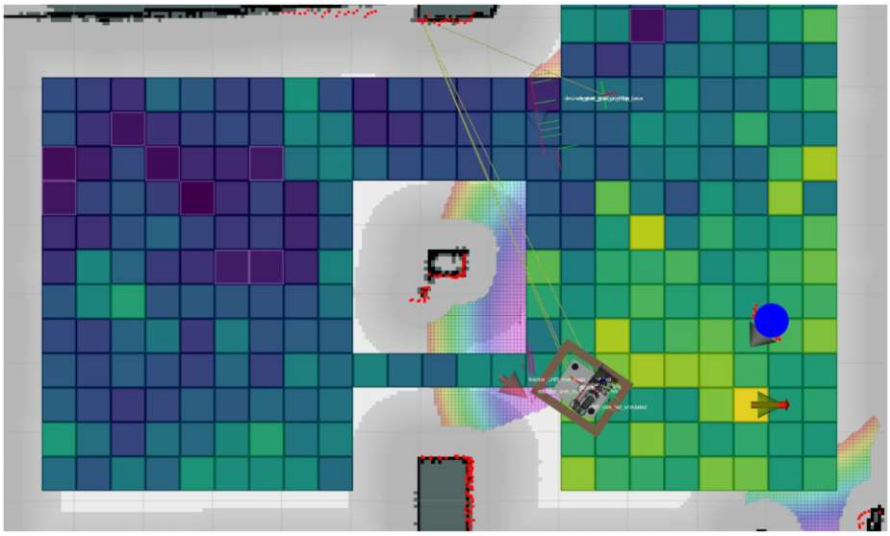


Figure 4.18: A visualization of the categorical distribution of the hybrid method agent. The more yellow the cell is, the more likely this cell is to be chosen as the goal position.

stack, which handled all low-level path planning, velocity control, and obstacle avoidance. The hypothesis was that this would leverage DRL for high-level strategic positioning while relying on the well-vetted ROS stack for robust navigation. However, this approach presented two significant drawbacks. First, limited dynamic agility. The standard ROS

navigation stack, while robust, lacked the agility for the highly dynamic, close-proximity interactions required by our pre-haptic positioning task. Furthermore, the agent's action space (the grid of valid goals) was intrinsically tied to a specific, static environment layout. Any change to the lab's configuration, such as moving a table, would invalidate the action space and require a complete retraining of the policy. Given these critical limitations in reactivity and generalization, we abandoned this hybrid model. We moved to an end-to-end approach where the agent is directly responsible for issuing continuous velocity commands.

4.4.2 Continous actions

Adopting a continuous action space transferred the full responsibility of collision avoidance to the DRL agent. Our second iteration drew inspiration from the local planning component of the ROS navigation stack, which uses a sliding local occupancy grid.

Autoencoder

We designed an autoencoder to process a similar local occupancy grid, centered on the robot. The objective was to pre-train this autoencoder to compress the high-dimensional grid into a low-dimensional, fixed-size latent vector z, as shown in Figure 4.19. This latent vector z was then concatenated with the other state information and fed to the policy network. The hypothesis was that this compressed representation would provide an abstract and generalizable summary of the robot's immediate surroundings.

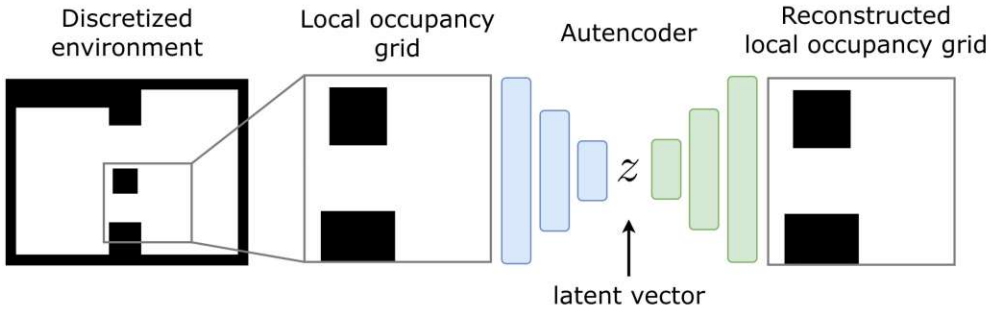


Figure 4.19: The autoencoder architecture for local occupancy grid compression. The encoder (blue) generates a latent vector z, which is fed to the RL policy. The decoder (green) is used only during the pre-training phase to reconstruct the input grid.

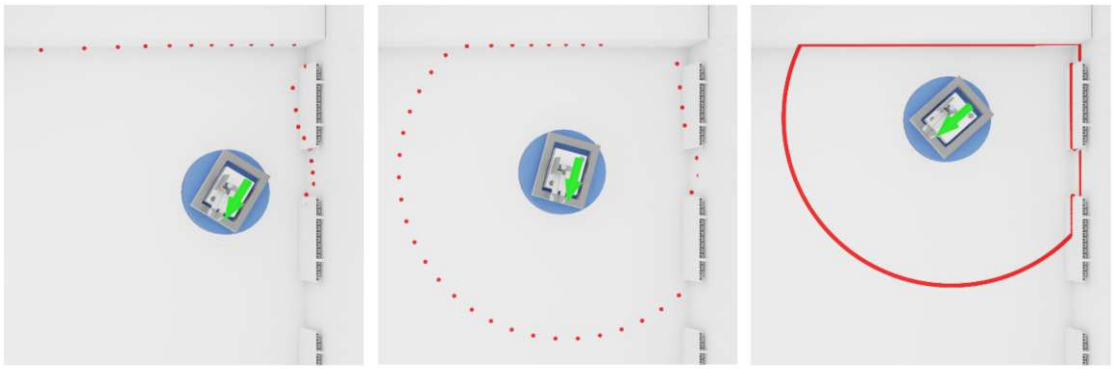
This agent learned to avoid static obstacles within the known lab environment. However, the autoencoder was trained on data samples generated only from our lab. Its learned latent representation did not generalize to unseen obstacle configurations in new environments. Additionally, the local occupancy grid, by design, was rotation-invariant relative to the world frame. This meant the agent could detect obstacle proximity but lacked a clear understanding of their direction relative to the robot's own heading. This led to unstable and erratic angular velocities when navigating near obstacles.

Due to these significant issues, we discarded this method and pivoted to a state representation that directly incorporates raw sensor data with a clear directional reference.

LiDAR

The autoencoder's failure, primarily due to its lack of rotational awareness, led us to adopt LiDAR data as the primary collision sensor input. LiDAR inherently provides a clear directional reference, as each ray has a known, fixed angle relative to the robot's frame. However, the 720-ray scan presented a high-dimensional input that proved difficult to train with a standard MLP. Our first iteration, shown in Figure 4.20(a), addressed this by sampling the scan, using only every 15th ray (48 rays total) at the hardware's full 30m range. While this agent learned static avoidance, it failed to generalize. We observed that the policy became unstable in open environments, apparently having learned a reliance on seeing distant obstacles to maintain stable control.

This led to the hypothesis that a purely local perceptive field would be more robust and generalizable. We implemented this by clamping the LiDAR ranges to a maximum of 3 m, as shown in Figure 4.20(b). This local awareness, combined with the diverse obstacle layouts of the procedurally generated tower Figure 4.12, produced the first SU agent that successfully generalized to unseen environments.



(a) (b) (c)

Figure 4.20: Iterations of different LiDAR state representations. (a) unclamped sampled LiDAR ranges with 48 rays. (b) clamped sampled LiDAR ranges with 48 rays. (c) clamped LiDAR ranges with 720 rays. The final model uses (c) in combination with a 1D CNN.

Although the sampled-and-clamped (48-ray) LiDAR representation was successful, its low horizontal resolution led to failures in some edge cases, such as collisions with thin or sharp obstacles. To resolve this, we adopted a 1D CNN method, inspired by the work of Zarrar et al. [59]. This network functions as a collision avoidance module, allowing the policy to process the full-resolution 720-ray scan in an end-to-end manner Figure 4.20(c). The 1D CNN learns to extract spatial features, such as gaps, edges,

and surfaces, directly from the raw sensor data. This module, when combined with our boundary-aware inflation layer penalty p_i Equation 4.13, resulted in significantly smoother, safer, and more reliable movement around all obstacle types. This became the final, definitive state representation for both the SU and MU navigation policies.

4.4.3 MU policy split

While the end-to-end SU agent was now successful, this monolithic approach repeatedly failed for the MU scenario. The agent did not converge and was unable to effectively minimize haptic response times. We identified the core challenge as a conflict of objectives: the agent was being asked to simultaneously perform complex, low-level navigation and predict which of the two users would issue the next haptic request. To resolve this complexity, we decomposed the task into two independent policies:

- **GuideRL**: A strategic policy whose only role is to predict the target user.
- **NavRL**: A navigation policy, almost identical in architecture to the SU agent, responsible for navigating to the goal provided by GuideRL.

This hierarchical separation of concerns proved highly effective. GuideRL focuses solely on the user-prediction problem, while NavRL focuses on the solved problem of safe navigation. The specific implementations of these two policies are detailed in Section 4.2.

CHAPTER 5

Evaluation and Results

This chapter details the evaluation of our DRL-based positioning framework. We first outline the experimental setup, define the metrics and hypotheses, and describe the baseline algorithms used for comparison. Finally, we present the quantitative results for both single-user (SU) and multi-user (MU) scenarios.

5.1 Experimental Setup

To ensure fair and reproducible comparisons, all experiments were conducted in a controlled simulation environment. The lab workspace and the CoboDeck robot platform were simulated in Gazebo [60]. CoboDeck is based on a Robotnik Kairos rover with a 0.91 × 1.13 m footprint and omnidirectional Mecanum wheels. A testing framework, implemented in Unity 3D as described by Mortezapoor et al. [23], was used to manage the experiments. This framework controls the movement of the simulated users by replaying pre-recorded trajectories, while ROS is responsible for the movement of the robot. The Unity 3D framework observes and logs the robot's behaviour, which is controlled by either our DRL policy or a baseline algorithm. The software was distributed across two workstations. The primary host (Ubuntu 22.04, Intel Core i9-14700KF, 128 GB RAM, Nvidia RTX 4090) ran the Gazebo simulation and the primary Unity instance. For the MU setup, a second PC (Ubuntu 22.04, Intel Core i7-9900K, 32 GB RAM, Nvidia RTX 2080Ti) ran the second Unity instance, connected via LAN.

5.1.1 Test Data and Procedure

To move the simulated users, we selected 100 consecutive trajectories from our recorded human motion dataset. Crucially, these trajectories were not part of the training set. At the end of each trajectory, a haptic request was issued by the user.

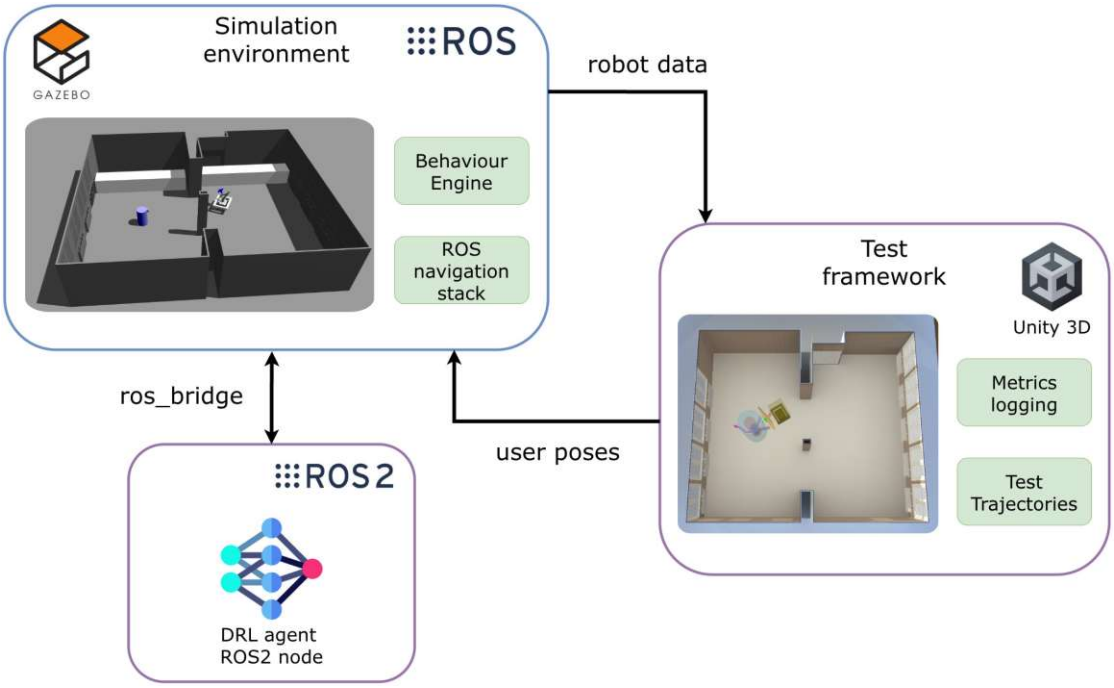


Figure 5.1: Overview of the simulation and testing framework with the DRL method. The DRL agent, running as a ROS2 node, controls the robot within the Gazebo simulation, while a Unity 3D-based framework drives user motion and logs performance metrics. In the MU scenario, a second Unity 3D instance is connected to the host instance, providing ROS and the testing framework with the second user position.

- During the *Haptic* behaviour: The ROS navigation stack took control to drive to the interaction point and deploy the haptic prop.

- During the *Chase* behaviour: The robot's positioning was governed by one of the test algorithms (our DRL policy or a baseline).

- Safety override: If the behaviour engine detected a dangerously close proximity to a user (< 1.5 m), the *Escape* behaviour took control to drive the robot to a safe position away from the user's path, after which control was returned to one of the test algorithms.

- During the *Inform* behaviour: If, for some reason, *Escape* could not find a safe position, *Inform* alerts the user to stop their movement. Additionally, it visualizes the robot in the VR environment until the distance user-robot distance is big enough. However, if the robot is stuck in a corner, it could happen that *Inform* is stuck in an endless loop. To mitigate this, if 10 s in the *Inform* behaviour pass, the user walks back along the trajectory until the robot is unstuck.

In the MU scenario, the second user was assigned the same 100 trajectories but with a small trajectory offset. This ensured that the user paths were different and did not overlap. This trajectory data was fixed across all experimental conditions. Ensuring a fair comparison between methods required identical start conditions for each of the 100 test trajectories. While trivial for the SU scenario, this presented a significant challenge in the MU case. We observed that during long test runs, the robot's starting state at the beginning of a new trajectory would "drift", becoming inconsistent across different algorithms. This was because minor timing differences due to stops caused by the *Inform* behaviour would compound, causing the haptic requests to occur at different times. This, in turn, meant the robot would be in a different position when the next trajectory began, invalidating a direct comparison. To eliminate this variable and ensure each of the 100 trajectories was evaluated from an identical, reproducible starting point, we developed a strict MU-specific test procedure that reset the environment for every single trajectory pair:

1. Both users are spawned at the start of their respective trajectories for the current test case.
2. The robot is spawned at one of four predefined, collision-free start locations.
3. The trajectory replay and the test algorithm are initiated.
4. At the precise moment the first user issues a haptic request, the second (non-requesting) user is programmatically removed from the environment. This was a critical step, as we observed cases where the second user would interfere by stopping near or at the first user's HIP.
5. The robot (now with the second user removed) completes the haptic request, metrics are logged, and the entire process is repeated for the next trajectory pair in the test set.

5.2 Evaluation Metrics and Hypotheses

The evaluation is designed to test three primary hypotheses:

- *H1*: *"The DRL-based approach is safer than the baseline algorithms"*
- *H2*: *"The DRL-based approach is more responsive than the baseline algorithms"*
- *H3*: *"The DRL-based approach is more efficient than the baseline algorithms"*

To test these hypotheses, we define two categories of metrics:

5.2.1 Responsiveness and Efficiency Metrics

These metrics quantify how quickly and efficiently the robot can fulfil haptic requests.

Haptic Positioning Time T_{pos} This is our primary responsiveness metric. It measures the time (in seconds) from the start of a haptic request to the moment the mobile platform reaches the correct pose to begin its arm movement. The total response time is:

$$T_{resp} = T_{pos} + T_{arm}, \tag{5.1}$$

where $T_{arm} \approx 3.3$ s is independent of the positioning strategy. Thus, all comparisons are based on T_{pos}. In the MU scenario, only the positioning time to the first requesting user is recorded and relevant.

Distance to HIP The distance (in meters) the robot travelled to reach the HIP at the time a haptic request is issued. This is highly correlated with T_{pos} but is included to help apply the findings to robots with different speed capabilities. Furthermore, the robot might be close to the position of the HIP, but the rotation of the robot is not aligned with the HIP, causing a delay in reaching the correct rotation.

Total Travelled Distance The total path length (in meters) traversed by the robot during the entire 100-trajectory test run. This serves as a measure of motion efficiency and energy consumption.

Successful Haptic Serves A count of haptic requests that were successfully fulfilled by the robot (i.e., the prop was correctly placed) while the request was still active.

5.2.2 Safety Metrics

These metrics assess the risk of harm to users or the robot.

Robot Collisions The total number of collisions between the robot and either a user or the environment.

Escape Activations The total number of time the behaviour *Escape* was triggered. This metric is a critical indicator of pre-collision risk, as it counts how often the robot was in an imminently dangerous position that required a safety override.

5.3 Baselines

We evaluate our DRL approach against two baseline algorithms for pre-haptic positioning.

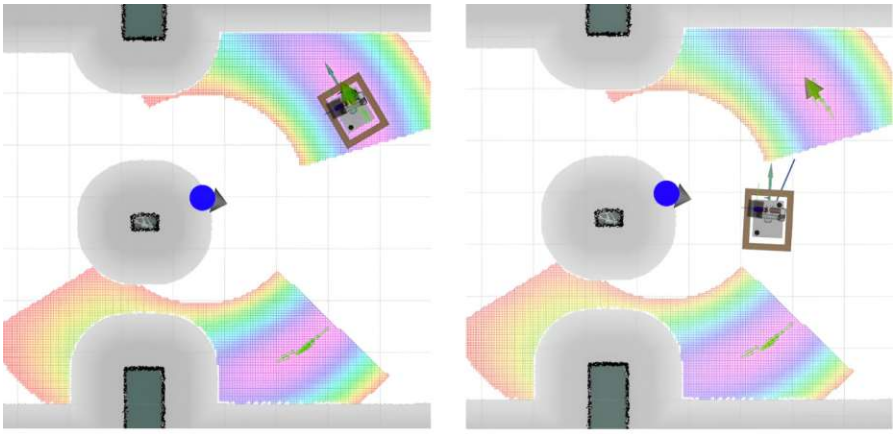
5.3.1 Static Positioning

This is a minimal "do-nothing" algorithm that serves as a lower bound for performance. The robot remains stationary at the location of its last haptic interaction. It only moves in two cases: a new haptic request is received, or the *Escape* behaviour is triggered by an imminent collision. This baseline is used in both SU and MU scenarios.

5.3.2 Heuristic Positioning

This baseline, referred to as Heuristic Positioning, was proposed by Mortezapoor et al. [10] for the CoboDeck platform. It generates a *Chase Space* consisting of two arcs on either side of the user. This space is populated with candidate *Chase Poses* that are oriented to allow the robot to easily drive forward or backward to the HIP.

Each pose is weighted, and after checking against a static occupancy map to exclude collisions, the robot moves to the best available pose. This algorithm was designed only for the SU case and is therefore not used in our MU evaluation.



(a) (b)

Figure 5.2: Visualization of the *Chase Space* proposed by Mortezapoor et al. [10]. (*a*) The robot moves to the *Chase pose* with the most weight. Potential *Chase Poses* are visuzalized with small green arrows in the high weight area (magenta). (*b*) A haptic request is issued, and the robot moves towards the HIP. The large green arrow visualizes the last *Chase Pose* the robot moved to before.

5.4 Results

This section provides an overview of the results across both test scenarios (SU and MU). The performance is evaluated against our three hypotheses concerning safety (*H1*), responsiveness (*H2*), and efficiency (*H3*). A comprehensive summary of all metrics for all test conditions is presented in Table 5.1. The subsequent figures visualize these key comparisons: Figure 5.3 details the safety outcomes, while Figure 5.5 and Figure 5.6 illustrate the distributions for responsiveness.

5.4.1 SU scenario

In the SU scenario, we compare our DRL method (SU-DRL) against two baselines, the static baseline (SU-Static) and the heuristic baseline (SU-Heuristic).

Table 5.1: Results of the evaluations in the SU and MU scenario with different pre-haptic algorithms. Bold numbers indicate the best performance. ↓ indicates that lower values are better, while ↑ indicates that higher values are better. ∗ The missing haptic interaction in both cases was due to a failure in the ROS navigation system, independent of the positioning method.

		SU			**MU**	
		Static	Heuristic	DRL	Static	DRL
Responsiveness and Efficiency	Haptic positioning time Median [IQR] in s ↓	4.84 [2.27]	3.25 [0.73]	**0.90 [0.22]**	6.04 [3.07]	**1.07 [1.39]**
	Distance to HIP Median [IQR] in m ↓	4.28 [3.29]	2.35 [0.41]	$\mathbf{1 \times 10^{-4}}$ $\mathbf{[7 \times 10^{-4}]}$	5.46 [4.77]	$\mathbf{3 \times 10^{-3}}$ **[0.44]**
	Total Travelled Distance in m ↓	**827**	1824	1909	**804**	1882
	Successful haptic serves Served (Requested) ↑	99* (100)	99* (100)	**100 (100)**	100 (100)	100 (100)
Safety	Robot collisions ↓	0	0	0	0	0
	Escape activations ↓	166	214	**3**	128	**15**

Safety (*H1*)

The safety metrics provide strong support for our first hypothesis *H1*. As shown in Figure 5.3 (left), the SU-DRL method required drastically fewer safety overrides than either baseline. The SU-DRL agent triggered the *Escape* behaviour only 3 times in total. This represents a 98.2% reduction compared to the 166 activations for the SU-Static method and a 98.6% reduction compared to the 214 activations for the SU-Heuristic method. This result demonstrates that the agent's learned proactive avoidance is far more robust and anticipatory than both the baseline's purely reactive measure and the rule-based heuristic's navigation. It is important to note that zero collisions were recorded for all three methods, which confirms the effectiveness of the *Escape* behaviour as a critical final safety net.

Responsiveness and Efficiency (*H2, H3*)

The results for responsiveness and efficiency show a clear trade-off, strongly confirming hypothesis *H2* while rejecting *H3*.

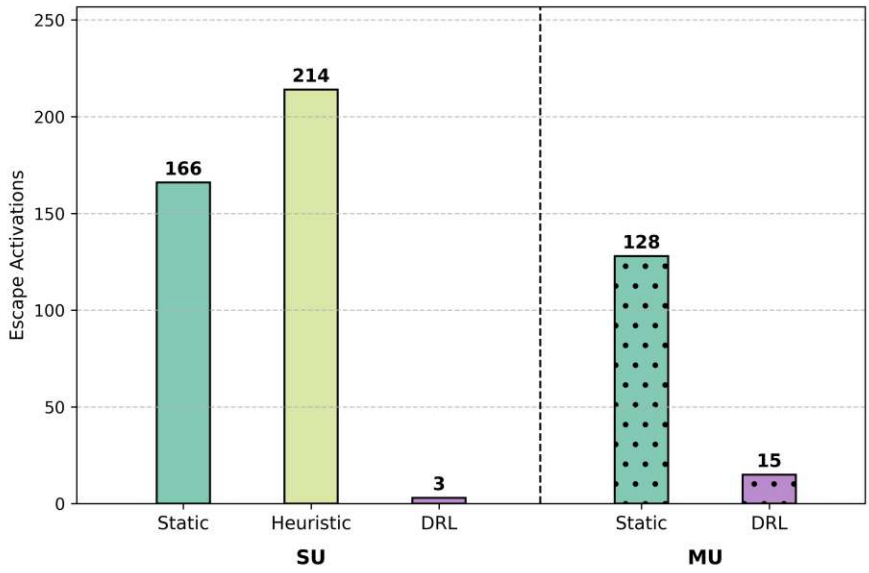


Figure 5.3: Count of total *Escape* behaviour activations for each method for SU (left) and MU scenario (right).

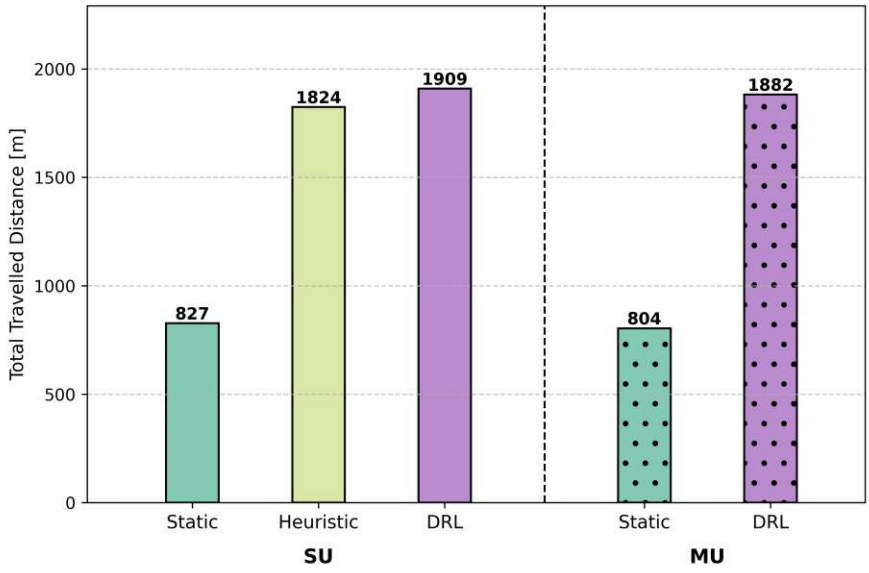


Figure 5.4: Total Travelled Distance for each method. SU (left) and MU scenario (right).

Responsiveness (*H2*) The data provides clear, statistically significant validation of H2: the SU-DRL method is substantially more responsive than both baselines. The Distance to HIP (Figure 5.5, (top)) highlights the agent's superior proactive positioning. The SU-DRL method achieved an astonishing median distance of only 1×10^{-4} m [IQR

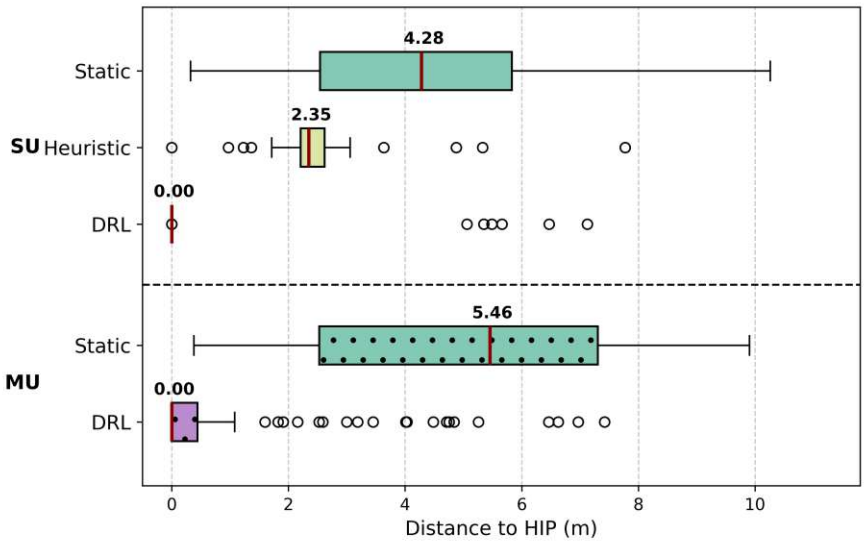


Figure 5.5: Distance to HIP boxplots for static positioning, heuristic positioning, and our method (DRL positioning) in both SU (top) and MU scenarios (bottom).

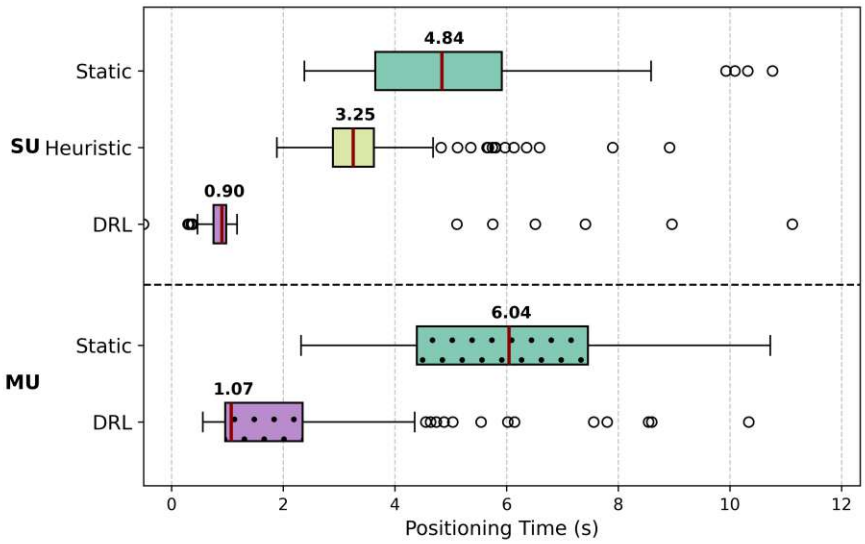


Figure 5.6: Haptic positioning times T_{pos} boxplots for static positioning, heuristic positioning, and our method (DRL positioning) in both SU (top) and MU scenarios (bottom).

7×10^{-4}]. This is a significant improvement over both the Heuristic method's 2.35 m [IQR 0.41] and the Static method's 4.28 m [IQR 3.29]. This proximity to the goal translates

directly into reduced Haptic Positioning Times T_{pos}, as seen in Figure 5.6 (top). The SU-DRL method's median T_{pos} was 0.90 s [IQR 0.22], which is 3.6x faster than the Heuristic baseline (3.25 s [IQR 0.73]) and 5.4x faster than the Static baseline (4.84 s [IQR 2.27]).

A Friedman test confirmed a significant effect of the positioning method on both T_{pos} ($\chi^2(2) = 133.4, p < 0.001$, Kendall's $W = 0.681$) and Distance to HIP ($\chi^2(2) = 136.7, p < 0.001$, Kendall's $W = 0.697$). Pairwise Wilcoxon signed-rank tests with Holm's correction confirmed that the SU-DRL method was significantly more responsive than the Static baseline (for T_{pos}: $Z = 8.33, p < 0.001, r = 0.84$; for Distance: $Z = 8.35, p < 0.001, r = 0.84$) and the Heuristic baseline (for T_{pos}: $Z = 7.28, p < 0.001, r = 0.74$; for Distance: $Z = 6.99, p < 0.001, r = 0.71$).

Efficiency (*H3*) This enhanced responsiveness, however, comes at the cost of motion efficiency, leading us to reject H3. A clear trade-off is evident in the Total Travelled Distance (Figure 5.4). The SU-Static method was the most efficient, moving only 827 m. In contrast, the proactive methods travelled significantly more: the SU-DRL agent travelled 1909 m, and the SU-Heuristic method travelled 1824 m. This demonstrates that the DRL agent's high-responsiveness, high-safety performance is achieved through active, anticipatory repositioning, which has an energy cost comparable to the rule-based heuristic method. Both SU-Static and SU-Heuristic served 99 out of 100 haptic serves, while SU-DRL served all haptic requests successfully. The missing haptic interaction in both cases was due to a failure in the ROS navigation system, independent of the positioning method.

5.4.2 MU scenario

In the MU scenario, we compare our DRL method (MU-DRL) against the static baseline (MU-Static). As noted in Section 5.3, the heuristic baseline does not apply to this scenario. The results largely follow the same patterns of high safety and responsiveness at the cost of efficiency that were observed in the SU case, even with the added complexity of a two-user environment.

Safety (*H1*)

The safety metrics once again strongly support our first hypothesis *H1*. The results, summarized in Figure 5.3, show that the MU-DRL method is demonstrably safer than the static baseline. The MU-Static method triggered 128 *Escape* activations, whereas our MU-DRL method required this safety override only 15 times, a reduction of 88.3%. This finding is even more significant than in the SU case, as the agent must not only avoid collisions but must do so while navigating between two dynamic users. The drastic reduction in *Escape* activations shows that the agent's learned, proactive collision avoidance is highly effective and inherently safer than the baseline's purely reactive safety measure. It is noteworthy that, similar to the SU scenario, both positioning algorithms

resulted in zero collisions, which confirms the effectiveness of the *Escape* behaviour as a critical safety override.

Responsiveness and Efficiency (*H2, H3*)

The responsiveness and efficiency results provide a more nuanced picture, confirming *H2* while rejecting *H3*, a conclusion that perfectly mirrors the findings from the SU scenario.

Responsiveness (*H2*) The data provides a clear and statistically significant validation of our second hypothesis *H2*: the MU-DRL method is vastly more responsive than the static baseline. The distance to HIP, visualized in Figure 5.5, serves as the most direct measure of the MU-DRL method's performance. Our method maintained a median distance of 3×10^{-3} m [IQR 0.44] to the correct user's HIP, indicating that the GuideRL policy was highly successful in predicting the target user, and the NavRL policy was able to execute the navigation with high precision. In stark contrast, the baseline method had a median distance of 5.46 m [IQR 4.77]. This positional advantage translates directly to the median positioning times T_{pos}, as shown in Figure 5.6. Positioning times T_{pos} for MU-DRL were 1.07 s [IQR 1.39], while MU-Static was, on average, 5 s slower with a median of 6.04 s [IQR 3.07].

A Wilcoxon signed-rank test confirmed that these differences are statistically significant. The results show that the differences were significant for both positioning times ($Z = 7.667, p < 0.001, r = 0.767$) and distances to HIP ($Z = 7.884, p < 0.001, r = 0.788$). In both cases, the p-value was exceptionally small ($p \approx 10^{-14}$), and the large effect sizes ($r \geq 0.5$) indicate that the DRL agent's superior responsiveness is substantial.

Efficiency (*H3*) This superior responsiveness, however, comes at the cost of efficiency, leading us to reject *H3*. This trade-off was also observed in the SU case. Our analysis revealed that while the MU-Static baseline algorithm moved a total of 804 m, the proactive MU-DRL method travelled more than double that distance, accumulating 1882 m (Figure 5.4). This increased travel is the explicit cost of the proactive strategy of constantly repositioning to anticipate the next haptic request from one of two users. Finally, despite the added complexity, both methods successfully served 100 out of 100 haptic requests.

CHAPTER 6

Discussion

This chapter interprets the evaluation results, validating the DRL agent's superior safety and responsiveness against the proposed hypotheses (*H1-H3*). It examines the necessary trade-off between haptic positioning time and motion efficiency, analyzes the critical role of the hierarchical policy decomposition, and explores the framework's broader applicability to dynamic Human-Robot Interaction tasks.

6.1 Interpretation of Results

The evaluation detailed in Chapter 5 provides strong evidence for the effectiveness of the proposed DRL framework. Overall, the results demonstrate that our approach is both safer and significantly more responsive than the baseline methods.

6.1.1 Safety (*H1*)

In terms of safety, the DRL agent's proactive collision avoidance was highly effective, reducing the number of critical *Escape* behaviour activations by 99% in the single-user (SU) scenario compared to the heuristic baseline (3 vs. 214 activations) and by 88% in the multi-user (MU) scenario compared to the static baseline (15 vs. 128 activations). This stems from two key factors. First, it learned an anticipatory policy that understands the need to maintain a safe buffer from users and walls, rather than reacting only after a safety boundary is breached. Second, the DRL agent's end-to-end architecture allows for a much faster reaction time. The policy inference operates at 50 Hz, feeding velocity commands directly to the robot.

In contrast, the heuristic baseline, which combines the *Chase* behaviour logic with the full ROS navigation stack, involves a much slower, more deliberative planning loop. This overhead makes it fundamentally less equipped to handle the fast, dynamic movements of a human user. The static baseline has no proactive logic at all and relies entirely on

the reactive *Escape* behaviour, which accounts for its 166 activations. Interestingly, but not surprisingly, the heuristic baseline had more *Escape* activations (214) than the static baseline in the SU scenario. Close-proximity pre-haptic positioning strategies are more likely to produce user movement obstructions, with the benefit of being more responsive, consistent with the findings of CoboDeck [10]. The drop in *Escape* activations for the static baseline in the MU scenario stems from the test framework's reset logic. Because the robot respawns in a safe, distant position away from users, the likelihood of immediate collisions is reduced compared to the SU scenario.

In summary, the DRL agent combines the proactive logic of a well-designed reward function with the high-frequency reaction time of a lightweight neural network controller, resulting in a system that avoids dangerous situations altogether rather than just reacting to them.

6.1.2 Responsiveness and Efficiency (*H2, H3*)

The data provides overwhelming support for *H2*, showing the DRL agent to be faster in pre-haptic positioning. The DRL agent's median haptic positioning time T_{pos} was 3.6x faster than the heuristic baseline in the SU scenario (0.90 s vs. 3.25 s) and 5.6x faster than the static baseline in the MU scenario (1.07 s vs. 6.04 s). This was a direct result of the agent successfully learning to minimize its distance to the Haptic Interaction Pose (HIP), achieving a median Distance to HIP of just 1×10^{-4} m (SU) and 3×10^{-3} m (MU). The ROS navigation stack assumes control after the haptic request and stops movement upon reaching a minimum goal threshold (0.2 m). This strictly defined stopping condition accounts for the small Distance to HIP values recorded. This responsiveness is a direct consequence of the superior safety and precision discussed above. The agent's primary positive rewards r_d, r_g incentivized it to constantly minimize the distance to the HIP, resulting in sub-centimeter Distance to HIP values in both scenarios. The heuristic baseline, however, cannot safely operate this close to the user. As established, its slower planning loop requires it to maintain a large, hard-coded safety buffer, the *Chase Space*, to compensate for its own poor reaction time. This safety offset, which is a workaround for its own limitations, inherently prevents responsive positioning, as the robot is forced to be several meters away from its target.

The DRL agent's combination of proactive safety and fast control eliminates the need for this large, hard-coded buffer. It is safe enough and fast enough to chase the HIP with high precision, allowing it to be in the perfect position the moment a haptic request is issued. Its superior safety, therefore, is exactly what makes the higher responsiveness possible.

Hypothesis *H3* was rejected in both scenarios, as the DRL agent travelled significantly more than the static baseline. The DRL agent's total travel distance (1909m SU, 1882m MU) was comparable to the proactive heuristic baseline (1824m) and significantly higher than the static baselines. This result is not a failure, but a deliberate and necessary trade-off for a responsive system. A system that must be ready to respond instantly must

be in constant, anticipatory motion. The static method's low travel distance is merely a by-product of it doing nothing. Furthermore, the "efficiency" of the static method is misleading. Its 827 m / 804 m of travel was composed almost entirely of high-velocity, high-acceleration *Escape* manoeuvres. These sudden, high-jerk movements, while covering less total distance, result in a high-energy drain and significant mechanical strain on the robot's hardware. The DRL agent's higher travel distance represents a different and preferable efficiency profile. Its motion, while constant, was actively smoothed by the action smoothness and acceleration smoothness penalties in its reward function. This results in a predictable, constant energy draw and low mechanical stress, which is far more desirable for long-term hardware health than the erratic movements of the static baseline. The DRL agent trades total distance for motion quality and hardware longevity.

6.2 Takeawys

6.2.1 Hierachical Decomposition

A key methodological contribution of this thesis is the hierarchical decomposition of the MU positioning problem. As noted in Subsection 4.4.3, initial attempts to train a single, monolithic MU policy repeatedly failed to converge. The agent was unable to learn a stable policy, highlighting a fundamental conflict of objectives. It was being asked to simultaneously solve two distinct and complex problems: low-level, high-frequency motor control for safe navigation, and high-level, strategic prediction of human intent. The decision to decompose this problem into two independent policies, GuideRL and NavRL, proved to be the critical breakthrough for the MU scenario. This hierarchical separation of concerns was highly effective for several reasons:

- It solved the conflict of objectives. By splitting the task, each policy could be optimized for one simple, well-defined goal. GuideRL became a lightweight, strategic expert whose only job was to solve the prediction problem. NavRL became a navigation agent whose only job was to follow a goal, while evading obstacles.
- It simplified the learning problem. GuideRL, with a small state space (15 dimensions) and a simple MLP architecture, was able to train to 99% accuracy in approximately 5 min. It became a fast, reliable predictor of user intent.
- It enabled knowledge re-use. The NavRL policy was, in essence, a slightly more complex version of the already successful SU agent. By providing it with a clean, stable goal pose from GuideRL, we transformed the complex MU problem into a simple navigation task that we had already proven our architecture could solve.

The success of this decomposition is a key takeaway. It demonstrates that complex, multi-objective robotics tasks can often be made tractable by intelligently separating high-level strategic decision-making from low-level position control. The exceptional safety and responsiveness metrics seen in the MU evaluation are a direct result of this decoupled design.

6.2.2 Iterative Development Process

The final, successful DRL architecture was not an initial design, but the result of an iterative development process documented in Section 4.4. Our initial "start simple" hybrid approach, which attempted to use DRL for high-level goal selection and the ROS navigation stack for low-level control, failed. It highlighted the critical need for high-frequency, agile control, which the deliberative ROS stack could not provide. This led us to an end-to-end continuous control paradigm. Our subsequent attempts to process environment data using a rotation-invariant local occupancy grid and an autoencoder also failed. This iteration demonstrated that while the agent could detect obstacle proximity, it lacked the crucial directional awareness needed for stable navigation, resulting in erratic angular velocities. This failure led to the adoption of 2D LiDAR as the primary sensor input, as it intrinsically provides this directional reference. The final key challenge was processing this high-dimensional (720-ray) input. While a simple sampled-and-clamped LiDAR representation showed initial promise, it was the adoption of the 1D CNN that provided the robustness to handle all obstacle types by processing the full-resolution scan.

This methodological journey was critical in identifying the four key design choices that ultimately led to success:

- End-to-end continuous control to achieve the reactivity required for a dynamic environment.
- Raw, direction-aware sensor input (LiDAR) to provide rotationally-aware environmental perception.
- A robust network (1D CNN) to effectively process high-dimensional sensor data and avoid edge-case failures.
- Hierarchical task decomposition (GuideRL/NavRL) to manage the "conflict of objectives" in the complex MU scenario.

6.2.3 Generalizability

The applicability of the proposed DRL method extends beyond haptic feedback, offering a generalized framework for any HRI scenario where a mobile agent must maintain close proximity to a human user without obstructing their movement. For instance, consider an autonomous luggage assistant in a busy airport terminal. In this scenario, the robot must proactively follow a specific traveler or group of travelers to ensure their belongings are always accessible (responsiveness), yet it must also navigate a chaotic environment filled with other passengers and static obstacles (safety). The GuideRL policy could be adapted to predict who is most likely to access their belongings next, while NavRL ensures the robot moves through the crowd without collisions. Similarly, in industrial or workshop settings, a robot could carry heavy tools or parts for technicians working

around a large vehicle. Rather than waiting in a fixed location, the robot would use proactive positioning to be close to the technicians, anticipating their next moves to present the necessary equipment immediately, thereby reducing physical strain and delays. Even in retail environments, an automated shopping assistant could track a customer's gaze and position, ensuring the cart is always close by for placing items. In all these cases, the core challenge of balancing the need for immediate availability while ensuring safety in a shared space is identical to the problem solved in this thesis.

CHAPTER 7

Limitations and Future Work

Despite the promising results, this study has several key limitations that must be acknowledged and that frame the scope of the findings.

Simulation-to-Real Gap The entire study was conducted in a simulation. While noise was added to the LiDAR sensor to improve robustness, this does not fully capture the complexities of the real world. Unmodeled variables such as network/sensor latency, imperfect motor responses, and the significant inertia of a 130kg robot (versus the ideal holonomic controller in simulation, see Figure 7.1) will present major challenges. Therefore, the policy cannot be expected to work on the physical robot without further modification. Instead, it will be a significant undertaking, requiring extensive safety-constrained testing, retraining, and fine-tuning to bridge the Simulation-to-Real gap. This thesis provides a strong and validated baseline for this challenge. To address the Simulation-to-Real gap in the holonomic controller, as shown in Figure 7.1, a DNN-based model could be developed. This model would be trained with supervised learning on a dataset of the physical robot's real-world manual operation, learning to predict its actual movement dynamics and thus closing the gap.

Furthermore, in the real world, haptic requests occur randomly with a varying interaction duration and might be triggered before the user stops their movement. This could pose a problem to the proposed system that expects rigid and structured haptic request timings for reproducible conditions. Incorporating this stochasticity into the train and test framework could improve the transition to the real system.

Human Motion Dataset The policies were trained on a dataset from 3 expert participants. It is unknown how the policies would react to the novel or highly erratic movements of a naive user, such as running at speeds over 2 m/s, which were not present in the training trajectories or test trajectories. Furthermore, there were no recordings done with two users colocated in the same environment. Instead, random single-user

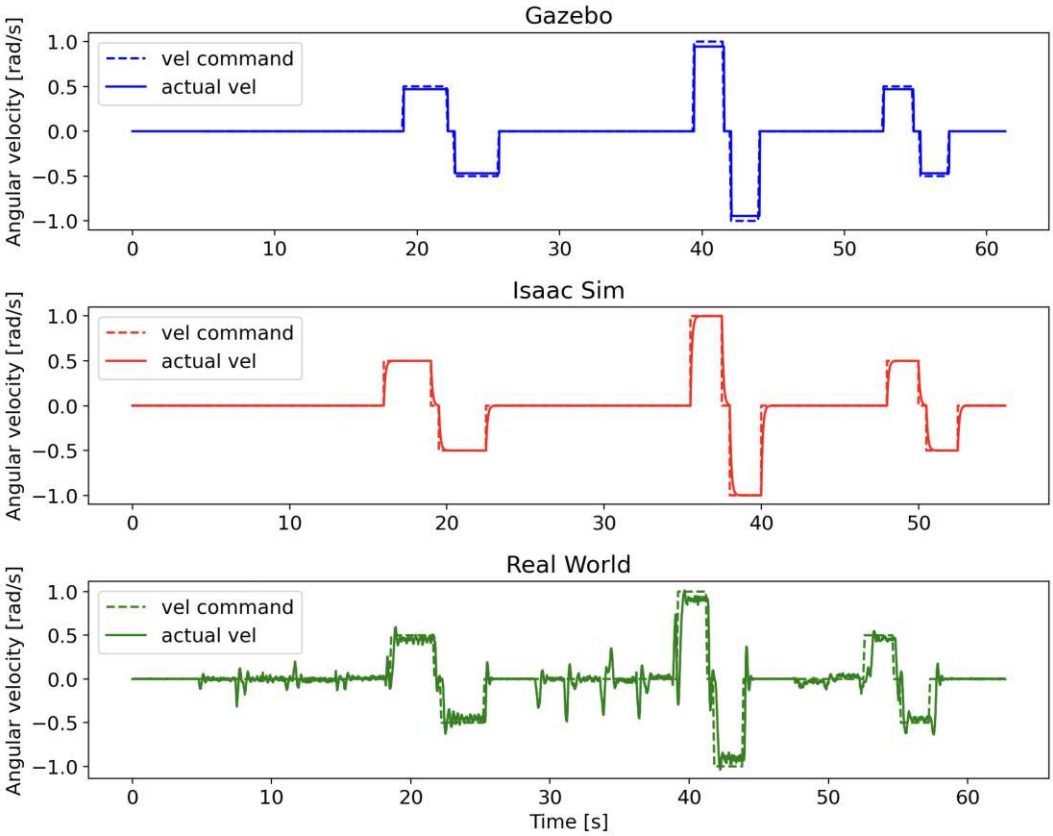


Figure 7.1: Comparison of issued angular velocity commands and actual recorded velocities between Gazebo, Isaac Sim, and real-life. The robot velocities in real life were tracked with the same Qualysis optical cameras used for recording the human motion dataset. This comparison highlights the significant gap between simulated and real-life velocity data.

trajectories were paired, which occasionally caused unnatural movements (users moving into each other). A new multi-user human motion dataset, with more diverse participants, would enhance the generalizability of the proposed method.

Generalization of Collision Avoidance While the agent was trained in diverse environments to promote generalization, the final evaluation was conducted in a single, fixed environment replicating the laboratory. The agent's ability to navigate a completely novel, unseen environment with a different obstacle layout has not yet been formally tested.

Partial DRL Control The DRL agent is not a complete, fail-safe system. It only governs the *Chase* behaviour. The system's safety and functionality are still critically dependent on traditional, ROS-based methods for the *Escape* and *Haptic* behaviours.

The DRL-based approach itself has no final-stage safety guarantees. Future work should explore replacing the *Escape* and *Haptic* behaviours with DRL agents. A DRL-based *Escape* policy could offer significantly higher reactivity and lower latency than the current ROS-based planner, which is subject to path-planning overhead. A DRL-based *Haptic* policy could learn complex, high-precision arm movements. By incorporating user-intent prediction (e.g., from GuideRL), such a policy could even begin the robot arm's movement before the haptic request is explicitly made, further reducing the total arm positioning time T_{arm}.

Scalability of the MU method The current GuideRL/NavRL architecture is constrained by a fixed input size hard-coded for $N = 2$ users. Scaling this framework to handle a dynamic, arbitrary number of users ($N > 2$) requires investigating flexible network architectures, such as those utilizing attention mechanisms, rather than naively truncating the state space to the nearest neighbors. However, this truncation to the nearest neighbours could prove to be a valid strategy. Further investigation into the scalability of the MU method remains open for future work. Additionally, future iterations of the GuideRL policy could significantly improve haptic intent modeling by incorporating richer state modalities, including user gaze and hand-pose estimation.

CHAPTER 8

Conclusion

Safely and effectively positioning a mobile Encountered-Type Haptic Display (mETHD) in shared spaces with HMD-wearing users presents a critical challenge. The primary objective of this thesis was to develop a safe, responsive, and robust positioning system for the CoboDeck robot, designed to replace existing methods and extend functionality to complex, multi-user VR scenarios.

An end-to-end deep reinforcement learning framework was developed to control the CoboDeck mobile base. This solution utilizes a 1D Convolutional Neural Network (CNN) to process 2D LiDAR data for robust collision avoidance and features a novel hierarchical decomposition of policies (GuideRL and NavRL) to successfully manage the multi-user challenge. To evaluate its performance against traditional methods, the DRL approach was compared in simulation against two baselines: static standby positioning and a heuristic algorithm.

The results demonstrated conclusively that the DRL-based approach can outperform traditional methods. The DRL-based approach proved to be significantly safer, reducing safety-critical *Escape* events by over 88% compared to both static and heuristic baselines. It was also significantly more responsive, cutting haptic positioning times by up to 6x. This superior performance was enabled by the agent's high-frequency proactive control. Unlike traditional methods that require large safety buffers to compensate for planning latency, the DRL policy can safely maintain a position close to the haptic interaction point. This responsiveness came at the logical and acceptable cost of reduced motion efficiency, as the agent was deliberately optimized to prioritize safety and low-latency interaction over energy conservation.

This thesis successfully validates the use of deep reinforcement learning for dynamic, close-proximity human-robot interaction in immersive environments. The findings and methodologies presented here lay the groundwork for more intelligent, proactive robotic

systems that can operate safely alongside humans, offering a robust framework for safe navigation in general collaborative robotics, independent of VR applications.

Overview of Generative AI Tools Used

Google Gemini served as the primary AI assistant, utilized for formatting LaTeX tables, improving English sentence structure, and generating the German abstract (with heavy editing). The text was further checked using the free version of Grammarly. ChatGPT, included with GitHub Copilot, aided in generating simple scripts in Python for plotting data. All AI suggestions were critically evaluated and revised before inclusion.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] E.-L. Sallnäs and K. Rassmus-Gröhn *et al.*, "Supporting Presence in Collaborative Environments by Haptic Force Feedback," *ACM Trans. Comput.-Hum. Interact.*, vol. 7, no. 4, p. 461–476, 2000.

[2] T. Novacek and M. Jirina, "Overview of Controllers of User Interface for Virtual Reality," *PRESENCE: Virtual and Augmented Reality*, vol. 29, pp. 37–90, 2020.

[3] S. J. Biggs and M. A. Srinivasan, "Haptic interfaces," in *Handbook of virtual environments*, pp. 133–156, 2002.

[4] H. G. Hoffman and A. Hollander *et al.*, "Physically Touching and Tasting Virtual Objects Enhances the Realism of Virtual Experiences," *Virtual Reality*, vol. 3, no. 4, pp. 226–234, 1998.

[5] R. Kovacs and E. Ofek *et al.*, "Haptic PIVOT: On-Demand Handhelds in VR," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, p. 1046–1059, Association for Computing Machinery, 2020.

[6] D. Schmidt and R. Kovacs *et al.*, "Level-Ups: Motorized Stilts that Simulate Stair Steps in Virtual Reality," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, p. 2157–2160, Association for Computing Machinery, 2015.

[7] R. Gomi and R. Suzuki *et al.*, "InflatableBots: Inflatable Shape-Changing Mobile Robots for Large-Scale Encountered-Type Haptics in VR," in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, Association for Computing Machinery, 2024.

[8] K. Hirota and M. Hirose, "Development of Surface Display," in *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp. 256–262, 1993.

[9] Y. Yokokohji and R. Hollis *et al.*, "What You can See is What You Can Feel-Development of a Visual/Haptic Interface to Virtual Environment," in *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*, pp. 46–53, 1996.

[10] S. Mortezapoor and K. Vasylevska *et al.*, "CoboDeck: A Large-Scale Haptic VR System Using a Collaborative Mobile Robot," in *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pp. 297–307, 2023.

[11] R. G. Prasuna and S. R. Potturu, "Deep Reinforcement Learning in Mobile Robotics – A Concise Review," *Multimedia Tools and Applications*, vol. 83, no. 28, pp. 70815–70836, 2024.

[12] K. Yan and J. Gao *et al.*, "Deep Reinforcement Learning Based Mobile Robot Navigation Using Sensor Fusion," in *2023 42nd Chinese Control Conference (CCC)*, pp. 4125–4130, 2023.

[13] F. Quiroga and G. Hermosilla *et al.*, "Position Control of a Mobile Robot through Deep Reinforcement Learning," *Applied Sciences*, vol. 12, no. 14, 2022.

[14] "Isaac Sim - Robotics Simulation and Synthetic Data Generation | NVIDIA Developer." `https://developer.nvidia.com/isaac/sim`[Online; accessed 2025-06-27].

[15] M. Mittal and C. Yu *et al.*, "Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.

[16] J. Schulman and F. Wolski *et al.*, "Proximal Policy Optimization Algorithms," 2017.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. A Bradford Book, 2018.

[18] J. Schulman and P. Moritz *et al.*, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," 2015.

[19] J. Schulman and S. Levine *et al.*, "Trust Region Policy Optimization," pp. 1889–1897, PMLR, 2015.

[20] L. Kästner and J. Lil *et al.*, "Enhancing Navigational Safety in Crowded Environments using Semantic-Deep-Reinforcement-Learning-based Navigation," in *2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 87–93, 2022. ISSN: 2475-8426.

[21] N. Rudin and D. Hoeller *et al.*, "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning," in *Proceedings of the 5th Conference on Robot Learning*, pp. 91–100, PMLR, 2022.

[22] OpenAI and J. Achiam *et al.*, "GPT-4 Technical Report," 2024.

[23] S. Mortezapoor and M. Ghazanfari *et al.*, "Safety for Mobile Encountered-Type Haptic Devices in Large-Scale Virtual Reality," *Frontiers in Virtual Reality*, vol. Volume 6 - 2025, 2025.

[24] P. Schillinger and S. Kohlbrecher *et al.*, "Human-robot Collaborative High-level Control with Application to Rescue Robotics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2796–2802, 2016.

[25] J. M. Zutell and D. C. Conner *et al.*, "ROS 2-Based Flexible Behavior Engine for Flexible Navigation," in *SoutheastCon 2022*, pp. 674–681, 2022.

[26] M. Quigley and B. Gerkey *et al.*, "ROS: an open-source Robot Operating System," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[27] S. Macenski and T. Foote *et al.*, "Robot Operating System 2: Design, Architecture, and Uses in the Wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[28] W. McNeely, "Robotic Graphics: a new Approach to Force Feedback for Virtual Reality," in *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp. 336–341, 1993.

[29] V. R. Mercado and M. Marchal *et al.*, ""Haptics On-Demand": A Survey on Encountered-Type Haptic Displays," *IEEE Transactions on Haptics*, vol. 14, no. 3, pp. 449–464, 2021.

[30] E. Vonach and C. Gatterer *et al.*, "VRRobot: Robot Actuated Props in an Infinite Virtual Environment," in *2017 IEEE Virtual Reality (VR)*, pp. 74–83, 2017.

[31] Y. Kim and S. Kim *et al.*, "Synthesizing the Roughness of Textured Surfaces for an Encountered-Type Haptic Display Using Spatiotemporal Encoding," *IEEE Transactions on Haptics*, vol. 14, no. 1, pp. 32–43, 2021.

[32] V. R. Mercado and T. Howard *et al.*, "Alfred: the Haptic Butler On-Demand Tangibles for Object Manipulation in Virtual Reality using an ETHD," in *2021 IEEE World Haptics Conference (WHC)*, pp. 373–378, 2021.

[33] M. Takagi and J. Arata *et al.*, "A new Desk-Top Encounter-Type Haptic Device with an Actively Driven Pen-Tablet LCD Panel," *Advanced Robotics*, vol. 27, no. 6, pp. 407–415, 2013.

[34] T. Batik and H. Brument *et al.*, "Shiftly: A Novel Origami Shape-Shifting Haptic Device for Virtual Reality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 31, no. 5, pp. 2331–2341, 2025.

[35] T. Ikeda and K. Fujita *et al.*, "LoopBot: Representing Continuous Haptics of Grounded Objects in Room-scale VR," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, Association for Computing Machinery, 2024.

[36] M. Hoppe and P. Knierim *et al.*, "VRHapticDrones: Providing Haptics in Virtual Reality through Quadcopters," in *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia*, p. 7–18, Association for Computing Machinery, 2018.

[37] P. Abtahi and B. Landry *et al.*, "Beyond The Force: Using Quadcopters to Appropriate Objects and the Environment for Haptics in Virtual Reality," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, p. 1–13, Association for Computing Machinery, 2019.

[38] Y. Yixian and K. Takashima *et al.*, "ZoomWalls: Dynamic Walls that Simulate Haptic Infrastructure for Room-scale VR World," 2020.

[39] R. Suzuki and H. Hedayati *et al.*, "RoomShift: Room-scale Dynamic Haptics for VR with Furniture-moving Swarm Robots," 2020.

[40] Y. Wang and Z. T. Chen *et al.*, "MoveVR: Enabling Multiform Force Feedback in Virtual Reality using Household Cleaning Robot," 2020.

[41] X.-T. Truong and T. D. Ngo, "Toward Socially Aware Robot Navigation in Dynamic and Crowded Environments: A Proactive Social Motion Model," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 4, pp. 1743–1760, 2017.

[42] R. Kirby and R. Simmons *et al.*, "COMPANION: A Constraint-Optimizing Method for Person-Acceptable Navigation," in *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 607–612, 2009. ISSN: 1944-9437.

[43] E. A. Sisbot and L. F. Marin-Urias *et al.*, "A Human Aware Mobile Robot Motion Planner," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 874–883, 2007.

[44] S. Bultmann and R. Memmesheimer *et al.*, "Anticipating Human Behavior for Safe Navigation and Efficient Collaborative Manipulation with Mobile Service Robots," 2025. arXiv:2410.05015 [cs].

[45] A. Eirale and M. Martini *et al.*, "Human Following and Guidance by Autonomous Mobile Robots: A Comprehensive Review," *IEEE Access*, vol. 13, pp. 42214–42253, 2025.

[46] J. Ahn and M. Kim *et al.*, "Formation-Based Tracking Method for Human Following Robot," in *2018 15th International Conference on Ubiquitous Robots (UR)*, pp. 24–28, 2018.

[47] A. Eirale and M. Martini *et al.*, "Human-Centered Navigation and Person-Following with Omnidirectional Robot for Indoor Assistance and Monitoring," *Robotics*, vol. 11, no. 5, 2022.

[48] A. Eirale and M. Martini *et al.*, "RL-DWA Omnidirectional Motion Planning for Person Following in Domestic Assistance and Monitoring," in *2023 9th International Conference on Automation, Robotics and Applications (ICARA)*, pp. 86–90, 2023. ISSN: 2767-7745.

[49] Y. F. Chen and M. Liu *et al.*, "Decentralized Non-Communicating Multiagent Collision Avoidance with Deep Reinforcement Learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 285–292, 2017.

[50] Y. F. Chen and M. Everett *et al.*, "Socially Aware Motion Planning with Deep Reinforcement Learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1343–1350, 2017. ISSN: 2153-0866.

[51] M. Everett and Y. F. Chen *et al.*, "Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3052–3059, 2018. ISSN: 2153-0866.

[52] B. Xue and M. Gao *et al.*, "Crowd-Aware Socially Compliant Robot Navigation via Deep Reinforcement Learning," *International Journal of Social Robotics*, vol. 16, no. 1, pp. 197–209, 2024.

[53] S. S. Samsani and M. S. Muhammad, "Socially Compliant Robot Navigation in Crowded Environment by Human Behavior Resemblance Using Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5223–5230, 2021.

[54] L. Kästner and B. Fatloun *et al.*, "Human-Following and -guiding in Crowded Environments using Semantic Deep-Reinforcement-Learning for Mobile Service Robots," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 833–839, 2022.

[55] M. Andrychowicz and A. Raichuk *et al.*, "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study," 2020.

[56] Qualisys AB, Gothenburg, Sweden, "Qualisys Motion Capture System," 2025.

[57] B. O. Community, "Blender - a 3D Modelling and Rendering Package," 2018.

[58] A. Serrano-Muñoz and D. Chrysostomou *et al.*, "skrl: Modular and Flexible Library for Reinforcement Learning," *Journal of Machine Learning Research*, vol. 24, no. 254, pp. 1–9, 2023.

[59] M. M. Zarrar and Q. Weng *et al.*, "TinyLidarNet: 2D LiDAR-based End-to-End Deep Learning Model for F1TENTH Autonomous Racing," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[60] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, an open-source Multi-robot Simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154 vol.3, 2004.