Reinforcement Learning for Shared Autonomy in Powered Wheelchair Navigation

A thesis submitted for the degree of
Doctor of Philosophy in Electronic Engineering

Sotirios Chatzidimitriadis

University of Kent
School of Engineering

January 2023

Page Count: 165

Abstract

Assistive robotics is witnessing a surge in research focusing on designing algorithms and frameworks that offer personalized support to users, considering their intentions and adapting system responses accordingly. This thesis delves into the integration of artificial intelligence in assistive technologies for powered wheelchairs, with a primary emphasis on the challenging problem of shared control through reinforcement learning.

Shared control, also known as shared autonomy, has been extensively studied, especially in the context of powered wheelchairs. Many wheelchair users rely on aid to enhance their everyday autonomy, particularly those who cannot use conventional joystick control interfaces, often encountering frustrations, fatigue, and compromised safety. Existing shared control methods typically involve blending human and autonomous controller decisions or predicting user goals to act autonomously. Unfortunately, such approaches often rely on assumptions like known goal sets, world dynamics models, and user behavior models, which limit adaptability. Motivated by the shortcomings of prior approaches and inspired by recent machine learning advances, this thesis introduces a novel shared control method using deep reinforcement learning within a continuous action space, which lifts the reliance on the aforementioned assumptions.

Initially, a reinforcement learning agent is developed to autonomously navigate complex indoor environments without the need for a map. The agent is trained using a virtual robotic wheelchair and rigorously validated against popular path planning methods. Subsequently, artificial noise is injected into the learned model to simulate disabled user input, enabling the training of an end-to-end shared control system. A modification in the typical reinforcement learning objective ensures compliance with user intentions while simultaneously maximizing future rewards associated with the assistive nature of the system. The shared control system receives noisy user commands and sensor data to generate corrective control commands for the wheelchair. Rigorous simulations and real-world trials with human users demonstrate significant reductions in collisions and increased obstacle clearance, albeit with a trade-off in user satisfaction.

Additionally, this thesis presents a non-intrusive, vision-based head-control interface for powered wheelchairs, employing face detection and head pose estimation. Through human user trials, the effectiveness and performance of this interface are benchmarked, confirming its viability as an alternative to the standard joystick interface. Notably, when combined with the shared control system in further real-world trials, the proposed assistive system proves adept at compensating for the less accurate input of this more challenging interface, resulting in a remarkable 92% reduction in collisions and improved overall adequacy.

In summary, this thesis introduces a mapless autonomous navigation method for powered wheelchairs, a novel shared control framework employing deep reinforcement learning, and a non-intrusive vision-based head-control interface. The proposed assistive system is empirically validated, showcasing its substantial impact on enhancing user autonomy and safety in powered wheelchairs.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Konstantinos Sirlantzis, for offering me the opportunity to work on this fascinating and meaningful subject for my PhD. His continuous support and guidance have been invaluable throughout my research journey.

I want to extend my thanks to my friends and colleagues, especially Paul, Saber, Odysseas, and Michael, for their support, camaraderie, and the memorable moments we shared along the way. Your presence made this challenging journey much more enjoyable. A special shoutout to my partner in crime, Antonis, with whom we navigated the highs and lows together during our parallel PhD adventures. And, of course, I am deeply thankful to Eleni for being a constant source of support and guidance, always there when I needed it.

To my sister Zinovia and brother Anestis, thank you for always being there with your positive vibes, humor, and endless encouragement. I hereby declare myself the wisest sibling of all!

My heartfelt gratitude goes to Poppy, the true MVP of patience and support. From enduring my rants about technical challenges to helping me overcome non-technical hurdles, your listening ear and invaluable insights have been a lifeline during difficult times.

Lastly, my deepest appreciation goes to my parents. Your unwavering love, support, and encouragement have been the driving force behind my achievements and personal growth. I am forever grateful for the values and lessons you have instilled and keep instilling in me.

To everyone mentioned here, as well as others who have been part of my journey in their own ways, I sincerely thank you. Each of you has left an indelible mark on my PhD experience, which is what made it unique and forever memorable.

This is to certify that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgements or in footnotes, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a degree.

Sotirios Chatzidimitriadis

Contents

Acronyms

AI	Artificial Intelligence
APF	Artificial Potential Field
CNN	Convolutional Neural Network
DLAFF	Dynamic Localized Adjustable Force Field
DQN	Deep Q-Network
DWA	Dynamic Window Approach
EPW	Electric Powered Wheelchair
IMU	Inertial Measurement Unit
KL	Kullback-Leibler
MDP	Markov Decision Process
MLP	Multilayer Perceptron
RL	Reinforcement Learning
ROS	Robot Operating System
SAC	Soft Actor-Critic
SLAM	Simultaneous Localization and Mapping
TEB	Time Elastic Bands
WST	Wheelchair Skills Test

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter introduces the core focus of the thesis: the development of assistive technologies for powered wheelchairs using machine learning methods. It highlights the motivation behind this research and presents the key research questions and contributions. Additionally, an overview of the thesis structure is provided to guide readers through the subsequent chapters.

1.1 Background and Motivation

The field of robotics has the potential to significantly enhance our lives, both industrially and domestically, by providing support for physical and cognitive tasks. Over the past decades, technological advancements have greatly improved the capabilities of robots, enabling them to perform complex computations and interact more effectively with the world through advanced sensors and actuators. Concurrently, the field of *Artificial Intelligence* (AI) has undergone a remarkable revolution, leading to the emergence of Artificially Intelligent Robots. These robots surpass traditional programming limitations and can undertake intricate tasks, making decisions beyond predefined instructions. As technology progresses, we can envision robots collaborating with humans as trusted partners in a wide range of tasks.

Within this context, assistive robotics has gained considerable traction in recent

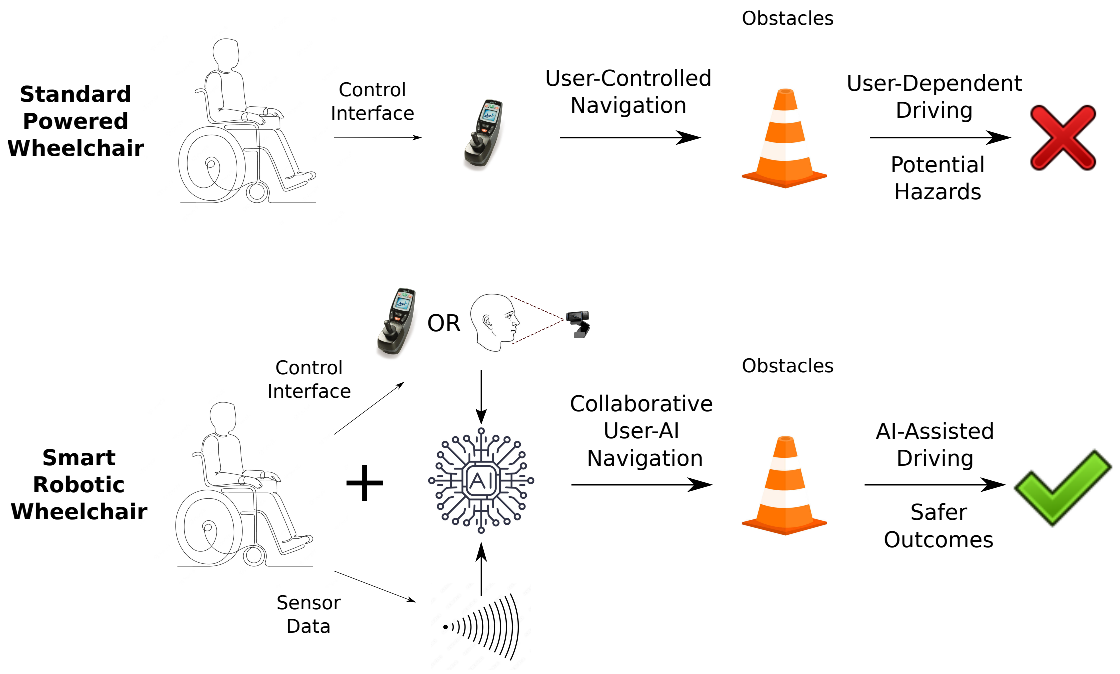


Figure 1.1: From Standard to Smart: Augmenting Powered Wheelchair Navigation. The figure showcases the transformation from a conventional powered wheelchair, navigated using a joystick with potential risks, to a smart robotic wheelchair. The smart wheelchair utilizes sensors, intelligent algorithms, and an option of a head-control interface, enabling collaborative control with AI assistance. By incorporating environmental data and intelligent decision-making, the smart robotic wheelchair enhances safety during navigation, providing personalized assistance to the user.

years. This vast field, which has been the subject of ongoing research for decades, holds the potential to positively impact a significant portion of the population and society as a whole. Powered wheelchairs, as mobile assistive robots, play a particularly crucial role in enhancing the independence, self-esteem, and social participation of individuals with physical disabilities. With factors such as the rapidly rising percentage of the geriatric population and the prevalence of chronic diseases, the global market for powered wheelchairs is expected to experience substantial growth in the coming years. Recent reports estimate that the disabled assistive devices market will grow at a compound annual growth rate (CAGR) of 5.5% from 2019 to 2026 [1], and the powered wheelchair market will register a CAGR of over 6.3% between 2022 and 2030 [2]. These statistics underscore the importance of relevant technologies and the need for further improvements.

Powered wheelchairs can be categorized into standard powered wheelchairs and *smart wheelchairs*, the latter being equipped with additional capabilities beyond manual mobility to provide further assistance and enhance patient independence and comfort [3]. This augmentation involves the use of various sensors, specialized electronics, and computer algorithms, all of which are necessary to create *assistive technologies* for robotic wheelchairs. Fig. 1.1 illustrates the transformation from a conventional powered wheelchair to a smart robotic wheelchair, enabling collaborative control with AI assistance, aimed at enhancing navigation safety and providing personalized assistance to the user. Although assistive technologies encompass a wide range of functions, such as communication aids, posture improvement, and obstacle-alerting systems, this thesis focuses on the primary function of navigation and control, as many wheelchair users face challenges in this regard due to physical and/or cognitive impairments.

Specifically, this thesis centers on the problem of *Shared Autonomy* (or Shared Control) within the context of assistive mobile robots. Shared Autonomy refers to a setup that incorporates inputs from both the user and the robotic system, executing control actions that optimize the situation while aligning with the user's intentions and behavior [4, 5]. The user and the assistive robot collaborate to achieve a common navigation goal, with an emphasis on safety and user comfort. Another aspect addressed in this thesis concerns the type of input provided by the user. While the majority of powered wheelchair users rely on a joystick for control, individuals with arm movement impairments must utilize alternative interfaces to operate their wheelchairs.

Existing works on shared control solutions often make several assumptions, such as a known set of possible goals, a known dynamics model of the world, and a known user behavior model [6, 7, 8]. Moreover, many of these methods blend human input with suggestions from conventional path planners, often lacking certain safety guarantees or flexibility in their objectives [9]. On the other hand, existing alternative control interfaces, like head-array switches or the Sip 'n Puff switch [10], are typically invasive methods that are cumbersome to use and offer limited control resolution. Consequently, there is a pressing need to improve the current state-of-the-art.

Machine learning, which has shown tremendous promise in recent years and is poised to transform the field of robotics, remains underutilized in assistive technologies for powered wheelchairs. This thesis proposes the utilization of the Reinforcement Learning (RL) framework, a branch of machine learning for solving sequential decision-making problems, to develop a novel shared-control system that departs from conventional methods while addressing several of their limitations. The proposed method is a self-learning system that can be trained in simulation without requiring a human user. It can subsequently be transferred to a real wheelchair to augment the user's continuous stream of inputs by leveraging information derived from the smart wheelchair's onboard sensors. Additionally, this thesis introduces a novel vision-based interface for controlling a wheelchair through head movements. This non-invasive, low-cost, and intuitive interface can be combined with the shared-control system to enhance usability and safety. The proposed systems are thoroughly analyzed and validated through real-world experimentation.

1.2 Research Aim and Objectives

Motivated by the significant potential of assistive robotics and the remarkable advancements in deep learning, this thesis aims to leverage machine learning progress to enhance the state-of-the-art in assistive technologies for powered wheelchairs.

Specifically, this work focuses on the development of a novel shared-control system for wheelchair driving. The system aims to minimize assumptions about the user and the environment while maintaining flexibility in optimizing objectives. To achieve this, the thesis proposes the use of reinforcement learning, as its formulation as a sequential decision-making framework is well-suited for this task. Furthermore, within the same framework, this thesis investigates the viability of an autonomous navigation strategy formulated as an RL policy, which maps environmental observations to control actions. This approach seeks to improve certain aspects of conventional path planning, such as the reliance on maps and high computational demands. Importantly, as discussed in Chapter 4, this autonomous navigation model can be utilized in the training process of the shared-control system to simulate a wheelchair user, eliminating the need for actual human input.

Given the sample inefficiency typically associated with RL systems, the use of simulation environments for training becomes crucial, primarily due to their ability to provide low-cost and safe training samples. However, despite the remarkable achievements of RL applications in recent years, such as surpassing human performance in complex games like Go [11] and Dota 2 [12], and predicting protein structures [13], training reliable RL systems remains notoriously challenging, particularly when it comes to real-world application. Considering this, this thesis also explores the effectiveness of the proposed shared-control system, trained in a simulated environment with a simulated user, when transferred to a physical robot with human users.

Lastly, the thesis aims to enhance the adequacy of existing alternative control interfaces for wheelchair users with limited or no arm function by employing a computer vision approach and combining it with the proposed assistive system. To realize the aforementioned aims, this thesis establishes the following objectives:

1. Formulate an autonomous navigation strategy for robotic wheelchairs as a reinforcement learning problem, eliminating the need for a pre-existing map and relying solely on the robot's onboard sensors.
2. Utilize the drive actions of the trained autonomous navigation model to simulate a realistic wheelchair user model by introducing artificial noise. Incorporate this simulated user model into the training loop of a shared-control system, also based on the RL framework. The shared-control system should effectively assist the user in achieving their goals while respecting their intentions, without requiring knowledge of the user's behavior model or predefined goals.
3. Transfer the developed shared-control system to a physical powered wheelchair and assess its effectiveness through comprehensive validation.
4. Design an innovative vision-based head control interface for powered wheelchairs that is non-invasive, intuitive to use, and customizable to individual user preferences.

5. Enhance the adequacy of the head control interface by integrating it with the shared-control system, leveraging the capabilities of the shared-control system to improve usability, user experience, and overall performance.

1.3 Research Questions

Provided the objectives presented in the previous section, this thesis seeks to answer the following research questions concerning smart robotic wheelchairs:

- **Research Question 1:** Can we develop an intelligent system capable of mapless autonomous navigation in complex indoor environments, relying solely on onboard sensor data and processing? Can this system successfully execute the necessary maneuvers for proficient wheelchair driving?
- **Research Question 2:** Is it feasible to design and implement an intelligent system that takes into account user input and utilizes sensor data to enhance user driving in various environments with different goals? Can this system demonstrate effectiveness in both simulated and real-world environments?
- **Research Question 3:** How can we enhance existing alternative control interfaces, specifically through the design and implementation of a non-invasive head-control system that can be easily deployed on a powered wheelchair?
- **Research Question 4:** To what extent can the intelligent shared-control system improve the adequacy of alternative control interfaces? Can the integration of the shared-control system enhance the performance and usability of the head-control interface?

By addressing these research questions, this thesis aims to contribute valuable insights and advancements in the field of smart robotic wheelchairs, with a primary focus on the problem of shared control. The findings of this research will not only enhance our understanding and development of shared control techniques in the context of wheelchair navigation but also have the potential to extend and benefit other domains within the broader field of smart robotics and human-robot interaction.

1.4 Key Contributions

This section serves to highlight the notable contributions of the thesis, which can be summarized as follows:

1. **Autonomous Navigation**: The thesis presents an intelligent system for mapless autonomous navigation in complex indoor environments using onboard sensor data and processing. The trained autonomous agent surpasses human performance and conventional motion planners, demonstrating high success rates in executing maneuvers required for competent wheelchair driving. The proposed approach eliminates the need for a map, reduces computational demands, and can be applied to other non-holonomic mobile robots, contributing to advancements in the field of smart robotic wheelchairs and navigation for various robotic platforms.

2. **Shared Control**: Introducing a novel policy-based shared-control framework, this thesis presents a system that dynamically follows user intentions while optimizing its own objectives, without relying on preconceived assumptions about the user, environment, or task. With its inherent flexibility and adaptability to the user, task, and sensory input, the proposed system effectively assists users in reaching their goals and navigating safely. This significant contribution enhances the field of shared control and human-robot interaction, providing new insights and advancements in the field of shared control and human-robot interaction.

3. **Real-World Application**: The thesis demonstrates the practical application of the proposed shared-control system in a real-world scenario. The assistive model trained in simulation, and with a simulated user model, successfully transfers to a physical robot, assisting human users in achieving their goals with increased clearance and reduced collisions. This high-

lights the effectiveness and potential real-world application of reinforcement learning-based systems in the field of smart robotic wheelchairs.

4. **Vision-Based Head-Control Interface**: A non-intrusive and intuitive-to-use head-control interface is introduced, leveraging vision-based neural-network head pose estimation. The system only requires only a standard web camera and offers continuous turning angles, customization to individual range of motion, and compatibility with compact computers that can be installed on electric wheelchairs. Real-world trials validate the practicality and effectiveness of this head-control interface, contributing to the advancement of control interface design for wheelchair users, with a specific focus on individuals with quadriplegia.

5. **Enhanced Control Interface Adequacy**: By combining the head-control interface with the shared control system, the thesis demonstrates the improvement in control interface adequacy. The shared-control system significantly reduces the number of collisions, maintains a larger clearance, and improves completion times and user perception of control and safety, particularly for more challenging-to-use interfaces. This underscores the importance of coupling alternative control interfaces with the shared control system to enhance their adequacy and overall safety.

1.5 Publications

- Chatzidimitriadis, S., Oprea, P., Gillham, M., & Sirlantzis, K. (2017, September). Evaluation of 3D obstacle avoidance algorithm for smart powered wheelchairs. In 2017 seventh international conference on emerging security technologies (EST) (pp. 157-162). IEEE.

- Chatzidimitriadis, S., & Sirlantzis, K. (2022, May). Deep Reinforcement Learning for Autonomous Navigation in Robotic Wheelchairs. In International Conference on Pattern Recognition and Artificial Intelligence (pp. 271-282). Cham: Springer International Publishing. *

- Bafti, S. M., Chatzidimitriadis, S., & Sirlantzis, K. (2022). Cross-domain multitask model for head detection and facial attribute estimation. IEEE Access, 10, 54703-54712.

- Chatzidimitriadis, S., Bafti, S. M., & Sirlantzis, K. (2023). Non-Intrusive Head Movement Control for Powered Wheelchairs: A Vision-Based Approach. IEEE Access. *

Note: Publications marked with an asterisk (*) have directly contributed to the thesis.

1.6 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 presents the literature regarding autonomous navigation and shared control methods for mobile robots, as well as the relevant works that have been done specifically for powered wheelchairs. Chapter 3 establishes some basic concepts of reinforcement learning and based on that framework it introduces an autonomous navigation method, which is developed and tested in simulation and is benchmarked against conventional navigation methods and human performance. Chapter 4 introduces a novel shared control framework, that makes use of the autonomous method of Ch. 3 and modifies the objective of maximum entropy reinforcement learning so that the resulting system is compliant with the user. The system is developed and validated in simulation, but is also transferred to a real powered wheelchair, where trials with human participants showcase its efficacy. Chapter 5 focuses on a vision-based control interface for powered wheelchairs. First, the chapter presents the relevant works that have appeared in the literature and discusses their limitations. Then, it describes the design of a novel head-control interface and validates its effectiveness through real-world trials. Lastly, the novel interface is combined with the shared control system of Ch. 4, where another set of real-world trials shows how the provided assistance can improve the usability of such an interface, which is less accurate compared to the standard joystick interface. Finally, Chapter 6 concludes the thesis by summarizing its contributions and by proposing future research venues.

CHAPTER 2

Autonomous and Semi-Autonomous Methods for Mobile Robots

This chapter presents an overview of existing research on autonomous and assisted navigation for Electric Powered Wheelchairs (EPWs). It focuses on the primary components of these navigation systems and emphasizes relevant literature concerning powered wheelchairs. Additionally, the chapter explores machine learning approaches used in these systems, which align with the methodology used in subsequent technical chapters of the thesis. The literature review serves as a foundational resource for developing novel autonomous and assistive technologies for EPWs.

2.1 Introduction

A powered wheelchair serves as a crucial technology for individuals with mobility issues, enabling them to maintain their independence. The usability of a powered wheelchair plays a vital role in facilitating daily activities for its users. Hence, an EPW should strive to provide effortless operation, promote safety, and prevent fatigue and frustration throughout the day. Assistive technologies have emerged as solutions with these desired characteristics, offering enhanced capabilities and ease of use in powered wheelchairs, which individuals with disabilities typically prefer over more challenging alternatives [14].

To further enhance the wheelchair experience, comfortable seating and proper posture are also of paramount importance. However, this thesis primarily focuses on improving the mobility and control aspects of an EPW. Specifically, it aims to transform a standard powered wheelchair into a "smart robotic wheelchair" by leveraging various sensors and machine learning algorithms. These technologies enable the wheelchair to possess autonomous and semi-autonomous capabilities.

In this chapter, we begin by exploring autonomous navigation methods for mobile robots and the relevant research conducted in the context of powered wheelchairs. We then delve into the concept of *shared control*, which serves as the central theme of this thesis. We discuss prominent methods that have been applied to wheelchair control, establishing the foundation for the exploration of shared control. Additionally, we present a departure from conventional approaches and highlight the utilization of machine learning techniques in addressing the shared control problem. This alternative approach holds promise as a state-of-the-art solution for achieving wheelchair semi-autonomy.

2.2 Autonomous Robots

For a robot to be deemed as truly autonomous, it has to be able to navigate through the environment. In order to achieve that, the robot needs to solve the following questions: "Where am I?", "Where to go?" and "How to get there?". The foremost question, also called the problem of *localization*, is about determining the position of the robot on a given map, at any given time. "Where to go?" is concerned with the desired destination of the robot, which can be an input of various forms and from different interfaces, but needs to be mapped to a position on a map. Finally, "How to get there?" refers to the robot's capability of planning a feasible and efficient plan to the destination and executing it while avoiding obstacles. Figure 2.1 provides a high-level modular architecture for autonomous navigation in mobile robots, encompassing perception, localization, cognition, and motion control modules, each contributing to the robot's ability

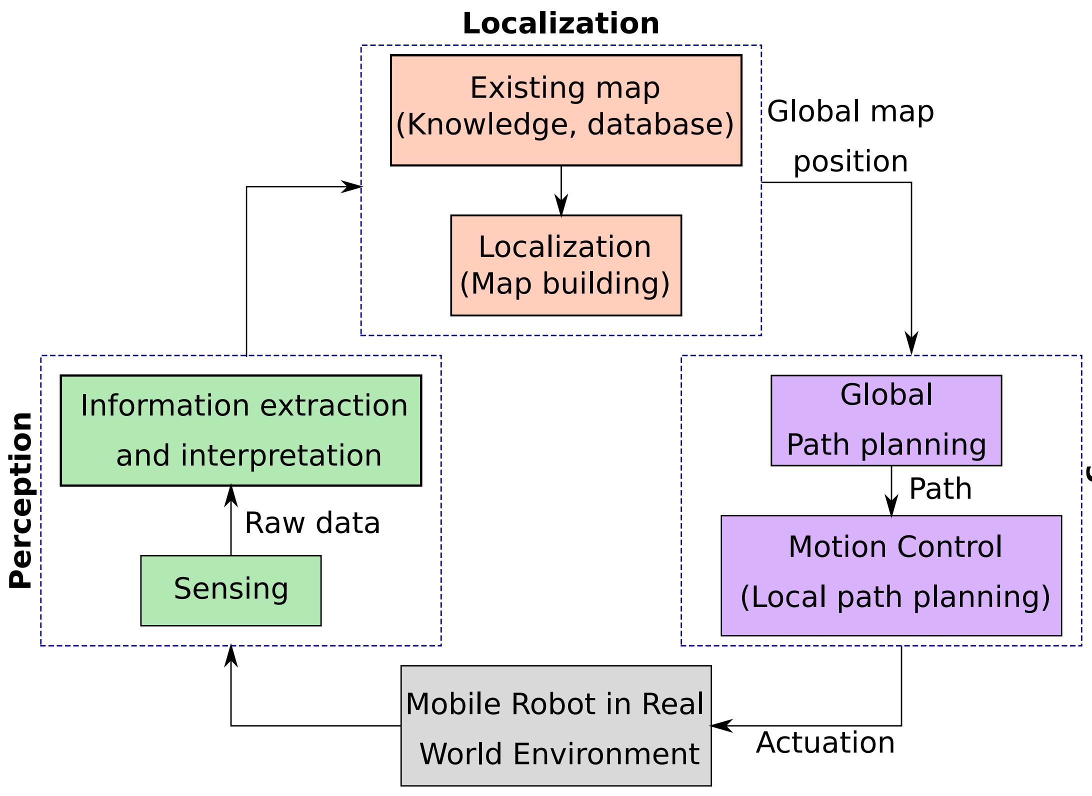

Localization
Existing map (Knowledge, database)
Localization (Map building)
Global map position
Perception
Information extraction and interpretation
Raw data
Sensing
Cognition
Global Path planning
Path
Motion Control (Local path planning)
Mobile Robot in Real World Environment
Actuation

Figure 2.1: Typical Control Structure of Autonomous Navigation in Mobile Robots. The figure showcases the high-level modular architecture for autonomous navigation in mobile robots. It includes perception for sensing and data interpretation, localization for map knowledge and positioning, cognition for global path planning, and motion control for local path execution, enabling efficient real-world navigation.

to answer these fundamental questions. In the following sections, we provide an overview of the state-of-the-art methods for localization, path planning and motion control, providing a base on the family of algorithms that can be used to make a robot autonomous.

2.2.1 Localization

As mentioned before, the term *localization* refers to figuring out where a robot is on a given map, which is an essential requirement to making decisions about future actions. The pose of the robot in space is defined by its location in the Cartesian Space, which can be described by its coordinates x, y, z, and its orientation, which can be described by the *Euler angles*, or equivalently the *roll* (r), *pitch* (p) and *yaw* (y) [15].

Normally, a mobile robot is equipped with sensors to monitor its own motion (*e.g.*, wheel encoders, inertial sensors), and when a kinematic model of the robot is available, it can be used to compute an estimate of its location relative to its starting position. This is known as *odometry* or *dead reckoning.* However, due to errors present in the sensor measurements, which are integrated over time, the location estimates become increasingly unreliable as the robot navigates the environment. To correct the errors in the dead reckoning estimates, localization algorithms can leverage information from the environment, gathered from additional sensors, and correlate them to the information contained in a map.

The formulation of the localization problem depends on the type of the sensors used to observe the environment, as well as the type of the map. Traditionally, the problem of map building and localization has been addressed with the following two approaches: 1) Map building and subsequent localization 2) Continuous map building and updating. In the first approach, a (usually teleoperated) robot goes through the environment to collect information from various positions and then processes it to build a map. Once the map is available, the robot can then collect information from its current position, and compare it with the constructed map to estimate its current pose. In the second approach, the robot is able to autonomously explore the environment and build a map, or update an existing one, while it is moving. A well-known family of algorithms that fit within this approach is Simultaneous Localization and Mapping (SLAM)[16].

The types of maps that represent the environment can be generally classified into two categories: *feature-based* and *location-based* [17]. Feature-based maps describe the shape of the environment at specific locations, which are the locations of objects contained in the map (*e.g.*, landmarks). On the other hand, location-based maps offer a label for any location in the world, which means that they contain information not only about where objects are in the environment, but also about their absence (*i.e.*, free space). One of the most widely-used location-based representations is the *occupancy grid map*, which provides a discretized representation of an environment, where each coordinate is assigned a binary value that indicates whether or not a location is occupied by an object.

Given the existence of a map, the robot can utilize its onboard range sensors (*e.g.*, laser rangefinder) to measure its distance from the objects of interest in a feature map or its distance from the nearest occupied regions in a grid map. As mentioned before, localization algorithms can use this information, along with the odometry input, to estimate the robot's location on the map. However, in the real world the sensor information is corrupted by noise, and therefore, it is necessary to estimate not only the robot's location, but also the uncertainty associated with it, since assuming perfect location estimates can lead to catastrophic consequences. For that reason, various probabilistic methods have been proposed to solve the localization problem.

One of the most well-known localization algorithms is *Markov localization*[18]. Markov localization is a straightforward application of the Bayes filters and works on both feature and grid maps. A special case of Markov localization is the *Extended Kalman filter localization* (EKF) algorithm[19], which is comparatively more efficient but only works on feature maps. A closely related algorithm is the *unscented Kalman filter localization* (UKF)[20], which uses the unscented transform to linearize the motion and measurement models instead of computing their derivatives, thus, providing an alternative and more accurate approach to considering the impact of noise on the estimation process. EKF and UKF work on the problem of *position tracking*, where it is assumed that the initial robot pose is known. The *multi-hypothesis tracking* filter (MHT)[21] is a more computationally expensive extension of the basic EKF that works on the more generic problem of *global localization*, where the initial robot pose is not known.

Other prominent algorithms that can solve the global localization problem are the *grid localization* and the *Monte Carlo localization* (MCL)[22]. Grid localization uses a histogram filter to represent the posterior belief over the pose space, and even though it works well, it is quite inefficient when applied on a fine-grained grid that is required for good results. MCL is a very popular algorithm that utilizes particle filters to represent the robot's belief over possible poses. An improvement upon the basic MCL is the *adaptive* MCL (AMCL), which can dynamically adjust the number of particles in the filter to trade-off between localization accuracy and processing speed.

Numerous variations of the algorithms presented above, as well as many other proposed localization techniques, can be found in the literature. Namely, other approaches include vision-based tracking systems, beacon-based and radio frequency schemes, converting the robot localization to dynamic optimization and applying *particle swarm optimization* (PSO)[23] and more. Localization is an as critical as hard problem in mobile robot navigation, the solution of which depends on the application domain and the environment of interest, as well as the available sensors and computing power.

2.2.2 Path Planning

Path planning is one of the most critical capabilities of an intelligent robot. The problem of path planning refers to finding an optimal, collision-free path to navigate from the current position to a goal position. Due to the huge volume of literature on conventional path planning, many different classifications can be made considering how the different algorithms function. However, a common division is between *global planners* and *local planners.*

Global planning algorithms are concerned with static and offline path planning, given prior knowledge of the environment (existence of a map), and also localization knowledge, where their goal is to calculate a path from the current location to a desired destination. In order to achieve that, these algorithms execute a search operation in a set of samples, that consists of the different states and configurations that are reachable by the robot, including the initial robot states and the goal states. Differing in their approach to executing the search, global planner algorithms include *graph search algorithms*, *sampling-based algorithms*, and *intelligent algorithms.*

As the name implies, graph search algorithms operate on graphs built upon maps of the environment, as described in the previous section, and are a discretized representation of all configurations reachable by the robot (also known as the *C-Space*). Graph search algorithms visit, fully or partially, this graph until they find a subset of samples that can form a path to connect the initial and the goal state. Widely used algorithms of this family include *Dijkstra's algorithm* [24], $A^\star$ [25], $D^\star$ [26], as well as *Any-angle algorithms* (e.g. $T^\star$ [27], Anya [28]), which can produce paths that are not restricted to specific orientations, since their waypoints do not have to be placed in neighboring nodes of the graph (in contrast to *Edge-restricted algorithms*).

Sampling-based algorithms also operate in the C-space, however, they focus on creating or modifying the samples within it iteratively. Even after a feasible path is discovered, they can keep working to find more optimal ones. A complex environment might require a large number of samples in order to approximate an optimal solution, thus demanding large memory resources. Two of the most prominent sampling-based algorithms are the Rapidly exploring Random Tree method (RRT) [29] and the Probabilistic Roadmap Method (RPM) [30]. RRT creates a tree structure, starting from an initial point, and extending it by sampling new points from the C-Space. If RRT detects that the goal point has been added to the tree, it can then backtrack the tree to its origin to recover the path. PRM starts with a series of C-space samples, and for each one of those initial samples, it creates a tree by generating more samples and attempting to connect them to their nearest neighbors. When the goal configuration is included in the resulting graph, a graph search method (like $A^\star$) is then used to find the shortest path in the roadmap.

Lastly, intelligent algorithms include nature-inspired approaches which utilize techniques that simulate biological evolution and insect foraging behaviors in nature to generate paths that result from the evolution of a population. The *Genetic Algorithm* (GA) [31], along with its variations, are based around individuals that contain binary genes, which encode solutions (*i.e.*, solving the path planning problem). Those individuals can reproduce, combine their genes and mutate, and given enough iterations, this evolution process leads to convergence of the algorithm. *Swarm Optimizers* use agents (usually modeled after animals) that move in free space, and through iterations they can create patterns that converge to paths leading to the goal. The *Particle Swarm Optimization* (PSO) [23] algorithm and the

Ant Colony Optimizer (AC0) [32] are prime examples of evolutionary algorithms.

2.2.3 Motion Planning

Motion planning is concerned with generating short, feasible robot trajectories that satisfy some criteria. The main criterion that all motion planning methods take into account is collision avoidance, which, as the name implies, considers trajectories that avoid colliding with obstacles. *Local motion planning algorithms* are used to calculate collision-free trajectories and can be used either standalone or on top of a global motion planner, while they focus on dynamic planning considering the local information of the robot's environment.

Local planners that react at each instant to the presence of obstacles, are also known as *Reactive planners* and need to be efficient in order to be able to produce a timely reaction, while they also require sufficient real-time feedback provided by the robot. A major approach for tackling the reactive planning problem is by utilizing fields, two prominent such methods being the *Artificial Potential Field*s (APFs)[33] and the *Vector Field Histogram* (VFH)[34]. In APF, the resulting motion of the robot moving in the artificial field is given by a sum of external virtual forces, which can be either repulsive or attractive. Naturally, surrounding obstacles generate repulsive forces, so that collision with them is prevented, while a goal position generates an attractive force so that the robot moves towards it. In order to tackle a major drawback of this method, which is getting stuck in local minima, various works have proposed combining APF with other methods, like evolutionary algorithms [35] or PSO [36]. VFH evaluates the density of obstacles around the robot by creating a polar histogram and then steers the robot towards the angle with the lowest density of obstacles. Another class of reactive planners are the *Bug algorithms* [37, 38, 39], which are among the earliest and simplest planners, and perform goal-seeking behaviors while only assuming local knowledge of the environment. Lastly, a very important reactive planner that needs to be mentioned is the *Dynamic Window Approach* (DWA) [40], which was designed to mitigate the disadvantages of APF. It works by computing a set of feasible velocities that can be reached in the next time step and then taking the subset of only the safe velocities, which are the ones that would not result in a collision. The algorithm evaluates a score of the trajectories of the sampled velocities (applied for a specified time frame), based on the speed of the robot, its clearance (proximity to obstacles), and also its orientation with respect to the goal. Finally, the velocity command associated with the highest-scoring trajectory is the one that is sent to the mobile base.

In the family of local planners, local optimization algorithms modify a pre-existing path according to the obstacles they encounter and place priority on computational efficiency rather than optimality. Elastic bands is a framework that has been proposed to close the gap between global and reactive planning [41], and works with deforming an initial collision-free path to maintain clearance from obstacles, as changes are detected in the environment. *Time Elastic Bands* (TEB) [42] is an extension to Elastic Bands that includes time constraints, using a weighted multi-objective optimization framework that gives it a high degree of flexibility.

In terms of artificial intelligence, *Fuzzy Logic* and *Machine Learning* have been utilized for producing collision-free paths in known or unknown scenarios. Fuzzy logic depends on the use of fuzzy rules to produce robust controllers [43] that can navigate through unstructured terrain [44] or avoid dynamic obstacles [45, 46]. Controllers trained with machine learning mainly include the use of neural networks. For example, Engedy [47] introduced a neural network planner, trained on data that were labeled with the potential field method, to avoid static and dynamic obstacles. Tai [48] utilized a deep convolutional neural network to perform obstacle avoidance, while the control commands acting as the labels during the training process were provided by a human operator. Other works combine fuzzy logic with neural networks, to train neuro-fuzzy controllers for robotic navigation [49, 50, 51].

One of the main arguments for using machine learning in motion planning, but also robotic systems in general, is its potential to automatically adjust a huge number of system parameters, that would otherwise need to be manually tuned, therefore making the system more robust and reducing the effort required by human experts. Such methods, which are a focus of this work, will be discussed in more detail in the following section, with an emphasis on reinforcement learning methods.

2.2.4 Machine Learning in Motion Planning

Within the broadness of the machine learning framework, many works have utilized *Supervised Learning* (SL) with deep neural networks, which has proven itself to be a powerful tool to classify and process data, for mobile robot navigation. An important work is presented in [52], where semantic information was extracted from images, through deep networks, to decide driving actions for an autonomous vehicle. Another major paradigm utilizing the SL framework is presented in [53], where authors mapped laser range findings and the target position to moving commands, via a deep learning model, and with Dijkstra's planner along with DWA acting as the expert for labeling the data. Producing such datasets, however, is costly, time-consuming, and prone to errors, which might negatively impact the training outcome.

Other popular methods include *imitation learning* (IL) and *Inverse Reinforcement Learning* (IRL), which both try to solve the problem of optimal control with a data-driven approach. IL is about learning an expert strategy, as a mapping between observations and actions (*e.g.*, neural network), by data provided by an expert. For example, Hussein *et al.* [54] learned a policy by combining demonstrations and active learning, to perform navigation tasks in a 3D simulated environment based on raw visual input. In IRL the goal is to infer a reward function from expert demonstrations, instead of directly learning the policy. A good example of IRL application is that of learned navigational policies in a socially adaptive manner [55, 56]. However, these methods heavily rely on high-quality demonstration data, which can be expensive and time-consuming to collect, while generalization to unseen paradigms is challenging. Other works combine SL with RL, as in [57], that trains a global planner by mimicking an expert, and a local planner that is trained with deep Q-learning. The switching strategy between the planners, though, is simplistic, the global planner can only navigate towards a specific number of goals, and both planners work with discrete actions.

In a pure RL framework, many works rely solely on vision to perform navigation. In [58] a feature vector is extracted from a raw image, which is fed as input to a RL agent to control a real slot car through four discrete actions. Zhu *et al.* [59] used a high-quality rendering simulation framework to perform visual navigation on a SCITOS robot, again considering four actions. The authors in [60] propose the use of auxiliary tasks to provide a denser training signal, while their implementation also works with a discrete action space, and depends on the capacity of a stacked Long short-term memory (LSTM) architecture which underperforms in large environments. Using depth instead of RGB images, increases the transferability of a learned policy in simulation to the real world, as shown in [61] where a navigational policy was learned through raw depth images. This work utilizes a rather large network and outputs discrete control commands. In other works, state representation learning (SRL) is utilized to capture useful and dense information from high-dimensional observations, such as in [62], where it is argued that learning successor features from images can improve knowledge transfer between tasks and speed up the training process. However, again this implementation simplified the problem by learning discrete actions and was not tested in complicated environments.

Zeng *et al.* [63] proposed the addition of a Gated Recurrent Unit (GRU) [64] layer to the typical architecture of the Asynchronous Advantage Actor Critic (A3C) [65] actor-network, in order to enhance it with a memory mechanism and improve the continuous navigation task in challenging environments. Tai *et al.* [66] is one of the few other works that learned a continuous control policy, aimed for a Kobuki-based turtlebot, and relying only on sparse 10-dimensional laser ranges, the robot's velocity, and the relative target position. Marchesini [67] suggests that the discrete action space is a viable solution for performing mapless navigation, however, the experiments are performed with a turtlebot in simple environments and only the angular velocity is taken into account, whereas the linear velocity is set to a constant value (also restricting static rotational movements). Finally, Francis *et al.* [68] trained an RL-based local planner and combined it with PRM as the global planner to provide waypoints for the RL agent.

The approach to autonomous navigation this thesis adopts is also based on the RL framework, which has not yet been utilized for powered wheelchair navigation, as will also be evident in the following section concerning autonomous wheelchairs. However, due to the characteristics associated with powered wheelchairs, the proposed approach needs to satisfy some requirements that are lacking from the relevant literature presented in this section. These are 1) Use of the continuous action space to enable fine maneuvering 2) A sensor arrangement that can cover a bulky robot, such as an EPW, and which can be easily transferred to the real world 3) A lightweight model that can be deployed on the onboard electronics of an EPW. Contrary to the above, most methods in the literature use the discrete action space, they utilize expensive sensors (*e.g.*, LiDARs) or cameras that do not transfer easily to the real world and often propose complex models that have significant computational requirements. More details further justifying those claims will be provided in Chapter 3.

2.2.5 Autonomous Wheelchairs

The current state-of-the-art in the field of EPW navigation includes systems capable of performing obstacle avoidance, simultaneous localization and mapping (SLAM), path planning, and motion control. The majority of the autonomous navigation solutions[69, 70, 71] rely on constructing a map of an environment and utilizing a global planner, along with a localization method and a local planner to reach a desired goal in the map, similar to what was described in the previous sections.

Aiming for indoor navigation, the method introduced in[72] integrated points of interest into a floor plan, in which an adjacency matrix was added to enable the use of the $A^\star$ algorithm for path planning. Gao *et al.* [73] focused on synthesizing 3-D landmark maps to improve localization performance of a smart wheelchair, with the mapping process, however, requiring a high-fidelity sensor suite, including two LiDAR modules and a digital video camera. Another method using landmarks is introduced in [74], where the authors used AprilTags, a type of fiducial marker, to localize the wheelchair and perform a navigation routine between rooms.

Morales *et al.* [75] took into account the human comfort in the path and the trajectories produced for autonomous navigation, by augmenting the standard geometric map built via SLAM to a comfort map and using a modification of the $A^\star$ algorithm for path planning. In [70], the ROS navigation stack was integrated with the EPW to perform mapping, localization, and planning. Similarly, the authors in [71] used SLAM, along with the global planner provided by the ROS navigation stack, combined with an adaptive controller to account for the changing system dynamics. Finally, Grewal *et al.* [69] utilized computer vision to identify possible destinations for the user to choose, before applying Dijkstra's algorithm along with the Dynamic Window Approach to navigate towards the chosen target. However, the destination scanning process takes a long time to complete, and it requires high computational power such as a laptop or PC integrated into the wheelchair.

All these approaches are based on traditional path and motion planning methods, thus sharing similar disadvantages, such as the dependency on existing maps and/or landmarks, extensive tuning, use of high-fidelity sensors, and significant computational demands. This thesis proposes a machine learning approach, that does not require a map, only utilizes cost-effective sensors, and is also computationally efficient.

2.3 Shared Control

Despite the various works done in autonomous navigation and its impressive advances (*e.g.*, automotive/aviation industries), handing over full control to an autonomous system in a realistic setup is not yet possible given the current technological capabilities. Full automation can be achieved in environments that can be predicted with high accuracy and where the consequences of failure are acceptable. However, in more complex environments there is a high likelihood of encountering conditions for which the automation system cannot guarantee a robust or safe response, and where at least some level of human control is required to complete the desired task.

Shared control is an instance of human-robot collaboration, where a human user and a robotic system jointly arrive at the decisions or control actions needed to achieve some goal. Research on this topic has proposed a great variety of methods, as well as a number of different definitions of what constitutes shared control varying between different works and application domains. This work adopts the definition given in [5]:

In shared control, human(s) and robot(s) are interacting congruently in a perception-action cycle to perform a dynamic task, that either the human or the robot could execute individually under ideal circumstances.

The definition above is open to many forms of shared control, while it excludes the cases where the system or the human act individually (full autonomy, manual control, or traded control). In this context, shared control can be classified into two general categories: *input-mixing shared control* and *haptic shared control*. In the first approach, the control action of the system is a blend between the human's input and a controller's (autonomous system's) output, while in haptic shared control the resulting control action occurs at the force level [76, 77, 78].

Even though the approach followed in this thesis is not that of a typical shared control system, it is still concerned with the input-mixing shared control type, where the relevant research differs mostly in how the user's input is interpreted and how the blending takes place. This section explores the most widely adopted shared control methods in the literature, as well as methods utilizing the machine learning framework which are more relevant to our approach. Special focus is placed on the methods that are appropriate to use with EPWs.

2.3.1 User intention prediction

In the shared autonomy setup, in order for the assistive system to be an effective collaborator, it requires, in most cases, knowing the user's goal. This is usually something that is not known a priori, and thus it has to either be continuously indicated by the user, which is hard and impractical, or it needs to be inferred by the system. A system can achieve this by utilizing the user's stream of actions and/or information from the environment gathered by sensors [6]. Methods that require knowledge of the user's goal follow the so-called *predict-then-act* approach [79, 80, 81, 82], which splits the task of shared autonomy into two parts: (1) predict the user's goal with high probability (2) provide assistance for the predicted goal, potentially proportional to the level of confidence of the prediction, or even act autonomously to achieve that goal.

There are various approaches to estimating the user's intended trajectory, even though that field of research is still maturing. A straightforward approach is fixing a goal pose at a short distance in the direction indicated by the user's command [83, 84]. However, the assumption of a fixed goal is naive, since it does not take into account factors like the user's capability of indicating the proper direction, or the type of control interface that is used. More accurate estimates of the intention can be derived by dynamically determining the goal. Since the user's intention can be expressed as desired velocities (linear and angular), the magnitude of the intended direction can also be useful to determine the desired goal (e.g. deflecting the joystick more may indicate a further distance [79]). More sophisticated methods like Hidden Markov Model (HMM)-based methods predict sub-tasks during execution, treating the intent as a latent state [85, 86].

Other successful work for predicting user goals has been done by utilizing maximum-entropy inverse optimal control (MaxEnt IOC) [87], also known as IRL, where the user is modeled as a stochastic policy approximately optimizing some cost function. In this framework, the probability of a trajectory decreases exponentially with cost, and a distribution over goals from user inputs is derived, depending on how efficiently the inputs achieve each goal. However, performing IOC in continuous, high-dimensional domains is challenging, because such algorithms are usually much more computationally demanding than the corresponding "forward" control methods [88]. Laplace's method on the optimal trajectory between any two points has been used to tackle the computational infeasibility of the aforementioned approach, and thus, approximate the distribution over goals during shared control teleoperation [80]. Finally, Generative Adversarial Nets (GANs) [89] have been used to directly learn a policy that mimics the user [90], and also deep neural networks utilizing importance sampling have simultaneously learned a cost function and a policy consistent with user demonstrations [91].

Despite the various works that have been done on inferring user goals, predict-then-act methods have only demonstrated their effectiveness in simple scenarios with few goals [79, 86, 92, 80]. Furthermore, it is often impossible to accurately predict the user's goal until the end of execution (*e.g.*, cluttered scenes), in which case these methods provide little to no assistance. This is contradictory to our requirement of a system that should be able to provide assistance at all times, even in the absence of a goal or if the confidence for a single low is low. For that purpose, and as will be further explained later in Section 4, this thesis adopts a simpler yet effective approach to represent the intent of the user. *That is, modeling the intent as a probability distribution, as for example, a normal distribution over the range of goal poses or velocities centered on the indicated direction [93].*

2.3.2 Reactive planning

In order to modify the user's commands in a meaningful way, shared control needs a group of algorithms capable of planning motion. As discussed in Sec. 2.2.2, the type of a planner can be classified as either global or local. In the case of shared control (for powered wheelchairs and mobile robots in general), a local planner, and more specifically a reactive planner, is better suited to the problem, as by leveraging information from local sensors it is capable of adapting to unknown environments and dynamic obstacles. Reactive planners like APF, VFH, DWA or fuzzy logic controllers, that were described in Section 2.2.3, are particularly suited for the purpose of producing safe trajectories with a short term scope.

Having described the process of obtaining a model of the user's intention, as well suggestions from a local planner, we will now explore the types of techniques used to combine the two inputs to derive the resulting command of a shared control system.

Algorithm 1 The Dynamic Window Approach

```
1: Begin DWA(robotPose, robotGoal, robotConfig, currentV)
2: $desiredV \leftarrow calculateV(robotPose, robotGoal)$
3: $laserscan \leftarrow getLaserData()$
4: $V_d \leftarrow dynamicWindow(currentV, robotConfig)$
5: $opt_{cost} = -1$
6: for $(v_x, \omega_z)$ in $V_d$ do
7:     $dist \leftarrow calcObstacleDistance(v_x, \omega, laserscan, robotConfig)$
8:     $breakDist \leftarrow calcBreakingDistance(v_x)$
9:     if $dist > breakDist$ then ▷ robot is able to stop before reaching obstacle
10:        $heading \leftarrow hDiff(robotPose, goalPose, v_x, \omega_z)$
11:        $clearance \leftarrow (dist - breakDist)/(dmax - breakDist)$
12:        $cost \leftarrow costFunction(heading, clearance, abs(desiredV[0] - v_x))$
13:        if $cost > opt_{cost}$ then
14:            $opt_{v_x} \leftarrow v_x$
15:            $opt_{\omega_z} \leftarrow \omega_z$
16:            $opt_{cost} \leftarrow cost$
17:        end if
18:    end if
19:    set robot trajectory to $opt_{v_x}$, $opt_{\omega_z}$
20: end for
21: END
```

2.3.3 Blending strategies

Blending is one of the most widely used shared control methods due to its computational efficiency, simplicity, and empirical effectiveness. It refers to combining the user's command or their predicted intention with the trajectory of a motion planner. Such methods attempt to bridge the gap between highly assistive methods, which restrict user autonomy and control, and methods that offer minimal assistance, therefore burdening the user.

The most common approach to blending is called *linear blending* [94, 79, 81, 95, 82], and is simply a weighted sum of the user's and the path planner's trajectories. Denote the human control input at a given timestep t as $u^h(t)$ and the input of the controller as $u^c(t)$. Then, mathematically the linear blending law is defined as:

$$u^{LB}(t) = K_h u^h(t) + K_c u^c(t), \qquad K_h + K_c = \mathbf{1}, \tag{2.1}$$

where $u^{LB}(t)$ is the resulting control input, K_h and K_c are arbitration matrices, and $\mathbf{1}$ is the identity matrix. Figure 2.2 provides a visual representation of a typical linear blending strategy, where both the user and the autonomous controller offer separate suggestions, which are combined using an arbitration function with a control weight a. The resulting combined action is then executed by the robot, enabling collaborative control and striking a balance between user autonomy and system assistance. Linear blending techniques differ in how this weight is computed, usually changing dynamically according to several criteria, such as safety, directivity, agreement, or some combination of the above [96]. Of course, increasing the weight of the controller equals greater assistance at the expense of user control. This usually happens when the mobile robot is closer to obstacles in order to enhance safety. However, several shared control paradigms, especially in the field of assistive robotics, place special focus on the user being in control. In order to ensure that the robot acts in a way that most agrees with the user's commands, an agreement objective can be used so that more control is given to the user when the robot's trajectory is very different from the user's[97]. Despite its ease of implementation and its popularity, the formulation of linear blending is not suitable for complex and dynamic environments, since the simple addition of trajectories cannot guarantee collision-free movement or stable control of the robot [9].

Probabilistic shared control (PSC) is a work that attempts to improve on linear blending, by guaranteeing safety in its theoretical formulation [83]. This framework establishes a joint probability distribution between the user's intended trajectory and the path planner's calculated trajectory, which includes an agreeability function measuring the similarity between those two random variables. The selected control command for each time step is the suggested velocity of the path planner that maximizes the joint probability distribution. However, the resultant control law is entirely dependent on the planner's implementation and its safety guarantees, and not on the user's input [98].

Another approach to blending, which is tightly linked to the work done in this thesis, is with policy-based methods, which assume that the user selects an action

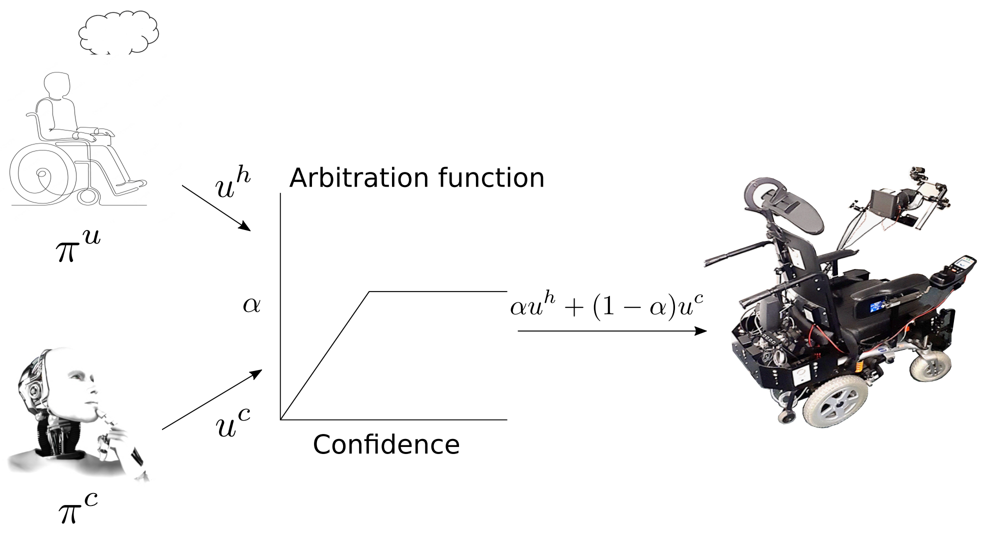


Figure 2.2: Blending Strategy for Shared Control Teleoperation. Both the user and the autonomous controller follow their own policy and provide separate suggestions u_h and u_c for achieving their goals. An arbitration function (typically a linear one) is used to combine the two suggestions, with the control-trade-off between the two determined by a control weight a. The combined action is passed to the robot for execution.

in order to minimize the user costs according to some function, and the system policy attempts to minimize its own expected sum of costs (provided with a distribution over goals) given the user input. *Contrary to previous blending techniques, this approach does not treat user and robot actions separately but selects the final action directly as the optimization output [7, 6].* More details on such methods, as well as on methods utilizing the reinforcement learning framework, will be given in the following section.

2.3.4 Policy-based Shared Control

This section focuses on methods that, contrary to methods using blending, directly output a control action for the robot, by optimizing some cost function that also takes the user input into account, as also depicted in Fig. 2.3. It also focuses on methods that model the problem of shared control as a sequential decision-making problem and utilize the reinforcement learning framework to solve it. The works

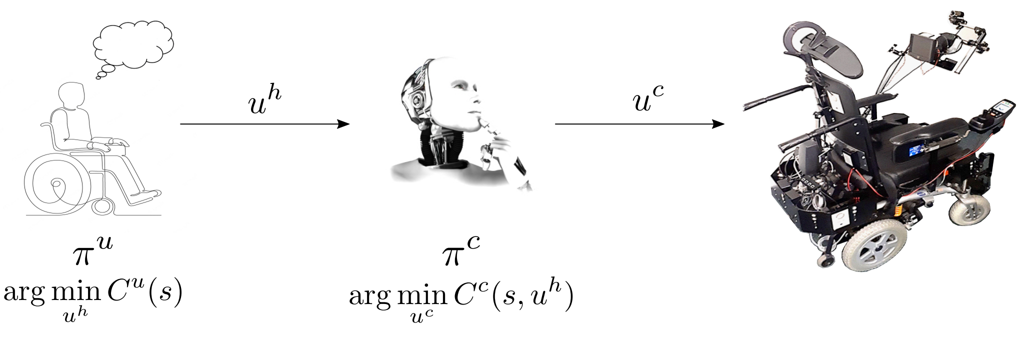


Figure 2.3: Policy-Based Strategy for Shared Control Teleoperation. The user is modeled as a policy π^u which selects actions that minimize the expected sum of user costs $C^u(s)$. The system is also modeled as a policy π^c that minimizes its own sum of costs $C^c(s, u^h)$, conditioned on the user input. The system action u^c, which is optimized given the user action, is directly passed to the robot for execution.

presented in this section are conceptually close to the work done in this thesis, more specifically the proposed RL framework for tackling the problem of shared control, which will be presented in Chapter 4.

In 2013, Hauser [6] presented a policy-based method that provides assistance for a distribution over goals. The proposed system uses a sampling-based planner (similar to RRT) to produce collision-free trajectories which follow the predicted user goal distribution, while it iteratively updates the distribution given user inputs. Javdani *et al.* [99] generalized this work, by formulating the problem of shared control as a partially observable Markov decision process (POMDP) and assuming that the user's goal is fixed and does not change based on the autonomous assistance system's actions, thus placing the burden of goal inference entirely on the system. Furthermore, this framework enables the use of any cost function for which a value function can be computed, instead of assuming a fixed form for the cost function, while the authors used hindsight optimization to approximate an optimal solution for the POMDP. Another line of work, which again models the shared control problem as a POMDP and is focused on user goals, is presented in Human-Robot Mutual Adaptation [8] that adopts a game-theoretic approach to shared control, where either the autonomous system guides an adaptable user or complies with a stubborn user. The POMDP is used to learn the adaptability of the user, which models the likelihood of the user adapting to the autonomous assistance. Even though this is a more general model compared to [99], it is computationally intractable for continuous states and actions.

Several works have used RL for determining the control weight(s) of a blending strategy for shared control. For example, the work presented in [100] uses the value-based RL algorithm *State-action-reward-state-action* (SARSA) to determine the control weight between an autonomous motion planner and user intent for commanding a walking-aid robot, optimizing for safety and control smoothness. In a similar fashion, Xi [101] also used SARSA to learn the control weight between the user and the DWA, selected as the motion planner for navigating an EPW, while both the state and the action space were discretized. Another recent development following the same concept is presented in [102], which extends the state-of-the-art *Twin Delayed Deep Deterministic Policy Gradient* (TD3) to develop an RL agent that outputs the shared control ratios for a blended control input, which combines the user's input along with the input of three autonomous controllers that perform different functions (*i.e.*, clockwise and counterclockwise wall-following, collision avoidance) to assist the navigation of a simple mobile robot. Muelling *et al.* [81] dynamically adjusts the arbitration factor in a linear blending paradigm, based on the confidence of the predicted intent, which is assumed to be driven by an agent that follows a policy to minimize a goal-chasing cost function.

Approaches that adopt the use of an arbitration function, however, share the same disadvantages as the approaches described in 2.3.3. Catastrophic failure can occur as a result of combining two independent decisions without evaluating the action that will be ultimately executed [9]. On the other hand, methods that infer the user's goal from their input and autonomously act to achieve it[6, 7, 81, 8] tend to assume 1) a known dynamics model of the world 2) a known set of possible goals 3) a known user policy given a goal. These assumptions, though, act as a limiting factor to an adaptable and general shared control system that should be able to handle real-world tasks. Deriving an accurate state transition model can be very challenging in practice, while assuming a fixed goal representation constrains the flexibility of the system in pursuing goals that have not been taken into account. Finally, a goal inference algorithm that has been trained to detect specific patterns or behaviours will not be able to adjust to erratic or suboptimal user input.

Reddy *et al.* [103] lifted the above assumptions, by proposing an end-to-end approach to the problem of shared control, where environmental observations and user input are directly mapped to control actions. This study introduces a shared control framework based on model-free RL, that removes the need for known dynamics, a particular goal representation, or even a user behaviour model. The system accepts the environment's state, the human's commands, and an inferred goal (optionally) as inputs and outputs the optimal action that most closely matches the human command, with the only form of supervision being the task reward. In order to achieve that, the authors introduce a modification to the typical action selection process of the popular *Deep Q-Network* (DQN) algorithm, where the selected action is a feasible action that is the closest to the user's suggestion. An action is considered feasible if it is not much worse than the optimal action (the one with the highest Q-value), something that is determined by a user-defined hyperparameter. An extension to this work was done by You *et al.* [104], who further modified it to dynamically adapt the arbitration parameter controlling the tolerance of the system to suboptimal human suggestions defined in [103].

The work on shared control presented in this thesis shares the same motivation with Reddy's work, in the context of aiming to train an end-to-end system via model-free RL, which makes the minimum possible assumptions about the world and the user. However, Reddy's proposed approach has an important limitation, which is that it only works with a discrete action space. Furthermore, even if the agent does not have direct access to goal-relevant information, it still receives rewards for completing user goals, which might overfit its behaviour to the environment it is trained in and consequently hurt its generalization capabilities.

Inspired by that work, but also its limitations, this thesis builds a flexible and *generic framework for performing continuous shared control, that does not necessarily assume the existence of specific user goals, but should still be able to provide meaningful assistance in a versatile manner, depending on its parameterization. Shared control is formulated as an end-to-end system, which takes as input the current user intention along with environmental observations, and directly outputs a control command for the mobile platform, instead of having to explicitly derive the user intention or requiring input from a local planner. The agreeability of the system with the user, which is of critical importance, is imposed on an algorithmic level, as part of the optimization objective that is used to train the shared control system. The methodology and effectiveness of this approach are presented in Chapter 4.*

2.3.5 Shared Control for Wheelchairs

NavChar [105] was one of the earliest works done towards shared autonomy in powered wheelchairs. This work, modified the VFH algorithm, replacing it with the Minimal VFH (MVFH) method, in order to take into account the user input from a joystick and perform obstacle avoidance, door passing or wall following. Other projects that proposed a multi-mode solution, include the Bremen Autonomous Wheelchair [106] with four distinct modes of operation that are manually chosen, or RobChair [107] that utilised fuzzy logic to switch between different behaviours (*e.g.*, collision detection, wall following). Some of the main issues with those early efforts mentioned above, are the pre-determined or heuristic ways of deciding the mode of operation without taking into account the intent of the user, as well as the limited control that the user is allowed. More recently, Vanhooydonck *et al.* [108] proposed a framework, where modules that implement different assistive behaviours (*e.g.*, safe stopping, obstacle avoidance etc.) can be selected and activated, depending on the most probable user intention which is estimated by an implicit user model.

Linear blending, which was described in Section 2.3.3, has also been widely used in shared control for powered wheelchairs. An example is presented in [94], where the authors dynamically adjusted the control weights of a linear blending between the user and a reactive controller, by solving a multi-objective optimization problem with manually defined indices corresponding to safety, comfort and obedience. Another closely related work is shown in [101], that uses RL to dynamically adjust the user's control authority based on some specified design requirements. Erdogan [109] presented four control-sharing paradigms, where a user signal and an autonomy signal are linearly blended to assist the user with arriving at their goal, which can be either an immediate one or a high-level goal. Similarly, [110] implements a linear blending solution, where the parameter that controls the sharing between user and autonomy is determined by whether a forward projection of the user's command results in a collision, and by a confidence metric associated with an observed navigation goal. Despite the different variations, the issues associated with linear blending (see Sec. 2.3.3) are still present in the methods implementing it. The authors in [83] implemented probabilistic blending, using the DWA to generate safe trajectories, and compared it to a basic type of linear blending (averaging of trajectories). Their results showed an improvement in safety, with a trade-off in distance travelled and completion times, still however, relying on the effectiveness of DWA to produce safe trajectories.

Apart from the popular DWA planner, other local planners (see Section 2.2.3) have also been adapted and put into use for the problem of shared control in powered wheelchairs. Carlson [79] introduced a collaborative system which produces safe mini-trajectories based on the elastic band method and guides the wheelchair towards the safe mini-trajectory once the system is confident about the goal of the user, while the user maintains control of the speed. Urdiales *et al.* [111] utilized the Potential Fields Approach to return a motion vector that can be combined with the joystick vector to provide a reactivate layer of assistance in a continuous and adaptive manner. A similar approach is adopted in [112], where a potential force field acts as a moderating input on the user's desired trajectory, while the geometry of the platform is also taken into account instead of simply being treated as a point mass. The work in [113] extended this method by splitting the geometry of the platform into layers based on the shape of the wheelchair and used a 3-D depth sensor allowing the algorithm to consider the height of the obstacles in relation to the wheelchair and adjust its correction accordingly.

In a more relevant framework to this thesis, some approaches have treated the problem of shared control for powered wheelchairs as a sequential decision-making problem. Ghorbel *et al.* [114] proposed a decision-theoretic model of control, where the collaborative control task is modelled as a POMPD with uncertainty over the destination. This method utilizes the framework of shared autonomy for robotic manipulators [7], which was described in the previous section, and adapted accordingly to address the challenges of navigation. However, that approach assumes a known set of goals, while it uses dynamic programming to solve the POMPD, thus, not being well suited to solving environments of high complexity or handling a richer state space. Similar limitations are encountered in [115], where policy iteration (a Dynamic Programming technique similar to policy-based RL methods explained in Sec. 3.2.2) is used to solve shared control formulated as a time-optimal stochastic shortest path problem, after optimal maps of the average time to hit obstacles ahead of the wheelchair have been pre-computed. This approach works in mapped environments, for specific user models and for a finite state space.

This thesis focuses on designing a novel framework that tackles the drawbacks described in the methods that are currently used in shared autonomy for EPWs. We want to allow integration of multiple modes of use (support different modes of operation/assistive behaviours), enable customization to the individual user, and also have an implementation that is fast enough to run smoothly on an onboard computer. More importantly, the approach the thesis adopts does not necessarily assume known maps or goals, but rather assists the users only based on local information. Shared control for PW navigation is realized as a sequential decision-making problem and builds on ideas from the broader field of shared control and the methods presented in Section 2.3.4, to improve upon the existing shared control methods for EPWs.

2.4 Summary

This chapter provided the background of methodologies used for autonomous and semi-autonomous mobile robot navigation, along with their relevant applications for powered wheelchairs. The most established approach for autonomous navigation involves mapping the environment and integrating localization methods with global and local planners. However, recent advancements in deep neural networks and machine learning have garnered significant attention, aiming to address the limitations of traditional methods. Notably, the use of the RL framework has been emphasized, which will be employed in this thesis to design an autonomous navigation system for powered wheelchairs, a novel application in the existing literature.

The chapter also introduced the concept of *shared control*, a form of semi-autonomy, and deconstructed it into its main elements, particularly in the context of input-mixing shared control. Similar to the autonomous navigation section, an overview of the methods that utilize machine learning to perform shared control has been thoroughly presented. Identified gaps in existing methods include: 1) Blending methods without safety guarantees, 2) Policy-based methods with several assumptions (*e.g.*, known world dynamics, known set of goals, known user policy), and 3) Methods that address those assumptions but use discrete control. *Importantly, the review on shared control methods concluded with a discussion of the state-of-the-art in powered wheelchairs, where reinforcement learning has not yet been effectively proposed or applied as an end-to-end assistive system.*

The thesis aims to employ reinforcement learning in developing an autonomous and semi-autonomous solution for powered wheelchair navigation. In both approaches, the system directly maps environmental observations to continuous control commands for the robot. To train the shared control system, the autonomous system acts as a mock user, involving user participation in the process. This concept is inspired by a modification of the DQN algorithm presented in Reddy et al.'s work [103], which focuses on discrete shared control and system-user agreeability. However, in this thesis, a distinct and more versatile adaptation to another RL algorithm is proposed to enable continuous shared control. By addressing various assumptions from previous works, this new algorithm provides a generic shared control framework for effective human-robot cooperation, specifically tailored for powered wheelchair application.

CHAPTER 3

Introduction of an Autonomously Navigated Wheelchair

This chapter introduces a model-free learning approach for autonomous navigation with nonholonomic robots, specifically powered wheelchairs. The proposed method employs a reinforcement learning agent capable of autonomously pursuing goals in a continuous action space, relying solely on environmental information from the robot's sensors, without requiring a map. The effectiveness and efficiency of this approach are extensively evaluated through simulation experiments in both familiar and unfamiliar environments. Comparisons are made against human performance and state-of-the-art motion planners. The objectives of this chapter are twofold: 1) Introduce a novel method for autonomous powered wheelchair navigation using RL, which improves certain aspects of existing methods in mobile robot navigation. 2) Develop a user model, a crucial component in the methodology presented in Chapter 4, where a novel shared control methodology will be introduced.

3.1 Introduction

It is not uncommon, that EPW users, due to poor motor control, inconvenient control interfaces, fatigue, or other reasons, face difficulties with coping with daily maneuvering tasks. In those cases, an autonomous navigation system would be greatly beneficial in alleviating part of their daily burden and enhance their independence.

As already discussed in Section 2.2.5, the state-of-the-art autonomous navigation methods for powered wheelchairs rely on mapping the environment of interest, and proceed to utilize widely adopted global and local planners for following given goals on the map. The work presented in this chapter takes a different direction, which adopts a learning approach to the problem, by taking advantage of the recent advances in the RL field. The advantage of such an approach is that it can tackle several drawbacks of the traditional methods: a) the need of a map, b) high computational costs, c) extensive hyperparameter tuning and d) expensive sensors.

Similar approaches have been encountered in the literature for mobile robot navigation (recall Section 2.2.4), however, this work has some important distinctions, aimed at building a more realistic system, appropriate to use with an EPW. Contrary to most of the methods in the relevant literature which control the robots through discrete commands, this work addresses the problem of navigation in the continuous action space. This is a necessary requirement for avoiding rough navigation behaviours, that might cause discomfort to the user, while it enables finer control for maneuvering in cluttered/challenging environments. Another requirement is that the system comes with a sensor setup that can be easily transferred from simulation to the real world, and that is relatively low-cost. Moreover, the sensor arrangement should be able to fully cover a bulky robot such as an EPW. Therefore, the chosen range sensors for detecting obstacles are ultrasonic sensors, instead of RGB cameras (poor real-world transferability) or LiDARs (expensive). Finally, a small and easily trainable neural network architecture is utilized, as opposed to complex architectures found in other works, that might not be applicable for real-time use on the onboard electronics of the robot.

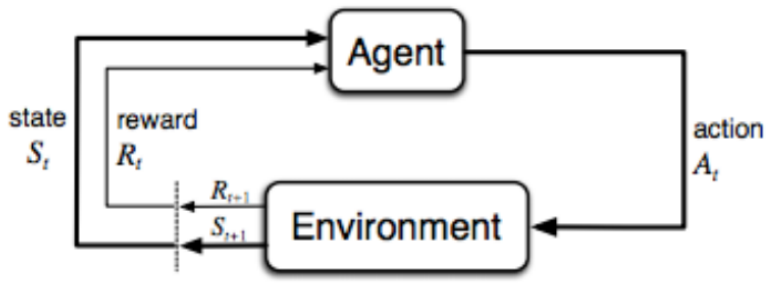
The motivation behind the work of this chapter is twofold: to improve upon the existing state-of-the-art in EPW navigation by proposing the use of the RL framework, and to create a model of an EPW user, which is a necessary requirement for training an assistive system as will be shown in Chapter 4. The rest of this chapter will focus on the first aspect, by presenting the method, its implementation and validation, while the use of the trained system as a model user will be discussed in Section 4.2.1. Before delving into the methodology, a brief overview of the deep reinforcement learning framework, as well as the simulation setup that forms the basis for the development and experimentation of the work described in the following chapters are first presented.

3.2 Background to Reinforcement Learning

The computational field of Reinforcement Learning is a framework that addresses the problem of sequential decision making, the same way behavioral neuroscience tries to understand it in terms of how humans and animals select actions in the face of reward and punishment. RL in its modern form stems from two separate lines of research; that of Sutton and Barto [117, 116] who developed the core algorithms and concepts of RL, and that of Bertsekas and Tsitsiklis [118] who developed stochastic approximations to Dynamic Programming (DP) methods (neuro-dynamic programming). The fusion of these works, made it possible to formalize the behaviorally-inspired heuristic RL algorithms in terms of optimality, while providing tools for analyzing their convergence properties.

In RL the main idea lies in learning through interaction. In the RL setup there is an agent (*e.g.*, a wheeled robot) that interacts with the environment it is in and learns to adapt its behaviour based on a reward signal it receives from that environment. By successive interactions new information is produced, which is then used by the agent to update its knowledge in a fashion that will maximize the expected return (cumulative, discounted reward) it will receive over its lifetime (Fig. 3.1).

Even though RL is not a new area of machine learning, it has lately gained increased momentum due to various successes stemming from its combination with deep neural networks. This gave birth to Deep Reinforcement Learning (DRL), which made possible the scaling of prior work in RL to high-dimensional problems, by taking advantage of the function approximation properties of neural networks



Source: Sutton's textbook [116]

Figure 3.1: The Core Principle of Reinforcement Learning.

and their ability of learning low-dimensional feature representations, while providing a means of tackling the curse of dimensionality, unlike traditional tabular and non-parametric methods.

3.2.1 Markov Decision Process (MDP)

Any problem in RL can be formulated mathematically as a *Markov Decision Process* (MDP). An MDP is a tuple (S, A, T, γ, R) where:

- S is a set of states
- A is a set of actions
- $T(s_{t+1}|s_t, a_t)$ are the state transition dynamics that map a state-action pair at time t onto a distribution of states at time t+1
- $\gamma \in [0, 1]$ is called the discount factor, where lower values place more emphasis on immediate rewards and greater values look more into future rewards
- $R(s_t, a_t, s_{t+1})$ is the reward function which returns an immediate scalar reward after a transition to a new state

The agent chooses its actions according to a policy π which in general is a mapping from states to a probability distribution over actions: $\pi : S \rightarrow p(A = a|S)$. The goal of RL is to find an optimal policy $\pi^\star$ that achieves the maximum expected return from all states:

$$\pi^\star = \arg\max_{\pi} \mathbb{E}[R|\pi] \tag{3.1}$$

A key property of MDPs is the *Markov property*, which states that the next state is only dependent on the current state, or in other words, that the future is conditionally independent of the past given the present state. This assumption, which is adopted by the majority of RL algorithms, requires the environment (or the states) to be fully observable, but of course that rarely applies to the real world. A generalisation of MDPs is a *Partially Observable MDP* (POMDP), where the agent does not have access to the full information regarding the current state, but receives an observation that is a subset of the state's information, and which might also contain information from previous states. The agent uses the observations it gathers to form a belief of what state the system is currently in, the so-called belief state, and is expressed as a probability distribution over the states. The solution of the POMDP is a policy indicating the optimal action for each belief state.

Finally, an MDP can either be *episodic*, where the state is reset after each episode of length T (the trajectory or rollout of the policy) or it can be *non-episodic* (*continuous*), where $T = \infty$, thus the trajectory is of potentially infinite length.

3.2.2 Kinds of RL Algorithms

Since RL is currently a very active research field, and with a rapidly changing landscape, there is no single algorithm that best works on all problems, but rather a large pool of them with varying characteristics. The aim (*i.e.*, optimizing an agent's policy to maximize expected return) remains the same, but the approach can change, each algorithm having its own trade-offs and tackling different challenges out of the numerous that exist in the field. The rest of this section will provide an overview of the most prominent categories of RL algorithms.

Model-based vs Model-free

One major distinction between algorithms in RL can be made from whether the agent has access to or needs to learn a model of the environment. Such a model refers to a function that predicts state transitions as well as rewards.

Given a model, an agent can predict what is going to happen next without having to interact with the environment, and can therefore plan accordingly (*e.g.*, use Dynamic Programming to find the optimal solution). By simulating the environment and planning ahead, calculated results can then be distilled into a learned policy, which can grant a substantial improvement to sample efficiency and significantly speed up learning compared to methods that do not have a model of the environment available.

However, a ground-truth model of the environment is rarely available to the agent, in which case an agent that follows a model-based approach will have to learn the model purely from experience. This can be challenging in many aspects, for example, bias in the model which can be exploited by the agent, causing it to overfit and perform poorly in the real environment, or the existence of inaccuracies in the model which can cause instability, the more the further into the future the agent looks.

On the other hand, model-free methods learn policies directly from interactions with the environment, and even though they tend to be less sample efficient, they are easier to implement and tune. Currently, those methods are considered more popular and have been more extensively developed and tested. This thesis only focuses on model-free RL.

Value-based vs Policy-based

A major branching of model-free RL algorithms lies in what needs to be learned. Before delving deeper into the two major approaches that are used, it is first useful to explain what a *value function* is.

The value function measures the goodness of a state, giving an estimate of how much future reward can be gained from that state onwards. The future reward, also known as **return**, briefly mentioned in Sec. 3.2.1, is the total sum of discounted rewards going forward. More formally, the return G_t starting from time t is given by:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\inf} \gamma^k R_{t+k+1}, \tag{3.2}$$

where γ is the discount factor, which penalises the rewards in the future. If $\gamma = 0$, only the immediate reward is taken into account, whereas if $\gamma = 1$ all future rewards are considered with an equal weight.

The **state-value** of a state s is the expected return when being in this state at time t:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s], \tag{3.3}$$

given a policy π is followed for picking future actions. Similarly, the **action-value**, or the **Q value**, of a state-action pair is defined as:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a], \tag{3.4}$$

measuring the value if at state s the agents picks action a and follows policy π onwards. The probability distribution from policy π over the actions can also be used to formulate the value function as:

$$V_\pi(s) = \sum_{a \in A} Q_\pi(s, a) \pi(a|s) \tag{3.5}$$

The difference between action-value and state-value defines the action **advantage** function,

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s), \tag{3.6}$$

which intuitively provides an estimate of how good or bad a specific action is for a given state compared to the others, on average. Naturally, the optimal value function will produce the maximum return:

$$V^\star(s) = \max_\pi V_\pi(s) \qquad\qquad Q^\star(s,a) = \max_\pi Q_\pi(s,a) \tag{3.7}$$

And it is the optimal policy that achieves the optimal value functions:

$$\pi^\star = \arg\max_\pi V_\pi(s) \qquad\qquad \pi^\star = \arg\max_\pi Q_\pi(s,a) \tag{3.8}$$

Consequently:

$$V_{\pi^\star} = V^\star(s) \qquad\qquad Q_{\pi^\star}(s,a) = Q^\star(s,a) \tag{3.9}$$

It is also important to mention the **Bellman equation** in MDPs, which is ubiquitous in RL and provides the optimization target for value functions. A value function can be decomposed into a sum of the immediate reward that is received when transitioning to the next state, and the discounted future rewards that can be received from that future state onwards, following a policy π. Formally,

$$V_\pi(s) = R(s,\pi(s)) + \gamma \sum\nolimits_{s'} P(s'|s,\pi(s))V_\pi(s'), \tag{3.10}$$

where P is the transition probability distribution over next states the agent can transition to, when being in state s and picking an action according to policy π. The equation for the optimal policy is known as the *Bellman optimality equation* and is given by:

$$V_{\pi^\star}(s) = \max_a \{R(s,a) + \gamma \sum\nolimits_{s'} P(s'|s,a)V_{\pi_\star}(s')\} \tag{3.11}$$

Since some basic definitions have been established, the following subsection discusses the major approaches of representing and training agents in model-free RL.

Q-Learning vs Policy-Optimization

Methods of the *Q-learning* family focus on learning an approximator $Q_\theta(s,a)$ of the optimal action-value function $Q_\star(s,a)$. The objective function that is used to train the approximator (a neural network in the case of DRL) is based on the Bellman equation (Eq. 3.11) described in the above section. Each optimization step can use data collected at any point during training, regardless of the policy followed by the agent when that data was collected. The optimal policy is obtained by Eq. 3.8, which links $Q^\star$ with $\pi^\star$, and therefore the actions taken by the Q-learning agent are given by:

$$a(s) = \arg\max_a Q_\theta(s,a) \tag{3.12}$$

Policy optimization methods in DRL also use an approximator, but this time to explicitly represent a policy as $\pi_\theta(a|s)$. They are typically combined with another approximator that learns the value or the advantage function, used for indicating how to update the policy. The parameters θ of the policy network are optimized either directly by gradient ascent on an objective function $J(\pi_\theta)$ (which usually involves the advantage estimate), or indirectly by maximizing local approximations of $J(\pi_\theta)$. The updates in the optimization process use data that are collected each time acting with the most recent version of the policy.

Directly optimizing the policy to be learned tends to make policy optimization methods stable and reliable, but this comes with the price of requiring a large number or samples for the training process. Contrary, Q-learning methods are more sample efficient, since they can reuse data and experience they gather more effectively. On the other hand, when the Q function (*i.e.*, reward function) is too complex to be learned it is likely that Q-learning will fail, whereas policy gradients can still converge to a good policy since they operate in the policy space. Furthermore, Q-learning is not capable of learning stochastic policies neither use continuous actions, things that policy optimization methods naturally do since they are designed to model probability distributions. Finally, a big drawback of policy gradients is the fact that they can suffer from high variance in estimating the gradient of the expectation of the rewards, which can heavily affect the stability of the learning algorithm.

Actor-Critic Methods

Since RL methods are rarely mutually exclusive, it is possible to mix different approaches that complement each other. *Actor-critic* combines the policy gradient with value function estimation, in order to improve the sample efficiency of policy gradient methods and reduce the variance of the advantage function, thus making the gradient less volatile. The actor is the network that models the policy, whereas the critic is the network that models the value function. By introducing the critic, the number of samples that needs to be collected for each policy update is reduced. Also, the *Temporal Difference* (TD) technique [119] is used, thus it is not needed to collect all the samples until the end of an episode. Algorithm 2 summarises the main principle of the Actor-Critic algorithm.

Algorithm 2 The Actor Critic Algorithm

```
1: Initialize critic network $V^\pi_\phi$ and actor network $\pi_\theta$ with weights $\phi$ and $\theta$
2: **repeat**
3:     take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}, r)$
4:     update $V^\pi_\phi$ using target $r + \gamma V^\pi_\phi(\mathbf{s'})$
5:     evaluate $A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma V^\pi_\phi(\mathbf{s'}) - V^\pi_\phi(\mathbf{s})$
6:     $\nabla_\theta J_\theta \approx \bigtriangledown_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) A^\pi(\mathbf{s}, \mathbf{a})$
7:     $\theta \longleftarrow \theta + \alpha \nabla_\theta J_\theta$
8: **until** convergence
```

On-policy vs Off-policy

This categorization of RL algorithms has already been indirectly mentioned. *On-policy* methods refer to agents that always follow their own policy. Each time a policy is updated, new samples from that policy need to be generated. The agent acts, improves, then acts again, and so on.

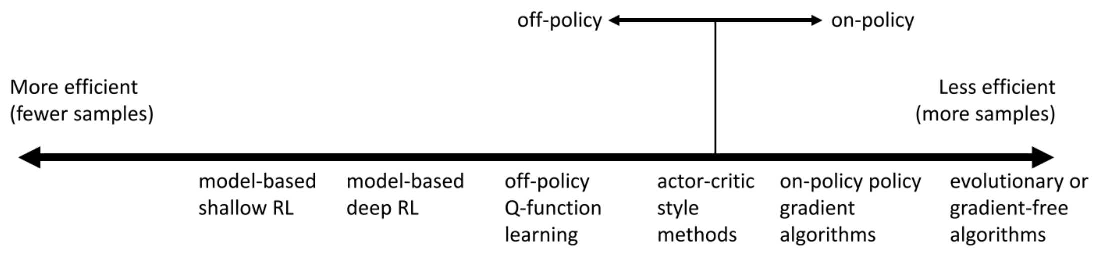

off-policy
on-policy
More efficient (fewer samples)
Less efficient (more samples)
model-based shallow RL
model-based deep RL
off-policy Q-function learning
actor-critic style methods
on-policy policy gradient algorithms
evolutionary or gradient-free algorithms

Figure 3.2: On-Policy vs Off-Policy Methods Sample Efficiency.

Off-policy algorithms, on the other hand, can improve the policy without generating new samples from that policy. This means that the policy being updated does not have to be the same policy the agent acts on. Off-policy methods provide better exploration and utilize samples more efficiently.

While policy optimization algorithms are mostly on-policy, Q-learning algorithms are almost always off-policy. Actor-critic methods, however, can be either on-policy or off-policy depending on the specific implementation. For a rough mapping of on-policy and off-policy methods in terms of their sample efficiency, refer to Fig. 3.2.

Experience Replay

Learning robust policies in RL requires a significant number of interactions with the environment, through which the agent improves its knowledge regarding its behaviour and the value of states and actions, for which the ground truths are not available in advance. On top of the fundamental challenge of generating sufficient data to train a RL policy, the inputs (observations) and targets (rewards) are continuously changing over the course of training, making it difficult to maintain a stable training process. A simple, yet powerful idea of improving sampling efficiency (*i.e.*, minimize the number of experiences that need to be generated to learn robust and high-performing policies) and training stability is the *experience replay*, used with off-policy DRL.

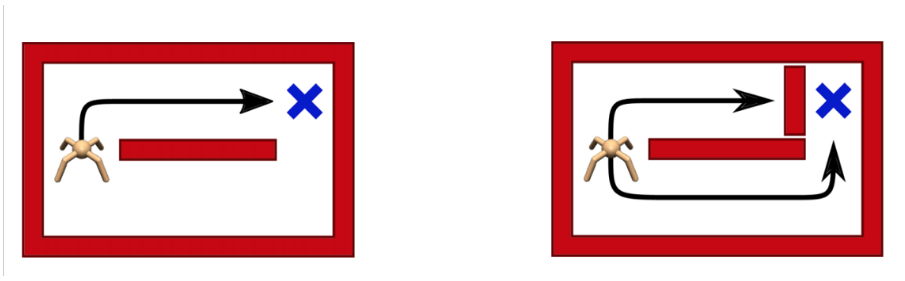
The main idea of experience replay lies in storing previously experienced environment interactions and use that experience to update the policy, instead of having to regenerate new experience for every update. Experience is represented as a *transition*, which consists of five signals that compose a training sample. These are: 1) State (s) 2) Action (a) 3) Reward (r) 4) Next State (s') 5) Done signal (d), a binary signal indicating whether the current transition is the final one in a given rollout/episode. Symbolically, the transition quintuple is given as: $T = (s, a, r, s', d)$. Transitions are stored into a buffer, known as the *Experience Replay Buffer*, typically in a first-in-first-out (FIFO) fashion.

Over time, the buffer fills with experience gathered by the agent following different policies, and forms a diverse dataset that allows the agent to revisit and learn from their memories. For updating the policy, a random subset of the dataset can be drawn from the buffer and used to train the involved neural networks in a mini-batch, supervised learning style. This has be proven to be highly effective in stabilizing the training, by decorrelating updates and avoiding the rapid forgetting of rare experiences[120, 121]. Several works in the literature have proposed more sophisticated and efficient strategies of sampling from the replay buffer (*e.g.*, Prioritized Experience Replay (PER)[121] or Hindsight Experience Replay (HER)[122]), however, this thesis is concerned with utilizing a standard replay buffer implementation.

3.2.3 Maximum Entropy Reinforcement Learning

Entropy, in the context of information theory, measures the amount of information or uncertainty associated with the possible outcomes of a random variable. In the context of RL, entropy relates to the unpredictability of the actions the agent chooses according to a given policy. Naturally, the greater the entropy, the more random the actions of the agent. For RL algorithms that define the actions as a probability distribution (*e.g.*, policy-gradient and actor-critic methods), the entropy of a policy in a state s_t is defined by the expected negative log-likelihood of the policy:

$$\mathcal{H}(\pi_\theta(s_t)) = \mathbb{E}_{a\sim\pi_\theta(s_t)}[-\log \pi_\theta(s_t, a)] \tag{3.13}$$



Source: Berkeley CS 294-112: Deep Reinforcement Learning

Figure 3.3: Example of a Diversified Policy.

A policy that starts randomly and, over the course of training, learns to take specific sequences of actions to accomplish its task, is expected to converge towards a more predictable and deterministic behavior, becoming less random. However, several lines of research have suggested making the action-selecting policy more unpredictable by adding an ”entropy bonus” term to the loss function. Counterintuitive as this may sound, it is an attempt to tackle one of the greatest challenges in RL: the *exploration vs. exploitation* trade-off. Exploitation refers to the agent exploiting its current knowledge and greedily picking actions with the maximum expected return. However, this can lead the agent to quickly converge to a locally optimal policy, but not necessarily a globally optimal one. Exploiting its current beliefs prevents the agent from exploring different strategies that might lead to greater long-term rewards. In this case, an entropy bonus forces the agent to act more randomly, thereby boosting exploration and avoiding local optima.

Adding an entropy-associated term to the loss function (also known as *entropy regularization*) as a one-step bonus is a concept found in many current state-of-the-art, on-policy DRL methods, such as the Asynchronous Advantage Actor Critic (A3C) [65], or the Proximal Policy Optimization (PPO) algorithm [123]. However, these approaches apply the entropy bonus only to the current state, without considering future states. In this regard, several lines of work in the literature [124, 125, 126] suggest that an even better approach is to optimize for the long-term sum of entropy, similar to how agents learn to maximize the long-term, cumulative reward. In this case, the optimal policy is the policy that maximizes both future discounted rewards and long-term entropy:

$$\pi^\star = \arg\max_{\pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\inf} \gamma^t \left(r_t + \alpha \mathcal{H}_t^\pi \right) \right] \tag{3.14}$$

The definitions of the value and the Q function also change to include the entropy bonuses from each timestep:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\inf} \gamma^t \left(r_t + \alpha \mathcal{H}_t^\pi \right) \middle| s_0 = s \right] \tag{3.15}$$

This formulation results in agents that strive to accumulate as much reward as possible while retaining adaptability in the long term. The significant advantage of this approach, compared to the one-step entropy bonus, is that it provides more robust performance under potential changes both in the agent's knowledge about the environment and in the environment itself. Fig. 3.3 showcases a simple example of a diversified policy that can follow a different path when the environment changes, enabling it to reach its goal effectively. This RL framework, formally known as *Maximum Entropy Reinforcement Learning*, constitutes the primary focus of this thesis.

3.3 Simulation Setup

3.3.1 Robot Operating System

The *Robot Operating System* (ROS) [127] is an open-source framework, which provides a set of tools and services that help in building robot applications. Its major functionalities include sensor interfacing, package management, message passing between processes, communication between different devices, and sensor output visualisation, among others.

The programs that use the ROS framework are termed as *ROS Nodes*, and are most commonly written in either C++ or Python. It is possible to have multiple nodes running in parallel and independently, but communication between them is

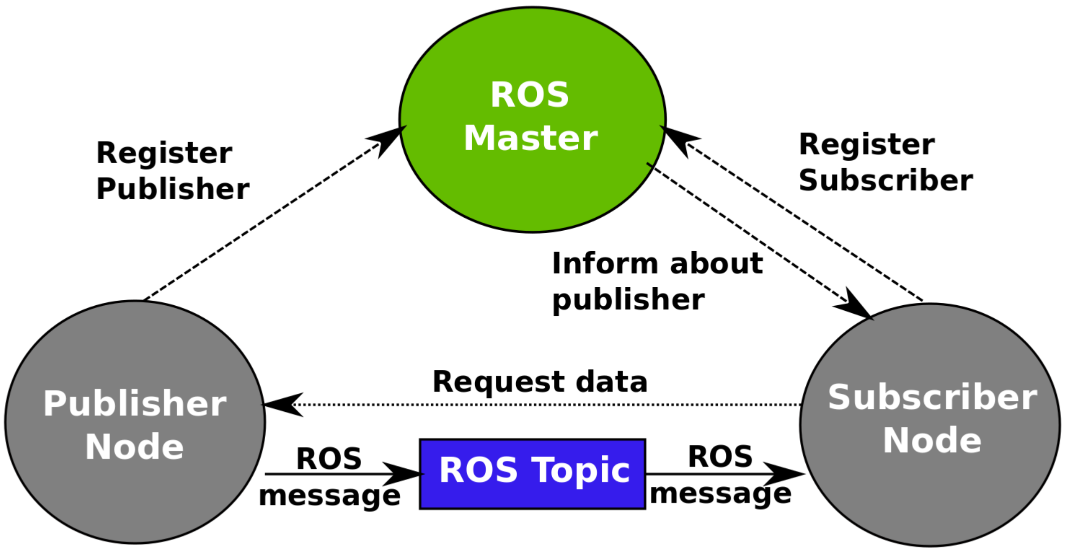


Figure 3.4: Communication Between ROS Nodes Using Topics.

also possible through the use of *topics*. The communication between nodes uses the standard Transmission Control Protocol/Internet Protocol (TCP/IP) and is established by the *ROS Master* that provides naming and registration services to the nodes in the ROS system, so that they can locate each other. A node can act as a *ROS Publisher*, sending messages to a topic, as a *ROS Subscriber*, listening to messages published on a topic, or as both (see Fig. 3.4). This abstracted functionality allows easy and robust communication between different nodes (processes), a very desired property in robotic applications that contain multiple components, without having to reinvent the wheel.

This thesis negotiates systems of high complexity, including robot control, multiple sensors, a simulation environment, a reinforcement learning agent and its interaction with the environment. Therefore, having access to a direct way of establishing communication between all of those components is of paramount importance. Furthermore, being able to transfer the technologies that are trained and tested in the simulation to the real world is also one of the main goals of this work. ROS makes this transition possible with only minimum interventions in the code, as long as the real robot has been made compatible with ROS. Because of these powerful capabilities it offers, ROS has been chosen as the main framework of this thesis, acting as the backbone of the various system implementations that will be presented in the following sections and chapters.

3.3.2 Gazebo

Gazebo [128] is an open-source 3-D simulation tool for designing and testing robots. Gazebo offers multiple robust and high fidelity physics engines, a wide variety of sensors useful for robotic applications, as well as programmatic and graphical interfaces. Additionally, Gazebo provides a set of APIs for interfacing with ROS, which makes the development of robotic applications very convenient. The same code that is run in simulation can also be run on the physical robot with minimal or no changes.

Advantages of Simulation in Robotic Technology Development and Reinforcement Learning

Training reinforcement learning policies is hard and usually very time consuming, since typically a great number of data samples is required to learn useful behaviours for solving the task at hand. Furthermore, those samples are not available before training, as in supervised learning, but rather need to be collected by the agent that is interacting with the environment. However, using a real robot to collect the required experience comes with multiple challenges, which might make the learning process infeasible. On the other hand, a simulation environment can tackle many of those challenges and allows learning RL policies in a more reliable, fast, safe and repeatable way. In this regard, a simulator:

- Is low-cost and low-risk
- Provides easily customizable virtual space for establishing environments for training and testing
- Allows testing of different sensors and sensor arrangements, without need of the actual hardware
- Allows speeding up the real-time factor (faster simulation steps), and consequently the training time

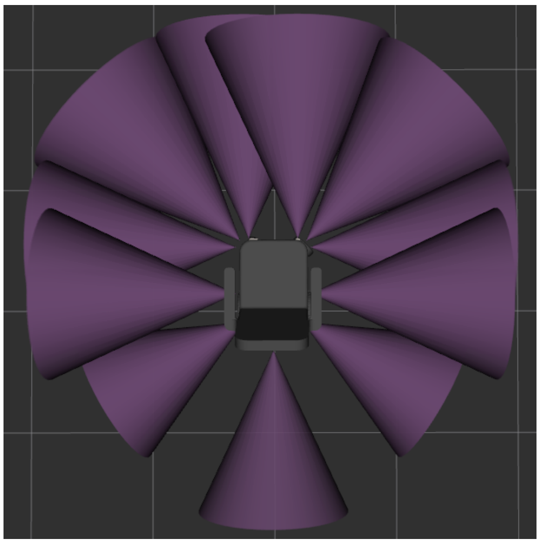


Figure 3.5: The Virtual Wheelchair Model. Each cone is a visualization of an ultrasonic distance sensor's field of view.

This thesis uses Gazebo as the main tool for training and testing the reinforcement learning agents that will be described further on, within custom made virtual environments. For that purpose, a virtual robotic wheelchair has also been designed, in a way to closely match the one used in the real physical experiments in terms of sensor setup and control. The next section will describe the virtual EPW in detail.

3.3.3 The virtual EPW

In Gazebo, a robot is composed of "links" and "joints". Links represent the solid parts of the robot and come with collision, inertial, and visual properties. On the other hand, joints are the movable parts of the robot (except for the "fixed" joint type), connecting links, and possessing kinematic and dynamic properties.

Each link is defined by its shape, which can be a basic type like box, sphere, or cylinder, or a more detailed description using an appropriate mesh file. The collision tag makes the link a rigid body, while the inertia tag specifies the moment of inertia (MOI) of that rigid body with mass m. The visual tag enables the link to become visible in Gazebo. Joints, on the other hand, must be defined as specific types (*e.g.*, fixed, continuous, revolute, prismatic), which determine the type of movement they can execute. To fully define a joint, it is necessary to provide the links it connects and its axis of movement. All of these aspects are defined in a Universal Robotic Description Format (URDF) file, the established format for describing robots in ROS, comprising various XML elements nested in hierarchical structures. To make the URDF file usable in Gazebo, it needs to be converted to a Simulation Description Format (SDF) file, which not only specifies the robot structure but also describes the world in which the robot exists.

The 3D wheelchair model used in this thesis was adapted from the Argallabs smart wheelchair project [129] (see Fig. 3.5). The wheelchair employs a differential drive, a two-wheeled drive system where each wheel is controlled independently. Additionally, there is a pair of free-rolling wheels (caster wheels), allowing the robot to rotate almost on the spot. The original model was modified from a front-wheel drive to a rear-wheel drive wheelchair model using URDF, and various properties such as dimensions (chassis, drive/caster wheels, drive axle length, etc.), mass, and inertia of the wheelchair were adjusted to match the specifications of the in-house real-world wheelchair used in the experiments described in later chapters. In addition, various sensors, which will be detailed in the next section, were added as links to the EPW model. A contact bumper sensor was also included on the main frame of the EPW model to detect collisions in subsequent simulation experiments.

For simplicity, the moment of inertia tensor of the wheelchair's base can be calculated assuming the chassis has the shape of a solid cuboid with width w, height h, depth d, and mass m. In this case, the inertia of the base is given by:

$$I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+h^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+d^2) \end{bmatrix} \tag{3.16}$$

Fig. 3.6 depicts the communication among various components in the simulation through the ROS framework, including the simulation environment, wheelchair sensor data, and the controller, which accepts velocity commands and publishes the robot's odometry. The "RobotNode" is the high-level process that takes the simulation and wheelchair data as inputs and generates driving commands for the

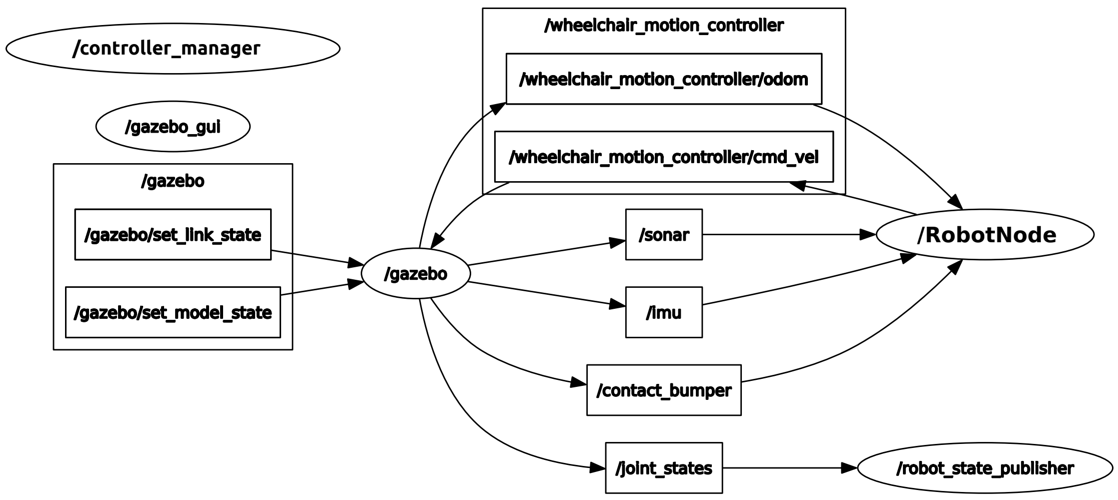


Figure 3.6: ROS Gazebo Agent Graph. The graph shows how the different components of the system communicate with each other through the use of ROS topics.

virtual wheelchair.

3.4 Methodology

3.4.1 MDP Formulation

This chapter focuses on an end-to-end learning framework for performing mapless autonomous navigation in complex environments. The problem is modelled as an episodic MDP, with a finite time horizon T. As described in Section 3.2.1, a MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R})$, where $\mathcal{S}$ is a set of continuous states, $\mathcal{A}$ a set of continuous actions, $\mathcal{T}$ an unknown transition function, γ a discount factor, and $\mathcal{R}$ the reward function. The discrete timestep within an episode is denoted with $t \in \{0, 1, .., T-1\}$. The state transition dynamics $\mathcal{T}(s_{t+1}|s_t, a_t)$ map a state-action pair at time t onto a distribution of states at time $t+1$, with $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$, however, they will not be used here since a model-free approach will be followed. The agent, at each timestep, chooses its actions according to a policy $a_t = \pi(s_t)$, then applies them to the environment, observes a new state s_{t+1} and receives a reward $r(s_t, a_t) \in \mathcal{R}$. Given this setting, the goal is to find an optimal policy $\pi^\star$ that achieves the maximum expected return $G_t = \sum\limits_{t=0}^{T} \gamma^t r(s_t, a_t)$ from all states.

State Space

Ultimately, the aim of such a system is to be transferred into the real world and for a real purpose, and thus, the arrangement of sensors that was chosen was such to closely resemble the real EPW used in the experimental trials described in Chapter 4 and Chapter 5. The sensors that were used in the virtual model, as shown in Figure 3.5, consist of 11 ultrasonic sensors surrounding the EPW, a pair of wheel encoders that provide odometry information, as well as an *Inertial Measurement Unit* (IMU). The sonar sensors are positioned in a way to fully cover the front view, as well as the sides of the EPW, with minimal overlapping between them. Each of the sonar sensors has a 50-degree field of view and can measure up to 3 meters far.

Utilizing the sensors described above leads to forming of the state space at each

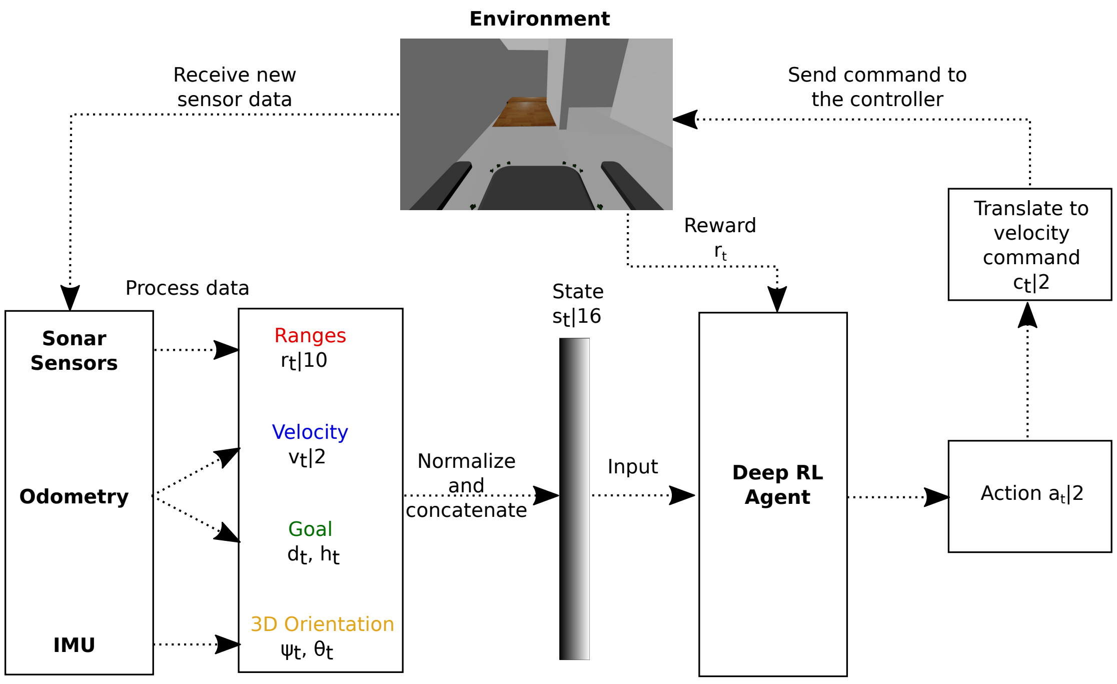


Figure 3.7: Overview of the Autonomous System Control Process.

timestep t which consists of: 11 sonar range measurements ($\vec{r_t}$), the linear and angular velocity of the robot provided by the odometry ($\vec{v_t}$), the distance and relative heading to the current goal calculated from the current robot and goal positions (d_t, h_t), and finally the pitch (θ_t) and the yaw (ψ_t) of the robot converted from the quaternions provided by the IMU.

Instead of directly including the yaw as part of the state, it was observed that it is more useful to integrate the robot's heading information with the angle between the robot and the current goal. More specifically, the difference between the two is calculated, essentially encoding the information of how much the robot would need to turn, in order to align itself with the goal. The process for producing a sensible angle bounded in $[-\pi, \pi]$ is provided below:

The pitch, however, is directly included as part of the state space and it is important to help the agent identify inclinations, for example when climbing up a ramp. In total, the above measurements form a state space of 16 continuous variables, which are normalized in $[0, 1]$ before being fed to the actor-critic networks.

Since the simulation provides an ideal environment, contrary to the real world, artificial noise is injected into the aforementioned measurements in order to train

Algorithm 3 Heading difference between robot and goal

```
1: Inputs $h_t, \theta$
2: Let $dd = 0$
3: $diff = h_t - \theta$
4: $d = |diff| \pmod{2\pi}$
5: if $d > \pi$ then
6:     $dd = 2\pi - d$
7: else
8:     $dd = d$
9: end if
10: if $(diff > 0$ and $diff \leq \pi)$ or $(diff \leq -\pi$ and $diff \geq -2\pi)$ then
11:     $sign = 1$
12: else
13:     $sign = -1$
14: end if
15: dd = dd * sign
    return dd
```

the agent under more realistic conditions. More details on that will be provided in section 4.6.1 of the next chapter, which includes real-world experiments.

Actions

The autonomous navigation RL agent has access to two actions, corresponding to the commands sent to the differential drive controller to move the virtual wheelchair. These actions are the linear and angular velocities, ranging from -1 to 1, facilitating convergence of the neural networks. Inside the environment, these velocities are remapped to the actual desired velocities that will be sent to the controller to drive the EPW. The differential drive kinematics can be expressed in terms of robot velocities as follows:

$$v = \frac{v_R + v_L}{2} \qquad \omega = \frac{v_R - v_L}{d} \tag{3.17}$$

Here, v represents the platform body's linear velocity, ω its angular velocity, $v_{R,L}$ are the drive wheel velocities, and d is the axial distance between the two drive wheels.

The agent's selection of linear and angular velocity determines the specific maneuver executed by the robot (*e.g.*, moving forwards, moving backwards, rotating on the spot, etc.). Throughout the training process, the agent needs to implicitly understand the outcomes of its actions on the robot. Fig. 3.7 provides an overview of how the autonomous navigation system receives sensor data from the environment, processes it to form the state vector fed to the RL agent, and how the agent outputs actions that are translated into velocity commands and sent to the robot's controller for direct control of its movement.

Rewards

The task the agent needs to learn is how to navigate from some point A to another point B, and therefore needs to be rewarded whenever it manages to reach its goal. However, if this was the only reward it received, given its initial randomness and the complex sequence of actions that needs to be followed in order to reach a rewarding state, the reward would be seen very rarely, and thus the convergence of the algorithm would require an unreasonable amount of time. This is known as the *sparse reward problem* in RL. One of the most straightforward and efficient ways to tackle this issue is with *reward shaping.* Reward function shaping attempts to convert a sparse reward scheme to a dense one using domain/expert knowledge, and provide additional information to the agent to accelerate learning. Many different reward functions have been investigated for successfully learning a robust autonomous goal tracking behaviour, and the most successful one is presented below.

To ingrain the agent with the desired behavior, the reinforcement learning signal received from the environment, which is the reward function at timestep t, is defined as follows:

$$r(s_t, a_t) = l_d r_d(s_t, a_t) + l_s r_s(s_t, a_t) + l_g r_g(s_t, a_t) - t_p, \tag{3.18}$$

where l_d, l_s and l_g are scaling factors. r_d is the reward related to the distance difference from the current goal, between two consecutive timesteps ($d_t = \|p_t^{x,y} -$

$g^{x,y}\| - \|p^{x,y}_{t-1} - g^{x,y}\|$, $p^{x,y}_t$ being the robot's position at timestep t and $g^{x,y}$ being the current goal's position)

$$r_d(s_t, a_t) = \begin{cases} \alpha & |d_t| < d_{min}, \\ -d_t & d_t < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.19}$$

The first branch of the above reward penalizes the agent for not moving (when the distance difference is below a specified threshold, denoted as d_{min}, given that the odometry will not return two exactly equal consecutive readings), which is observed in situations such as when the robot gets stuck at a wall corner, or when trying to get up a ramp but does not accelerate enough. The second branch of the positional reward, returns a reward proportional to the distance difference between the current and the previous timestep. This incentivises the agent to close the distance between itself and the current goal.

r_s is an auxiliary sensor reward to penalize dangerous proximity to obstacles, by defining two danger zones, and penalizing when any of the range sensors (indicated as s_{any}) returns a reading smaller than a multiple of the minimum safety range (s_{min}), at which we consider that we have crashed. This reward is designed to provide the robot with safety awareness, and thus act as an extra safety feature for the particular application, given the increased risks of driving an EPW.

$$r_s(s_t, a_t) = \begin{cases} \beta_1 & s_{any} < 1.5 s_{min}, \\ \beta_2 & s_{any} < 2.5 s_{min}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.20}$$

Lastly, r_g indicates the reward for reaching the current goal within a specified radius of 0.5 meters.

$$r_g(s_t, a_t) = \begin{cases} \gamma & \text{if current goal achieved,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.21}$$

The term t_p in the reward function refers to a constant penalty that is given at every timestep, to incentivize the agent to move towards the goal as soon as possible, otherwise it would accumulate a large negative reward. This penalty also prevents the policy from converging to strategies that would lead it to ceaselessly circle around in order to avoid collisions. For crashing the agent is penalized by receiving a total reward of $r(s_t, a_t) = r_{crash}$. Table 3.1 showcases the actual values of the reward function parameters that were used for training the agent.

Table 3.1: Autonomous Policy Reward Function Parameters.

Parameter	**Value**
d_{min}	0.1 meters
α	-0.25
s_{min}	0.5 meters
β_1	-0.25
β_2	-0.125
γ	100
t_p	-0.1
r_{crash}	100

Terminal States

The episode is reset when the robot crashes, when it overturns, as well as when it remains in the same position for several consecutive steps (stuck or not moving), so as not to fill the replay buffer (see Sec. 3.2.2) with useless experience and slow down the training. Crashing is checked for through the range sensor readings, in case they fall below the accepted threshold, and through a bumper sensor installed on the main body of the robot, capable of detecting contacts. Overturning can take place with specific rotational movements around corners, or by falling off a ramp. It can be determined by checking whether the robot's pitch and roll have exceeded their allowed thresholds. The episode also resets when the maximum number of timesteps is reached, which discourages the robot from retaining a low linear speed when not necessary, since it has to complete the track before the episode resets.

3.4.2 Solving the MDP

The solution of the above MDP has been achieved by using the maximum entropy reinforcement learning framework, and more specifically the *Soft Actor-Critic* (SAC) Algorithm [130]. SAC is an off-policy, actor-critic algorithm that is used with continuous action spaces and trains a policy that maximizes a trade-off between expected return and entropy, in order to increase the randomness of the policy and boost exploration. Its objective is given by:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}\,(\boldsymbol{s_t, a_t}) \sim \pi[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{3.22}$$

where $\mathcal{H}$ is the entropy of the policy and α is a parameter that controls the trade-off between the reward and the entropy. Instead of the original algorithm, a modified version was used [131], which automatically tunes the α parameter throughout training, by trying to match a user-defined target for the desired entropy. This is very convenient because it allows setting an arbitrary reward function, without having to explicitly tune its magnitude.

SAC concurrently learns a policy π_θ and two Q-functions Q_{ϕ_1}, Q_{ϕ_2}, which are represented by neural networks with parameters θ, ϕ_1 and ϕ_2 respectively. During training, it stores a collection of $(s, a, r, s', d)_{i=1}^{N}$ transition tuples in a replay buffer $\mathcal{D}$, from which it periodically randomly samples a batch of transitions and performs stochastic gradient descent for minimizing the following loss objectives:

$$L_Q(\phi_i) = \mathbb{E}_{s,a,r,s',d\sim\mathcal{D}}\left[(Q_{\phi_i}(s, a) - y(r, s', d))^2\right], \tag{3.23}$$

where the target y is given by:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{j=1,2} Q_{\phi_{targ,j}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

$$L_{\pi}(\theta) = \mathbb{E}_{s\sim\mathcal{D},\tilde{a}'\sim\pi_\theta}\left[\alpha \log \pi_\theta(\tilde{a}', s) - \min_{j=1,2} Q_{\phi_j}(s, \tilde{a}')\right] \tag{3.24}$$

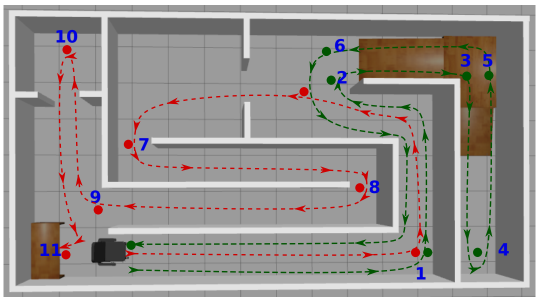
The stochastic policy in the continuous action space, is represented through a diagonal Gaussian distribution. A diagonal Gaussian distribution is a special case of a multivariate Gaussian distribution, described by a mean vector, $\boldsymbol{\mu}$, and a covariance matrix, $\boldsymbol{\Sigma}$, where the covariance matrix only has entries on the diagonal, and therefore can be represented by a vector. The policy uses a neural network with parameters θ, which maps states to mean actions, $\mu_\theta(s)$, and in the case of SAC, also maps states to log standard deviations, $\log \sigma_\theta(s)$.

Given a mean $\mu_\theta(s)$, a standard deviation $\sigma_\theta = e^{\log \sigma_\theta(s)}$ and a noise vector ξ from a spherical Gaussian ($\xi \sim \mathcal{N}(0, I)$), an action sample can be computed with

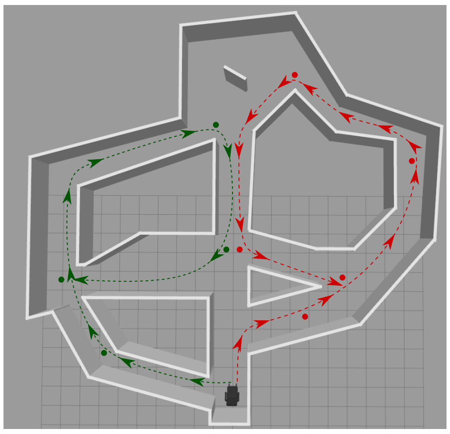
$$a_\theta(s, \xi) = \mu_\theta(s) + \sigma_\theta(s) \odot \xi \tag{3.25}$$

SAC bounds the actions in the finite range $[-1, 1]$, by squashing them through the *tanh* function: $a = tanh(a_\theta(s, \xi))$

The reasoning behind choosing this particular algorithm is four-fold. First, SAC deals with the continuous action space, which is a requirement of this work to allow smoother control of the mobile robot and enable complex maneuvering. Second, the proven success of the particular algorithm for solving similar MDPs in an efficient way, being a sample efficient and stable algorithm. Third, the fact that SAC requires relatively little hyper-parameter tuning compared to other RL algorithms. Fourth, utilizing the maximum entropy RL framework allows learning a general policy that can adapt to unseen conditions, a property which is desirable both for navigating in a variety of different environments, but also for creating a more diverse and realistic user model that will be needed in the next chapter of the thesis. The objective function that is optimized in SAC for the policy network is shown below:



(a) The Training Environment.



(b) The Unseen Environment.

Figure 3.8: The Virtual Scenarios Implemented in Gazebo. The green and red lines indicate the desired paths for the 1st and the 2nd scenario respectively, whereas the coloured dots show the approximate locations of the subgoals along those paths.

3.5 Training and Evaluation of the Autonomous Agent

3.5.1 Training

Setup

For training the autonomous agent, a custom map was designed in the simulation environment, inspired by the scenario defined in the European project, ADAPT[1], to test wheelchair user skills according to the *Wheelchair Skills Test* (WST) [132] protocol. The WST describes a range of driving skills needed for assessing the capacity and performance of an EPW user, including a series of challenging tasks as shown in Table 3.2. The track is split into two scenarios, which in turn include a number of subgoals the agent needs to accomplish, in order to fully navigate the paths as marked in Figure 3.8a. At each training episode, a scenario is selected randomly, as well as a random subgoal within that scenario. The initial pose of the robot is also randomised, within some bounds, along the track. The reason behind this is to integrate diverse experiences for the replay buffer, so that the

[1]Assistive Devices for empowering disAbled People through robotic Technologies

Table 3.2: Wheelchair Skills Test Tasks Integrated in the Training Scenario (Fig. 3.8a).

Subgoal	**Task**
1	Perform 90 degree turn in narrow corridor (1.5m width)
2	Position for ascending inclined ramp (1.4m width)
3	Ascent of 5% incline
4	Descent 10% incline and turn in place in narrow corridor
5	Ascent of 10% incline
6	Descent of 5% incline
7	Pass through 90cm door and perform 90-degree turn in a very narrow space
8	Perform U-turn in very narrow corridor (1m width)
9	Exit corridor performing a wide turn
10	Enter lift area through 90cm door and turn in place to exit lift
11	Position under desk/table (for working, eating etc.)

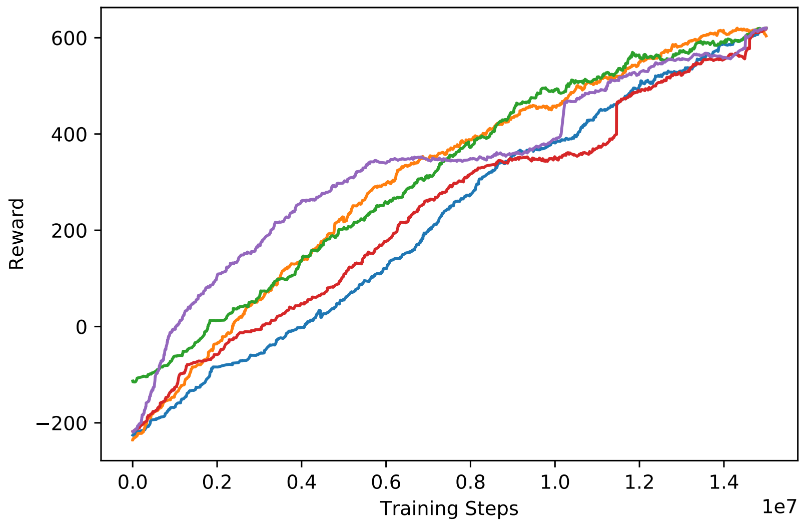
agent learns faster while minimizing the risk of the model getting stuck at a local minimum, by converging early on to a weak policy unable to explore further and improve its strategy.

For the purpose of setting up the reinforcement learning problem and creating the environment for training the autonomous agent, the OpenAI Gym library was employed [133]. Additionally, the well-established Stable Baselines library was utilized [134], offering a suite of reinforcement learning algorithms, with the SAC algorithm specifically employed to train the agent. These choices provided a robust foundation for conducting the training process and facilitated the exploration of various RL techniques in the context of the developed system.

Training details

The SAC algorithm's hyperparameters were tuned through trial and error, resulting in the parameters shown in Table 3.3. The actor-critic neural network architecture utilized standard Multilayer Perceptron (MLP). During training, the performance of the agent was periodically evaluated in a separate test environment, over the span of 5 episodes, in order to identify and save the best one. The maximum linear speed of the robot was set to 0.67 m/s, while the angular velocity was bound in $[-1.3, 1.3]$ rad/s.

Figure 3.9: Average Reward During training for Five Autonomous Navigation Policies with Different Random Seeds. The x-axis represents the training steps, while the y-axis represents the average reward achieved by each policy. The policies are tested to assess their sensitivity to the inherent stochasticity of the training process. By using different random seeds, we can observe variations in the learning trajectories and understand the impact of randomness on policy performance.



Parameter	**Value**
Optimizer	Adam
Learning rate	3e-4
Discount(γ)	0.99
Replay buffer size	10e6
Num. of hidden layers (all nets.)	2
Num. of hidden units per layer	256
Num. of samples per minibatch	256
Nonlinearity	ReLU
Training steps	15e6

Table 3.3: Hyperparameters of the SAC Algorithm for the Autonomous Navigation Policy. The table displays the specific hyperparameters used to train the SAC algorithm, which is employed to develop the autonomous navigation policy, and play a crucial role in shaping the learning process.

Table 3.4: Autonomous Policy Performance Statistics on the Training and Evaluation Environments. For the path planners only success/failure and completion times are marked.

Experiment		**Training env. (Fig. 3.8a)**			**Unseen env. (Fig. 3.8b)**		
Scenario	Method	Success	Collisions	Time	Success	Collisions	Time
Sc.1 (Green line)	RL	88%	0.12	53.5s	86%	0.14	45.2s
	RL(safe)	98%	0	54.7s	94%	0	46.6s
	HUM	87.5%	1.6	77.8s	100%	0.4	47.2s
	TEB	✓	N/A	75.8s	✓	N/A	58.7s
	DWA	✗	N/A	143.1s*	✓	N/A	103.4s
Sc.2 (Red line)	RL	74%	0.26	79.4s	87%	0.13	47.8s
	RL(safe)	89%	0	82.8s	97%	0	49.3s
	HUM	75%	5.9	118.7s	100%	2.4	62.3s
	TEB	✗	N/A	-*	✓	N/A	65.2s
	DWA	✗	N/A	210.9s*	✓	N/A	94.6s

* In case of failure, human intervention was used to free the robot and proceed with the scenario execution. The time before aborting a target and the human intervention time were not considered in the results.

3.5.2 Evaluation

For evaluating the proposed method, the training process was repeated five times, using different random seeds, resulting in five policies representing the best agent of each run. Performing several runs was done to gather quantitative results, and test the method's sensitivity to the stochasticity involved in the training process. Figure 3.9 presents the results, in terms of average reward over the training course. It can be observed, that given enough training time the models converge to similar achieved rewards, demonstrating the method's robustness to randomness between different runs. This robustness is enhanced by enabling the agent to start off at various points of the course, as well as the deterministic nature of the environment. Additionally, all models resulted in high entropy policies (in the context of RL entropy measures the randomness in actions), which indicates their ability to generalize and adapt to situations not experienced during training.

The performance of the trained agents was initially tested on the scenarios they were trained on, for a span of 100 episodes on each scenario, while picking deterministic actions (the mean of the policy's output). An important tweak employed during testing time is the addition of an extra *safety mechanism* on top of the agent's output, to aid collision-free navigation (see Fig. 3.10). In this case, the agent picks stochastic actions (samples from the policy's output distribution), and the mechanism activates when any of the range sensor measurements are below the minimum safety threshold (s_{min}) and the chosen action would move the robot towards the direction of the obstacle. Instead of resetting the episode as in during training, a zero command is instead issued to the controller to prevent the crash, and allow the agent to continue its task if it manages to pick an action from the stochastic policy that would move it away from the obstacle. Table 3.4 presents the average success rate of fully navigating each scenario, the average completion time (in steps), the overall scenario completion percentage, and the average number of collisions per episode of the best performing agent, both with and without using the extra safety mechanism. It is observed that the RL agent scores a high success rate (88%) for the first scenario, while the success drops for the second scenario (74%) due to the very demanding maneuvers that are required for completion. Enabling the safety mechanism drastically improves the success rates of the agent, while reducing the collisions to zero, only for a very slight trade-off in completion times.

The models were then tested in a different, unseen scenario (Figure 3.8b), to evaluate how well the method has generalized in terms of driving behaviour and goal tracking. Compared to 3.8a, this one is less linear, more open-spaced, and requires slightly less complex maneuvers to navigate. Again, two separate paths are designed along the track, each with its respective subgoals, for which the results are presented in Table 3.4. The results suggest that the agent has not overfitted to the specific scenarios it was trained on; rather, it can navigate an unseen environment with high success rates, comparable to those of the training environment. The number and the positions of the defined subgoals are set arbitrarily, but placing them relatively close to each other and before steep changes in the map topology helps the agent progress the track more smoothly, given the absence of global knowledge of the map. Also, in this case, applying the extra safety mechanism results in a significant increase in the agent's success.

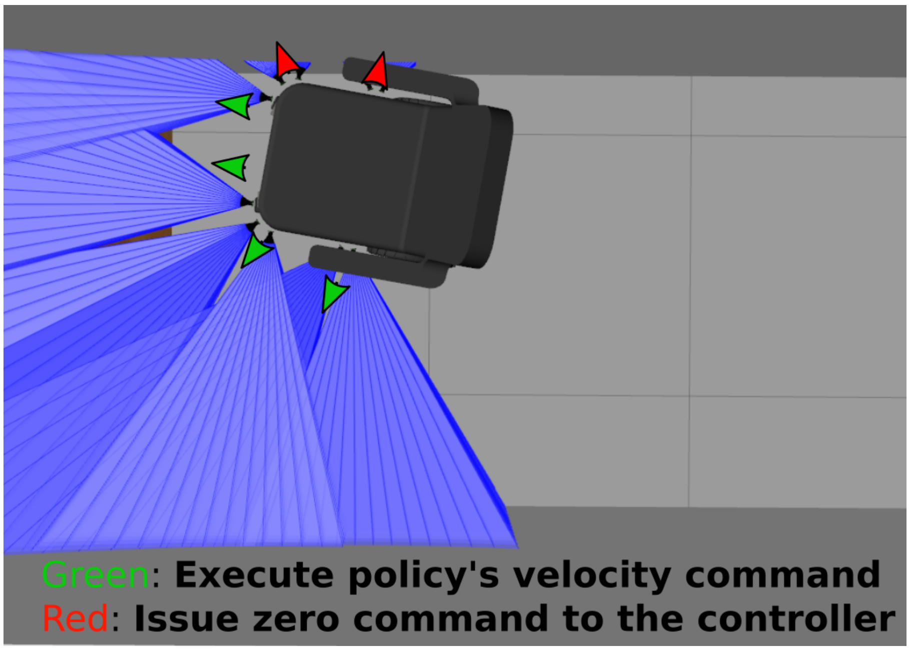

Green: Execute policy's velocity command
Red: Issue zero command to the controller

Figure 3.10: Display of the Extra Safety Navigation Mechanism. Red arrows indicate directions that the safety mechanism prevents from being applied due to the close proximity of obstacles, whereas green arrows show the directions that the safety mechanism allows the robot to move towards since there are no nearby obstacles below the minimum safety threshold.

Baseline Comparison

The deep-RL trained motion planner was compared with human performance, as well as with two widely used motion planners. For a fair comparison, both the humans and the path planners were provided with the same subgoals and velocity limits as the RL agent. A *hardware-in-the-loop* (HIL) simulation was set up for the human trials, incorporating a real joystick controller connected via a USB-to-TTL Serial interface to a Linux machine running the simulation (figure 3.11). Eight healthy human participants were given some time to familiarize themselves with the setup and were then asked to navigate the scenarios as fast and accurately as possible. The trials were not stopped in case of a crash, as was done with the RL agent.

Traditional path planning was implemented through the navigation stack provided by ROS. A 2-D occupancy grid map was initially created, using laser-based SLAM, and the Adaptive Monte Carlo Localization (AMCL) approach was used for localizing the robot in the map. A global plan was produced by utilizing the

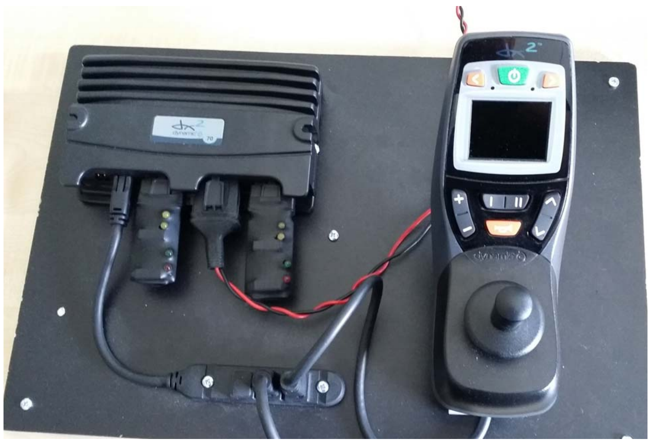


Figure 3.11: Hardware Joystick Controller, that can be connected to a PC through TTL Serial communication. Two fake loads are used to emulate the wheelchair's drive motors and allow the controller to transmit signals.

A* algorithm on the constructed grid map, whereas for generating velocity commands for collision-free path following, both the Dynamic Window Approach and the Time Elastic Bands approach (recall section 2.2.3) were explored.

The relevant results are presented in Table 3.4. It is observed that some of the human participants were not able to complete the scenarios of the training environment, which was due to getting stuck in tight spots and not being able to reverse. Overall, it was quite challenging to avoid crashing, at least not without moving extremely slowly. The path planners were also unable to fully complete both scenarios of the training environment. Even though the global planner managed to create a valid path to the goals, the local planners were unable to produce the necessary moving commands to guide the robot through specific spots, and thus the navigation was aborted. With DWA the robot speed was quite slow, while the planner was particularly problematic with rotating on the spot. On the other hand, TEB's overall performance was better than DWA, producing smoother trajectories and driving the robot faster. However, it was still unable to navigate the robot through tight spots and narrow pathways, despite significant efforts to tune it properly. These results occurred even after extensive tuning of various parameters of the planners, such as the inflation radius of obstacles in the costmap, the footprint of the robot, path and goal-related cost factors, etc. In the unseen environment, which is simpler in terms of the required maneuvers, both the humans and the planners were successful in navigating the given scenarios.

Table 3.5: Autonomous Agent Observation Space Ablation Results.

Experiment		**Metrics**				
Scenario	Model	Suc^{S}	Suc^{G}	Collisions	Steps	Rewards
Sc1 (green)	Baseline	88%	93.3%	0.12	1070	503.4
	+2 Past	93%	96.3%	0.07	1089	538.6
	No IMU	0%	61.3%	0.42	2155	314.7
	LiDAR	0%	68.75%	0.06	2688	702.1
Sc2 (red)	Baseline	74%	83.2%	0.26	1588	297.9
	+2 Past	80.0%	89.25%	0.2	1693	375.9
	No IMU	0%	44.6%	0.44	2307	93.4
	LiDAR	0%	33.2%	0.84	1017	123.8

Ablation study for the Observation Space

An ablation study was also performed, in order to determine the importance of including specific observations in the state space. More specifically, this section explores the effect on the performance of the agent when it cannot observe its orientation through the IMU sensor, or when a history of past measurements is added to the observation space. It also explores the potential of learning a successful model, if instead of the chosen sonar arrangement, a LiDAR is used as the only means of accessing information about obstacles in the environment, as has been done in other works [66].

For better evaluating the performance between RL agents, the recorded metrics include the full scenario completion rate (reaching all goals), the partial completion rate (number of goals reached), the steps per episode (before the episode is terminated due to completion, crashing or reaching the time limit) and the reward per episode, averaged over a span of 100 test episodes. Table 3.5 presents the performance of each policy evaluated on the training scenarios. The results show that including a history of past observations at each state improves the baseline model in terms of success and collision rates. This result is expected since including information about the past introduces a form of memory to the model, that helps it take better decisions in situations where only the current sensor measurements are not enough to decide the optimal action (*e.g.*, steep changes in the map topology). Concatenating past information to the current state has also been shown to be effective in other works [120], whereas for properly integrating memory into the model, a different neural network architecture would be required (*e.g.*, recurrent connections). However, this was purposely omitted in this work to avoid significantly higher training and inference times.

Excluding the IMU measurements from the observations of the agent, also seemed to have a significant effect on the trained policy, with the success rates dropping compared to the baseline model. This is mostly accounted to the absence of information regarding the robot's heading. Since the wheelchair is a bulky and non-symmetrical robot (in contrast to the turtlebot platform for example, which has been extensively used in closely related works [66, 102]), having a sense of its own heading provides valuable information for executing fine maneuvers and for being able to accurately orient itself towards a goal position. Finally, replacing the sonar sensors with a single forward-facing LiDAR sensor, installed at the front of the wheelchair, also resulted in a steep decline in performance. Again, due to the geometry of the wheelchair, a single LiDAR does not provide enough information about the surrounding obstacles to allow the execution of certain maneuvers without resulting in a collision.

Continuous vs Discrete Action Space

This section investigates the importance of using a continuous action space versus a discrete one for performing navigation in challenging environments. A new policy is learned through the popular DQN (more specifically an improved version of the original algorithm, using a dueling network architecture and double q-learning), and is compared to the best-performing continuous agent. In this case, the observation space, as well as all of the task-related configurations are kept the same as with the baseline SAC model. The only difference in training the DQN model is that the action space is discretized so that the agent can choose one out of 15 discrete actions at each timestep. Each action corresponds to a pair of a linear and an angular velocity command, where the linear command component can take one out of three values evenly spaced in $[0, v_{xmax}]$, and the angular command can take one out of five possible values, evenly spaced in $[-\omega_{zmax}, \omega_{zmax}]$.

Results show a steep decline in performance when the discrete model is tested on the training environment. More specifically, in the first scenario the DQN agent achieves goals with a 75% partial completion success rate, compared to 93% of the baseline model. In the second scenario, which demands finer maneuvering to navigate the narrow corridors, the partial success rate drops even further to 37% compared to 83% that the continuous agent achieves, whereas the DQN agent never manages to fully complete the scenario, with a crash rate of 100% at goal 7 (perform 90-degree turn in very narrow space). This result justifies the use of the continuous action space to navigate complex indoor environments, where a limited degree of control freedom is not sufficient to perform the required maneuvers for successful navigation. Allowing the discrete agent a larger number of available actions could tackle this issue, but would result in a more challenging and time-consuming training process.

3.6 Discussion

The aim of this chapter was to introduce an end-to-end autonomous navigation method for mobile robots, by leveraging a realistic sensor setup without requiring the existence of a map. The method was specifically designed for use with powered wheelchairs, where a simulated model was utilized to learn a goal-following navigation policy through reinforcement learning, within a virtual scenario that contained a series of tasks inspired by the WST protocol. Results showed that the proposed method can effectively learn policies capable of reaching challenging goal locations, in a collision-free manner, and adapt to situations not experienced during training. The method was compared with state-of-the-art planners, human performance, and also variations of itself, highlighting the importance and the effect of different design choices, especially regarding the observations available to the agent.

Including human participants in this study was mainly aimed at putting into perspective the challenging task of controlling a bulky mobile platform. Naturally, the perceived task for a user would be quite different in the real world compared to the simulation, but still, the participants who focused on completing the trials as fast as possible had a significant number of collisions, whereas those who focused on accuracy had large completion times. This indicates the challenge of combining speed and accuracy for non-expert users when fine maneuvering is required, and without the help of an assistive system. The traditional path planners both failed in completing the most challenging scenario, while they were successful in the easier tasks. However, they both required extensive tuning and proved to be slower in achieving the tasks, while being significantly more computationally intensive compared to the RL approach. When measured on the same hardware, the average frequency at which drive commands were sent resulted in 7 Hz with TEB, 13 Hz with DWA, and 115 Hz with RL (without fixing the control timestep). This is an important factor to consider when deploying the solution in a real-world robot with an onboard computer.

Using a sensor setup that can provide sufficient information about the obstacles surrounding the robot proved to be of paramount importance, as when the sonar sensors were replaced with a front-facing LiDAR sensor the performance of the autonomous policy dropped dramatically. The results also showed the benefit of adding some memory capabilities to the model, as concatenating past observations to each state improved the resulting model's performance. The IMU sensor, which can provide the agent with a sense of the robot's orientation also seemed to play an important role in the policy's maneuvering and goal-following capabilities. Using a discrete action space, with a limited number of available commands, did not allow the agent to complete the most challenging tasks, as it led to rough navigation behaviors compared to the much richer continuous action space.

Finally, an additional safety mechanism was introduced, which works on top of the trained policy and was shown to greatly aid in avoiding collisions. A more sophisticated approach would be to calculate the admissible velocities as in the DWA (trajectories that do not lead to a collision within a given control time window) and prevent the execution of non-safe trajectories. However, that would incur an additional computational cost, while a simpler implementation as the one presented in this work seemed to work well enough, reducing the collisions to zero when used along with the autonomous agent.

A limitation of the proposed work is that it does not directly deal with the localization problem, which is a critical issue to address for transferring the learned policy to the real world. The odometry that is calculated solely from wheel encoders tends to accumulate error over time, leading to wrong estimations of the robot pose, and therefore erroneous goal-related observations for the autonomous agent. This issue could be tackled with the current design, by using an EKF to fuse encoder and IMU information and provide more robust robot pose estimates. Alternatively, another localization method from the ones presented in Sec. 2.2.1 could be utilized.

3.7 Chapter Summary

This chapter proposed an RL-based method for autonomous wheelchair navigation in complex indoor environments, only relying on low-cost onboard sensors. More specifically, the chapter presented an end-to-end system, which takes in sonar and IMU sensor measurements, along with a desired goal's information, in the form of polar coordinates, and directly outputs control commands for the powered wheelchair. The system was trained in simulation and utilized the maximum entropy RL framework, and more particularly the Soft Actor-Critic Algorithm, to learn a robust navigational policy in the continuous action space. The proposed method is also applicable for any non-holonomic, differential drive robot, however, particular emphasis was given to the sensor arrangement, and also the design of the task and the reward function, to account for the specific needs of an EPW, in terms of its geometry and the requirements of fine maneuvering and safety. Results in simulation show that the learned policy can effectively complete challenging tasks and also adapt to new ones, with the aim lying in providing an option for effortless and safe navigation in the daily routine of EPW users. On top of the policy's output, a simple but effective control mechanism can be applied to further promote safety. The proposed method is able to handle cases the baseline methods are unable to while being significantly less computationally intensive.

CHAPTER 4

A Novel RL-based Shared Control Method

This chapter presents the main objective of the thesis: a novel shared control methodology for assisted navigation, building upon the autonomous system introduced in the previous chapter using maximum entropy RL. The modified formulation ensures compliance with the user's intentions by replacing the added entropy term with a user-modeling-based term, minimizing the difference between agent and user actions represented as a probability distribution. The introduction of a "state risk" formulation aids in identifying cases where the system must deviate from the user's commands to prevent harm. By leveraging the autonomous agent from Chapter 3 as a disabled user model with artificial noise injection, we train a compact neural network to produce real-time corrective driving commands for wheelchair users, making minimal assumptions about users or the environment. The experimental section encompasses simulation and real-world experiments, showcasing improved goal-reaching and safety for simulated user models. Real-world validation demonstrates successful user guidance with a strong emphasis on safety, albeit with a slight trade-off in user satisfaction.

4.1 Introduction

The definition of shared control and the various methods found in literature of realizing it, along with their limitations, have already been discussed in Sec. 2.3.

The motivation behind the work presented in this chapter is to design an end-to-end system that is able to assist the user in a meaningful way, while making the minimum possible assumptions about the world and the user's goals. Ideally, the system should be able to follow the user's immediate intention, but gently modify their input when needed, to produce a safer and more effortless motion. Another requirement of the system is to be able to provide assistance in the continuous action space, while being fast enough to run on the onboard electronics of a mobile robot.

This chapter adopts a policy-based approach to shared control, by designing a system that optimizes some cost function to directly output a control action for the robot, while taking into account the user's input. This is achieved through the RL framework, and more specifically by modifying the SAC algorithm, which was also used in the previous chapter to train an autonomous agent. The autonomous agent is utilized here to emulate a user that will be part of the training process.

The rest of the chapter will describe the different components of the proposed system and will highlight the relevant contributions. The different aspects of the system are put under inspection through a number of experiments and comparisons run in simulation. The system is also transferred to a real robot (EPW), where further experimentation is conducted to test the proposed method's effectiveness, transferability to the real world, and its ability to assist actual human users, while only having been trained with a simulated user.

4.2 Simulating User Driving

Sec. 2.3 introduced the definition of shared control, summarized as an instance of human-robot collaboration, where a human user and a robotic system are interacting congruently in a perception-action cycle to arrive at the decisions needed to achieve some goal. Since this thesis explores a learning-based approach to the shared control problem, training such a system requires the existence of user input data, which is the very input the system should learn how to correct, when correction is needed.

If the shared control problem was framed as a supervised learning problem, it would be possible to collect human data, along with environmental observations, label them in an appropriate way (if the optimal actions were known), and thus create a dataset through which a prediction model could be learned. However, in the case of reinforcement learning, the learning is achieved through constant interaction between the agent and the environment, and therefore it is not possible to use pre-collected data as one would in a supervised learning setup. Rather, there needs to be a user, or a model of a user, providing a continuous stream of commands while the learning takes place.

Of course, having a real person producing control commands throughout the training of such a system would be highly impractical, if not infeasible, due to the sheer number of interactions that are needed for the convergence of the system. Another idea would be to utilize a path planning algorithm, along with a motion controller and a localization algorithm (similar to the autonomous navigation approaches presented in Sec. 2.2), in order to produce moving commands for the robot in the place of an actual user. However, this approach would also be problematic due to the high computational demands of such algorithms, the need for a map, and their deterministic nature, which would result in a slow and predictable user model. Finally, a user model could be learned through expert demonstrations (*e.g.*, imitation learning/behavioral cloning), but that would require high-quality and sufficiently diverse data, which would be hard and time-consuming to collect.

Instead, this thesis proposes the use of a trained, autonomous reinforcement learning agent to act as the user. This approach satisfies several requirements that make it appropriate to use for the purposes of this work:

1. The autonomous navigation agent can provide a single driving command "on-demand" at any given timestep, which follows the same format as the training process for the shared control system.

2. The autonomous navigation system shares similar sensory input with the shared control system (as will be seen later in this chapter), which allows for an efficient implementation by not requiring additional sensors and by

reusing the already processed data at each learning step.

3. The decision-making of the autonomous navigation system is quite fast (recall Sec. 3.6), therefore, it will only have a minor impact on the training time of the shared control system. This would not be the case if using a traditional navigation method which is quite computationally demanding.

4. The autonomous navigation agent can be perturbed with additive noise, but still take "rational" decisions to navigate towards a goal, similar to what a user with some disability would do. The following section further clarifies the importance of this point.

The agent that will be used in the place of a human user is the optimal agent of the work presented in Chapter 3. The next section discusses how the optimal trained policy can be corrupted to simulate disabled user input.

4.2.1 Erroneous User Input Models

The purpose of the shared control system is to slightly modify the user input, when needed, in order to improve driving performance and avoid risky situations (*e.g.*, collisions). However, the system should fully respect the user's intention when the user drives optimally. The autonomous driving agent of Ch. 3 was trained for optimal performance, and as shown by the relevant results (Sec. 3.5.2), it is expected to navigate towards targets with significantly high accuracy, while avoiding collisions. Therefore, directly providing its control commands as user input for the assistive system would not be very helpful, since the assistive system would not get to experience faulty driving behavior to learn how it should correct it accordingly.

Instead, the output of the autonomous agent should be modified to mimic a disabled user and provide flawed input for the assistive system. For that purpose, the control actions suggested by the autonomous agent can be "polluted" by artificial noise, before being utilized as user input. Depending on the type of noise that is injected into the stream of the autonomous commands, it is possible to create a variety of disabled user models. The work done within the scope of this thesis is not to accurately simulate specific real-world disabilities, but rather provide a noisy, and as diverse as possible range of inputs to the assistive system so that it can learn how to provide assistance within a wide range of situations and user behaviours.

The different types of noise and the resulting faulty user models that have been explored in this work are presented below:

1. *Laggy user*: A "laggy" user is a user with a slow reaction time (due to either mental or physical limitations), that lags in changing their control actions. Such a user can be modelled by repeating the previous control action, instead of generating a new one, at each timestep, with a fixed probability p. This way, each action is repeated for a number of steps that follows a geometric distribution.

Algorithm 4 Laggy User Action Selection

Input: User input at timesteps t and $t-1$, $\vec{u_t^h}, \vec{u_{t-1}^h}$, the probability of repeating the previous action, p
Output: Corrupted action, $\vec{u_t^c}$

```
1: Sample random number in [0,1] from the uniform distribution, $X \sim U(0,1)$
2: if $p > X$ then
3:     $\vec{u_t^c} = \vec{u_{t-1}^h}$
4: else
5:     $\vec{u_t^c} = \vec{u_t^h}$
6: end if
7: return $\vec{u_t^c}$
```

2. *Shaky user*: A "shaky" user is a user that lacks accuracy in their control actions due to poor motor control (*e.g.*, Parkinson's disease). Such a user can be modelled by injecting Gaussian noise of a fixed standard deviation σ around their input.

Algorithm 5 Shaky User Action Selection

Input: User input at timestep t, $\vec{u_t^h}$, action space bounds a_{LOW}, a_{HIGH}, standard deviation of Gaussian noise, σ
Output: Corrupted action, $\vec{u_t^c}$

```
1: Calculate n samples from Gaussian distribution, equal to the number of actions, $X_1, X_2..X_n \sim \mathcal{N}(0, \sigma^2)$
2: Add noise to the user's action, $\vec{u_t^c} = \vec{u_t^h} + \vec{X}$
3: Clip resulting actions to be within the allowed bounds, $\vec{u_t^c} = clip(\vec{u_t^c}, a_{LOW}, a_{HIGH})$
4: return $\vec{u_t^c}$
```

3. *Twitchy user*: A "twitchy" user is a user that sometimes completely loses control of their actions, possibly due to some neurological condition. Such a user can be modelled by selecting a completely random action (sampled from the set of available actions), at each timestep, with a fixed probability p.

Algorithm 6 Twitchy User Action Selection

Input: User input at timestep t, $\vec{u_t^h}$, the action space A, the probability of choosing a random action, r
Output: Corrupted action, $\vec{u_t^c}$

```
1: Sample random number in [0,1] from the uniform distribution, $X \sim U(0, 1)$
2: if $r > X$ then
3:     Sample a random action from the action space, $\vec{u_t^c} \sim A$
4: else
5:     $\vec{u_t^c} = \vec{u_t^h}$
6: end if
7: return $\vec{u_t^c}$
```

4.2.2 User Intent as a Probability Distribution

Given a noisy user model, its outputs can be used as the desired control actions which should be corrected, if needed, by the assistive system. However, since the aim here is to build a generic assistive system that does not necessarily have any prior knowledge of the user's disability or goals, it is only the noisy command that gets exposed to the system, and not the original command or the type of noise that was injected to it.

That being said, by simply assuming that there is potentially some error associated with the provided command, the noisy command can be then represented as a probability distribution over actions by the assistive system. The usefulness of that will become evident in the following section, where the methodology will be explained in depth. Another assumption that is adopted in this work, is that the user's velocity distribution is solely based on the current input command, which should approximately indicate their intended goal trajectory. Assuming a memory-less probability distribution simplifies the framework, while it allows users to change their minds and is more tolerant to input errors.

Even though many different distributions can be used to model the noisy action, for reasons of practicality and convenience, we adopt the formulation presented in [93], where the intent is modeled as a normal distribution over the range of velocities centered on the indicated direction. Hence, the noisy command will be modeled as a random variable $\boldsymbol{u^c}$ sampled from a diagonal Gaussian distribution, with the mean vector $\boldsymbol{\mu}$ centered on the current commands and a diagonal covariance matrix $\boldsymbol{\Sigma}$ chosen as part of the system design.

$$\boldsymbol{u^c} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{4.1}$$

Alternatively, a Gaussian distribution could be fitted to a window of n past commands, which could be weighted so that the most recent commands would have a larger impact on the resulting distribution. This would introduce some form of short-term memory of the user's intentions and would dynamically adjust the covariance matrix. However, such an approach would require more tuning (the size of the window and the respective weights) and would further complicate the analysis of the proposed framework's effectiveness. Therefore, in this work we will only consider modeling the most recent user command as an indication of the user's intentions.

4.3 User Compliant Reinforcement Learning

This section presents the proposed methodology for realizing shared control through deep reinforcement learning, by modifying the maximum-entropy RL framework to incorporate the user's intention as a prior distribution that guides the agent's exploration.

4.3.1 Maximum Entropy RL Objective Modification

A non-negotiable requirement of a shared control system, especially one designed to enhance a disabled person's autonomy, is its ability to closely follow the user's intentions and intervene only when necessary, rather than solely prioritizing other objectives and ignoring the user's input. In other words, such a system should place particular focus on respecting the user's intent.

In this work, explicit goal inference is not performed. Instead, the user's actions directly influence the assistive policy, which must learn to implicitly decode the user's intent. The agent perceives its environment based on observations from various sensors, and it considers the user's intention, another source of observations, as part of the external environment. Thanks to the expressive power of artificial neural networks, the agent can discover complex relationships between user controls and observations of the physical environment without explicitly assuming the existence of a goal.

Unlike most approaches to shared control that utilize blending strategies between the user's intention and an autonomous controller's suggestion, the proposed policy-based system directly outputs the final command for the robot. However, a fundamental challenge of this approach is adapting standard reinforcement learning to closely follow the user's intentions. The system needs to be able to leverage human input without significantly interfering with the goals they are trying to achieve. Additionally, ignoring the user's input, despite it being a source of irritation, can negatively affect the user's feedback loop, where they observe the consequences of their actions and adjust their inputs accordingly.

The most straightforward approach to address this problem would be to encode the desired behavior of respecting the user's intention within the reward function. However, designing such a reward function that enforces action similarity has several issues. For instance, it might lead to conflicting terms in the reward function, possibly rewarding risky actions that match the user's input but lead to crashes. Achieving numerical balance by tuning all reward terms is challenging and may cause the agent to exploit the reward in unpredictable ways.

In addition to the challenge of enforcing action similarity, there is the issue of quantifying the similarity itself. Numerous distance metrics exist, each leading to different learned behaviors. Attempts to embed action similarity within the reward function using metrics like Euclidean distance and cosine similarity proved unsuccessful, despite extensive parameter fine-tuning. These approaches resulted in the robot either following random movements while avoiding crashes or sticking too closely to the user's input, leading to poor assistance. Consequently, it became evident that action similarity should be treated as a separate objective, distinct from maximizing rewards.

A similar challenge was addressed in the literature by Reddy *et al.* [103]. They used deep Q-learning to approximate a state-action value function, modifying the algorithm to choose a high-value action closest to the user's input, and with the tolerance of the system to sub-optimal actions controlled by a hyper-parameter $a \in [0, 1]$. However, this approach is limited to discrete action spaces, which is not applicable to our work dealing with continuous action spaces.

To enforce action similarity in the continuous action space, we treat both the user's input, as well as the agent's actions as probability distributions, and aim to keep those distributions close together. The user's input can be modeled as a probability distribution, as shown in Sec. 4.2.2. Modeling the agent's actions as a probability distribution can be done simply by using an RL algorithm that deals with continuous, stochastic policies. As described in Sec. 3.2.3, SAC is such an algorithm, of the actor-critic family, that learns a diagonal Gaussian policy. Revisiting the objective function 3.22 of maximum entropy RL, it is reminded that the policy is optimized for maximizing the future rewards, as well as maximizing the entropy (*i.e.*, the randomness) of the selected actions.

Maximizing the policy's randomness aims to address the exploration-exploitation problem and enhance the policy's generalization capability. However, since the assistive system involves a human-in-the-loop, it does not need to explore the entire action space to provide a useful output. Instead, the system should provide an output as close as possible to the user's intention and only correct it when necessary, without deviating significantly from the original command. Therefore, in order to guide the policy towards a more efficient exploration strategy for the problem of shared control, the second term of equation 3.22 is replaced with another term, which minimizes the discrepancy between the user's input and the system's output.

To achieve this, we utilize the *Kullback-Leibler* (KL) *divergence* [135], also known as the *relative entropy*, to measure the similarity between the user's probability distribution over actions and the policy's distribution. The KL divergence, commonly used in statistics and machine learning, is a measure of similarity between two distributions P and Q of a continuous random variable, and is defined as:

$$D_{KL}(p||q) \overset{\text{def}}{=} \int p(x) \log \frac{p(x)}{q(x)} dx \tag{4.2}$$

The KL divergence is calculated between the user's action distribution $p(a^h \mid \cdot)$, assumed to be a diagonal Gaussian (recall Sec.4.2.2), and the policy's distribution $\pi_\theta(a \mid s)$, which is also a diagonal Gaussian (recall Sec. 3.4.2). The dot $(\cdot)$ in the user's distribution indicates the user's own private state based on which they select their actions, and which is not accessible to the system, while the policy's actions are conditioned on the current state accessible to the agent.

The choice to model the user's distribution as a Gaussian is motivated by the closed-form solution for the KL divergence between two Gaussians. This avoids the need for computationally intensive methods like Monte Carlo rollouts for approximation. The calculation of the KL divergence between diagonal Gaussians can be further simplified by evaluating the KL divergence between the respective univariate distributions at each action dimension and then summing over them:

$$D_{KL}(p||\pi) = -\frac{1}{2}\sum_{k=1}^{n}\left(\log\frac{\sigma_{\pi k}}{\sigma_{p_k}} + \frac{\sigma_{p_k}^2 + (\mu_{p_k} - \mu_{\pi k})^2}{2\sigma_{\pi k}^2} - \frac{1}{2}\right) \tag{4.3}$$

It's important to note that the KL divergence is not symmetrical. Given two distributions p and q, $D_{KL}(p||q) \neq D_{KL}(q||p)$. When p is the true distribution to be approximated by a prediction distribution q, $D_{KL}(p||q)$ is known as the *forward KL divergence*, and $D_{KL}(q||p)$ is called the *reverse KL divergence*. In this context, the user's action distribution p is assumed to be the true distribution, while the policy π is the prediction distribution trying to approximate p.

Minimizing the reverse KL divergence would lead the policy π towards a *mode-seeking* behavior, where it effectively ignores parts of the true distribution that are probable, as long as it covers other probable parts. However, for our objective of enforcing action similarity, the forward KL divergence is chosen. This results in a *mode-covering* behavior, where the policy π must cover all regions of high probability in p, but is not penalized for having high probability masses where p does not, such as an action that avoids a crash.

By replacing the entropy term of the original maximum entropy RL objective with the forward KL divergence term, the new objective becomes:

$$J(\pi) = \sum_{t=0}^{T}\mathbb{E}\left(\boldsymbol{s_t}, \boldsymbol{a_t}\right) \sim \pi\left[\underbrace{r(s_t, a_t)}_{\text{reward term}} - \alpha\underbrace{\mathcal{D_{KL}}[p(a_t^h|\cdot) \parallel \pi(a_t|s_t)]}_{\text{forward KLD term}}\right] \tag{4.4}$$

where a_t^h is the action chosen by either the autonomous agent or the human driving the EPW.

This new objective now encourages both the maximization of the return and the minimization of the divergence between the system's output and the user's desired input. The next section explains how the parameter α, which controls the trade-off between the two terms of the objective function (Eq. 4.4), can be automatically adjusted.

4.3.2 Adjusting User Autonomy

In the original Soft Actor-Critic algorithm [130], the authors introduced a temperature parameter α to balance the trade-off between the reward and the entropy term. However, in the new objective formulation, where the entropy term is replaced with the KL divergence term (explained in the previous section), α is now responsible for adjusting the trade-off between the reward and the similarity of user-system actions. This parameter essentially influences the policy's strategy in finding a balance between maximizing rewards and adhering to the user's intentions, effectively controlling the *autonomy level* of the user.

To automatically adjust the parameter α and avoid the need for manual tuning, Haarnoja et al. [131] proposed an algorithm that treats α as a learnable parameter. This is achieved by formulating policy learning as a constrained optimization problem:

$$\max_{\pi_{0:T}} \mathbb{E}_{\rho_\pi} \left[\sum_{t=0}^{T} r(s_t, a_t) \right] s.t. \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[-\log \pi_t(a_t|s_t) \right] \geq \bar{\mathcal{H}} \ \ \forall t \tag{4.5}$$

where $\bar{\mathcal{H}}$ is a desired minimum expected entropy. This optimization problem is in turn formulated as the dual problem, by introducing the temperature parameter α. The optimal policy at time t is now a function of the dual variable α_t, and after solving for $Q_t^\star$ and $\pi_t^\star$ by recursively optimizing Eq. 4.5, the optimal dual variable $a_t^\star$ is given by:

$$a_t^\star = \arg\min_{a_t} \mathbb{E}_{a_t\sim\pi_t^\star} \left[-\alpha_t \log \pi_t^\star(a_t|s_t;\alpha_t) - \alpha_t \bar{\mathcal{H}} \right] \tag{4.6}$$

In a similar fashion, an update mechanism of α for the shared control objective (Eq. 4.4) can be derived by formulating the following constrained optimization problem:

$$\max_{\pi_{0:T}} \mathbb{E}_{p_\pi} \left[\sum_{t=0}^{T} r(s_t, a_t) \right] s.t. D_{KL}[p(a_t^h|\cdot) \parallel \pi(a_t|s_t)] \leq \delta \ \ \forall t \tag{4.7}$$

where δ is a target divergence between user action prior and policy, similar to the target entropy $\bar{\mathcal{H}}$ in the original SAC formulation. By introducing the temperature parameter α, and following the proof in [136], the dual problem can be formulated as:

$$\min_{\alpha>0} \max_{\pi} \sum_{t=0}^{T} r(s_t, a_t) + \alpha \left(\delta - D_{KL}[p(a_t^h|\cdot) \parallel \pi(a_t|s_t)] \right) \tag{4.8}$$

Which leads to the **modified update objective** for α:

$$\arg\min_{a>0} \mathbb{E}_{a_t \sim \pi} \left[\alpha\delta - \alpha D_{KL}[p(a_t^h|\cdot) \parallel \pi(a_t|s_t)] \right] . \tag{4.9}$$

Now, δ becomes a hyperparameter to the shared control system, that dictates how closely the assistive system should follow the user commands on average. The temperature parameter α is in turn automatically adjusted during training, according to Eq. 4.9, to force the policy to respect that constraint.

4.3.3 Incorporating State Risk Estimation

In the previous sections, we discussed the formulation of the shared control problem by modifying the maximum reinforcement learning framework and the use of the hyperparameter δ to adjust the user's autonomy level. However, δ only controls the average similarity between the system and the user, which may lead to suboptimal behaviors. For instance, the system might prioritize the reward over obeying the user's intentions and end up ignoring the user's input. Additionally, even if the user drives in a risk-free way, the system may still deviate from their input by the amount dictated by δ.

To address these issues, we introduce an additional mechanism to guide the system in identifying states where assistance is required or not. This mechanism involves estimating the risk associated with each state and incorporating it into the observation space. The risk is represented by a scalar value in the range of $[0, 1]$, where 0 indicates no risk, and 1 indicates maximum risk. This risk estimation considers the proximity to obstacles and the user's intended velocities. Specifically, the risk increases exponentially based on the proximity to obstacles in the direction of the user's input, and higher intended velocities near obstacles are considered riskier.

The risk estimation is made accessible to the agent by including it as part of the observation space. Moreover, it also affects the contribution of the KL divergence between the user and the system calculated at each timestep as part of the shared control's objective (Eq. 4.4).

The risk associated with a state is represented by a vector of two elements, corresponding to the risks of executing the intended linear and angular velocities. Let $\boldsymbol{r} = [r_v, r_\omega]$ denote this risk vector.

The magnitude of the user's input influences the risk calculation, with higher intended velocities near obstacles being considered riskier than lower ones. For a given user intention $\boldsymbol{u}_t^h = [v_t, \omega_t]^T$ at timestep t, the risk in the linear direction is calculated using the formula:

$$r_v(t) = e^{-\frac{minFrontRange - minSafetyRange}{v_t}}, \tag{4.10}$$

Similarly, the risk in the angular direction is calculated as:

$$r_\omega(t) = e^{-\frac{minSideRange - minSafetyRange}{\omega_t}}, \tag{4.11}$$

Here, $minFrontRange$ represents the minimum distance reading from the robot's forward-facing sensors, $minSideRange$ is the minimum range from the side sensors in the direction indicated by the user input, and $minSafetyRange$ is the minimum distance below which a collision is considered imminent and at which we want the risk to take its maximum value.

Once the risk values are calculated, they are used to discount the KL divergence between the user's and the system's action probabilities, thereby influencing the shared control objective (Eq. 4.4). The discounting is represented as follows:

$$D'_{KL}(p||\pi) = (1 - \boldsymbol{r}) * D_{KL}(p||\pi) \tag{4.12}$$

Essentially, the larger the risk value, the more the KL divergence will be discounted, leading the policy to place greater emphasis on the reward term of the loss function. This prioritizes the agent's objectives, such as collision avoidance. Conversely, lower risk values lead to a higher contribution of the KL divergence term, resulting in stronger adherence to the user's intentions.

By incorporating state risk estimation and adjusting the KL divergence based on the risk values, the shared control system gains the ability to adapt its behavior, striking a balance between the user's input and the agent's objectives, while considering the potential risks in the environment. This approach ensures safer and more efficient shared control between the human and the autonomous agent.

4.3.4 Method Overview

The proposed method is summarized below both in a graphical (Fig. 4.1) and an algorithmic form (Alg. 7).

Algorithm 7 Modified Soft Actor Critic for Shared Control

```
1: Inputs Trained autonomous policy $p_\psi(a^u|\cdot)$, Reward function $r(s_t, a_t)$, discount $\gamma$, target update rate $\tau$, target divergence $\delta$, learning rates $\lambda_\pi, \lambda_Q, \lambda_\alpha$ etc.
2: Initialize replay buffer $D$, policy $\pi_\theta(a_t|s_t)$, critic $Q_\phi(s_t, a_t)$, target network $Q_{\bar{\phi}}(s_t, a_t)$
3: for each step in number of training steps do
4:     Observe state $s$ and $s_a$ and select autonomous agent action $a^u \sim p_\psi(\cdot|s_a)$
5:     Inject noise to the selected action to emulate a disabled user's input $a^h$
6:     Model the noisy input as a diagonal Gaussian probability distribution $p(a^h|\cdot) = \mathcal{N}(a^h, \Sigma)$, where $\Sigma$ is either treated as a predefined hyperparameter or is dynamically adjusted with user actions
7:     Estimate the risk factor $r$ of the state, where $r \in [0, 1]$
8:     Augment the shared control state vector $s$ with user input $a'$ and risk factor $r$
9:     Select action $a \sim \pi_\theta(\cdot|s)$
10:    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
11:    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
12:    If $s'$ is terminal, reset environment state.
13:    if it's time to update then
14:        for each gradient step do
15:            Randomly sample a batch of transitions, $B = (s, a, r, s', d)$ from $\mathcal{D}$
16:            Compute targets for the Q functions:
               $$y = r(s,a)+\gamma(1-d)\left[\min_{i=1,2} Q_{\phi_{targ,i}}(s', \tilde{a}') - \alpha D_{KL}\left(p(a^h|\cdot), \pi_\theta(\tilde{a}'|s')\right)\right], \tilde{a}' \sim \pi_\theta(\cdot, s')$$
17:            Update policy weights:
               $$\theta \longleftarrow \theta - \lambda_\pi \nabla_\theta \left[\min_{i=1,2} Q_{\phi_i}(s, a) - \alpha D_{KL}\left(p(a^h|\cdot), \pi_\theta(a|s)\right)\right]$$
18:            Update critic weights:
               $$\phi \longleftarrow \phi - \lambda_Q \nabla_\phi \left[\frac{1}{2}\left(Q_\phi(s, a) - y\right)^2\right]$$
19:            Update target alpha:
               $$\alpha \longleftarrow \alpha - \lambda_\alpha \nabla_\alpha \left[\alpha\left(D_{KL}(p(a^h|\cdot), \pi_\theta(a|s)) - \delta\right)\right]$$
20:            Update target network:
               $$\bar{\phi} \longleftarrow \tau\phi + (1-\tau)\bar{\phi}$$
21:        end for
22:    end if
23: end for
    return trained policy $\pi_\theta(a_t|s_t)$
```

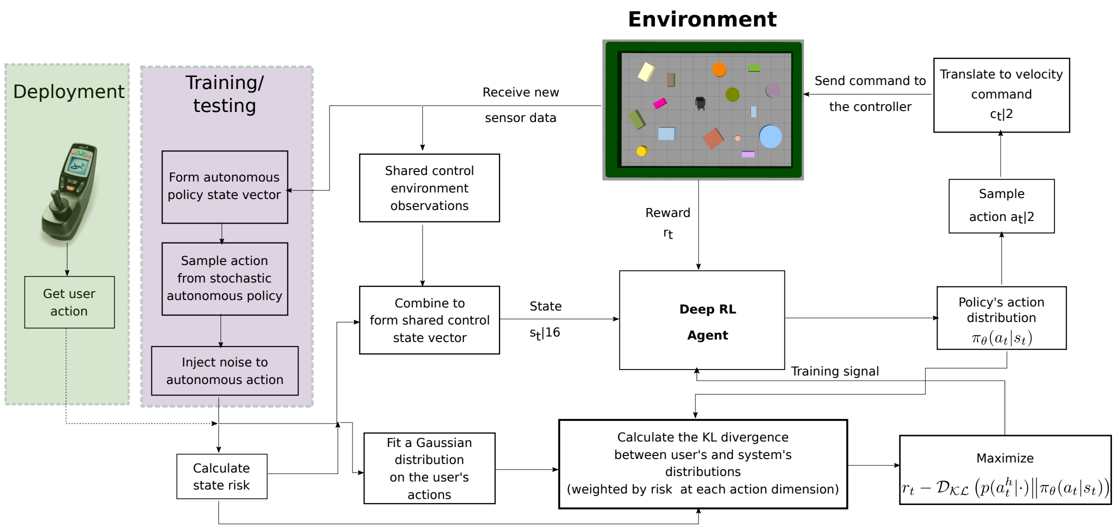

Environment
Deployment
Training/ testing
Get user action
Form autonomous policy state vector
Sample action from stochastic autonomous policy
Inject noise to autonomous action
Receive new sensor data
Shared control environment observations
Combine to form shared control state vector
State s_t|16
Calculate state risk
Fit a Gaussian distribution on the user's actions
Calculate the KL divergence between user's and system's distributions (weighted by risk at each action dimension)
Reward r_t
Deep RL Agent
Training signal
Send command to the controller
Translate to velocity command c_t|2
Sample action a_t|2
Policy's action distribution $\pi_\theta(a_t|s_t)$
Maximize $r_t - \mathcal{D}_{\mathcal{KL}}\left(p(a_t^h|\cdot)\middle\|\pi_\theta(a_t|s_t)\right)$

Figure 4.1: Overview of User-Compliant Reinforcement Learning. The shared control system is trained with autonomous input perturbed by artificial noise for robustness. During deployment, direct user input is used, enabling a seamless transition to real-world scenarios. The policy maximizes rewards while minimizing KL divergence between user and system actions, discounted by estimated state risk. The system's adaptability is enhanced through the incorporation of the hyperparameter δ, which allows fine-tuning of the user's autonomy level. The proposed approach strikes a balance between user intent and agent efficiency, promoting a safe and reliable policy-based shared-control approach.

4.4 MDP Formulation

Similar to the methodology followed in Sec. 3.4.1, the shared control problem is modeled as an episodic MDP, with finite time horizon T. S is a set of continuous states, A a set of continuous actions, γ a discount factor and R the reward function. At each discrete timestep t the agent chooses an action $a_t \in A$ according to a policy $\pi(s_t)$, observes a new state s_{t+1} and received a reward $r(s_t, a_t) \in R$. The goal is to find an optimal policy $\pi^\star$ that achieves the maximum expected return $G_t = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$ from all states, while respecting the user's intention as much as possible.

State Space

The shared control system utilizes a similar sensor arrangement with the autonomous navigation system of Ch. 3. The sonar sensor measurements remain the same, while the velocity of the robot derived from the odometry is also included as part of the state. The IMU, however, is excluded from the assistive system, since it is meant to be transferred to the real world and an inaccurate IMU sensor might cause more harm than good. Furthermore, the robot's heading is not as important in this case, since in the assistive mode with minimal assumptions it is not necessary for the robot to orient itself towards a goal. For the same reason, goal-related observations are also excluded from the state. Instead, the user's intention at each timestep (which during training will be the noisy autonomous agent's actions) is included as part of the state. Finally, the state space is augmented with an additional observation for each user command, that corresponds to the risk associated with that particular command at the given state. Details on how the risk is defined and calculated are provided in Sec. 4.3.3.

Overall, the state at timestep t consists of 17 observations, which are 11 sonar range measurements ($\vec{d_t}$), the instantaneous linear and angular velocity of the robot ($\vec{v_t}$), the user's intention ($\vec{u_t^h}$) and the associated risk factors ($\vec{r_t}$).

Actions

The actions available to the assistive agent are the same ones available to the user (and the autonomous agent, Sec. 3.4.1). The shared control policy should be able to directly output the moving commands for the robot, since the goal is to build an end-to-end assistive model.

Rewards

The reward signal for the task of assisting a human driver is what will determine the type of assistance that will be provided by the system. Since the objective of following the user's intentions has been decoupled from the reward function, the latter defines the task the agent should accomplish if it was acting independently.

Even though assistance can take many forms (*e.g.*, smoother trajectories, goal-following behavior, "parking" assistance, etc.) which could be encoded in the reward function, in this work we focus on the simplest, yet the most essential one; avoiding collisions. For that purpose, the reward function r_t, at timestep t, is defined as:

$$r_t(s_t, a_t) = t_r + l_p r_p(s_t, a_t) + l_v r_v(s_t, a_t), \tag{4.13}$$

where l_p and l_v are scaling factors. t_r is a constant, positive reward given at each timestep for incentivizing the survival of the agent. r_p is a penalty term for discouraging proximity to obstacles, hence encouraging increased clearance, and is defined as:

$$r_p(s_t, a_t) = e^{-k(minSensorRange-minSafetyRange)}, \tag{4.14}$$

where k is a coefficient regulating the shape of the exponential. Finally, r_v is a term rewarding the translational velocity of the robot, in a linear fashion, for encouraging forward motion and helping users reach their goals faster. The agent is also penalized with a large negative reward if the wheelchair crashes or overturns.

Terminal States

The conditions that terminate an episode are the same as in the autonomous agent 3.4.1. Namely, those are: crashing, overturning, getting stuck, and reaching the maximum number of allowed steps per episode.

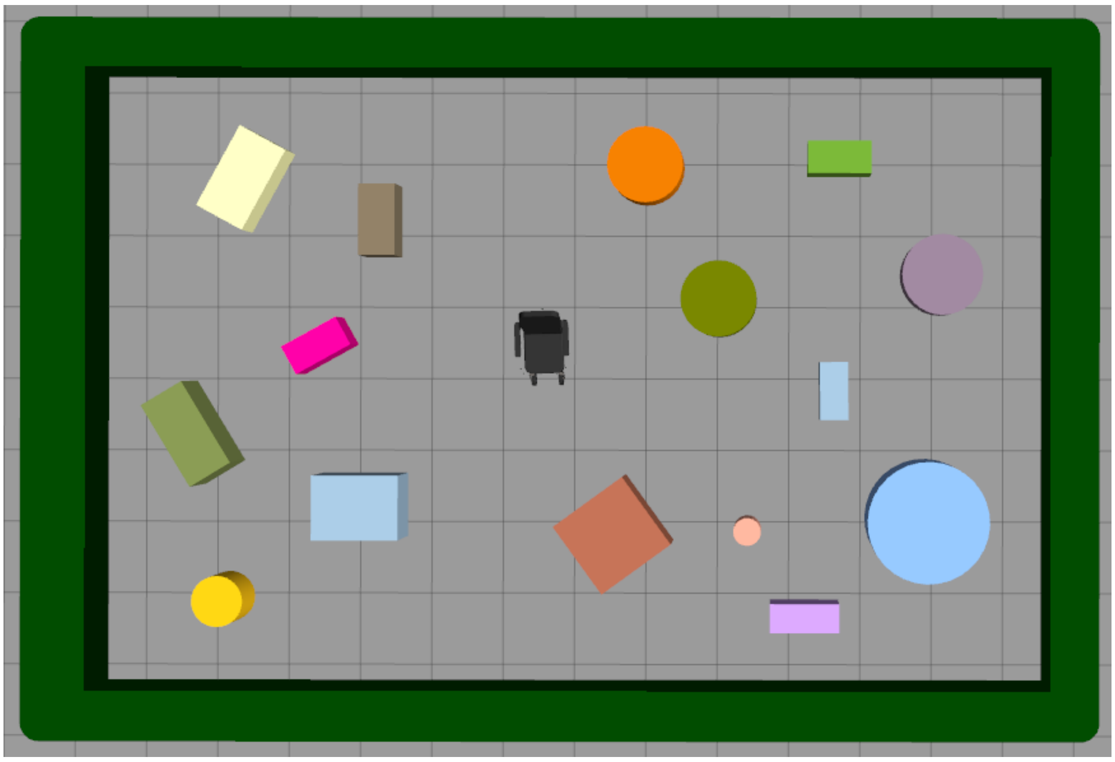


Figure 4.2: Training Scenario for Shared Control. The virtual environment includes periodically re-spawned obstacles in random positions and shapes, along with randomly generated goal positions within the scenario boundaries. This deliberate variation creates a diverse range of scenarios, promoting better generalization of the shared control agent during training.

4.5 Simulation Experiments

4.5.1 Training

Setup

The training of the system takes place in a virtual environment (shown in Fig. 4.2), featuring procedurally generated obstacles of various shapes placed randomly throughout the map, subject to distance and boundary constraints. At the beginning of each episode, the robotic wheelchair is also randomly positioned within the environment. The baseline autonomous agent is then assigned a random goal location within the free space on the map and generates movement commands to reach it.

To simulate real-world scenarios, one of the noise models (see Sec. 4.2.1) is applied to corrupt the input received by the baseline agent. This corrupted input, along with other environmental observations, is then used as the observation input for the shared control system. During training, the system learns to maximize future rewards while simultaneously minimizing the KL divergence between its output and the noisy input, up to a specified target.

The incorporation of randomness in the training task is intentional, as it enhances the generalization capability of the policy. By exposing the system to a wide variety of different configurations the policy becomes more adaptable and capable of handling diverse real-world situations.

Training Details

The modified SAC algorithm for shared control (Alg. 7) is utilized to train multiple assistive policies. The actor-critic architecture used in the training process is illustrated in Fig. 4.4, featuring a shared featured network utilized by both the actor and the critic.

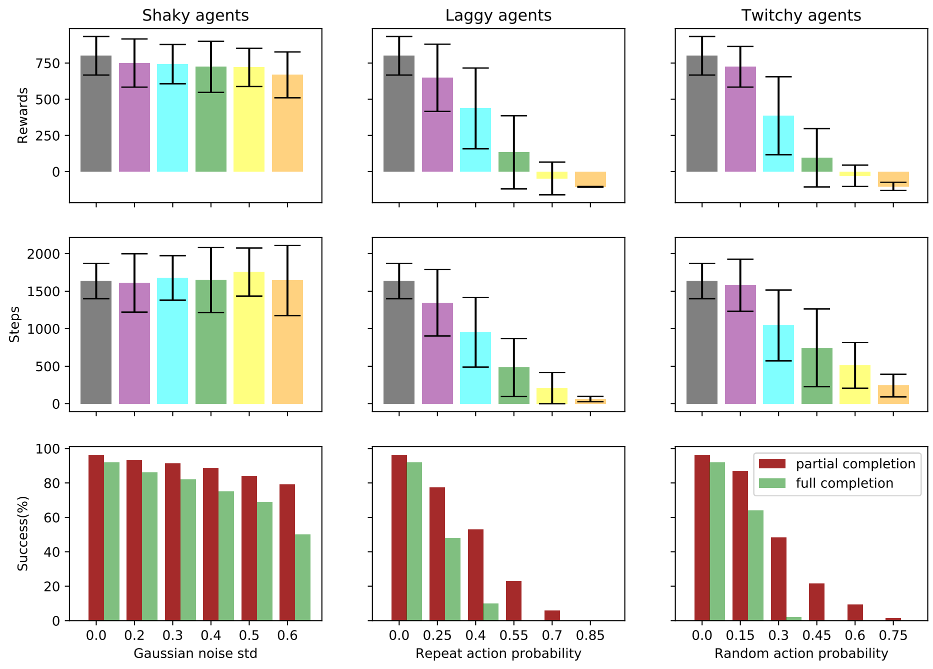


Figure 4.3: Evaluation of the Autonomous Agent Under Different Noise Models. Each column represents a distinct noise model, while each row corresponds to a different performance metric: rewards, episode steps, and success rates. Each subfigure illustrates the performance of the agent with varying levels of disturbance specific to that noise model. For example, in the shaky agent subfigure, different levels of Gaussian noise are depicted. The results provide insights into the agent's performance under diverse noise conditions, facilitating the evaluation of potential user models and their impact on the optimal agent's behavior.

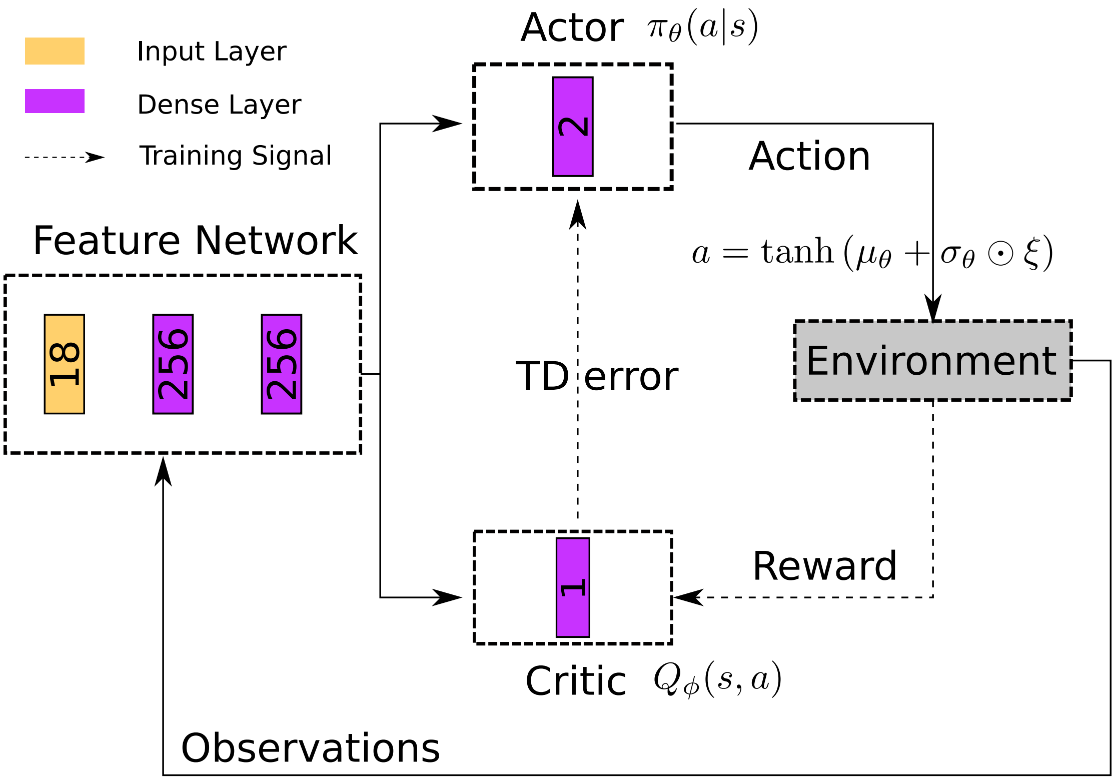


Figure 4.4: Architecture of the Actor-Critic Model for Shared Control. The feature network is shared between the actor and the critic. The actor network generates the actions, while the critic network outputs the Q state-action value. The actor's actions are applied to the environment, while the environment provides rewards to the critic network used to update it. The observations received from the environment are used as input to the shared feature network.

Throughout training, each policy's performance is periodically evaluated in a separate test environment over five episodes, and the best-performing policy is selected as the optimal one for deployment. The actor-critic architecture and SAC-specific hyperparameters are kept consistent across different models, while the parameters of the additional components in the modified version (proposed in the chapter) are varied to study their impact on the assistive policy. The robot's controller parameters and control timestep are tuned according to the physical robot and are further detailed in Sec. 4.6.2, which covers the real-world experiments.

For choosing the noise parameters of the erroneous user models, we evaluate the performance of each user model on the scenario of Fig. 3.8a, where all marked goal positions are combined in a single scenario for the evaluation task. The autonomous agent model deployed for this comparison, and for acting as the user model in the training loop of the assistive agent, is the optimal one that was presented in Ch. 3. From 100 runs for each user model, statistics are collected and presented in Fig. 4.3. We observe that the autonomous agent is quite robust to Gaussian noise since the shaky agent's success rates only decrease slightly when the noise distribution's standard deviation is increased. Thus, the maximum tested standard deviation of 0.6 is selected for the shaky user model. For the laggy and twitchy models, we observe an exponential decrease in performance when increasing the probability of repeating the previous action and the probability of selecting a random action, respectively. To ensure the assistive agent learns to provide meaningful assistance, we select values that lead to suboptimal performance of the autonomous navigation agent but still enable it to roughly navigate toward the goal positions. Therefore, based on the obtained statistics, for the laggy user model, we choose an action repeat probability of 0.55, and for the twitchy user model, we select a random action sampling probability of 0.45.

Table 4.1: Success rates of Training and Testing the Shared Control Method with Different User Models. A different "target divergence" (td) parameter value is used for each model trained with a different user.

	Evaluation user			
Training user (td)	Optimal	Shaky	Laggy	Twitchy
None (-)	0.93	0.81	0.23	0.21
Shaky (0.3)	0.94	0.86	0.51	0.6
Laggy (0.5)	0.96	0.84	0.81	0.55
Twitchy (1.2)	0.59	0.48	0.41	0.56

4.5.2 Evaluation

Testing Shared-Control Performance and Adaptability to Diverse Users

The evaluation begins with an analysis of the proposed method's performance under different simulated users. The central hypothesis is that the shared control method is able to improve a user's performance, despite not knowing anything about the world's dynamics, the user's policy, or the existence of a particular set of goals. The simulated users allow the validation and testing of different aspects of the shared control method before deploying the system in the real world.

Initially, three assistive models are trained in the scenario of Fig. 4.2, each with a different user model producing the desired driving commands for the robotic wheelchair. Their performance is then tested on the unseen-during-training scenario of Fig. 3.8a for all the marked goals (1-11) and the associated WST tasks. We are mainly interested in investigating two things 1) How well can the assistive policies perform when employed in a challenging environment for which it knows nothing about (obstacles, goals, etc.) 2) How well can the assistive policies adapt to users with different behavior policies (distinct types of errors), and especially to the ones it didn't experience during training time.

Table 4.1 presents the success rates (ratio of reached goals to total goals) of each assistive model when tested with each user model in a span of 100 episodes. The first row of the table corresponds to the performance of the pre-trained autonomous agent, without any assistance, and under the effect of different noise models. We observe that the assistive models trained with the shaky and the laggy user improve the performance of the autonomous agent, when the latter performs either optimally or under a noise model. For example, the assistive agent trained with the laggy user improves the optimal autonomous agent's success rate by 3%, the shaky agent by 3%, the laggy one by 58% and the twitchy one by 34%. The only case where assistance does not help all the users is with the model trained with the twitchy agent, which performs worse for both the optimal autonomous agent (-34%) and the shaky one (-33%), whereas it improves the laggy (18%) and twitchy (35%) models. Despite this assistive model's poor performance in terms of goal completion rates, it still develops a collision avoidance behaviour (as do the other assistive models), driven by the reward function, as the crash rates reduce dramatically compared to using no assistance (-2%, -14%, -34% and -86% for each user model respectively).

As the assistive agent is not provided with any goal-related information or reward, explicitly reaching goals is not part of its objectives. Rather, the agent follows the user input to the degree indicated by the "target divergence" parameter, while trying to maximize the cumulative rewards it receives. Still, the results indicate that the shared control system does help the users reach their goals more often, when the system is trained with a user model that provides informative input, even when the system is tested on an unseen, complex scenario. For example, the shaky user model has some perturbation around the optimal action, but on average its input will still point to the desired goal location. Similarly, the laggy user will delay changing its intended input, but when it does it will be aimed towards a goal. In both of those cases which implement goal-signaling policies, the shared control system was able to leverage the erroneous user input, along with the observations it received from the environment, to successfully follow the user goals. However, in the case of the twitchy user model, which chooses a completely random action almost half the time (45% probability), the input was not informative enough for the system to be able to reach the underlying goals with the same success rates.

Another takeaway is that each assistive model performs best when evaluated with

Table 4.2: Evaluation of the Proposed Shared Control System on Different Users, measured by the scenario Success Rate (SR), the robot Crash Rate (CR) and the Episode Step Ratio (ESR).

	No assistance			**Original SAC**			**User-Compliant SAC**		
User	SR	CR	ESR	SR	CR	ESR	SR	CR	ESR
Optimal	0.93	0.2 ± 0.4	0.77 ± 0.17	0.54	0.12 ± 0.32	0.88 ± 0.32	0.96	0.1 ± 0.3	0.7 ± 0.2
Shaky	0.81	0.5 ± 0.5	0.82 ± 0.23	0.33	0.08 ± 0.27	0.92 ± 0.27	0.84	0.24 ± 0.33	0.76 ± 0.25
Laggy	0.23	1.0 ± 0	0.24 ± 0.19	0.25	0.0 ± 0.0	1.0 ± 0.0	0.81	0.36 ± 0.48	0.69 ± 0.21
Twitchy	0.21	1.0 ± 0	0.37 ± 0.26	0.16	0.17 ± 0.37	0.84 ± 0.37	0.55	0.48 ± 0.5	0.83 ± 0.17

the user model it was trained with. While this result is expected, we observe that each policy can still be effective and learn useful assistive behaviors for user models they did not experience during training. For example, the assistive policy trained with the laggy user is almost as effective in assisting the shaky user as the assistive policy trained with the shaky user in the loop. Overall, the policy trained with the laggy user seems to be the best performing one among the other policies and the different user models they were evaluated with, and is chosen as the optimal one for the rest of the comparisons in this section.

It should be noted that the results presented in Table 4.1 occurred with the shown values of the target divergence (TD) parameter for each user model, which were set empirically, and not through an exhaustive search for the optimal values. Generally, decreasing the TD parameter increases the policy's compliance with the user input, while increasing it allows the policy to relax its compliance and focus more on optimizing its own objectives.

Comparison with Maximum Entropy RL

The justification for modifying the maximum entropy RL objective -to minimize the divergence between the system and the user, instead of maximizing the entropy of the policy- has been thoroughly explained in Sec. 4.3. In this section, this justification is experimentally verified by training an assistive policy using the original maximum entropy RL objective, through the conventional SAC algorithm.

In this case, incentivizing the system to keep its decisions close to the user's intention is achieved by inserting an additional term in the reward function (Eq. 4.13), which rewards the agent based on a distance metric between its actions and the user's. This term, which we call "agreeability" or "action-similarity" function is defined as:

$$\psi(a_t^h, a_t) = e^{-k(\|\boldsymbol{a_t^h} - \boldsymbol{a_t}\|)}, \tag{4.15}$$

where $\|\boldsymbol{a_t^h} - \boldsymbol{a_t}\|$ is the l^2-norm of the action vectors' (human's and system's) difference and k is a free parameter controlling the strength of the attraction between the two vectors. The smaller the difference between the two vectors the larger the reward the agent receives. The task and robot related parameters for training the new model remain the same, as well as the common SAC parameters and the neural network architectures. The user model used to provide input for the training process is the laggy user, for consistent comparison with the best model of the proposed shared control system.

Table 4.2 presents the comparative results of different user models using no assistance, using the assistive model trained with the original SAC algorithm, and using the proposed user compliant SAC algorithm, evaluated on 100 episodes. The results are presented in terms of average goal completion rates, crash rates and the fraction of steps before episode termination (either due to crashing or completing all goals) to the maximum number of allowed steps. We observe that using the proposed system significantly improves the statistics across all user models compared to using no assistance. The assistive agent is able to reach goals with higher success rates than the solo autonomous agent, while reducing the crash rates by a large margin. The number of steps before episode termination decreases for the cases of the optimal and the shaky user, due to the term in the reward function that incentivizes higher translational velocities, while it increases for the cases of the laggy and the twitchy user, which is accounted to the increased survival rates (64% and 52% less collisions on average respectively).

On the other hand, for the assistive model trained with the original SAC algorithm different results are observed. In this case, the goal reaching rates drop significantly, even when compared to the solo autonomous agent, across all user models. However, the crash rates decrease by a large margin, while the number of steps before episode termination approaches the maximum number of allowed steps, indicating a large "survival" rate. In practice, by following this approach (embedding the action similarity in the reward function) the learned policy prioritizes avoiding collisions over respecting the user's intentions, since a collision would result in a much larger negative reward than the positive reward that would be gained by following the user. Therefore, the trained agent avoids getting into states that are highly likely to result in a collision (*e.g.*, narrow corridors) and ignores the respective user prompts by moving away from the obstacles and entering a circle-following like behavior. When the scaling factors of the reward function were tuned to place higher importance to the action similarity than avoiding collisions, the opposite behavior was observed. The agent would blindly follow the erroneous user input leading to crashing into obstacles.

Of course, neither of those policies are appropriate for a shared control system, indicating that a scalar quantity (the reward) is not sufficient for this purpose, since it can be easily exploited by an RL agent and lead to undesired behaviors. These results highlight the importance of decoupling the user compliance objective with the rest of the shared control objectives.

Comparison with APF method

The proposed method is benchmarked against another shared control method, of the Artificial Potential Fields family (recall Sec. 2.2.3). The *Dynamic Localized Adjustable Force Field* (DLAFF) [112, 113] method employs the concept of an active window/frame containing a nonlinear adjustable force field which is elliptically shaped to provide a better mathematical relationship between the repulsive force and the kinematic of the platform. In this method, the inner ellipse provides a zone in which the physical boundary of the platform is fully contained and the outer ellipse provides the furthest extent of the repulsive field, where the repulsive force is determined by Eq. 4.16 along the vector $P - r$ (Fig. 4.5) to the near-

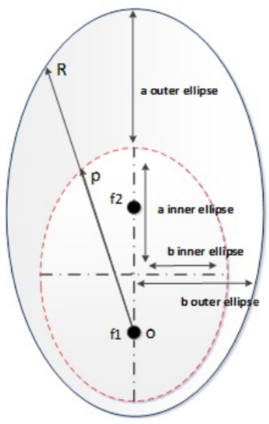


Figure 4.5: DLAFF Collision Avoidance Ellipses.

est obstacle in each quadrant, where that repulsive force acts to damp the motor drive outputs. The shape of the ellipses is dynamically adjusted according to the platform's velocity.

$$F = 1 - \frac{1}{\exp\left((R - p)/k\right)}, \tag{4.16}$$

where k is a user-defined coefficient that controls the shape of the exponential (*i.e.*, the steepness of the damping force relative to the obstacle's distance from the inner ellipse).

Like other non-learning-based motion planning algorithms, DLAFF requires accurate knowledge of the obstacles in the local environment (polar coordinates of each obstacle). However, the obstacle information is derived from the platform's sensor measurements. We explore two sensor arrangements, with the DLAFF method, to investigate the utilization of sensor information between a classic approach and a neural network approximation approach in the shared control context; 1) Using the same array of sonar sensors available to the assistive agent 2) Using a laser range finder sensor (a Hokuyo LiDAR).

For each sensor arrangement, the performance of the DLAFF method was recorded when operating on the output of the optimal, and the erroneous autonomous navigation policies. The results (Table 4.3) show that with both sensor setups, the method performs well with the optimal user, slightly improving its success rate. However, we observe that the performance drops significantly in both cases

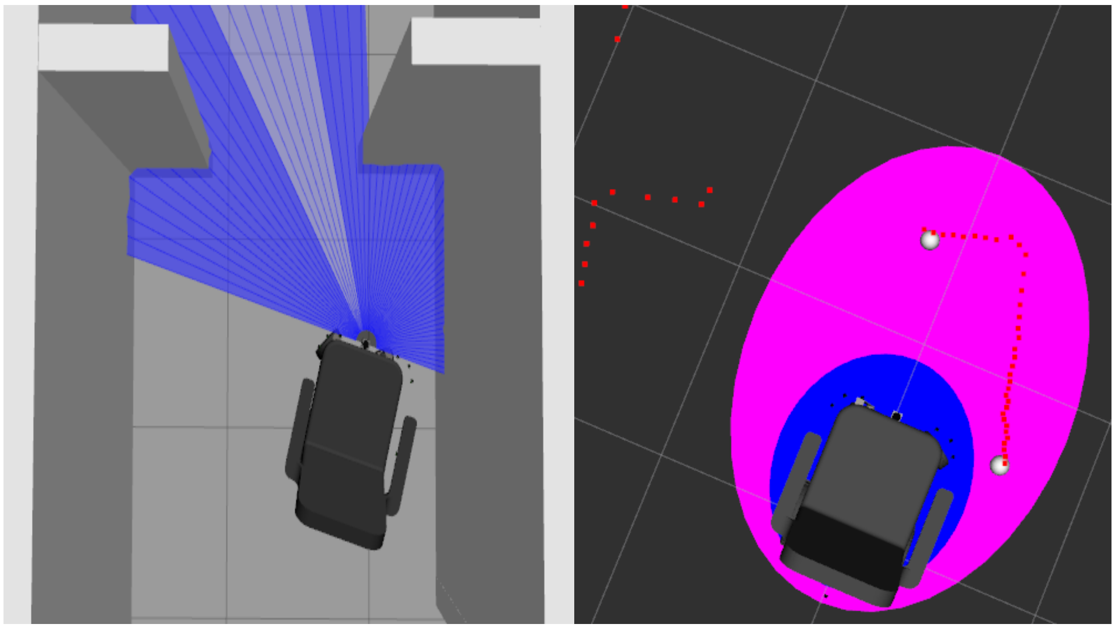


Figure 4.6: Display of the DLAFF Collision Avoidance Method in Simulation. The algorithm leverages range sensor readings (marked with red) to detect the two nearest obstacles at each quadrant (white spheres), and calculate a repulsive force that damps the motor outputs (Eq. 4.16), dependent on the obstacles' distance from the zone defining the platform's physical boundary (blue ellipse).

Table 4.3: Comparative Results with the DLAFF Shared Control Method. The proposed model outperforms DLAFF, which relies on specific obstacle angle knowledge. When using the same sonar setup, DLAFF underperforms, while replacing the sonars with a LiDAR fails to provide sufficient surrounding information for safe navigation, leading to worse performance. This demonstrates the advantage of a neural network-based policy, which can effectively learn collision avoidance strategies arbitrarily leveraging the available sensor information.

	No assistance		**Shared Control RL**		**APF-Sonar**		**APF-LiDAR**	
User	Success	Crashes	Success	Crashes	Success	Crashes	Success	Crashes
Optimal	0.93	0.2 ± 0.4	0.96	0.1 ± 0.3	0.95	0.12 ± 0.32	0.94	0.16 ± 0.37
Shaky	0.81	0.5 ± 0.5	0.84	0.24 ± 0.33	0.7	0.68 ± 0.47	0.63	0.68 ± 0.47
Laggy	0.23	1.0 ± 0	0.81	0.36 ± 0.48	0.2	1.0 ± 0	0.26	1.0 ± 0
Twitchy	0.21	1.0 ± 0	0.55	0.48 ± 0.5	0.31	0.92 ± 0.27	0.23	1.0 ± 0

when used with the noisy policies, while the RL shared-control method clearly outperforms it. In the case of the sonar sensors, which detect obstacles in a cone, the exact angle to the obstacles is not provided, but instead, the detected obstacles are always assumed to be in the center of each sensor's horizontal field of view. Given the sensors' 50° FoV, this necessary assumption allows a big error margin that negatively impacts the method's effectiveness in several cases.

Conversely, the LiDAR sensor is able to provide the exact angles to the detected obstacles. However, from a practical perspective, the positioning of the sensor is a critical consideration for it to fulfill its purpose. Fig. 4.6 shows a LiDAR installed on the virtual EPW, between, and slightly beyond, the footrests of the wheelchair, able to freely perform its 180° scan. However, it is clear that the sensor can only cover the forward-facing obstacles in respect to the platform, but does not provide any information on the sides of the bulky EPW. This leads the robot to crash in multiple cases since the algorithm does not have sufficient information to prevent collisions when the user sloppily maneuvers it around tight spots.

On the other hand, the neural network-based policy, which maps a given state (including the range sensor measurements) to control actions for the robot, in the absence of perfect information can still arbitrarily leverage the available information to learn an effective collision-avoidance strategy, while also optimizing for other objectives. Of course, the effectiveness of DLAFF, and all collision-avoidance methods, can be improved with more sophisticated sensor data processing (filtering, triangulation, etc.), while sensor fusion can be used to exploit the advantages of different sensors and improve coverage. However, it is important to note that these solutions come with increased computational and financial costs, which should be considered accordingly in a real-world implementation.

The Benefit of Risk Awareness

The previous sections were concerned with shared control policies that were trained with the modified SAC objective for user compliance, but without risk awareness (RA). Section 4.3.3 described the concept of estimating the risk associated with a

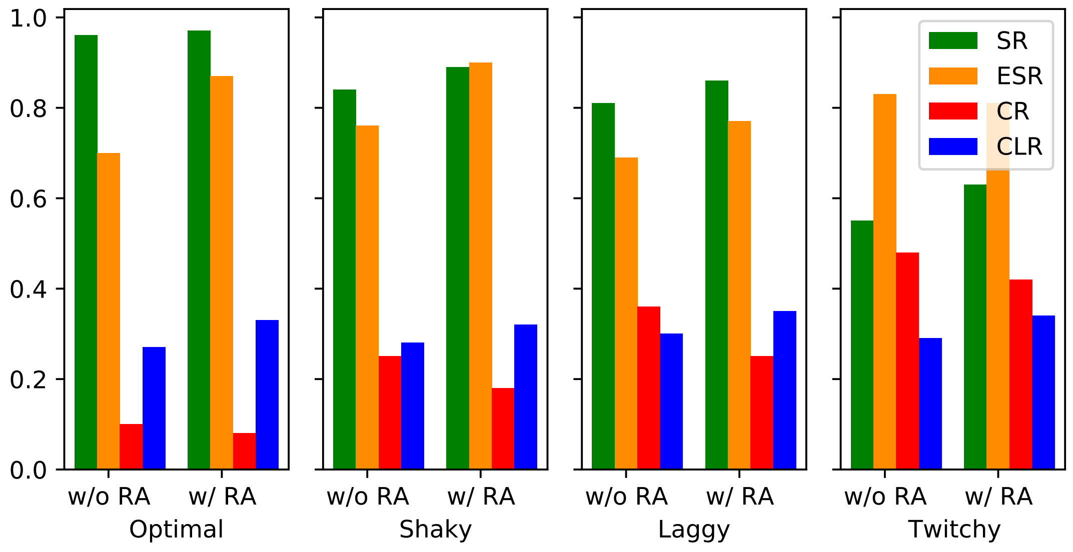


Figure 4.7: Comparative Results for the Shared Control Model With and Without Risk Awareness (RA) for Different User Models. The statistics are presented in terms of goal completion success rates (SR), episode step ratio (ESR), crash rates (CR), and clearance (CLR).

particular state, and how it can be incorporated in the proposed method to better guide the policy towards the situations where it should deviate, or not, from the user. In this section, we train a shared control policy (with the laggy user in the loop), as before, but also including the risk estimation step, which leads to the discount of the KL divergence as shown in Eq. 4.12.

The trained model is tested in the combined scenario of Fig. 3.8a, and its performance is benchmarked against the assistive agent trained with the laggy user presented in the previous sections. Figure 4.7 shows the comparative results between the two models, measured by the goal completion success rates, the episode step ratio, the crash rates, and the clearance. We observe that the assistive model with RA performs consistently better than the baseline model in terms of increased success rates and reduced crash rates across all user models. Furthermore, when using RA there is also an increase in clearance, since when the agent approaches an obstacle it is not forced to follow the user as closely (due to the KL divergence discount by the risk factor), rather it focuses on optimizing its own objectives, part of which is to keep away from obstacles. However, this behavior comes with a cost in larger completion times, due to the robot not following the optimal path towards a goal position (as is the intention of the autonomous agent), but deviating from it to keep further away from obstacles when possible.

Despite the increase in completion times, we recall that the agent is agnostic to the existence of goals or to what the user is trying to achieve. Furthermore, the other metrics that were considered showed a consistent improvement across different user models, satisfying the main shared control objective, which is safer driving. Also, the risk estimation function can be tuned to adjust the risk levels in different situations according to user preference, thus, providing a more flexible framework for customizing the shared control system to a wide range of use cases. For the reasons above, incorporating the concept of risk into the proposed shared control approach is deemed a beneficial addition that improves the method's performance and flexibility.

4.6 Real World Experiments

4.6.1 The Robotic Wheelchair

The wheelchair that was used for the real-world experiments is a Spectra XTR2 model (Figure 4.8), modified accordingly for compatibility with the Robot Operating System (ROS). Its basic processing unit is a UDOO QUAD/DUAL board, which is connected to the main sensors of the wheelchair, as well as the control unit through the joystick controller (a DX2-REM550/551 Advanced Joystick Remote model). A NVIDIA Jetson Xavier board is connected to the UDOO through Serial, to allow ROS message-passing (*i.e.*, sensor data, control commands), and to enable the execution of higher level tasks, like running the policy network.

Encoders

A pair of incremental rotary encoders was utilized (one encoder per driving wheel) for calculating the odometry data (defined as the use of motion sensors to determine the robot's change in position over time) and for enabling visualization of the trajectory executed by the wheelchair throughout the trials. The equations

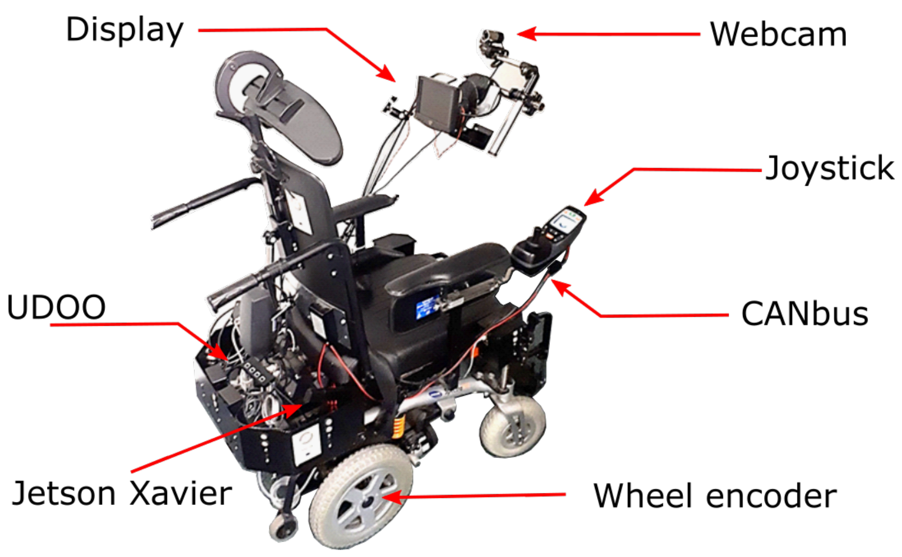


Figure 4.8: The Robotic Wheelchair Used in the Real-World Experiments.

below have been used to derive the velocity and the displacement of the EPW via the encoder information, based on the differential drive kinematics.

For a full revolution of the wheel, the encoder performs a certain number of “ticks” (t_{pr}). Given that information, along with the radius (r) of the drive wheels, we can then calculate the distance that corresponds to each of those ticks as

$$mms_{per.tick} = \frac{2\pi r}{t_{pr}}. \tag{4.17}$$

By measuring the difference in ticks between two consecutive measurements of an encoder (d_{ticks}), and from the kinematic equations that describe a differential drive platform (shown in Eq. 3.17), it is possible to calculate the robot’s displacement and change in heading from the following equations:

$$d_{xy} = \frac{mms_{per.tick}}{2} \cdot (d_{ticks.right} + d_{ticks.left}) \tag{4.18}$$

and

$$d_{th} = \frac{mms_{per.tick}}{l} \cdot (d_{ticks.right} - d_{ticks.left}). \tag{4.19}$$

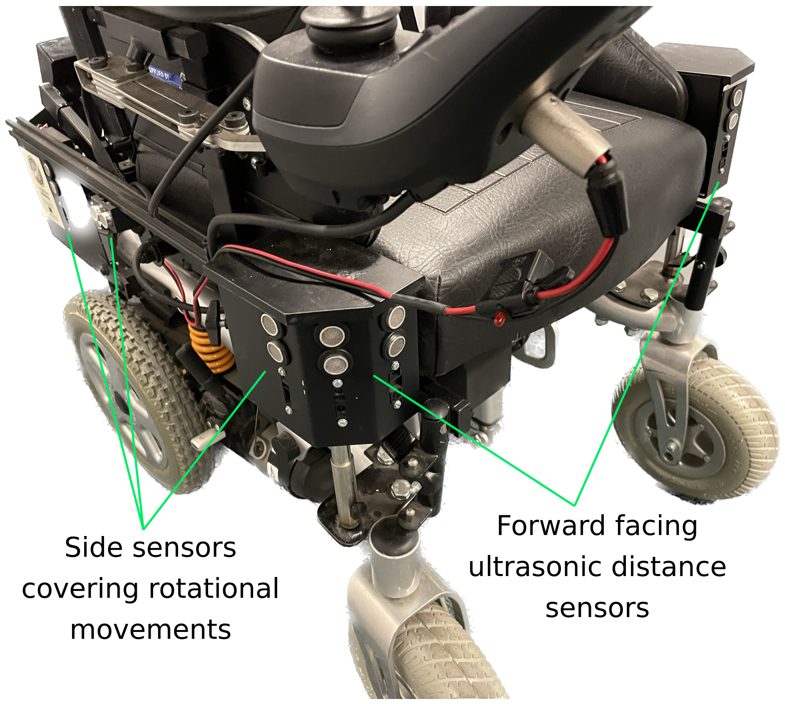

Side sensors covering rotational movements
Forward facing ultrasonic distance sensors

Figure 4.9: Display of the Robotic Wheelchair's Range Sensors.

Improved Ranging Sensors

A total of 11 sonar sensors were installed around the wheelchair to provide obstacle information. In contrast to 2D laser scanners that only detect obstacles at a specific height, sonars are great at detecting obstacles at different heights, which of course is very much desired from a practical implementation aspect. Furthermore, they are much cheaper compared to laser scanners, and therefore a number of them can be installed around the bulky and non-symmetrical shape of a wheelchair to provide wide coverage.

The simulation environment, in the early stages of experimentation with the autonomous agent, helped in gaining insights about how the sonars should be positioned in order to account for blind spots and ensure that the agent would have sufficient information to navigate through challenging spots (*e.g.*, steep turns in narrow corridors). An older arrangement of the sonars on the wheelchair was then modified, by changing the sensor positions accordingly. Furthermore, in the original setup, the sensors were grouped in zones, each zone using an Arduino Mini to collect the zone's sonar data, and then passing the data to the UDOO board through serial communication, on request. This implementation allowed a refresh rate of new data of approximately 5 Hz, which was not fast enough for controlling the wheelchair at 20 Hz. Instead, this setup was replaced by connecting all sensors to a single Arduino Nano through I2C, and then passing all the collected data to the UDOO board through Serial, thus, achieving a much higher sampling rate of 13 Hz.

The sonars that were used were the SRF08. The SRF08 allows customizing the ranging distance, as well as the analogue gain of the sensor. The ranging distance was reduced from a maximum of 6 m to 3 m (to match the simulated sonars) and the gain was tuned accordingly to allow firing the sensors more rapidly, further increasing the sampling rate. Reducing the sensor's gain also reduces cross-talk (interference) between adjacent sensors, since it reduces the module's sensitivity to detecting weaker echoes that result from sound waves bouncing off obstacle surfaces. However, cross-talk between adjacent sensors was still a big issue causing quite erroneous measurements. On that end, instead of firing the sonars sequentially, an alternative firing pattern was designed, so that two adjacent sensors would never fire together, but would instead fire in an alternating manner. This solution caused a drop in the refresh rate but almost eliminated the effect of cross-talk between the sensors.

Finally, the accuracy of the readings is affected by multiple factors (*e.g.*, obstacle's material and shape, wavelength of the emitted sound wave, the altitude at which the ranging takes place, etc.), which are hard to take into account. The SRF08's manufacturer quotes an error of 3-4 cm, which can be smaller or greater depending on the situation. In practice, it was observed that the error was very small ($\sim$2 cm) in a static setup where the wheelchair was not moving, but it greatly varied when the wheelchair moved, especially so during rotational movements. To compensate for those errors, median filtering was applied to a window of past measurements, which greatly smoothed the sonar readings in time and provided more consistent measurements. The size of the window controls the trade-off between stability in measurements and refresh rate, and after experimentation, a window size of 3-5 seemed to perform well enough.

Communication Protocols

The decision to employ the I2C and Serial protocols for inter-board and sensor communications, as opposed to the widely recognized "golden standard" of the industry, the Controller Area Network (CAN) protocol, was carefully deliberated, taking into account various factors such as simplicity, resource limitations, rapid prototyping, and specific application needs. By opting for the I2C protocol for sensor communication, the implementation process was streamlined owing to its inherent simplicity, requiring minimal components and wiring. This resulted in improved debugging efficiency and heightened reliability of the communication system, as it involved minimal overhead and harnessed a two-wire interface. Moreover, the I2C protocol facilitated seamless integration and communication with multiple sensors. Similarly, the adoption of the serial protocol for inter-Arduino communication was justified by its straightforwardness and compatibility with the Arduino platform, synergistically complementing the overall system design. Given the specific requirements of the sensor data transfer, characterized by low data rates and short distances, the employment of the I2C and serial protocols proved to be appropriate, negating the necessity for a more intricate universal bus like CAN.

4.6.2 From Simulation to the Real World

The advantages and motivation of using simulation for training reinforcement learning policies, like the ones presented in this thesis, have already been described in Sec. 3.3.2, including reasons of scalability, cost, and safety. However, transferring a learned policy from simulation to the real world is one of the biggest challenges reinforcement learning currently faces, with policies trained in simulation often suffering from severe degradation in performance when transitioning to the real world. This is caused due to the simulation's inability to exactly match and capture the real world in terms of physics, which can be summarized in two dimensions in the case of robotics. First is the sensing part, where the simulation cannot accurately model the noisy sensor data from the robot's sensors which can be affected by several real-world factors. The other aspect is the discrepancy in the effect of the actuation commands, the robustness of which depends on the quality of the simulator, the simplifying assumptions of the physics engine, and the unpredictability in robot dynamics.

To bridge the gaps between simulation and real-world, two lines of actions were taken; calibration of the simulated robot and *domain randomization*. Calibration refers to tuning the design characteristics of the robot in simulation, in a way that matches the real one as much as possible. More specifically, the design choices that were taken into account are:

1. *The physical properties of the robot* - The simulated wheelchair model was tuned to be as close to the real one as possible. That includes its various dimensions (*e.g.*, size of the chassis, size of the wheels, axle length between the drive wheels), its drive system (rear-wheel drive) and its inertial properties.
2. *The controller parameters* - The controller of the real wheelchair was parameterized in a custom way to create a driving profile suitable for the trials. Interfacing with the controller was possible thanks to a special module that was provided by the wheelchair's manufacturer. However, the resulting maximum allowed velocities and accelerations (linear and angular), are not directly accessible, but are a result of the aforementioned parameterization. The exact numbers were derived through repeated experimentation with the customized profile, measuring traveling distances and time, and verified through manual recordings and odometry calculations. The discovered parameters were then replicated with the simulated differential drive controller.
3. *The sensors* - The number of sensors, their placement on the robot, their field of view and their error of measurement, as given by the manufacturer, have been replicated in the simulated wheelchair.

The above steps are the minimum requirements for replicating the real robot in the simulation. However, a policy that learns a mapping, from environmental observations to actuation commands, for that robot under the ideal circumstances of

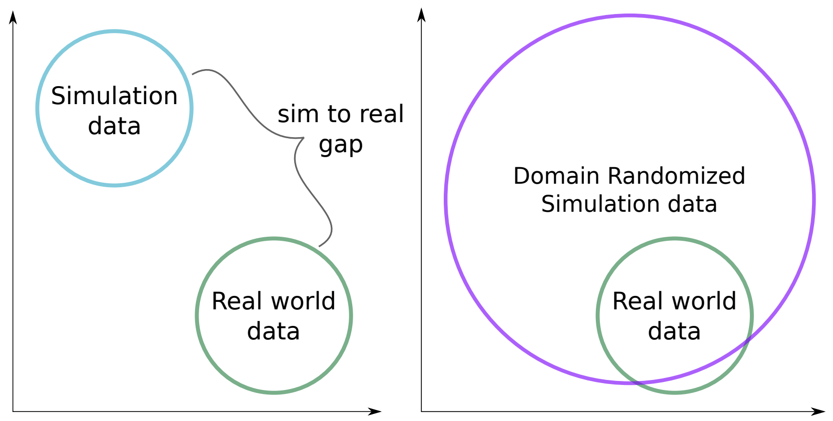


Figure 4.10: The Domain Randomization Concept. Randomizing the simulation data increases the variance of their distribution and forces the policy to adapt better to perturbations and inconsistencies in the input and output space, aiming to also capture the real-world distribution.

a simulator, will possibly struggle to handle the imperfect real world. A popular methodology that facilitates the desired "sim-to-real" transition is *domain randomization*[137]. Domain randomization refers to randomizing different aspects of the simulated environment, with the aim of training a model that works in a rich distribution across all variations and randomized properties, hoping that the real world is a sample of that distribution that the trained model will be able to capture. More specifically, the aspects that were randomized within the context of this work are:

1. *Raw sensor data* - Normal noise $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ was added to the measurements coming from the simulated sonar sensors. Since the manufacturer of the SRF08 sensors that were used quotes a measurement error of up to 0.03-0.04 m, the standard deviation of the noise was chosen as $\sigma = 0.015$, so that 99.73% of the measurement errors fall within $3\sigma = 0.045m$.

2. *Wheelchair's mass/inertia* - Once every a fixed number of episodes, the mass of the robot would be modified as sampled from a uniform distribution, and the resulting inertia of the wheelchair's chassis would be recalculated for that new mass (Eq. 3.16).

3. *Control timestep* - The timestep Δt between actions varies every step according to $\Delta t \sim \Delta t_0 + Exp(\lambda)$, where $\Delta t_0 = 0.05s$ is the default control

timestep (20 Hz), and $Exp(\lambda)$ is an exponential distribution with rate parameter $\lambda = 1/100$. This is to make the system more robust against any minor delays that might occur in the real system and the hardware's communication pipeline.

Other physical dynamics features, like damping and friction coefficients of the joints, friction between the wheels and the floor, or controller-related parameters, were not randomized, since the commercial wheelchair's controller should already be programmed to take these factors into account when converting the joystick high-level command to currents for the drive motors.

Based on the above, a new shared control policy was trained in simulation, based on the best-performing model of the experiments presented in the previous section, with a laggy user in the loop and with risk awareness. The trained model was then transferred to the "brains" of the wheelchair, the Jetson Xavier, with only minor modifications to the code for receiving the actual data from the wheelchair's sensors and the user's input from the joystick, and for passing the inferred commands in the appropriate format to the wheelchair's controller.

4.6.3 Experimental Protocol

Participants were asked to drive the physical wheelchair in the scenario shown in Fig. 4.11 using the standard joystick controller. The scenario was designed as a shortened real-world version of the virtual scenario of Fig. 3.8a and includes some of the WST tasks shown in Table 3.2 (*e.g.*, perform 90-degree turn in a narrow corridor, enter lift area and turn in place to exit). The task was to complete the circuit in a as fast and safe manner as possible in two separate trials; one would be with provided assistance and one without. The participants were unaware whether the assistive system would be in effect or not, and in order to counterbalance any order effects [79], odd-numbered participants undertook their first trial with assistance before moving on to the trials without assistance. Conversely, even-numbered participants undertook their first try without any assistance and their second one with the assistive system in effect.

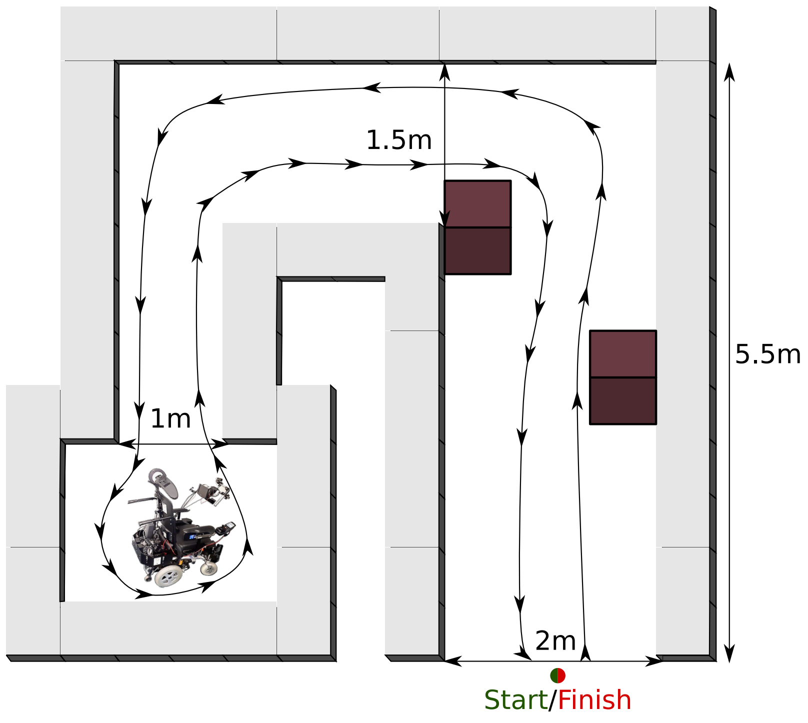


Figure 4.11: The Real-World Scenario for Testing the Shared Control System with Human Participants. The task involves driving around the circuit, starting and finishing at the same location. The scenario includes WST tasks like entering the lift area and performing a turn in place to exit.

Before the trials started, each participant was given up to 10 minutes to freely drive the wheelchair around the room and familiarize themselves with using the joystick controller, as well as the shared control system. In total, 13 able-bodied volunteers aged between 18 and 57 were recruited to participate in the experiments. Objective data including the produced control commands, the wheelchair's trajectory, and the distance from obstacles were automatically recorded during the trials. Subjective data were also gathered at the end of each trial, where the participants were asked to fill in a brief questionnaire about their experience with the control mode they had just used (without actually knowing which mode it was). The questionnaire consisted of the statements in Fig. 4.16, for which the participants were asked to indicate how strongly they agreed on a five-point Likert scale (1 = strongly disagreed, 5 = strongly agreed).

4.6.4 Results

Task Metrics

The evaluation metrics that were employed to assess the performance of each participant were: distance travelled, task completion time, clearance (average minimum distance to nearest obstacle) and number of collisions. Furthermore, the similarity between the user's commands and the assistive system's final control command was also calculated to measure the level of obedience of the latter to the user.

Ideally, a better performing control will result in shorter distance, less time, more clearance and less collisions. However, the most safety-critical measure to quantify the performance of the shared control system is the number of collisions. Driving a powered wheelchair through the standard joystick interface is a rather straightforward task, especially for a healthy person in body and mind, even without prior experience of using an EPW. Furthermore, the given task was of moderate difficulty and the wheelchair's speed profile was set towards the slow side as a health and safety measure.

As a result, all the participants were able to successfully complete the task, most of them fast and accurately. Fig. 4.12 shows that only two participants experienced collisions when they were given no assistance, which were caused due to the users executing sharp turns while trying to complete the task as fast as possible. Conversely, no collisions took place while the assistive mode was active. However, the increased safety came at a cost of slower speeds. The trained policy with risk awareness did not allow fast, sharp turns when in close proximity to obstacles, especially when the users were deflecting the joystick to the maximum (recall from Eq. 4.10 that the magnitude of the intention affects the risk), rather it either pushed the wheelchair away from the nearest obstacle or slowed it down. This resulted in an inherent cost of using the shared controller in terms of completion times for the specific course, which has also been observed in other works on shared control[79, 83, 138]. The participants took an average of 52.1 (SD 4.69)

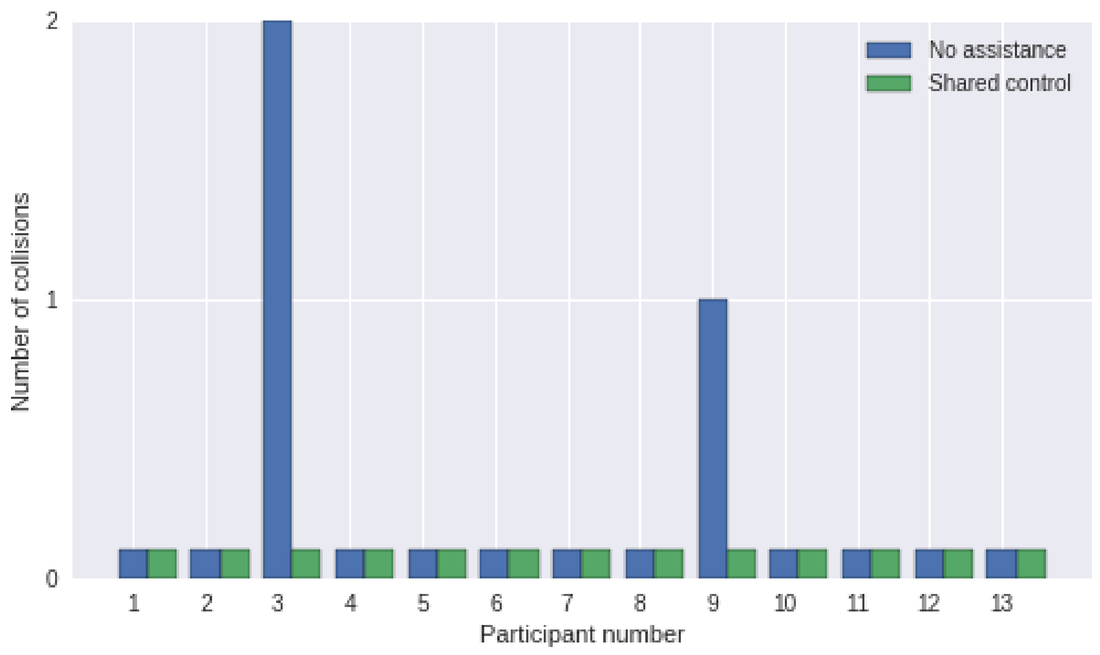


Figure 4.12: Number of collisions each participant experienced under each mode of operation, while using the standard joystick interface.

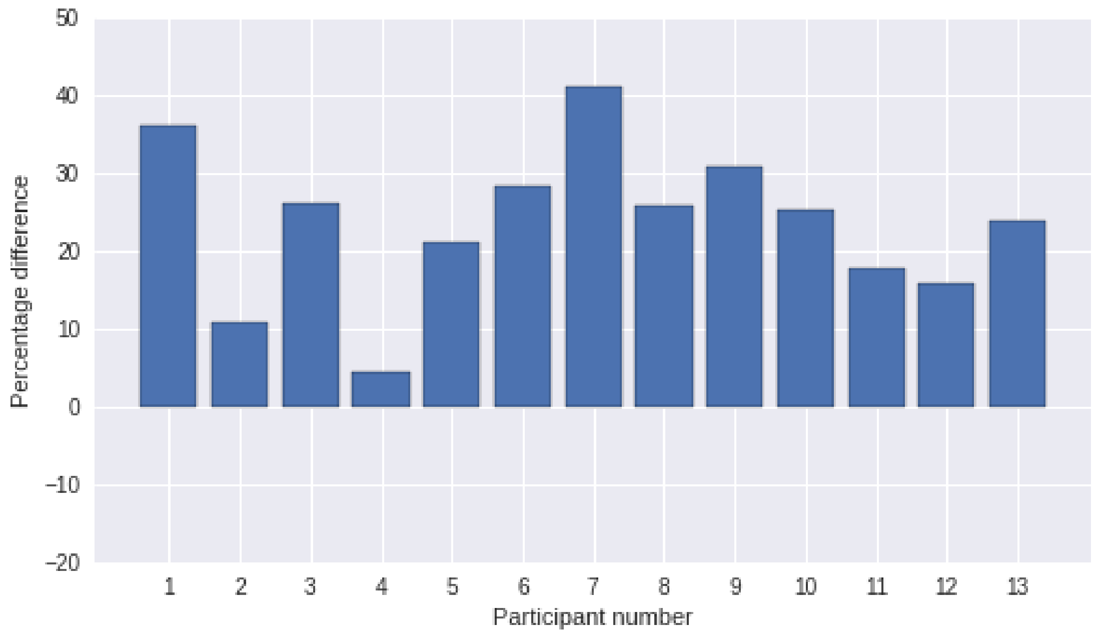


Figure 4.13: Difference in the participants' completion times when using the assistive system with the joystick interface. The system trades off speed for safety.

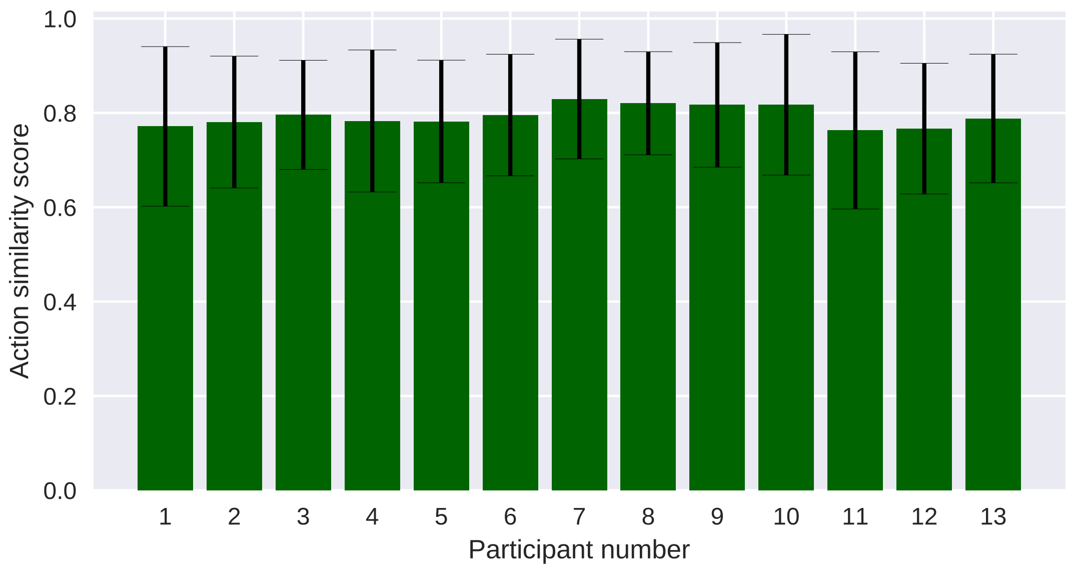
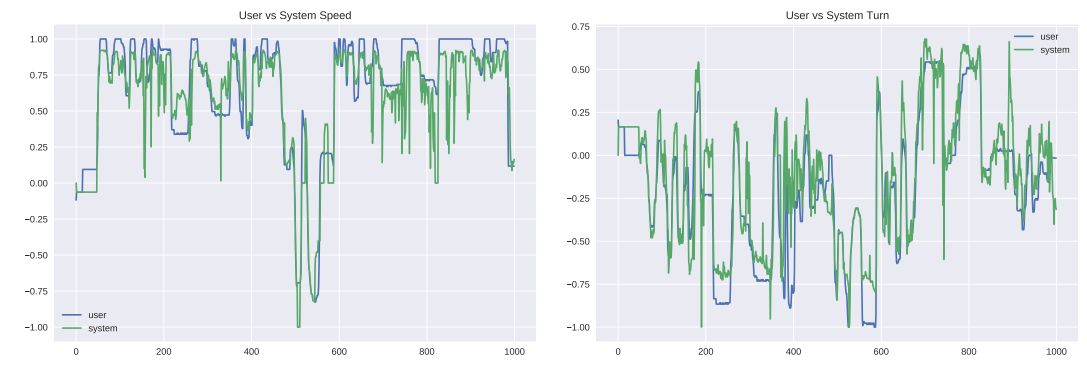


Figure 4.14: Averaged Action Similarity Score: Indicates the proximity of the assistive system's output to the user's desired trajectories, averaged over the run.

s to complete the run without assistance, which increased by an average of 12 s to complete the run with assistance. Since not all users require the same amount of assistance, Figure 4.13 shows the completion times as a percentage difference between the two modes, for each participant, as a more realistic measure. It can be observed that for some users (*e.g.*, 2nd, 4th), the completion time difference is negligible, which shows that the shared controller allows high speeds, when the robot is not in close proximity to obstacles.

For further exploring the safety aspect of the system, the average clearance of each run was also calculated. It was found that the shared control system increased the clearance compared to the runs with no provided assistance by an average of 9.3% among participants, with the average clearance of the runs without assistance being 38.7 (SD 3.2) cm and the respective metric for the runs with assistance being 42.3 (SD 1.9) cm. This result is consistent with the policy's reward objective of trying to keep distance from obstacles in order to receive a smaller penalty. Finally, the agreeability function (Eq. 4.15) was used to calculate how close the system's actions were to the users' desired trajectories. Calculating the average agreeability over all the participants resulted in a score of 0.79 (SD 0.02), whereas Fig. 4.14 shows the mean score, along with the deviation, of each participant. Overall, the scores are on the high side, indicating that the system respected the users'



(a) Speed commands (b) Turn commands

Figure 4.15: Participant's Trial Commands: Demonstrating raw user and system commands during the assistive system's usage. The system closely follows the user's input, but deviates when necessary to avoid collisions and maintain sufficient clearance from obstacles.

	QUESTIONS
Q1	THE WHEELCHAIR WAS EASY TO MANOUEVRE
Q2	THE WHEELCHAIR BEHAVED AS I EXPECTED
Q3	THE WHEELCHAIR EXECUTED MY COMMANDS
Q4	I HAD TO CONCENTRATE HARD TO DRIVE THE WHEELCHAIR
Q5	IT FELT NATURAL DRIVING THE WHEELCHAIR
Q6	I WAS ABLE TO EXECUTE THE TASKS QUICKLY
Q7	I FELT SAFE WHILE PERFORMING THE TASKS

Figure 4.16: Shared Control Trials Questionnaire. Each statement is scored by the degree of the participant's agreement on a five-level Likert scale (1=strongly disagree, 5=strongly agree).

intentions, and not too much assistance was required. Indicatively, the plots of a single participant's commands and the respective assistive system's commands are presented in Fig. 4.15.

Participant Feedback

Each participant indicated the degree to which they agreed with the statements in Fig.4.16 at the end of each trial (with and without assistance, not having been informed which mode they had just operated). The results show that, on average, participants tended to agree that the wheelchair was easy to maneuver, behaved as

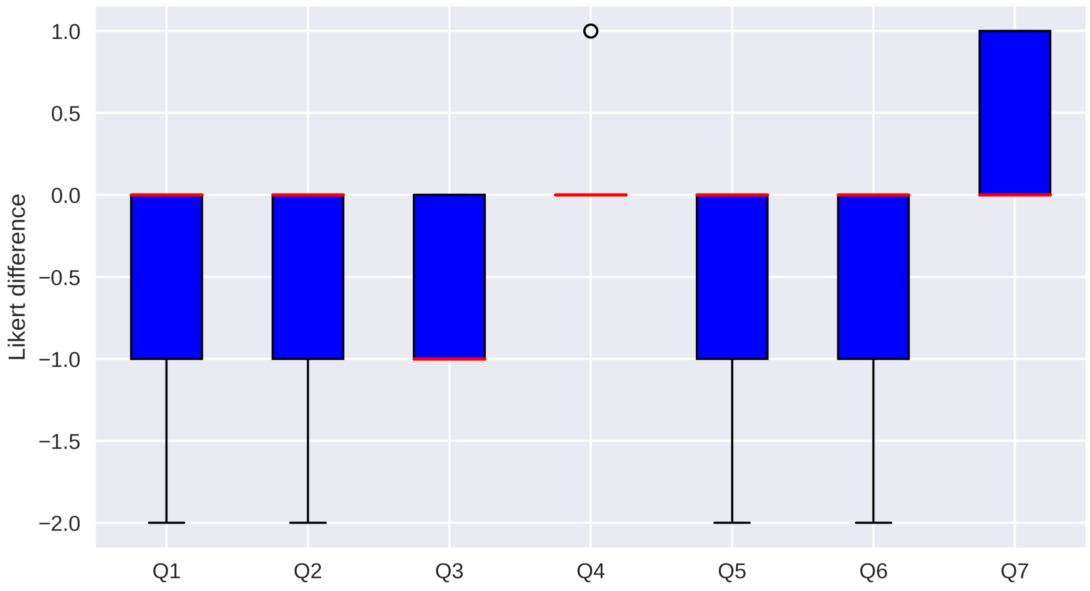


Figure 4.17: Agreement degree with Fig. 4.16 statements when using the shared control system compared to no assistance.

expected, and was intuitive to use, while they felt safe and only had to concentrate moderately hard on the task. The largest standard deviation across the subjects' answers was found in how hard they had to concentrate, and naturally, each person has their own expectations and perspectives of how they perceive task difficulty and personal effort.

To better understand how the participants perceived the difference between using the assistive system or not, the questionnaire data was also used to compare the Likert ranking of each statement, for each participant. The graph in Fig.4.17 shows the difference in the degree to which participants agreed with each statement, when using the assistive system. It can be observed, that most responses were in favour of using the wheelchair without assistance, with a statistically significant trend found in statements Q1 ($p = 0.029$), Q2 ($p = 0.011$), Q3 ($p = 0.012$) and Q6 ($p = 0.048$). This result is not surprising, since all participants were healthy and did not face any issues in driving the wheelchair efficiently. The shared control system, however, slowed them down and slightly altered their intended trajectories to increase clearance from obstacles, preventing them from taking sharp turns and completing the track as fast as possible. Participants did tend to agree that they felt safer when using the assistive system, however, no statistical significance was found($p = 0.174$).

4.7 Discussion

The setting that was examined within the proposed framework for shared control was mostly concerned with reactive planning. The system did not use any explicit goal inference to deduce the user's goals, but instead had to learn to implicitly decode the user's intent and comply with it, while optimizing for other objectives like safety. Results in simulation showed that the system can indeed enhance safety when used with an erroneous user model, but it also helps users with informative input reach their goals more often, even in environments it knows nothing about. Furthermore, the shared-control system exhibited adaptability to different user behaviors, however, it performed the best when used along with the same type of driving model it was trained on. This implies that training the system with user input that mimics a specific type of disability will result in more effective assistance for that type of disability.

Further customization of the system is also possible in various ways, due to the flexibility of the proposed framework. More specifically, changing the parameters, or even the type, of the distribution that models the user input, adjusting the value of the target divergence parameter, and tuning, or using a different, risk estimation process, all affect the resulting shared control policy. The same applies to the reward function that dictates the assistive component of the policy, which was kept simple within the scope of this work, but could be further engineered to provide assistance in more intricate ways. On the downside, those components need to be manually tuned, and reaching the desired result (an optimally performing policy) can be very time consuming and require significant computational resources. The same limitation was encountered in this work, which had to make some necessary compromises, like keeping the same user distribution throughout the various experiments and heuristically tuning the TD parameter and the risk estimation function. However, the promising results this thesis presented, without an exhaustive search for optimality, are indicative of the potential of the proposed framework for improved and specialized performance provided with the required resources.

Comparing our method with a conventional RL approach showed the necessity of the proposed algorithmic modification for compliance of the system with the user. The conventional approach was not able to maintain a balance between respecting user intentions and avoiding collisions, resulting in either ignoring the user or following them too closely and crashing. Another comparison was made with a more typical shared control approach, not based on machine learning, but on artificial potential fields. The APF method required precise obstacle information (distances, but also angles) to calculate an appropriate repulsive force that would prevent a collision, and thus, it was not able to utilize the sonar sensor setup to do so reliably. This showcases the advantage of a deep learning approach in such a case, where a neural network can arbitrarily leverage imperfect information in an effective manner.

Finally, transferring the shared-control model to the robotic wheelchair made a strong case for the applicability of an RL system in the robotics domain. The system that was trained in a simulation environment, and with a simulated user, was effectively transferred to the real world with no further tuning. Even though the participants' subjective feedback hinted at some dissatisfaction with the system interfering with their driving, the objective metrics revealed that the system allowed participants to achieve their goals, and do so in a safer manner. Some takeaways from the conducted experiments are that: 1) the trial scenario was relatively simple for healthy users, in which case assistance was not particularly needed 2) the system was trained in a generic way, in terms of the user model that was used to train it and the reward function that focused on avoiding obstacles, and not reaching specific goals 3) the same model was used for all participants, whereas a more careful selection of hyperparameters or fine-tuning of the model through user data could potentially improve performance for individual use.

4.8 Chapter Summary

This chapter presented a novel shared-control method based on the RL framework. The proposed method can be used to train an end-to-end system, which accepts as inputs user commands along with sensor data from the robot's onboard sensors, and directly outputs control commands for the robot, which should be compliant with the user's intention. In order to achieve this compliance, the method modifies the objective of maximum entropy reinforcement learning, so that instead of maximizing future rewards and policy entropy, it instead maximizes future rewards and minimizes the KL divergence between the user's intention (modeled as a Gaussian probability distribution) and the system's policy. This allows the assistive system to optimize for its own objectives, as dictated by the reward function, while being forced to keep its decisions close to the user's intention. To better guide the system when to keep close to or deviate from the user, a state risk estimation step was also introduced, which quantifies the risk of an intended control action based on the proximity to obstacles and the magnitude of the user intention.

Evaluation of the proposed system in simulation demonstrated its ability to adapt to users with different driving behaviors and aid them in avoiding collisions and reaching their goals with greater success, despite the system being agnostic to the users' policies or goals. The system was also benchmarked against a conventional max-entropy RL approach, as well as another shared control method based on APF, both of which were outperformed by a large margin. Finally, the chapter focused on transferring the proposed method to a real robotic wheelchair, by training a shared control policy with several aspects of the simulation being randomized to bridge the sim-to-real gap. Able-bodied participants used the wheelchair, with and without the assistive system, to navigate an indoor scenario while the statistics of the respective trials were recorded. The system, which was trained in a simulated environment with simulated users, effectively collaborated with all participants to complete the real-world scenario, with zero collisions recorded opposed to a total of three when using no assistance. However, the system hindered the users in navigating the scenario as fast and sharp as possible, resulting in less reported user satisfaction.

This chapter achieved the second and the third objective that were set in Sec. 1.2 and contributed a novel and generic shared control framework that offers a lot of flexibility in how it can be used. Different aspects and extensions of the proposed system can be further explored in future work, while its realistic applicability as an assistive technology in powered wheelchair navigation should be verified by trials with actual wheelchair users.

CHAPTER 5

Extension of the Assistance to an Alternative Control Interface

In this chapter, a novel, hands-free method for controlling a powered wheelchair is introduced. The chapter discusses existing alternative interfaces and their limitations, leading to the design and implementation of the proposed system. Utilizing a simple web-camera to track head movements, the system translates them into moving commands for an EPW. Real-world trials with healthy participants validate the system's performance. Furthermore, more challenging trials assess the head-control interface's effectiveness when combined with the shared control method introduced in Chapter 4, resulting in a remarkable 92% decrease in collisions using the assistive system. This chapter's objectives are twofold: 1) Introduce a robust and reliable control method as an alternative to the conventional joystick interface, addressing the inadequacies of existing interfaces for individuals with quadriplegia. 2) Investigate the impact of the novel shared control method on a less accurate control interface than the conventional joystick. The findings underscore the potential of the proposed head-control system in enhancing powered wheelchair accessibility and safety for users with mobility challenges.

5.1 Introduction

Loss of personal mobility due to spinal cord injury is a major challenge in the lives of affected individuals. According to a wide study conducted in the U.S. [139], about 45% of Spinal Cord-Injured (SCI) people end up with incomplete quadriplegia, and 13.3% end up with complete quadriplegia, rendering all four limbs paralyzed. While motorized wheelchairs have been instrumental in promoting independence among disabled individuals, traditional interaction interfaces such as joystick control are not applicable for quadriplegics who have lost hand function. Until today, some hands-free approaches have been developed for quadriplegia-disabled people to enable wheelchair control, with the aid of eyes, shoulders, speech, head, sip 'n puff [10], etc. Due to the nature of the quadriplegia patients' disability, operating the wheelchair by tracking head movements is a unique alternative solution that has drawn the attention of the research community.

Despite the importance of such a wheelchair-driving assistive system, literature shows that this area is understudied with only a few works having successfully applied it on a practical level. The relevant research can be classified in two main approaches regarding head gesture estimation; *invasive* and *non-invasive*. As reflected by the name, *invasive* approaches suggest the use of a sensing element (*e.g.*, tilt sensor, accelerometer, touch sensor, etc.) that requires contact with the user's head, and which can limit their movement range or cause discomfort. For instance, [140] has implemented a combination of a tilt and an accelerometer sensor, attached on the user's head for steering the wheelchair, or [141] that used two vertical and horizontal tilt sensors, requiring direct contact to the user's head, to operate a wheelchair. On the other hand, *non-invasive* approaches are usually image-oriented and do not require contact with the user's head. In this case, computer vision techniques can be utilized for image-based head pose estimation [142], as for instance [143], that proposed the use of a Kinect sensor for head gesture estimation to enable wheelchair control

The current work proposes a novel *non-invasive* method, based on visual head pose estimation, which is implemented on the EPW introduced in Section 4.6.1, while its performance is explored in a real-world scenario. Before presenting the method, the next section summarizes the existing control interfaces for powered wheelchairs and describes their limitations.

5.2 Powered Wheelchair Interfaces

Following the distinction between *invasive* and *non-invasive* approaches made in the introduction, a broader classification of methods can be made between *sensor-based* and *vision-based.* This section explores the most prominent sensor-based and vision-based head-controlled wheelchair studies.

5.2.1 Sensor-based Approaches

Due to their simplicity and reliability, sensor-based techniques have been the focus of hands-free wheelchair control. Currently, the conventional method used by quadriplegic patients for wheelchair control is the Sip-and-puff system, which allows basic control through a plastic tube mounted on the wheelchair. This system, even though low-cost and easy to use, has been reported to feel cumbersome, awkward and very slow. An alternative proposal to the Sip-and-puff is the Tongue Drive System (TDS) [144], which detects the tongue motion by using a magnet and magnetic sensors. This system, however, requires the tongue to be pierced, which can be quite uncomfortable, and also allows a limited number of commands. Other methods that have been explored by the research community rely on tracking the head position, which has been achieved with a diversity of sensors including accelerometer, tilt sensors, touch switches, or ultrasounds.

As one of the first sensor-based head-controlled wheelchairs,[145] introduced a powered wheelchair steering technique with the aid of head movements, derived by ultrasonic range measurements. In this study, they have mapped the head orientation, estimated by two ultrasound sensors, into four discrete commands; right, left, moving forward and stop. Another contactless sensor-based approach,[146] explored the performance of a head-controlled EPW, where the motion was recorded by an array of four infrared LEDs and a camera. The number of infrared LEDs in the vision of the camera (installed at the back of the user's head, on the head rest) change according to the yaw angle of the user's face, and the discrete commands of turning right and left are derived from the number of LEDs that are within the camera's field of view. [147] utilized an interface with mounted proximity infrared sensors, to track the eyeball motion of a user and detect intentional blinks and eye motions, which are mapped to driving commands for the wheelchair. Using inertial sensors is another method of deriving the head orientation for controlling EPWs, and it is explored in several studies. [148] is such an example, that used a MPU 6050 triple axis accelerometer and a gyroscope for monitoring the user's head motion and ultimately controlling a prototype EPW. However, no actual experiments verifying the effectiveness of the proposed method are presented. The authors embedded the accelerometer on the visor of a cap that needs to be worn by users. Similarly,[149] used an accelerometer to get the head position feedback before feeding that signal to an Arduino board to process the data and control a toy car, instead of a wheelchair. As an another example of sensor-based techniques,[150] used a gyroscope, attached on the user's head by a headband, for steering a real wheelchair. The proposed model has limited accuracy, whereas the accumulative error caused by the integration of the gyroscope output (known as the drift problem) has been considered negligible. Some other sensor-based approaches are presented in[141, 151, 140], where tilt sensors were utilized for X and Y displacement detection, which was then translated to discrete wheelchair movement commands. The effectiveness of the tilt sensor-based approach in[141] was verified through trials run on Spinal Cord Injured (SCI) patients, by performing some basic tests like travelling in a straight line or measuring the braking distance after sending a stop command. Another direction of sensor-based methods in wheelchair control focuses on the use of the Electroencephalography (EEG) technique[152, 153, 154]. This approach, however, depends on using an electrode cap placed on the user's scalp for acquiring the brain signals, and faces important challenges, like a small signal-to-noise (SNR) ratio and multiple different noise sources which corrupt the main signal. It is currently considered very challenging to design and implement such a system, that is also reliable, accurate and flexible.

The methods presented above, usually come with a number of drawbacks. Installing the sensors, as well as maintaining them, due to the high probability of getting damaged, can incur high costs. Furthermore, the inconvenience, or the wearing discomfort, caused to the users by solutions such as the TDS, head switches, devices attached to the user's head etc., is as another important drawback. Finally, the majority of the existing approaches operate with a discrete set of moving commands, thus, limiting the amount of available control to the user.

5.2.2 Vision-based Approaches

Vision-based approaches for powered wheelchair control are much rarer compared to sensor-based approaches. The most widely explored vision-based approach is based on eye-tracking, where a number of different methods have been proposed for implementing it in a robust manner [155, 156, 157, 158]. Such methods typically work with capturing images of an eye, and using image processing techniques (*e.g.*, Hough transform) to track the position of the eye pupil and then map it to a control direction for the wheelchair. One major shortcoming of such an approach is the limited reliability of tracking the eye pupil. Results can greatly vary depending on the quality of the image, the size or the colour of the pupil, the lighting conditions and the use of glasses or contact lenses. Furthermore, eye gaze control can be quite taxing to its user, since it requires constant eye movement and focus, which can cause dizziness or similar symptoms.

Despite eye-tracking, other methods utilizing cameras and computer vision algorithms have also been used on head-controlled wheelchairs, however, their number is limited. One such method is proposed in [143], where the head position is estimated with the aid of a Kinect and three landmarks on the head, by utilizing the optical flow technique. The estimated position is then mapped to four discrete commands for steering a wheelchair. However, the proposed method was only tested in controlling a pointer for following the path on a screen. Hu *et al.* [159] proposed a head gesture based interface for controlling a wheelchair, by using a combination of Viola-Jones [160], a well known face detection algorithm based on Haar features, and the Camshift algorithm for face detection and template matching for classifying the face posture. The reliability of these algorithms, though, is limited for such an application (Viola-Jones is not as effective detecting tilted or turned faces, while Camshift has poor accuracy). In a similar manner, [161] used a Haar cascade classifier for face and eyes detection, estimating their position to generate control commands for the wheelchair's motors. Notwithstanding the fast speed of the Haar classifier, it requires careful tuning, while it is known for high false positive detection rates and for being sensitive to the lighting conditions. No validation experiments were conducted for the aforementioned studies.

Other works propose using additional inputs, like mouth movements [162] or eye winking [163] along with the head position. However, this approach results in added complexity and discomfort using the system, while it allows more room for errors both from the system and the user side. Some studies including [162, 164], that have demonstrated their proposed systems in real world scenarios, provide implementations on laptops mounted on the wheelchair, which can harm the practical viability of a system that demands a low-power and mobile solution. Furthermore, in these works there is absence of subjective metrics and overall user experience evaluation, which is vital for assessing the actual usability of such systems.

A different type of approach is presented in [165], where the authors proposed an active vision approach, and instead of using a frontal camera pointing at the user's face they use an on-head camera. Halawani *et al.* used SIFT (Scale-Invariant Feature Transform) points to monitor head motion, and by comparing interest points extracted from consecutive image frames they obtain the movement direction in a discrete manner and steer the wheelchair in a real-world scenario. This method relies on identifying and matching stable and repeatable points in different images, and even though it is a vision-based approach, it requires contact between the user and the camera (being installed on their head), which might cause inconvenience similar to sensor-based approaches.

5.2.3 Limitations of Head Control Interfaces

Exploring the literature on head-controlled wheelchairs reveals that very few of the proposed solutions have been put into real use. Namely, the Sip-and-puff, TDS and eye-gaze technologies are the ones being used today by quadriplegic patients. Sip-and-puff and TDS are invasive methods, that can be burdensome to use for prolonged periods of time, while they offer limited control freedom. Eye-gaze, though vision-based and non-invasive, also does not allow fine control, while it can be unreliable and taxing to the user. Other works based on computer-vision are quite rare, and are usually sensitive to environmental conditions (including varying illumination, indoor and outdoor environments, cluttered backgrounds, and shadows[166]), or their effectiveness has not been explored in depth by being applied to a real system and tested with human subjects.

This study presents a vision-based, non-intrusive, cost-efficient system for controlling a wheelchair via head movements, only relying on a simple web camera. The system robustly performs face detection and head pose estimation, improving upon similar existing methods[159, 161], by combining a state-of-the-art neural network and a fast and accurate facial landmark detector. Furthermore, the system design includes a calibration process that allows customization to each individual's range of motion and a control logic which allows users to maintain full control over a powered wheelchair. Furthermore, the control can be done in a continuous manner, something that is not present in other works. The system has been deployed on a commercial wheelchair and its effectiveness has been evaluated on a real world scenario with healthy human participants, which will be presented later in Section 5.4.

5.3 Methodology

A robust head-controlled wheelchair requires two important elements; a reliable head pose estimation algorithm and a robust translation of the orientation to commands for the robotic wheelchair. Therefore, the proposed method consists of

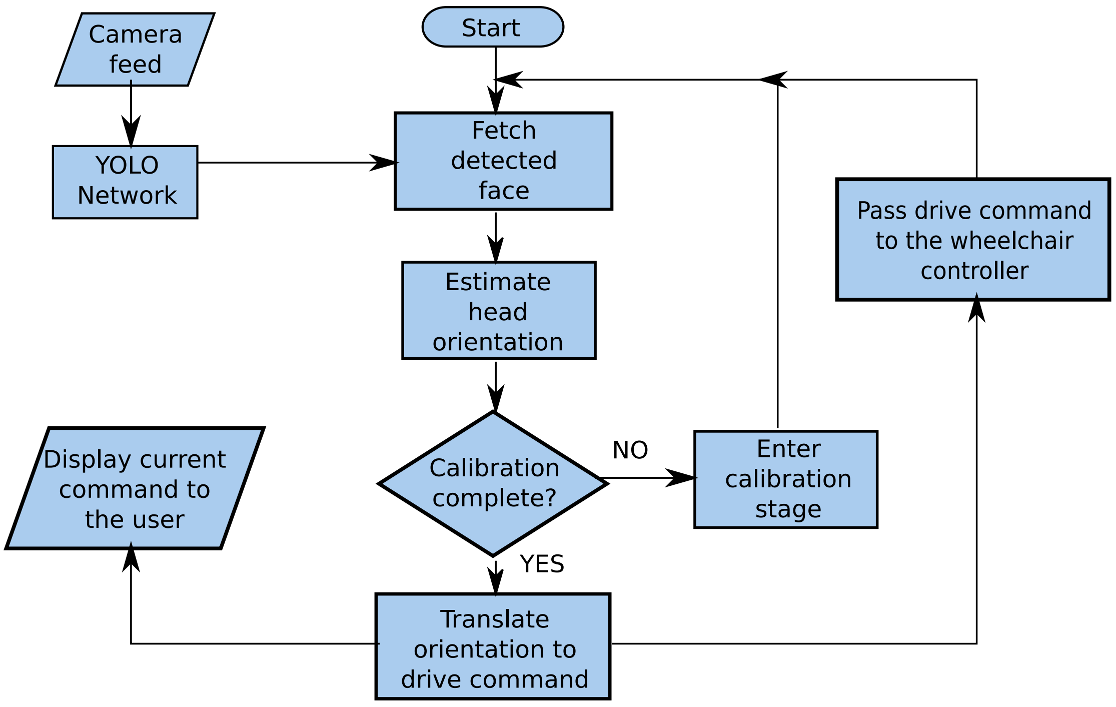

Camera feed
YOLO Network
Start
Fetch detected face
Estimate head orientation
Calibration complete?
NO
YES
Enter calibration stage
Pass drive command to the wheelchair controller
Display current command to the user
Translate orientation to drive command

Figure 5.1: The Head Control System's Flowchart.

two main modules, one for estimating the head orientation (pitch and yaw), and another one for translating the derived orientation to velocity commands, before passing them to the control module of the EPW. The head pose estimation step utilizes "YOLO" (You Only Look Once), a widely-used Convolutional Neural Network (CNN), along with a facial landmark detector, which enables the calculation of the head pose. The output of the head pose estimation is mapped to moving commands for the robotic wheelchair, using both discrete control commands (predetermined velocity values for turning), as well as continuous control commands (turning is a linear function of the head's yaw). A more detailed explanation is provided in the rest of this section, where the relevant methodology is described.

5.3.1 Head Pose Estimation

For estimating the head orientation, in the pitch and yaw dimensions, two modules combining machine learning and more traditional computer vision techniques are utilized. The first module is responsible for detecting the head of a person within an image, whereas the second one is used to approximate the orientation of the detected head. Head detection uses the well-known YOLOv3 network [167], which

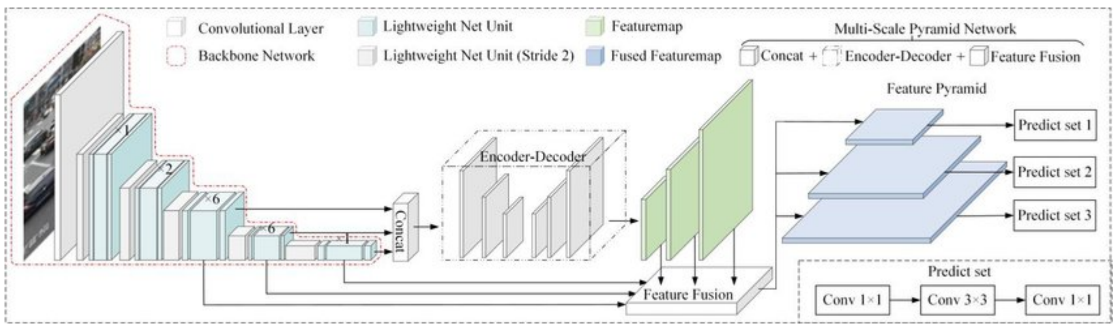


Figure 5.2: An overview of the Mini-YOLOv3, by Qi-Chao-Mao *et al.* , licensed under CC BY 4.0.

is a fast and efficient Fully Convolutional Neural Network (FCNN), capable of detecting multiple objects in almost real-time. More specifically, for this study the mini-YOLOv3 network was used (a miniature version of the full YOLOv3), which was retrained to specifically detect faces. The reason mini-YOLOv3 was used is for improving the performance, in terms of frames per second (fps), due to its much smaller size compared to the full YOLOv3 network. The accuracy of mini-YOLOv3 is comparable to the original one, and even more so in this case, where we only care about detecting the closest face to the camera, a rather easy task for the particular network. Furthermore, the effectiveness of mini-YOLOv3 as a real-time object detector for embedded applications has also been shown in the literature [168]. The implementation that was used was "DarknetROS" [169], which, as the name implies, plays well with using it alongside ROS, that acts as the backbone of the overall system (details about ROS have been provided in Section 3.3.1).

The detected face is then passed on to the second module, which uses computer vision techniques in order to locate the facial features contained in the image. For specifically, the feature extraction is realized with the *facial landmark detector* included in the "Dlib" library, which is an implementation of the work done in [170], that learns an ensemble of randomized regression trees to detect landmarks on a face image. The particular method was chosen for its proved effectiveness and high speed [171], vital for achieving real-time processing times. The feature of interest that is selected is the tip of the nose, the position of which provides a good approximation of the head's orientation. The nose position is given in x and y coordinates of the given frame, but since the bounding box of the face is

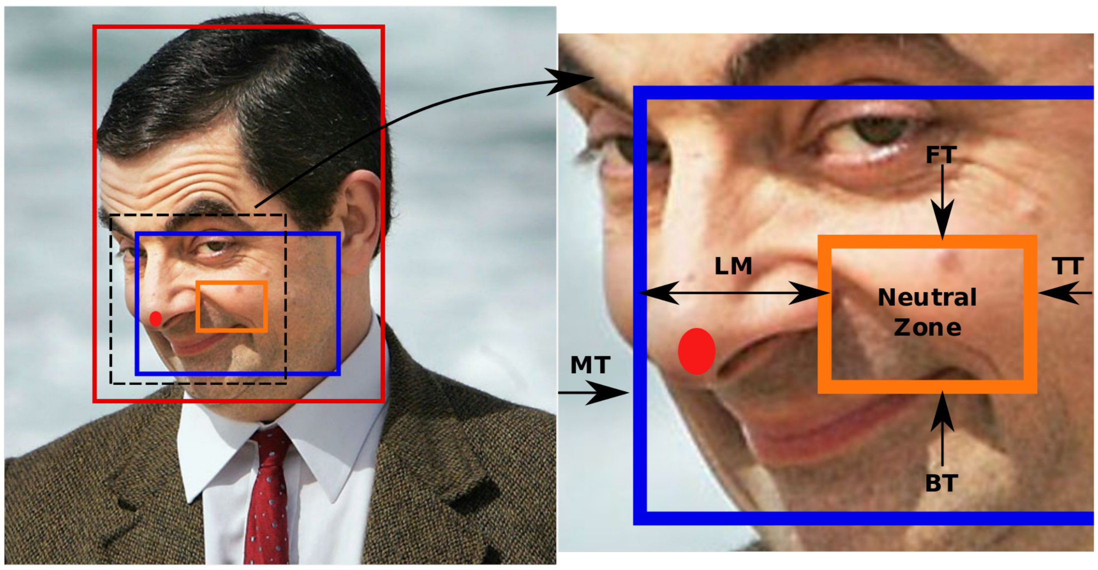
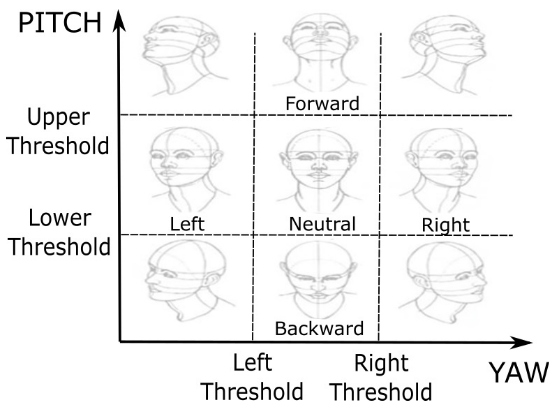


Figure 5.3: **Left** - The detected face (red bounding box) and the calibration boundaries, against which the nose tip position (red circle) is checked. The orange bounding box indicates the limits of the neutral zone, and the blue the limits of the maximum motion in both directions, as calibrated by the user (see Sec. 5.3.2) **Right** - **FT**: The threshold for initiating forward movement **BT**: The threshold for initiating backward movement **TT**: The threshold for initiating turning **LM**: The interval at which the horizontal nose tip position is linearly mapped to angular velocities (only in continuous mode) **MT**: The threshold beyond which the maximum allowed turning speed is set

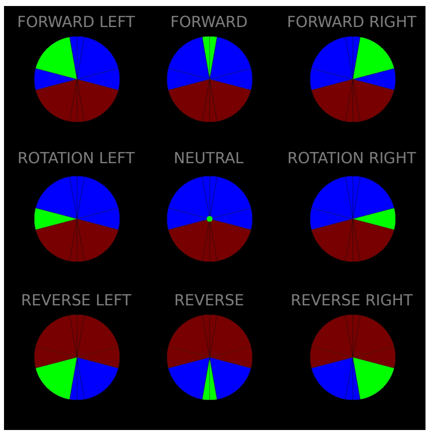
of variant size (depending on the distance of the face to the camera), the head orientation is calculated as percentages of the nose's location in the image relative to the current frame's size (rows × columns). Naturally, the vertical coordinate changes along with the pitch movement of the head, whereas the horizontal with the yaw movement as shown in Figure 5.4a.

Smoothing the input

Given the noisy nature of the pose estimation, instead of directly using the current pitch and yaw values for moving the EPW, a moving average is computed using the *sliding window method* for the past k frames. This makes the implementation much more robust for two reasons: Firstly, it prevents the translation of noisy estimates to immediate driving commands, and secondly, it allows space



(a) Mapping of head position to moving commands.



(b) The moving commands displayed as a virtual joystick to the user.

Figure 5.4: Visualization of the Head Control System's Functionality.

for the user to make errors, like for example sudden tweaks of the head, without instantaneously altering their driving course. Of course, the size of the window is adjustable and dependent both on the hardware (fps of the head detection) and also the preference/capability of the individual user, and therefore needs to be tuned accordingly. For example, given the last k pitch measurements, the current smoothed pitch value would be given by: $\bar{p}_n = \frac{1}{k}\sum_{i=1}^{k} p_i$

For the next moving average calculation, and in order to save processing time, the calculation can be simplified by reusing the previous moving average. A new pitch value p_{n+1} comes into the window, whereas the oldest value p_{n-k+1} drops out, thus we get: $\bar{p}_{n+1} = \bar{p}_n + \frac{1}{k}(p_{n+1} - p_{n-k+1})$.

5.3.2 Head Pose to Control of the Robotic Wheelchair

Following the extraction of the values corresponding to the pitch and the yaw of the head, the control script responsible for translating those values to velocity commands, as well as implementing the control logic to operate the EPW, is applied. Here, it is important to mention that in the case where either the camera loses the user's face, or in the exceptional case that there is no valid face detection or pose estimate, no control command will be issued to the wheelchair, which will cause it to come to a halt.

A valid head pose can be translated in either a *discrete* or a *continuous* manner, where in the first case the discrete mode (DM) maps the pitch values to three regions (“moving forward”, “moving backward”, “neutral”), and likewise the yaw values to three regions (“turning right”, “turning left”, “neutral/inactive”). In the second case, the continuous mode (CM) retains the same mapping for the pitch, but uses a linear mapping for the yaw values to angular velocities. The robot velocities obey the differential drive kinematics, as expressed in Eq. 3.17.

The two modes and the control flow of the system, which is also summarized in Figure 5.1, are further explained below.

Discrete control

Before sending commands to move the robotic wheelchair, a calibration phase needs to first take place. In the calibration phase, the user is given some fixed time to set their “neutral zone”, which is defined by the maximum and minimum values of the pitch and yaw estimates, when the user moves their head very slightly in all directions, while seated in their neutral/comfortable straight position. The neutral zone can be visualized as a virtual box around the user’s nose, where, while the nose remains there, no velocity commands are being sent to the EPW.

After the calibration is complete, we enter the driving phase, during which the current orientation is checked relative to the neutral zone. When the pitch value exceeds the upper bound of the neutral zone in the vertical direction, the linear velocity v_x of the EPW is set to a predefined positive value, and remains so as long as the vertical position does not exceed the lower bound of the neutral zone. If the latter happens, then the linear velocity is set back to zero, unless it remains there for a set amount of time, in which case “reverse” is enabled, setting the linear speed to a predefined negative value. Reverse can then be cancelled by again exceeding the upper vertical bound.

Similarly, the angular speed ω of the EPW is set to a predefined positive or negative value, when the yaw value exceeds the right or left boundaries (in the horizontal direction) of the neutral zone respectively. The difference when turning, is that the head position needs to remain either right or left, and beyond the neutral thresholds, to maintain a non-zero angular speed. This is done purposely to make turning more responsive, as well as easily cancelable simply by moving the head back to the neutral position. When the linear velocity is set to zero, the user can turn their head right or left to make the EPW rotate on the spot towards the desired direction.

Continuous control

With “continuous control” we refer to producing continuous, instead of constant values, for controlling the angular speed of the EPW. In this mode, the linear speed again remains constant as previously described. The control process for continuous mapping is similar to the discrete one, with two important differences. Firstly, an additional step is added in the calibration phase, where the user has to also set their maximum (comfortable) range of motion in the yaw direction, in order to obtain another boundary in the horizontal direction outside the neutral one. Given this additional boundary, any right or left head movement that exceeds the neutral zone, can now be mapped to a continuous value between a minimum one (a fraction of the maximum angular speed, $\omega_{min} = \alpha \cdot \omega_{max}, \alpha \in [0, 1)$), up to the maximum allowed one (ω_{max}) when the head is at (or exceeds) the edge of the outer boundary. For example, in the case of a right turn of the head, and given the calibrated right neutral boundary nb_r, the far right boundary b_r, and the estimated *yaw* value y, the angular velocity mapping is described below:

$$\omega(y) = \begin{cases} 0, \text{if } y \leq nb_r \\ \omega_{min} + \frac{\omega_{max} - \omega_{min}}{b_r - nb_r}(y - nb_r), \text{if } y > nb_r \text{ and } y \leq b_r \\ \omega_{max}, \text{otherwise.} \end{cases} \tag{5.1}$$

The same calculation also applies for a left turn, but checked in respect to the left calibrated boundaries and mapped to negative angular speeds. A graphic depiction of the above process is provided in Figure 5.3.

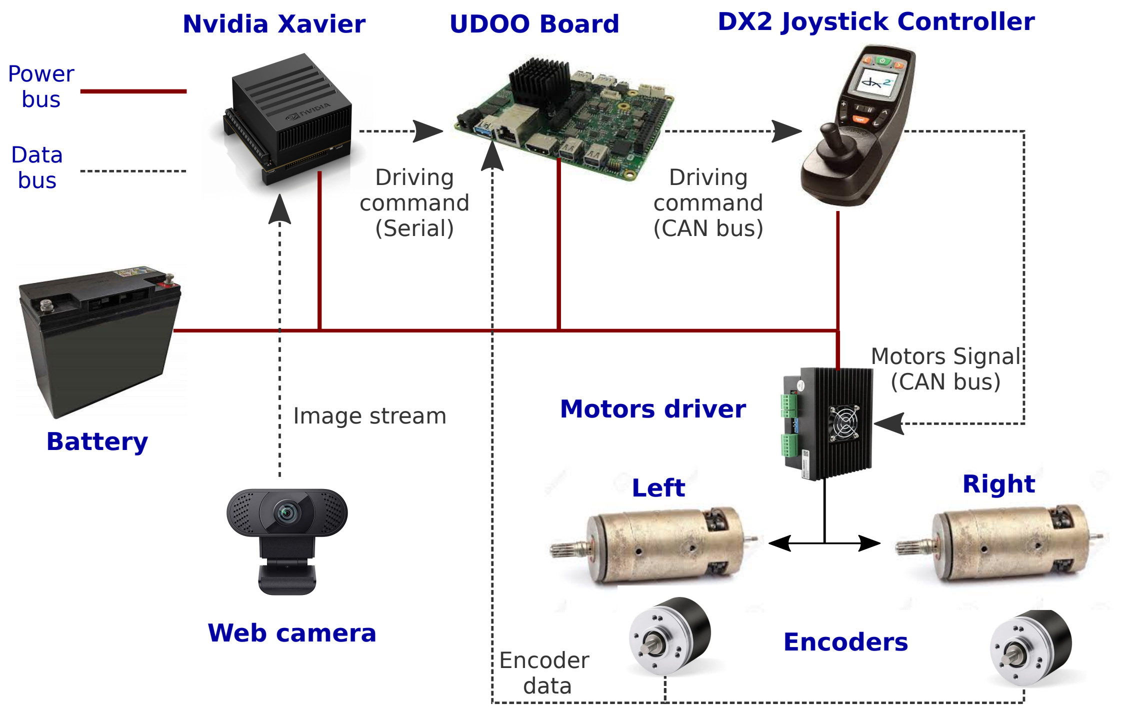


Figure 5.5: The Robotic Wheelchair's Hardware Setup.

5.4 Validating the Novel Vision-based Interface

In order to test the performance of both approaches, a set of experimental tests was conducted on a real EPW, in which the behavior of users, models and wheelchair has been extensively explored.

5.4.1 Experimental Protocol

In this study, 14 healthy people were recruited to participate in the experimental tests. We run three experiments per participant, where they were asked to navigate the wheelchair through the designed track (as shown in Figure 5.6) in the following modes: (i) using the manufacturer's standard joystick interface (ii) using the head-control method in DM (iii) using the head-control in CM. The age of the participants ranged between 21 and 56 years old. For safety reasons, an emergency button was connected to the wheelchair to disconnect the main power in the case of an emergency, and for overall safety. The participants were also instructed to use the joystick (which would override the remote head commands)

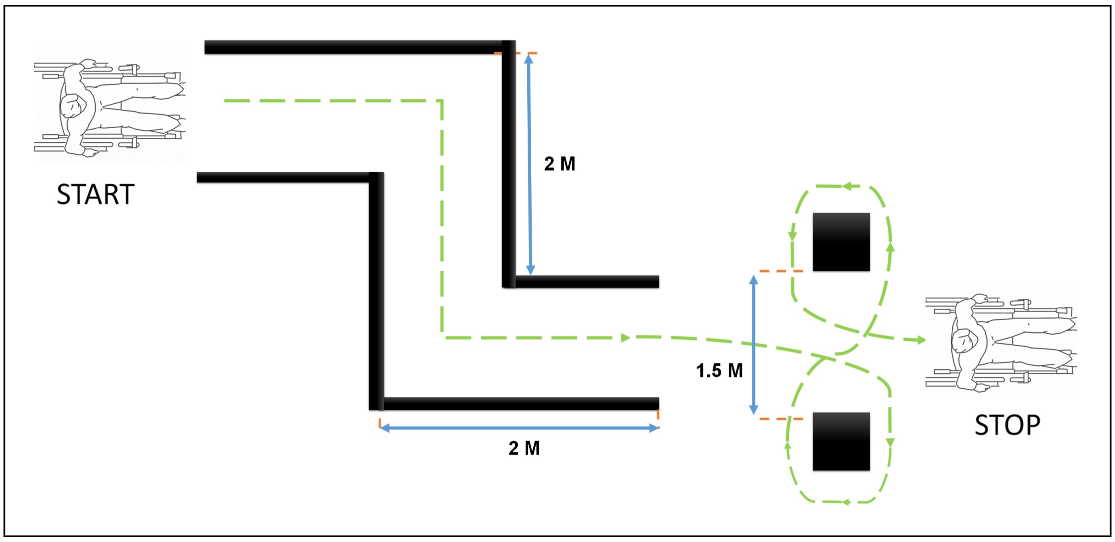


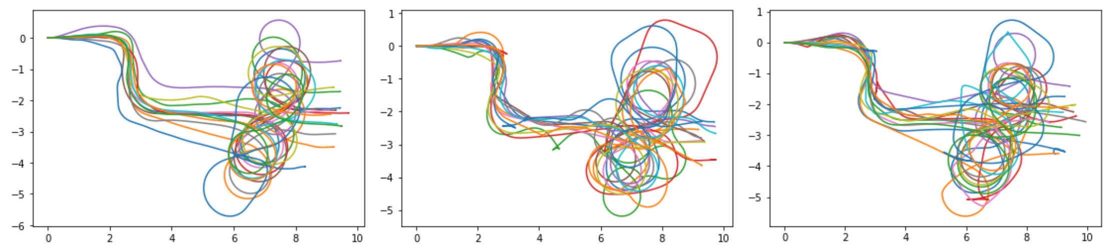
Figure 5.6: The Experimental Route Layout for Validating the Novel Head-Control Interface.

in case they lost control and were about to collide, but not for correcting the trajectory of the wheelchair. Before each trial, every participant was provided with three opportunities to calibrate the system.

The participants were given up to 15 minutes to familiarize themselves with the setup as well as the driving behaviour of the EPW, and also a 2-minute break between the different experimental rounds, in order to minimize the effect of tiredness or any other unwanted factors. An interview was conducted after the trial to gauge the participant's satisfaction with the system and to collect some subjective measurements. The participants were asked to provide a number between one and five (1-5), stating their experience with using a powered wheelchair.[1]

For the sake of performance evaluation, the *trajectory*, *number of collisions*, *travelling time* and *travelling distance* were recorded throughout the trial runs. The number of collisions, as well as the travelling time, were recorded by the researchers overviewing the trials, whereas the trajectory and the travelling distance were extracted from the odometry data derived from the pair of encoders installed at the EPW's wheels (see Sec. 4.6.1). The performance of the participants has been evaluated qualitatively and quantitatively and is discussed in the following chapter.

[1]Some of the participants were members of the research teams working in the area of assistive technologies, therefore having experience with wheelchairs, despite being non-disabled.



(a) Joystick (b) Head control in DM (c) Head control in CM

Figure 5.7: The Trial Trajectories as Recorded from the Odometry.

Table 5.1: Participant Trial Results in the Three Driving Modes.

Mode	**Statistics**					
	Distance (m)		Time (s)		Collisions*	
	Mean	Std	Mean	Std	Mean	Std
Joystick	21.2	1.31	67	0.08	0	0
DM	23.4	2.73	92	0.29	2.84	2.07
CM	23.2	2.13	81	0.23	2.5	2.44

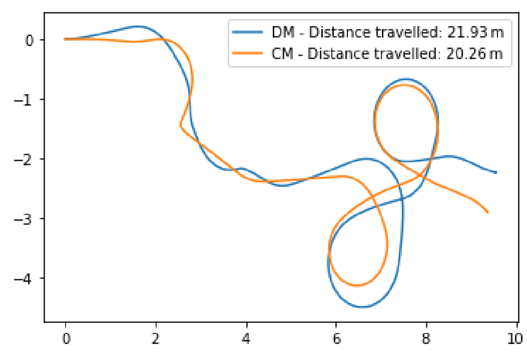
* As collision we regard any contact of the wheelchair with obstacles, plus any momentarily intervention with the standard joystick controller, in cases where users felt they were not in control or panicked

5.4.2 Results

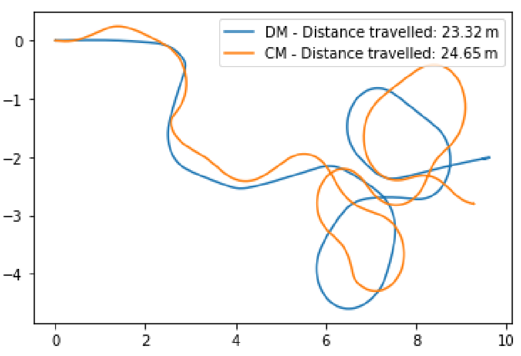
The participants' performance was evaluated in each of the three modes with a set of evaluation metrics as:

- Mean and standard deviation of traveling time
- Mean and standard deviation of traveling distance
- Number of collisions
- A number of subjective measurements inspired by the Nasa Task Load Index [172] (*only in head-control mode*)

The summary statistics of the travelling distance, travelling time and the number of collision are presented in Table 5.1. For gaining a better insight of the participants' performance in each mode, the travelling trajectories, constructed from the recorded odometry, are also presented in Figure 5.7.



(a) Recorded trajectories of an experienced (5) user.



(b) Recorded trajectories of an inexperienced (1) user.

Figure 5.8: Trajectories of an Experienced vs an Inexperienced User of Head Control in DM and CM. The experienced user takes better advantage of the CM, managing to perform sharper and more accurate turns compared to the DM. On the other hand, the inexperienced user has a more "shaky" performance in the CM, thus performing worse than the DM which has a more predictable behaviour.

Analysis

For the 14 participants a total of 28 distinct runs were conducted using the head control system (2 modes per participant) and another 14 using the standard joystick interface, making a total of 42 runs. For a fair comparison, the commands issued from the joystick were limited to match the maximum allowed velocities of the head control mode. Out of the 28 head-controlled runs, 27 were successfully completed, whereas there was 1 case where the participant could not complete the trial, due to poor calibration of the system ("reverse" could not be activated). As stated in the study protocol (Section 5.4.1), each participant had up to three tries to successfully calibrate the system.

Comparing the *discrete* versus the *continuous* translation of the head pose estimation to moving commands reveals that the *continuous* performed slightly better, in terms of completion times, but also shorter trajectories and smaller number of collisions. This can also be verified visually by comparing the resulting trajectories between Figure 5.7b (*discrete*) and Figure 5.7c (*continuous*). This result was expected, and is accounted to the finer control (or greater resolution) for turning that the *continuous* implementation allows. However it was observed that the CM performed better for some participants, while for others the DM was the better option, especially for the ones who had no prior experience with using an EPW. To explore this observation further, the completion times of the participants are split into two groups; one with the absolute amateurs in wheelchair driving (1), and the other with the rest of the participants that had little to much experience (2-5). On those groups a two sample t-test (Welch's t-test, alpha level of 0.05) was performed, where a statistical significance was found in completion times when using the CM ($p = 0.045$). On the other hand, a significant trend was not found for the respective test in the DM ($p = 0.41$).

Furthermore, a correlation analysis was run between the participants' self-reported wheelchair experience and the users' performance in terms of completion times. The metric that was used was the *Pearsons correlation coefficient* (PCC) [173], which is an interpretable measure of linear correlation between two sets of data. The scores of -0.19 and -0.46 were obtained for the *discrete* and *continuous* mode respectively. The negative correlation shows an inverse relationship between completion times and experience (the more experience the shorter the completion times), with the correlation in the DM being small, and in the CM being moderately high. This is an indication, that having no prior experience with driving a wheelchair does not greatly affect the performance in the DM, but does affect the performance in the CM, making it possibly a better option for experienced users. The DM working better for inexperienced users also makes sense intuitively, since it allows more room for error, while the CM might be harder to operate (more sensitive to horizontal head movements), but also has the capacity of performing better than the DM due to the finer control it allows. Figure. 5.8 shows an example of recorded trajectories by an experienced and an inexperienced user, which highlights the results presented above.

To compare the performance between the joystick and head-control modes, we conducted pairwise statistical comparisons using the completion times of the participants. We first compared the joystick and head control in discrete mode using a two-sample t-test (Welch's t-test, alpha level of 0.05). The analysis showed a significant difference between the two modes ($t(24.67) = -6.04, p < 0.001$), with the joystick mode having significantly shorter completion times. We then compared the joystick and head-control in continuous mode using the same statistical test. The results revealed a significant difference between the two modes ($t(18.12) = -4.58, p < 0.001$), with the joystick mode again having significantly shorter completion times. Furthermore, in the DM we have an increase of 10.4% for the trajectory length compared to the joystick, whereas in the CM we have an increase of 9.4%. These findings suggest that the joystick mode offers better performance than the head-control modes, which is in line with our expectations due to the easier use and better accuracy of the joystick. However, it is worth noting that the head-control modes were not significantly inferior to the joystick mode, indicating that they could serve as a viable alternative for individuals who struggle with operating a joystick.

Looking at the statistics of the collisions that occurred during the head-control runs, we observe high standard deviations (2.07 in DM, and 2.44 in CM). The people that got accustomed to the system, drove with a good level of control and had no to very few collisions. On the other hand, some participants were unable to adjust to the system within the allotted time of 15 minutes. This was due to either failing to calibrate the system properly or getting nervous during the trials. As a result, they lost control and crashed multiple times before completing the course.

Subjective Measurements

Despite the objective measurements that were recorded and analyzed, evaluating the participants' experience of using the system is also important. For that purpose, measurements of the NASA Task Load Index (NASA-TLX), a widely used, multidimensional assessment tool that rates perceived workload in order to assess a system's effectiveness, were utilized. More specifically, the NASA-TLX measurements were used to quantify the following questions:

- *Mental demand* - How mentally demanding was the task?
- *Physical demand* - How physically demanding was the task?

- *Performance* - How successful were you in accomplishing what you were asked to do?
- *Frustration* - How insecure, discouraged, irritated, stressed, or annoyed were you?

The participants were asked to provide their own evaluation on the above categories on a scale of 1-20, 1 being "Very low" and 20 being "Very high", in order to assess their overall experience with using the head control system (not differentiating between the two modes). The results are presented in Table 5.2.

Overall, we can see that the participants reported a low physical demand (mean of 5.6), but higher mental demand and frustration (9.3 and 7.1 respectively), although still reasonably low. The physical demand has a low standard deviation (3.9), indicating agreement of the participants, whereas the deviation increases when it comes to the mental demand (5.2) and frustration (5.5), showing that the task was less taxing for some of the participants, but more taxing to others. This result also agrees with the observation of the previous section, since running a correlation analysis between the experience and the frustration and the mental demand of the participants resulted in a PCC of -0.34 and -0.29 respectively, indicating an effect of the experience on the perceived effort of the users. On the contrary, a PCC of -0.13 between experience and physical demand shows a weak correlation of how the users' physical comfort was affected by experience. Additionally, calculating the PCC between mental demand and the number of collisions, results in 0.04 in DM and 0.44 in CM. These results reveal that the mental demand that was experienced by the participants was mostly accounted to the CM, rather than the DM, for which the correlation was negligible. On the other hand, the participants reported a high evaluation of their own performance (15.0), with the lowest deviation (2.8), implying that overall they felt confident with their ability of using our system successfully for the task they were given.

Table 5.2: Participants' Subjective Measurements Results.

Metrics	**Statistics**		
	Mean	Std	Median
Mental Demand(1-20)	9.3	5.2	8
Physical Demand(1-20)	5.6	3.9	5
Performance(1-20)	15.0	2.8	15
Frustration(1-20)	7.1	5.5	5
Wheelchair Experience(1-5)	2.4	1.5	2.0

5.4.3 Discussion

The results of the conducted trials showed that the non-disabled participants were able to successfully use the head-controlled system and complete the designed track, demonstrating the effectiveness of the system in matching the use of a standard joystick controller. Although the joystick proved to be a faster and more accurate means of control, the head-controlled system was not far behind, especially given the difficulty difference between the tasks and the limited experience of the participants with the head-controlled system.

In terms of the head-control mode, the comparison between the continuous and discrete modes showed that the continuous mode performed slightly better in terms of smoother and more accurate control, particularly for users with more experience in driving a wheelchair. However, this was not the case for all participants, and the discrete mode may be a better option for inexperienced users as it provides more room for error. The statistical comparisons performed in the study support these findings, with a significant difference in completion times between the continuous and discrete mode for users with some experience in wheelchair driving, and a moderate negative correlation between completion time and experience for the continuous mode.

Furthermore, the comparison with the joystick control mode revealed that head-control with either the continuous or discrete mode can produce comparable results, albeit with slightly longer completion times and trajectory lengths. However, the absence of collisions during joystick runs indicates that joystick control remains the most reliable and safe option.

Calibration of the head-controlled system was found to be an essential factor in successful runs, with good calibration allowing full control of the wheelchair within a reasonable range of head movement. More work will need to be done on this aspect to ensure more consistent calibration results, as for example adding more visual cues to guide the user, while an additional step that allows users to set their desired velocity ranges would provide better customization. Changes in user posture, which can occur due to factors such as fatigue or differences in terrain, can have a considerable impact on the accuracy of the calibrated system during use. To address this, a dynamic re-calibration process during driving would be a valuable addition, allowing the system to adapt to such changes, leading to a more comprehensive and resilient solution.

The reports of the participants revealed that the most demanding part of the experiment was the calibration/training part, where they needed to familiarize themselves with the system and build confidence in using it. However, given the low reported physical demand and the potential reduction of mental demand and frustration with more experience, the head-controlled system may be a realistic and appropriate option for usage by actual wheelchair users. Further research and experimentation will need to be conducted to verify this hypothesis.

The findings through experimentation revealed the operational characteristics of the system and identified factors contributing to its performance, including control frequency, calibration process, and user experience. The control frequency of the system was addressed by achieving an overall frequency of around 30Hz, which proved adequate to smoothly control our EPW. The use of mini-YOLOv3 played a crucial role in this, as when using the full YOLOv3 instead, we only achieved a frequency of $\sim$5Hz, which significantly affected the final performance of the system at a preliminary testing stage. The calibration process was found to be sensitive and not very intuitive for some users, however, building more experience with the system makes the process easier and more effective over time, which in some cases was not possible to be achieved within the short training time provided in the experiment. We are confident that adding more visual cues to guide the calibration process will greatly improve its robustness, and therefore, the driving performance.

Reverse movement is supported by our design, but it should be combined with the feed of a rear-view camera to make it fully usable. Alternative filtering methods could be used to translate head position to velocity commands, and combining the output of our system with a collision avoidance module would ensure safe navigation. Finally, and most importantly, the system needs to be tested with real powered wheelchair users (especially quadriplegics in a clinical trial setting) to verify its end goal effectiveness for the target population.

5.5 Investigating the Effects of Assistance to the Usability of Alternative Control Interfaces

Now that the novel vision-based interface has been introduced, and validated as a viable, yet less accurate, control interface for an EPW, this section explores whether the shared control method, introduced in Ch. 4, can be used to improve the usability of such a system. This section presents real-world experimentation combining the vision-based interface for controlling the EPW with the RL-based assistive system, and analyzes the relevant findings. The head control mode was set to the continuous mode, in order to better match the continuous nature of the shared control system, but also to provide a bigger challenge to the participants and highlight the effects of the assistance.

5.5.1 Experimental Protocol

The experimental protocol that was followed, is identical to the protocol described in Sec. 4.6.3, where the shared control system was tested with the standard joystick interface. 13 participants were asked to navigate the scenario shown in Fig. 4.11 to the best of their ability, while using the head-based control interface, in two separate trials. One of the trials would be with the assistive system active and the other one without, while that occurred in a pseudo-randomized way, with

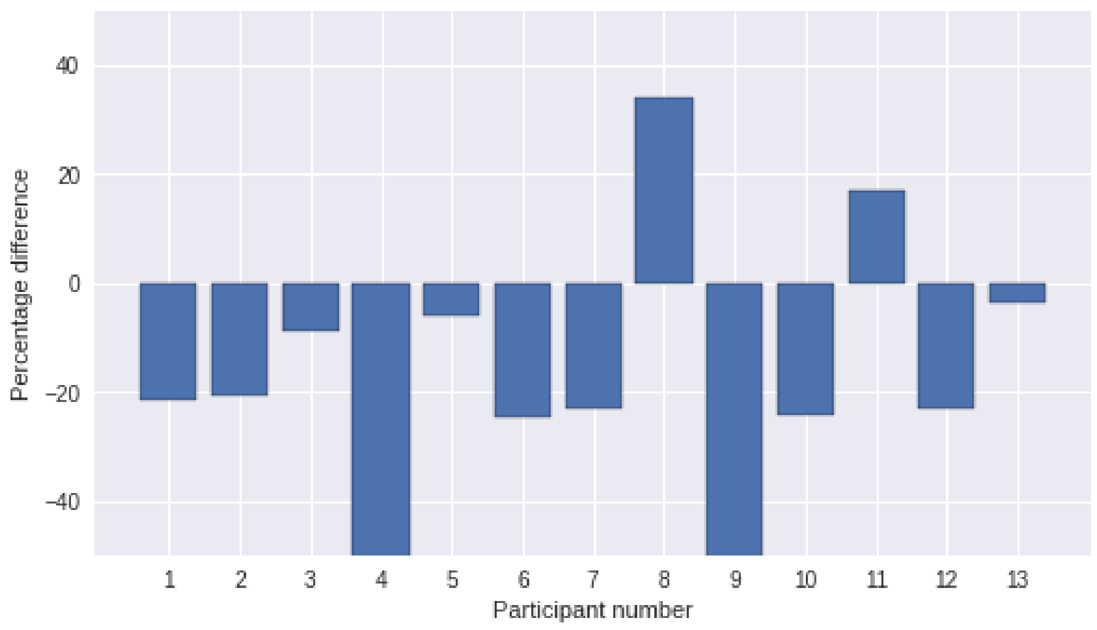


Figure 5.9: Differences in Participants' Completion Times when Using the Assistive System with the Head-Control Interface. The faster completion times while using assistance are directly attributed to fewer collisions occurring.

the participants being unaware of the mode they would operate each time. The participants were given up to 10 minutes to familiarize themselves with the control interface, and at the end of each trial they were asked to fill a questionnaire about their experience (Fig. 4.16).

5.5.2 Results

Task Metrics

The objective metrics that were used to assess the performance of each run were again: distance travelled, task completion time, clearance (average minimum distance to nearest obstacle), number of collisions, as well as the agreeability between the user's (head) commands and the assistive system's final control command.

In contrast to the joystick trials (Sec. 4.6), where the able-bodied users faced little difficulty in maneuvering the designed track, the head-control trials proved to be much more challenging for the majority of the participants. The completion times between the trials with different control interfaces are not directly comparable in

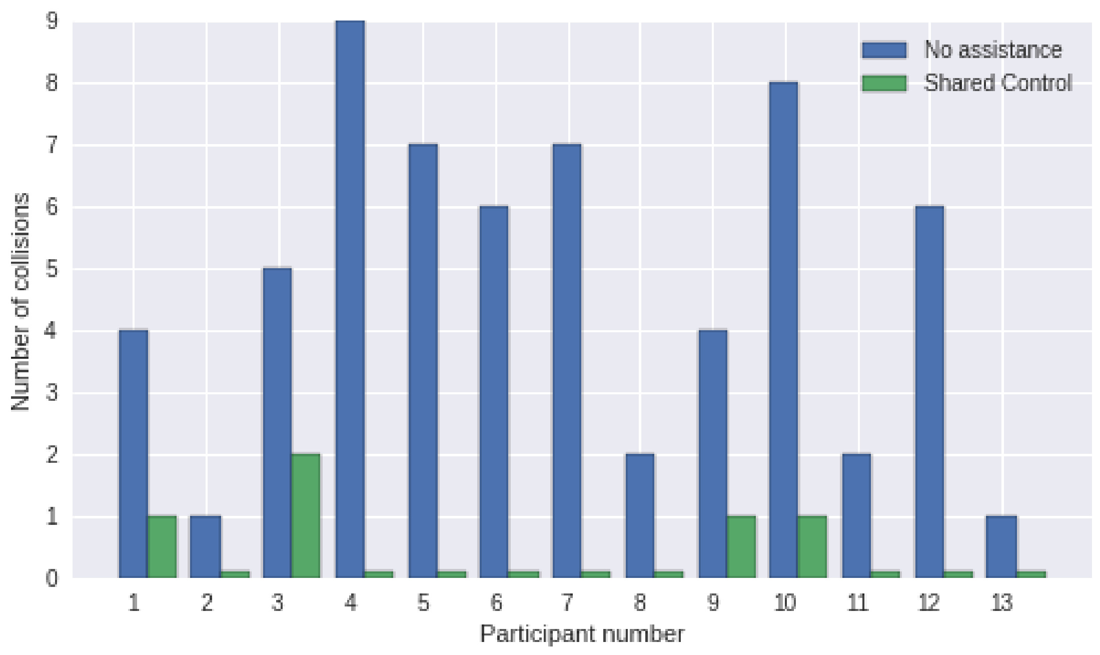


Figure 5.10: Number of Collisions Each Participant Experienced Under Each Mode of Operation, while using the head-control interface.

this case, since the linear speed command of the head control was a constant value, at a fraction of the maximum allowed speed of the wheelchair. However, when using no assistance, participants took 159.36 (SD 51.16) s, on average, to complete the track with the head control, whereas when using assistance the completion time dropped to an average of 135.4 s (SD 33). This is a decrease of 15%, compared to the respective trials with the joystick, where there was an increase of 22.3% in completion times when using assistance. Even though the assistive system was shown to slow down near obstacles, this decrease in completion times was a direct effect of the collisions that were prevented. During the runs with no provided assistance, all participants experienced at least 1 collision, with an average of 4.77 (SD 2.58) for all participants, while when assistance was provided collisions significantly dropped to an average of 0.38 (SD 0.62); a decrease of 92%. Fig. 5.9 shows the difference in completion times between the two modes (with and without assistance) for each participant, while the number of collisions is shown in Fig. 5.10.

In terms of the average minimum distance to obstacles, the no-assistance runs had an average clearance of 31.8 (SD 3.3) cm, whereas the clearance for the runs with assistance had an average of 36.8 (SD 3.4) cm. Again, as with the joystick trials, the shared control system exhibited its ability to maintain a greater distance

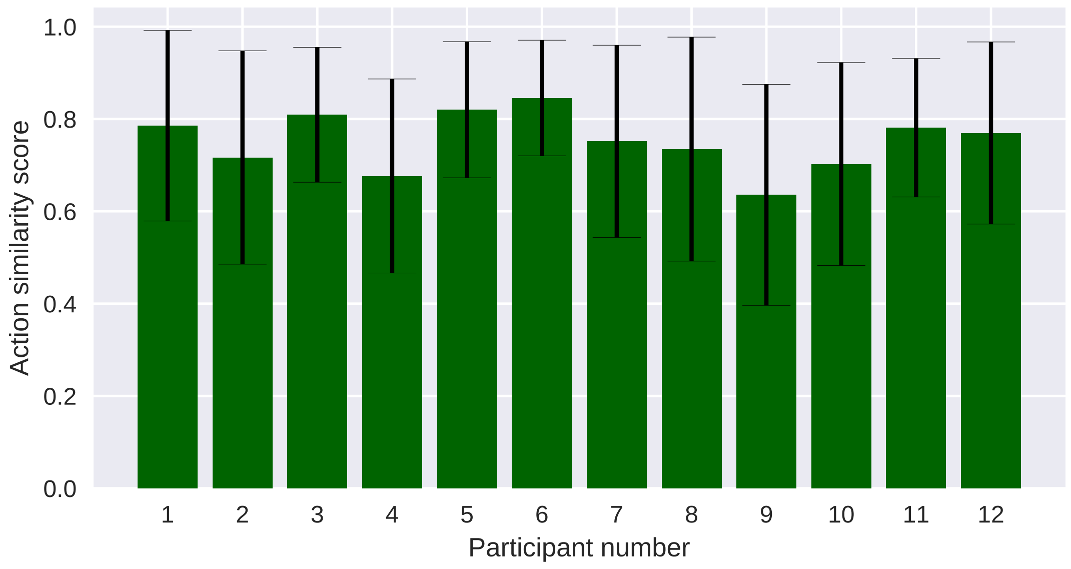


Figure 5.11: Averaged Action Similarity Score: Measures the closeness between the assistive system's output and the user's intended trajectories, averaged over the run.

from obstacles. Finally, in terms of action similarity between user and system, the average similarity had a score of 0.74 (SD 0.08), which is lower and with greater deviation compared to the respective result of the joystick trials (0.79 (SD 0.02)). This shows that the assistive system needed to deviate more from the users on average, which is an expected result since the driving performance of the participants with the head-control interface was significantly worse compared with the joystick interface. Fig. 5.11 shows the agreeability score of each participant's run with the shared control system.

Participant Feedback

At the end of each trial the participants were asked to indicate their degree of agreement with the statements in Fig. 4.16. The results show that the landscape of answers changes dramatically compared to the respective one for the joystick trials. Fig. 5.12 shows the difference in the degree of agreement between participants for each statement. Users, on average, felt that when they were provided with assistance the wheelchair was easier to manoeuvre, behaved more as they expected, had to concentrate less to drive it and they felt safer while performing the tasks.

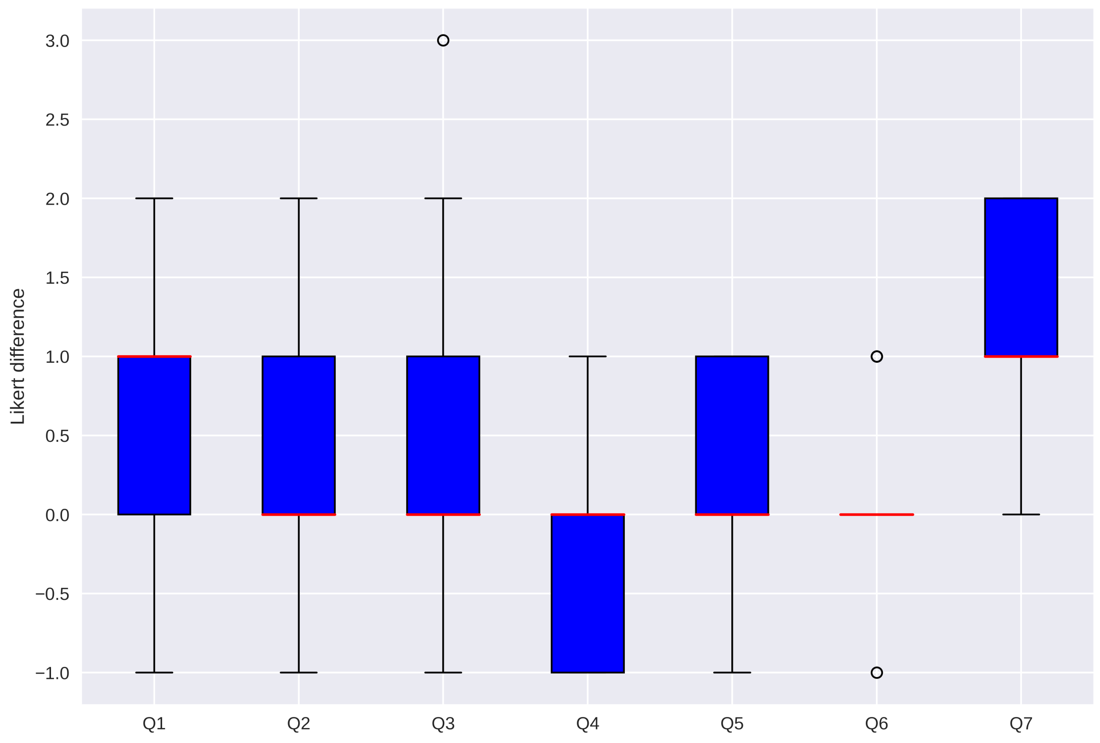


Figure 5.12: Agreement Degree with Fig. 4.16 Statements When Using the Shared Control System Compared to No Assistance.

A one-way ANOVA revealed that there was a statistically significant difference in mean Likert score between using assistance, or not, for the statements Q1 - "The wheelchair behaved as I expected" (p=0.035) and Q7 - "I felt safe while performing the task" (p=0.005).

Discussion

The majority of the collisions took place within the quite restricted space of 2x1.5 m (emulating an elevator) at the end of the enclosed circuit. After entering that space, in order to escape it, participants had to get the wheelchair to come to a halt and then proceed to carefully turn on the spot until the wheelchair was aligned with the narrow corridor. However, due to a number of factors, like the short reaction time that was required, inaccurate calibration of the head-control system, lack of experience with using it, and poor initial alignment of the wheelchair within the enclosed space, this maneuver proved to be the most challenging one for the majority of the participants. Users that were close to the walls while rotating, would often panic, executing sudden head movements (the system is best driven with short, smooth head movements) that resulted in either faster rotation or initiating forward movement of the robot, which in turn resulted in several collisions. In those cases, when assistance was provided, the system would stop the wheelchair before colliding, or slightly push it toward the direction of free space. This greatly helped participants escape the enclosed space and overcome the fear of colliding, allowing them to regain control of the wheelchair and complete the circuit.

An interesting behavior the shared control system exhibited, and which was noted by the majority of the participants, was the wheelchair aligning itself with the walls when turning to face a corridor. This was particularly evident in the narrowest corridor (1 m.), which several users found it challenging to traverse without assistance. While using the assistive system and turning to face that corridor, the wheelchair would position itself parallel to the corridor walls and then keep a straight trajectory even if a turn command was issued by the user.

These observations, regarding the shared control system, were also imprinted in both the objective metrics (*e.g.*, shorter completion times, fewer collisions), but also the subjective feedback provided by the participants (*i.e.*, easier maneuvering, felt safer). Those results are in contrast with the respective results when operating the wheelchair with the standard joystick interface, which the able-bodied users could handle easily and felt that the shared control system was more of an impediment rather than helpful. However, when using a less accurate and more erroneous interface, the shared control system provided much-needed assistance, as reflected by both the evaluation metrics and the participants' opinions.

5.6 Chapter Summary

This chapter introduced a novel vision-based interface for controlling a powered wheelchair through head movements. The method relies on using a simple web camera directed at the user's face, with the stream of frames being passed to a Convolutional Neural Network, that performs face detection in a fast and robust manner. The detected face is then passed to a facial landmark detector that locates the user's nose as a reference point, through which the pose (orientation) of the face is estimated. The estimated pose is finally translated to moving commands for the robotic wheelchair, which can be either discrete or continuous. The method was validated through real-world experiments, which showed that novice users were able to successfully use the head-control interface, executing a series of maneuvers to navigate a simple track (96% success rate). Furthermore, users reported that using the interface was not physically demanding, but was moderately mentally demanding, while they felt confident in being able to perform well with it.

After validating the head-control interface, the chapter presented more real-world experiments, in a different and more challenging scenario, where head-control was combined with the shared control system introduced in Ch. 4. The aim was to investigate the efficacy of shared control when used with a less accurate and more erroneous interface compared to the standard joystick. The results showed that when provided with assistance, users were able to complete the navigation task in a safer and faster manner, compared to having no assistance (92% fewer collisions and 15% decrease in completion times). Furthermore, their overall satisfaction with using the head-control interface also increased, with users reporting, with statistical significance, that the wheelchair behaved more as they expected, and that they felt safer while operating it.

CHAPTER 6

Conclusions

This concluding chapter provides a comprehensive overview of the thesis, emphasizing its significant contributions to the fields of shared control and assistive technologies for powered wheelchair navigation. It begins by discussing the main achievements, including the successful development of a shared control system and its promising real-world application. The chapter then delves into the key steps taken to transfer the shared control system to real-world scenarios. Following that, it explores future research directions, proposing potential enhancements to the shared control framework. Finally, the chapter concludes with insightful remarks, highlighting the research's impact and identifying related challenges for future exploration.

6.1 Discussion of Contributions

The principal aim of this thesis was to explore improvements in assistive technologies for robotic wheelchairs through machine learning methods, with a focus on the problem of shared control using the reinforcement learning framework. Existing methods for autonomous navigation in powered wheelchairs often rely on expensive sensors and have significant computational demands. Semi-autonomous navigation systems, including shared control systems, typically use linear blending or rely on conventional local planners and accurate prediction of user goals. In addition, current head-control interfaces for powered wheelchairs are often invasive and offer limited control freedom. Motivated by recent progress in artificial intelligence, we posed the following research questions:

- Can we develop an intelligent system capable of mapless autonomous navigation in complex indoor environments, relying solely on onboard sensor data and processing, while executing proficient wheelchair driving maneuvers?
- Is it feasible to design and implement an intelligent system that enhances user driving in various environments with different goals, utilizing sensor data and user input, and demonstrating effectiveness in both simulated and real-world environments?
- How can we enhance existing alternative control interfaces, specifically through the design and implementation of a non-invasive head-control system that can be easily deployed on a powered wheelchair?
- To what extent can the intelligent shared-control system improve the adequacy of alternative control interfaces, and how does the integration of shared control enhance the performance and usability of the head-control interface?

In Chapter 3, we investigated the first research question by training an autonomous agent using continuous driving actions and onboard sensor data. The agent excelled in challenging scenarios, surpassing human performance with significantly reduced collision rates (by approximately 85%) and faster completion times (around 30% faster). In comparison to conventional motion planners, which struggled to finish certain scenarios and took longer to complete others, the autonomous agent performed notably better. Additionally, the trained model demonstrated a significant computational advantage over conventional planners, boasting an average control frequency approximately 9-16 times higher. Furthermore, it eliminated the necessity for a map, showcasing its ability to adapt and navigate without relying on pre-existing environmental data, as opposed to traditional approaches. However, due to the lack of a map and the lack of long-term memory, the RL-based agent should be provided with a series of subsequent goals in order to reach long-term navigation goals. Taking those characteristics into consideration, a potential real-world application could involve realizing short-term goals of a wheelchair user in a demanding indoor environment for alleviating part of their daily burden (*e.g.*, enter/exit elevator, park under table, drive to bed, etc.). From a more general perspective, the proposed method can be used with any non-holonomic mobile robot to perform navigation with continuous actions in a safe and efficient manner.

Chapter 4 introduced a novel shared-control framework to address the second research question. Through the modification of the objective in maximum entropy reinforcement learning, the proposed system effectively aligns with user intentions while optimizing its own objectives. This leads to increased goal achievement and collision avoidance, despite the system's lack of awareness regarding the environment's goals or user policies. In simulation experiments, the shared-control system significantly improved the success rates of baseline and erroneous autonomous models by up to 58%. Moreover, the system's risk awareness aspect, which discounts compliance based on the risk associated with obstacles and user actions, was validated in simulations. The risk-awareness model consistently achieved over 5% higher success rates, approximately 8% increased clearance, and reduced crash rates. However, this came with a trade-off in completion times, reaching up to 10%. Compared to a conventional, artificial potential fields-based shared control method, the proposed system demonstrated notably higher success rates (up to 60%) and lower crash rates (up to 64%). Distinguished from other techniques like blending, the shared control method adopts a unique policy-based, efficient, and flexible framework that can be customized for various user models, tasks, and sensor arrangements.

To further investigate the second research question, real-world experiments were conducted using a physical robot. The shared-control system, trained in simulation and with a simulated user, successfully assisted human users in a real-world scenario, resulting in a 9.3% improvement in clearance and a complete elimination of collisions compared to using no assistance. These outcomes confirm the efficacy of the proposed shared-control method and provide compelling evidence for its practical applicability in real-world settings.

In Chapter 5, we addressed the third research question by introducing a vision-based non-invasive head-control interface. Unlike invasive methods found in the wheelchair market and literature, the proposed interface only required the installation of a normal web camera, making it easy to set up. The system utilized head pose estimation, powered by a state-of-the-art convolutional neural network, to enable wheelchair control with continuous turning angles. Real-world trials demonstrated that users could successfully execute maneuvers and navigate a track with high success rates (96%) and reasonable completion times, with increased trajectory lengths (10%) compared to the standard joystick interface. The resulting implementation was efficient, running on a compact computer like the Jetson Xavier, which can be easily integrated into an electric wheelchair and powered by its battery. A calibration process allowed for system customization based on individual ranges of motion. Overall, the findings suggest that the introduced head-control interface presents a promising and practical solution for quadriplegic patients, offering enhanced user experience and improved wheelchair control without the need for invasive methods.

In our investigation of the fourth research question, we integrated the head-control system with the trained assistive agent from Chapter 4. This combined system was evaluated with able-bodied human participants in a demanding task, involving the navigation of a narrow corridor, avoiding static obstacles, and maneuvering in a confined space resembling an elevator. When coupled with the assistive agent, the head-control system exhibited remarkable performance improvements compared to using the head-control system without assistance, with a substantial 92% reduction in collisions and a notable 16% increase in clearance. For the standard joystick interface, which is generally easier to use, the assistive system reduced collisions but resulted in longer completion times and decreased user satisfaction. In contrast, for the more challenging head-control interface, the assistive system achieved notable reductions in completion times (15%) and statistically significant improved user perception of control and safety. These results emphasized the effectiveness of our shared-control framework and highlighted the importance of integrating it with alternative control interfaces to enhance their adequacy and safety.

Through detailed evaluations and experimentation, this thesis has made significant contributions to the field of assistive technologies for robotic wheelchairs and advanced the state-of-the-art in shared-control. The achievements include the development of an intelligent system for mapless autonomous navigation, the introduction of a flexible shared-control framework, the design of a non-invasive head-control interface, and the enhancement of alternative control interfaces. These advancements pave the way for more efficient and user-friendly assistive technologies, ultimately improving the lives of individuals who rely on robotic wheelchairs. Incorporating these advancements into real-world applications has the potential to empower individuals with mobility impairments, granting them greater independence and opportunities for social inclusion. The outcomes of this research not only address the research questions but also contribute to the growing body of knowledge in the field of assistive robotics, offering practical and impactful solutions for those in need.

6.2 Key Steps in Transferring the Shared Control System to the Real World

One of the most challenging aspects of this work, which we also consider one of its major contributions, was transferring the shared control system to the real world. Despite not being a perfected commercially-ready system, the self-learning system trained solely in simulation and with a simulated user was able to provide assistance to human users driving the real robot. The process of transferring the shared control system to the real world involved the following key steps:

1. Use of a simulator with a high-fidelity physics engine and tuning of the physical parameters of the virtual robot to match the real robot.

2. Training of a high-entropy autonomous navigation policy to produce diverse

input similar to an ideal user following goals.

3. Corrupting the autonomous policy with artificial noise to simulate user disability and model erroneous input as a probability distribution.
4. Modifying the RL objective to minimize the KL divergence between the user's and the system's action distributions, in addition to maximizing rewards.
5. Designing an appropriate training task for the shared control agent to learn a generic assistive behavior.
6. Randomizing sensor measurements and robot-related parameters during training to bridge the simulation-real world gap.
7. Choosing an appropriate range sensor arrangement to provide sufficient information for collision avoidance while ensuring acceptable update rates.

6.3 Future Work

In this section, we outline the potential directions for future work to enhance the proposed shared control framework:

6.3.1 Adaptability to the User

The adaptability of the shared control system to individual users is an important area of further investigation. While the current work used generic noise models to simulate disabled user behavior, additional customization options can be explored to improve the system's effectiveness. These options include:

- Fitting different distributions to model specific disabilities, as long as an analytical solution exists for the KL divergence with the policy network's output distribution.

- Optimizing the target KL divergence according to user needs, taking into account specific user preferences and requirements.
- Tuning the risk estimation function based on user preferences to achieve an appropriate balance between cautiousness and efficiency.
- Fine-tuning the policy using real user driving data to further personalize the assistive behavior.

6.3.2 Adaptability to the Task

The shared control framework can be further adapted to specific tasks and objectives. While the thesis focused on a reward function encouraging clearance from obstacles and collision avoidance, customization of the reward function can yield more specialized assistive policies. Additional objectives or considerations can be incorporated, such as compliance with social norms, producing smoother trajectories, or accounting for specific disabilities.

Furthermore, the system's adaptability to the task can be improved by considering the availability of user goals or intentions. By augmenting the system's observations and rewards to incorporate this information, the shared control system can become a more effective collaborator in helping users reach their goals.

6.3.3 Adaptability to the Sensory Input

Providing sufficient information to the shared control agent is crucial for effective learning and decision-making. While the proposed ultrasonic sensor arrangement proved effective in this work, alternative or additional sensors can be explored to enhance the system's perception capabilities. Sensor fusion techniques can be utilized to combine the advantages of different sensors, such as ultrasonic and Time-of-Flight (ToF) sensors, to provide richer and more robust environmental information.

For example, augmenting the proposed setup with ToF sensors can enable obstacle detection at longer distances and with better accuracy, while the sonars can provide more useful information for shorter distances and obstacles at different height levels. This kind of sensor fusion setup would be more robust towards different obstacle materials and light conditions.

It is worth noting that the use of cameras can provide very rich information about the environment, but it comes with trade-offs such as longer training times, more powerful hardware requirements, and challenges in transferring the system to the real world. Careful consideration should be given to the cost and feasibility of using camera-based systems in practical applications.

6.4 Conclusions and Final Remarks

In conclusion, this work has successfully developed a flexible, self-learning framework for sharing control between a wheelchair user and an intelligent system, with minimal assumptions about the environment or the user. The framework's adaptability to the user, task, and sensory input provides ample room for customization and optimization based on individual needs and preferences. Moreover, the presented framework goes beyond robotic wheelchairs and holds potential for various human-robot collaboration instances, such as piloting drones or performing robotic-assisted surgeries. As smart wheelchairs serve as excellent test benches, they offer a unique opportunity to explore and advance novel robotic technologies, including control algorithms, sensors, and human-robot interaction paradigms.

However, several significant barriers still need to be addressed before widespread adoption of smart wheelchair technologies, as presented in this thesis, can be achieved for real-world use. Autonomous or semi-autonomous navigation methods must consider social conventions and adapt to prevent potentially embarrassing or dangerous situations, while effectively accounting for the movements of people and objects in dynamic environments. Another critical focus area is the improvement of sensing systems to provide cost-effective solutions that operate reliably across diverse environmental conditions and surface materials.

Deep Reinforcement Learning, the primary framework utilized in this thesis, is still in its early stages and faces several challenges that demand further investigation for broader applicability in real-world scenarios. Sample efficiency in high-dimensional state and action spaces remains a critical concern. Additionally, addressing robust reward assignment for multi-dimensional or hierarchical tasks, handling partial observability, and ensuring the preservation of safety constraints during exploratory learning phases or system operation are important areas for future exploration.

In light of these challenges, it is evident that reinforcement and machine learning techniques are poised to play an increasingly vital role in various aspects of our daily lives, particularly in the crucial field of healthcare. This work makes a compelling case for the application of deep RL in the real world and contributes to the advancement of assistive technologies, ultimately aiming to improve the quality of life and promote the independence of individuals who require such technologies. By pushing the boundaries of smart wheelchair technologies and reinforcement learning, this research drives the realization of a future where technology empowers individuals and enhances their overall well-being.

Bibliography

[1] J. Kavita and S. Onkar, "Elderly and Disabled Assistive Devices Market Overview 2026," Tech. Rep., 2019. [Online]. Available: https://www.alliedmarketresearch.com/elderly-and-disabled-assistive-devices-market

[2] S. Ugalmugale and R. Swain, "Powered Wheelchair Market," Tech. Rep., 2022. [Online]. Available: https://www.gminsights.com/industry-analysis/powered-wheelchair-market

[3] J. Leaman and H. M. La, "A comprehensive review of smart wheelchairs: past, present, and future," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 4, pp. 486–499, 2017.

[4] S. Udupa, V. R. Kamat, and C. C. Menassa, "Shared autonomy in assistive mobile robots: a review," *Disabil. Rehabil. Assist. Technol.*, vol. 0, no. 0, pp. 1–22, 2021. [Online]. Available: https://doi.org/10.1080/17483107.2021.1928778

[5] D. A. Abbink, S. Member, T. Carlson, M. Mulder, J. C. F. De, F. Aminravan, T. L. Gibo, and E. R. Boer, "A Topology of Shared Control Systems – Finding Common Ground in Diversity," *IEEE Trans. Human-Machine Syst.*, vol. 48, no. 5, pp. 509–525, 2018.

[6] K. Hauser, "Recognition, prediction, and planning for assisted teleoperation of freeform tasks," *Robot. Sci. Syst.*, vol. 8, pp. 121–128, 2013.

[7] S. Javdani, S. S. Srinivasa, and J. A. Bagnell, "Shared autonomy via hindsight optimization," *Robot. Sci. Syst.*, vol. 11, 2015.

[8] S. Nikolaidis, Y. X. Zhu, D. Hsu, and S. Srinivasa, "Human-Robot Mutual Adaptation in Shared Autonomy," *ACM/IEEE Int. Conf. Human-Robot Interact.*, vol. Part F1271, pp. 294–302, 2017.

[9] P. Trautman, "Assistive Planning in Complex, Dynamic Environments: A Probabilistic Approach," *Proc. - 2015 IEEE Int. Conf. Syst. Man, Cybern. SMC 2015*, no. June 2015, pp. 3072–3078, 2016.

[10] I. Mougharbel, R. El-Hajj, H. Ghamlouch, and E. Monacelli, "Comparative study on different adaptation approaches concerning a sip and puff controller for a powered wheelchair," *Proc. 2013 Sci. Inf. Conf. SAI 2013*, pp. 597–603, 2013.

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[12] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.

[13] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[14] R. E. Cowan, B. J. Fregly, M. L. Boninger, L. Chan, M. M. Rodgers, and D. J. Reinkensmeyer, “Recent trends in assistive technology for mobility,” *Journal of neuroengineering and rehabilitation*, vol. 9, no. 1, pp. 1–8, 2012.

[15] J. Diebel *et al.*, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[16] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part I,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, 2006.

[17] S. Thrun, B. Wolfram, and F. Dieter, *Probabilistic Robotics.* MIT Press, 2006, vol. 21, no. 3.

[18] D. Fox, W. Burgard, and S. Thrun, “Markov localization for mobile robots in dynamic environments,” *J. Artif. Intell. Res.*, vol. 11, pp. 391–427, 1999.

[19] J. J. Leonard and H. F. Durrant-Whyte, “Mobile robot localization by tracking geometric beacons,” *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 376–382, 1991.

[20] S. J. Julier and J. K. Uhlmann, “New extension of the Kalman filter to nonlinear systems,” in *Signal Process. Sens. fusion, target Recognit. VI*, vol. 3068, 1997, pp. 182–193.

[21] P. Jensfelt and S. Kristensen, “Active global localization for a mobile robot using multiple hypothesis tracking,” *IEEE Trans. Robot. Autom.*, vol. 17, no. 5, pp. 748–760, 2001.

[22] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” in *Proc. 1999 IEEE Int. Conf. Robot. Autom. (Cat. No. 99CH36288C)*, vol. 2, 1999, pp. 1322–1328.

[23] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proc. ICNN’95-international Conf. neural networks*, vol. 4, 1995, pp. 1942–1948.

[24] E. W. Dijkstra and Others, “A note on two problems in connexion with graphs,” *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

[25] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[26] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," in *Robot. Res.* Springer, 2007, pp. 239–253.

[27] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *J. Artif. Intell. Res.*, vol. 39, pp. 533–579, 2010.

[28] D. D. Harabor, A. Grastien, D. Öz, and V. Aksakalli, "Optimal any-angle pathfinding in practice," *J. Artif. Intell. Res.*, vol. 56, pp. 89–118, 2016.

[29] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[30] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[31] K.-F. Man, K.-S. Tang, and S. Kwong, "Genetic algorithms: concepts and applications [in engineering design]," *IEEE Trans. Ind. Electron.*, vol. 43, no. 5, pp. 519–534, 1996.

[32] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, 2006.

[33] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE Int. Conf. Robot. Autom.*, vol. 2, 1985, pp. 500–505.

[34] J. Borenstein, Y. Koren, and Others, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, 1991.

[35] R. Raja, A. Dutta, and K. S. Venkatesh, "New potential field method for rough terrain path planning using genetic algorithm for a 6-wheel rover," *Rob. Auton. Syst.*, vol. 72, pp. 295–306, 2015.

[36] Z. Zhou, J. Wang, Z. Zhu, D. Yang, and J. Wu, "Tangent navigated robot path planning strategy using particle swarm optimized artificial potential field," *Optik (Stuttg).*, vol. 158, pp. 639–651, 2018.

[37] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," *IEEE Trans. Robot. Autom.*, vol. 13, no. 6, pp. 814–822, 1997.

[38] I. Kamon, E. Rimon, and E. Rivlin, "Tangentbug: A range-sensor-based navigation algorithm," *Int. J. Rob. Res.*, vol. 17, no. 9, pp. 934–953, 1998.

[39] R. A. Langer, L. S. Coelho, and G. H. C. Oliveira, "K-Bug, a new bug approach for mobile robot's path planning," in *2007 IEEE Int. Conf. Control Appl.*, 2007, pp. 403–408.

[40] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[41] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *[1993] Proc. IEEE Int. Conf. Robot. Autom.*, 1993, pp. 802–807.

[42] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, “Efficient trajectory optimization using a sparse model,” in *2013 Eur. Conf. Mob. Robot.*, 2013, pp. 138–143.

[43] J. Yi, X. Zhang, Z. Ning, and Q. Huang, “Intelligent robot obstacle avoidance system based on fuzzy control,” in *2009 First Int. Conf. Inf. Sci. Eng.*, 2009, pp. 3812–3815.

[44] H. Seraji and A. Howard, “Behavior-based robot navigation on challenging terrain: A fuzzy logic approach,” *IEEE Trans. Robot. Autom.*, vol. 18, no. 3, pp. 308–321, 2002.

[45] P. G. Zavlangas and S. G. Tzafestas, “Motion control for mobile robot obstacle avoidance and navigation: a fuzzy logic-based approach,” *Syst. Anal. Model. Simul.*, vol. 43, no. 12, pp. 1625–1637, 2003.

[46] Y. Yan and Y. Li, “Mobile robot autonomous path planning based on fuzzy logic and filter smoothing in dynamic environment,” in *2016 12th World Congr. Intell. Control Autom.*, 2016, pp. 1479–1484.

[47] I. Engedy and G. Horváth, “Artificial neural network based local motion planning of a wheeled mobile robot,” in *2010 11th Int. Symp. Comput. Intell. Informatics*, 2010, pp. 213–218.

[48] L. Tai, S. Li, and M. Liu, “A deep-network solution towards model-less obstacle avoidance,” in *2016 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2016, pp. 2759–2764.

[49] A. Zhu and S. X. Yang, “An adaptive neuro-fuzzy controller for robot navigation,” in *Recent Adv. Intell. Control Syst.* Springer, 2009, pp. 277–307.

[50] P. K. Mohanty and D. R. Parhi, “A new intelligent motion planning for mobile robot navigation using multiple adaptive neuro-fuzzy inference system,” *Appl. Math. Inf. Sci.*, vol. 8, no. 5, p. 2527, 2014.

[51] M. M. Joshi and M. A. Zaveri, “Reactive navigation of autonomous mobile robot using neuro-fuzzy system,” *Int. J. Robot. Autom.*, vol. 2, no. 3, p. 128, 2011.

[52] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “DeepDriving: Learning affordance for direct perception in autonomous driving,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, no. Figure 1, pp. 2722–2730, 2015.

[53] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, “From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1527–1533, 2017.

[54] A. Hussein, E. Elyan, M. M. Gaber, and C. Jayne, “Deep imitation learning for 3D navigation tasks,” *Neural Comput. Appl.*, vol. 29, no. 7, pp. 389–404, 2018. [Online]. Available: https://doi.org/10.1007/s00521-017-3241-z

[55] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, “Socially Compliant Mobile Robot Navigation via Inverse Reinforcement Learning,” *Int. J. Rob. Res.*,

p. 0278364915619772, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/04/0278364915619772

[56] B. Kim and J. Pineau, "Socially Adaptive Path Planning in Human Environments Using Inverse Reinforcement Learning," *Int. J. Soc. Robot.*, vol. 8, no. 1, pp. 51–66, 2016.

[57] X. Zhou, Y. Gao, and L. Guan, "Towards goal-directed navigation through combining learning based global and local planners," *Sensors (Switzerland)*, vol. 19, no. 1, 2019.

[58] S. Lange, M. Riedmiller, and A. Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," *Proc. Int. Jt. Conf. Neural Networks*, 2012.

[59] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. - IEEE Int. Conf. Robot. Autom.*, 2017, pp. 3357–3364.

[60] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to navigate in complex environments," *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, 2019.

[61] K. Wu, M. A. Esfahani, S. Yuan, and H. Wang, "Learn to Steer through Deep Reinforcement Learning," *Sensors (Basel).*, vol. 18, no. 11, 2018.

[62] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2017-Septe, pp. 2371–2378, 2017.

[63] J. Zeng, R. Ju, L. Qin, Y. Hu, Q. Yin, and C. Hu, "Navigation in unknown dynamic environments based on deep reinforcement learning," *Sensors (Switzerland)*, vol. 19, no. 18, 2019.

[64] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[65] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning.* PMLR, 2016, pp. 1928–1937.

[66] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2017-Septe, 2017.

[67] E. Marchesini and A. Farinelli, "Discrete Deep Reinforcement Learning for Mapless Navigation," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 10 688–10 694, 2020.

[68] A. Francis, A. Faust, H. T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T. W. E. Lee, "Long-Range Indoor Navigation with PRM-RL," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1115–1134, 2020.

[69] H. S. Grewal, N. Thotappala Jayaprakash, A. Matthews, C. Shrivastav, and K. George, "PCL-Based Autonomous Wheelchair Navigation in Unmapped Indoor Environments," *2018 9th IEEE Annu. Ubiquitous Comput. Electron. Mob. Commun. Conf. UEMCON 2018*, pp. 291–296, 2018.

[70] R. Li, M. A. Oskoei, K. D. McDonald-Maier, and H. Hu, "ROS based multi-sensor navigation of intelligent wheelchair," *Proc. - 2013 4th Int. Conf. Emerg. Secur. Technol. EST 2013*, pp. 83–88, 2013.

[71] D. Sinyukov, R. Desmond, M. Dickerman, J. Fleming, J. Schaufeld, and T. Padir, "Multi-modal control framework for a semi-autonomous wheelchair using modular sensor designs," *Intell. Serv. Robot.*, vol. 7, no. 3, pp. 145–155, 2014.

[72] U. Yayan, B. Akar, F. Inan, and A. Yazici, "Development of indoor navigation software for intelligent wheelchair," *INISTA 2014 - IEEE Int. Symp. Innov. Intell. Syst. Appl. Proc.*, pp. 325–329, 2014.

[73] C. Gao, M. Sands, and J. Spletzer, "Towards Autonomous Wheelchair Systems in Urban Environments," *Springer Tracts Adv. Robot.*, vol. 62, pp. 13–23, 2009.

[74] R. Alkhatib, A. Swaidan, J. MarzoGBR, M. Sabbah, S. Berjaoui, and M. O. Diab, "Smart autonomous wheelchair," *BioSMART 2019 - Proc. 3rd Int. Conf. Bio-Engineering Smart Technol.*, 2019.

[75] Y. Morales, N. Kallakuri, K. Shinozawa, T. Miyashita, and N. Hagita, "Human-comfortable navigation for an autonomous robotic wheelchair," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 2737–2743, 2013.

[76] L. Devigne, F. Pasteau, M. Babel, V. K. Narayanan, S. Guegan, and P. Gallien, "Design of a haptic guidance solution for assisted power wheelchair navigation," in *2018 IEEE Int. Conf. Syst. Man, Cybern.* IEEE, 2018, pp. 3231–3236.

[77] D. A. Abbink, M. Mulder, and E. R. Boer, "Haptic shared control: smoothly shifting control authority?" *Cogn. Technol. \& Work*, vol. 14, no. 1, pp. 19–28, 2012.

[78] F. Mars, M. Deroo, and J.-M. Hoc, "Analysis of human-machine cooperation when driving with different degrees of haptic shared control," *IEEE Trans. Haptics*, vol. 7, no. 3, pp. 324–333, 2014.

[79] T. Carlson and Y. Demiris, "Collaborative control for a robotic wheelchair: Evaluation of performance, attention, and workload," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 42, no. 3, pp. 876–888, 2012.

[80] A. D. Dragan and S. S. Srinivasa, "A policy-blending formalism for shared control," *Int. J. Rob. Res.*, vol. 32, no. 7, pp. 790–805, 2013.

[81] K. Muelling, A. Venkatraman, J.-S. Valois, J. Downey, J. Weiss, S. Javdani, M. Hebert, A. B. Schwartz, J. L. Collinger, and J. A. Bagnell, "Autonomy infused teleoperation with application to bci manipulation," *arXiv preprint arXiv:1503.05451*, 2015.

[82] S. Jain and B. Argall, "Probabilistic Human Intent Recognition for Shared Autonomy in Assistive Robotics," *ACM Trans. Human-Robot Interact.*, vol. 9, no. 1, pp. 1–23, jan 2020.

[83] C. Ezeh, P. Trautman, L. Devigne, V. Bureau, M. Babel, and T. Carlson, "Probabilistic vs linear blending approaches to shared control for wheelchair driving," *IEEE Int. Conf. Rehabil. Robot.*, pp. 835–840, 2017.

[84] D. A. Sanders, "Using Self-Reliance Factors to Decide How to Share Control Between Human Powered Wheelchair Drivers and Ultrasonic Sensors." *IEEE Trans. neural Syst. Rehabil. Eng. a Publ. IEEE Eng. Med. Biol. Soc.*, vol. 25, no. 8, pp. 1221–1229, aug 2017.

[85] D. Aarno and D. Kragic, "Motion intention recognition in robot assisted applications," *Rob. Auton. Syst.*, vol. 56, no. 8, pp. 692–705, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889007001704

[86] W. Yu, R. Alqasemi, R. Dubey, and N. Pernalete, "Telemanipulation Assistance Based on Motion Intention Recognition," in *Proc. 2005 IEEE Int. Conf. Robot. Autom.*, 2005, pp. 1121–1126.

[87] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[88] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," *arXiv preprint arXiv:1206.4617*, 2012.

[89] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *Adv. Neural Inf. Process. Syst.*, vol. 3, 2014.

[90] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[91] C. Finn, S. Levine, and P. Abbeel, "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization," *Proc. 33Rd Int. Conf. Mach. Learn.*, 2016.

[92] H. S. Koppula and A. Saxena, "Anticipating human activities for reactive robotic response." in *IROS*, 2013, p. 2071.

[93] E. Demeester, A. Hüntemann, D. Vanhooydonck, G. Vanacker, A. Degeest, H. Van Brussel, and M. Nuttin, "Bayesian estimation of wheelchair driver intents: Modeling intents as geometric paths tracked by the driver," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 5775–5780, 2006.

[94] Q. Li, W. Chen, and J. Wang, "Dynamic shared control for human-wheelchair cooperation," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 4278–4283, 2011.

[95] D. Gopinath, S. Jain, and B. D. Argall, "Human-in-the-loop optimization of shared autonomy in assistive robotics," *IEEE Robot. Autom. Lett.*, vol. 2, no. 1, pp. 247–254, 2017.

[96] C. Urdiales, M. Fzed-Carmona, G. Peinado, and F. Sandoval, "Metrics and benchmarking for assisted wheelchair navigation," *IEEE Int. Conf. Rehabil. Robot.*, no. June, p. 51, 2013.

[97] M. Geravand, C. Werner, K. Hauer, and A. Peer, "An Integrated Decision Making Approach for Adaptive Shared Control of Mobility Assistance Robots," *Int. J. Soc. Robot.*, vol. 8, no. 5, pp. 631–648, 2016.

[98] C. Ezeh, P. Trautman, C. Holloway, and T. Carlson, "Comparing shared control approaches for alternative interfaces: A wheelchair simulator experiment," *2017 IEEE Int. Conf. Syst. Man, Cybern. SMC 2017*, vol. 2017-Janua, pp. 93–98, 2017.

[99] S. Javdani, H. Admoni, S. Pellegrinelli, S. S. Srinivasa, and J. A. Bagnell, "Shared autonomy via hindsight optimization for teleoperation and teaming," *Int. J. Rob. Res.*, pp. 1–26, 2018.

[100] W. Xu, J. Huang, Y. Wang, C. Tao, and L. Cheng, "Reinforcement learning-based shared control for walking-aid robot and its experimental verification," *Adv. Robot.*, vol. 29, no. 22, pp. 1463–1481, 2015. [Online]. Available: http://dx.doi.org/10.1080/01691864.2015.1070748

[101] L. Xi and M. Shino, "Shared Control Design Methodologies of an Electric Wheelchair for Individuals with Severe Disabilities using Reinforcement Learning," *J. Adv. Simul. Sci. Eng.*, vol. 7, no. 2, pp. 300–319, 2020.

[102] C. Tian, S. Shaik, and Y. Wang, "Deep reinforcement learning for shared control of mobile robots," *IET Cyber-systems Robot.*, vol. 3, no. 4, pp. 315–330, 2021.

[103] S. Reddy, A. D. Dragan, and S. Levine, "Shared autonomy via deep reinforcement learning," *arXiv preprint arXiv:1802.01744*, 2018.

[104] S. You, Y. Kang, Y.-B. Zhao, and Q. Zhang, "Adaptive arbitration for minimal intervention shared control via deep reinforcement learning," in *2021 China Automation Congress (CAC)*. IEEE, 2021, pp. 743–748.

[105] S. P. Levine, D. A. Bell, L. A. Jaros, R. C. Simpson, Y. Koren, and J. Borenstein, "The NavChair Assistive Wheelchair Navigation System," *IEEE Trans. Rehabil. Eng.*, vol. 7, no. 4, pp. 443–451, 1999.

[106] T. Roefer and A. Lankenau, "Ensuring safe obstacle avoidance in a shared-control system," *IEEE Symp. Emerg. Technol. Fact. Autom. ETFA*, vol. 2, pp. 1405–1414, 1999.

[107] G. Pires, R. Aradjo, U. Nunes, and A. T. de Almeida, "RobChair - a powered wheelchair using a behaviour-based navigation," *Int. Work. Adv. Motion Control. AMC*, pp. 536–541, 1998.

[108] D. Vanhooydonck, E. Demeester, A. Hüntemann, J. Philips, G. Vanacker, H. Van Brussel, and M. Nuttin, "Adaptable navigational assistance for intelligent wheelchairs by means of an implicit personalized user model," *Rob. Auton. Syst.*, vol. 58, no. 8, pp. 963–977, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2010.04.002

[109] A. Erdogan and B. D. Argall, "The effect of robotic wheelchair control paradigm and interface on user performance, effort and preference: An experimental assessment," *Rob. Auton. Syst.*, vol. 94, pp. 282–297, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2017.04.013

[110] B. D. Argall, "Modular and adaptive wheelchair automation," *Springer Tracts Adv. Robot.*, vol. 109, pp. 835–848, 2016.

[111] C. Urdiales, J. M. Peula, M. Fdez-Carmona, C. Barrué, E. J. Pérez, I. Sánchez-Tato, J. C. Del Toro, F. Galluppi, U. Cortés, R. Annichiaricco, C. Caltagirone, and F. Sandoval, "A new multi-criteria optimization strategy for shared control in wheelchair assisted navigation," *Auton. Robots*, vol. 30, no. 2, pp. 179–197, 2011.

[112] M. Gillham and G. Howells, "A Dynamic Localized Adjustable Force Field Method for Real-Time Assistive Non-Holonomic Mobile Robotics," *Int. J. Adv. Robot. Syst.*, vol. 12, no. 10, 2015.

[113] S. Chatzidimitriadis, P. Oprea, M. Gillham, and K. Sirlantzis, "Evaluation of 3D obstacle avoidance algorithm for smart powered wheelchairs," *Proc. - 2017 7th Int. Conf. Emerg. Secur. Technol. EST 2017*, pp. 157–162, 2017.

[114] M. Ghorbel, J. Pineau, R. Gourdeau, S. Javdani, and S. Srinivasa, "A Decision-Theoretic Approach for the Collaborative Control of a Smart Wheelchair," *Int. J. Soc. Robot.*, vol. 10, no. 1, pp. 131–145, 2018.

[115] C. S. Teodorescu and T. Carlson, "Assistme: Policy iteration for the longitudinal control of a non-holonomic vehicle," *arXiv preprint arXiv:2202.02569*, 2022.

[116] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning.* MIT press Cambridge, 1998, vol. 135.

[117] A. G. Barto, R. S. Sutton, and C. Watkins, *Learning and sequential decision making.* University of Massachusetts Amherst, MA, 1989.

[118] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming.* Athena Scientific, 1996.

[119] H. Seijen and R. Sutton, "True online TD (lambda)," in *Int. Conf. Mach. Learn.* PMLR, 2014, pp. 692–700.

[120] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[121] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv Prepr. arXiv1511.05952*, 2015.

[122] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

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

[124] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 3, pp. 2171–2186, 2017.

[125] S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," *arXiv preprint arXiv:1805.00909*, 2018.

[126] J. Schulman, X. Chen, and P. Abbeel, "Equivalence between policy gradients and soft q-learning," *arXiv preprint arXiv:1704.06440*, 2017.

[127] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Work. Open Source Softw.*, vol. 3, no. Figure 1, 2009.

[128] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, vol. 3, pp. 2149–2154, 2004.

[129] M. Derry and B. Argall, "Automated doorway detection for assistive shared-control wheelchairs," in *2013 IEEE Int. Conf. Robot. Autom.* IEEE, 2013, pp. 1254–1259.

[130] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[131] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and Others, "Soft actor-critic algorithms and applications," *arXiv Prepr. arXiv1812.05905*, 2018.

[132] R. L. Kirby, J. Swuste, D. J. Dupuis, D. A. MacLeod, and R. Monroe, "The Wheelchair Skills Test: A pilot study of a new outcome measure," *Arch. Phys. Med. Rehabil.*, vol. 83, no. 1, pp. 10–18, 2002.

[133] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.

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

[135] S. Kullback and R. A. Leibler, “On Information and Sufficiency,” *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[136] K. Pertsch, Y. Lee, and J. Lim, “Accelerating reinforcement learning with learned skill priors,” in *Conference on robot learning.* PMLR, 2021, pp. 188–204.

[137] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ Int. Conf. Intell. Robot. Syst.* IEEE, 2017, pp. 23–30.

[138] S. P. Parikh, V. Grassi, V. Kumar, and J. J. Okamoto, “Usability study of a control framework for an intelligent wheelchair,” in *Proc. 2005 IEEE Int. Conf. Robot. Autom.* IEEE, 2005, pp. 4745–4750.

[139] National Spinal Cord Injury Statistical Center, “Spinal cord injury facts and figures at a glance.” *J. Spinal Cord Med.*, vol. 33, no. 4, pp. 439–440, 2016.

[140] S. Kumar, N. Dheeraj, and S. Kumar, “Design and Development of Head Motion Controlled Wheelchair,” *Int. J. Adv. Eng. Technol.*, vol. 8, no. 5, pp. 816–822, 2015.

[141] Y. L. Chen, S. C. Chen, W. L. Chen, and J. F. Lin, “A head oriented wheelchair for people with disabilities,” *Disabil. Rehabil.*, vol. 25, no. 6, pp. 249–253, 2003.

[142] S. M. Bafti, S. Chatzidimitriadis, and K. Sirlantzis, “Cross-Domain Multitask Model for Head Detection and Facial Attribute Estimation,” *IEEE Access*, vol. 10, pp. 54 703–54 712, 2022.

[143] F. Abedan Kondori, S. Yousefi, L. Liu, and H. Li, “Head operated electric wheelchair,” *Proc. IEEE Southwest Symp. Image Anal. Interpret.*, pp. 53–56, 2014.

[144] J. Kim, H. Park, J. Bruce, D. Rowles, J. Holbrook, B. Nardone, D. P. West, A. Laumann, E. J. Roth, and M. Ghovanloo, “Assessment of the tongue-drive system using a computer, a smartphone, and a powered-wheelchair by people with tetraplegia,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 24, no. 1, pp. 68–78, 2016.

[145] J. M. Ford and S. J. Sheredos, “Ultrasonic head controller for powered wheelchairs,” *J. Rehabil. Res. Dev.*, vol. 32, no. 3, pp. 280–284, 1995.

[146] D. J. Kupetz, S. A. Wentzell, and B. F. BuSha, “Head motion controlled power wheelchair,” *Proc. 2010 IEEE 36th Annu. Northeast Bioeng. Conf. NEBEC 2010*, pp. 2–3, 2010.

[147] M. Jain, S. Puri, and S. Unishree, “Eyeball Motion Controlled Wheelchair Using IR Sensors,” *Int. J. Comput. Inf. Eng.*, vol. 9, no. 4, pp. 1012–1015, 2015.

[148] J. W. Machangpa and T. S. Chingtham, “Head Gesture Controlled Wheelchair for Quadriplegic Patients,” *Procedia Comput. Sci.*, vol. 132, no. Iccids, pp. 342–351, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.05.189

[149] S. Prasad, D. Sakpal, P. Rakhe, and S. Rawool, “Head-motion controlled wheelchair,” *RTEICT 2017 - 2nd IEEE Int. Conf. Recent Trends Electron. Inf. Commun. Technol. Proc.*, vol. 2018-Janua, pp. 1636–1640, 2017.

[150] M. Bureau, J. M. Azkoitia, G. Ezmendi, I. Manterola, H. Zabaleta, M. Perez, and J. Medina, “Non-invasive, wireless and universal interface for the control of peripheral devices by means of head movements,” *2007 IEEE 10th Int. Conf. Rehabil. Robot. ICORR'07*, vol. 00, no. c, pp. 124–131, 2007.

[151] S. H. Chen, Y. L. Chen, T. S. Kuo, C. Y. Chu, and C. N. Hung, “M3S-based electrical wheelchair with head-controlled device,” *Annu. Int. Conf. IEEE Eng. Med. Biol. - Proc.*, pp. 4917–4920, 2006.

[152] E. J. Rechy-Ramirez and H. Hu, “An electric wheelchair controlled by head movements and facial expressions: uni-modal, bi-modal, and fuzzy bi-modal modes,” in *Handb. Res. Investig. Artif. Life Res. Dev.* IGI Global, 2018, pp. 1–30.

[153] Z. T. Al-qaysi, B. B. Zaidan, A. A. Zaidan, and M. S. Suzani, “A review of disability EEG based wheelchair control system: Coherent taxonomy, open challenges and recommendations,” *Comput. Methods Programs Biomed.*, vol. 164, pp. 221–237, 2018. [Online]. Available: https://doi.org/10.1016/j.cmpb.2018.06.012

[154] I. A. Mirza, A. Tripathy, S. Chopra, M. D'Sa, K. Rajagopalan, A. D'Souza, and N. Sharma, “Mind-controlled wheelchair using an EEG headset and arduino microcontroller,” in *2015 Int. Conf. Technol. Sustain. Dev.*, 2015, pp. 1–5.

[155] F. Ben Taher, N. Ben Amor, and M. Jallouli, “A multimodal wheelchair control system based on EEG signals and Eye tracking fusion,” *INISTA 2015 - 2015 Int. Symp. Innov. Intell. Syst. Appl. Proc.*, 2015.

[156] P. S. Gajwani and S. A. Chhabria, “Eye motion tracking for wheelchair control,” *Int. J. Inf. Technol. Knowl. Manag.*, vol. 2, no. 2, pp. 185–187, 2010.

[157] C. S. Lin, C. W. Ho, W. C. Chen, C. C. Chiu, and M. S. Yeh, “Powered wheelchair controlled by eye-tracking system,” *Opt. Appl.*, vol. 36, no. 2-3, pp. 401–412, 2006.

[158] N. Wanluk, S. Visitsattapongse, A. Juhong, and C. Pintavirooj, “Smart wheelchair based on eye tracking,” *BMEiCON 2016 - 9th Biomed. Eng. Int. Conf.*, 2017.

[159] H. H. Hu, P. Jia, T. Lu, and K. Yuan, “Head gesture recognition for hands-free control of an intelligent wheelchair,” *Ind. Rob.*, vol. 34, no. 1, pp. 60–68, 2007.

[160] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, 2001.

[161] R. Solea, A. Margarit, D. Cernega, and A. Serbencu, “Head movement control of powered wheelchair,” *2019 23rd Int. Conf. Syst. Theory, Control Comput. ICSTCC 2019 - Proc.*, pp. 632–637, 2019.

[162] J. S. Ju, Y. Shin, and E. Y. Kim, “Vision based interface system for hands free control of an intelligent wheelchair,” *J. Neuroeng. Rehabil.*, vol. 6, no. 1, pp. 1–17, 2009.

[163] L. M. Bergasa, M. Mazo, A. Gardel, R. Barea, and L. Boquete, “Commands generation by face movements applied to the guidance of a wheelchair for handicapped people,” *Proc. - Int. Conf. Pattern Recognit.*, vol. 15, no. 4, pp. 660–663, 2000.

[164] E. Perez, C. Soria, O. Nasisi, T. F. Bastos, and V. Mut, “Robotic wheelchair controlled through a vision-based interface,” *Robotica*, vol. 30, no. 5, pp. 691–708, 2012.

[165] A. Halawani, S. ur Réhman, H. Li, and A. Anani, “Active vision for controlling an electric wheelchair,” *Intell. Serv. Robot.*, vol. 5, no. 2, pp. 89–98, 2012.

[166] H. G. Yashoda, A. M. Piumal, P. G. Polgahapitiya, M. M. Mubeen, M. A. Muthugala, and A. G. Jayasekara, “Design and development of a smart wheelchair with multiple control interfaces,” *MERCon 2018 - 4th Int. Multidiscip. Moratuwa Eng. Res. Conf.*, pp. 324–329, 2018.

[167] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016.

[168] Q. C. Mao, H. M. Sun, Y. B. Liu, and R. S. Jia, “Mini-YOLOv3: Real-Time Object Detector for Embedded Applications,” *IEEE Access*, vol. 7, pp. 133 529–133 538, 2019.

[169] M. Bjelonic, “YOLO ROS: Real-Time Object Detection for ROS.” [Online]. Available: https://github.com/leggedrobotics/darknet{_}ros

[170] V. Kazemi and J. Sullivan, “One millisecond face alignment with an ensemble of regression trees,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1867–1874, 2014.

[171] K. Khabarlak and L. Koriashkina, “Fast Facial Landmark Detection and Applications: A Survey,” *J. Comput. Sci. Technol.*, vol. 22, no. 1, pp. 12–41, 2022.

[172] S. G. Hart, “NASA-task load index (NASA-TLX); 20 years later,” in *Proc. Hum. factors Ergon. Soc. Annu. Meet.*, vol. 50, no. 9. Sage publications Sage CA: Los Angeles, CA, 2006, pp. 904–908.

[173] J. Benesty, J. Chen, Y. Huang, and I. Cohen, “Pearson correlation coefficient,” in *Noise Reduct. speech Process.* Springer, 2009, pp. 1–4.