Summer 2024

Safe and Efficient Operation of Mobile Robots in Indoor Environments: A User-Centric Shared Control System with High-Level Navigation Capabilities

Ahmet Saglam
Old Dominion University

SAFE AND EFFICIENT OPERATION OF MOBILE ROBOTS IN INDOOR ENVIRONMENTS: A USER-CENTRIC SHARED CONTROL SYSTEM WITH HIGH-LEVEL NAVIGATION CAPABILITIES

by

Ahmet Saglam
B.S. August 2008, Turkish Military Academy, Türkiye
M.E. December 2019, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

MODELING AND SIMULATION ENGINEERING

OLD DOMINION UNIVERSITY
August 2024

Approved by:

Yiannis Papelis (Director)

Chung Hao Chen (Member)

Hong Yang (Member)

James F. Leathrum, Jr (Member)

ABSTRACT

SAFE AND EFFICIENT OPERATION OF MOBILE ROBOTS IN INDOOR ENVIRONMENTS: A USER-CENTRIC SHARED CONTROL SYSTEM WITH HIGH-LEVEL NAVIGATION CAPABILITIES

Ahmet Saglam
Old Dominion University, 2024
Director: Dr. Yiannis Papelis

Hospitalization and isolation can be a traumatic experience for immunocompromised children, especially because they are separated from their families and friends. Social robots have been proposed as a way to improve the quality of care for children hospitalized in isolation by providing alternative means of social interaction and support. Remote control of such robots in a hospital setting, particularly where safety is a major concern, can be a daunting task for young patients.

This dissertation introduces a multilevel shared control system for mobile robots, specifically companion robots in hospital-like indoor spaces. The system integrates user inputs with algorithmic semi-autonomous control at multiple levels of operation with the goal of only overriding user control to avoid collisions. At the foundational level, direct joystick control allows for immediate navigation, while higher levels introduce advanced functionalities such as corridor detection, corridor following, room-to-room navigation, and human-following capabilities.

At the core of the system is **U**ser-**Cen**tric **T**angent **B**ug for **B**lended Control, U-CenTB^2, a safe and efficient blended control algorithm implementing a modified Tangent Bug with a risk assessment strategy. U-CenTB^2 ensures safety through collision avoidance

without prior knowledge of the environment. As a user operates the robot in a building, it dynamically recognizes corridors, adding on a corridor-follower mode that intelligently avoids obstacles and enhances remote operation convenience. The system's adaptability can also be extended to a human follower mode that allows it to follow a recognized person. Additionally, by constructing a topological map, it is able to conduct future high-level tasks such as autonomously returning home or navigating to specific rooms.

To evaluate the performance of the algorithm, we present a simulation-based performance evaluation design for shared control algorithms by conducting batch simulations via Monte Carlo method. U-CenTB^2 is evaluated through this methodology. The results demonstrate the algorithm’s efficacy in preventing collisions and adhering to user inputs, thereby offering a significant contribution to teleoperation of assistive mobile robots.

Dedicated to my family for their constant love and support.

And to my dear friends Ismail Yolacici, Mehmet Demir, Murat Kose, Sener Kisak, Ferhat Keten, Enes Yilmaz, Selcuk Topal, and all others who have been unjustly imprisoned in Turkiye for years. Your strength, resilience, and unwavering spirit in the face of injustice inspire me every day. May this work serve as a small testament to your courage and a reminder that you are not alone.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Yiannis Papelis, for his invaluable guidance and support during these challenging years. His support extended beyond academics, touching every aspect of my family's life.

I am also profoundly thankful to the members of my dissertation committee, Dr. Chung Hao Chen, Dr. Hong Yang, and Dr. James F. Leathrum, Jr, for their insightful feedback and constructive criticism, which significantly enhanced this dissertation.

I extend my heartfelt gratitude to Dr. Amela Sadagic, my Master of Science advisor at the Naval Postgraduate School (NPS), and the faculty at the MOVES Institute of NPS for their invaluable contributions to my academic journey.

To my beloved wife, Menekse, and our children, Gonca, Orhan, and Nilufer, your love, patience, and understanding have been my constant source of strength. To my parents and siblings, thank you for your encouragement and belief in me.

I am grateful to my friend and colleague, Dr. Engin Baris, for his support and camaraderie. Also, I acknowledge my dear friends Ismail Yolacici, Mehmet Demir, and others who remain unnamed but not forgotten, whose strength and resilience continue to inspire me.

This dissertation is a testament to the resilience of the human spirit and the power of dreams. It serves as a reminder that even amid adversity, new paths can be found, and goals can be achieved.

Lastly, I am grateful to everyone who contributed to this dissertation in any capacity. Your support, whether big or small, has been greatly appreciated.

Thank you all.

NOMENCLATURE

2D	two dimensional
3D	three dimensional
Comm	communication
GPS	Global Positioning System
LIDAR	Light Detection and Ranging
SC	shared control
SW	smart wheelchair

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

The primary objective of this dissertation is to develop a shared control algorithm designed to facilitate safe teleoperation of mobile robots, particularly within indoor environments such as hospitals. While the operational context includes environments where social robots are used, the core aim is to ensure safe and efficient remote control. In this chapter, we outline the motivation behind this research, detail the proposed work, discuss assumptions and limitations, and present the overall structure of the dissertation.

1.1 Motivation

The motivation for this work is profoundly inspired by the story of David Carey, a young patient whose experiences at Children's Hospital of The King's Daughters (CHKD) emphasized the critical need for improved social connectivity for hospitalized children [1]. David's story is a touching reminder of the isolation many patients endure, driving our research to explore shared control systems that can offer both companionship and control to children in similar situations, making technological compassion a reality.

It is not uncommon for young oncology patients like David to be confined to their rooms during specific treatment phases that can last for weeks or even months. During that time, young patients cannot leave their room, and visitors and visit times are severely limited. The resultant social isolation, depression, and lack of control can have a detrimental effect on the patient. Hospitals often employ trained service animals and trained handlers that can visit a patient for

limited time periods; however, such solutions do not scale and are costly. Using specially designed service robots is a promising technology that can improve the quality of care for young patients in isolation [2], [3]. These robots can provide a vital connection to the outside world, enabling patients to engage with their environment without physical mobility [4]. A patient can tele-operate such a robot while interacting with persons near the robot, allowing a child to go on a walk with their parents, visit other patients, remotely explore, and gain back some amount of control and self.

However, it is also critically important to maintain safety and follow hospital rules while operating the robot. Unlike other use cases, autonomous robot operation is not desired in this case, as the patient gaining a sense of control by directly operating the robot is required. In this case, shared control is a viable alternative. Under shared control, the operator controls the robot but is augmented by an automatic controller to avoid collisions or other dangerous situations or even to ensure that the patient cannot navigate the robot into no-access areas.

Shared control offers several benefits over conventional remote controllers when fully autonomous operation is not the goal. With shared control, users can safely and efficiently operate the robot through blended operation without any training, resulting in smoother and more effective control [1]-[3]. Shared control has already demonstrated significant potential across various fields, including search and rescue operations [8] and assistive devices like smart wheelchairs [9] or surgical robots [10]. Shared control presents a promising solution for improving quality of care for immunocompromised children that need to be hospitalized in isolation.

However, shared control systems frequently prioritize safety, potentially compromising user control intuitiveness and comfort. For instance, in smart wheelchair applications, scenarios typically involve navigating to specific user-selected targets on a predefined map, rather than embracing direct control based on user inputs. Furthermore, when these systems attempt to predict

user intention, they rely on trajectory estimation methods utilizing fixed-time forward simulation. This approach can lead to inaccurate robot pose estimations at lower speeds, as it fails to account for the reduced movement increments. Besides, these pose estimation methods do not adjust their calculations based on input velocities, owing to their reliance on fixed time steps, thus neglecting the variable nature of user-directed speed and direction changes. Last but not least, while current systems might blend direct control with autonomous navigation for specific tasks, they often remove control from the user for these tasks, contradicting the direct control principle based on user inputs.

In addition to existing challenges in designing intuitive control strategies that effectively combine human and robotic inputs [5]-[7] in shared control systems, another critical aspect in research and development of these systems is their performance evaluation. One major challenge in evaluation is the need for human subjects to operate the robot. Moreover, the test environment in these experiments is often static and hard to modify, limiting the range of scenarios that can be explored or confidence in the robustness of an algorithm in the presence of different scenarios. These constraints make it difficult to comprehensively evaluate blended techniques' adaptability and performance under various conditions.

Acknowledging these challenges, this dissertation concentrates on developing a shared control algorithm to facilitate safe teleoperation of robots. Moreover, the research incorporates a simulation-based performance evaluation of shared control algorithms.

1.2 Development and Evaluation of Shared Control System for Mobile Robots

The purpose of this dissertation is twofold: to develop a shared control system to ensure safety and efficiency for indoor mobile robot teleoperation in hospital-like environments and

to design a comprehensive simulation platform to evaluate the performance of shared control algorithms rigorously, addressing the typical challenges associated with real-world testing. The two objectives are closely linked, with a joint focus on improving both the implementation and dependability of teleoperation of mobile robots.

The primary research question this dissertation seeks to answer is: How can a shared control algorithm be developed to improve the safety and efficiency of teleoperated mobile robots in complex indoor environments, such as hospitals? Specifically, this research focuses on engineering a solution that integrates real-time risk assessment and user-centric control strategies to avoid collisions while maintaining intuitive user control. The U-CenTB2 algorithm is designed to address these challenges by blending risk-based decision-making with a modified Tangent Bug approach, ensuring that the robot can navigate in indoor environments safely and effectively under human control.

1.2.1 Multilevel Shared Control System

To achieve our purpose, we present a multilevel shared control system, where users maintain control at every level of robot navigation, incorporating new capabilities without compromising control. Figure 1 offers a visual breakdown of the system's modular design. As the levels progress, the robot integrates more environmental data into its operations. At the foundational Level-0, user interaction is facilitated through direct joystick commands on a tablet interface, with the system utilizing only immediate obstacle data to guide safe and smooth direct teleoperation.

Advancing to Level-1, the robot employs its corridor detection capabilities to refine joystick commands into corridor-specific navigational directives, offering a simplified and

intuitive driving experience within these environments. With the integration of a topological map at Level-2, the robot gains the ability to perform more complex tasks, such as autonomously locating and moving to specific rooms. At Level-3, the robot is capable of following a person through hallways while maintaining obstacle avoidance, offering interactive and dynamic behavior under user command.

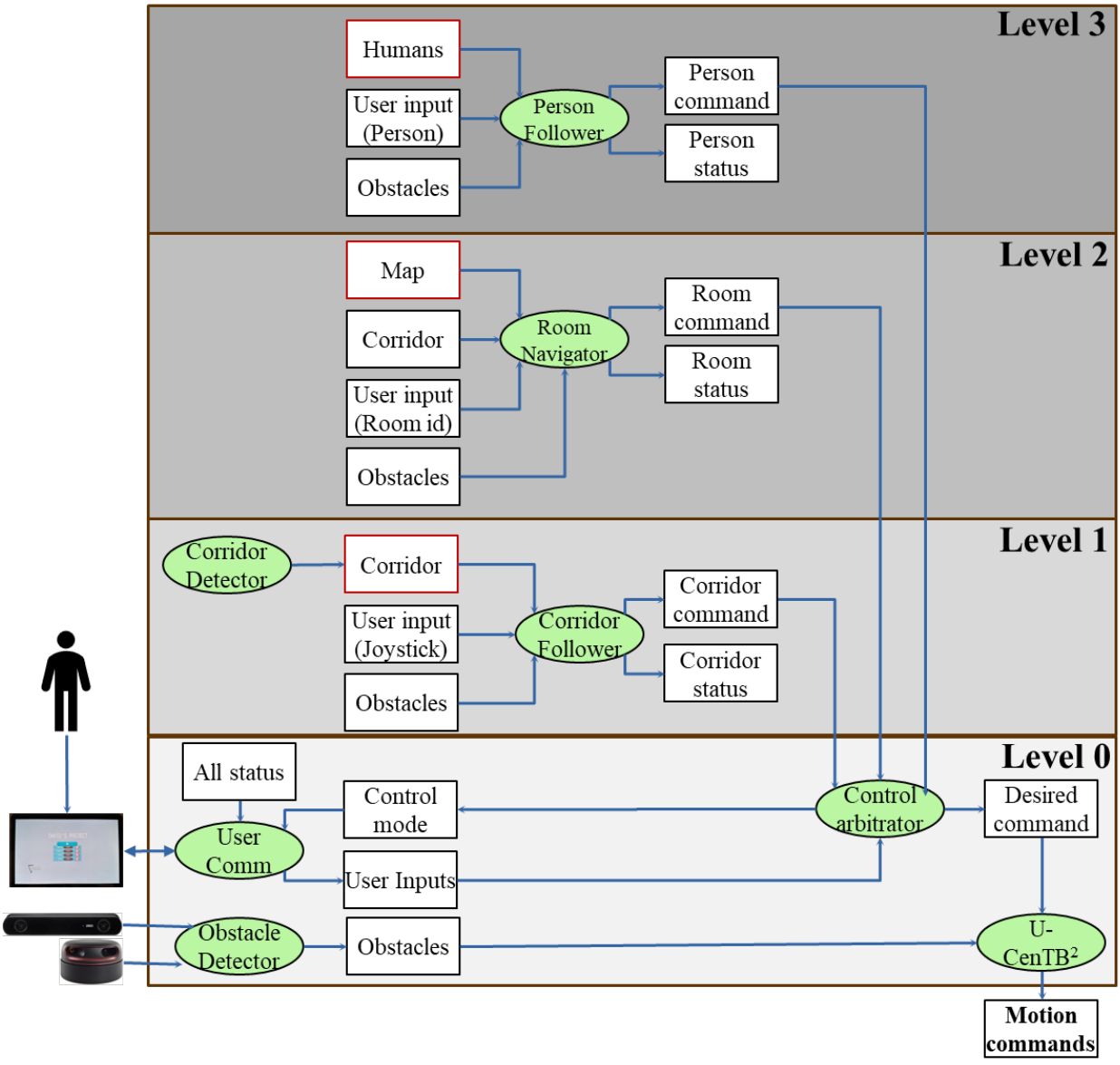

Figure 1. The overview of the multilevel shared control system.

For this dissertation, we took on the implementation of the modules in Level-0 and Level-1, but other modules in higher levels can be integrated into the system any time in the future as the presented system allows for such integrations. Moreover, the corridor detection module is introduced as an add-on algorithm, enhancing techniques for effective navigation within corridors. While the primary focus remains on collision avoidance in open room settings, the corridor module is presented to showcase its potential for integration. This inclusion not only demonstrates the algorithm's capabilities but also highlights future opportunities for its application in providing guidance based on corridor tracking.

1.2.2 Simulation Approach for Evaluation of Shared Control Algorithms

In addition to the aforementioned shared control system, we present a simulation-based performance evaluation design, utilizing Monte Carlo method to assess the effectiveness of a shared control algorithm in real-world scenarios. The simulation design aims to address the evaluation challenges inherent in shared control algorithms. In our setup, we run batch simulations to cover various possible scenarios, helping us understand how well the algorithm performs under different conditions. A key aspect of our simulation is the modeling of user input. We replicate realistic user commands using a combination of goal-driven inputs based on a global planner from Robot Operating System [11] augmented by noise-added velocities to emulate human-like random inputs. During the noisy intervals, Gaussian-distributed variabilities are added to the command velocities, simulating the inherent variability and unpredictability often seen in human control behavior [31]-[35]. To the best of our knowledge, no other study incorporates Gaussian distributions when modeling human input to evaluate the performance of teleoperated mobile robots. Furthermore, we developed a robot model

used in our simulations as a close replica of its real-world counterpart, ensuring that the results are as applicable to real scenarios as possible.

The simulation employs synthetic environments that are designed to resemble real settings, such as a hospital floor with various obstacles and layouts, providing increased confidence about the applicability of the results to the real world. Finally, we introduce three performance metrics that reflect both the effectiveness and efficiency of such algorithms in a simulation: Number of Collisions, Engagement Ratio, Velocity Deviation. These metrics can determine how well the control algorithm performs in environments that closely mirror real-world conditions.

1.2.3 Contributions

Our work contributes to the field of mobile robot teleoperation in indoor settings, particularly aiming at use-cases for telepresence social robots. The main contribution as well as the core module of the dissertation is **U-CenTB²**, a risk-based blended control approach, which implements a modified Tangent Bug algorithm with a user-driven directionality principle, wherein temporary goals are set in alignment with intended user commands rather than with a fixed global goal.

The dissertation further contributes to the literature with the development and implementation of the following:

- **Modular Multilevel Shared Control System**: Development of a shared control system that allows users to maintain control at various levels, enhancing safety and navigation in indoor environments like hospitals. The system's modular design supports integration of future functionalities and customization to meet specific

operational needs, enhancing adaptability and scalability.

- **Corridor Navigation and Obstacle Detection**: Implementation of modules for efficient corridor detection, corridor navigation and a method for obstacle detection using raw lidar scan and point cloud data, which rely on computationally efficient techniques without the need for trained AI models.
- **Simulation-Based Evaluation**: Development of a comprehensive simulation-based approach employing Monte Carlo method to measure the performance of shared control algorithms under varied conditions.
- **Performance Metrics**: Introduction of specific performance metrics, i.e., engagement ratio and velocity deviation to quantify effectiveness and efficiency of shared control algorithms.

1.3 Assumptions and Limitations

1.3.1 Assumptions

Users:

- It is assumed that humans using a controller have a basic understanding and ability to operate remote-control interfaces such as tablets.

Environment:

- Robot operates in indoor environments without a pre-built map.
- Indoor environment will generally consist of structured spaces like corridors and rooms that are amenable to detection and navigation by robot's sensors.
- Corridors are assumed to be primarily straight or gently curved and structured with widths up to 5 meters and heights up to 3 meters, suitable for the robot's sensor

capabilities for detection and navigation.

Robot and Tablet (Controller) Capabilities:

- Robot is treated purely in terms of kinematics, disregarding dynamics such as forces and torques for implementation of the shared control algorithm. This simplification assumes that the robot's movements can be adequately described and controlled using geometric and velocity parameters alone. Nonetheless, the robot model in the simulation reflects these features to closely replicate the movements of its real-world counterpart.
- Robot is assumed to operate within specific linear and rotational speed limits that are appropriate for safe and efficient teleoperation. These limits are set based on the robot's design to prevent collisions and handle its weight and are as follows:
    - The minimum and maximum linear speed are 0.2 m/s and 0.7 m/s.
    - The minimum and maximum rotational speed are 1.0 rad/s and 2.0 rad/s.

  The limits are fixed for typical operation but can be manually adjusted by users before startup to suit different operational needs or specific tasks. Once set, the algorithm adheres to these speed limits throughout operation, with no automatic deviations unless manually reconfigured prior to startup.
- Robot is assumed to have reliable odometry from its sensors, providing accurate information about its movement and positioning within the environment.
- Robot is a two-wheeled differential drive robot and is designed not to operate in reverse.

Simulation Platform:

- The Monte Carlo simulation method effectively captures a broad range of user input

variations.

- Simulated environments accurately represent real-world hospital settings.
- Performance metrics (collisions, engagement ratio, velocity deviations) adequately reflect the effectiveness and efficiency of a shared control algorithm.

1.3.2 Limitations

Focus on Technical Evaluation:

- While the dissertation proposes a shared control system for social robots such as emotional support robots, it primarily focuses on the development and technical evaluation of a shared control algorithm. We do not include evaluations of overall system usability, social and/or emotional impact or qualitative metrics related to a user's emotional and psychological experience with the robot. Such assessments require different methodologies and metrics, which are outside the scope of this dissertation.

Implementation Levels:

- The current scope of implementation is limited to the first two levels of the proposed work, leaving higher-level functionalities (i.e., room navigation, human following) for future integration.

Robot's Constraints:

- Robot is modeled primarily using kinematics, focusing on geometric and velocity parameters to simplify the control algorithm. This approach, while reducing computational complexity, does not account for dynamics such as forces and torques, which could impact accuracy in dynamic or load-variable environments.

Environmental Adaptability:

- The standardization of navigation algorithms based on assumed corridor dimensions and shapes does not account for irregular or unpredictably structured areas within various hospital settings.

Simulation Constraints:

- The evaluation methodology relies on simulation and might not fully capture real-world complexities such as dynamic obstacles.
- The user input model utilizes Gaussian-distributed variability, which might not perfectly represent all user behavior.

1.4 Structure of the Dissertation

The remainder of this dissertation is structured as follows.

Chapter 2: Literature Review: This chapter provides a comprehensive analysis of existing research in relevant areas. It begins with an overview of social robots and their use in healthcare, particularly with telepresence technology. The focus then shifts to the spectrum of autonomy in robotic systems, highlighting where shared control approaches fall in this spectrum. Next, various algorithms used in shared control systems are examined, followed by a discussion of methodologies for evaluating shared control algorithms and related gaps.

Chapter 3: U-CenTB^2 Algorithm: This chapter introduces the core contribution of this dissertation, the U-CenTB^2 algorithm. It begins by describing the kinematics model and technologies used in "David's Robot." It then details the risk assessment strategy used on two speed levels and defines high-risk behaviors that trigger U-CenTB^2 for safe and efficient teleoperation.

Chapter 4: Corridor Detector Module: This chapter presents a method for fast and efficient corridor detection using Point Cloud data from a single depth camera. The corridor detection module belongs to the Level-1 layer in our proposed multilevel shared control system and outputs corridor information that enables smooth navigation along hallways. This chapter contains the explanation of Point Cloud processing along with the use of the Hough Transform on depth camera data and the methodology for corridor detection.

Chapter 5: Additional System Modules: This chapter outlines crucial supporting components of the proposed multilevel system. It describes modules for user communication, control arbitration, corridor following and obstacle detection.

Chapter 6: A Simulation-Based Approach for Evaluating Shared Control Algorithms for Mobile Robots: This chapter addresses the challenges of real-world evaluations of shared control algorithms. A simulation-based approach is proposed and described, along with implementation details within a simulated environment.

Chapter 7: Experimental Results and Discussion for U-CenTB^2 Algorithm: This chapter presents the results of experiments designed to evaluate the performance of the U-CenTB^2 algorithm. Key performance metrics are defined, the experimental setup is outlined, and the results are analyzed and discussed in detail.

Chapter 8: Summary and Future Work: The dissertation concludes with a summary of the key findings and contributions. It also outlines potential directions for future research to extend and enhance the proposed robotic system and its shared control approach.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we discuss the previous work related to shared control.

2.1 Social Robots

Social robots represent a unique category of robots specifically designed to interact with humans in meaningful and engaging ways [17] - [19]. Unlike traditional industrial robots, which primarily focus on repetitive tasks in structured environments, or even general-purpose service robots designed for tasks like cleaning, social robots are meant to connect with people on an emotional and social level [20] - [22].

Telepresence Social Robots

A comprehensive review in [23] highlights technical methods and application areas of telepresence social robots. Notably, robots such as NAO [24], [25] and Pepper [26] have been used in healthcare settings to create friendship bonds with young oncology patients, alleviating their pain and distress. Moreover, telepresence robots, such as Keepon and the Huggable, have been used as therapeutic interventions for children with autism and hospitalized children, respectively [27], [28]. These robots offer a simplified social stimulus that is engaging to children on the autism spectrum and provide a comforting form factor for remote family and friends to interact with sick children in hospitals. In [24], [25], [29], studies

demonstrate that humanoid robots with various communication abilities can significantly benefit children, encouraging them to be more interactive and cooperative during treatment sessions. In eldercare, on the other hand, social telepresence robots help seniors interact with others, reducing social isolation [30], [31].

However, telepresence social robots face specific challenges when interacting with young patients, including ensuring safe operation despite erratic input [32] and maintaining engagement in dynamic hospital settings [33]. Shared control approaches can address these challenges by smoothing out imprecise commands and leveraging autonomy to keep users engaged [34].

2.2 Spectrum of Autonomy and Shared Control

Robotic systems operate on a spectrum of autonomy, ranging from fully human-controlled (teleoperation) to fully autonomous (Figure 2). Teleoperated robots offer precision and responsiveness but can place a high cognitive burden on the human operator [35], [36], potentially compromising safety and efficiency in complex or dynamic environments like hospitals [2], [3]. Fully autonomous robots, while promising independence, often lack the adaptability to handle the unpredictable nature of human behaviors and changing circumstances [37], [38]. Furthermore, in some situations, full autonomy may not be a desired feature, e.g., a person having a sense of control by directly operating the robot is desired [3].

Shared control systems seek a balance between human and robotic capabilities, offering potential advantages for safe and efficient operation. The success of shared control systems depends on smooth transitions between autonomy levels, clear communication of intent between human and robot, and adaptability to individual user needs [39] - [40].

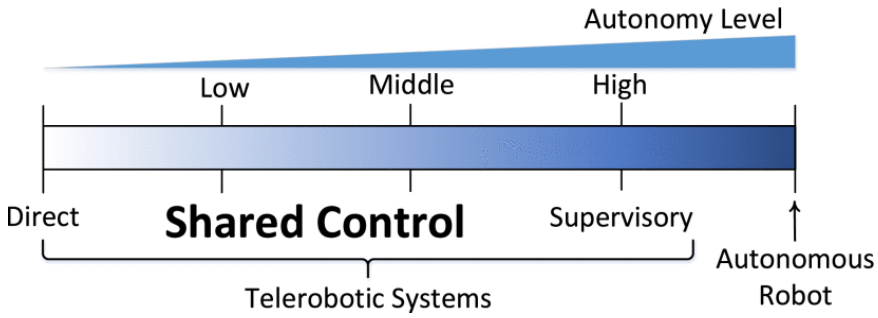

Figure 2. The spectrum of the autonomy in robotic systems [39].

2.3 Algorithms in Shared Control Systems

A shared control algorithm operates on the principle of maintaining user intent while ensuring safety and efficiency. It processes user commands and evaluates them against a set of criteria, such as obstacle proximity, robot's velocity, trajectory optimization, and so on. Figure 3 outlines a typical workflow for shared control algorithms in mobile robots. Inputs to the algorithm include user-generated command velocities, sensor data from the robot's environment, and/or pre-determined navigational goals. These inputs converge in the proposed shared/blended control algorithm, which then applies its logic such as "weighted blending function" [41], [32] to determine the final command outputs that direct the robot's movements. The outputs are a combination of the user's original commands and the algorithm's autonomous decisions, aiming to smooth out erratic inputs, prevent potential collisions, or ease the robot's motions for certain tasks.

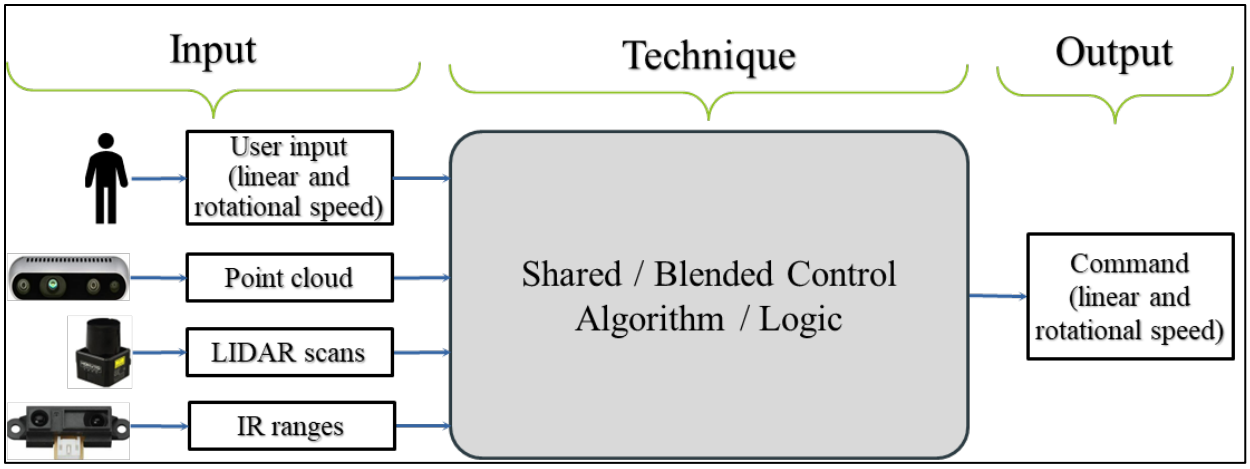

Figure 3. An overview of shared control algorithm workflow.

Traditional shared control methods continuously blend the user's intended command with the command velocity either linearly [32], [42], [43] or using some sort of probabilistic approach [45]-[47]. In linear approaches, a trajectory is estimated based on the user's desired velocity for a small amount of time in the future. Depending on the sensory data, an output driving command is computed. Probabilistic approaches, on the other hand, account for uncertainty in human input. Here, the real blending of the human's desired velocity obtained via a joystick and the path planner's velocity has been modeled using probability distributions. While linear blending most of the time does not guarantee safety in unstructured places, probabilistic approaches rely on global plans [47].

In addition to linear and probabilistic blending, studies in [49]-[51] classify shared control techniques for smart wheelchairs into three main categories. The first one is goal prediction-based methods, where a goal is estimated about where the user wants to travel. This is also known as prediction of intent. Any autonomous navigation technique can be employed as a shared control technique by using goal predictors. For instance, using landmarks like man-made rails on the ground, one can make an educated approximation about

the desired trajectory [42], [51]. Nevertheless, losing control of the robot completely to "software" is not desirable. Also, incorrectly inferring the user's ultimate aim is another highlighted drawback.

The second category relies on a set of navigation behaviors such as direct-control mode or autonomous mode that are activated in different contexts, e.g., traversing a hallway. In [52], [53], for example, the control of the robot alternates between autonomous navigation and manual control. The main disadvantage of these methods is that the user may become confused by the automatic switching of navigation modes. Lack of proper feedback often creates stress and frustration that may result in complete abandonment of the assistance.

The last category uses a continuous shared control approach, where user input is combined with collision avoidance interface [48]. This approach mostly relies on reactive navigation methods. Here, motion commands are computed from the simulated desired velocity and the obstacles around the robot. Studies in [55]-[57] do this merging via potential filed algorithms [57], [58]. Continuous shared control is also employed in [59], [60] by considering kinematic constraints. However, these methods suffer from being only applicable in local navigation with slow driving capabilities.

A recent survey [9] on smart wheelchairs (SW) explores teleautonomy options, among other SW features, for people with all types of disabilities. According to the authors, it is suggested to have an adaptive shared control depending on the user's capabilities. For example, if the user can create a global plan, then it would be better to help with only collision avoidance [46], [61]. Another crucial discovery is offering assistance when and as needed [51]. Therefore, a person should be able to take control anytime he/she wants to.

Additionally, [9] discusses modern semi-autonomous navigation techniques based on

task-specific operating mode selection [62]. These modes include machine learning [64]-[66], following [67]-[71], localization and mapping [72]-[76], and navigation assistance [77]-[80].

Even though unique contributions have been made through the above-mentioned research and development for SW users, to the best of our knowledge and through the literature review, we have not seen any efforts where shared control is maintained at different layers of navigation while adapting to the new features at each level, particularly for hospitalized people or users who have limited or no visual contact with the robot.

2.4 Evaluation of Shared Control Algorithms

Blended or shared control techniques, a key element in facilitating cooperation between human operators and intelligent machines [80], have been a focus in robotics since the advent of telerobotics [2]-[4]. In addition to existing challenges in designing intuitive control strategies that effectively combine human and robotic inputs [5]-[7], a critical aspect in research and development of these systems is their performance evaluation. Typically, this evaluation involves experiments where users remotely navigate a robot through an obstacle course [8]-[11]. In such experiments, two categories of metrics are used: quantitative and qualitative. Qualitative metrics are out of the scope of this paper. For quantitative, task completion time [12]-[13], number of collisions [85], [86], and intervention level [87] are common metrics used to assess the performance of an algorithm.

However, the evaluation process for these systems presents its own set of challenges. One major challenge is the need for human subjects to operate the robot. Moreover, the test environment in these experiments is often static and hard to modify, limiting the range of scenarios that can be explored or confidence in the robustness of an algorithm in the presence of different

scenarios. These constraints make it difficult to comprehensively evaluate the blended techniques' adaptability and performance under various conditions.

Some researchers benefit from simulations to mitigate these challenges. Physics-based, high fidelity simulators allow for development, verification, and validation of robotics systems [88]. They provide an ideal environment to test and refine robotics algorithms under various conditions that mimic actual settings [89], [90]. However, the development and assessment of shared control algorithms pose a unique challenge that applies to both physical and simulation-based testing, namely the requirement for human input.

In one study related to a semi-autonomous control system for ground vehicles, researchers assessed the effectiveness of their methodology in simulations [91]. Here, to simulate operator inputs, they used a pure pursuit driver model [92], a path tracking algorithm typically employed in autonomous vehicles. While this model provides inputs to mimic human behavior, the input lacks variability and randomness often found when humans manually control a robot.

In another work by [85], the authors investigated learning-based semi-autonomous controllers for search and rescue robots through extensive experiments conducted on a simulation platform with human subjects. A similar approach was used by [93], where the researchers tested their model predictive control based shared control method running experiments on the ANVEL simulator [94] with human subjects. The authors in [95], on the other hand, proposed a VFH+ [96] based blended control technique for teleoperated mobile robots. Again, they employed three professional robot users to evaluate the performance of their work on a high fidelity simulator, Gazebo [97]. While this mixed approach, i.e., virtual environment and real users, alleviates the challenges associated with physical testing environments, it still bears the complexity of involving actual users, which introduces variability and requires proper experiment design.

2.5 Summary

In summary, the literature highlights the critical role of shared control systems in advancing autonomous behaviors while preserving essential human oversight, particularly in sensitive environments such as healthcare. These systems aim to offer a balanced approach that leverages strengths of both human intuition and robotic precision to improve safety, efficiency, and user experience. However, development of an algorithm to use in such systems, especially in real-life use cases introduces challenges such as pose estimation, maintaining user control, the need for a global map, and so on. Even though having a pre-made map is feasible, allowing rapid adaptation of a robot in different hospitals makes it hard to depend on that map.

On the other hand, the evaluation of shared control algorithms remains a complex challenge, requiring both qualitative and quantitative metrics to assess performance accurately. Traditional evaluation methods often involve user-operated obstacle courses, but these can be limited by static and hard-to-modify environments.

The proposed multilevel shared control system and the simulation-based performance evaluation approach aim to address these gaps, offering a solution for safe and efficient indoor mobile robot teleoperation.

CHAPTER 3

U-CENTB^2: RISK-BASED BLENDED CONTROL ALGORITHM WITH USER-CENTRIC TANGENT BUG

Teleoperated mobile robots in shared spaces like hospitals hold the potential to enhance patient care and improve social interaction. However, a critical challenge lies in ensuring safe and efficient navigation under the direct control of a user who may not have full visibility of the environment. In a high-stakes setting, even minor collisions can have significant consequences. Traditional obstacle avoidance algorithms, such as the classic Tangent Bug, excel at guiding robots around obstacles but primarily focus on reaching a fixed goal and are counter-intuitive when factoring human control which typically is greedy and less systematic. This can conflict with the dynamic and unpredictable nature of teleoperation where the user's intent should be prioritized.

To address this challenge, we propose U-CenTB^2, a shared control algorithm that adapts the strengths of Tangent Bug for user-driven scenarios. U-CenTB^2 prioritizes user input by setting temporary goals aligned with their intended trajectory, while continuously assessing risks in the environment. A three-tiered system (No-Risk, Low-Risk, High-Risk) determines when obstacle avoidance must temporarily override direct user control. This blended approach seeks to empower users by enabling them to navigate freely, while also providing a crucial safety net through risk management.

In this chapter, we detail the implementation of U-CenTB^2. We start with a brief summary of the Tangent Bug algorithm and then explain the risk-based state machine, which governs control

transitions. Next, we present the risk evaluation methods used in Normal Speed and Slow Speed modes. We then delve into High-Risk behaviors of U-CenTB^2. Finally, we present the key features and specifications of David's robot, on which U-CenTB^2 is implemented.

3.1 Tangent Bug: A Brief Recap

Tangent Bug algorithm [98] is a path planning method that combines motion-to-goal behavior with boundary following. It aims to navigate from a starting point to a target while avoiding obstacles. The algorithm operates by using sensor data to identify contours of obstacles and calculate tangent points, which are potential transition points around the perimeter of obstacles. With a goal location known, Tangent Bug assesses these tangent points to choose a direction that optimally balances progress towards the goal and distance around an obstacle. The robot then follows the boundary of the obstacle to the chosen tangent point. When an obstacle blocks the direct line of sight to the goal, Tangent Bug switches to a boundary-following mode. The robot traverses the edge of the obstacle until it either reaches the goal or finds a point where the goal is visible again and is closer than the point where it first encountered the obstacle.

Tangent Bug prioritizes reaching a defined destination effectively, which is ideal in fixed-goal scenarios. Although variants [35]-[37] of this algorithm enhance capabilities for better navigation, real-world implementations still face challenges. Firstly, the focus on reaching a predefined destination can conflict with teleoperation scenarios where the user's intent might change dynamically. Strict adherence to a fixed goal can override user-desired paths and reduce the sense of control. Secondly, in complex environments, Tangent Bug can become trapped in local minima situations, where it endlessly circles an obstacle without making progress towards an intended target. Lastly, Tangent Bug's focus on localized obstacle avoidance can lead to

unnecessarily long or convoluted routes, especially when the user has a clear overall path in mind. In addition to these challenges sensor inaccuracies, dynamic environments, and complex obstacle shapes further limit the capabilities of the algorithm.

The Tangent Bug algorithm provides a robust foundation for obstacle avoidance but exhibits the above limitations, especially in teleoperation due to its fixed-goal focus and potential for suboptimal pathing. Our U-CenTB^2 algorithm leverages the strengths of Tangent Bug's obstacle avoidance while addressing its shortcomings for user-driven scenarios. Specifically, U-CenTB^2 introduces:

Dynamic Goals: Temporary goals are continuously generated based on user input, prioritizing their intended path and ensuring responsiveness rather than strict adherence to a single, fixed goal.

Risk-Based Control: A three-tiered risk assessment system (No-Risk, Low-Risk, High-Risk) determines when autonomous obstacle avoidance interventions are necessary. This balances user control with safety guarantees.

Path Optimization: Risk evaluation can incorporate user input to avoid the local minima and unnecessary detours associated with the classic Tangent Bug.

3.2 U-CenTB^2 Algorithm State Machine

Our algorithm determines the robot's motion commands based on a risk state machine with three distinct levels: *No-Risk, Low-Risk*, and *High-Risk*. These states and their transitions are governed by the detection and evaluation of potential collision threats based on user input and obstacle proximity. Figure 4 depicts the state transitions.

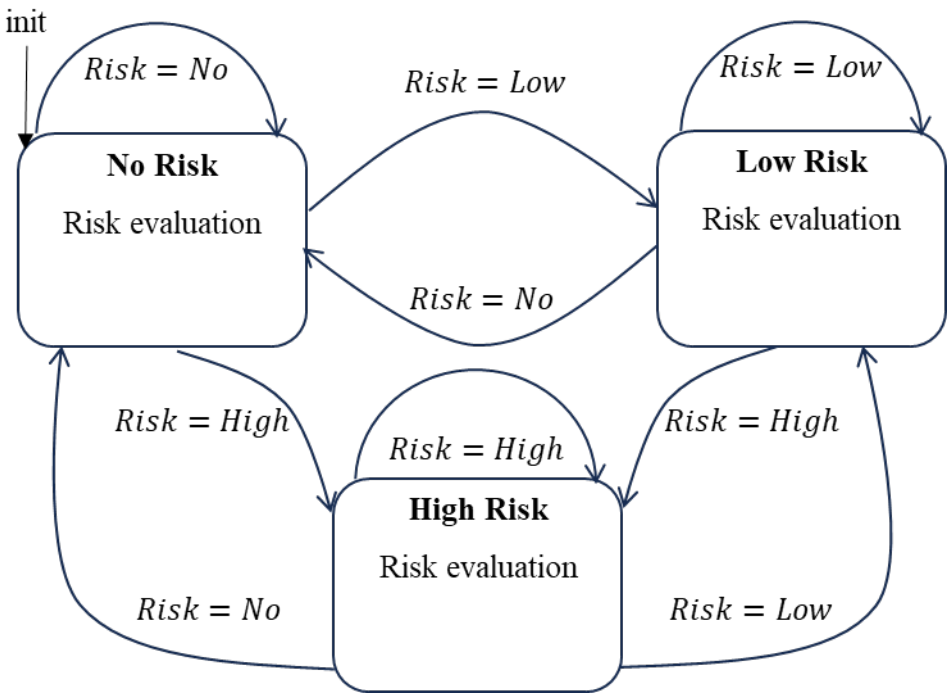

Figure 4. Risk state machine that controls the robot's motions.

Let the system states be denoted as S_i where i ranges over the set of possible states. The transitions between states are governed by conditional expressions based on the risk level. Then, the state transition function δ is defined as follows:

$$\delta(S_i, R_k) = S_j$$

where R_k is the risk condition derived from evaluating the risk associated with the current state of operation; k and j ranges over the set of possible risks and states, respectively.

State Definitions and Transitions:

No-Risk State ($S_{No-risk}$): The robot operates in this state when no immediate collision risks are detected. The robot executes user commands directly. Transition to other states occur as follows:

$$\delta(S_{No-risk}, R_{Low}) = S_{Low-risk}$$

$$\delta(S_{No-risk}, R_{High}) = S_{High-risk}$$

Low-Risk State ($S_{Low-risk}$): Activated when a potential collision is detected at a distance that allows for precautionary speed adjustments without requiring immediate action. The robot reduces desired speed to a safer limit while still adhering to the user's directional input. Transition to other states occur as follows:

$$\delta(S_{Low-risk}, R_{No}) = S_{No-risk}$$

$$\delta(S_{Low-risk}, R_{High}) = S_{High-risk}$$

High-Risk State ($S_{High-Risk}$): Engaged when a potential collision is detected at a distance that requires immediate action to safely navigate around obstacles. Here, the U-CenTB^2 algorithm computes a velocity that safely navigates around obstacles and then resumes following user commands. Transitions to other states occur as follows:

$$\delta(S_{High-risk}, R_{No}) = S_{No-risk}$$

$$\delta(S_{High-risk}, R_{Low}) = S_{Low-risk}$$

In our algorithm, we assume obstacles are represented as vertices of a rectangle, which can be at any orientation. Nonetheless, the logic can be applied to obstacles represented in any form.

3.3 Risk Evaluation Overview

Every time the user issues a motion command, the algorithm conducts a risk evaluation to identify potential collisions along its projected trajectory, with the workflow outlined in Figure 5. We use two modes of collision detection depending on desired speed, namely *normal* and *slow*. The purpose of this dual-speed strategy is to maintain high levels of safety across all operational

speeds by employing a more refined collision detection method when the robot moves slowly, where it might "sneak" into an obstacle because the changes in position are so minimal that they might not trigger a collision threat using traditional pose estimation methods.

The 'normal' and 'slow' speed thresholds for the robot are pre-set parameters that are determined based on the specific characteristics of the robot's platform, such as its weight and the responsiveness of its motor controllers. These thresholds are not dynamically calculated through an algorithmic process; instead, they are derived from empirical testing and expert assessments of the robot's handling and operational capabilities within its operating environment.

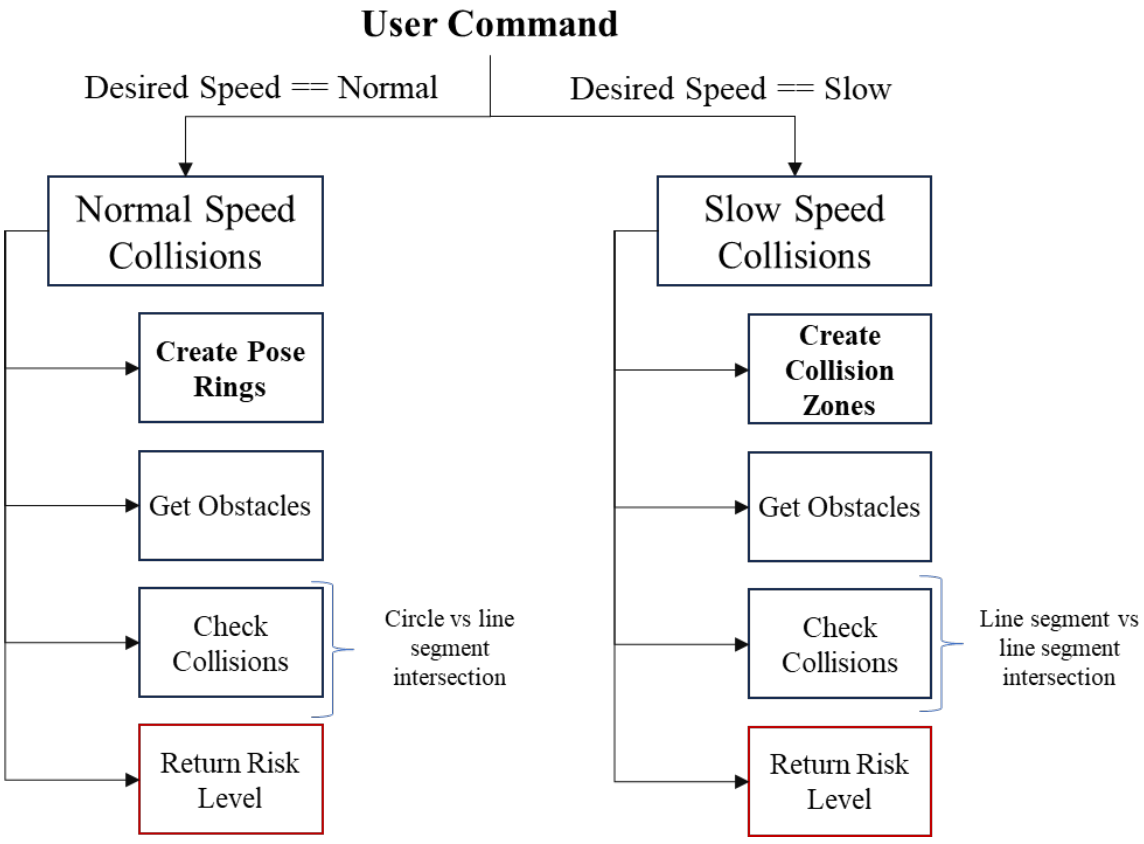

Figure 5. Workflow of risk evaluation under dual-speed strategy.

3.4 Normal Speed Risk Evaluation

Risk assessment at normal speeds involves creation and evaluation of pose rings. Figure 6 illustrates the concept of pose rings, a term used in this dissertation to describe the methodology for assessing collision risks at normal speeds. Pose rings are hypothetical future positions of the robot, spaced at fixed intervals along its intended path. Each ring represents a potential future position of the robot, providing a systematic way to evaluate the likelihood and severity of potential collisions. The parameters in Figure 6 are explained in detail in the following section.

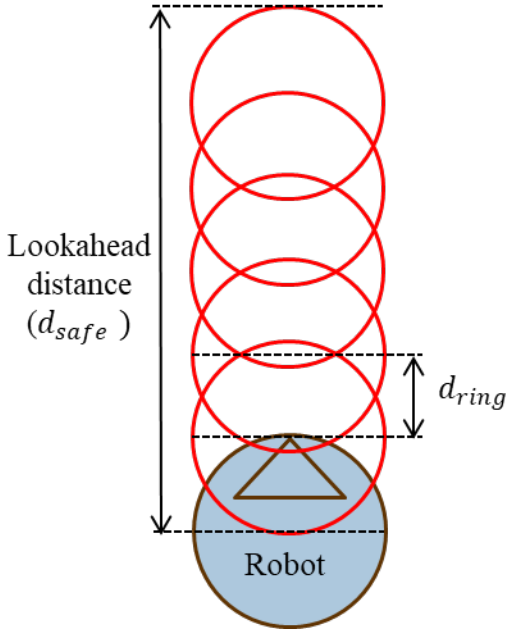

Figure 6. Illustration of pose ring given a forward speed to the robot.

3.4.1 Creation of Pose Rings

Unlike traditional fixed-time trajectory estimations, where overall lookahead projection

fluctuates at different speeds, we offer a spatially consistent approach for future pose prediction. Instead of relying on a constant time step and total simulation time, we dynamically adjust simulation steps (n_{steps}) and delta time (Δt), based on user-defined parameters, which include lookahead distance (d_{safe}), number of rings (n_{rings}), and interval steps per ring (n_{r_steps}). Using n_{r_steps} reduces integration error in trajectory prediction by ensuring smaller, more accurate time steps (Δt), thus enhancing the fidelity of pose estimations between rings.

In order to understand a differential drive robot's estimated trajectory in the future, we first explain the equations of motion for differential drive robots and then detail how the robot pose is estimated through the rings.

3.4.1.1 Equations of Motion for Differential Drive Robots

A differential drive system is characterized by two independently driven wheels on either side of a robot, allowing for a full range of movements by varying speeds and directions of the wheels. An example of two-wheeled robot with a caster wheel on the back is shown Figure 7, left.

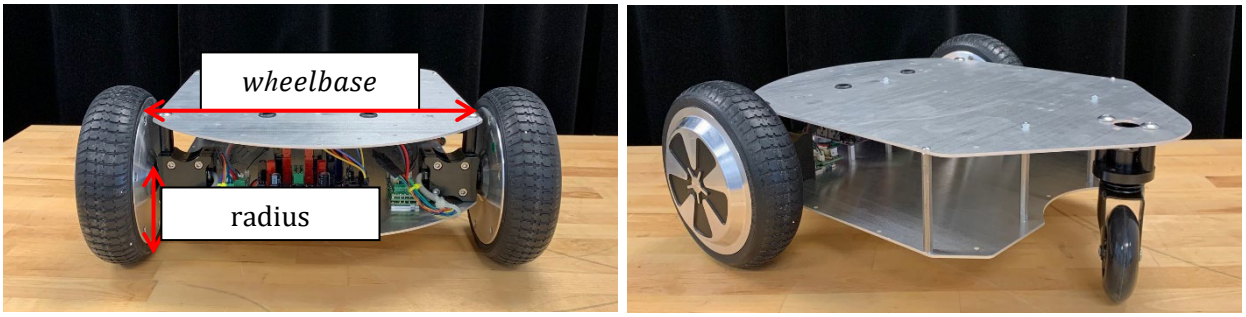

Figure 7. Two-wheeled robot base with a caster wheel in the back.

Motion commands (linear and rotational speed) from the user or the control algorithm are

translated into rotational speeds for left and right wheels using differential drive inverse kinematics. The formula to obtain individual wheel speed is as follows:

$$\boldsymbol{v_{left}} = \frac{v - \left(\omega * \frac{wheelbase}{2.0}\right)}{r};$$

$$\boldsymbol{v_{right}} = \frac{v + \left(\omega * \frac{wheelbase}{2.0}\right)}{r},$$

where:

v is desired (user or algorithm generated) linear velocity in meters/second,

ω is desired (user or algorithm generated) angular velocity in radians/second,

v_{left} are v_{right} are desired velocities on left and right wheels in radians/second,

r is radius of wheels in meters,

$wheelbase$ is distance between two wheels of the robot in meters.

Differential drive forward kinematics, on the other hand, is used to predict a robot's future pose based on its current position, orientation, and the velocities of its left and right wheels. Integrating this with intended control commands enables prediction of the robot's future trajectory, a key technique in blended control algorithms for collision avoidance. The equations for differential drive forward kinematics are as follows:

$$\boldsymbol{v} = (v_{left} + v_{right}) * r / 2;$$

$$\boldsymbol{\omega} = (v_{right} - v_{left}) * r / wheelbase;$$

$$\boldsymbol{x'} = x + v * \Delta t * \cos\theta;$$

$$\boldsymbol{y'} = y + v * \Delta t * \sin\theta;$$

$$\boldsymbol{\theta'} = \theta + \omega * \Delta t,$$

where:

Δt is the delta time,

x, y, θ are current position and orientation at time t,

x', y', θ' are future position and orientation at time $t + \Delta t$.

3.4.4.2 Calculation of Pose Rings

To calculate these pose rings, we use the following procedure:

- Determine n_{steps}, d_{ring}, and Δt.

$$\boldsymbol{n_{steps}} = \frac{n_{r_{steps}}}{n_{rings}}, \qquad \boldsymbol{d_{ring}} = \frac{d_{safe}}{n_{rings}}, \qquad \boldsymbol{\Delta t} = \frac{d_{ring}}{|v|.n_{r_{steps}}} \tag{1}$$

  where d_{ring} is distance between rings (m) and v (m/s) desired linear speed.

- Given the robot's current pose (x, y, θ) and $\boldsymbol{\Delta t}$, estimate a new pose (x', y', θ') after each time step using the kinematic equations for a differential drive robot.

$$x' = x + v.\Delta t.\cos(\theta);$$

$$y' = y + v.\Delta t.sin\,(\theta);$$

$$\theta' = \theta + \omega.\Delta t.$$

- Repeat new pose calculations for n_{steps} to estimate the robot's trajectory while storing only poses that correspond to a ring location because the rings are already sampled at a distance that ensures there is no gap between them to effectively detect obstacles. This not only reduces computational load but also improves reaction times to dynamic

changes in the environment.

3.4.2 Collision and Risk Assessment with Rings

After generating pose rings, we assess potential collisions by checking for intersections between the rings, which use the robot's width as their radius and the edges of any obstacles. If no intersections are found, the robot is at No-Risk state. However, if an intersection occurs, the risk state is updated based on the proximity of the intersecting ring to the robot. Intersections with rings closer to the robot, such as rings 1 and 2, are categorized as High-Risk, while intersections with outer rings, such as rings 3, 4, and 5, indicate a Low-Risk situation.

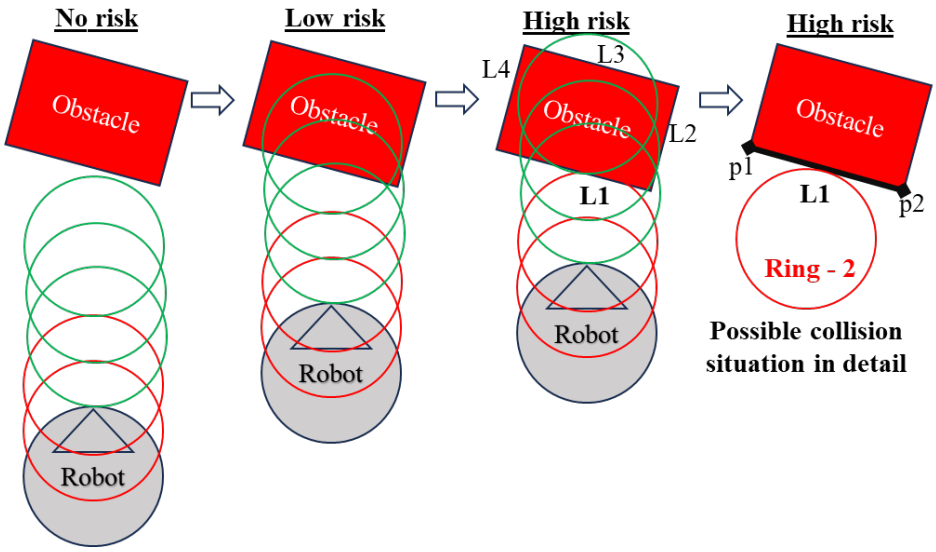

Figure 8. Risk evaluation example with pose rings.

In the example provided in Figure 8, when a user commands the robot to move forward and no ring intersects with obstacles, the robot remains in the No-Risk state. As the robot advances,

and outer rings (3, 4, and 5) begin to intersect with an obstacle's line segment (L1), the system transitions to a Low-Risk state. If the user continues the same command and Ring-2 intersects with the L1 line segment, the risk state escalates to High-Risk. At this point, we record the robot's current orientation (shown as a black triangle in Figure 8) and the endpoints of the intersecting line segment (p_1, p_2). This data is crucial for the High-Risk state, wherein the U-CenTB^2 algorithm is activated to navigate around the obstacle.

3.5 Slow Speed Risk Evaluation

For slow speed user inputs, we evaluate risk by dividing the robot's surrounding area into six quadrants (front left, front, front right, back right, back, and back left) based on the robot's current orientation and desired command, addressing all potential directions of interaction.

Each quadrant is defined by four vertices (p_1, p_2, p_3, p_4) that outline its boundary. Central vertices (p_1, p_2) are located along the robot's main axes. Extended vertices (p_3, p_4) set outer boundaries and are determined by:

$$p = \left(d.cos(\theta_p), d.sin(\theta_p) \right) \tag{2}$$

where d is the distance used as a basis for determining how far out a quadrant extends from the robot's base; θ_p is angle for each vertex j defining the boundary of a quadrant, calculated relative to predefined angle (θ) for constructing a quadrant. Figure 9 illustrates the front quadrant located in front of the robot.

Based on input velocities, the algorithm selects a quadrant relevant to the robot's intended direction for collision assessment. For instance, consider a scenario where the robot receives a

command to move forward with a slight left turn, indicated by a positive linear speed and a positive angular velocity. In this case, the front-left quadrant is selected for collision check.

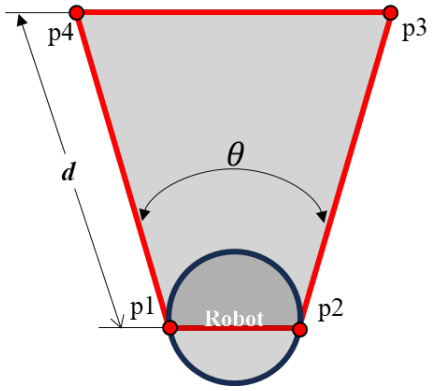

Figure 9. Construction of a collision quadrant.

The algorithm then evaluates the risk by checking for intersections between the line segments that define both the selected quadrant and obstacles. If an intersection is detected with any edge of an obstacle within the chosen quadrant, the system triggers a High-Risk state. In this case, similar to the procedure followed during normal speed evaluations, the robot's current orientation and endpoints of the intersecting line segment are recorded.

3.6 High-Risk Behavior with U-CenTB^2

During teleoperation of the robot, whenever risk evaluation yields the High-Risk state, the output motion commands are computed by the proposed U-CenTB^2. It sets dynamically temporary goals based on the user's steering preferences or, in the absence of specific input, chooses a direction that minimizes rotation from the current heading. If setting a goal is not feasible, the robot defaults to moving tangentially to the obstacles.

3.6.1 Setting a Goal for U-CenTB^2

Unlike traditional go-to goal and boundary-following behaviors that prioritize the shortest path to the goal when going around an obstacle and recognizing the limitations of boundary-following in real-world applications where precise control over the robot's motion in response to range sensor data can be challenging, we establish short-term goals that will not only help navigate smoothly around obstacles but also accommodate a user's preferences by maintaining a course that closely matches their intended trajectory.

In scenarios where a user provides steering input, U-CenTB^2 prioritizes the user's intended direction when setting goals. If the user steers left or right, the algorithm selects a reference point on the corresponding side of the obstacle to align with the desired rotation. This allows the robot to avoid obstacles while adhering closely to the user's last known command, effectively blending autonomous navigation with user control. In the scenario on the right in Figure 10, a user wants to move the robot forward while steering right but will hit obstacle if they continue to send the same commands. The algorithm chooses p_2 as the goal reference to steer right as the user wanted to turn in this direction.

On the other hand, when there is no user steering input, the algorithm autonomously selects a goal reference point that minimizes the robot's rotation to avoid obstacles based on the following criteria:

- *Find obstacle angle.*

$$\theta_{obstacle} = \frac{arctan(y_2 - y_1)}{(x_2 - x_1)} \tag{3}$$

where $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ are two points defining an obstacle, and $\theta_{obstacle}$

is the angle of the line segment connecting these points relative to a fixed coordinate system, i.e., odometry frame.

- *Find relative orientation.*

$$\theta_{relative} = \theta_{robot} - \theta_{obstacle} \tag{4}$$

where $\theta_{relative}$ is relative orientation between the robot's current orientation θ_{robot} and $\theta_{obstacle}$.

- *Select reference point* $\boldsymbol{p_r}$.

$$p_r \begin{cases} use\ p_2\ if\ -\frac{\pi}{2} < \theta_{relative} < 0\ or\ \frac{\pi}{2} < \theta_{relative} < \pi \\ use\ p_1\ otherwise \end{cases} \tag{5}$$

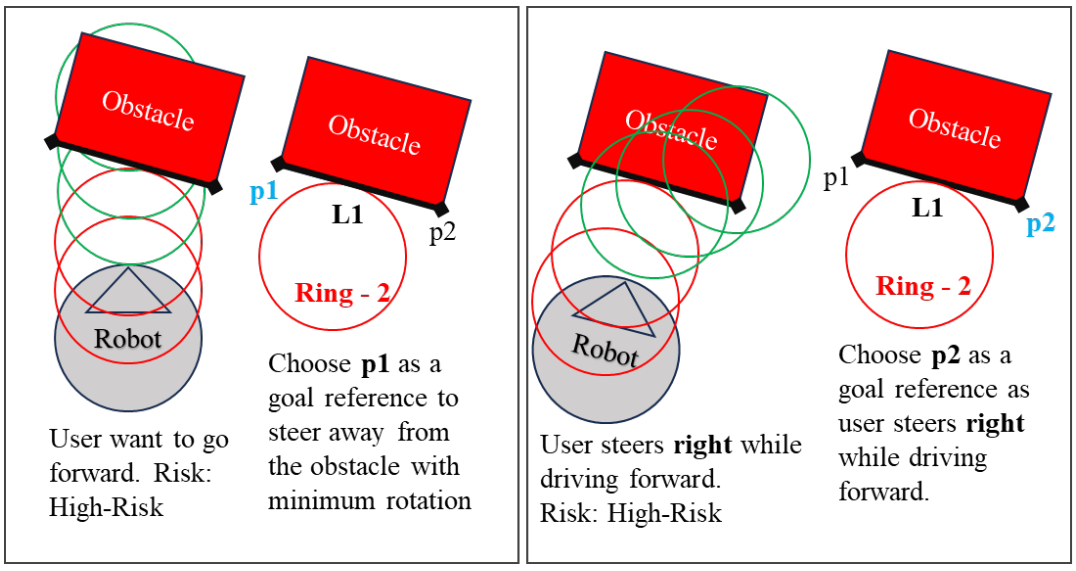

Figure 10. Selection of p. Left: No directional command. Right: A command with right turn.

In Figure 11's top-left image, for example, **p1** is selected as p_r because it provides the least rotational adjustment from the robot's current heading, thus optimizing an avoidance maneuver. If the robot's trajectory towards the obstacle is perpendicular, then **p1** is chosen as the default reference point, simplifying the decision-making process in the absence of user direction. Once the reference point p_r is selected, a goal is searched with the following procedure and illustrated in Figure 11's bottom images:

- An offset (p_{offset}) is created relative to p_r based on the robot's radius (r) with a buffer and the obstacle angle:

$$p_{offset} = (cos\ (\theta_{obstacle}) * 2 * r,\ sin\ (\theta_{obstacle}) * 2 * r) \tag{6}$$

- The offset is assessed for potential collisions with the obstacle. Should a collision be anticipated, an alternative offset from the non-reference endpoint of the obstacle is then calculated.
- Once a collision-free p_{offset} is found, a vertical adjustment is made to this point to ensure the robot's path is safely directed away from the obstacle. After adjusting for vertical distance, the resulting point is set as the new goal for the robot.

Goal setting is a dynamic process and involves continuous reevaluation and adjustment of the goal based on the situation and user commands. If a goal is not feasible, the robot defaults to moving tangent to the obstacle, mirroring traditional boundary-following behavior but as a last resort.

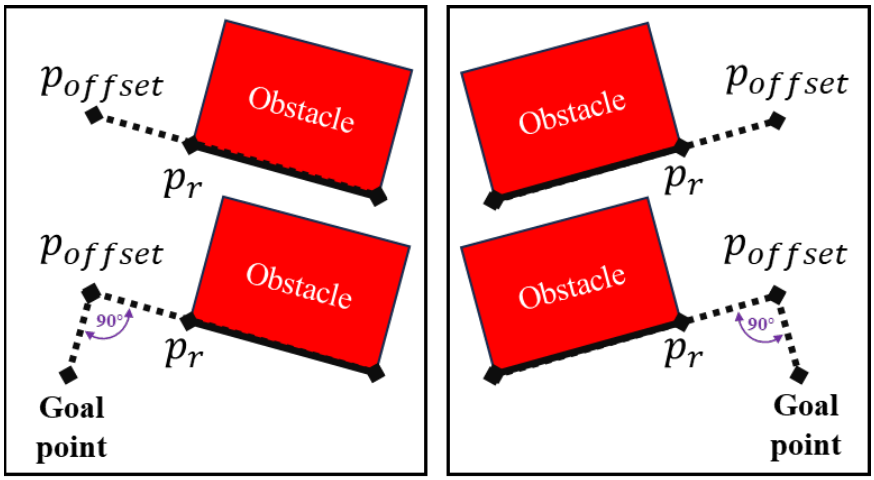

Figure 11. Setting goal based on the selected reference point.

3.6.2 Motion Commands by U-CenTB^2

After a goal is set, U-CenTB^2 issues go-to-goal motion commands to orient the robot towards the goal and then initiates movement towards this goal while continually evaluating collision risks and the criteria to exit the High-Risk state. If a goal is not set, then go-tangent motion commands are computed to move tangentially to obstacles.

For go-to-goal, the robot calculates a straight-line distance to the goal and adjusts its linear speed accordingly. If this distance is within a certain threshold, it indicates proximity to the goal, and if the path is clear given the desired input, the robot proceeds under user commands. When aiming for a goal, it checks for potential collisions and may seek alternative goals if obstacles are detected. In go-tangent, the algorithm initially decides the direction of the tangential motion around the obstacle based on the user's input and relative orientation. Then, the robot orients towards the tangent path. While moving tangentially to avoid obstacles, it reassesses risks and seeks new goals.

The last robot heading recorded during lower risk states is used for pose estimation of the

last intended path and collision assessment. If the user continues sending forward commands, the robot's pose rings are evaluated against the obstacles, using the last known heading, not the current heading, to determine if High-Risk state persists. In Figure 12, the user issues forward commands from time zero (t_0) to four (t_4). At t_1, upon entry into High-risk state, the robot logs its current orientation and the obstacle's endpoints. Between t_2 and t_3, the algorithm takes control of the robot to avoid a collision, utilizing the logged information to determine the goal and exit condition. At t_4, the algorithm assesses that the last recorded desired trajectory is safe to continue and hands control back to the user.

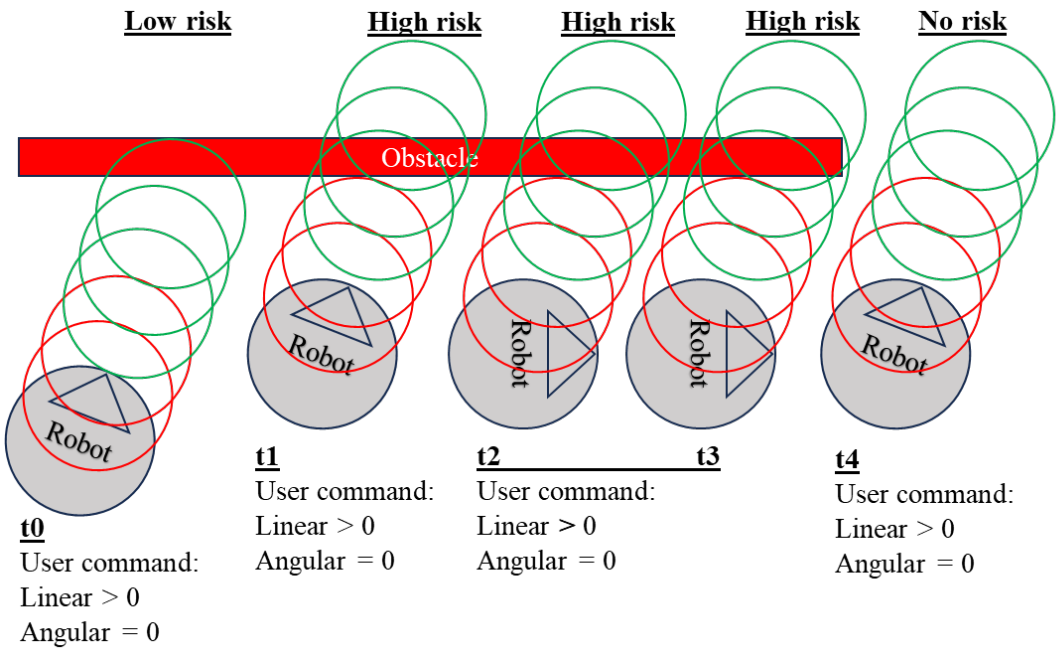

Figure 12. High-Risk risk evaluation when desired linear speed greater than zero.

CHAPTER 4

CORRIDOR DETECTOR MODULE

This chapter introduces the corridor detector module, which is designed to demonstrate additional algorithms that can be integrated into the shared control system for mobile robots. The corridor detection and following techniques presented here highlight the system's potential for enhanced navigational capabilities within structured environments such as hospital corridors. While the primary focus of this dissertation is on collision avoidance in open spaces, the corridor detector module showcases opportunities for future integration and expansion.

The ability to accurately detect corridor-like structures is essential for efficient navigation of mobile robots in indoor environments, where the lack of GPS functionality poses significant challenges. In our previous work [102], we introduced a novel methodology for real-time corridor detection utilizing a single depth camera. This approach, integral to our proposed multilevel shared control system, is designed to identify corridors reliably by analyzing wall structures, even in cluttered settings where obstacles such as trash cans or chairs are present. Operating continuously in the background across all levels of the proposed shared control system, the corridor detection module primarily contributes its outputs to Level 1, enabling smooth navigation along hallways and allowing the robot to be controlled via simple joystick commands without the risk of collision.

This chapter details our corridor detection methodology within the proposed multilevel control system. Section 4.1 reviews relevant literature and outlines the advantages of our method. Section 4.2 discusses data capturing and processing and explains our use of Hough Transform to identify corridors. Section 4.3 presents our findings alongside real-life examples. The chapter

concludes with a summary that recaps the key points discussed.

4.1 Introduction

Effective and efficient robot teleoperation in indoor settings is heavily reliant on the robot's ability to sense its environment, detect obstacles, and continuously create motion plans in dynamic conditions. Unlike outdoor robots, which can utilize Global Positioning System (GPS) for navigation, indoor robots face significant challenges due to the absence of GPS signals. In this sense, the ability to identify walls and understand the layout of corridors becomes crucial. These corridors provide essential navigational cues in environments where maps may not be available or are incomplete, ensuring safe movement and effective obstacle avoidance.

4.1.1 Related Work

Researchers have developed various methodologies for detecting corridor-like structures, depending on the specific needs of the system and the sensors available. Early methods that relied on single cameras, such as those discussed in [104] - [106], focused on simple image processing techniques like vanishing point detection to enhance corridor recognition. However, these methods often struggle with complex geometries, varied lighting conditions, and dynamic changes in the environment.

The introduction of depth data marked a significant advancement in corridor detection techniques. For example, Zhou et al. [106] demonstrated how depth cameras could be used for robust door and corner detection, enhancing corridor navigation. Methods such as Randomized Hough Transform [107] have been used to extract planes from depth images efficiently, aiding in the detection of corridor-like structures with high speed and accuracy. In [108], researchers

focused on creating a 3D model of hallways using stereo vision, where they analyzed depth data captured from stereo cameras to reconstruct hallway geometries. Yet, these depth-based techniques not only require high-quality data and significant computational resources but also may not perform well in cluttered environments.

Exploring computational efficiency, Gupta et al. [109] utilized edge devices like Raspberry Pi for corridor segmentation, emphasizing computational efficiency and deployment on resource-constrained platforms. While this approach conserves computing resources, it may not adapt well to complex or frequently changing corridor layouts due to its reliance on basic edge detection and region classification.

Techniques like RANSAC [110] and 3D Hough Transform [111] offer detailed environmental reconstructions but are limited in cluttered environments where transient objects can obstruct the detection of corridor walls and also require significant computational resources. Similarly, projection of 3D Point Clouds into 2D occupancy maps [112] also faces challenges in these settings, as objects within the corridor can prevent the creation of clear corridor geometry.

In structured settings like mines or warehouses, methods like those used by Larsson et al. [113], which combine laser data with Hough Transform, are fast and effective. However, they, too, can struggle in less structured or dynamic settings. Similarly, real-time wall detection methods that use ultrasonic sensors and cameras, as presented by Saffiotti [114], depend heavily on the accuracy and range of the sensors, which can be compromised by environmental factors like ambient noise and surface materials.

In addition to the existing limitations, current methods often require corridor dimensions and the robot's pose to be known beforehand [105], [108], [114]. Addressing these challenges, we present a corridor identification method that leverages Point Cloud processing and Hough

Transform using depth information. Our technique extracts 2D occupancy maps, which are binary images, from a 3D Point Cloud at a user-defined region of interest. By overlaying slices of these binary images, we create a final map where lines indicating corridor walls are identified using the Hough Transform. This approach allows for dynamic detection of corridors without the need for prior knowledge of corridor dimensions or the robot's orientation.

Our method also simplifies the processing pipeline by reusing one of the occupancy slices for obstacle detection, which is used as an input for the Obstacle Detector module in Chapter 5.4. Moreover, unlike other Hough Transform-based methods, ours does not require complex edge detection algorithms like Canny [115] or Sobel [116], thereby reducing computational complexity and enhancing real-time processing capabilities. Lastly, the method does not require any expensive laser scanners or multi-sensors to achieve its goal. A single, low-cost, and off-the-shelf depth camera is sufficient to acquire the required data.

4.2 Corridor Detection

The Corridor detection process consists of two submodules depicted in Figure 13: Point Cloud Processor and Corridor Detector. The former submodule begins with capturing 3D Point Cloud data using a depth camera. Point Cloud is first filtered to remove irrelevant points outside a designated region of interest, then transformed to align with the robot’s coordinate system. Then, these points are projected onto 2D occupancy maps to create corridor and obstacle binary images. The latter submodule processes corridor images produced by the Point Cloud Processor. It applies Hough Transform to identify potential corridor lines, which are then analyzed to determine the corridor's geometric center line and right and left wall lines.

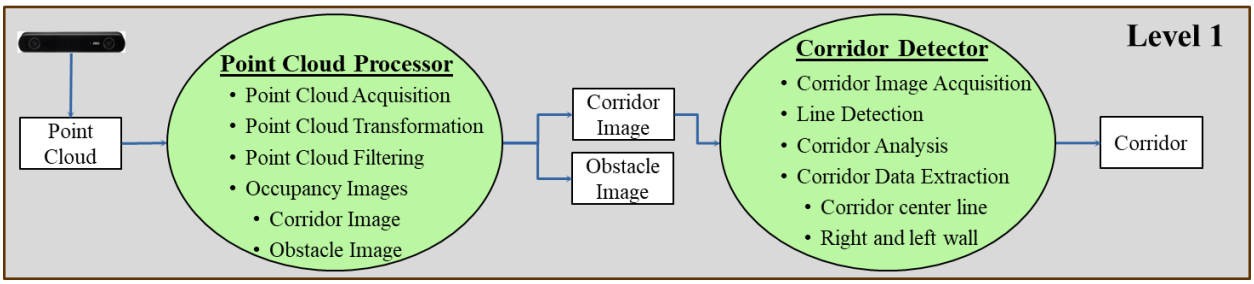

Figure 13. Overview of the corridor detection process.

4.2.1 Point Cloud Processor

This submodule receives Point Cloud data from the depth sensor and processes it to create a wall image and an obstacle image.

First, it captures dense three-dimensional Point Cloud data by subscribing a ROS message published by the depth camera's built-in software. Then, to enhance processing efficiency and focus on relevant data, the processor filters out extraneous information and ensures that only data points within a predefined region of interest (ROI) are retained for further processing. To achieve this, the Crop Box Filter from the Point Cloud Library [117] is applied to the points. The Crop Box Filter operates by defining a 3D bounding box with minimum and maximum limits along the x, y, and z coordinates. The filter can be mathematically represented as

$$Filtered\ Cloud = \left\{ p \in Raw\ Cloud : \begin{matrix} x_{min} \leq p_x \leq x_{max}\ and \\ y_{min} \leq p_y \leq y_{max}\ and \\ z_{min} \leq p_z \leq z_{max} \end{matrix} \right.$$

where p represents a point in the point cloud with coordinates (p_x, p_y, p_z), and $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$ are the coordinates that define the boundaries of the crop box.

The next step in the processor is coordinate transformation. The depth camera generates three-dimensional Point Cloud data in the camera frame. As we will construct our shared control scheme relative to the robot, the points need to be transformed into the robot's local coordinate system. When a camera is mounted to the robot with its optical axis parallel to the ground, the transformation from the camera's coordinate system to the robot's coordinate system involves both rotation and translation components. In this case, a point defined as $p_c = (p_{c\_x}, p_{c\_y}, p_{c\_z})$ in the camera coordinate frame can be transformed to $p_r = \left(p_{r_x}, p_{r_y}, p_{r_z}\right)$ in the robot's frame using the transformation matrix T:

$$p_r = T \,.\, p_c$$

where p_r, T, and p_c are defined as follows:

$$\begin{bmatrix} p_{r_x} \\ p_{r_y} \\ p_{r_z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & cos\,\theta & -sin\,\theta & T_y \\ 0 & sin\,\theta & cos\,\theta & T_z \end{bmatrix} . \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} p_{c_x} \\ p_{c_y} \\ p_{c_z} \\ 1 \end{bmatrix} \tag{7}$$

This equation applies rotation and translation defined by θ, which represents the orientation of the camera relative to the robot's base frame and $\left(T_x, T_y\ T_z\right)$, which is the physical offset between the camera and the robot's base frame.

Following coordinate transformation, Point Cloud data is segmented into multiple slices or layers, each corresponding to a horizontal cross-section of the environment at different vertical intervals. Each slice is converted into an occupancy grid where cells are marked as occupied or unoccupied based on the presence of point data within that slice. An occupancy grid, which is also

referred as occupancy map or occupancy image, O, is defined as a two-dimensional grid based on a user-defined ROI in front of the robot (Figure 14) with the following resolution:

$$width = \frac{(x_{max} - x_{min})}{grid_{size}}, \qquad height = \frac{(y_{max} - y_{min})}{grid_{size}} \tag{8}$$

where $grid_{size}$ is a user-defined parameter representing the size of each cell in meters. Similarly, $(x_{min},\ y_{min})$ and $(x_{max},\ y_{max})$ are the same user-defined ROI parameters used in the Crop Box filter above.

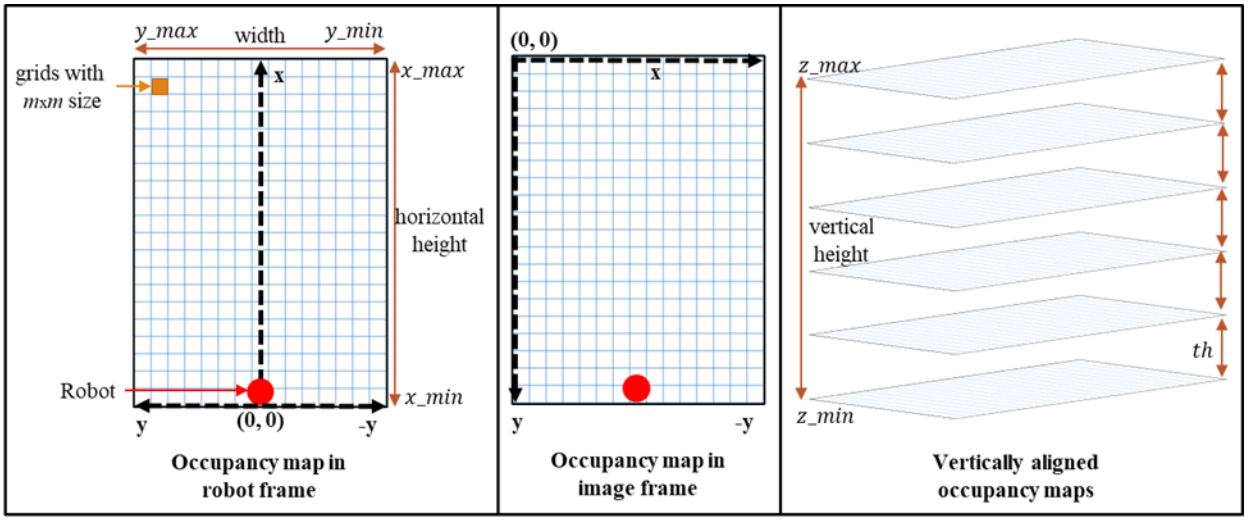

Figure 14. Occupancy map generation and set of vertically aligned maps.

If we project all the points in 3D space onto a single occupancy map and try to find walls, it would not be enough to discriminate corridor walls because of noise and objects inside the corridor. Therefore, we create layers of vertically aligned occupancy-maps in order to project points at the same level (slice) onto the same level map and then overlay these layers.

The points are sliced vertically based on a predefined slice thickness th. This results in N layered maps each representing a horizontal cross-section of the environment:

$$N = \frac{z_{max} - z_{min}}{th}$$

where z_{min}, z_{max} define the vertical limits of the ROI, correlating to typical corridor height and ground level, respectively.

Now that the ROI and Point Cloud slices are defined, and occupancy images are created, we continue with converting each slice into an occupancy image by projecting 3D points onto their corresponding 2D images. Each cell, $O(x, y)$, in these images is initialized as "unoccupied" and then updated based on the presence of points:

$$O(x, y) = \begin{cases} 1\ if\ occupied \\ 0\ if\ unoccupied, \end{cases}$$

The grid indices (x_{index}, y_{index}) of a point (x, y) on the image are determined using the following:

$$x_{index} = width - \left(\frac{\left(p_{r_x} - x_{min}\right)}{grid_{size}}\right) \quad (9)$$

$$y_{index} = height - \left(\frac{\left(p_{r_y} - y_{min}\right)}{grid_{size}}\right) \quad (10)$$

Once all points are projected, two images are extracted from the resulting maps: corridor image and obstacle image. The process of creating corridor image involves aggregation and thresholding (Figure 15):

- The sum of occupancies across all images for each cell is calculated by vertically aggregating the occupancy data from all layers and represented as:

$$S(x,y) = \sum_{i=1}^{N} O_i(x,y) \tag{11}$$

- Finally, a threshold τ is applied to identify significantly occupied cells to obtain the corridor image, C:

$$C(x,y) = \begin{cases} 1 & if\ S(x,y) \geq \tau \\ 0 & otherwise \end{cases} \tag{12}$$

In such a corridor image, C, because we have seen so many samples across the height, we assume that if we can find two lines, they would be walls of a corridor.

The obstacle image, on the other hand, can be created depending on specific application needs. As illustrated in Figure 16, obstacle image may utilize the first layer of occupancy images to capture ground-level obstacles. However, to address the limitations of other sensors, such as a 2D LiDAR, and to consider the robot's operational height and typical obstacle profiles, multiple layers of occupancy maps may be aggregated.

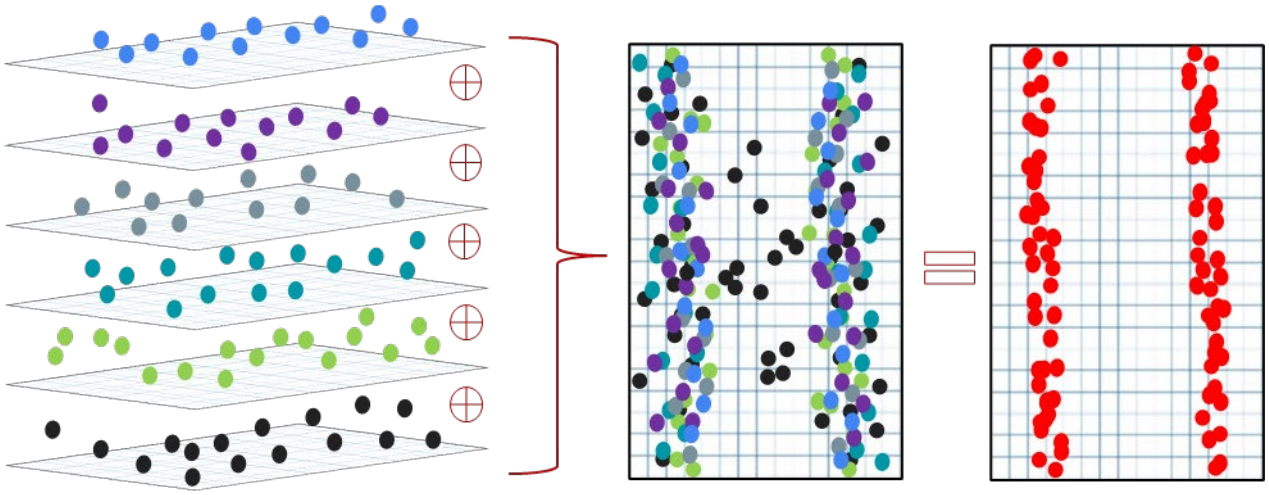

Figure 15. Wall image generation. Left: 3D Points and their matching occupancy map. Center: Cells vertically added up onto a final map. Right: Final map (image) after thresholding, representing the corridor wall.

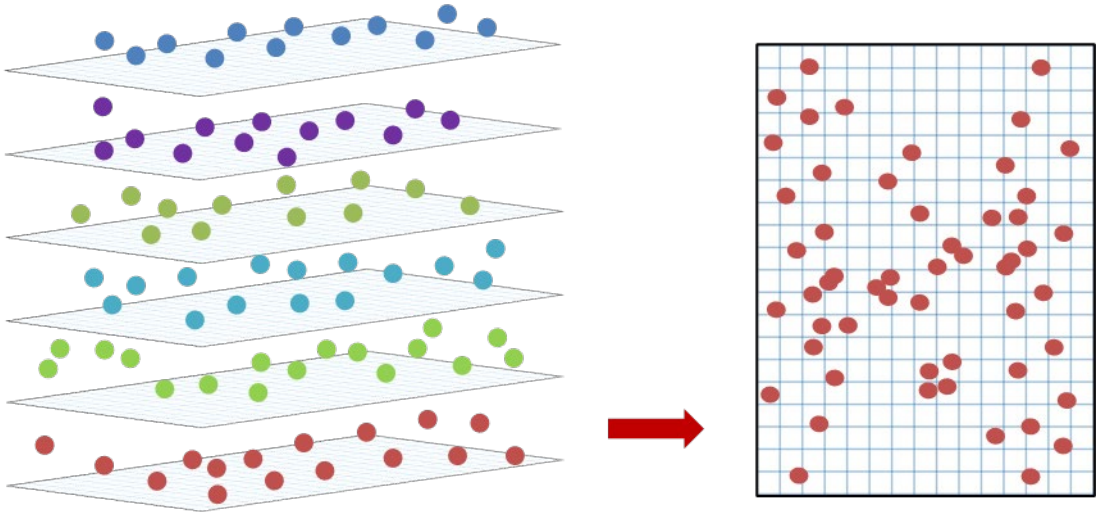

Figure 16. Obstacle image creation. Obstacle detection uses the first map from the ground floor.

4.2.2 Corridor Detector

Corridor detector analyzes the created corridor image, which is in fact a binary image representing occupancies, to find lines that match with the corridor walls. We use Hough transform on the resulting image to extract the lines of the corridor walls. Because our input is already a binary edge image, unlike most existing vision-based methods, we do not need to perform edge detection before the implementation of Hough Transform.

Hough transform [118] is typically used to detect lines and circles in an image. In parametric or normal form, a line can be represented as $\rho = x\cos\theta + y\sin\theta$ where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured counter-clockwise (Figure 17, left). Using two parameters in the line definition, Hough transform quantizes the parameter space into finite *accumulator cells.* Each point in Cartesian coordinates is transformed into Hough Space. The accumulator cells that satisfy the line equation are incremented. The cells with the maximum values yield the most prominent lines (Figure 17, Right).

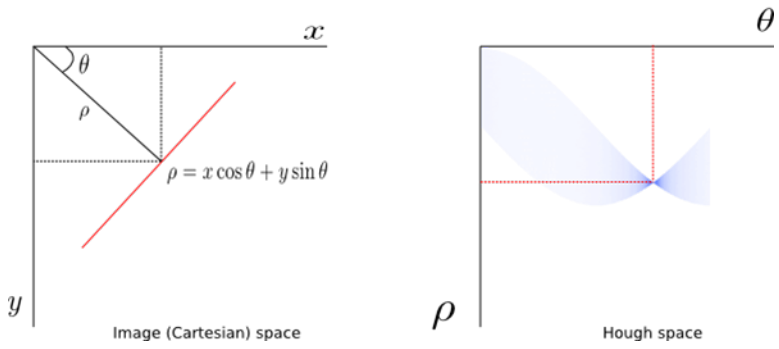

Figure 17. Line definition in image and Hough space.

In standard Hough Transform, all pixels in the image are transformed into the parameter space and then compared against the line equation. On the other hand, a variant of Hough Transform, Probabilistic Hough Transform, PTH [119] is a more efficient implementation of the regular Hough Transform. The PTH randomly selects a subset of pixels, maps them into Hough space, and then finds lines. While using such a subset helps reduce computational time, it does not sacrifice the accuracy in cases like ours. Therefore, we use this version of Hough Transform.

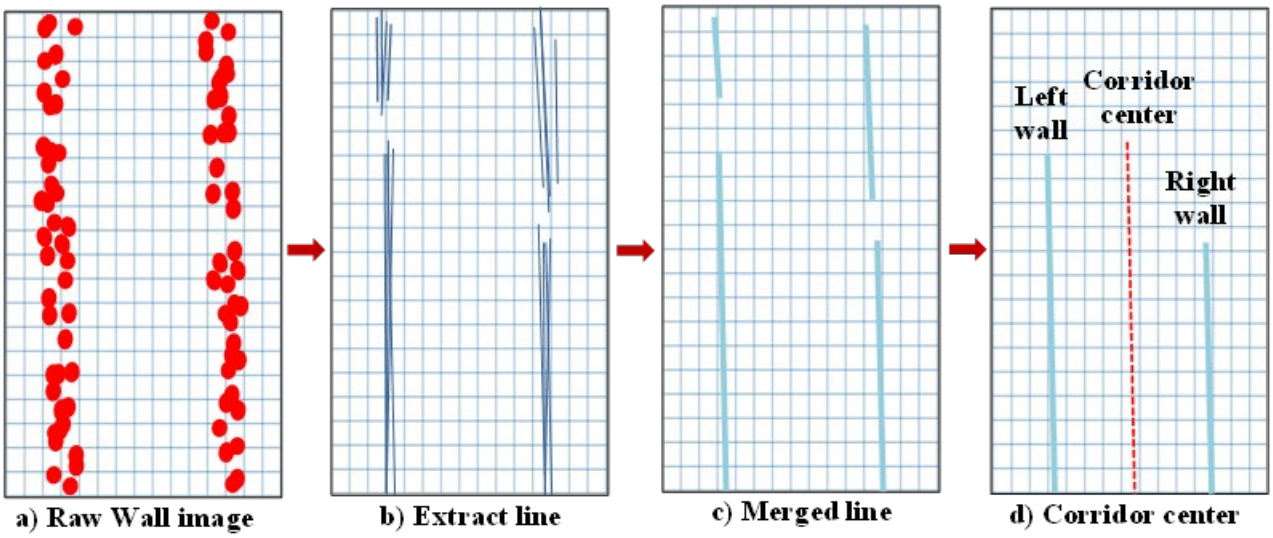

Figure 18. Application of the PTH on wall image and detection of corridor center.

We apply the PHT on the wall image (Figure 18-a) using an OpenCV library [120] that finds line segments in a binary image. Because of the noise in sensor readings, the resulting line segments do not perfectly match with the corridor walls (Figure 18-b). After merging the line segments based on the angle between them and the distance between centers of segments, we obtain a result that closely fits the corridor walls (Figure 18-c). However, because of the structure of a corridor, there may be openings, turns, office doors, etc. In such cases, the resulting line

segments may mislead the detection phase. Therefore, we further reduce the line segments by picking the only two closest to the robot. Moreover, we reject if these lines are not parallel, collinear, and their endpoints are located on the same side of the robot. Out of the two closest line segments, we extract the corridor center, which will be used as a guide point to control the robot for navigation in the corridor (Figure 18-d).

4.3 Corridor Detector Experimental Results and Discussion

We conducted a separate experiment specifically to demonstrate the performance of the corridor detector module. In this scenario, the robot autonomously finds and follows the center of the corridor. Here, our purpose is not to showcase the corridor-following behavior through joystick inputs but, rather, to highlight the efficiency of the corridor detector as the robot moves down the corridor.

The module was tested on an earlier version of our custom-built robot equipped with a Mynt-eye depth camera (Figure 19). This camera is mounted on the upper front of the robot and is fixed to look parallel to the ground. It features a field of view with dimensions D:121° H:105° V:58° and produces depth resolutions of 1280x720, generating point clouds at 10 Hz. All methodologies were implemented in a ROS environment utilizing OpenCV libraries. The robot ran on an Ubuntu 18.04 computer, powered by an Intel Core i5 CPU with 1.60GHz across 8 processors and 30 GB of memory.

Experiments were conducted in a corridor measuring 2.30 meters in width and 4.0 meters in height. We tested slice thickness values ranging from 10 cm to 50 cm. While smaller values increased the computational load, larger values reduced the effectiveness in detecting corridors. We discovered that a thickness of 30 cm provided the best balance of efficiency and effectiveness.

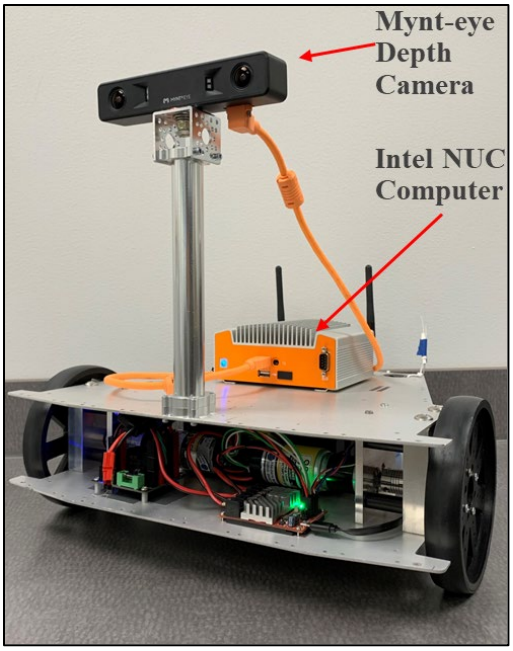

Figure 19. The robot base used in corridor detection experiments.

Objects were placed along the corridor to simulate realistic conditions. Snapshots from these experiments show the detection of the corridor and autonomous navigation within it (Figure 20). Initially, the robot, intentionally positioned slightly towards the left wall (Figure 20-a), had no prior knowledge of the environment or its location within it. By activating the wall detector, it successfully identified the corridor walls and adjusted its path towards the center of the corridor (Figure 20-b). Subsequently, it continued along the center of the hallway (Figure 20-c, d), consistently searching for the corridor. Figure 20-b, d illustrate the effectiveness of the proposed work in detecting corridor walls even at corners.

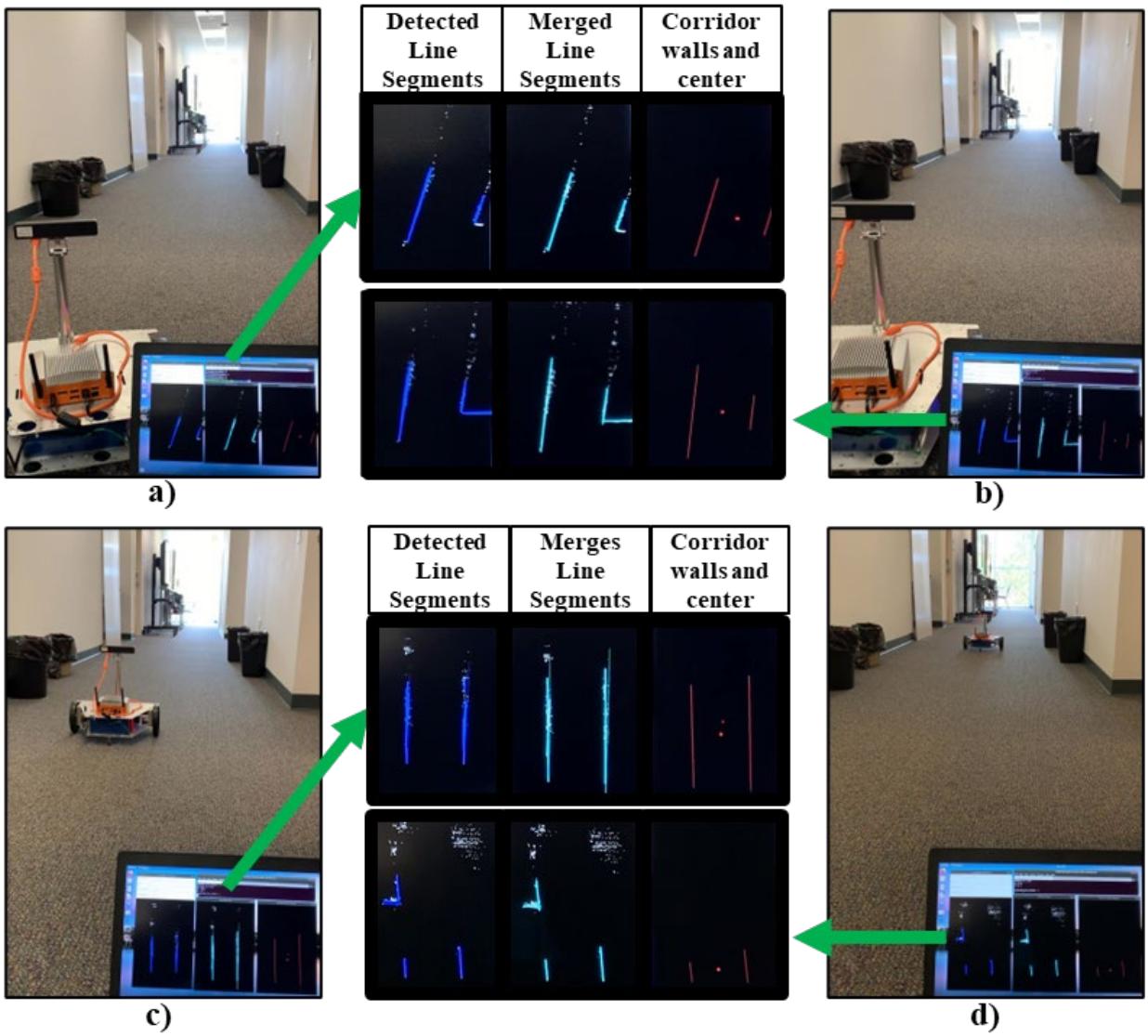

Figure 20. Corridor detection experiments: Obstacles were located next to walls.

In another case as shown in Figure 21-top, the corridor detector successfully identified the corridor center even though objects such as a chair and a trash can were located along the hallway. The bottom images in Figure 21 show the steps in determining the center of the corridor. The left-most image displays the occupancy image generated by Point Cloud Processor. The image with the blue line segments depicts line segments detected by Probabilistic Hough Transform. The image with green line segments is the result of merging the detected line segments from the

previous image. Lastly, the right-most image with red line segments shows the corridor walls and the center of the corridor.

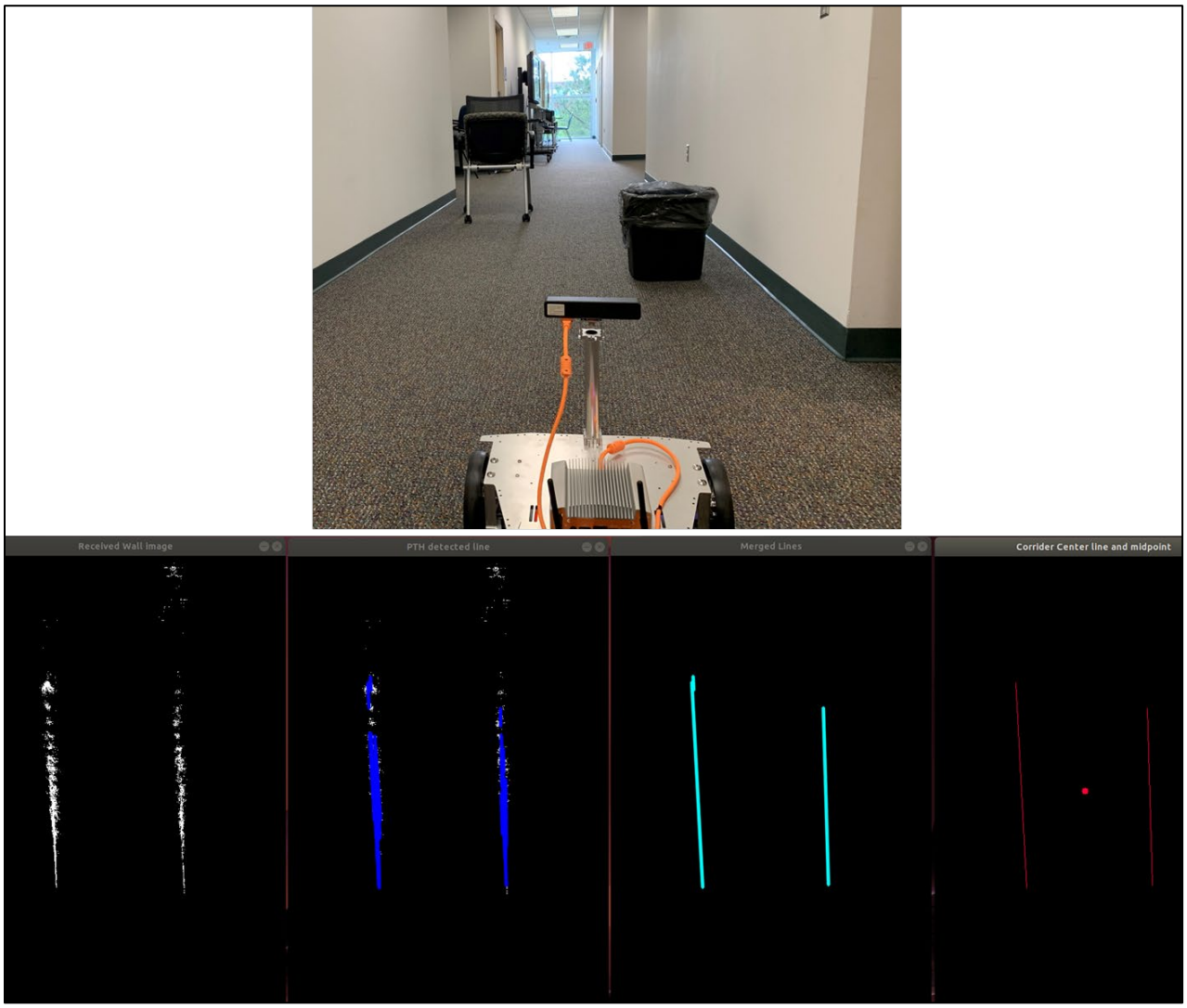

Figure 21. Corridor detection experiments: Obstacles were located along the hallway.

We also measured the elapsed time to run one iteration of each operation in corridor detection on the various computers. The specifications of the computers are given in Table 1.

Table 1. Test computers to run the corridor detector.

Computers	Processor	Memory
Raspberry Pi 4	Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	4 GB
Intel NUC	Intel Core i5-8265U CPU @ 1.60GHz x 8	30 GB
DELL	Intel Core i7-8260HQ CPU @ 2.90GHz x 8	15 GB

In terms of memory consumption and computational complexity, the most expensive tasks are executed in the first module, where the camera and point processor handles 3D points and creates 2D images out of these points continuously. Nonetheless, the results (Figure 22) show that the algorithm is efficient enough to run in real time on a Raspberry Pi.

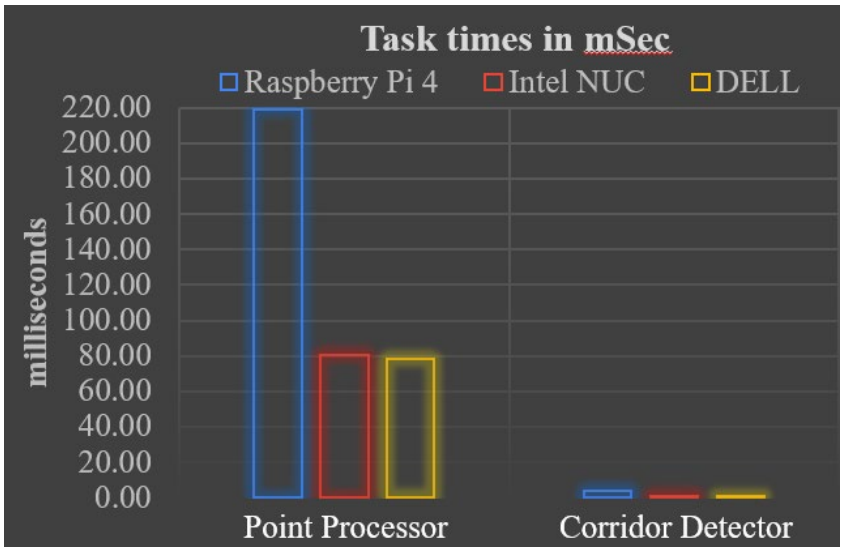

Figure 22. Run times for corridor detection processes.

The corridor detector module sends out the corridor data at a 10 Hz rate, which is fast enough

for real-time mobile robot navigation in blended control settings. The results show that the proposed work is resilient against items located in a corridor. The depth readings from the camera are vulnerable to the lighting conditions and reflections. For an online application in a real-life environmental condition, this is inevitable. Nonetheless, the presented method overcomes the noise issue and enables the robot to detect the corridor walls.

CHAPTER 6

ADDITIONAL SYSTEM MODULES

This chapter discusses the additional system modules that comprise the other primary functionalities of the proposed multilevel shared control system. The modules covered in this chapter include User Communicator (Comm), Control Arbitrator, Obstacle Detector, and Corridor Follower.

As depicted in Figure 1, our multilevel shared control system offers a user-centric control scheme, where a user is the main controller of the overall system. Using a control interface device, such as a tablet, the user communicates with the robot. They decide how to operate the robot and when to take over control using this interface. If a person opts to drive with a remote controller, transmitting joystick values from the interface suffices.

User Comm module is responsible for establishing connection between the robot and user through such a control device. When the user drives the robot using a joystick from the interface device, User Comm sends user inputs to **Control Arbitrator** module. These inputs include desired command velocities and control mode. The arbitrator receives these inputs, gives immediate control to the user, and depending on the control mode, it then forwards the arbitrator commands to the U-CenTB^2 module. U-CenTB^2 then takes care of the obstacles detected by **Obstacle Detector** module and passes safe drive commands a low motion controller.

During operation of the robot, if a corridor is identified and the user opts to engage **Corridor Follower** mode, Control Arbitrator allows Corridor Follower's commands pass to U-CenTB^2. Corridor Follower is a higher-level mode, receiving corridor message, obstacles, and user

commands. In this module, user commands and obstacles are handled differently to be able to smoothly go down in a corridor while avoiding collisions. Although Room Navigator and Person Follower are not implemented within the scope of this dissertation, their designed functionality would follow a similar integration pattern, waiting for user mode selection and then taking control as directed. At any time, the user can take over control from any of these modules and still drive safely with the joystick inputs.

5.1 User Communicator

5.1.1 Introduction

User Communicator module is the primary interface between a human operator and the robot within the proposed multilevel shared control system. Rather than only sending out driving commands, this module enhances the operator's situational awareness by providing intuitive feedback [121] when controlling a robot in a hospital-like environment. It acts as a bridge, establishing continuous communication between the robot and the user's input device such as a tablet configured to send data in the correct format.

Responsible for both transmitting desired user velocity and control mode messages to the robot and relaying the status messages from various system components back to the user (Figure 23), the communicator helps users be aware of the environmental conditions and the robot's operational status in real time.

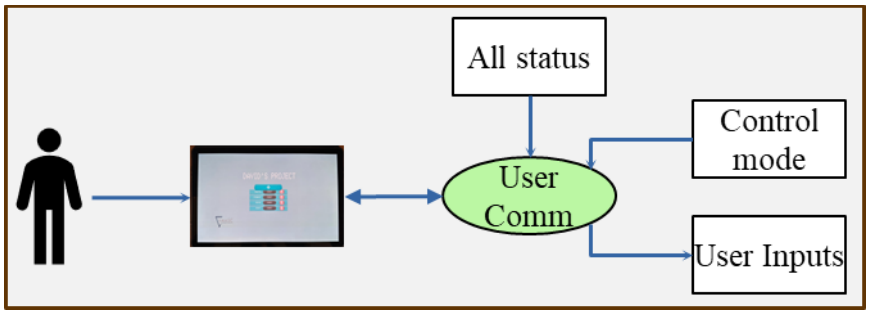

Figure 23. User Communication Module.

5.1.2 Communication Details

User Communicator operates as a ROS node on the robot, utilizing UDP (User Datagram Protocol) for fast and efficient communication between the robot and the user's input device. UDP protocol is chosen for its low overhead and quick message delivery capabilities, which are essential for real-time control of mobile robots.

The module initializes a UDP server on the robot, binding to a specific port to listen for incoming packets. These packets originate from a pre-configured tablet or smartphone application acting as the UDP client. The server processes different types of messages such as driving commands, system queries, and emergency stop signals. It ensures that only valid and authorized commands are executed by the robot.

Given the critical nature of its operation within hospital-like settings, User Communicator incorporates basic security measures to ensure that communications are protected against unauthorized access. For example, only devices with pre-approved IP addresses are allowed to communicate with the robot, minimizing the risk of unauthorized control commands. Furthermore, each received command is validated against a set of allowed commands and

parameters to prevent malicious or malformed inputs from affecting the robot's operation.

Last but not least, User Communicator provides feedback to the user about the robot's operational status and environmental conditions. This feedback includes:

- **Status Updates**: Regular updates on the robot's battery level, operational mode, and any fault conditions are sent to the operator's device.
- **Environmental Data**: The module also sends information from various sensors on the robot, like proximity sensors and cameras, to help the operator navigate and monitor the surroundings effectively.

5.1.3 Summary

User Communicator, as a ROS node on the robot, serves as an essential communication bridge between the robot and a human operator. It ensures that commands sent from the user's input device are received, authenticated, and executed securely and efficiently. The node's design prioritizes real-time responsiveness, reliability, and security. Future developments could include advanced security enhancements and more robust real-time communication protocols to further improve its functionality and reliability.

5.2 Control Arbitrator

5.2.1 Introduction

The purpose of Control Arbitrator is to manage switching of control authority between different modules within the proposed system. Functioning as a mutually exclusive switch, it ensures that only one module operates the robot at any given time.

As illustrated in Figure 24, the arbitrator receives user inputs and specific module

commands, such as person, room, or corridor commands. Based on the user's selected control mode and other criteria discussed later, it then sends command velocity messages to the U-CenTB^2 module and current control mode to User Communicator. The user, as the overall supervisor of the system, maintains ultimate control and can override or select the active control module at any time.

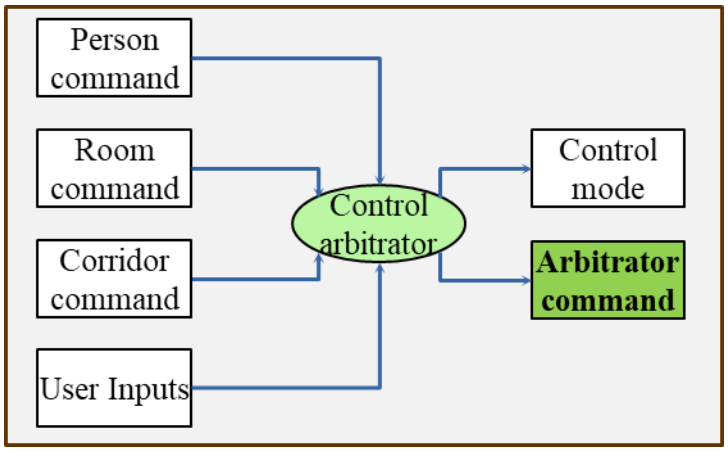

Figure 24. Overview of Control Arbitrator module.

5.2.2 Decision-making in the Arbitrator

Figure 25 details how the arbitrator continuously evaluates several key factors in the decision-making process to manage the flow and priority of control commands from various sources. The decision-making process begins with the arbitrator checking readiness of the robot and user. If either is not ready, the arbitrator issues a "Stop Command" to ensure the system remains in a safe state until all conditions for operation are met.

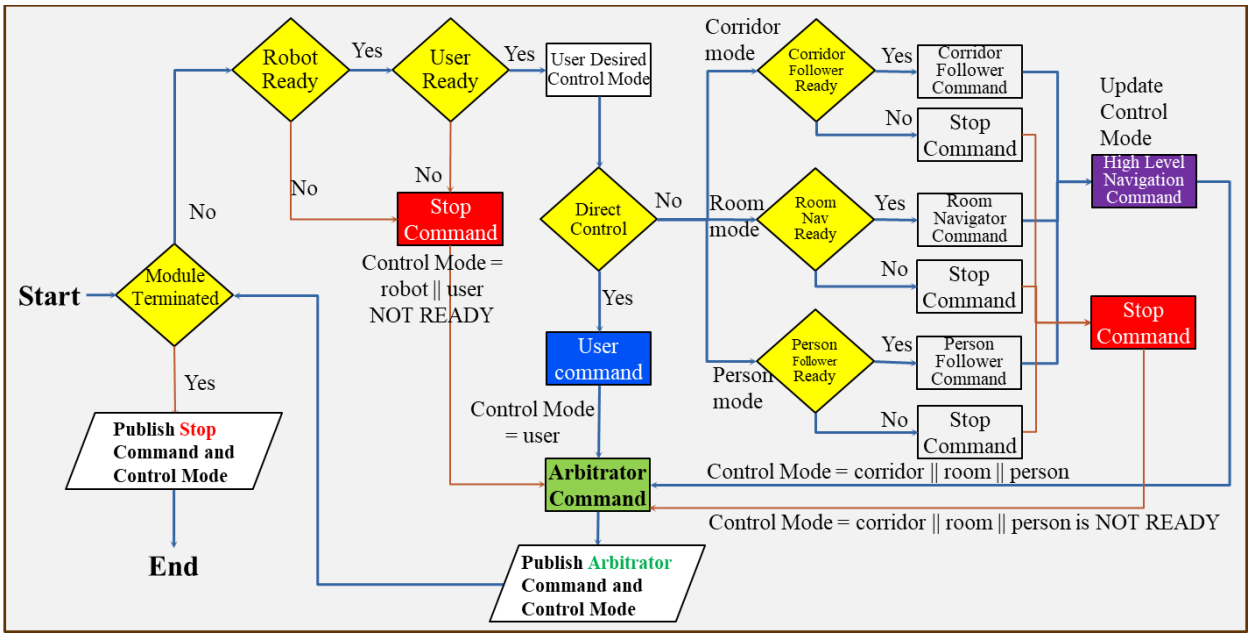

Figure 25. Decision making process in the Control Arbitrator.

Once the system confirms both the robot and user are ready, Control Arbitrator evaluates the user's desired control mode. Based on this preference, the arbitrator routes control commands:

- **Direct Control (Teleoperation):** If in direct control mode, the user's desired velocity commands are forwarded directly to the U-CenTB^2 module.
- **Module-Specific Control:** If a specific navigation mode is selected and the respective module (like Corridor Follower, Room Navigator, or Person Follower) is ready, velocity commands from that module are sent to U-CenTB^2. If the selected control module is not ready, a "Stop Command" is issued to prevent unintended actions. The implementation of Room Navigator and Person Follower is currently beyond the scope of this dissertation but designed to be easily integrated into the system in the future.

During teleoperation of the robot, the arbitrator continuously monitors each module's

status and switches control based on the current mode and system conditions. Besides managing the velocity command, it updates current control mode back to the user interface through User Communicator, keeping the human operator informed about who is controlling the robot, which helps the improve user's awareness and system transparency [122].

5.2.3 Summary

Control Arbitrator serves as a central decision-making entity within our proposed multilevel shared control system, effectively managing the transition of control between various operational modules and the human operator. In addition to providing effective control transitions based on desired user commands, the module also prepares the system for future expansions by allowing easy integration of additional modules like Room Navigator and Person Follower.

5.3 Corridor Follower

5.3.1 Introduction

The Corridor Follower module operates within Level-1 of our multilevel shared control system (Figure 26). It receives ROS messages on detected corridors and obstacles from Corridor Detector and Obstacle Detector, respectively. This information, combined with user inputs, allows the module to make real-time corridor-specific motion commands, providing an intuitive operating experience within these environments.

The outputs of the module are command velocity messages that go to the Control Arbitrator module and status information received by user. If the user enables this control mode, then Control Arbitrator forwards its commands to the U-CenTB^2 controller. In this configuration, U-CenTB^2 acts as an additional safety guard since obstacles were already taken

care of by Corridor Follower.

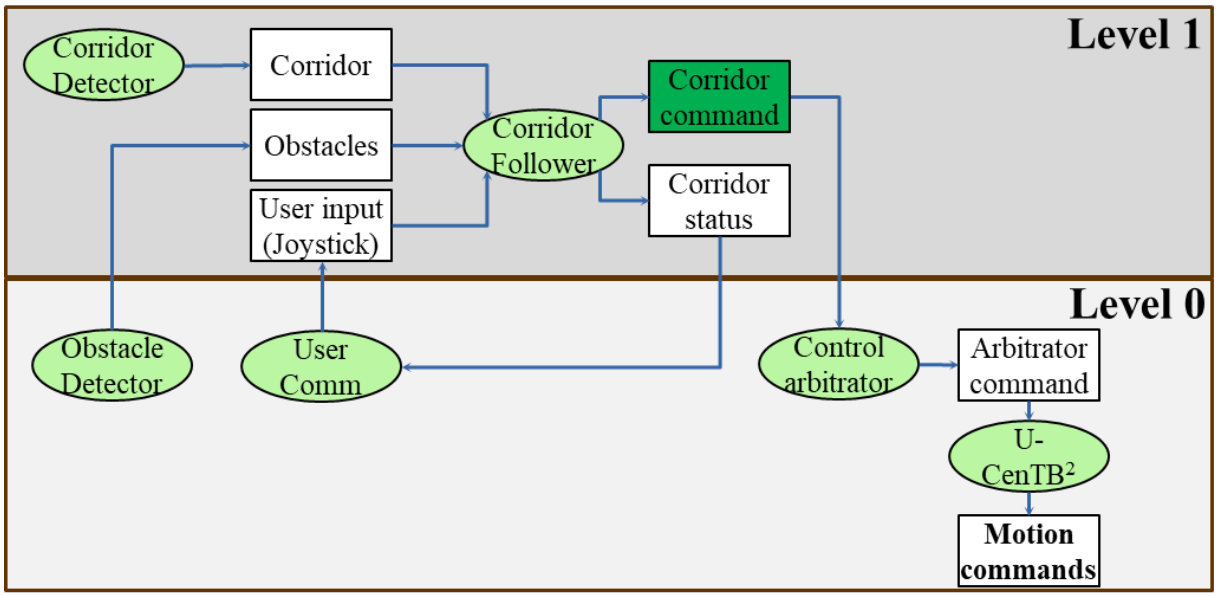

Figure 26. Corridor Follower in the overall system.

In this module, our goal is to enable the robot to navigate around obstacles while maintaining the intended direction of the user. This is particularly challenging in environments without a global map. By leveraging corridor-like structures, we provide the robot with contextual understanding, enabling it to distinguish between simple turning commands and more complex navigational adjustments. For example, a "left" joystick command does not merely indicate a mechanical turn but instructs the robot to shift to the left lane within the corridor. This functionality enables the robot to continue moving forward in the desired direction and avoid obstacles intuitively without random stops or direction changes.

5.3.2 Related Work

Navigation in corridor-like environments is usually studied for autonomous robots. Several approaches have been developed over the years focusing on different aspects such as visual odometry, sensor fusion, reinforcement learning, and social navigation. In [123], Zhou et al. proposed a visual potential field method for corridor navigation and obstacle avoidance. This method uses visual information to generate potential fields that guide the robot's movements, allowing it to navigate corridors and avoid obstacles. Similarly, [124] explored visual odometry for mobile robots, demonstrating how visual information can be used to improve localization and navigation in corridors. However, their reliance on visual data and the robot's autonomy as a decision-maker makes them vulnerable to challenges in the environment such as lighting conditions or objects within a corridor. If, for example, the robot cannot detect the corridor, it will be stranded in the hallway, which is an undesirable situation especially in places like hospitals.

In terms of sensor fusion, Howard et al. [125] proposed a method for combining information from multiple sensors to improve accuracy and robustness of mobile robot navigation in corridor environments. Their approach highlights the importance of integrating data from various sensors to enhance the perception and decision-making capabilities of mobile robots.

Reinforcement learning has emerged as a powerful tool for training robots to navigate complex environments. Park et al. [126] presented a learning-based approach to improve multi-robot hallway navigation, focusing on optimizing both efficiency and safety without tuning internal parameters. This approach leverages decentralized learning to enhance navigation in narrow spaces, making it adaptable to various environmental layouts.

Nonetheless, it requires large training data as well as multiple robots.

Similarly, Sharma and How [127] introduced a socially acceptable planner for high-speed ground robot navigation in crowded hallways. Their planner aims to balance robot speed and human comfort by executing "peek-and-pass" maneuvers to avoid the "robot freezing problem" commonly encountered in dynamic environments. Although their work helped improve navigation in hallways, the methodology does not work without a global map.

To enhance corridor navigation for shared control of mobile robots, existing research is limited. Millán et al. [128] investigated the use of brain-computer interfaces (BCIs) for controlling mobile robots in corridor environments, demonstrating the feasibility of using BCIs for intuitive robot control. They mapped user's mental states to high-level commands such as "Forward, Left Turn, Right Turn". Even though their work primarily focuses on identifying mental states without specific details on the actual control algorithm, it shows how associating a user's intention with high-level guidance commands improves driving a mobile robot in corridor settings.

Although the body of research on corridor navigation and obstacle avoidance for autonomous mobile robots is rich, the literature is limited in the field of teleoperation, specifically shared control of indoor mobile robots. Our proposed corridor follower technique builds on these foundations by effectively utilizing real-time sensor data from LIDAR and depth cameras to continuously adjust the robot's path, ensuring alignment with the corridor and intuitive obstacle avoidance based on user commands.

5.3.3 Corridor Follower Algorithm

Upon receiving relevant data, our algorithm evaluates user commands based on the

criteria explained below. Depending on the operator's desired behavior, it generates motion commands to drive in a corridor. The workflow in the algorithm is as follows:

- Determine robot direction.
- Determine intended user action.
- Execute intended action.

5.3.3.1 Robot's Relative Direction in a Corridor

In order to interpret a user action from the given velocity commands, we need to know the orientation of the robot and corridor. Without a global map, the robot must figure out its understanding of orientation, i.e., its direction, in a corridor. To achieve this, we determine the robot's direction (d_r) based on the angular difference (θ_d) between its current heading (θ_r) and corridor orientation (θ_c). Then, θ_d is normalized to make sure it remains within the range of $[-\pi, \pi]$:

$$\theta_d = \theta_r - \theta_c;$$

$$\theta_d = \theta_d - 2\pi \left[\frac{\theta_d + 2\pi}{2\pi}\right],$$

Using θ_d, the robot's direction (d_r) is classified into one of four possible states:

$$d_r = \begin{cases} FORWARD, & |\theta_d| \leq \frac{\pi}{4} \\ LEFT, & \frac{\pi}{4} < \theta_d < \frac{5\pi}{4} \\ RIGHT, & \frac{-5\pi}{4} < \theta_d < \frac{-\pi}{4} \\ BACKWARD, & |\theta| \geq \frac{5\pi}{4} \end{cases} \tag{13}$$

In such classification, FORWARD direction indicates that the robot's heading is nearly aligned with the corridor's orientation, while LEFT or RIGHT indicates significant deviations to either side relative to the corridor's forward direction. It is important to note that the robot is assumed to not move backwards. Therefore, even if $BACKWARD$ classification is shown here, its corresponding action will not be discussed in this dissertation.

5.3.3.2 Intended User Action

Once the robot's direction is determined, the algorithm translates desired velocities into high-level actions. We assume that the user is pursuing one of five actions when operating a robot in a corridor:

GO_FORWARD: Go down the corridor.

ROTATE_LEFT: Turn left on the spot.

ROTATE_RIGHT: Turn right on the spot.

MOVE_LEFT: Shift left within the corridor while moving forward.

MOVE_RIGHT: Shift right within the corridor while moving forward.

EXIT: Exit the corridor to enter an opening.

STOP: Stop all movement.

These actions are derived using input velocities and the robot's direction:

$$Desired\ action = \begin{cases} GO\_FORWARD, & if\ \vartheta < and\ |\omega| < \omega_{threshold} \\ ROTATE\_LEFT, & if\ \vartheta = and\ \omega > \omega_{threshold} \\ ROTATE\_RIGHT, & if\ \vartheta = and\ \omega < -\omega_{threshold} \\ MOVE\_LEFT, & if\ \vartheta > and\ \omega > \omega_{threshold} \\ MOVE\_RIGHT, & if\ \vartheta = and\ \omega < -\omega_{threshold} \\ EXIT, & if\ (\vartheta < and\ |\omega| < \omega_{threshold}) \\ & and\ (d_r = LEFT\ or\ d_r = RIGHT) \\ & and\ Open = true \\ STOP, & otherwise \end{cases} \quad (14)$$

where ϑ and ω are linear and rotational velocities, respectively; $\omega_{threshold}$ is a predefined threshold to decide turn actions; d_r is the robot's direction relative to the corridor, which is obtained using Equation (13); $Open$ is an indicator showing whether there is an opening such as a room, intersection, T-junction, or corner.

Go Forward

For GO_FORWARD action, initially, the algorithm determines a goal point, which serves as the robot's temporary navigational target. This goal is derived from the midpoint of the corridor line, representing a virtual guide that the robot follows to stay aligned within a corridor. Then, appropriate velocity commands are calculated to reach this goal position.

Upon detecting an obstacle directly in its path, the robot calculates a temporary lateral left or right shift in the corridor depending on the obstacle's position relative to the robot. After successfully navigating past the obstacle, the robot realigns itself with the corridor's centerline. This requires recalculating the corridor's orientation and adjusting the robot's path to converge back to the centerline, effectively "resetting" its trajectory towards the set goals.

Move Left and Move Right

MOVE_LEFT and MOVE_RIGHT actions enable the robot to dynamically adjust its position within the corridor. Here, we calculate necessary lateral shifts by determining the corridor's orientation and distances to corridor walls. By adjusting the endpoints of the corridor center line based on vertical displacement, the robot can smoothly shift left or right while continuing to move forward.

For instance, in the MOVE_LEFT, the robot calculates the corridor's orientation and the distances to the left and right walls. It then determines the vertical displacement required to shift left while maintaining a safe distance from the walls to ensure the robot only shifts when

it is safe to do so.

Exiting the Corridor

When the robot needs to exit the current corridor, linear speed is reduced to half of the desired speed to ensure a smooth and controlled exit. The angular velocity remains unchanged to allow the robot to navigate the turn or exit accurately.

Stop

STOP action in the Corridor Follower algorithm is responsible for halting the robot's movement whenever necessary. In the follower, this action is invoked under several conditions:

- Timeout: If the system does not receive any user input for a predefined duration, indicating a loss of control signal or user intervention, the robot stops.
- User input: If the user still has connection to the robot but does not send any velocity commands, the robot stops.
- Invalid user input: When user input does not correspond to a valid movement command or if the robot's direction is uncertain, the robot stops.
- Obstacle Avoidance: If an obstacle is detected and the robot cannot safely move left or right to avoid it, the robot stops.

5.3.4 Summary

The Corridor Follower module is designed to simplify the operation of a mobile robot within corridor environments by mapping joystick inputs to high-level navigational commands. It relieves users of low-level controls when navigating a robot within a corridor by automatically adjusting the robot's movements in response to the corridor's layout and any

obstacles. The corridor detection and following algorithm detailed in this dissertation serves as an auxiliary component of the broader navigation system. Although the main focus of the dissertation is on collision avoidance in open spaces, this module is included to demonstrate its additional benefits and the feasibility of integrating such a system for enhanced corridor navigation. It represents an opportunity to expand the system's capabilities, suggesting how future work could seamlessly incorporate this technology to improve overall navigational guidance within structured environments.

5.4 Obstacle Detector

5.4.1 Introduction

The Obstacle Detector module within our multilevel shared control system employs an image-based approach similar to the one we developed and published in [129]. In that study, we utilized raw LIDAR data and point cloud information, projecting these onto an occupancy map to efficiently identify obstacles without relying on AI-driven models. Obstacle detection is crucial for ensuring safe and efficient operation of mobile robots, particularly in dynamic and unstructured environments like hospitals. In the context of teleoperated robots with shared control, this becomes even more critical as both human operators and autonomous algorithms contribute to navigation decisions. Traditional approaches to obstacle detection often rely on pre-built maps or require significant computational resources for real-time perception, which may be impractical in shared control systems where quick response times are vital. To address these challenges, our system utilizes real-time sensor data from a 2D LIDAR and a depth camera to detect and avoid obstacles.

Obstacle Detector analyzes images from two types of sensors: a 2D LIDAR and a depth

camera. The LIDAR provides a 2D view, primarily capturing data in the horizontal plane which does not include obstacles that may be above or below this plane. The depth camera covers a different part of the spatial environment but with a limited field of view. By merging data from both sensors, the detector achieves more comprehensive coverage of the surrounding area.

The obstacle detection process involves converting sensor readings into images, which are then analyzed to detect potential obstacles (Figure 27). Specifically, the obstacle image from the depth sensor is outputted by Point Cloud Processor, as detailed in the Corridor Detector chapter. In this section, we will briefly review existing obstacle detection methods, explain how LIDAR scans are converted into a binary image, and then provide details about the obstacle detection process from these obstacle images.

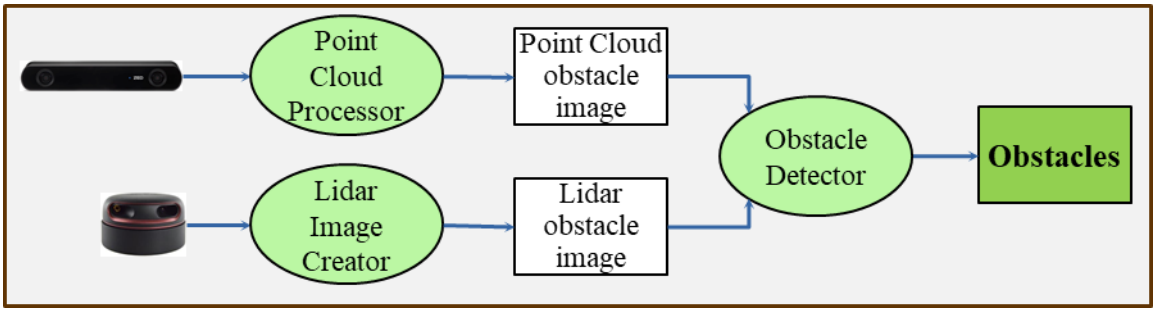

Figure 27. Inputs and output of Obstacle Detector Module.

5.4.2 Related Work

A variety of sensor modalities are employed for obstacle detection, each with unique strengths and weaknesses. Early approaches often utilized simple range sensors, such as

ultrasonic or infrared sensors, to detect objects in the robot's immediate vicinity [130], [131]. While these methods are computationally efficient, they provide limited information about the environment and may struggle with complex obstacle shapes or cluttered environments.

LIDAR sensors have become increasingly popular for obstacle detection due to their ability to provide precise range measurements. They can be categorized into 2D and 3D variants. 2D lidar sensors, which scan a single plane parallel to the ground, are commonly used in mobile robots due to their affordability and ease of use [132], [133], [134], [135]. Several algorithms have been developed specifically for 2D LIDAR, including occupancy grid mapping [136] , where the environment is discretized into cells, and the probability of occupancy is estimated based on sensor readings. Other approaches, such as polar histogram methods, utilize the polar representation of LIDAR data to efficiently identify obstacles and their boundaries [137]. While 2D LIDAR offers advantages in terms of cost and processing speed, its limited vertical field of view may result in missed obstacles or underestimation of their size.

3D LIDAR sensors, on the other hand, capture a more complete 3D representation of the environment. Methods like Iterative Closest Point (ICP) algorithm are often used with 3D LIDAR data to accurately map the environment and detect obstacles [138], [139], but the computational complexity of 3D LIDAR processing can be a bottleneck for real-time applications, especially on resource-constrained mobile platforms.

Vision-based systems, using monocular, stereo, or depth cameras, are another common method for obstacle detection. These systems offer rich visual data that can be processed using a variety of techniques [140], [141], [142]. Deep learning-based methods, such as Convolutional Neural Networks (CNNs), have shown impressive results in object detection

and classification tasks, enabling real-time and accurate detection of a wide range of obstacles [143], [144]. However, they often require significant computational resources and large amounts of annotated training data.

The fusion of multiple sensor modalities can further enhance obstacle detection capabilities. By combining information from different sensors, such as 2D LIDAR and depth cameras, the system can overcome the limitations of individual modalities and achieve a more comprehensive understanding of the environment. Nonetheless, this requires fast and efficient techniques for teleoperation of mobile robots. In this sense, image processing techniques can extract meaningful obstacle information from multiple sensor data. Morphological operations, such as dilation and erosion, are commonly used to enhance object boundaries and reduce noise [145]. Contour detection algorithms are effective in identifying object outlines from binary images, enabling the segmentation of obstacles from the background. To simplify obstacle representation and facilitate collision avoidance, these contours can be approximated with polygons or rotated rectangles.

5.4.3 LIDAR Image Creation

The creation of LIDAR images is handled by Lidar Image Creator, a separate ROS node. This submodule converts raw 2D LIDAR data into binary occupancy images. A binary occupancy image, as previously described in the Corridor Detector chapter, is a 2D grid representation of the environment that indicates whether each cell is occupied (1) or unoccupied (0). These images are generated by mapping LIDAR scans onto a grid, associating each measurement with its corresponding cell. In our algorithm, a binary image is generated based on a user-defined ROI around the robot, using the same values as in Corridor Detector.

The process in the module starts with receiving LIDAR scan messages in the Polar coordinate system and then converting them into Cartesian coordinates to determine x and y positions in the LIDAR sensor's frame using the equations:

$$x_{lidar} = r.\cos(\phi);$$

$$y_{lidar} = r.\sin(\phi),$$

where r is the range, and ϕ is the angle of the LIDAR beam. In our setup, the sensor is located at the robot's base. Therefore, there is no further coordinate transformation required at this point.

Next, using Equations (9) and (10) from Chapter 4.2, cartesian coordinates are mapped to grid cells in the image. Cells are marked as occupied if they contain obstacles or unoccupied otherwise, creating a binary image.

5.4.4 Obstacle Detection Process

The obstacle detection process, which runs in a ROS node, involves several steps to analyze the binary occupancy images generated from LIDAR and the depth camera. Figure 28 outlines the overall process.

Receiving and Merging Images

First, binary images generated by Lidar Image Creator and Point Cloud Processor are subscribed. These images are then merged to create a comprehensive obstacle image. Combining LIDAR and depth camera images involves overlaying the binary images to form a unified obstacle image. It is important to note that this process is not true sensor fusion but a practical approach to leveraging the strengths of both sensors. The LIDAR sensor provides fast and accurate data, while

the depth camera covers different spatial areas. By summing the images, we can quickly and efficiently detect obstacles without the computational overhead of a full sensor fusion process.

If I_{LIDAR} and I_{depth} are the binary images from LIDAR and depth sensors, respectively, the merged image, I_{merged} is given by:

$$I_{merged}(x, y) = I_{LIDAR}(x, y) + I_{depth}(x, y) \tag{15}$$

where (x, y) are the pixel coordinates. If only one of these images is ready by the time the function summing these images is called, then I_{merged} is set to the available image.

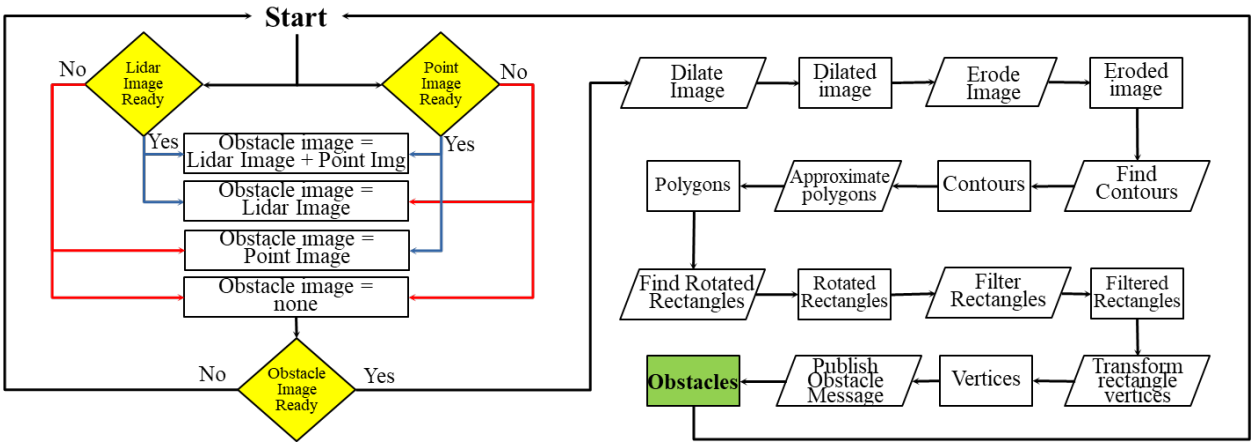

Figure 28. Detailed process flow in the obstacle module.

Morphological Operations

LIDAR data contains noise due to sensor limitations or environmental factors. After creating the binary image in the previous section, it is necessary to perform morphological operations to remove small, isolated noise points and obtain a cleaner binary image before proceeding with

further analyses.

Morphological operations are a set of mathematical functions, known as non-linear filters in image processing. Two basic morphological operators are Dilation and Erosion. Dilation expands the boundaries of objects, while erosion removes pixels from object boundaries. We use OpenCV [120] libraries to perform Dilation and Erosion. These operations are performed on binary images using a small binary filter or kernel known as a structuring element. The structuring element scans the image and modifies the pixels based on its size and shape. Commonly used shapes for structuring elements include rectangles, ellipses, and crosses, as depicted in Figure 29, which showcases 5x5 structuring elements with different shapes.

Rectangle 5x5				
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Ellipse 5x5				
0	0	1	0	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	0	1	0	0

Cross 5x5				
0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

Figure 29. Structuring element shapes of size 5x5.

To achieve cleaner binary images, it is crucial to analyze various sizes and shapes of structuring elements while considering the order of applying dilation and erosion. In Erosion, size of the element determines the extent of shrinking performed, with larger shapes resulting in greater shrinkage. In Dilation, as structuring element increases in size, resulting areas of the objects also become larger, and isolated islands of pixels also increase in size. After experimenting with numerous operations, such as applying erosion followed by dilation or vice versa, using different structuring element shapes (rectangle, ellipse, cross), and sizes (2x2, 3x3, 5x5, 7x7, 9x9), we found

out that the following order of operations yielded the best results to detect obstacles:

- Dilation with a 5x5 rectangle-shaped element followed by,
- Dilation with a 9x9 rectangle-shaped element followed by,
- Erosion with a 7x7 rectangle-shaped element followed by
- Erosion with a 3x3 rectangle-shaped element.

With this order, we first group nearby areas into a single object and then remove isolated noisy regions. Also, using a larger structuring element for dilation allows for capturing neighboring pixels effectively, and a smaller structuring element for erosion helps preserve larger areas.

Contour Detection

Contours refer to the continuous curves or boundaries that delineate the shape of objects within an image. Similar to implementation of morphological operations, we retrieve contours with the help of an OpenCV function implementing the algorithm of [146]. To find contours using this method, we need to specify:

- *Contour retrieval mode*, which determines the hierarchical relationship between contours. Options include:
    - retrieving only the outermost contours,
    - retrieving all contours in a flat list,
    - retrieving all contours in a hierarchical tree structure,
- *Contour approximation method*, which determines how the contour points are approximated and compressed. One option is to compress horizontal, vertical, and diagonal segments into their respective end points. The other option is to store all the contour points without approximation.

Regarding the retrieval mode, we focus solely on the outermost contours since we do not

need to analyze parent-child relationships or inner object parts. As for the approximation method, although the second option preserves detailed contour information, it generates a large number of points, consuming more memory and slowing down subsequent processing. Since we did not observe significant improvements compared to the first option, we decided to compress the points.

Obstacle Representation and Publishing

The detected contours are then approximated with polygons by employing Douglas-Peucker's algorithm [147]. This algorithm simplifies a curve or polygon by recursively dividing it into smaller segments and then approximating each segment with a line. Therefore, reducing the number of vertices leads to computational efficiency and better generalization of the shape.

Next, we compute the minimum area rectangles based on the previously approximated contours using polygons. We utilize the 'minAreaRect' method implemented in OpenCV to find a rotated rectangle that completely encloses the input contours. This method computes the smallest rectangle that can contain the contour and returns the rectangle's center point, width, height, and rotation angle. Compared to other geometric shapes (Figure 30), the minAreaRect function ensures that the bounding box displayed in red in the left-most image in Figure 30 is the smallest possible, which helps in efficiently representing the obstacles.

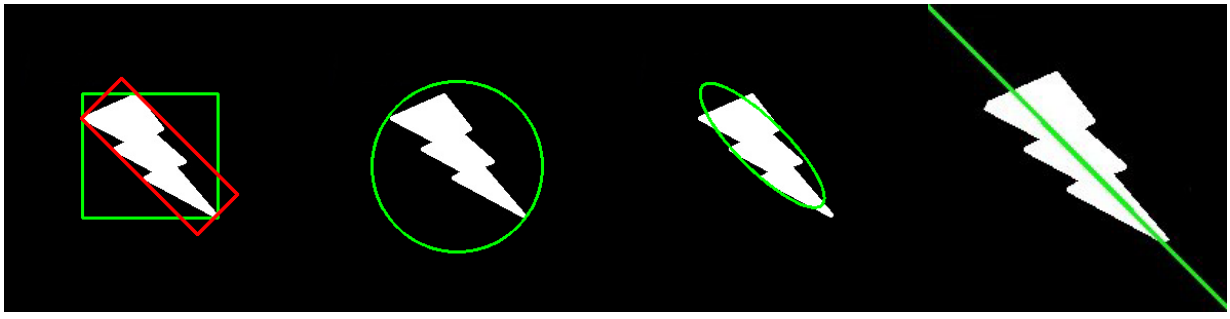

Figure 30. A comparison of shapes encloses contours. Image source [148].

Finally, instead of publishing all of the rotated rectangle information, which includes data we do not need, we extract the four vertices of each rectangle. These vertices are then transformed into the odometry frame and published as a ROS message.

Figure 31 provides an overview of the environment and visualization of the sensor data to showcase the obstacle detection process. The left image in the figure shows four objects with complex shapes, alongside the base of the robot model (white) within Gazebo simulator. The robot's forward direction is downward, and the robot's left side is to the right. The right image visually represents the sensor data, with red indicating LIDAR scans and white showing the processed (ROI extracted and transformed) Point Cloud in Rviz, a ROS-based visualization tool. The detected obstacles drawn from the vertices are outlined in blue.

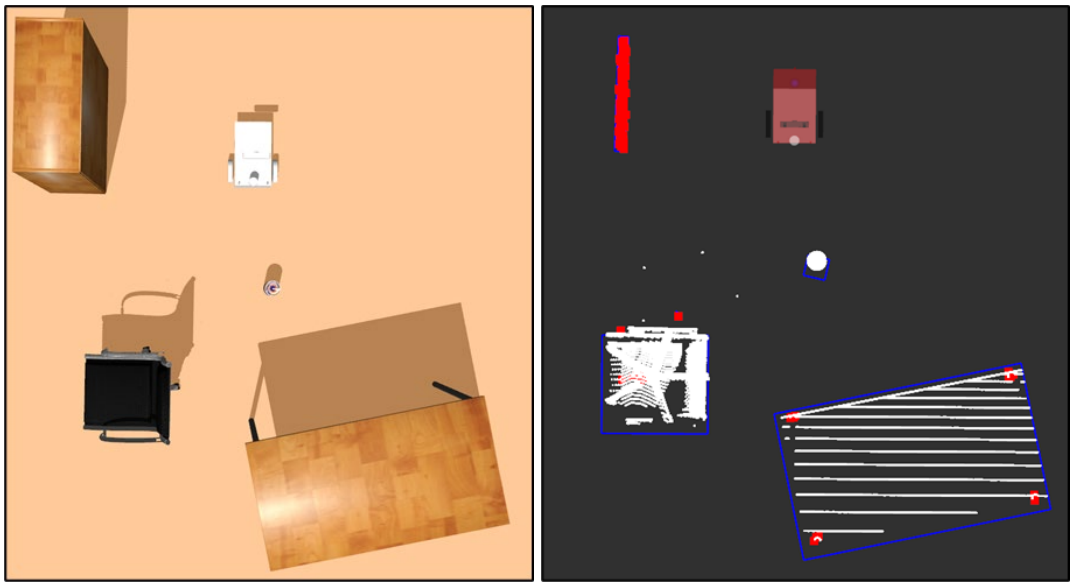

Figure 31.Visualization of detected obstacles from two sensors.

Figure 32 then illustrates each stage in the obstacle detection process. The LIDAR sensor

captures the bookcase on the robot's right and the legs of a table and chair, shown in red in the LIDAR Image. However, the can directly in front of the robot remains undetected by LIDAR (LIDAR image). The depth camera, meanwhile, captures this can and the table surface, although it fails to see the bookcase (Point Cloud Image). By overlaying the images created from both sensors, we achieve a complete representation of the scene (Overlayed Image). Subsequently, morphological operations are applied, contours are extracted, and finally, rotated rectangles are delineated around these contours (bottom right image in Figure 32). As discussed earlier, we extract, transform, and then publish the vertices of the detected rectangles. In fact, in Figure 32- the right image reconstructs these rectangles using only vertices for visualization purposes.

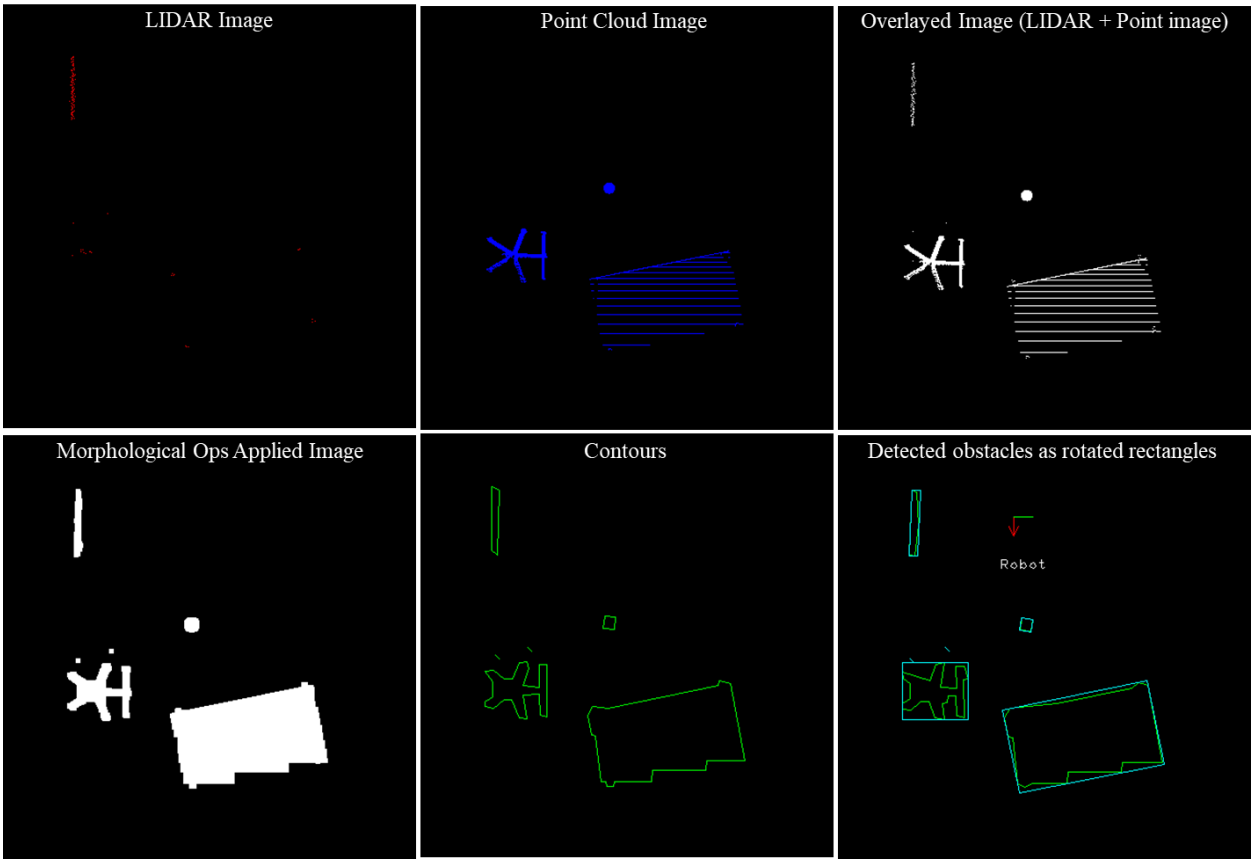

Figure 32. Illustration of steps obstacle detection process.

5.4.5 Summary

The obstacle detection process involves receiving and merging LIDAR and depth camera images to form a comprehensive obstacle image. Morphological operations are applied to enhance the image, followed by contour detection to identify the outlines of obstacles. These contours are then approximated with rotated rectangles, which are filtered for relevancy. Finally, vertices of detected rectangles transformed into the robot's coordinate system, is published for the control modules in the proposed multilevel shared control system. Future work includes integrating advanced sensor fusion techniques and incorporating a temporal aspect to the obstacle detection, such as an aging mechanism where obstacles can disappear or decrease in significance over time.

CHAPTER 6

A SIMULATION-BASED APPROACH FOR EVALUATING SHARED CONTROL ALGORITHMS FOR MOBILE ROBOTS

Performance evaluation of shared algorithms is challenging due to reliance on human feedback and limitations of physical test environments. To overcome these challenges, we propose a simulation approach for evaluating the effectiveness and efficiency of a shared control algorithm designed for mobile robots in realistic scenarios. A set of performance metrics is introduced to quantitatively assess the performance of the algorithm. Monte Carlo techniques are used to assess the robustness of the algorithm by running batch simulations in a fixed virtual world where both obstacle configurations and user inputs vary randomly across simulation scenarios. We evaluated the U-CenTB^2 algorithm under various experimental setups to showcase the effectiveness of the simulation approach and the algorithm. We aim to understand how well a shared control algorithm performs in environments that closely mirror real-world conditions while avoiding logistical constraints associated with human subject experimentation.

This chapter briefly discusses the experimental setup to assess the performance of shared control systems in real-world settings. Next, we present how we are replicating the actual blended system in a virtual environment, followed by a simulated experimental design. From there, a detailed explanation of our Monte Carlo approach and then synthetic user input modeling are provided. Then, we explain the details of running the simulations in batch mode.

6.1 A Typical Experimental Setup to Evaluate Shared Control Algorithms in Real-world

We illustrate a real-world experimental setup for performance evaluation of shared control algorithms in Figure 33. In this setup, human subjects are generally asked to drive the robot to specific locations using a remote controller. The environment where the robot operates is structured with randomly placed objects, mimicking potential real-world environments that the robot would need to navigate around.

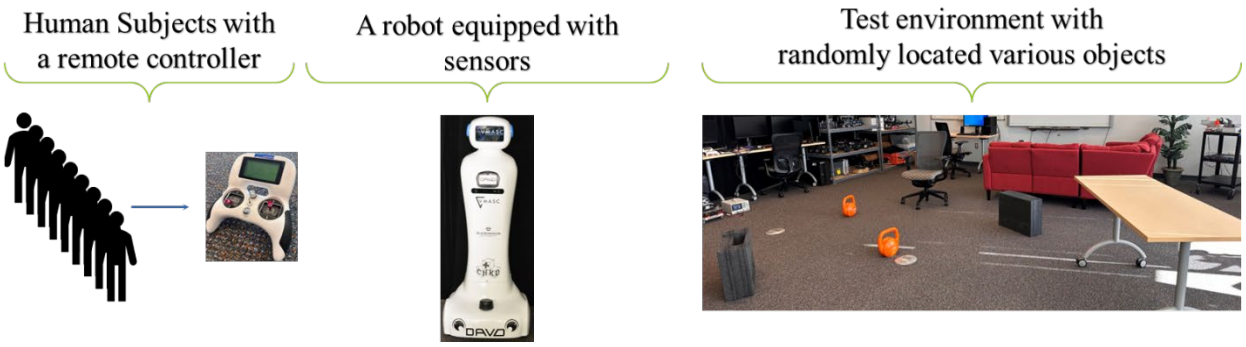

Figure 33. An Experimental Setup Example for Validation.

While this setup provides great insights into human-robot interaction and algorithm performance, it also presents several challenges. Human subjects introduce variability due to differences in reaction time, control preferences, and adaptability, which can lead to inconsistencies in data collection. Moreover, there are logistical complexities involved in recruiting and managing human participants, especially at an early design phase when repeated changes make it impractical to depend on human-based experimentation as a viable feedback mechanism. The test environment itself, despite being carefully arranged with obstacles, cannot fully replicate the unpredictability and diversity of real-world settings. The number and variety of

test scenarios are also limited by practical constraints such as space, time, and resources, potentially affecting the comprehensiveness of the validation process. These challenges underscore the importance of creating a versatile and controlled simulation environment to complement and enhance the validation of shared control algorithms.

6.2 The Proposed Simulation Approach for Evaluation of Shared Control Algorithms

In Figure 34, we present the simulated counterpart of a real-world shared control system. Unlike the real-world setup shown in Figure 3, user inputs here are generated by a simulation module instead of by a human operator, which will be discussed in detail in Section 6.2.4. Synthetic sensor data are created to emulate real sensor feedback from a robot's environment. These simulated inputs flow into a shared/blended control algorithm, just as they would in the actual scenario, and are processed to generate movement commands for the simulated robot.

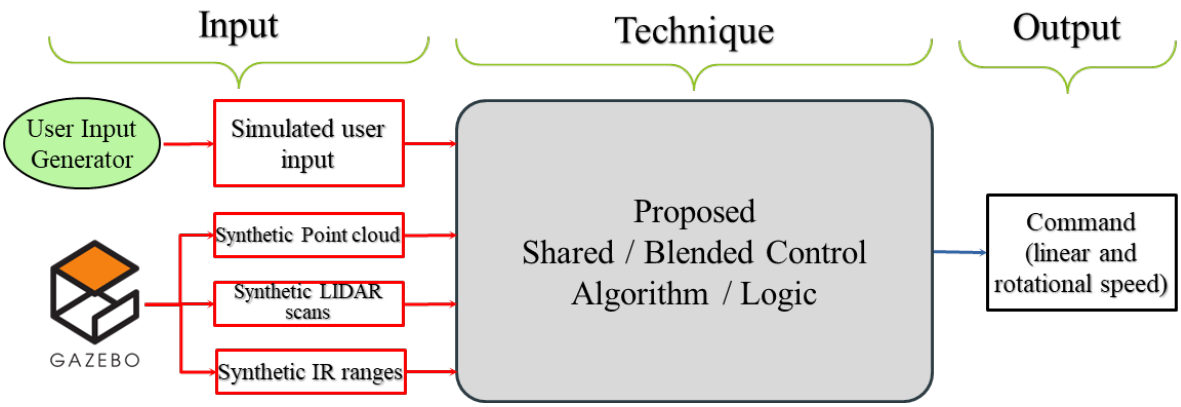

Figure 34. Simulation Design for Shared Control Algorithm.

An illustration of our simulation-based experimental setup is presented in Figure 35. Here,

a simulated robot was spawned into a room that has various objects. The robot is driven by blending the synthetic user inputs and control algorithm.

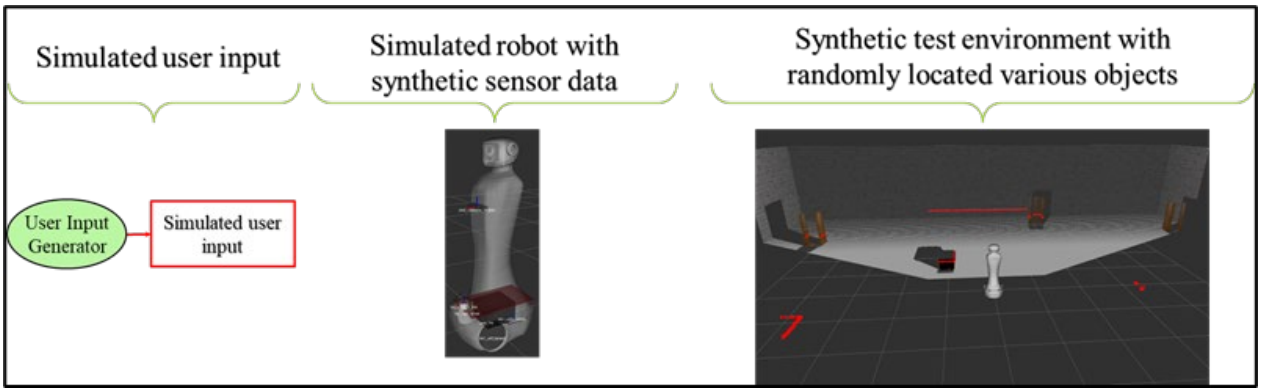

Figure 35. A depiction of our simulated experimental setup.

6.2.1 Monte Carlo Simulation Setup

A Monte Carlo approach is a statistical technique that allows us to assess not just the average performance of the algorithm but also its robustness and reliability under less common but possible situations. It is a way to ensure that the algorithm is not just effective in ideal conditions but also maintains its performance across a range of unpredictable real-world circumstances.

When testing the algorithm with human input, particularly allowing developers to act as human subjects, we identified specific issues that pose challenges to the algorithm. Our focus on the Monte Carlo setup is the variability that is likely to highlight these situations as opposed to varying parameters in which we already have high confidence.

In Table 2, we provide a detailed breakdown of the simulation parameters with an example setting. We categorize the parameters into 'Fixed Settings,' 'Pre-Defined Conditions,' and 'Dynamic

Variables'. 'Fixed Settings' is the static world that remains unchanged throughout all simulations. The 'Pre-Defined Conditions' detail types and range of obstacles that will populate the simulation world prior to each set of runs. The 'Dynamic Variables,' comprising the goal-driven desired linear and angular speeds, are live (created in runtime) and randomly generated user commands to simulate user inputs. While Table 2 represents a particular instance of our simulation settings and parameters, we emphasize that they can be modified to fit different scenarios. In Chapter 7, we provided an example with actual configurations.

Table 2. Detailed Scenario Configuration for Monte Carlo Simulations.

Fixed Settings for *All Runs*	**Pre-defined conditions for *Each Run Sets***			**Dynamic variables for *Each Run***			
Environment type	**Obstacle type**	**Number of obstacles**		**Desired linear speed (m/s)**		**Desired angular speed (rad/s)**	
		Low	**High**	**Min**	**Max**	**Min**	**Max**
Large Room	chair	1	2	0	1	-1.5	1.5
	sofa	1	1				
	table	1	2				

6.2.2 Modeling User Input

An essential component of our simulation approach is user input modeling. We generate synthetic user inputs by integrating a dual approach: **goal-driven commands** based on a Dijkstra [149] global planner from the ROS Noetic [11] Navigation stack, and **noise-added commands** to emulate human-like random inputs. We alternate between these two command types at specific time intervals, aligning with real-world scenarios where a user might have clear navigation goals intermittently interspersed with periods of less predictable control, such as slight deviations or

corrections in the path. The duration of these intervals is predetermined, with goal-driven commands typically lasting for longer periods (e.g., 15 seconds) followed by shorter periods (e.g., 5 seconds) of noisy inputs. This pattern aims to reflect human behavior, where individuals maintain focus on a given task for sustained periods, interspersed with brief lapses in attention or deviations from intended paths [31]-[35].

While our model integrates goal-driven commands and introduces noise-added commands to simulate human-like random inputs, it primarily focuses on the variability in execution rather than decision-making changes. Our current model does not simulate changes in human decisions, such as shifting objectives or altering navigational plans in response to sudden obstacles or other environmental changes. The model assumes that once a goal is set and the user is directing the robot towards it, the only deviations are those caused by random noise, not by deliberate decision changes.

The approach for introducing randomness into command velocities is detailed below:

- For linear velocity v: $v = v_{goal} + N(\mu_l, \sigma_l)$;
- For angular velocity ω: $\omega = \omega_{goal} + N(\mu_a, \sigma_a)$,

where v_{goal} and ω_{goal} are velocities directed towards current goal, and $N(\mu, \sigma)$ represents Gaussian noise added to these velocities, with μ and σ being mean and standard deviation, respectively. It is important to note that mean and standard deviation are constrained by 'Dynamic Variables' in Table 2.

We publish synthetic user inputs at 10 Hz, which is typical for real-life remote-control frequencies, by using the following ROS message type called *Twist*:

$$\boldsymbol{command} = Twist() \rightarrow \boldsymbol{command.linear.x} = v, \boldsymbol{command.angular.z} = \omega$$

where v and ω represent either goal-directed or Gaussian noise added velocities depending on the time interval they are generated. Randomness is determined once at the start of each noisy interval and applied consistently throughout its duration. This technique prevents Gaussian noise from cancelling itself out. Additionally, by maintaining a constant noise level for each interval, we avoid creating jittery or overly erratic movements that could detract from the realism of the simulation.

All inputs, categorized as goal-driven or noise-added, are systematically logged into a CSV file, enabling a thorough post-simulation analysis to assess the extent of noise integration.

Acknowledging the challenges in perfectly mimicking real user behavior, our current simulation approach represents an initial step towards capturing the fluctuating attention and error-proneness characteristic of human users. While not exhaustive, this model provides a structured design to emulate human-like input patterns, serving as a foundation for future enhancements.

6.3 Simulation Implementation Specifics

We implemented our evaluation method using the Robot Operating System (ROS) framework, the robotic simulator Gazebo, and a set of C++ programs, Python and shell scripts on an Ubuntu operating system. The proposed approach allows for black-box testing of a broad range of shared control algorithms developed for ROS-based mobile robots by integrating key performance metrics, such as engagement ratio and velocity deviations, directly within the algorithms.

6.3.1 Synthetic Environment Creation

In our simulation setup, we utilized Gazebo's *world* files to create a virtual test environment.

These worlds are custom-built to represent any kind of buildings such as an office, hospital, warehouse, and so on. To enhance the efficiency of our simulations, we opted for simplicity in our virtual models. Rather than using pre-existing world models that may contain unnecessarily detailed geometries, we chose to implement straightforward geometric shapes resembling buildings and instead add complexity based on the variety and density of obstacles within the spaces. Furthermore, we optimized the simulations by disabling non-critical environmental effects like shadows, wind, and atmospheric conditions, focusing computational power on crucial algorithm dynamics and interactions for clearer performance assessment.

6.3.2 Running Batch Simulations

To efficiently handle numerous simulations required by the Monte Carlo method, we developed a suite of scripts and roslaunch files dedicated to automating the execution of various scenarios (Figure 36). These automation scripts allow the simulations to be run unattended and can be configured to distribute simulations across a variety of computing resources, including separate computers, Docker containers, or cloud computing platforms for accelerated testing. Also, we designed another script that reads obstacle configurations from a CSV file and randomly places obstacles while ensuring that they do not overlap with each other and with existing objects in the simulation world. As part of our preliminary setup, we generate and store Gazebo “world” files populated with these randomly placed obstacles. These pre-generated world files are then referenced in the scenario configuration file for subsequent use in each simulation run. This step, while not mandatory, serves as a crucial optimization technique and significantly reduces computational overhead at the beginning of each run.

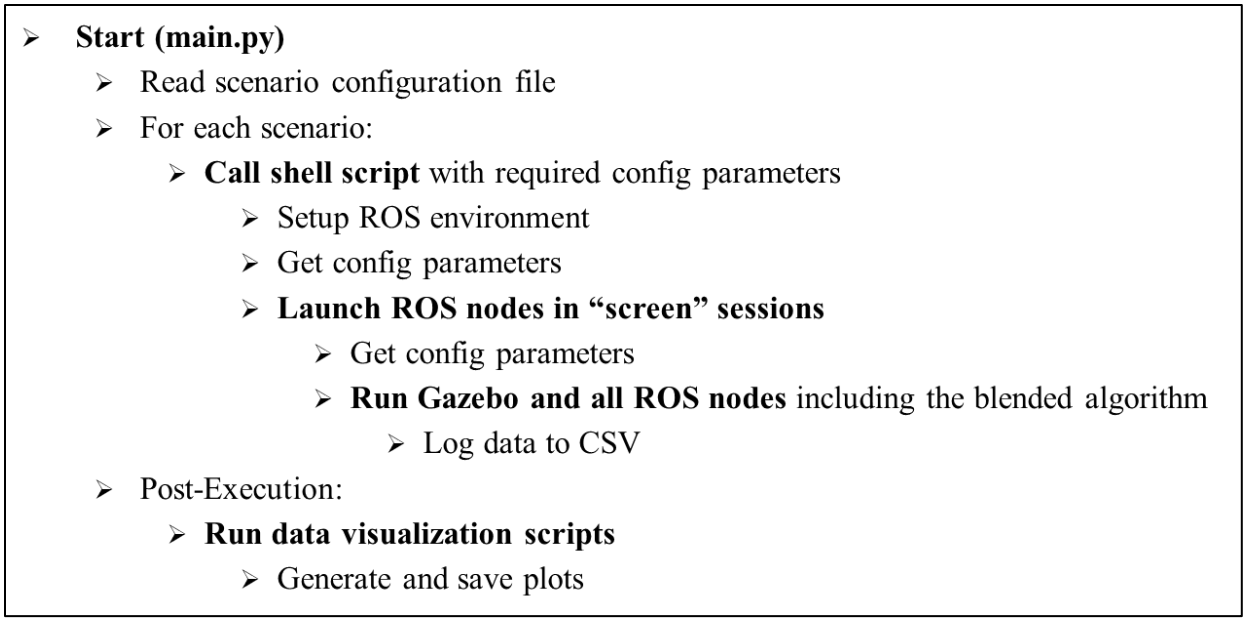

Figure 36. The workflow of the main script that runs batch simulations.

The coordination of simulation runs is conducted through a Python script (**main.py**). Once the virtual worlds are ready, the process in this main script begins with parsing the scenario configuration file. The configuration file is in YAML format and includes various parameters such as environmental settings, robot dynamics, and simulation-specific variables. Then, for each scenario, the main script calls a shell script, which is responsible for setting up the ROS environment and launching the required ROS nodes within individual terminal sessions —using the 'screen' terminal tool to manage these sessions. During each scenario, relevant data for performance metrics are logged into uniquely named CSV files by their corresponding nodes.

Upon completion of all scenarios, the main script invokes Python-based plotting scripts that automatically generate visualizations from the recorded CSV files, providing insights into each scenario's dynamics, such as the comparison between goal-driven and noisy command inputs. Finally, the plots are saved as image files.

CHAPTER 7

U-CENTB^2 EXPERIMENTAL RESULTS AND DISCUSSION

We evaluated the performance of the proposed U-CenTB^2 algorithm by running batch simulations based on Monte Carlo technique. In Chapter 6, we introduced a simulation approach for evaluating performance of blended control algorithms designed for mobile robots in realistic scenarios, where we create a simulated counterpart of a real-world test environment. The algorithm was not compared directly with existing shared control algorithms due to its unique integration of risk-based control and user-centric teleoperation, designed specifically for mobile robots. Given the specialized nature of this algorithm and the absence of directly comparable systems, the experimental section focuses on demonstrating its effectiveness within its intended context.

Typically, experimental setups with human participants involve navigating a robot from point A to point B, avoiding obstacles along the way. Given the complexities of remotely controlling a child-sized heavy robot and the potential for user error (due to distractions or lack of expertise), it is challenging for the user to flawlessly follow a designated path without occasionally colliding with obstacles. To realistically simulate these conditions, we integrated both global and local planners from the Robot Operating System (ROS) [11] navigation stack. The global planner emulates strategic pathfinding similar to how a person plans a route to a destination, and the local planner simulates immediate navigational adjustments required to avoid obstacles, reflecting a person's real-time decision-making process. The introduction of Gaussian noise aims to replicate the natural variability and errors in human control inputs.

Our experiments are based on the simulated counterpart of David’s robot (Figure 37) that

uses a range and a depth sensor. Simulation worlds are designed to resemble real settings using realistic objects, such as chairs, sofas, tables, bookshelves.

We implemented the algorithm using ROS framework with C++ on Linux platform. The simulation experiments are conducted in real-time on a Dell computer with Ubuntu OS, 12Th Gen Intel Core i7-12700Hx20, NVIDIA GeForce RTX3080 Ti GPU, and 64 GB RAM. We monitored CPU, GPU, and memory utilization throughout the testing phases. We observed effective resource usage, leveraging the available CPU and GPU capacities without full saturation, suggesting an efficient execution that maintains system responsiveness and stability.

We acknowledge that our model for human input, featured by its intentional errors and goal-driven commands, has not been validated against actual human behavior. Our goal is not to perfectly mirror human input but to present the algorithm with a range of challenging scenarios to test its effectiveness.

7.1 Experimental Setup

For our simulation tests, we have created a two-wheeled robot that carries the same types of sensors as the real one (Figure 37). Similar to an actual differential drive robot, the kinematics of the simulated robot are governed by the same equations to closely reflect real-life conditions. The physical and sensory attributes of our robot are detailed in a ROS URDF (Unified Robot Description Format) file. The kinematics, along with a suite of sensors, are defined in this file, with Gazebo plugins attached to simulate their real-world counterparts.

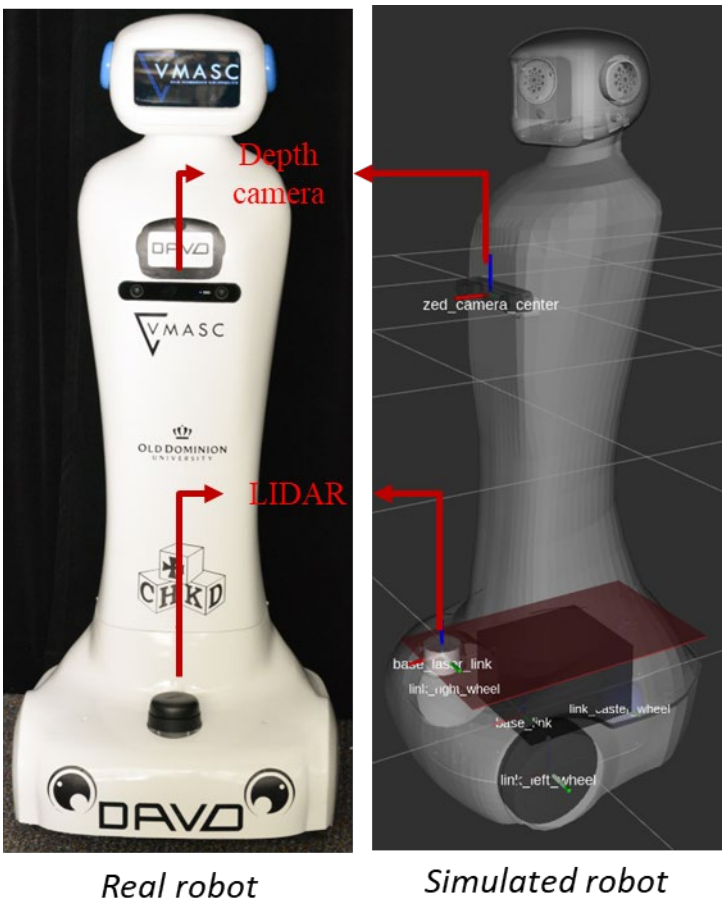

Figure 37. A simulated correspondence of the real robot.

Simulation methodology in Chapter 6 allows for custom configuration of simulation experiments based on a base world. We created a 50x50 m large space with walls for the base world. Then, we created three scenario files with the configuration shown in Table 3. The simulator randomly places these obstacles with random numbers in the “min” and “max” range. Figure 38 depicts screenshots from these environments along with the robot.

Table 3. Scenario configuration: Number of obstacles.

model	min	max	model	min	max	model	min	max
Grey tote	5	10	Chair	8	12	Side table	10	15
Cabinet	4	6	Coffee table	5	10	Side table2	5	10
Bookshelf	8	16	File cabinet	7	9	Sofa set	20	30
Cube storage	4	5	Office chair	10	15			

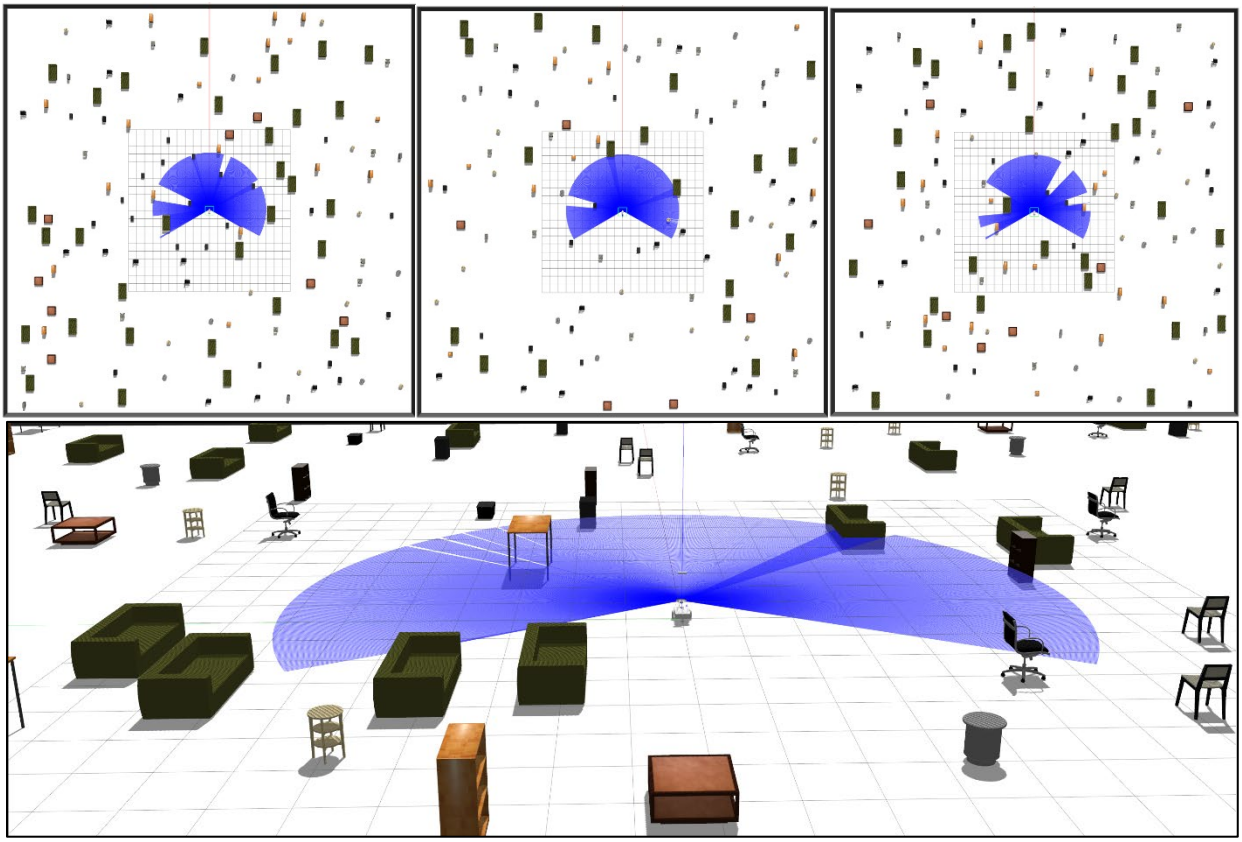

Figure 38: Screenshots from three simulation scenarios with randomly located obstacles in a 50x50 environment surrounded by black walls. Top images show top-down view. The bottom image demonstrates how obstacles look like from a different perspective.

Four navigational targets are set near the corner points in these worlds for the simulated user input. At each scenario iteration, the robot randomly selects one of these targets and starts navigating there. If the robot reaches its target, it randomly selects another corner as a target and starts navigating there. The commands from this navigation represent the goal-driven aspect of user input. We let the robot navigate for 15 seconds with these commands, then introduce Gaussian noise, i.e., 0.2 and 1.0 standard deviations for linear and angular velocities, to these commands for 5 seconds. In total, the experiments are conducted by running each scenario 600 seconds for 10 times.

7.2 Performance Metrics

In our assessment of a shared control algorithm, we measure its success using three principal performance metrics. These metrics are selected for their capacity to reflect the algorithm's efficiency and effectiveness in a simulated environment.

Number of Collisions: It tracks the frequency of contact between the robot and any obstacles, offering a direct measure of the algorithm’s ability to prevent collisions. To accurately measure this, we implemented a collision detection method in C++ using Gazebo events. The code listens for collision events from Gazebo involving the robot and logs each incident while distinguishing between actual collisions and mere contacts with benign elements like 'ground_plane'.

We discern continuous collision signals from a single event by introducing a count threshold, where a collision is only recorded if a certain number of time steps have passed without a collision, effectively filtering out repeated messages from the same incident. This method ensures that our collision count is not inflated by lingering effects of a single collision. Each recorded collision,

along with a timestamp, scenario, and iteration number, is logged into a CSV file.

Engagement Ratio: This metric reflects the portion of time during which the algorithm intervenes. For example, an Engagement Ratio of 65% indicates that for 65% of the time, the algorithm controlled the robot versus 35% of the time during which user inputs were passed straight through. In our simulations, 'user inputs' refer to synthetic inputs designed to emulate real user interactions, although real user inputs could be utilized in physical experiments. This metric is especially important in assessing the performance of the algorithm. A very high number is not desired as it indicates that the algorithm is effectively driving the robot most of the time, potentially causing the user to feel not in control. Conversely, a very low engagement ratio indicates that the algorithm did not intervene, which in turn indicates that this particular test has no value, as it presented no challenge to be addressed. For example, large spaces with sparse obstacles could lead to a low engagement ratio, but this does not necessarily indicate the algorithm's proactive performance. Therefore, it reveals the algorithm's involvement and ensures its performance is not artificially inflated in sparse environments.

This metric needs to be incorporated into the shared control algorithm. Let's define $T_{algorithm}$ as total time the algorithm is in control (engaged), and T_{total} as total runtime of the algorithm. The $Engagement\ Ratio$ can be expressed as:

$$Engagement\ Ratio = \left(\frac{T_{algorithm}}{T_{total}}\right) x\ 100,$$

where $T_{algorithm}$ is incremented by the duration of each loop cycle time, if the algorithm intervenes in user inputs during that cycle. When the algorithm finishes its execution, engagement is timestamped and logged, including scenario and iteration identifiers, into a CSV file.

Velocity Deviation: This metric evaluates a control algorithm's precision by comparing the desired user-set velocity with the actual velocity achieved by the robot. It is a crucial metric for determining the algorithm's ability to execute user commands closely.

We compute this metric by averaging the normalized deviations between actual and desired velocities over fixed intervals, e.g., every 5 or 10 seconds. Within each interval, velocity deviations are calculated at regular time steps, summed up, and then divided by the interval duration to normalize. Finally, an average of these normalized values across all intervals provides $Average\ velocity\ deviation$, a single metric that indicates how closely a robot follows the desired velocity.

Similar to engagement ratio, we measure deviations in velocity within the algorithm using the following formula:

$$Average\ velocity\ deviation = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{\Delta t}\sum_{j=1}^{M}\left(V_{actual}(t_{i,j}) - V_{desired}(t_{i,j})\right).\delta t\right),$$

where:

N is total number of fixed-duration intervals.

M is number of discrete time steps within each interval Δt.

Δt is duration of each interval (each interval is a fixed duration, e.g., 5 or 10 seconds).

$V_{actual}(t_{i,j})$ actual linear velocity measured at j-th time step of i-th interval.

$V_{desired}(t_{i,j})$ desired linear velocity measured at j.

$t_{i,j}$ specific time at which each measurement is taken within i-th interval.

In addition to average deviations, we record minimum and maximum velocity deviations for each scenario runtime for deeper insights.

7.3 Results and Discussion

Plots in Figure 39 display a snapshot of synthetic human commands from Scenario-2/Iteration-8 run. While the top plot represents linear speeds, the bottom one depicts angular velocities. Blue lines show goal-oriented inputs, demonstrating a more consistent pattern as the robot moves towards predefined goals, while red lines denote noisy velocities. We intentionally introduced too much noise to aggressively test the performance of the algorithm. The fluctuations in noisy linear velocity clearly indicate periods of increased and decreased speed, which reflect a user's natural variations while teleoperating a robot. Likewise, angular velocity exhibits random changes in direction, indicating the human tendency for overcorrection or uncertainty in turn commands.

The results of the collisions are plotted in Figure 40. Each bar represents the number of collisions at each iteration. Running simulations without our algorithm resulted in 461 total collisions, while ours was counted as only 11. In fact, we expected no collisions at all. However, we modeled our simulated robot as a counterpart of the real robot we have in our lab that does not have 360 degree sensing capability. We believe that full rotations right next to an obstacle might have caused these collisions. Yet, our algorithm successfully prevented collisions by 98% compared to without having the algorithm.

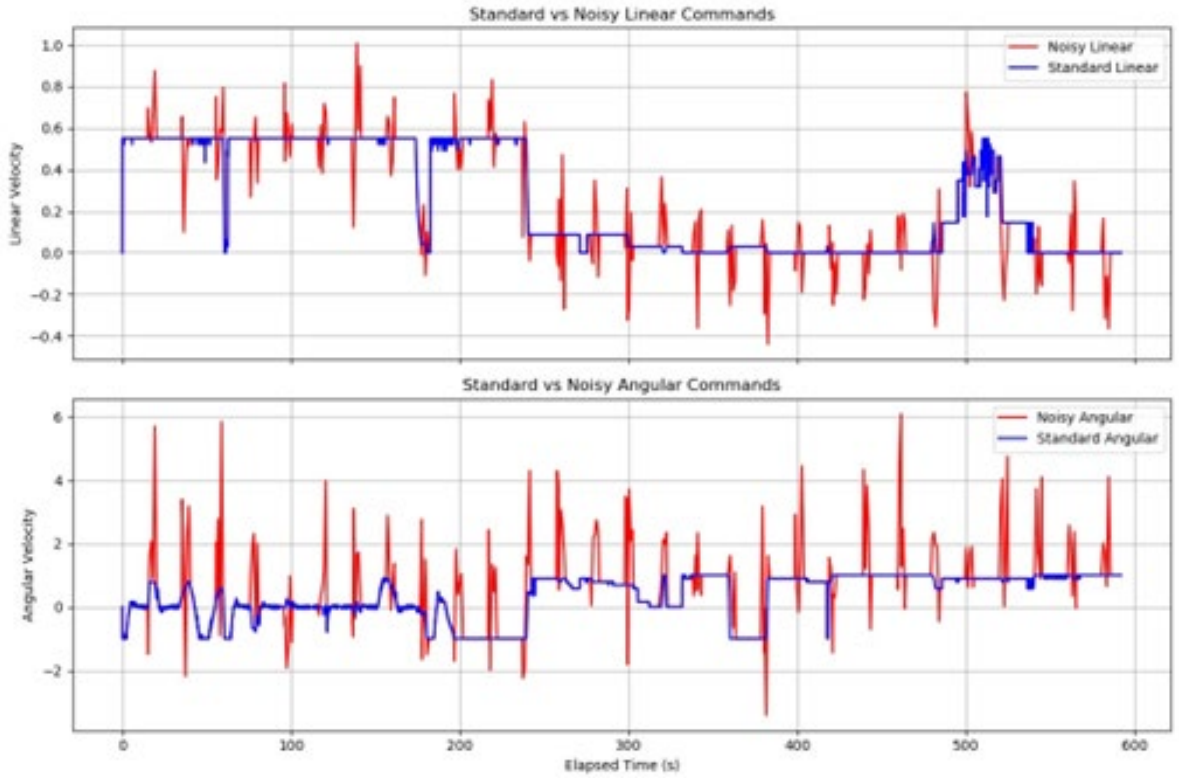

Figure 39. Simulated user inputs with goal-driven (blue) and the noise added (red) velocities.

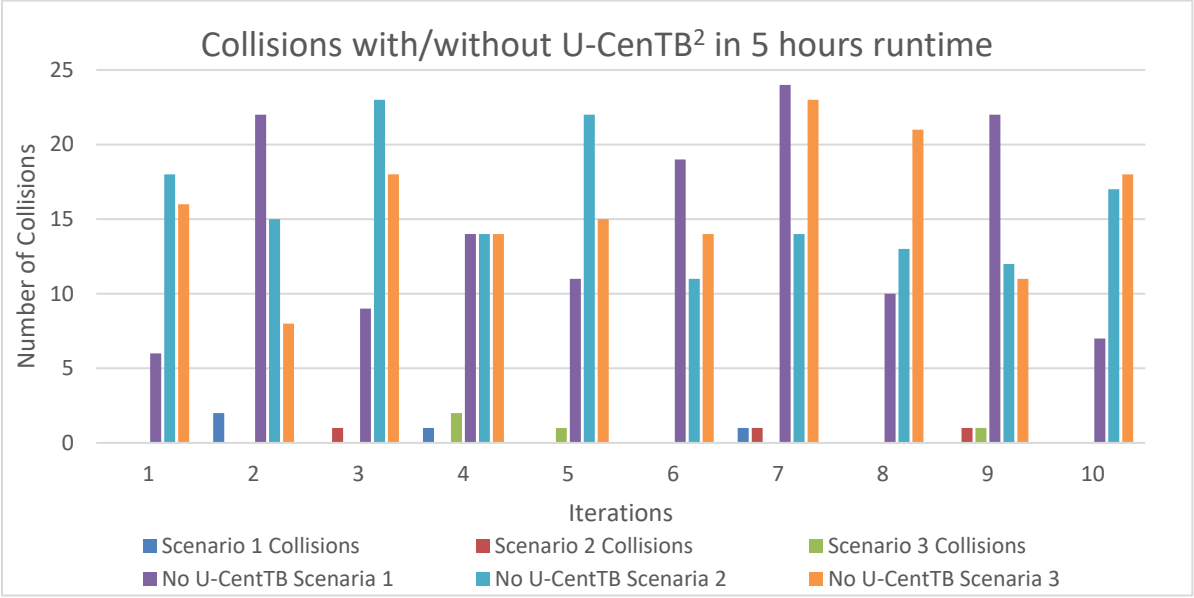

Figure 40: Number of collisions.

The velocity deviations between the intended and actual velocities of the robot are given in Figure 41. For linear velocity, we observed an average deviation ranging from -0.5 to -0.01 m/s with an overall average of -0.2 m/s. This indicates that the robot was slowed down most of the time to prevent collisions. The acceleration and deceleration of the robot may have contributed to the negative numbers since we considered only kinematics of the robot model for the metrics. Similarly, for angular velocity, the average deviation varied from -1.5 and 0.9 rad/s with an -0.2 rad/s overall average. According to the results, the robot steers away from the commanded directions when there is a collision risk. Even though the data show high peaks, especially in rotational deviations, it is important to note that the introduced noise to the user inputs are more than needed to simulate realistic human behavior. Therefore, we can still conclude that the robot mostly adheres to synthetic user inputs.

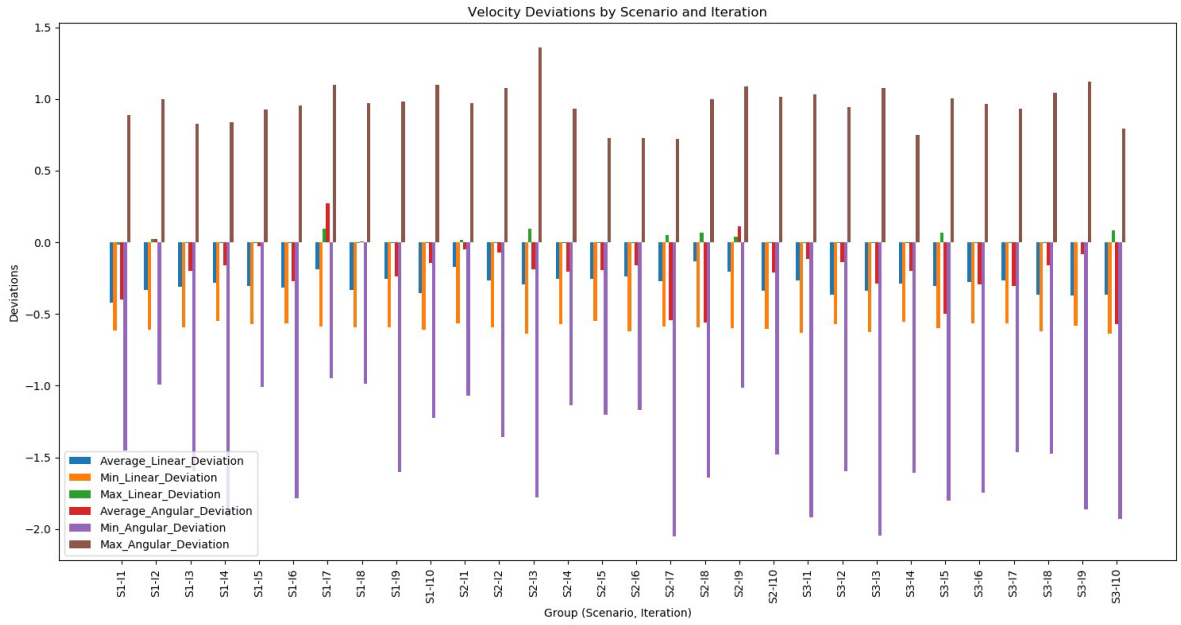

Figure 41. Velocity deviations.

For Engagement Ratios (Figure 42), we observed an average ratio ranging from 26% to 80%. An engagement ratio of 26% means that the algorithm controlled the robot for 26% of the time, while for the remaining 74% of the time, the input commands directly influenced the robot's actions. Conversely, an 80% ratio indicates that the algorithm predominantly controlled the robot. Across all observations, the average engagement ratio was 46%, indicating a balanced distribution of control between the algorithm and the human operator.

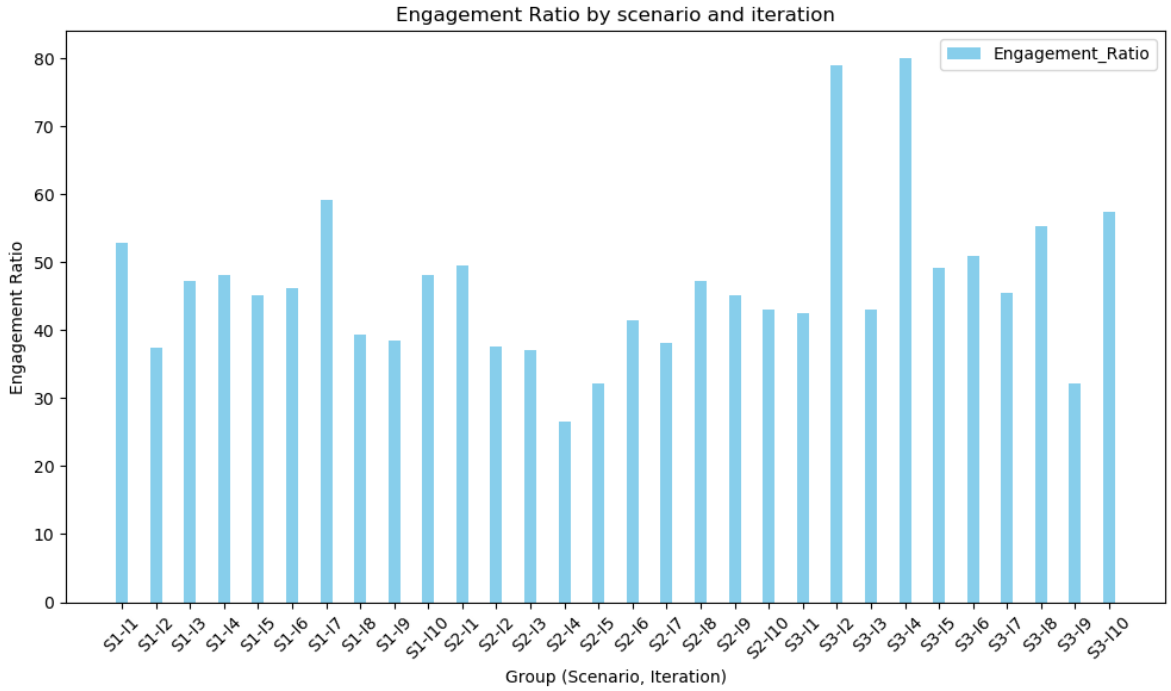

Figure 42. Engagement ratios.

CHAPTER 8

SUMMARY AND FUTURE WORK

8.1 Summary of the Dissertation

In this dissertation, we focused on developing a shared control algorithm designed to facilitate safe teleoperation of mobile robots, particularly within indoor environments such as hospitals. The research is motivated by the need for improved social connectivity for young patients, inspired by the story of a young patient, David Carey. We integrated user inputs with semi-autonomous control to ensure safe and efficient operation without compromising user control. At the core of the presented work is the U-CenTB^2 algorithm, a risk-based blended control approach that implements a modified Tangent Bug with a risk assessment strategy to avoid collisions dynamically.

The proposed multilevel shared control system operates at various levels. Level-0 offers direct joystick control for immediate robot control, while Level-1 refines commands for corridor-specific navigation using real-time corridor detection. While the focus of this dissertation has primarily been on developing and evaluating collision avoidance strategies in open environments, the corridor detection algorithm is included as a foundational element for future integration. This module not only adds to the versatility of the navigation system but also presents a clear path for subsequent enhancements that could leverage corridor-based guidance for more precise and reliable navigation.

Higher levels, which were planned for future integration, include advanced functionalities

such as room-to-room navigation and human-following capabilities. The system's modular design ensures adaptability and scalability, allowing for future enhancements and integration of these higher-level capabilities.

The dissertation also presented a comprehensive simulation-based approach for evaluating the performance of shared control algorithms using Monte Carlo method and running batch simulations. This simulation design models user inputs with Gaussian-distributed variabilities and evaluates the effectiveness of shared control algorithms under various conditions through performance metrics such as number of collisions, engagement ratios, and velocity deviations. We evaluated the proposed U-CenTB^2 algorithm through this methodology. The results demonstrated that the proposed system effectively balances user control with safety, providing a significant contribution to the teleoperation of assistive mobile robots in hospital environments.

The dissertation's main contributions are twofold: it advances the field of mobile robot teleoperation by developing a modular shared control system that includes implementation of a user-centric shared control algorithm, efficient corridor detection and obstacle avoidance techniques and proposing a comprehensive simulation-based evaluation approach for shared control algorithms. The work demonstrates the potential of integrating mobile robots into daily lives of young patients to enhance their hospital experience.

8.2 Future Work

Future research can extend the presented system by implementing and integrating higher-level functionalities such as room navigation and human following, which were beyond the scope of the current work. A major focus will be on integrating and thoroughly evaluating the corridor detection module, which, while demonstrated, was not fully assessed within the current study. This

will complement efforts to validate the simulation techniques used, particularly the modeling of user inputs, by comparing them with actual human behavior to refine their accuracy.

Additionally, incorporating advanced sensor fusion techniques and enhancing the obstacle detection module with a temporal aspect could further improve the system's adaptability and robustness in dynamic hospital environments. Evaluating the system's usability, social, and emotional impacts on users through qualitative studies is also crucial to understand its effectiveness in real-world scenarios. Finally, transitioning from a simulation-based evaluation to real-world testing with human subjects will be essential to validate the system's performance and ensure its reliability and safety in practical applications.

REFERENCES

[1] "DAVID'S PROJECT: Designing the first telehealth robot just for children." Accessed: Mar. 19, 2024. [Online]. Available: https://www.13newsnow.com/article/tech/davids-project-first-telehealth-robot-for-kids/291-c5d9126e-e2f8-4094-95ba-bb8740dca1cc.

[2] A. Meghdari, M. Alemi, M. Khamooshi, A. Amoozandeh, A. Shariati, and B. Mozafari, "Conceptual design of a social robot for pediatric hospitals," presented at the 2016 4th international conference on robotics and mechatronics (ICROM), IEEE, 2016, pp. 566–571.

[3] D. E. Logan *et al.*, "Social robots for hospitalized children," *Pediatrics*, vol. 144, no. 1, 2019.

[4] S. Cooper, A. Di Fava, C. Vivas, L. Marchionni, and F. Ferro, "ARI: The social assistive robot and companion," presented at the 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), IEEE, 2020, pp. 745–751.

[5] Y. Gao and C.-M. Huang, "Evaluation of socially-aware robot navigation," *Front. Robot. AI*, p. 420, 2021.

[6] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *Int. J. Robot. Res.*, vol. 35, no. 11, pp. 1289–1307, 2016.

[7] J. Müller, C. Stachniss, K. O. Arras, and W. Burgard, "Socially inspired motion planning for mobile robots in populated environments," presented at the Proc. of International Conference on Cognitive Systems, 2008.

[8] A. Abou Allaban, V. Dimitrov, and T. Padır, "A blended human-robot shared control framework to handle drift and latency," presented at the 2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), IEEE, 2019, pp. 81–87.

[9] J. Leaman and H. M. La, "A comprehensive review of smart wheelchairs: past, present, and future," *IEEE Trans. Hum.-Mach. Syst.*, vol. 47, no. 4, pp. 486–499, 2017.

[10] T. K. Stephens, N. J. Kong, R. L. Dockter, J. J. O'Neill, R. M. Sweet, and T. M. Kowalewski, "Blended shared control utilizing online identification: regulating grasping forces of a surrogate surgical grasper," *Int. J. Comput. Assist. Radiol. Surg.*, vol. 13, pp. 769–776, 2018.

[11] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," presented at the ICRA workshop on open source software, Kobe, Japan, 2009, p. 5.

[12] J. Bongard, "Probabilistic Robotics. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. (2005, MIT Press.) 647 pages," *Artif. Life*, vol. 14, no. 2, pp. 227–229, Apr. 2008, doi: 10.1162/artl.2008.14.2.227.

[13] R. Luo, R. Hayne, and D. Berenson, "Unsupervised early prediction of human reaching for human–robot collaboration in shared workspaces," *Auton. Robots*, vol. 42, pp. 631–648, 2018.

[14] Q. Li, Z. Zhang, Y. You, Y. Mu, and C. Feng, "Data driven models for human motion prediction in human-robot collaboration," *IEEE Access*, vol. 8, pp. 227690–227702, 2020.

[15] K. Haninger, C. Hegeler, and L. Peternel, "Model predictive control with gaussian processes for flexible multi-modal physical human robot interaction," presented at the 2022 International Conference on Robotics and Automation (ICRA), IEEE, 2022, pp. 6948–6955.

[16] K. Haninger, C. Hegeler, and L. Peternel, "Model predictive impedance control with Gaussian processes for human and environment interaction," *Robot. Auton. Syst.*, vol. 165, p.

104431, 2023.
[17] T. Fong, I. Nourbakhsh, and K. Dautenhahn, “A survey of socially interactive robots,” *Robot. Auton. Syst.*, vol. 42, no. 3–4, pp. 143–166, 2003.
[18] C. Breazeal, “Toward sociable robots,” *Robot. Auton. Syst.*, vol. 42, no. 3–4, pp. 167–175, 2003.
[19] H. Mahdi, S. A. Akgun, S. Saleh, and K. Dautenhahn, “A survey on the design and evolution of social robots—Past, present and future,” *Robot. Auton. Syst.*, vol. 156, p. 104193, 2022.
[20] R. Kirby, J. Forlizzi, and R. Simmons, “Affective social robots,” *Robot. Auton. Syst.*, vol. 58, no. 3, pp. 322–332, 2010.
[21] S. Naneva, M. Sarda Gou, T. L. Webb, and T. J. Prescott, “A systematic review of attitudes, anxiety, acceptance, and trust towards social robots,” *Int. J. Soc. Robot.*, vol. 12, no. 6, pp. 1179–1201, 2020.
[22] A. A. Scoglio, E. D. Reilly, J. A. Gorman, and C. E. Drebing, “Use of social robots in mental health and well-being research: systematic review,” *J. Med. Internet Res.*, vol. 21, no. 7, p. e13322, 2019.
[23] L. Almeida, P. Menezes, and J. Dias, “Telepresence social robotics towards co-presence: A review,” *Appl. Sci.*, vol. 12, no. 11, p. 5557, 2022.
[24] M. Alemi, A. Ghanbarzadeh, A. Meghdari, and L. J. Moghadam, “Clinical application of a humanoid robot in pediatric cancer interventions,” *Int. J. Soc. Robot.*, vol. 8, pp. 743–759, 2016.
[25] J. Dawe, C. Sutherland, A. Barco, and E. Broadbent, “Can social robots help children in healthcare contexts? A scoping review,” *BMJ Paediatr. Open*, vol. 3, no. 1, 2019.
[26] E. Coronado, X. Indurkhya, and G. Venture, “Robots meet children, development of semi-autonomous control systems for children-robot interaction in the wild,” presented at the 2019 IEEE 4th international conference on advanced robotics and mechatronics (ICARM), IEEE, 2019, pp. 360–365.
[27] H. Kozima, C. Nakagawa, and Y. Yasuda, “Interactive robots for communication-care: A case-study in autism therapy,” presented at the ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005., IEEE, 2005, pp. 341–346.
[28] W. D. Stiehl, J. K. Lee, C. Breazeal, M. Nalin, A. Morandi, and A. Sanna, “The huggable: a platform for research in robotic companions for pediatric care,” presented at the Proceedings of the 8th International Conference on interaction Design and Children, 2009, pp. 317–320.
[29] L. Cañamero and M. Lewis, “Making new ‘New AI’ friends: designing a social robot for diabetic children from an embodied AI perspective,” *Int. J. Soc. Robot.*, vol. 8, no. 4, pp. 523–537, 2016.
[30] J. M. Beer and L. Takayama, “Mobile remote presence systems for older adults: acceptance, benefits, and concerns,” presented at the Proceedings of the 6th international conference on Human-robot interaction, 2011, pp. 19–26.
[31] T.-C. Tsai, Y.-L. Hsu, A.-I. Ma, T. King, and C.-H. Wu, “Developing a telepresence robot for interpersonal communication with the elderly in a home environment,” *Telemed. E-Health*, vol. 13, no. 4, pp. 407–424, 2007.
[32] A. D. Dragan and S. S. Srinivasa, “A policy-blending formalism for shared control,” *Int. J. Robot. Res.*, vol. 32, no. 7, pp. 790–805, 2013.
[33] C. S. González-González, V. Violant-Holz, and R. M. Gil-Iranzo, “Social robots in

hospitals: a systematic review," *Appl. Sci.*, vol. 11, no. 13, p. 5976, 2021.
[34] A. Alabdulkareem, N. Alhakbani, and A. Al-Nafjan, "A systematic review of research on robot-assisted therapy for children with autism," *Sensors*, vol. 22, no. 3, p. 944, 2022.
[35] E. Yang and M. C. Dorneich, "The emotional, cognitive, physiological, and performance effects of variable time delay in robotic teleoperation," *Int. J. Soc. Robot.*, vol. 9, pp. 491–508, 2017.
[36] M. Moniruzzaman, A. Rassau, D. Chai, and S. M. S. Islam, "Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey," *Robot. Auton. Syst.*, vol. 150, p. 103973, 2022.
[37] T. B. Sheridan, "A review of recent research in social robotics," *Curr. Opin. Psychol.*, vol. 36, pp. 7–12, 2020.
[38] E. Barakova, K. Väänänen, K. Kaipainen, and P. Markopoulos, "Benefits, Challenges and Research Recommendations for Social Robots in Education and Learning: A Meta-Review," presented at the 2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), IEEE, 2023, pp. 2555–2561.
[39] G. Li, Q. Li, C. Yang, Y. Su, Z. Yuan, and X. Wu, "The Classification and New Trends of Shared Control Strategies in Telerobotic Systems: A Survey," *IEEE Trans. Haptics*, vol. 16, no. 2, pp. 118–133, Jun. 2023, doi: 10.1109/TOH.2023.3253856.
[40] M. Marcano, S. Díaz, J. Pérez, and E. Irigoyen, "A review of shared control for automated vehicles: Theory and applications," *IEEE Trans. Hum.-Mach. Syst.*, vol. 50, no. 6, pp. 475–491, 2020.
[41] M. Desai and H. A. Yanco, "Blending human and robot inputs for sliding scale autonomy," in *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, Aug. 2005, pp. 537–542. doi: 10.1109/ROMAN.2005.1513835.
[42] 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.
[43] L. Devigne, V. K. Narayanan, F. Pasteau, and M. Babel, "Low complex sensor-based shared control for power wheelchair navigation," presented at the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2016, pp. 5434–5439.
[44] A. Hüntemann, E. Demeester, E. Vander Poorten, H. Van Brussel, and J. De Schutter, "Probabilistic approach to recognize local navigation plans by fusing past driving information with a personalized user model," presented at the 2013 IEEE International Conference on Robotics and Automation, IEEE, 2013, pp. 4376–4383.
[45] E. Demeester, A. Huntemann, J. del R. Millan, and H. Van Brussel, "Bayesian plan recognition for brain-computer interfaces," presented at the 2009 IEEE International Conference on Robotics and Automation, IEEE, 2009, pp. 653–658.
[46] R. C. Simpson and S. P. Levine, "Automatic adaptation in the NavChair assistive wheelchair navigation system," *IEEE Trans. Rehabil. Eng.*, vol. 7, no. 4, pp. 452–463, 1999.
[47] A. Saglam and Y. Papelis, "A Simulation-Based Approach for Evaluating Shared Control Algorithms for Mobile Robots," in *2024 Annual Modeling and Simulation Conference (ANNSIM)*, IEEE, 2024, p. TBD.
[48] C. Urdiales *et al.*, "A new multi-criteria optimization strategy for shared control in wheelchair assisted navigation," *Auton. Robots*, vol. 30, no. 2, pp. 179–197, 2011.

[49] R. A. Cooper, “Intelligent control of power wheelchairs,” *IEEE Eng. Med. Biol. Mag.*, vol. 14, no. 4, pp. 423–431, 1995.
[50] R. Tang, “A Semi-autonomous Wheelchair Navigation System,” 2012.
[51] T. Carlson and Y. Demiris, “Human-wheelchair collaboration through prediction of intention and adaptive assistance,” presented at the 2008 IEEE International Conference on Robotics and Automation, IEEE, 2008, pp. 3926–3931.
[52] T. Carlson and Y. Demiris, “Collaborative control in human wheelchair interaction reduces the need for dexterity in precise manoeuvres,” presented at the Proceedings of" Robotic Helpers: User Interaction, Interfaces and Companions in Assistive and Therapy Robotics", a Workshop at ACM/IEEE HRI 2008, University of Hertfordshire, 2008, pp. 59–66.
[53] T. Rofer, C. Mandel, and T. Laue, “Controlling an automated wheelchair via joystick/head-joystick supported by smart driving assistance,” presented at the 2009 IEEE International Conference on Rehabilitation Robotics, IEEE, 2009, pp. 743–748.
[54] G. Bourhis and M. Sahnoun, “Assisted control mode for a smart wheelchair,” presented at the 2007 IEEE 10th International Conference on Rehabilitation Robotics, IEEE, 2007, pp. 158–163.
[55] J. Abascal, B. Bonail, Á. Marco, R. Casas, and J. L. Sevillano, “AmbienNet: an intelligent environment to support people with disabilities and elderly people,” presented at the Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility, 2008, pp. 293–294.
[56] A. C. Lopes, U. Nunes, L. Vaz, and L. Vaz, “Assisted navigation based on shared-control, using discrete and sparse human-machine interfaces,” presented at the 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology, IEEE, 2010, pp. 471–474.
[57] Y. Wang and G. S. Chirikjian, “A new potential field method for robot path planning,” presented at the Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), IEEE, 2000, pp. 977–982.
[58] J.-W. Park, H.-J. Kwak, Y.-C. Kang, and D. W. Kim, “Advanced fuzzy potential field method for mobile robot obstacle avoidance,” *Comput. Intell. Neurosci.*, vol. 2016, 2016.
[59] E. B. Vander Poorten, E. Demeester, E. Reekmans, J. Philips, A. Hüntemann, and J. De Schutter, “Powered wheelchair navigation assistance through kinematically correct environmental haptic feedback,” presented at the 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 3706–3712.
[60] M. Sato, T. Tomizawa, S. Kudoh, and T. Suehiro, “Development of a collision-avoidance assist system for an electric cart,” presented at the 2011 IEEE International Conference on Robotics and Biomimetics, IEEE, 2011, pp. 337–342.
[61] S. P. Parikh, V. Grassi, V. Kumar, and J. Okamoto, “Integrating human inputs with autonomous behaviors on an intelligent wheelchair platform,” *IEEE Intell. Syst.*, vol. 22, no. 2, pp. 33–41, 2007.
[62] R. C. Simpson, “Smart wheelchairs: A literature review,” *J. Rehabil. Res. Dev.*, vol. 42, no. 4, p. 423, 2005.
[63] R. Tang, X. Q. Chen, M. Hayes, and I. Palmer, “Development of a navigation system for semi-autonomous operation of wheelchairs,” presented at the Proceedings of 2012 IEEE/ASME 8th IEEE/ASME International Conference on Mechatronic and Embedded

Systems and Applications, IEEE, 2012, pp. 257–262.
[64] V. Tyagi, N. K. Gupta, and P. K. Tyagi, “Smart wheelchair using fuzzy inference system,” presented at the 2013 IEEE Global Humanitarian Technology Conference: South Asia Satellite (GHTC-SAS), IEEE, 2013, pp. 175–180.
[65] Y. Touati and A. Ali-Cherif, “Smart powered wheelchair platform design and control for people with severe disabilities,” *Softw. Eng.*, vol. 2, no. 3, pp. 49–56, 2012.
[66] K. Miyazaki, M. Hashimoto, M. Shimada, and K. Takahashi, “Guide following control using laser range sensor for a smart wheelchair,” presented at the 2009 ICCAS-SICE, IEEE, 2009, pp. 4613–4616.
[67] Y. Kobayashi, Y. Kinpara, E. Takano, Y. Kuno, K. Yamazaki, and A. Yamazaki, “Robotic wheelchair moving with caregiver collaboratively,” presented at the International Conference on Intelligent Computing, Springer, 2011, pp. 523–532.
[68] R. Murakami, L. Y. Morales Saiki, S. Satake, T. Kanda, and H. Ishiguro, “Destination unknown: walking side-by-side without knowing the goal,” presented at the Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction, 2014, pp. 471–478.
[69] R. Suzuki, T. Yamada, M. Arai, Y. Sato, Y. Kobayashi, and Y. Kuno, “Multiple robotic wheelchair system considering group communication,” presented at the International symposium on visual computing, Springer, 2014, pp. 805–814.
[70] T. Sugano, Y. Dan, H. Okajima, N. Matsunaga, and Z. Hu, “Indoor platoon driving of electric wheelchair with model error compensator along wheel track of preceding vehicle,” presented at the Proceedings of the 5th International Symposium on Advanced Control of Industrial Processes (2014b), 2014, pp. 219–224.
[71] H. P. Moravec, “Sensor fusion in certainty grids for mobile robots,” in *Sensor devices and systems for robotics*, Springer, 1989, pp. 253–276.
[72] L. Yang, X. Wu, D. Zhao, H. Li, and J. Zhai, “An improved Prewitt algorithm for edge detection based on noised image,” presented at the 2011 4th International congress on image and signal processing, IEEE, 2011, pp. 1197–1200.
[73] Z. Wei, W. Chen, and J. Wang, “3D semantic map-based shared control for smart wheelchair,” presented at the International Conference on Intelligent Robotics and Applications, Springer, 2012, pp. 41–51.
[74] S. Cockrell, G. Lee, and W. Newman, “Determining navigability of terrain using point cloud data,” presented at the 2013 IEEE 13th International Conference on Rehabilitation Robotics (ICORR), IEEE, 2013, pp. 1–6.
[75] Y. Uratsuji, K. Takemura, J. Takamatsu, and T. Ogasawara, “Mobility assistance system for an electric wheelchair using annotated maps,” *Adv. Robot.*, vol. 29, no. 7, pp. 481–491, 2015.
[76] P. Viswanathan, J. Little, A. Mackworth, and A. Mihailidis, “Adaptive navigation assistance for visually-impaired wheelchair users,” presented at the Proceedings of the IROS 2011 Workshop on New and Emerging Technologies in Assistive Robotics, 2011.
[77] Y. Ren, W. Zou, H. Fan, A. Ye, K. Yuan, and Y. Ma, “A docking control method in narrow space for intelligent wheelchair,” presented at the 2012 IEEE International Conference on Mechatronics and Automation, IEEE, 2012, pp. 1615–1620.
[78] M. R. M. Tomari, Y. Kobayashi, and Y. Kuno, “Enhancing wheelchair’s control operation of a severe impairment user,” presented at the The 8th International Conference on Robotic, Vision, Signal Processing & Power Applications, Springer, 2014, pp. 65–72.

[79] S. Jain and B. Argall, “Automated perception of safe docking locations with alignment information for assistive wheelchairs,” presented at the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2014, pp. 4997–5002.
[80] D. A. Abbink *et al.*, “A Topology of Shared Control Systems—Finding Common Ground in Diversity,” *IEEE Trans. Hum.-Mach. Syst.*, vol. 48, no. 5, pp. 509–525, Oct. 2018, doi: 10.1109/THMS.2018.2791570.
[81] R. C. Goertz, “Fundamentals of general-purpose remote manipulators,” *Nucleonics*, pp. 36–42, 1952.
[82] P. F. Hokayem and M. W. Spong, “Bilateral teleoperation: An historical survey,” *Automatica*, vol. 42, no. 12, pp. 2035–2057, 2006.
[83] D. Aarno, S. Ekvall, and D. Kragic, “Adaptive Virtual Fixtures for Machine-Assisted Teleoperation Tasks,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Apr. 2005, pp. 1139–1144. doi: 10.1109/ROBOT.2005.1570269.
[84] Q. Li, W. Chen, and J. Wang, “Dynamic shared control for human-wheelchair cooperation,” presented at the 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 4278–4283.
[85] A. Hong, O. Igharoro, Y. Liu, F. Niroui, G. Nejat, and B. Benhabib, “Investigating human-robot teams for learning-based semi-autonomous control in urban search and rescue environments,” *J. Intell. Robot. Syst.*, vol. 94, pp. 669–686, 2019.
[86] A. Gottardi, S. Tortora, E. Tosello, and E. Menegatti, “Shared control in robot teleoperation with improved potential fields,” *IEEE Trans. Hum.-Mach. Syst.*, vol. 52, no. 3, pp. 410–422, 2022.
[87] F. J. Ruiz-Ruiz, C. Urdiales, M. Fernández-Carmona, and J. M. Gómez-de-Gabriel, “A Reactive performance-based Shared Control Framework for Assistive Robotic Manipulators,” *ArXiv Prepr. ArXiv231103232*, 2023.
[88] A. Afzal, D. S. Katz, C. Le Goues, and C. S. Timperley, “Simulation for Robotics Test Automation: Developer Perspectives,” in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2021, pp. 263–274. doi: 10.1109/ICST49551.2021.00036.
[89] C. K. Liu and D. Negrut, “The role of physics-based simulators in robotics,” *Annu. Rev. Control Robot. Auton. Syst.*, vol. 4, pp. 35–58, 2021.
[90] H. Choi *et al.*, “On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward,” *Proc. Natl. Acad. Sci.*, vol. 118, no. 1, p. e1907856118, 2021.
[91] S. Anderson, S. Peters, K. Iagnemma, and J. Overholt, “Semi-autonomous stability control and hazard avoidance for manned and unmanned ground vehicles,” presented at the the 27th Army Science Conference, Orlando, Florida, USA, November 29-December 02, 2010, 2010, pp. 1–8.
[92] J. Morales, J. L. Martínez, M. A. Martínez, and A. Mandow, “Pure-pursuit reactive path tracking for nonholonomic mobile robots with a 2D laser scanner,” *EURASIP J. Adv. Signal Process.*, vol. 2009, pp. 1–10, 2009.
[93] J. Storms, K. Chen, and D. Tilbury, “A shared control method for obstacle avoidance with mobile robots and its interaction with communication delay,” *Int. J. Robot. Res.*, vol. 36, no. 5–7, pp. 820–839, 2017.
[94] P. J. Durst *et al.*, “A real-time, interactive simulation environment for unmanned ground vehicles: The autonomous navigation virtual environment laboratory (ANVEL),” presented

at the 2012 Fifth international conference on information and computing science, IEEE, 2012, pp. 7–10.

[95] P. Pappas, M. Chiou, G.-T. Epsimos, G. Nikolaou, and R. Stolkin, "Vfh+ based shared control for remotely operated mobile robots," presented at the 2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), IEEE, 2020, pp. 366–373.

[96] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," presented at the Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No. 98CH36146), IEEE, 1998, pp. 1572–1577.

[97] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," presented at the 2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566), IEEE, 2004, pp. 2149–2154.

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

[99] E. F. Mohamed, K. El-Metwally, and A. R. Hanafy, "An improved Tangent Bug method integrated with artificial potential field for multi-robot path planning," in *2011 International Symposium on Innovations in Intelligent Systems and Applications*, Jun. 2011, pp. 555–559. doi: 10.1109/INISTA.2011.5946136.

[100] A. A. Al-Haddad, R. Sudirman, C. Omar, and S. Z. M. Tumari, "Wheelchair motion control guide using eye gaze and blinks based on Bug algorithms," in *2012 IEEE-EMBS Conference on Biomedical Engineering and Sciences*, Dec. 2012, pp. 398–403. doi: 10.1109/IECBES.2012.6498151.

[101] A. M. Mohsen, M. A. Sharkas, and M. S. Zaghlol, "New Real Time (M-Bug) Algorithm for Path Planning and Obstacle Avoidance In 2D Unknown Environment," in *2019 29th International Conference on Computer Theory and Applications (ICCTA)*, Oct. 2019, pp. 25–31. doi: 10.1109/ICCTA48790.2019.9478801.

[102] A. Saglam and Y. Papelis, "Realtime Corridor Detection for Mobile Robot Navigation with Hough Transform Using a Depth Camera," presented at the 2021 21st International Conference on Control, Automation and Systems (ICCAS), IEEE, 2021, pp. 683–688.

[103] R. Ebrahimpour, R. Rasoolinezhad, Z. Hajiabolhasani, and M. Ebrahimi, "Vanishing point detection in corridors: using Hough transform and K-means clustering," *IET Comput. Vis.*, vol. 6, no. 1, pp. 40–51, 2012.

[104] W. Shi and J. Samarabandu, "Corridor line detection for vision based indoor robot navigation," presented at the 2006 Canadian Conference on Electrical and Computer Engineering, IEEE, 2006, pp. 1988–1991.

[105] J. H. Yoo, S.-W. Lee, S.-K. Park, and D. H. Kim, "A robust lane detection method based on vanishing point estimation using the relevance of line segments," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 12, pp. 3254–3266, 2017.

[106] Y. Zhou, G. Jiang, G. Xu, X. Wu, and L. Krundel, "Kinect depth image based door detection for autonomous indoor navigation," presented at the The 23rd IEEE International Symposium on Robot and Human Interactive Communication, IEEE, 2014, pp. 147–152.

[107] P. Kultanen, L. Xu, and E. Oja, "Randomized hough transform (rht)," presented at the [1990] Proceedings. 10th International Conference on Pattern Recognition, IEEE, 1990, pp. 631–635.

[108] G. Olmschenk and Z. Zhu, "3D hallway modeling using a single image," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 158–164.

[109] S. Gupta, R. Sangeeta, R. S. Mishra, G. Singal, T. Badal, and D. Garg, “Corridor segmentation for automatic robot navigation in indoor environment using edge devices,” *Comput. Netw.*, vol. 178, p. 107374, 2020.
[110] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[111] R. Hulik, M. Spanel, P. Smrz, and Z. Materna, “Continuous plane detection in point-cloud data based on 3D Hough Transform,” *J. Vis. Commun. Image Represent.*, vol. 25, no. 1, pp. 86–97, 2014.
[112] L. Xu, C. Feng, V. R. Kamat, and C. C. Menassa, “An occupancy grid mapping enhanced visual SLAM for real-time locating applications in indoor GPS-denied environments,” *Autom. Constr.*, vol. 104, pp. 230–245, 2019.
[113] J. Larsson, M. Broxvall, and A. Saffiotti, “Laser-based corridor detection for reactive navigation,” *Ind. Robot Int. J.*, 2008.
[114] H. Moradi, J. Choi, E. Kim, and S. Lee, “A real-time wall detection method for indoor environments,” presented at the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2006, pp. 4551–4557.
[115] J. Canny, “A computational approach to edge detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 6, pp. 679–698, 1986.
[116] O. R. Vincent and O. Folorunso, “A descriptive algorithm for sobel image edge detection,” presented at the Proceedings of informing science & IT education conference (InSITE), Informing Science Institute California, 2009, pp. 97–107.
[117] R. B. Rusu and S. Cousins, “3d is here: Point cloud library (pcl),” presented at the 2011 IEEE international conference on robotics and automation, IEEE, 2011, pp. 1–4.
[118] P. V. Hough, “Method and means for recognizing complex patterns,” Dec. 1962.
[119] N. Kiryati, Y. Eldar, and A. M. Bruckstein, “A probabilistic Hough transform,” *Pattern Recognit.*, vol. 24, no. 4, pp. 303–316, 1991.
[120] “OpenCV: Feature Detection.” Accessed: Mar. 26, 2021. [Online]. Available: https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga8618180a5948286384e3b7ca02f6feeb
[121] M. R. Endsley, “Toward a theory of situation awareness in dynamic systems,” *Hum. Factors J. Hum. Factors Ergon. Soc.*, vol. 37, no. 1, pp. 32–64, 1995.
[122] D. A. Lawrence, “Stability and transparency in bilateral teleoperation,” *IEEE Trans. Robot. Autom.*, vol. 9, no. 5, pp. 624–637, 1993.
[123] N. Ohnishi and A. Imiya, “Corridor navigation and obstacle avoidance using visual potential for mobile robot,” presented at the Fourth Canadian Conference on Computer and Robot Vision (CRV’07), IEEE, 2007, pp. 131–138.
[124] A. I. Comport, E. Malis, and P. Rives, “Accurate quadrifocal tracking for robust 3d visual odometry,” presented at the Proceedings 2007 IEEE International Conference on Robotics and Automation, IEEE, 2007, pp. 40–45.
[125] A. Howard, M. J. Matarić, and G. S. Sukhatme, “An incremental self-deployment algorithm for mobile sensor networks,” *Auton. Robots*, vol. 13, pp. 113–126, 2002.
[126] J. S. Park, B. Tsang, H. Yedidsion, G. Warnell, D. Kyoung, and P. Stone, “Learning to improve multi-robot hallway navigation,” presented at the Conference on Robot Learning, PMLR, 2021, pp. 1883–1895.
[127] L. Sharma and J. P. How, “Look Before You Leap: Socially Acceptable High-Speed

Ground Robot Navigation in Crowded Hallways," *ArXiv Prepr. ArXiv240313284*, 2024.
[128] J. R. Millan, F. Renkens, J. Mourino, and W. Gerstner, "Noninvasive brain-actuated control of a mobile robot by human EEG," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 1026–1033, 2004.
[129] A. Saglam and Y. Papelis, "Efficient Maritime Object Detection and Validation for Enhancing Safety of Uncrewed Marine Systems," 2023.
[130] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. Syst. Man Cybern.*, vol. 19, no. 5, pp. 1179–1187, 1989.
[131] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots in cluttered environments," presented at the Proceedings., IEEE International Conference on Robotics and Automation, IEEE, 1990, pp. 572–577.
[132] A. N. Catapang and M. Ramos, "Obstacle detection using a 2D LIDAR system for an Autonomous Vehicle," presented at the 2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), IEEE, 2016, pp. 441–445.
[133] L. Chen, J. Yang, and H. Kong, "Lidar-histogram for fast road and obstacle detection," presented at the 2017 IEEE international conference on robotics and automation (ICRA), IEEE, 2017, pp. 1343–1348.
[134] D. Hutabarat, M. Rivai, D. Purwanto, and H. Hutomo, "Lidar-based obstacle avoidance for the autonomous mobile robot," presented at the 2019 12th International Conference on Information & Communication Technology and System (ICTS), IEEE, 2019, pp. 197–202.
[135] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, "The obstacle detection and obstacle avoidance algorithm based on 2-d lidar," presented at the 2015 IEEE international conference on information and automation, IEEE, 2015, pp. 1648–1653.
[136] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.
[137] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, 1991.
[138] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets: Open-source library and experimental protocol," *Auton. Robots*, vol. 34, pp. 133–148, 2013.
[139] Y. He, B. Liang, J. Yang, S. Li, and J. He, "An iterative closest points algorithm for registration of 3D laser scanner point clouds with geometric features," *Sensors*, vol. 17, no. 8, p. 1862, 2017.
[140] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," presented at the Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2147–2156.
[141] C. Li, J. Ku, and S. L. Waslander, "Confidence guided stereo 3D object detection with split depth estimation," presented at the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 5776–5783.
[142] J. Mao, S. Shi, X. Wang, and H. Li, "3D object detection for autonomous driving: A comprehensive survey," *Int. J. Comput. Vis.*, vol. 131, no. 8, pp. 1909–1963, 2023.
[143] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," presented at the Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.
[144] J. O. Pinzón-Arenas and R. Jiménez-Moreno, "Obstacle detection using faster R-CNN oriented to an autonomous feeding assistance system," presented at the 2020 3rd

International Conference on Information and Computer Technologies (ICICT), IEEE, 2020, pp. 137–142.
[145] B. Jähne, *Digital image processing*. Springer Science & Business Media, 2005.
[146] S. Suzuki, "Topological structural analysis of digitized binary images by border following," *Comput. Vis. Graph. Image Process.*, vol. 30, no. 1, pp. 32–46, 1985.
[147] A. Saalfeld, "Topologically consistent line simplification with the Douglas-Peucker algorithm," *Cartogr. Geogr. Inf. Sci.*, vol. 26, no. 1, pp. 7–18, 1999.
[148] "OpenCV: Contour Features." Accessed: May 15, 2024. [Online]. Available: https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html
[149] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, 2022, pp. 287–290.

VITA

Ahmet Saglam
asagllam@gmail.com

Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23529

EDUCATION

Master of Engineering Aug 2017- May 2019
Computational Modeling and Simulation Engineering, Old Dominion University, Norfolk, VA

Master of Science (Coursework only). Sep 2015 - Dec 2016
Modeling, Virtual Environments, and Simulation, Naval Postgraduate School, Monterey, CA

Bachelor of Science Sep 2004 - Aug 2008
Systems Engineering, Military Academy, Ankara, Turkey.

PROFESSIONAL EXPERIENCE

Lead Project Scientist Jan 2022- Aug 2024
Office of Enterprise Research and Innovation, Old Dominion University, Norfolk, VA
Developed blended control algorithm in C++ for safe and efficient teleoperation of mobile robots and unmanned surface vessels.

Research Assistant Aug 2017- Dec 2021
Virginia Modeling, Analysis, and Simulation Center, Old Dominion University, Norfolk, VA
Developed efficient perception pipeline using 3D sensors in ROS; applied image processing techniques using OpenCV.
Developed a multi robot simulation environment on ROS / Gazebo.

ACADEMIC ACHIEVEMENTS

Ahmet Saglam and Yiannis Papelis, Safe and Efficient Operation of Emotional Support Robots: A Risk-Based Approach with User-Centric Tangent Bug for Blended Control. IEEE RO-MAN 2024. (Conference Paper, accepted).

Ahmet Saglam and Yiannis Papelis, A Simulation-Based Approach for Evaluating Shared Control Algorithms for Mobile Robots. ANNSIM 2024. (Conference Paper, accepted). Ahmet Saglam and Yiannis Papelis, Efficient Maritime Object Detection and Validation for Enhancing Safety of Uncrewed Marine Systems. I3M 2023. (Conference Paper, Best Paper).

Christopher Adolphi, Dorothy Dorie Parry, Yaohang Li, Masha Sosonkina, **Ahmet Saglam**, LiDAR Buoy Detection for Autonomous Marine Vessel Using Pointnet Classification. MSV 2023. (Conference paper)

Ahmet Saglam and Yiannis Papelis, Real-time Corridor Detection for Mobile Robot Navigation with Hough Transform Using a Depth Camera. MSV 2021. (Conference paper).

Ahmet Saglam and Yiannis Papelis, Scalability of sensor simulation in ROS-Gazebo platform with and without using GPU. SPRINGSIM 2020. (Conference paper).