Informatics Institute of Technology

In Collaboration with

University of Westminster, UK

University of Westminster, Coat of Arms

Beyond Line of Sight: Occlusion-Aware Perception for Human-Following Robots

A Dissertation by

T.M.S. Sachintha

w1960512 | 20221724

Supervised by

Prof. G.D.S.P. Wimalaratne

Submitted in partial fulfillment of the requirements of the

MSc in Advance Software Engineering degree at the University of Westminster.

July 2024

ABSTRACT

Human Following Robots (HFR)s have gained significant attention in recent years due to their potential applications in various domains, such as personal assistance, security, and entertainment. However, one of the significant challenges HFRs face is handling occlusions, where the human subject is partially or completely obstructed from the robot's sensors. This research project addresses this challenge by developing an occlusion-aware perception system for HFRs.

The proposed system integrates data from wearable trackers and indoor localization systems to accurately map the environment and track the human subject's movements. By combining multiple input sources, the system aims to overcome the limitations of vision-based tracking methods and effectively handle scenarios of complete occlusion. The research project involves designing algorithms for handling complete occlusion, implementing a simulated custom wearable tracking device, and building simulated environments for testing and validation.

Testing results reveals that the runtime for occlusion recovery increases with the number of static obstacles. In unknown environments, the average recovery time ranges from 10.06 seconds for 2 obstacles to 24.91 seconds for 9 obstacles, while in known environments, it improves to 7.09 seconds for 2 obstacles and 17.08 seconds for 9 obstacles. Accuracy testing demonstrates that the system maintains high success rates, with 100% successful recoveries and average times between 6.08 and 6.64 seconds for Ultra-Wideband (UWB) error margins of 5% to 10%. However, the recovery success rate drops to 80% when the error margin increases to 12% and 15%. These results underscore the system's potential in enhancing occlusion handling while highlighting areas for further refinement in diverse scenarios.

Keywords: Human-Following Robots, Occlusion Handling, Full Occlusion, Partial Occlusion,UWB Indoor Localization, Wearable Tracking Devices

Subject Descriptors

- Computer Systems Organizations → Embedded and Cyber-physical systems → Robotics → Robotic Control
- Computing Methodologies → Modeling and Simulation → Simulation Support Systems → Simulation Environments

DECLARATION

I hereby declare that this dissertation is the result of my own independent research and has not been previously submitted, nor is it currently being submitted, for any degree or other qualification at any university or institution. All sources of information, data, and references from external sources have been duly acknowledged and cited appropriately.

Student Name: T.M. Shakthi Sachintha

Registration Number: w1960512 | 20221724

Signature:  Date: 29/07/2024

ACKNOWLEDGMENT

The completion of this research project has been a challenging yet enriching journey. I am deeply grateful to those who supported me throughout this process.

Firstly, I would like to extend my heartfelt gratitude to my supervisor, Prof. G.D.S.P. Wimalaratne, for his invaluable guidance, support, and encouragement. His expertise has been fundamental to my research.

I also want to express my appreciation to Mr. Guhanathan Poravi for his significant contributions, continuous guidance, and support throughout the project for all the students.

I am thankful to the domain and technical experts, as well as the evaluators of this research, whose feedback helped me refine and improve my work.

Finally, my deepest thanks go to my friends and family, whose unwavering support and encouragement have been essential during this final year project and my entire university journey. Their belief in me has been a source of strength and motivation.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

HFR Human Following Robots

IMU Inertial Measurement Unit

UWB Ultra-Wideband

FOV Field of View

IDE Integrated Development Environment

LiDAR Light Detection and Ranging

UAV Unmanned Aerial Vehicle

RFID Radio Frequency Identification

BLE Bluetooth Low Energy

TOF Time Of Flight

EMA Exponential Moving Average

1 INTRODUCTION

1.1 Chapter Overview

The introduction chapter comprehensively overviews the challenges and complexities inherent in HFR. It delves into the intricacies of occlusions in HFR, exploring how these obstacles hinder tracking accuracy and navigation. The chapter highlights the importance of robust perception systems and algorithms to overcome occlusion-related challenges effectively. Additionally, it discusses various methods for subject tracking in HFR and examines the limitations of existing approaches, paving the way for innovative solutions. The chapter concludes by presenting the problem statement and research questions, setting the stage for the subsequent chapters to address these challenges and advance the field of HFR.

1.2 Problem Domain

1.2.1 Human Following Robots

HFR are autonomous robotic systems that track and follow a human target in various environments, facilitating human-robot interaction and collaboration. These robots are crucial in various applications, including manufacturing, healthcare, entertainment, and social interactions. HFR can be categorized into indoor and outdoor scenarios based on their operating environment (Islam, Hong, and Sattar, 2019).

Occlusions significantly impact HFR by obstructing the visual contact with the robot and the human subject. This leads to challenges in maintaining tracking accuracy and ensuring safe navigation. Robust perception systems and algorithms are necessary for effective occlusion handling.

1.2.2 Subject Tracking

Subject tracking in HFR is crucial for accurately following designated human subjects. Custom trackers, such as those utilizing LEDs or color tags, are commonly employed for this purpose (Nagumo and Ohya, 2001; Hassan et al., 2016; Jeyatharan, Ekanayake, and Sandaruwan, 2021). These trackers enable autonomous robots to track human trajectories effectively. However, Deep

Learning-based algorithms, like those using the Microsoft Kinect camera, offer alternative methods by analyzing soft biometrics and clothing color (Sun et al., 2016). Integration with Hokuyo laser sensors in the Robot Operating System (ROS) expands the tracking capabilities (Priyandoko et al., 2017). Despite these advancements, such methods may lack robustness, especially in scenarios with similar-looking subjects sharing common features like clothing color.

Moreover, integrating ultrasonic sensors presents a lighting-independent tracking solution in HFR. One approach utilizes a curved arrangement of three ultrasonic sensors to maximize leg tracking range (Tai et al., 2019). Another system employs a laser range scanner for shin detection. However, both methods face challenges such as misclassifying nearby objects and may not be ideal for crowded environments.

1.2.3 Occlusions

In HFR, occlusion manifests in two primary forms: partial and complete. Partial occlusion occurs when obstacles or other individuals partially obstruct the robot's view, which is common in dynamic environments like crowded spaces. Computer vision-based tracking systems, like that by Jeyatharan, Ekanayake, and Sandaruwan, 2021, adeptly handle partial occlusions. Techniques such as Single Shot Detectors (MobileNet) or SVM classifiers effectively detect and predict partial occlusions, enhancing tracking performance (Ye et al., 2023; Shu et al., 2012).

In contrast to partial occlusion, **Full Occlusion** occurs when the human subject is entirely obstructed from the robot's view, rendering them invisible to the system. Full occlusion can arise in various scenarios, including:

- **Doorways and Corridors:** In narrow passages, like doorways or corridors, the human subject may become fully occluded.
- **Obstructive Objects:** Large objects, such as vehicles or walls, can entirely obstruct the view of the human subject.
- **Crowded Spaces:** In dense areas or gatherings, numerous individuals moving in close proximity can hide the human subject from view.

Addressing full occlusion scenarios is a significant challenge for HFR systems, requiring robust methods to reacquire the target once it becomes fully occluded.

1.3 Problem Definition

The problem involves addressing the challenges of occlusion scenarios in human-following robots (HFRs). These challenges include partial and complete occlusions, which can hinder the robot's ability to track and follow a designated human subject accurately(Islam, Hong, and Sattar, 2019). Existing solutions, primarily reliant on AI-based computer vision programs, often need help with misclassification errors, particularly in scenarios with similar-colored clothing or shared body characteristics among individuals (Shu et al., 2012; Kulkarni and Pantawane, 2022). This highlights the need for alternative tracking methods, such as wearable devices, which offer more reliable localization data and improved performance in occlusion-prone environments (Jeyatharan, Ekanayake, and Sandaruwan, 2021). However, there needs to be more understanding of the optimal fusion strategies for integrating data from various sensors, such as IMU sensors and UWB localization modules, to enhance HFR performance in occluded scenarios (D. Feng et al., 2020). Additionally, current approaches often must adapt to diverse lighting conditions, impacting tracking accuracy and navigation. Hence, the research aims to develop novel algorithms and sensor fusion techniques to improve occlusion handling in HFRs, aiming to enhance tracking accuracy, robustness, and adaptability to real-world environments.

1.3.1 Problem Statement

Human-following robots encounter challenges when the human they are tracking is entirely obscured. Existing systems struggle in scenarios where visual contact is momentarily lost, leading to navigation issues.

1.4 Research Motivation

The motivation behind this research stems from the increasing demand for Human Following Robots (HFR) in diverse applications such as personal assistance, security, and entertainment. Despite their potential, HFRs face significant challenges due to occlusions, where the human subject may be partially or completely obstructed from the robot's sensors (Kumar Ashe and Krishna, 2021). This research aims to develop an occlusion-aware perception system that integrates data from wearable trackers and indoor localization systems (D. Feng et al., 2020), enhancing the robot's ability to track human movements accurately. By addressing the limitations

of existing vision-based tracking methods, this study contributes to advancing HFR technology, promoting safer and more efficient interactions in real-world environments. The findings could significantly impact the future development of autonomous robotic systems.

1.5 Research Gap

Tracking a Wearable: Existing research primarily relies on AI-based computer vision programs for human tracking in human-following robots (Kastner et al., 2022). However, these programs exhibit limitations, as they are susceptible to errors such as misclassification, particularly in scenarios where individuals wear similar-colored clothing or share similar body characteristics (Algabri and M. T. Choi, 2020). Consequently, there needs to be more understanding of the potential advantages of tracking wearable devices, which offer a more robust alternative for human tracking in occlusion-prone scenarios (Jeyatharan, Ekanayake, and Sandaruwan, 2021). Unlike AI-based computer vision programs, tracking wearable devices can provide more reliable and accurate localization data, thereby addressing the limitations associated with traditional tracking methods and enhancing the overall performance of HFR in challenging environments.

Fuse Different Sensor Inputs: While sensor fusion techniques have shown promise in enhancing the robustness of robot navigation systems D. Feng et al., 2020, there exists a gap in understanding the optimal fusion strategies for integrating data from Inertial Measurement Unit (IMU) sensors, UWB localization modules, and other ranging sensors to improve human-following robots' performance in environments with varying levels of occlusion. Current literature needs comprehensive studies exploring sensor fusion's synergistic effects on enabling seamless reconnection between robots and occluded human subjects in real-world scenarios.

Diverse Lightning Conditions: The current landscape of subject-tracking mechanisms predominantly relies on camera-based computer vision algorithms. However, in practical scenarios, lighting conditions variations can significantly impair these systems' performance, leading to target loss and navigation failures (Kulkarni and Pantawane, 2022). This limitation highlights a critical gap in the field of HFR, as existing approaches often need help to adapt to diverse lighting conditions.

1.6 Contribution to the Body of Knowledge

This section outlines the study's contributions to the problem and broader autonomous robot navigation research domains.

1.6.1 Contribution to Research Domain

This study significantly contributes to the research domain of **Autonomous Robot Navigation**. The research advances the broader field of autonomous navigation systems by addressing the specific challenges related to handling occlusions in HFR. Investigating novel localization methods and sensor fusion techniques enhances the capabilities of human-following robots. It informs the development of more robust and adaptable navigation systems for various autonomous robotic applications. Consequently, this research contributes to advancing the state-of-the-art in autonomous robot navigation and lays the foundation for future developments in the field.

1.6.2 Contribution to Problem Domain

This research contributes significantly to addressing the challenges within the problem domain of enhancing the occlusion handling in HFR, investigating the possibility of combining indoor localization and existing human subject tracking methods to implement an occlusion-aware perception. While numerous studies in the literature aim to minimize occlusions in HFR (Yuan et al., 2021), none have effectively addressed occlusions after they occur.

With the integration of wearable devices equipped with sensors such as Inertial Measurement Units (IMUs) or Ultra-Wideband (UWB) localization systems, the study offers novel solutions to improve the robustness and accuracy of human tracking in occlusion-prone scenarios. Additionally, combining data fromUWB localization modules, and other ranging sensors, provides valuable insights into enhancing occlusion-aware perception, thus enabling human-following robots to navigate effectively in challenging environments.

1.7 Research Challenges

This research aims to devise a mechanism for locating individuals whom robots are following, even when they move out of the robot's Field of View (FOV) to locations not previously recorded.

Addressing this scenario presents several challenges, outlined below:

Dynamic Localization: Ensuring accurate localization of individuals who have moved out of the robot's FOV requires overcoming challenges associated with dynamic and unpredictable movements.

Limited Environmental Knowledge: Robots may lack prior knowledge of areas outside their FOV, necessitating strategies to navigate and locate individuals in unfamiliar environments.

Real-Time Processing: Processing sensor data and implementing occlusion-aware perception algorithms in real-time imposes computational challenges, particularly for resource-constrained robotic platforms.

Sensor Reliability: Relying on sensors to detect and track individuals presents challenges related to sensor accuracy, noise, and environmental interference, which may affect the reliability of location information.

Robustness to Environmental Variations: Human-following robots must operate in diverse environments with varying lighting conditions, obstacles, and terrain. Ensuring the robustness of occlusion-aware perception systems to such environmental variations is a significant challenge.

1.8 Research Questions

Question 01: How do wearable devices for active human tracking, such as IMU sensors, or UWB localization systems, improve human-following robots in occlusion-prone scenarios?

Question 02: How does sensor fusion, combining data from, UWB localization, and other ranging sensors, improve human-following robots' performance in scenarios with different levels of occlusion?

Question 03: How can occlusion-aware perception, leveraging a combination of sensor data and adaptable algorithms, improve human following in different lighting conditions?

1.9 Research Aim and Objectives

1.9.1 Research Aim

Design, develop, and evaluate occlusion-aware perception algorithms for HFR by integrating wearable devices and sensor fusion techniques to enhance accuracy and robustness in tracking human subjects prone to occlusion, facilitating seamless navigation and interaction in dynamic environments.

To further elaborate, this project aims to investigate the effectiveness of wearable devices such as IMU sensors and UWB localization systems for active human tracking and explore sensor fusion strategies to integrate data from multiple sensors, ultimately improving occlusion handling and adaptability to diverse lighting conditions in HFR systems.

1.9.2 Research Objectives

Research Objectives	**Explanation**	**Learning Outcome**	**Research Question**
Problem Identification	**RO1**: To identify key challenges in existing HFR systems, particularly in handling occlusion scenarios. **RO2**: To determine the limitations of current tracking methods and their impact on HFR performance. **RO3**: To explore the need for improved occlusion handling techniques in HFR systems.	LO1, LO2	RQ1, RQ2

Literature Review	**RO1**: To explore the current direction in HFR systems. **RO2**: To identify the existing occlusion handling methods in the literature. **RO3**: To identify the computationally low cost methods localization and subject tracking.	LO1, LO2, LO4, LO6	RQ1, RQ2, RQ3
Requirement Analysis	**RO1**: To identify the hardware capabilities of the available sensor modules and localization technologies. **RO2**: To determine the specifications for sensor fusion algorithms to integrate data from wearable trackers and localization systems.	LO2, LO3	RQ1, RQ2, RQ3
Design	**RO1**: To develop a framework for indoor localization using selected methods to track human subjects and robot movements. **RO2**: To design algorithms for sensor fusion to integrate data from wearable trackers and localization systems, enabling accurate mapping for robot navigation.	LO1, LO3, LO5	RQ2, RQ3
Development	**RO1**: To implement the simulated custom wearable tracking device prototype. **RO2**: To build simulated environments and frameworks for testing and validation of the developed components.	LO6, LO4, LO5	RQ1, RQ2

Testing and Evaluation	**RO1**: To evaluate the performance of indoor localization methods and algorithms in tracking human subjects and guiding robots. **RO2**: To assess the overall effectiveness of the developed system in handling occlusion scenarios, different light conditions and improving HFR navigation capabilities.	LO2, LO5, LO6	RQ1, RQ2, RQ3

Table 1.1: Research Objectives

1.10 Chapter Summary

This chapter explored the intricate domain of HFR, focusing on the challenges posed by occlusions. We discussed the significance of HFR systems and their applications, highlighting the critical role of occlusion handling in ensuring effective human tracking and navigation. We identified critical research gaps and formulated the problem by analyzing subject-tracking methods and occlusion scenarios. The chapter also introduced the research questions, outlining the scope of the study and setting the foundation for further exploration.

2 LITERATURE REVIEW

2.1 Chapter Overview

This chapter delves into the complexities of HFR, explicitly focusing on the critical challenge of occlusion handling. Starting from a concept map to ease the literature survey, followed by introducing the concept of HFR, outlining various use cases and behavioral categories. The chapter examines diverse human subject tracking methodologies, from custom trackers and deep learning-based approaches to non-visual sensing techniques. It classifies occlusions (partial, full, dynamic) along with their causes and challenges for HFR systems. Further, the chapter reviews existing research addressing partial and full occlusion scenarios, exploring their strengths and limitations. Finally, a comprehensive survey is presented enabling technologies and algorithmic foundations in occlusion-aware HFR, including computer vision, depth sensing, localization, estimation techniques, and robot architectures.

2.2 Concept Map

The scope outlined in this review was deconstructed using a concept map (See Figure A: .1 Concept Map (Self - Composed)) following an extensive literature survey. This graphical representation was constructed to guarantee comprehensive coverage of literature about the problem domain, existing research, technologies, and evaluation methodologies.

2.3 Problem Domain

2.3.1 Human Following Robots

HFR are autonomous robotic systems that track and follow a human target in various environments, facilitating human-robot interaction and collaboration. These robots are crucial in various applications, including manufacturing, healthcare, entertainment, and social interactions. HFR can be further categorized into the followings based on how the subject and robots positions itself.

Variations of Human Following Robots

Guiding Robots: Guiding robots are designed to lead or direct a person to a specific location or destination. These robots typically use localization and navigation algorithms to determine the optimal path and guide the user safely. Kastner et al. (2022) presents a human guiding robot using deep reinforcement learning in crowded environments.

Side by Side Following: Similar to walking with a human partner, side-by-side following involves the robot accompanying the person at their side. This mode may be used when companionship or assistance is needed, such as for the elderly or individuals with mobility impairments (Xue et al., 2022).

Following Ahead: Some robots are programmed to follow ahead of the person, leading the way and capturing moments or assisting in tasks (Islam, Hong, and Sattar, 2019). For example, Unmanned Aerial Vehicle (UAV)s are often used in sports events to capture aerial footage, and they need to follow ahead of athletes or vehicles to maintain a clear view.

Following Behind: Following behind is perhaps the most typical behavior exhibited by HFR. These robots maintain a certain distance behind the person they are tracking, adjusting their speed and trajectory to keep pace and avoid collisions (Jeyatharan, Ekanayake, and Sandaruwan, 2021).

2.3.2 Human Subject Tracking

The primary objective of HFR is to follow the target human subject, requiring accurate detection and distinction from other individuals in the environment.

Utilizing a custom tracker attached to the human subject is a popular and effective method for achieving this goal. For example, a previous study introduces a novel approach where an autonomous mobile robot with a camera follows a human carrying a light-emitting device with two LEDs. The robot tracks the human's trajectory by capturing the direction from these LEDs in the image, leveraging the camera's pan and the on-off flashing LEDs (Nagumo and Ohya, 2001). In another study, Hassan et al. (2016) utilizes a computationally low-cost algorithm that relies on a specific color tag placed on the human subject, detected by a camera mounted on the robot. A similar method is employed by Jeyatharan, Ekanayake, and Sandaruwan (2021), where

a tricolored trapezoidal belt around the subject's waist is used to track the subject. Placing a custom tracker on the subject is a reliable and effective approach for HFR robotics applications.

In contrast, Deep Learning-based algorithms for human tracking have also been widely employed. One such example involves utilizing the Microsoft Kinect camera to track humans by analyzing soft biometrics, which includes body features like shoulder width, arm length, and clothing color, using both camera input and depth information from the Kinect module (Sun et al., 2016). Another study combines the Microsoft Kinect sensor with a Hokuyo laser sensor. It presents four tracking methods: face detection, leg detection, color detection, and person blob detection, implemented within the Robot Operating System (ROS) environment (Priyandoko et al., 2017). However, these methods may need more robustness in human tracking performance, as they can be susceptible to detecting similar subjects who share the same features, such as individuals wearing similar-colored clothing, leading to potential confusion and tracking errors.

Additionally, integrating ultrasonic sensors offers a solution for tracking subjects regardless of lighting conditions, as it operates independently of visual data. One approach utilizes three ultrasonic sensors arranged in a curve to maximize the range and effectively track the subject's legs (Tai et al., 2019). Another study presents a HFR system that utilizes a laser range scanner for tracking based on shin detection (Kawarazaki, Kuwae, and Yoshidome, 2015). However, both methods may encounter challenges such as misclassifying nearby objects (e.g., chairs or table legs) as human shins or legs, and they may not be suitable for crowded and dynamic environments. Tai et al. (2019) presents a solution for human tracking using a mobile robot that integrates laser-based leg detection and face recognition techniques, proving effective even in cluttered environments.

2.3.3 Types and Causes of Occlusions

In the context of HFRs, occlusion happens in three major forms: partial fully and dynamic occlusion.

Partial Occlusion and **Dynamic Occlusion** arises when the human subject is partially obstructed from the robot's FOV, which can happen for various reasons. In dynamic environments, such as crowded spaces, the presence of another walking human can lead to partial occlusion. Additionally, obstacles in the environment, such as furniture or equipment, and dynamic scene

changes, such as movements of objects or people, can intermittently obscure the human subject from the robot's perspective, contributing to partial occlusion scenarios. Computer vision-based human tracking systems are generally adept at handling these types of partial occlusion (Ye et al., 2023). For instance, Jeyatharan, Ekanayake, and Sandaruwan (2021) integrates a Single Shot Detector (MobileNet) model to track humans, utilizing a color-coded trapezoidal belt around the subject's waist. This model exhibits good efficiency in handling partial occlusion scenarios. SVM classifiers can be utilized to learn from part-based human detection models to dynamically distribute the classifier's score among its corresponding parts, enabling effective detection and prediction of partial occlusions while enhancing tracking performance (Shu et al., 2012).

In contrast to partial occlusion, Full Occlusion occurs when the human subject is entirely obstructed from the robot's view, rendering them invisible to the system. Fully occlusion can arise in various scenarios, including:

- **Doorways and Corridors:** In narrow passages, like doorways or corridors, the human subject may become fully occluded.
- **Obstructive Objects:** Large objects, such as vehicles or walls, can entirely obstruct the view of the human subject.
- **Crowded Spaces:** In dense areas or gatherings, numerous individuals moving in close proximity can hide the human subject from view.

Kumar Ashe and Krishna (2021) proposed a strategy to mitigate the occurrence of full occlusion scenarios by employing proactive measures. By leveraging map information, the robot strategically relocates to advantageous positions ahead of intersections and potential occlusions, thereby reducing the likelihood of losing track of the target individual. Nonetheless, this approach is constrained by the prerequisite of prior knowledge about the environment, rendering it unsuitable for dynamic environments.

2.4 Existing Work

2.4.1 Partial Occlusion Handling

The research by Ye et al. (2023) introduces a novel method for addressing the challenge of HFR in scenarios characterized by partial occlusion due to a monocular camera's limited field of view. The proposed method leverages any visible joint of the target person to estimate their location on the ground plane, utilizing a prior model of the person's joint heights. This approach integrates multiple components, including a 2D person detector, a 2D pose detector, a visible-joint-based location tracker, and a robot motion controller, to form a complete HFR system. Specifically, deep learning models such as YOLOX and AlphaPose are employed for 2D human bounding-box and joint detection, respectively. These models facilitate the detection of the person and their visible joints in the image plane. However, several limitations exist in this approach. The proposed solution incurs a high computational cost, which may pose practical challenges, especially in real-time applications. Moreover, the method assumes certain conditions, such as a standing person on the ground plane and a calibrated camera, which may not be applicable in all scenarios, limiting its generalizability. Furthermore, the approach's effectiveness heavily relies on the accuracy and robustness of the 2D joint detector, which may fail in cases of severe occlusion or complex poses. Lastly, the method does not account for factors such as the orientation or pose of the person, which could significantly impact the performance of the person following algorithms.

The paper presented by Algabri and M. T. Choi (2020) introduces a novel framework utilizing state-machine control, enabling a robot to track subjects in occluded and varying lighting conditions effectively. The framework incorporates a high-accuracy, low-computational-cost deep learning-based algorithm for real-time detection and tracking of individuals. However, the approach has several limitations. It relies on a separate computer for deep learning tasks, making the robot dependent and not self-sustainable. A constant Wi-Fi link is also necessary to communicate with the workstation, potentially limiting the robot's mobility. The method identifies humans based on color features, rendering it ineffective if multiple individuals wear similar colors. Furthermore, occlusions are only addressed within the camera's field of view, and the system struggles in low-light scenarios.

The research presented by Jeyatharan, Ekanayake, and Sandaruwan (2021) focuses on a HFR

system using behavior based tracking, aiming to navigate dynamic environments while following a target person. The study implements a tracker based target detection system using a custom made tri-colored belt and a fuzzy logic based simplified robot control architecture(subsumption). This approach enhances the robot's tracking accuracy, movement smoothness, and multitasking efficiency in dynamic environments. By leveraging advanced algorithms and components such as Convolutional Neural Networks (CNN), Single Shot Multibox Detector (SSD), Raspberry Pi, and ultrasonic sensors, the project contributes significantly to robotics for human-following applications. However, the study identifies limitations, including the Raspberry Pi 4 main control board's frame rate limitations due to high CPU power consumption. Additionally, the research did not address complete occlusion scenarios in the vision-based system, affecting target detection under certain conditions.

2.4.2 Fully Occlusion Handling

The study conducted by Hoang, Yun, and J. S. Choi (2017) introduces a robust recovery mechanism designed for HFR to navigate when the target is lost effectively. This mechanism capitalizes on prior information, including the human's previous positions before track loss, to predict and navigate toward the human target. Employing a probabilistic approach leveraging the Kalman Filter, the system anticipates unexpected human positions, enhancing the robot's ability to reacquire the target swiftly. Additionally, the method integrates support vector machine (SVM) techniques for detecting the lower part of the human body, further enhancing the tracking accuracy. However, a notable limitation of this approach is its dependency on having prior map information, which may only sometimes be readily available in dynamic environments.

Kumar Ashe and Krishna (2021) delves into the application of model predictive control (MPC) with an early-relocation (ER) strategy to enhance the navigation capabilities of HFR, particularly in scenarios involving intersections and occlusions. The MPC framework, coupled with the ER strategy, prioritizes maintaining the target person's visibility and integrating social representation into the controller's decision-making process. By leveraging map information, the robot proactively relocates to advantageous positions ahead of potential occlusions, minimizing the risk of losing track of the target individual. However, this approach necessitates prior knowledge of the travel environment to identify suitable ER points, posing a limitation in dynamic settings. While the 2D simulations demonstrate promising maneuvering capabilities and near

real-time person-following behavior, the study remains confined to simulation environments. Additionally, while the system effectively minimizes the likelihood of occlusion, it does not directly handle occlusion scenarios.

The paper presented by Yuan et al. (2021) introduces a novel method for laser-based intersection-aware human following with a mobile robot in indoor environments. By incorporating intersection information into a potential field-based human tracker, the system effectively addresses the challenge of occlusion caused by walls at corridor intersections. The robot constructs a topological map using an improved generalized Voronoi graph, enhancing navigation capabilities and mapping accuracy. Simulation and experimental results demonstrate the efficacy of the proposed method in avoiding occlusions and achieving robust human following and mapping performance. However, the method may not be suitable for human following in narrow corridors with frequent total occlusions, limiting its applicability in specific environments. Furthermore, the approach primarily focuses on improving the estimation accuracy of target states rather than continuous human following, and its high computational complexity could pose challenges in real-time implementation.

2.4.3 Summary of Existing Works

Citation	**Contributions**	**Limitations**
(Ye et al., 2023)	A novel method for robot person following under partial occlusion using visible joints.	Handles only the partial occlusion. Relies on the accuracy and robustness of the 2D joint detector, which fails in cases of severe occlusion or complex poses.

(Algabri and M. T. Choi, 2020)	Creates a high accuracy and low computationally cost Deep Learning based algorithm to detect and track people in real time.	Identifies human using color features prone to errors if everyone is wearing same color cloths. Occlusion is handled only camera FOV rotates and search If completely missed no recovery is possible.
(Jeyatharan, Ekanayake, and Sandaruwan, 2021)	Implements a tracker based target detection system with a tri-colored belt and a simplified robot control architecture (subsumption) based on fuzzy logic.	Only partial occlusion is handled, impacting target detection in certain conditions. The Raspberry Pi 4 main control board has limitations in frame rate due to high CPU power consumption.
(Hoang, Yun, and J. S. Choi, 2017)	Proposed a recovery mechanism for HFR leverages prior information, including the human's previous positions before losing track by employing a probabilistic approach using the Kalman Filter, unexpected human positions are predicted.	The recovery mechanism uses a probabilistic approach based on Kalman filter which may not account for the non-linear and uncertain dynamics of the human motion

(Kumar Ashe and Krishna, 2021)	Presents Model Predictive Control (MPC) framework for person-following robots that employs an Early Relocation (ER) strategy.	Need a map of the travelling location before hand to identify the ER (Early Relocation) points. Does not actually handles the occlusion but minimizing the chance of getting occluded.
(Yuan et al., 2021)	Simultaneous human following and topological mapping: Combines intersection-aware following and multihypothesis mapping for efficient exploration while following a human.	High-power laser scanner, which may be costly and pose safety risks. Not work well in cluttered or dynamic environments, where the laser beams may be blocked or scattered by obstacles or moving objects.

Table 2.1: Summary of existing works

2.5 Technological Review

2.5.1 Computer Vision

Computer vision-based algorithms serve as the cornerstone for subject detection in various HFR behaviors. Deep learning models like YOLOX, YOLOv5, Single Shot Multibox Detector (SSD), and OpenCV are extensively utilized for human detection and tracking in the literature (Kulkarni and Pantawane, 2022; Jeyatharan, Ekanayake, and Sandaruwan, 2021; Priyandoko et al., 2017). The robustness of HFR heavily relies on the effectiveness of computer vision-based subject tracking mechanisms.

2.5.2 Depth Sensing and Ranging

Depth sensing and ranging technologies play a crucial role in enhancing the perception capabilities of HFR. Light Detection and Ranging (LiDAR), stereo cameras, Time Of Flight (TOF) light sensors, and ultrasonic sensors are commonly used depth sensing technologies. LiDAR

systems emit laser pulses and measure the time it takes for the light to return, enabling precise 3D mapping of the environment (Algabri and M. T. Choi, 2020). Stereo cameras utilize two cameras to calculate depth by comparing disparities between corresponding points in the images (Kulkarni and Pantawane, 2022). TOF light sensors measure the time light travels to an object and back, providing accurate distance measurements (Yuan et al., 2021). Ultrasonic sensors emit high-frequency sound waves and calculate distance based on the time taken for the waves to reflect, suitable for short-range sensing applications (Tai et al., 2019). These depth-sensing technologies enable HFR to perceive and navigate their surroundings effectively, even in challenging environments with occlusions and varying lighting conditions.

2.5.3 Motion Sensing

Motion sensing technologies are essential for HFR to perceive and respond to changes in their environment and the movement of the target subject. IMU, magnetometers, Passive Infrared (PIR) detectors, and other motion-capturing devices are commonly used in such applications. IMUs consist of accelerometers and gyroscopes that measure linear and angular motion, providing information about the robot's orientation and movement (Bai et al., 2020). Magnetometers detect changes in magnetic fields, aidng in orientation and navigation tasks. PIR detectors sense infrared radiation emitted by objects, allowing the robot to detect the presence and movement of individuals within its vicinity (Yang et al., 2013). These motion-sensing technologies enable HFR to adapt their behavior dynamically based on the motion patterns of the target subject and changes in the surrounding environment.

2.5.4 Indoor Localization

Achieving effective occlusion handling requires the robot to accurately determine its location and that of the target subject relative to a local reference frame. Various methods exist for achieving indoor localization.

Dead Reckoning

Dead reckoning, an essential concept in navigation, entails determining the current location of a moving object by utilizing a previously known position and integrating estimates of speed, direction, and elapsed time (Kao, 1991). A similar path integration mechanism in biology illustrates

how animals update their position or heading estimates.IMU sensors play a pivotal role in dead reckoning by furnishing accurate directional data. Nonetheless, dead reckoning is vulnerable to cumulative errors, which can impact the precision of position estimates over time. Despite the availability of advanced navigational aids such as satellite systems, IMU-based dead reckoning remains widely utilized due to its capacity to provide precise directional information across various applications. Bai et al. (2020) presents a new technique for indoor pedestrian positioning using an IMU.

Proximity

Proximity-based indoor localization relies on proximity sensors to determine the distance between the robot and surrounding objects or landmarks. These sensors can include ultrasonic sensors, infrared sensors, or Radio Frequency Identification (RFID) tags. The robot can triangulate its position and navigate by measuring the proximity to known reference points within the environment. For example, ultrasonic sensors emit sound waves and measure the time taken for the waves to bounce back from nearby objects, allowing the robot to estimate distances. Similarly, infrared sensors detect the presence of objects based on their heat signatures, while RFID tags emit radio signals that the robot can detect to determine proximity. By utilizing these proximity sensors with appropriate algorithms, the robot can effectively localize itself indoors, even in occlusions (Zafari, Gkelias, and Leung, 2019).

UWB

UWB indoor localization is a high-precision positioning technology that transmits short-duration radio wave pulses across a broad spectrum. These pulses propagate through the indoor environment and are received by UWB-enabled devices, such as anchors and tags, allowing for precise distance measurements based on time-of-flight calculations (Zafari, Gkelias, and Leung, 2019). By deploying multiple UWB anchors at known locations throughout the environment and equipping the robot with a UWB tag, the system can determine the robot's position with centimeter-level accuracy through trilateration or multilateration techniques. UWB localization offers several advantages, including robustness against multipath interference, high accuracy even in dense and cluttered environments, and immunity to signal attenuation caused by obstacles or occlusions. Additionally, UWB technology provides low-latency measurements, making it more effective for robots to navigate and interact with their surroundings, even in complex

indoor environments with occlusions, ensuring reliable and precise localization for various applications (T. Feng et al., 2018).

Wifi - Bluetooth

Wi-Fi and Bluetooth-based indoor localization utilize wireless signals to determine the position of devices within indoor environments. Wi-Fi localization relies on signal strength measurements from nearby access points, comparing them to a database of known signal strength fingerprints for position estimation. Bluetooth-based localization uses Bluetooth Low Energy (BLE) beacons distributed throughout the space, detecting their unique identifiers to estimate device position through techniques like trilateration or fingerprinting. These technologies offer cost-effective and easy-to-deploy solutions for indoor positioning, although accuracy may vary based on factors like signal interference and infrastructure density (Zafari, Gkelias, and Leung, 2019).

Amongst the various indoor localization technologies, UWB stands out as one of the most promising and viable options. Unlike Wi-Fi and Bluetooth-based methods, UWB provides centimeter-level accuracy, making it particularly suitable for applications requiring precise positioning, such as asset tracking, indoor navigation, and robotics. Additionally, UWB systems offer robust performance in challenging environments with high multipath and non-line-of-sight conditions, making them suitable for deployment in complex indoor spaces like warehouses, manufacturing facilities, and retail stores. With its superior accuracy and resilience to interference, UWB technology presents a compelling solution for addressing the demanding requirements of indoor localization across various industries.

2.5.5 Communication

Establishing real-time, low-latency connectivity between the robot and the tracker is crucial. Several wireless communication technologies offer robust performance, improved range, and low latency, making them suitable.

BLE is a notable technology that enables energy-efficient communication over short distances. BLE is well-suited for establishing reliable connections between the robot and the tracker, facilitating seamless data exchange while conserving battery power.

Another option is Wi-Fi, which provides high-speed data transmission and broader coverage

compared to BLE. Wi-Fi connectivity allows for more extensive communication ranges, making it suitable for scenarios requiring longer-distance interactions between the robot and the tracker.

Additionally, Zigbee offers reliable wireless communication with low power consumption, making it suitable for applications where energy efficiency is paramount. Zigbee's mesh networking capabilities enable robust communication in complex environments, enhancing the reliability of data transmission between the robot and the tracker.

2.5.6 Micro-controllers

Arduino and Raspberry Pi are the most popular controller boards for similar robotic applications.

Raspberry Pi is a versatile single-board computer with robust processing power and a wide range of connectivity options. It is well-suited for tasks requiring complex computations, networking capabilities, and interfacing with various sensors and peripherals. With its support for programming languages like Python and C/C++, Raspberry Pi is a perfect choice for the central controller of the robot, facilitating tasks such as navigation, sensor fusion, and decision-making.

On the other hand, **Arduino** is a popular open-source hardware platform known for its simplicity and ease of use. It is ideal for controlling low-level hardware components and interfacing with sensors and actuators in real-time applications. Arduino boards are lightweight, energy-efficient, and cost-effective, making them suitable for implementing tracking devices worn by individuals. By leveraging Arduino's capabilities, it is feasible to design wearable trackers that communicate seamlessly with the central robot controller, enabling precise and responsive human following behavior.

2.5.7 Estimation Algorithms

Estimation algorithms play a crucial role in robotics and sensor fusion applications by enabling the estimation of the state of a system based on noisy sensor measurements. Among these algorithms, Kalman Filters and Exponential Moving Averages (EMA) are widely used. Kalman Filters are recursive algorithms that estimate the state of a dynamic system by combining predictions from a model with noisy sensor measurements, making them highly effective in linear, Gaussian scenarios. They offer a sophisticated approach by continuously updating predictions with new data. In contrast, EMA is a simpler and more straightforward method that gives more

weight to recent measurements, making it easier to implement but less adaptable to complex systems compared to Kalman Filters.

2.5.8 Robot Architectures

In robot architectures, subsumption and potential field architecture are prominent approaches when designing human following and obstacle scenarios. Subsumption Architecture, proposed by Rodney Brooks, is a behavior-based paradigm where multiple layers of behaviors are organized in a hierarchical structure. Each layer addresses a specific task or behavior, with higher layers capable of inhibiting lower layers' behaviors. This architecture emphasizes real-time responsiveness and robustness to environmental changes, making it suitable for dynamic and unpredictable environments often encountered by HFR. Jeyatharan, Ekanayake, and Sandaruwan (2021) has used this architecture to design a robot that traverses in a dynamic environment with obstacles and varying conditions while following a target person.

Potential Field Architecture relies on artificial potential fields to navigate robots through the environment. This approach generates attractive and repulsive forces based on the robot's goals and obstacles' locations, respectively (Yuan et al., 2021; Tsun, Lau, and Jo, 2018). The robot then moves along the gradient of the potential field, seeking to minimize its energy or reach a goal state. While this method offers simplicity and effectiveness in navigation tasks, it may need help with complex environments and local minima issues.

2.5.9 Simulation and Visualization Platforms

Simulation and visualization platforms are essential for developing and testing robotic systems. These platforms provide environments where algorithms can be implemented, tested, and evaluated before deployment on physical robots. One widely used platform is the Robot Operating System (ROS), which offers a comprehensive set of libraries and tools for developing robotic applications. ROS facilitates communication between different robotic system components and provides simulation capabilities through packages like Gazebo (Priyandoko et al., 2017). Python libraries such as OpenCV and TensorFlow are also commonly used for simulation and visualization tasks, offering flexible and efficient tools for image processing, machine learning, and deep learning. Additionally, Microsoft Robot Simulators provide realistic virtual environments for testing robot behavior and algorithms (Tsun, Lau, and Jo, 2018). Point cloud processing li-

braries like PCL (Point Cloud Library) are essential for handling 3D sensor data and performing object detection, tracking, and mapping tasks in simulated environments. In addition, Python UI and game development libraries like Turtle, Kivy, and Pygame can also be utilized to create simulations, virtual environments, and visualizations for robot simulations. These simulation and visualization platforms enable researchers and developers to iterate quickly and efficiently, accelerating the development process of HFR.

2.6 Evaluation

2.6.1 Experimental Setup and Test Scenarios

A rigorous experimental setup is crucial to thoroughly evaluate the performance of occlusion handling algorithms within HFR systems. Below is a breakdown of key considerations:

Environments

- **Varying Occlusion Complexity**: Environments with various occlusion levels, from simple partial occlusions to complex.
- **Maze-like Structure**: A maze with intersections is ideal for creating realistic and challenging occlusion scenarios. This can be modeled either physically or through software simulation.

Lighting Conditions

- **Diverse Lighting**: This evaluates the performance under varied lighting conditions, including bright daylight, dim indoor settings, and nighttime scenarios.
- **Vision-Based vs. Non-Vision-Based**: If the solution lacks vision-based algorithms, it is already prone to various lighting conditions. When using software simulations, this scenario is tedious to tackle.

2.6.2 Evaluation Metrics and Criteria

The following quantitative and qualitative metrics are used to comprehensively assess the performance of occlusion handling algorithms in HFR systems.

Quantitative Metrics

- **Tracking Accuracy**: Measure the system's ability to maintain accurate trajectories of targets even during periods of occlusion. This could be expressed as a percentage of correctly tracked targets or deviation from ground truth.
- **Localization Error**: Quantify the difference between the system's estimated target and actual positions. Lower localization error indicates better performance.
- **Occlusion Recovery Time**: Evaluate how quickly the system can re-establish accurate tracking after an occlusion event.
- **Computational Efficiency**: Assess the algorithm's computational cost and resource usage. This is crucial for real-time applications and deployments on resource-constrained hardware.

Qualitative Metrics

- **Robustness to Occlusion Types**: Analyze how well the algorithm handles different occlusion scenarios (e.g., complete vs. partial, static vs. dynamic occlusions).

2.7 Chapter Summary

This chapter introduces the fundamental problem of occlusion handling in human-following robotics applications. Categorizes HFR behaviors and analyzes various subject-tracking methods. Different types and causes of occlusions are explained, followed by a review of existing research on partial and full occlusion handling. The chapter emphasizes the critical role of computer vision, depth sensing, localization, and various estimation algorithms in developing robust HFR solutions. Lastly, a guideline for a comprehensive evaluation covering experimental setups, test scenarios, and relevant performance metrics is presented.

3 METHODOLOGY

3.1 Chapter Overview

This chapter presents the research methodology and project management approach adopted for the study on enhancing occlusion handling in human-following robots (HFR). The methodology encompasses various aspects, including the research philosophy, approach, choice, strategy, time horizon, techniques, and procedures. Additionally, we discuss the development methodology, requirement elicitation approach, design methodology, evaluation methodology, and resource requirements. The chapter also includes a schedule of deliverables and a summary of the project's methodology.

3.2 Research Methodology

An overview of the chosen scientific research methodology is given in the following table (See table 3.1), which is based on the Research "Onion" Model by Saunders.

Research Philosophy	The **pragmatism** approach was chosen over positivism, interpretivism, and realism for its emphasis on practicality and the need for iterative experimentation to optimize solutions. This approach acknowledges the importance of trial and error in selecting the most effective algorithms and methods.
Research Approach	The **deductive** strategy was selected among the available inductive and deductive research approaches because the research intends to evaluate the hypothetical statement and widely use a mixture of current theories.
Research Choice	This research aligns with **mixed methods** because it integrates quantitative and qualitative data collection approaches. This enables a comprehensive exploration of occlusion handling in HFR, considering technical performance metrics alongside user experiences and preferences.

Research Strategy	The most suitable research strategy for this study on enhancing occlusion handling in HFR is **experimental research**. This approach allows for systematic testing and evaluation of techniques and algorithms to improve human tracking accuracy and robustness in occlusion-prone scenarios.
Time Horizon	The **cross-sectional** time horizon, out of the two possible time horizons, longitudinal and cross-sectional, was chosen for this study since it allows for a snapshot analysis of the current state and effectiveness of occlusion handling techniques in various scenarios.
Techniques and Procedures	In this research, techniques and procedures involve leveraging data from wearable devices and sensors, developing occlusion detection algorithms, and conducting testing to enhance HFR capabilities.

Table 3.1: Research Methodology

3.3 Development Methodology

The **Prototyping Model** was selected from the available Software Development models because it makes it easy to create, assess, and iterate the prototype as needed. It allows for iterative refinement of occlusion handling algorithms and sensor integration in HFR systems, enabling gradual improvement based on feedback and testing.

3.4 Project Management Methodology

The project management methodology selected for this research endeavor is a hybrid approach combining Prince 2 and Agile techniques. Prince 2 offers a strategic-level framework, while Agile emphasizes flexibility and iterative improvements through continuous execution and reassessment. Integrating both methodologies provides a comprehensive management strategy tailored to the needs of the research project.

3.4.1 Schedule

Gantt Chart

Please refer the Figure B: .1 Project Timeline.

Deliverables and Dates

Please refer the table 3.2.

Deliverable	Date
Project Proposal Initial proposal of the research project	19 th February 2024
Literature Review Comprehensive analysis of the domain and related work	25 th March 2024
Software Requirement Specification Requirements to be catered in the final prototype	8 th April 2024
Initial Prototype Working prototype to demonstrate the concept	4 th May 2024
Final PSPD Final project specifications design and prototype	27 th May 2024
Test and Evaluation Report Performance evaluation against existing solutions	19 th June 2024
Thesis Document containing all the findings of the completed project	29 th July 2024
Research Paper A publication introducing the formulae for a **Occlusion-Aware Perception for Human Following Robots**	20 th August 2024

Table 3.2: Project Deliverables and Dates

3.5 Resource Requirement

The following are the determined hardware, software, data, and talent requirements for this project, along with their explanations, based on the goals, procedures, and features previously defined:

Hardware Resources

Since this is a software simulation based prototype the only hardware requirement is a working laptop itself; however when we implement this in real the following hardware resources are needed.

- **Arduino Micro-controller** - For the hardware prototyping and testing.
- **UWB Localization** - To pinpoint the location of the robot and the human subject in real environment.
- **Raspberry PI** - To be used as the robot controller device.
- **Radio Transceiver** - To communicate between the tracker and the robot.
- **Robot Base** - A wheel driven robot chassis is needed to integrate the proposed features.

Note: This **Robot Base** contains all the necessary sensor modules which requires to navigate the robot and handle obstacles avoidance. Mainly a LiDAR sensor or a stereo camera for depth estimation and optical ranging sensors for obstacle avoidance.

Software Resources

- **Arduino Integrated Development Environment (IDE)** - To program Arduino micro controllers.
- **Python** - To program Raspberry PI device and to carry-out the processing of point cloud and robot navigation system operations.
- **Javascript** - To create a web based 2D simulation environment to implement and test the prototype.
- **Unity3D** - To create 3D visualization of the simulation.

- **Visual Studio Code IDE** - To carry out the coding parts more efficiently.
- **Mendeley** - To manage references (Related papers).
- **Latex (Overleaf)** - To create all the documents and backup them.
- **GitHub** - To version control the codes and manage them.

Skill Requirement

For conducting this research effectively, the following skill requirements are essential:

- **Programming Skills**: Proficiency in Python, C/C++, and Arduino IDE for algorithm development, sensor fusion techniques, and microcontroller programming.
- **Embedded IoT and Electronics Proficiency**: Expertise in IoT technologies, sensor integration, wireless communication protocols, and microcontroller architectures (e.g., Arduino, Raspberry Pi). Additionally, knowledge of electronics design, circuit analysis, sensor interfacing, wireless communication, soldering, and troubleshooting electronics issues is essential.
- **Project Management**: Basic project management skills to plan, organize, and execute research activities within the specified timeline and budget, ensuring successful project completion.
- **Continuous Learning**: A dedication to continual learning and keeping aware of the most recent developments in robotics and technology in order to preserve the usefulness and relevance of research.

3.6 Risks and Mitigation

When undertaking any project, there will always be risks of different kinds and sizes. Risk factors in this particular setting may have theoretical, technical, or even environmental origins. Therefore, to effectively and efficiently avoid such possible hazards, it is imperative to identify them, their nature, and workable mitigation strategies (Refer the table 3.3).

Risk Item	**Severity**	**Frequency**	**Mitigation Plan**
Lack of domain knowledge	4	3	Continuously study relevant materials, seek support from domain and academic experts.
Hardware limitations	3	4	Conduct thorough research on hardware requirements, select reliable components, and conduct feasibility tests before implementation. Have backup hardware options available.
Software compatibility issues	3	3	Carefully evaluate software dependencies, ensure compatibility with chosen hardware, and maintain regular updates to address any compatibility issues that arise. Develop contingency plans for software failures.
Hardware malfunctions or damage	5	2	Implement proper handling and storage procedures for hardware components, perform regular maintenance checks, and invest in quality assurance measures. Maintain a contingency budget to replace damaged components promptly.

Table 3.3: Potential risks and mitigation

3.7 Chapter Summary

This chapter outlines the research methodology and project management approach for enhancing occlusion handling in human-following robots (HFR). Rooted in pragmatism, the methodology employs a deductive approach and mixed-methods research choice to address occlusion handling challenges comprehensively. The chosen experimental research strategy facilitates systematic testing and evaluation of occlusion handling techniques. Utilizing a cross-sectional time horizon, the study integrates techniques and procedures for algorithm development, sensor integration, and performance evaluation. Object-oriented analysis and design (OOAD) methodology guides system design, while the Prince 2 Agile hybrid technique ensures effective project management. Resource requirements, including hardware, software, and talent, and a schedule of critical deliverables are outlined. The methodology chapter provides a structured framework for conducting the research project and achieving its objectives.

4 SOFTWARE REQUIREMENTS SPECIFICATION

4.1 Chapter Overview

This Software Requirement Specification (SRS) chapter provides a comprehensive overview of the Human Following Robot HFR project, detailing various aspects such as stakeholder analysis, requirement elicitation methods, discussion of findings from interviews, literature review, and prototyping, along with context diagrams, use case diagrams, use case specifications, and requirements.

4.2 Rich Picture Diagram

The rich picture diagram visually represent the project environment, stakeholders, processes, and interactions of the proposed Occlusion Aware Perception system (See figure 4.1).

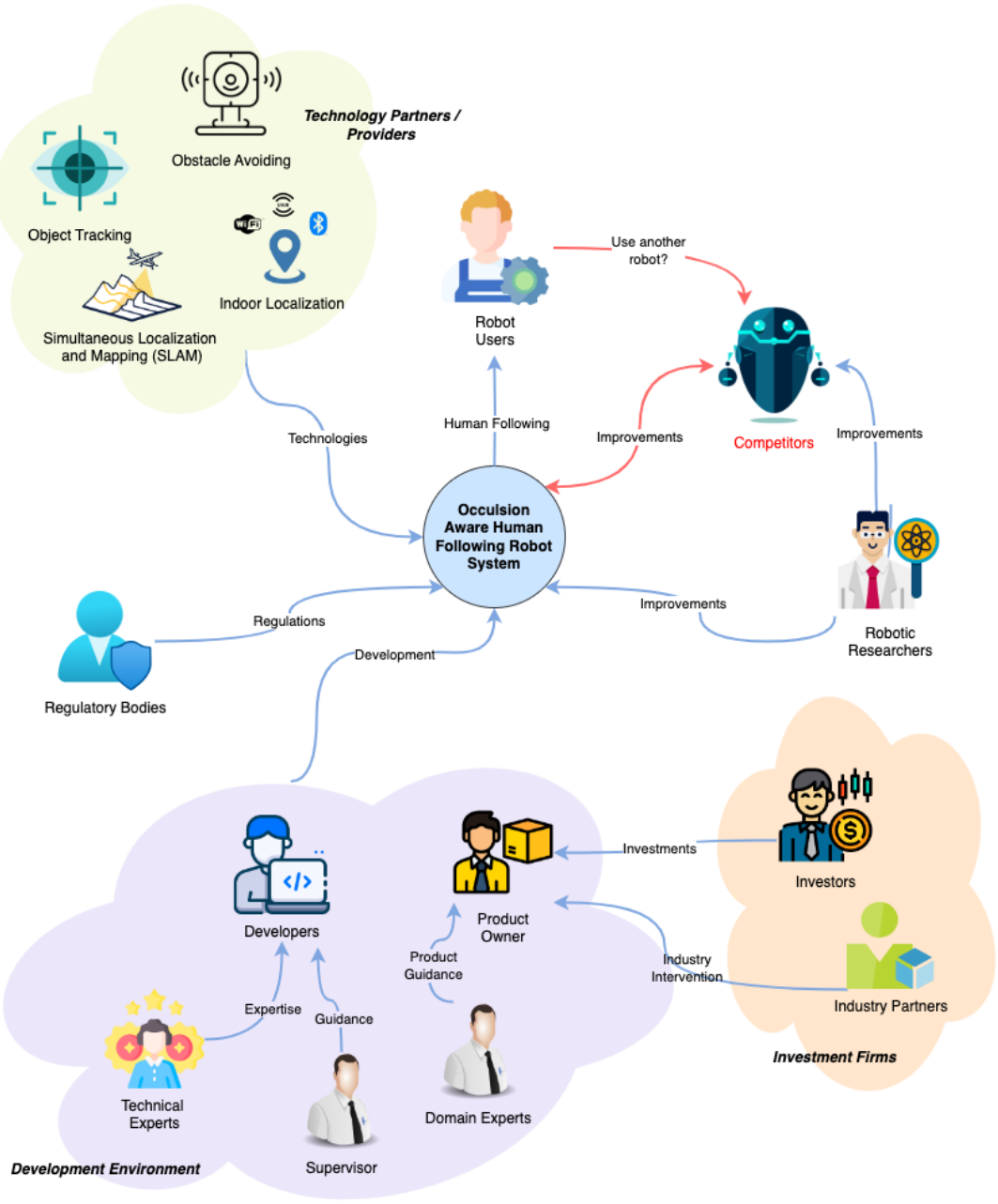

Figure 4.1: Rich Picture Diagram (Self composed)

4.3 Stakeholder Analysis

The stakeholder analysis includes a stakeholder onion model and descriptions of stakeholders involved in the project. Below is the stakeholder onion model for the project (See figure 4.2).

4.3.1 Stakeholder Onion Model

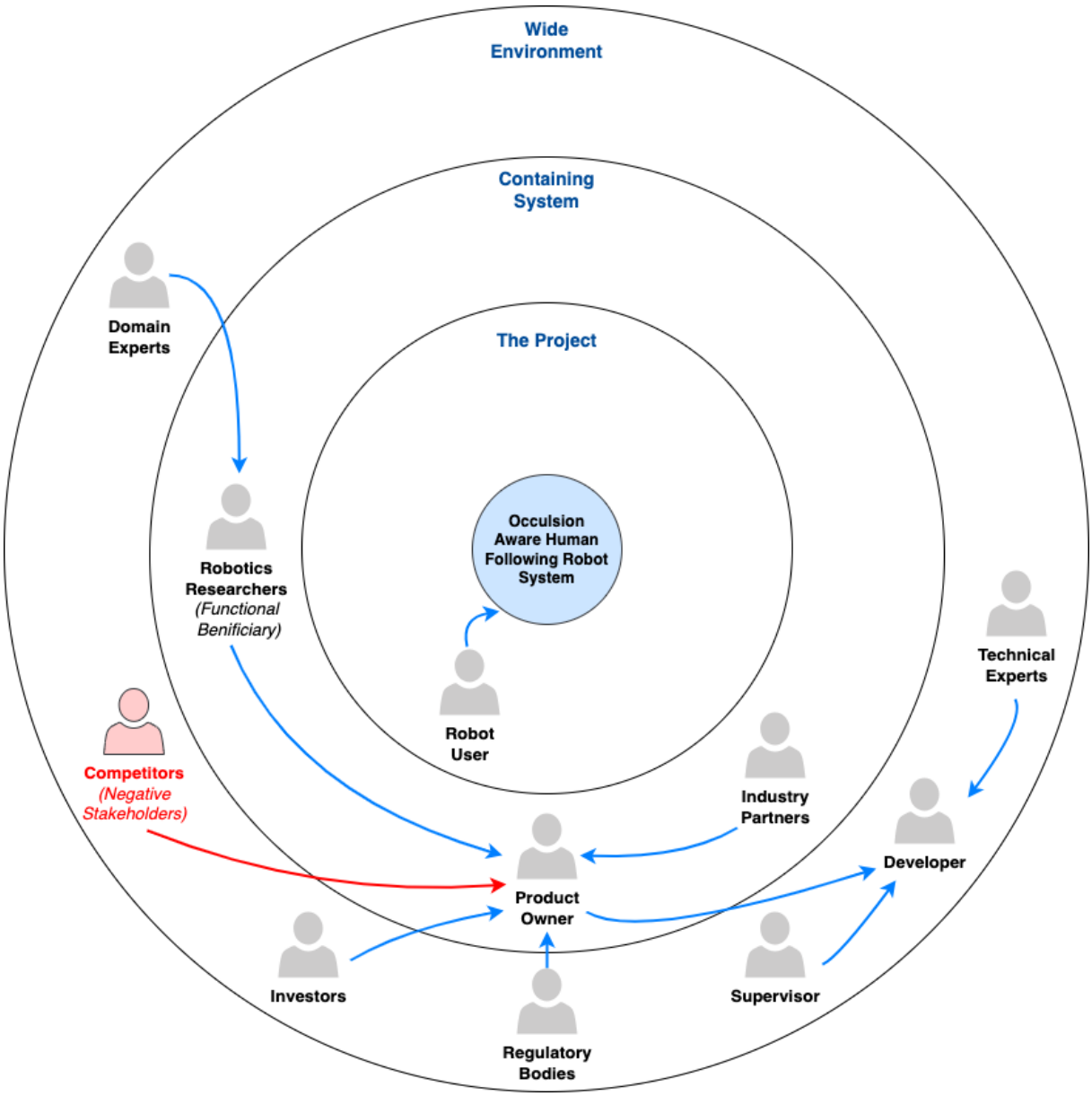

Figure 4.2: Stakeholder Onion Model (Self composed)

4.3.2 Stakeholder Description

Stakeholder	Role	Description
The Project Stakeholders		
Robot User	Normal operator	Uses the developed system for their benefit.
Containing System Stakeholders		
Industry Partners	Functional beneficiary	Use the findings to improve their products and shares feedback.
Product Owner	Operational beneficiary	Manage the whole project and resources. Takes advisors from experts and direct the project accordingly.
Robotics Researcher	Functional beneficiary	Enhances the perception system by findings new gaps and incorporating new outcomes from other researches.
Wider Environment Stakeholders		
Developer	Development Staff/-Maintenance	Develop the system and do bug fixes and maintains the code-base.
Domain Experts	Advisors/Experts	Quality controls the project and give supervision and their expertise to developers and product owner.
Technical Experts		
Supervisor		
Regulatory Bodies		
Investor	Financial beneficiary	Funds the project and gain profits from creating new products.
Competitor	Negative Stakeholder	Develop similar or improved versions of the project.

Table 4.1: Stake holder analysis

4.4 Requirement Elicitation Methods

To shape the requirements for this project, a combination of methods has been employed, each serving a specific purpose in gathering essential information and insights; these methods are justified below (See table 4.2).

Method 1: Interviews
Interviews with domain and academic experts were conducted initially. These experts possess invaluable insights and expertise in relevant fields, providing validation for the initial research idea. Their perspectives help refine the project scope and identify potential technologies and methodologies to employed. Engaging with these experts makes it easier to filter out irrelevant aspects and focus on critical areas of interest.
Method 2: Literature Review
A comprehensive literature review was undertaken to clarify the research gap further and understand the current state-of-the-art in the field. This process validates the research idea's relevance and uncovers additional gaps in existing systems. Analyzing prior research and developments aids in identifying areas where improvements or innovations can be made, guiding the project's direction.
Method 3: Prototyping
While the project primarily targets simulation-based development, initial hardware prototyping was undertaken to explore potential solutions for improving subject tracking in the HFR system. This prototyping phase allowed for hands-on experimentation and testing of various technologies and approaches. By designing and testing prototypes, it became possible to identify the most suitable technologies and methodologies for implementation. Additionally, it provided valuable insights into areas for improvement and potential challenges to be addressed.

Table 4.2: Requirement Elicitation Methodologies

4.5 Discussion of Findings

4.5.1 Findings from Interviews

Interviews were conducted with academic subject experts to gather insights into the project's scope and potential focus areas. Three academic experts were selected for interviews, comprising two specialists in robotics and embedded systems and one expert in unmanned aerial vehicles and virtual environments. Thematic analysis was performed to extract key themes and insights from the interviews.

Codes	**Theme**	**Analysis**
Virtual Environments and Simulations, Technological Recommendations, Project scope	Research gap and technological recommendations	All participants unanimously recommended Python as the primary programming language for the project, citing its versatility and extensive libraries. They emphasized the importance of developing initial simulations and working algorithms using a 2D gaming library due to its ease of use and rapid prototyping capabilities. Considering the time constraints, they advised against pursuing a hardware implementation and stressed that a simulation-based approach would suffice for achieving the project goals. Furthermore, they suggested utilizing Unity 3D for visualization purposes, noting its user-friendly interface and seamless integration with Python scripts, facilitating the development process without significant complexities.

Opportunities for Innovation	Potential areas for innovation or improvement	During the discussions, participants identified several potential areas for innovation and improvement within the project scope. One suggestion was to explore techniques commonly used in leader-following applications and adapt them to Human Following Robot HFR systems. Additionally, participants highlighted the potential for innovation in indoor localization and map building within HFR systems.
Challenges and Limitations	Identifying project challenges and limitations	Participants discussed various challenges and limitations that may arise during the project implementation. These include technical challenges related to sensor integration, algorithm development, and system optimization. Additionally, they highlighted potential limitations regarding resource constraints, such as time, budget, and availability of expertise.

Table 4.3: Findings from interviews

4.5.2 Findings from Literature Review

Finding	**Citation**
Incorporating Inertial Measurement Unit (IMU) sensors into indoor localization systems presents drawbacks due to error accumulation, leading to potential accuracy issues. However, integrating IMU and Ultra-wideband (UWB) technologies can enhance localization accuracy. Nevertheless, this approach necessitates complex mathematical algorithms for estimation, demanding significant computational power.	Bai et al., 2020; D. Feng et al., 2020

The authors introduced a method for tracking humans using UWB technology. The approach entails detecting and computing the position and orientation of the target person. However, this method has certain drawbacks, such as requiring approximately five UWB sensors per implementation, which can incur high costs, particularly when multiple robots need to be equipped.	T. Feng et al., 2018
Using provided map details or building a map while traveling is a good way of handling occlusions. These map details can be used to identify potential occlusion points and take mitigation actions.	Yuan et al., 2021; Kumar Ashe and Krishna, 2021
Utilizing a custom-created tracker placed on the human subject offers higher accuracy and lower susceptiblity to occlusions than computer vision-based human tracking methods. Trackers vary in type, from simple color-coded stickers to complex electronic trackers like UWB trackers.	Algabri and M. T. Choi, 2020; Jeyatharan, Ekanayake, and Sandaruwan, 2021

Table 4.4: Findings from literature review

4.5.3 Findings from Prototyping

Criteria	**Findings**
To validate the number of anchor points needed in the indoor localization system and where to place them.	Using too many anchor points is optional to track an object. A minimum of 3 anchor points is needed to track an object in 2D space. We can further reduce and increase the accuracy depending on where we place the anchors.

To evaluate the performance of a IMU based step tracker for localization.	A basic step counter was constructed using an Arduino microcontroller, an accelerometer, and a gyro sensor to track the movement path of the individual carrying the device. However, it was determined that this approach did not meet the necessary accuracy standards for the project.

Table 4.5: Findings from Prototyping

4.6 Summary of Findings

ID	Finding	Literature Review	Interviews	Prototyping
1	Technological Recommendations: Python is recommended as the primary programming language due to its versatility and extensive libraries.		X	
2	Utilize a 2D gaming library for initial simulations and working algorithms.		X	
3	Advised against pursuing hardware implementation, favoring a simulation-based approach.		X	
4	Suggested using Unity 3D for visualization due to its user-friendly interface and integration with Python scripts.		X	
5	Explore techniques from leader-following applications for Human Following Robot (HFR) systems.	X		
6	Investigate innovations in indoor localization and map building for HFR systems.	X		

7	Identified technical challenges related to sensor integration, algorithm development, and system optimization.		X	
8	Highlighted potential limitations regarding resource constraints such as time, budget, and expertise.		X	
9	Incorporating IMU sensors combined with UWB technology can enhance localization accuracy, but complex mathematical algorithms are required.	X		
10	UWB-based human tracking method introduced, but drawbacks include high sensor requirement and cost.	X		
11	Handling occlusions using provided map details suggested as an effective approach.	X		
12	Custom trackers placed on human subjects offer higher accuracy and lower susceptibility to occlusions compared to computer vision-based methods.	X		
13	Found that a minimum of 3 anchor points is needed for object tracking in 2D space.			X
14	Basic step counter using IMU sensors did not meet the necessary accuracy standards for the project.			X

Table 4.6: Summary of findings

4.7 Context Diagram

This context diagram provides a high-level overview of the HFR system and its interactions with users (See figure 4.3.

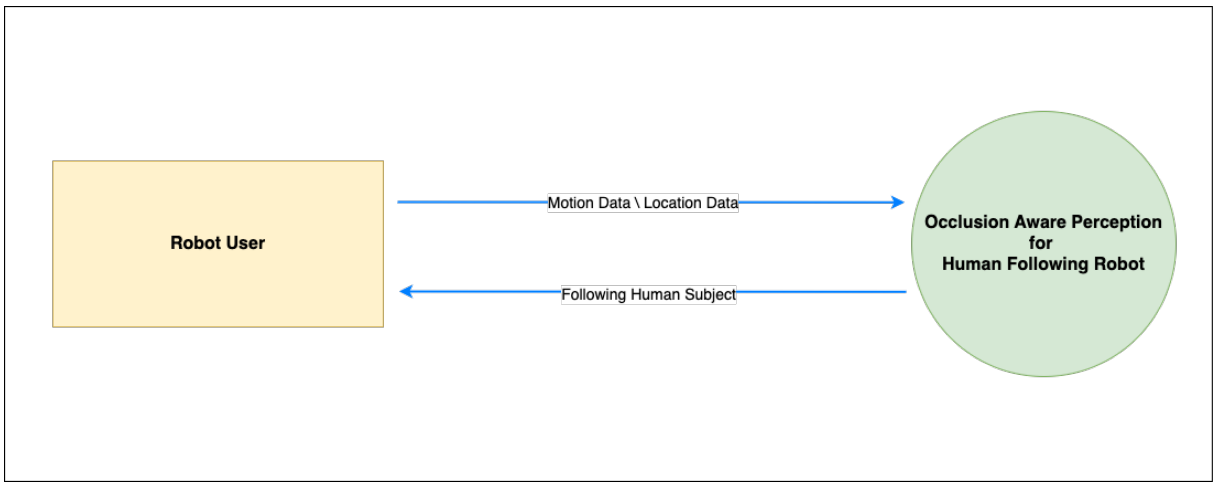

Figure 4.3: Context Diagram (Self composed)

4.8 Use Case Diagrams

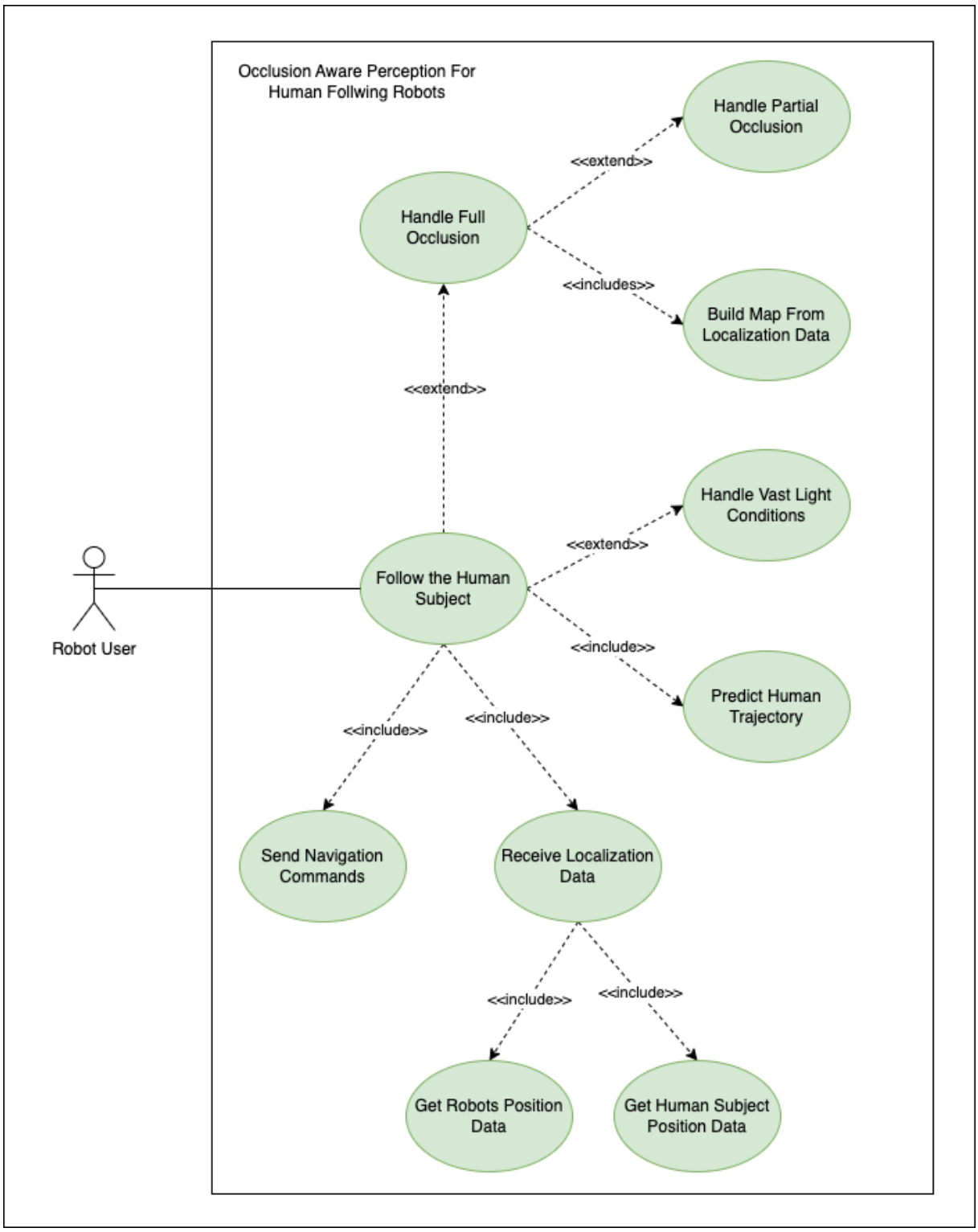

Figure 4.4: Use Case Diagram (Self composed)

4.9 Use Case Specification

Use Case Name	Follow Human Subject
Description	The robot follows the human subject as it moves; meanwhile, it internally maps where the human subject is located relative to the robot. This map information is used to regain track whenever a full occlusion occurs.
Actors	Human Subject
Preconditions	The robot is powered on and initialzed. The human subject is wearing the tracker. The robot has received location data from the localization module.
Post-conditions	The robot is following the human subject while maintaining a safe distance. The robot is continuously updating its path based on the human subject's movements.
Included Use Cases	Send Navigation Commands, Receive Localization Data, Predict Human Trajectory
Extended Use Cases	Handle Full Occlusion, Handle Vast Light Conditions
Basic Flow	1. The human subject starts walking. 2. The robot receives location data from the localization module. 3. Robot build a map internally of where the human subject is travels. 4. The robot sends commands to its motor controllers to follow the predicted trajectory of the human subject.
Alternative Flows	If the robot loses sight of the human subject completely (full occlusion), the robot uses the map details to gain the track of the human subject again (Handling Full Occlusion).
Exceptions	A sensor malfunction disrupts the perception system. The robot raises an error flag and might require manual intervention. An obstacle appears in the robot's path. The robot's navigation system must re-plan a path to avoid collision.

Table 4.7: Use Case Specification: Follow Human Subject

4.10 Requirements

Identified requirements for the project are categorized using the MosCoW convention. Functional and Non-Functional requirements are required below.

Must Have (MH)	Requirements which are must-have in the Minimum Viable Product (MVP).
Should Have (SH)	Requirements which are important but not critical for the project's immediate success.
Could Have (CH)	Additional features or that would be nice to have if time permits.
Won't Have (WH)	Requirements that are explicitly excluded from the project scope.

Table 4.8: Summary of "MoSCoW" prioritization levels

4.10.1 Functional Requirements

FR ID	Description	Priority
FR01	Implement essential sensors for human subject tracking in the simulated wearable device.	MH
FR02	Identify and implement an optimal indoor localization method for tracking human subjects and robot movements.	MH
FR03	Implement an algorithm to integrate data from localization and wearable trackers for constructing a map for robot guidance.	MH
FR04	Implement real-time data visualization features to monitor the tracking and mapping process during simulation.	SH
FR05	Implement path planing and optimal path selection algorithm to resolve occlusion when the environment is known.	SH
FR06	Implement frontier based environment mapping and resolve occlusion when the environment is unknown.	SH

FR07	Integration of a vision-based tracking system as a primary tracking mechanism.	WH

Table 4.9: Functional requirements

4.10.2 Non-Functional Requirements

NFR ID	Requirement	Description	Priority
NFR01	Accuracy	The system must achieve a high level of accuracy in handling occlusions and tracking human subjects within the simulated environment.	MH
NFR02	Performance	The system should not take much time to regain the track of the lost human subject due to fully occlusions.	MH
NFR03	Usability	The system should be easy to use and operate at any give scenario and should make clear and precise warning and instructions for user to operate.	SH
NFR04	Adaptability	The system should be easily integrated with existing primary tracking methods such as Computer vison based tracking algorithms.	CH
NFR05	Scalability	The robotic system should be designed with scalability in mind to accommodate potential future expansions or modifications; adding multiple robots to the environment is a potential scenario.	WH

Table 4.10: Non-Functional requirements

4.11 Chapter Summary

The Software Requirements Specification (SRS) chapter provides a holistic view of the stakeholder analysis, requirement elicitation methods, and findings. Through interviews, literature

review, and prototyping, key insights were gathered, including technological recommendations, opportunities for innovation, and challenges and limitations. These findings informed the context and use case diagrams, illustrating the system's interactions and functionality. A detailed use case specification also outlined the Follow Human Subject use case, detailing its actors, preconditions, post-conditions, and flows. The chapter is a foundation for understanding the project's objectives, scope, and requirements.

5 SOCIAL, LEGAL, ETHICAL & PROFESSIONAL ISSUES

5.1 Chapter Overview

This chapter examines the social, legal, ethical, and professional issues related to the occlusion-aware perception system. It outlines specific concerns in each category and presents mitigation strategies to address them.

5.2 Breakdown of Social, Legal, Ethical, and Professional Issues

Social Issues	**Legal Issues**
• No personal user information is collected. The system operates without storing or processing identifiable data. • The project remains neutral, with no engagement in political or religious perspectives.	• All tools and frameworks used are open-source, ensuring compliance with licensing requirements. • User inputs are not used for unrelated analysis, protecting user privacy. • All the collected data are handled compelling to the GDPR regulations.
Ethical Issues	**Professional Issues**
• Safety protocols and thorough testing ensure safe operation. • All information presented is original and properly cited, avoiding plagiarism.	• Security techniques like code minification and obfuscation protect against unauthorized access. • The project follows industry-standard development methods, including agile practices and rigorous testing. • The development environment is secured with the latest security patches, minimizing vulnerabilities.

Table 5.1: Breakdown of Social, Legal, Ethical and Professional Issues

5.3 Chapter Summary

This chapter identifies and addresses critical social, legal, ethical, and professional issues related to the occlusion-aware perception system, presenting strategies to mitigate these concerns and ensure responsible development.

6 DESIGN

6.1 Chapter Overview

In this chapter, the proposed system is design aspects are explored. The chapter commences with an outline of the design goals; subsequently, the system architecture is discussed, detailing the tiers and modules comprising the presentation, logic, and data tiers. The discussion includes a breakdown of components within each tier and their respective functionalities. Furthermore, the choice of design paradigm is explored. Finally, design diagrams are presented, including a component diagram illustrating the main system components and a class diagram depicting class associations and interactions.

6.2 Design Goals

Design Goal	**Description**
Performance	This project is handling a critical performance issue in HFR systems, which fails to track the subject in cases of occlusions. Once a full occlusion occurs, this backup system should intervene and regain track of the subject quickly. To handle this, developing a well-optimized path planing and recovery mechanism is necessary.
Accuracy	The system should be capable of minimizing errors and providing reliable position estimates to ensure the human following robot (HFR) can accurately regain track of the subject after occlusions.
Adaptability	Since this solution only applies when handling full occlusions, adding a primary subject tracking method (e.g., computer vision-based subject tracking), obstacle avoidance, or topological mapping is trivial. So, the solution must be developed to make adding more functionalities easier.
Scalability	Since the proposed solution contains a infrastructure setup as well, the solution should support multiple robots to be in the environment and they should be able follow different subjects at the same time without any conflicts. So the system should be designed in a way that is able to add more robots and subjects without utilizing more resources.

Table 6.1: Design Goals of the system

6.3 System Architecture Design

6.3.1 Architecture Diagram

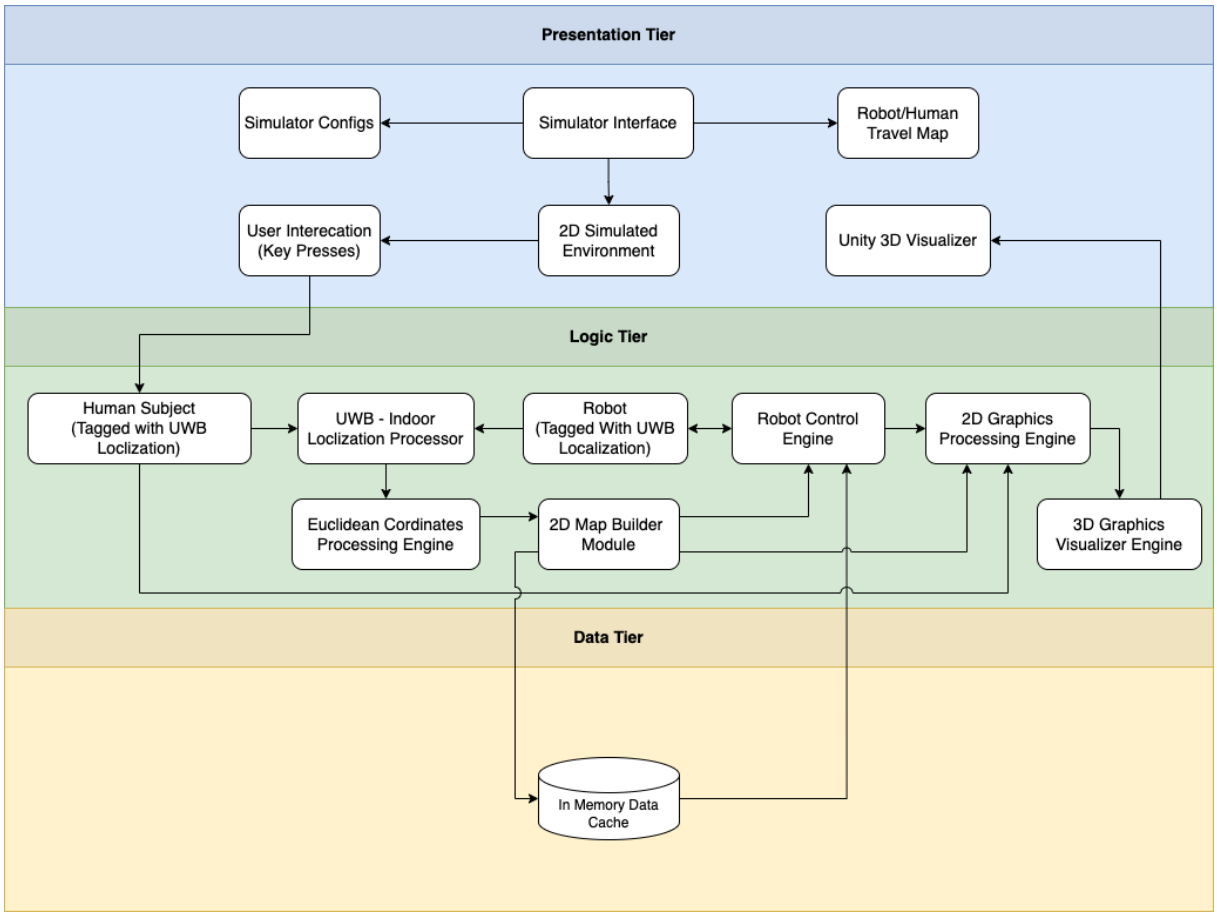

Figure 6.1: High Level Architecture (Self composed)

6.3.2 Discussion of Tiers

Presentation Tier

- **Simulator Interface** - This is the module where we can see what is happening inside the system. This is where we get user interactions into the system.
- **2D Simulated Environment** - This is where robot and human subject is placed. It contains obstacles and a boundary where robots and humans can travel. This is displayed in

HTML canvas using 2D graphics.

- **Travel Map** - This is also a 2D HTML canvas where we can see the map of the human subject and robot and their live traveling updates. This module mainly works with the indoor localization module's data.
- **Simulator Configs** - In this section, we can modify the simulator configurations, such as show or hide debug information.
- **Unity 3D Visualizer** - This module is to visualize the simulation in 3D environment.

Logic Tier

- **Human Subject** - This is the entity that can control user inputs taken from the UI (Presentation Tier). This is one of the movable objects in the system and it is tagged with a UWB tag to track the movements inside the environment.
- **Robot** - This is another movable object in the system, tagged with a UWB tag. Both human subjects and robots have an interface to get travel commands.
- **UWB-Indoor Localization Processor** - This module is responsible for handling indoor localization. There are UWB anchors placed in known places inside the environment, and UWB tags use the anchor points as a reference to estimate their location inside the environment. This process is handled by this module.
- **Euclidean Coordinates Processing Engine** - This simulation works on a Cartesian plane; all the distances and positions are estimated using the coordinates geometry to process these calculations this processing engine is used.
- **2D Map Builder Module** - This is a crucial component of the system. It is responsible for keeping track of the human subject and the robot where they have traveled in the environment. This map is used to make the robot's travel decisions in cases of full occlusions and missing human subjects. This map directly works with the UWB module data.
- **Robot Control Engine** - The robot control engine governs the behavior and movement of the robot within the environment. It receives travel commands, processes sensor data, and executes navigation algorithms to ensure safe and efficient movement while avoiding

obstacles and achieving specified objectives.

- **2D Graphics Processing Engine** - Responsible for rendering graphics and visual elements within the 2D simulated environment, this engine utilizes HTML canvas and graphics libraries to display objects, obstacles, and environmental features. It ensures smooth and responsive visualization of the simulation environment.
- **3D Graphics Visualizer Engine** - Operating within the Unity3D framework, the 3D graphics visualizer engine generates immersive three-dimensional visualizations of the simulated environment. It renders objects, terrain, and environmental elements with realistic textures and lighting effects, enhancing the user experience and providing additional insights into the simulation scenario.

Data Tier

- **In Memory Data Cache** - The map builder module uses a in-memory cache to keep the travel map for the current session. For the initial phase there is no hard requirement for a persistent data store.

6.4 Detailed Design

6.4.1 Choice of Design Paradigm

For this project, the Object-Oriented Analysis and Design (**OOAD**) paradigm has been chosen due to its suitability for modeling real-world scenarios and faciltating complex behavioral modeling and inheritance. OOAD allows for the representation of entities as objects, promoting encapsulation, abstraction, and code reuse. With TypeScript as the primary programming language supporting OOP features, OOAD aligns well with the project's objectives, ensuring a structured, maintainable, and extensible design.

6.4.2 Algorithm Design

Localization Assisted Occlusion Recovery Algorithm

This algorithm (Refere to algorithm 1) works even in an under-explored environment. It utilizes the already explored environment and try to recover occlusion in under-explored environment

using the localization data.

Algorithm 1 Handle Occlusion in Unknown Environment

Input: targetPosition, discoveredMap, robotTravelMap, robot

```
endPosition ← convertPositionToInt(targetPosition)
frontierDistanceThreshold ← 3
path ← []
path ← AstarPathPlanner.findPath(convertPositionToInt(robotTravelMap.getCurrentPositio
if path ≠ [] then
    moveAlongPath(path)
    return
end if
updateDiscoveredMap()
frontierPriorityQueue ← new FrontiersPriorityQueue()
frontiers ← identifyFrontiers()
for all frontier ∈ frontiers do
    frontierPriorityQueue.addFrontier(frontier, endPosition)
end for
while ¬frontierPriorityQueue.isEmpty() do
    frontier ← frontierPriorityQueue.popFrontier()
    if canMoveToPosition(frontier) then
        moveToFrontierInDiscoveredMap(frontier)
        updateDiscoveredMap()
        frontiers ← identifyFrontiers()
        for all newFrontier ∈ frontiers do
            frontierPriorityQueue.addFrontier(newFrontier, endPosition)
        end for
    end if
    if isPositionsNear(frontier, endPosition, frontierDistanceThreshold) then
        return
    end if
end while
path ← AstarPathPlanner.findPath(convertPositionToInt(robotTravelMap.getCurrentPositio
if path ≠ [] then
    moveAlongPath(path)
else
    Output: "No path found"
end if
```

6.5 Design Diagrams

6.5.1 Component Diagram

Main components and the sub components of the proposed system is illustrated in the following component diagram (See figure 6.2).

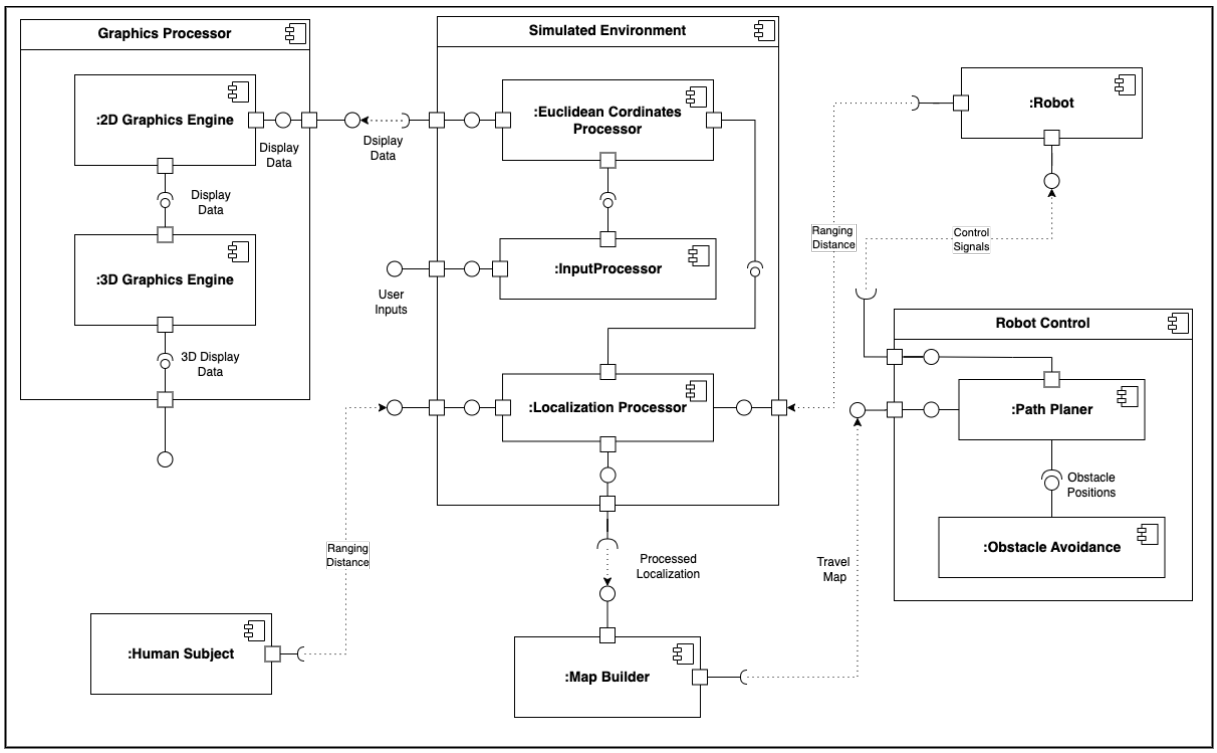

Figure 6.2: Component Diagram (Self composed)

6.5.2 Class Diagram

Classes and their associations and interactions of the implemented system is depicted in the following class diagram (See figure 6.2)

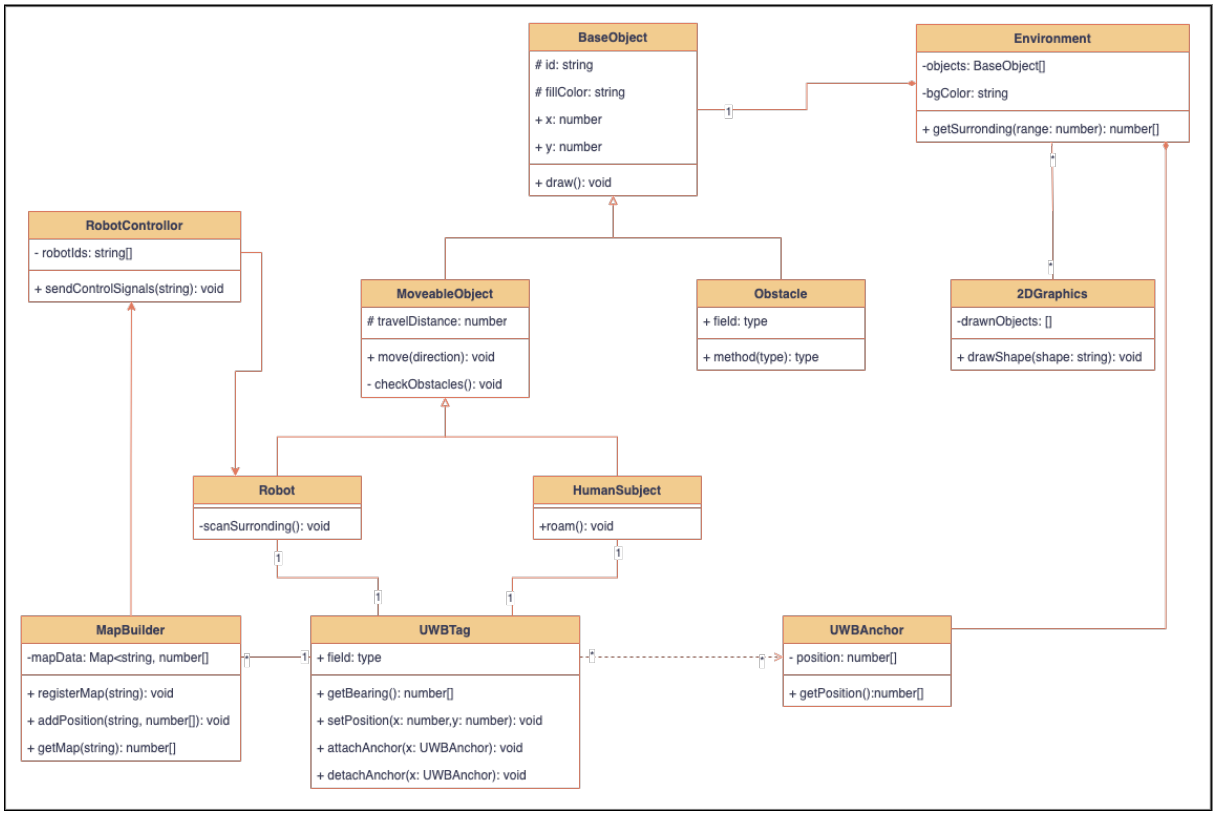

Figure 6.3: Class Diagram (Self composed)

6.6 Chapter Summary

The design goals underscore the critical aspects of the system, focusing on performance, usability, extendibility, and scalability to address the challenges of occlusion-aware perception in human-following robots. The system architecture delineates the tiers and modules, highlighting their roles and interactions in facilitating robust functionality. The choice of OOAD as the design paradigm ensures a structured and maintainable design. The design diagrams offer visual representations of the system's components and class structures, providing insights into system organization and relationships. Overall, the comprehensive design overview lays the foundation for the project's subsequent implementation and testing phases.

7 IMPLEMENTATION

7.1 Chapter Overview

This chapter focuses on selecting technologies and frameworks to implement the occlusion-aware perception system for human-following robots. It begins by presenting the technology stack diagram, outlining the key technologies employed in the development process. Subsequently, the selection of programming languages, development frameworks, IDEs, and other development tools is discussed in detail, along with justifications for their choices. The chapter concludes with a summary of the technology selection, highlghting the components and tools chosen for the implementation.

7.2 Technology Selection

7.2.1 Technology Stack

Technologies utilized to implement the prototype and the functionalities is shown in the below diagram (See figure 7.1).

Figure 7.1: Technology Stack (Self composed)

7.2.2 Programming Language Selection

Programming Language	**Justification for Selection**
Typescript	TypeScript (TS) has been chosen for its strong support of object-oriented programming (OOP) features, which aligns well with the OOAD paradigm selected for this project. Despite the languages features, its been selected to use the JavaScript asynchronous capabilities.
C#	C# is chosen particularly in the context of Unity3D development for the 3D visualization of the simulations.

Table 7.1: Programming Language Selection

7.2.3 Development Framework Selection

Framework	Justification for Selection
Konva	Konva is capable of creating interactive and visually appealing graphics in HTML5 canvas-based applications. It provides a robust set of tools and functionalities for drawing, animating, and manipulating graphical elements in a 2D space. Additionally, Konva's compatibility with TypeScript make it an ideal choice for implementing the graphical components of the system.
Express JS	Express.js it to building server-side applications and Restful APIs. It's main purpose to build a rest interface to plug the application with Unity3D visualization scripts.

Table 7.2: Development Framework Selection

7.2.4 IDE Selection

IDE	Justification for Selection
VS-Code	Visual Studio Code (VS-Code) is simple yet powerful editor with many builtin capabilities. Ease of debugging, syntax highlighting, intelligent code completion like features led VS-Code to be chosen as the primary IDE.
Typescriptlang Online Playground	This online Typescript playground is used to test and debug TS functions without much hassle.

Table 7.3: IDE Selection

7.2.5 Other Development Tools

Development Tool	Justification for Selection
Node.js	Node.js enables server-side JavaScript execution and providing a runtime environment for building the project. Node.js serves as a crucial component for running build scripts, managing dependencies.

Unity3D	Unity3D is utilized for visualizing the simulated environment in a 3D space, offering advanced rendering capabilities, physics simulations.
webpack	Webpack is a build tool and module bundler optimizing and packaging frontend assets, including JavaScript files, CSS stylesheets, and image assets.
Git & GitHub	Version control the project and secure cloud storage for the source files.

Table 7.4: Other Development Tools Selection

7.2.6 Summary of Technology Selection

Component	**Tools**
Programming Languages	Typescript, C#
Development Frameworks	Express.JS, Konva
IDEs	VS-Code, Typescriptlang Online Playground
Version Control	Git & GitHub
Other	Node.js, Unity3D, webpack

Table 7.5: Other Development Tools Selection

7.3 Implementation of Core Functionalities

This project utilizes Node.js with webpack to bundle all components, enabling browser execution to meet visualization requirements. The browser-based visualizations are managed by a wrapper class that encapsulates the functionalities of the Konva library. Konva employs HTML canvas for rendering graphic visualizations.

7.3.1 Simulated Environment

The first step is to implement a virtual environment and a simulation platform. In top of that virtual environment all the other implementation takes place. This enviroment is 100x100 2D box. All the coordinates are processed in this 100x100 scale; only the visualising output is scaled up. The environment initialization is done through a configuration file; this configurations are used to place the obstacle and determine the drawing scale (See figure 7.2).

```
const configs = {
    scale: globalConfigsProvider.getConfig("mapScale"),
    env: {
        width: 100,
        height: 100
    },
    // cords: [x, y], dims: [width, height]
    obstacles: [
        { cords: [0, 0], dims: [25, 10] },
        { cords: [10, 20], dims: [15, 15] },      You, 2 months ago • some refactoring
        { cords: [10, 45], dims: [20, 15] },
        { cords: [0, 70], dims: [30, 20] },
        { cords: [40, 85], dims: [25, 15] },
        { cords: [40, 70], dims: [10, 15] },
        { cords: [75, 85], dims: [25, 15] },
        { cords: [60, 40], dims: [30, 35] },
        { cords: [40, 40], dims: [20, 20] },
        { cords: [70, 10], dims: [20, 20] },
        { cords: [35, 10], dims: [25, 20] },
    ]
}


function createEnvironment(configs: any, canvas: Canvas) {
    const env = new Environment(configs.env.width, configs.env.height, canvas);
    configs.obstacles.forEach((config: any) => {
        const obstacle = new Obstacle(config.dims[0], config.dims[1], config.cords[0], config.cords[1]);
        env.addObject(obstacle);
    });
    return env;
}

function initGraphicCanvases() {
    UWBMapCanvasImpl.setScale(globalConfigsProvider.getConfig("localizationMapScale"));
    UWBMapCanvasImpl.drawBackdrop({ width: 100, height: 100, fillColor: "#FFFFF1", strokeColor: 'black' });

    ExploredMapCanvasImpl.setScale(globalConfigsProvider.getConfig("lidarMapScale"));
    ExploredMapCanvasImpl.drawBackdrop({ width: 100, height: 100, fillColor: "#FFFFFF", strokeColor: 'black' });

    CanvasImpl.setScale(globalConfigsProvider.getConfig("mapScale"));
}

export function createSimulator(canvas: Canvas) {

    initGraphicCanvases();

    const env = createEnvironment(configs, canvas);

    // Add robot object to environment
    const robot = new Robot(5, 95, env);
    robot.setID("robot-A");
    env.addObject(robot);
```

Figure 7.2: Simulated Environment Setup

Code Explanation: This code sets up a simulated environment for a robot and a human, complete with obstacles and Ultra-Wideband (UWB) positioning. It initializes a graphical canvas and configures the environment's size and obstacles. The environment includes a robot and a human, each equipped with a UWB tag for positioning. Three UWB anchors are placed in the environment to assist with accurate positioning.

The code also creates map builders to track the movement of both the robot and the human. Every 80 milliseconds, the map builders record their positions based on UWB readings. A robot controller is initialized to manage the robot's movements within this environment.

Full code for the the simulator setup is in the Appendix D: .1.1.

7.3.2 Map Building

Timer events periodically obtain location data (X,Y Coordinates) from the UWB tags on both the robot and the human subject, using these coordinates to map their travel paths. To simulate real-world conditions, an error margin is added to the UWB localization, depicting sensor inaccuracies through slight imperfections in the map drawings. To smooth out the outliers and keep the map in correct state a Exponential Moving Average (EMA) is introduced to map module (Refer the code 7.3).

```
You, 6 days ago | 1 author (You)
export interface MapEntry {
    color: DrawingColor;
    positions: Position[];
}

You, 6 days ago | 1 author (You)
class MapBuilder {
    private map: Map<string, MapEntry> = new Map<string, MapEntry>();
    private canvas: Canvas;
    private emaAlpha: number;
    private emaPositions: Map<string, Position>;

    constructor(emaAlpha: number) {
        this.canvas = UWBMapCanvasImpl;
        this.emaAlpha = emaAlpha;
        this.emaPositions = new Map<string, Position>();
    }

    registerMap(mapName: string, color: DrawingColor, startingPosition: Position) {
        this.map.set(mapName, { color, positions: [] });
        this.emaPositions.set(mapName, startingPosition);
        this.drawMapLegend();
    }

    addPosition(mapName: string, position: Position) {
        const mapEntry = this.map.get(mapName);
        if (mapEntry) {
            // Update EMA
            const lastEmaPosition = this.emaPositions.get(mapName) || { x: 0, y: 0 };
            const emaPosition = {
                x: this.emaAlpha * position.x + (1 - this.emaAlpha) * lastEmaPosition.x,
                y: this.emaAlpha * position.y + (1 - this.emaAlpha) * lastEmaPosition.y
            };
            this.emaPositions.set(mapName, emaPosition);

            // Add smoothed position
            mapEntry.positions.push(emaPosition);      You, 6 days ago • stable version
            this.drawMapLinePolygon(mapName);
        }
    }

    getMap(mapName: string) {
        return this.map.get(mapName);
    }

    drawMapLegend() {…
    }

    drawMapLinePolygon(mapName: string) {…
    }
}
```

Figure 7.3: Map Builder

Code Explanation: This code defines a ‘MapBuilder‘ class that tracks and smooths positions

using an EMA for multiple maps identified by name. This class manages a collection of maps, each consisting of positions and a color. It also maintains a separate collection of smoothed positions using EMA, updating the smoothed position whenever a new position is added. The class includes methods to register new maps, add positions, retrieve map data, and draw the map legend and lines on a canvas. The ‘MapBuilder‘ is instantiated with an ‘emaAlpha‘ value, which dictates the smoothing factor for the EMA.

Additionally, the code provides a function ‘getMapBuilder‘ to get an instance of ‘MapBuilder‘ for a specific map name. If the map doesn’t already exist, it registers the map with a specified starting position and color. The returned instance offers methods to add new positions, get map data, and retrieve the current smoothed position. This setup allows for continuous and smooth tracking of positions, which can be visually represented on a canvas with reduced noise from erratic UWB readings.

Full code for the the map builder implementation is in the Appendix D: .1.2.

7.3.3 Handling Occlusion

Handling occlusion is done regarding two scenarios, handle occlusion in known environment (Refer to the code 7.4) and handle occlusion in unknown environment (Refer to the code 7.5). When the environment is known beforehand it is straightforward and path planing algorithm is used to calculate the path to the human subject to resolve the occlusion. When the environment is unknown exploration and dynamic path planning is used. Frontier based exploration is used given the priority to the closest frontiers to the target.

Handling Occlusion When Environment is Known

```
async handleOcclusionInKnownEnvironment(targetPosition: Position, map: number[][]) {
    const startPosition = convertPositionToInt(this.getLatestRobotPosition());
    const endPosition = convertPositionToInt(targetPosition);

    const path = AstarPathPlanner.findPath(startPosition, endPosition, map).slice(0, -5);
    this.drawAStarPath(path);
    await this.moveAlongPath(path);
}
```

Figure 7.4: Handling Occlusion in Known Environment

Code Explanation: The ‘handleOcclusionInKnownEnvironment‘ function manages the robot’s pathfinding when occlusions occur in an environment with a known map. It first determines the robot’s current position and the target position, then uses the A* pathfinding algorithm to find a path, excluding the last five steps to avoid collision. It visualizes the path on the canvas and moves the robot along this path.

Handling Occlusion When Environment is Unknown

```
async handleOcclusionInUnknownEnvironment(targetPosition: Position) {
    const endPosition = convertPositionToInt(targetPosition);
    const frontierDistanceThreshold = 3; // Set a distance threshold for filtering out similar frontiers
    let path: Position[] = [];
    path = AstarPathPlanner.findPath(convertPositionToInt(this.getLatestRobotPosition()), endPosition, this.discoveredMap);
    if (path.length === 0) {
        console.log("No path found");
    } else {
        // draw path
        this.drawAStarPath(path);
        // move robot along the path
        await this.moveAlongPath(path);
        return;
    }
    // Perform initial scan and update the map
    await this.updateDiscoveredMap();
    const frontierPriorityQueue = new FrontiersPriorityQueue();
    const frontiers = this.identifyFrontiers();
    // Sort the frontiers based on distance
    frontiers.forEach(frontier => {
        frontierPriorityQueue.addFrontier({ ...frontier, distance: 0 }, endPosition);
    });

    do {
        const frontier = frontierPriorityQueue.popFrontier();
        if (!frontier) {
            break;
        }
        if (this.canMoveToPosition(frontier)) {
            await this.moveToFrontierInDiscoveredMap(frontier);
            this.updateDiscoveredMap();
            const frontiers = this.identifyFrontiers();
            // draw filtered frontiers
            frontiers.forEach(({ x, y }) => {
                ExploredMapCanvasImpl!.drawRectangle(`filtered-frontier-${x}-${y}`, x, y, 1, 1, "blue", "blue", 1);
            });
            // Sort the frontiers based on distance
            frontiers.forEach(frontier => {
                frontierPriorityQueue.addFrontier({ ...frontier, distance: 0 }, endPosition);
            });
        }
        if (this.isPositionsNear(frontier, endPosition, frontierDistanceThreshold)) {
            break;
        }
    } while (!frontierPriorityQueue.isEmpty());
    // Perform A* path planning
    path = AstarPathPlanner.findPath(convertPositionToInt(this.getLatestRobotPosition()), endPosition, this.discoveredMap);
    if (path.length === 0) {
        console.log("No path found");
    } else {
        // draw path
        this.drawAStarPath(path);
        // move robot along the path
        await this.moveAlongPath(path);
    }
}
```

Figure 7.5: Handling Occlusion in Unknown Environment

Code Explanation: The ‘handleOcclusionInUnknownEnvironment‘ function deals with occlusions in an environment where the map is unknown. It first attempts to find a path using A* path finding. If no path is found, it updates the discovered map using LiDAR readings to identify new frontiers (unexplored areas). These frontiers are prioritized based on their distance to the target. The robot then moves to the nearest frontier, updating the discovered map and re-evaluating the frontiers until a path to the target can be found and followed. The process includes visualizing both the frontiers and the path on the canvas.

Full code for the robot controller implementation is in the Appendix D: .1.3.

7.3.4 Path Planning

A* (A-Star) path planning algorithm is implemented to calculate an optimal path when the environment is known (See the implementation 7.6). Since A* is a guided search algorithm, it can assess paths according to both their predicted cost and distance to the target. Heuristic information is used by A* to enhance search efficiency.

```
const heuristic = (pos0: Position, pos1: Position): number => {
    return Math.abs(pos0.x - pos1.x) + Math.abs(pos0.y - pos1.y); // Manhattan distance
};

export class AstarPathPlanner {
    public static findPath(start: Position, end: Position, grid: number[][]): Position[] {
        const openList: PathNode[] = [];
        const closedList: Set<string> = new Set();
        const startNode = new PathNode(start, null, 0, 0, 0);
        const endNode = new PathNode(end, null, 0, 0, 0);
        startNode.h = heuristic(start, end);
        startNode.f = startNode.g + startNode.h;
        openList.push(startNode);
        const getNeighbors = (node: PathNode): PathNode[] => {
            const { x, y } = node.position;
            const neighbors: PathNode[] = [];
            const directions = [
                [0, -1], // left
                [0, 1],  // right
                [-1, 0], // up
                [1, 0] // down
            ];
            for (const [dx, dy] of directions) {
                const nx = x + dx;
                const ny = y + dy;
                if (nx >= 0 && ny >= 0 && ny < grid.length && nx < grid[0].length && grid[ny][nx] === 0) {
                    neighbors.push(new PathNode({ x: nx, y: ny }, node, 0, 0, 0));
                }
            }
            return neighbors;
        };
        const reconstructPath = (endNode: PathNode): Position[] => {
            const path: Position[] = [];
            let currentNode: PathNode | null = endNode;
            while (currentNode !== null) {
                path.push(currentNode.position);
                currentNode = currentNode.parent;
            }
            return path.reverse();
        };
        while (openList.length > 0) {
            let currentNode = openList.reduce((prev, curr) => (prev.f < curr.f ? prev : curr));
            if (currentNode.position.x === endNode.position.x && currentNode.position.y === endNode.position.y) {
                return reconstructPath(currentNode);
            }
            openList.splice(openList.indexOf(currentNode), 1);
            closedList.add(`${currentNode.position.x},${currentNode.position.y}`);
            const neighbors = getNeighbors(currentNode);
            for (const neighbor of neighbors) {
                if (closedList.has(`${neighbor.position.x},${neighbor.position.y}`)) {
                    continue;
                }
                neighbor.g = currentNode.g + 1;
                neighbor.h = heuristic(neighbor.position, endNode.position);
                neighbor.f = neighbor.g + neighbor.h;
                const existingNode = openList.find(node => node.position.x === neighbor.position.x
                    && node.position.y === neighbor.position.y);
                if (!existingNode || neighbor.g < existingNode.g) {
                    openList.push(neighbor);
                }
            }
        }
        return []; // No path found
    }
```

Figure 7.6: A Path Planner*

Code Explanation: This code implements the A* pathfinding algorithm using TypeScript. It defines a ‘PathNode‘ class to represent nodes in the path, including their position, parent node, and cost values (g, h, f). The ‘AstarPathPlanner‘ class contains the ‘findPath‘ method, which takes a start position, end position, and a grid representing the map. The method opens a list of nodes to be traversed and a list of nodes already traversed. It calculates the heuristic (Manhattan distance) to calculate the cost from the this node to the destination. The algorithm iteratively processes nodes with the lowest cost (f), exploring their neighbors, updating costs, and reconstructing the path when the end node is reached. If no path is found, it returns an empty array.

Full code for the A* path planner is in the Appendix D: .1.4.

7.3.5 Multitrilateration

This function uses the mathematical principle of trilateration, where the intersection of three circles with known centers and radius determines the position of the object. The function extracts coordinates and distances from the readings, calculates intermediate values, sets up and solves linear equations to find the intersection point, and returns the estimated position (See the implementation 7.7).

```
function getPositionFromUwbBearing(readings: TagBearing[]): Position {
    // Check if there are exactly 3 readings
    if (readings.length !== 3)
        throw new Error("Invalid readings: Must have 3 readings");

    const anchor1 = readings[0];
    const anchor2 = readings[1];
    const anchor3 = readings[2];

    // Get coordinates and distances
    const x1 = anchor1.anchorPosition.x;
    const y1 = anchor1.anchorPosition.y;
    const r1 = anchor1.distance;

    const x2 = anchor2.anchorPosition.x;
    const y2 = anchor2.anchorPosition.y;
    const r2 = anchor2.distance;

    const x3 = anchor3.                (property) TagBearing.anchorPosition: Position
    const y3 = anchor3.anchorPosition.y;
    const r3 = anchor3.distance;

    // Calculate intermediate values
    const x1Square = x1 * x1;
    const y1Square = y1 * y1;
    const x2Square = x2 * x2;
    const y2Square = y2 * y2;
    const x3Square = x3 * x3;
    const y3Square = y3 * y3;
    const r1Square = r1 * r1;
    const r2Square = r2 * r2;
    const r3Square = r3 * r3;

    // Calculate trilateration
    const A = 2 * (x2 - x1);
    const B = 2 * (y2 - y1);
    const C = r1Square - r2Square - x1Square + x2Square - y1Square + y2Square;
    const D = 2 * (x3 - x2);
    const E = 2 * (y3 - y2);
    const F = r2Square - r3Square - x2Square + x3Square - y2Square + y3Square;

    // Calculate intersection point
    const x = (C * E - F * B) / (E * A - B * D);
    const y = (C * D - A * F) / (B * D - A * E);

    return { x, y };
}
```

Figure 7.7: Estimating Position from UWB Bearings

Full explanation of this function is in the Appendix C: UWB Trilateration Function.

7.4 Simulator Interface

Simulator interface contains controls to adjust the simulation parameters, simulated environment visualizing pane, map builder visualizer and LiDAR based environment exploration map (Refer to figure 7.8).

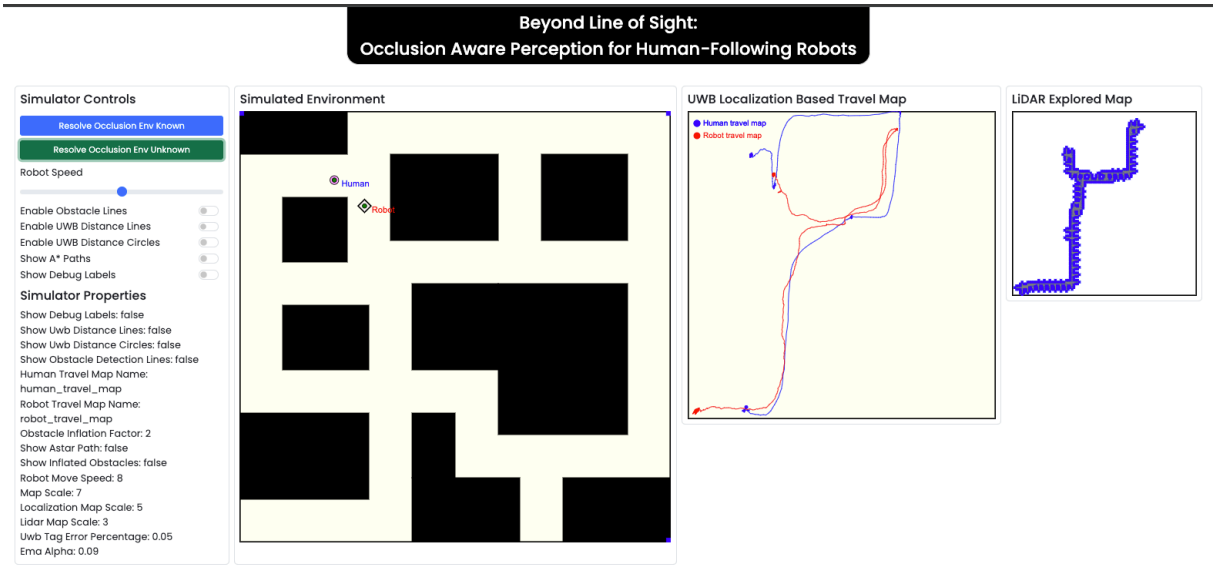

Figure 7.8: Simulator Interface

7.5 Chapter Summary

The chapter explores the technology selection process for implementing the occlusion-aware perception system. TypeScript and C# are chosen as the primary programming languages, while frameworks like Express.js and Konva are selected to facilitate server-side development and 2D graphics rendering, respectively. Visual Studio Code is adopted as the primary IDE for its robust features and compatiblity with TypeScript, with additional tools such as Node.js, Unity3D, and Git & GitHub utilized for various development tasks. Overall, the technology selection provides a solid foundation for implementing the system's core functionalities.

8 TESTING

8.1 Chapter Overview

This chapter presents a detailed analysis of the testing phase for the proposed solution. It outlines the objectives and goals of testing, the testing criteria used, and the methodologies for both functional and non-functional testing. The chapter includes tables summarizing the results of functional requirement testing, module and integration testing, and performance testing. It also covers accuracy testing and highlights the limitations of the testing process.

8.2 Objective & Goals of Testing

The primary objective of the testing phase for this project is to ensure the proposed system's reliability, accuracy, and robustness through software simulations. The goals are as follows:

- **Validate Functionality**: Ensure that all components of the system function as intended within the simulation environment.
- **Assess Accuracy**: Measure the accuracy of the localization system, path planning, and map building, specifically the UWB-based localization, in tracking and navigation.
- **Optimize Performance**: Evaluate the system's performance within the simulation to identify any bottlenecks or inefficiencies.
- **Identify and Fix Bugs**: Detect and resolve any software bugs or issues within the simulation.

8.3 Testing Criteria

The testing criteria for this project have been established to evaluate the system comprehensively, aiming to ensure that the implemented system aligns closely with the expected outcomes. The criteria are designed to cover both functional and structural aspects of the system.

- **Functional Quality**: This entails evaluating the system's functionalities and essential technical components to see how well it complies with the established design and func-

tional specifications.

- **Structural Quality**: This entails testing the system's non-functional needs to make sure it satisfies performance, accuracy, and scalability requirements.

8.4 Functional Requirement Testing

Identified functional requirements on the Chapter 4 are tested and the results summarised in the table 8.1.

#	FR ID	Test Case	Expected Result	Actual Result	Status
1	FR01	There Must be a wearable tracker that human subject can wear.	Attached wearable with UWB tag should communicate with the system.	Attached wearable with UWB tag communicates with the system.	Passed
2	FR02	There Must have a indoor localization system that can track position of human and the robot.	UWB localization system should track and send the location data to the main controller every 100ms.	UWB localization system tracks and sends the location data to the main controller every 100ms.	Passed
3	FR03	There Must be a map builder module records the way points of the user and robot sequentially.	The mapper module should store the location data in an Array preserving the order they arrive.	The mapper module stores the location data in an Array preserving the order they arrive.	Passed
4	FR04	The simulator Should have a real time display of localization processings.	The system should display real-time location data and mapping on the user interface.	The system displays real-time location data and mapping on the user interface.	Passed

5	FR05	The system Should have a path planing algorithm to calcuate the optimal path to resolve and occlusion when the environment is known.	The system should calculate and display the optimal path avoiding occlusions in a known environment using A* path planning algorithm.	The system calculates and displays the optimal path avoiding occlusions in a known environment using A* path planning algorithm.	Passed
6	FR06	There Should have a way of resolving occlusions when the environment is not known prior.	The system should explore and map the environment, updating the map and path dynamically to resolve occlusions.	The system explores and maps the environment, updating the map and path dynamically to resolve occlusions.	Passed

Table 8.1: Testing of Functional Requirements

8.5 Module and Integration Testing

Module	**Input**	**Expected Output**	**Actual Output**	**Status**
Simulator Setup	Environment settings, obstacle positions	Environment initialized with correct settings and obstacles in place	Environment initialized with correct settings and obstacles in place	Passed
Map Builder	Localization data from UWB tracker	Sequentially stored waypoints of the robot and human subject	Sequentially stored waypoints of the robot and human subject	Passed
Robot Controller	Commands from the path planner	Robot moves according to the path planner commands	Robot moves according to the path planner commands	Passed

Path Planner	Current location data, destination point	Optimal path to the destination avoiding obstacles	Optimal path to the destination avoiding obstacles	Passed
2D Visualizer	Localization data, map data	Real-time visualization of the robot and human subject positions	Real-time visualization of the robot and human subject positions	Passed
Coordinates Processor	Raw localization data	Processed coordinates with reduced noise and error	Processed coordinates with reduced noise and error	Passed

Table 8.2: Module and Integration Testing

8.6 Non-Functional Testing

8.6.1 Performance Testing

Initially the performance of the occlusion recovery algorithm under different numbers of static obstacles was evaluated. This test is done without knowing anything about the environment. This includes exploring the environment and find the human subject. The experiment setup is described in Appendix E: Functional Testing.

Environment Is Unknown

Frontier based environment exploration is used along with A* path planing algorithm to find the path to the target human subject.

# Static Obstacles	**Average Time to Recover Occlusion (s)**
2	10.06
3	15.34
6	18.66
9	24.91

Table 8.3: Number of Obstacles Vs. Runtime for Occlusion Recovery (Env unknown)

Environment Is Known

Since the environment setup is previously known; A* path planning algorithm is used to estimate optimal path to the human subject.

# Static Obstacles	Average Time to Recover Occlusion (s)
2	7.09
3	9.78
6	14.32
9	17.08

Table 8.4: Number of Obstacles Vs. Runtime for Occlusion Recovery (Env known)

Comparison of Runtime

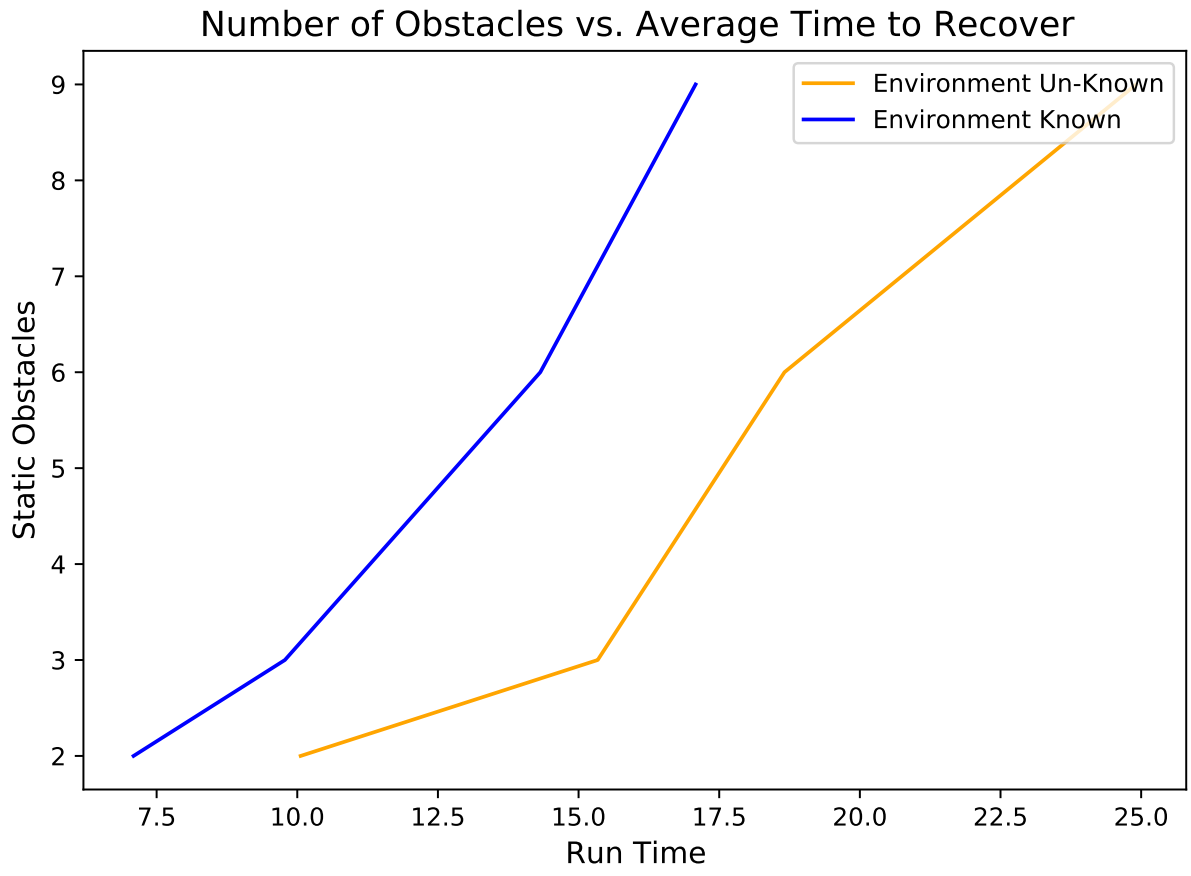

Figure 8.1: Comparison of runtime When Environment known vs unknown

Observations and Conclusion

- As the number of obstacles increases, the runtime of the occlusion recovery algorithm increases. However as the environment is explored incrementally the runtime becomes less.
- This indicates the algorithm's run time affects by the unexplored territory of the environment and as it is explored even the number of obstacles are increased the runtime can be lower.

Resource Usage for Environment Discovery and Occlusion Handling Algorithm

Below figure (Refer figure 8.2) is the profiling report of resource usage during running the simulation of recovering occlusions when the environment is unexplored and unknown initially. The simulation is carried on for 2 minutes of time with different places of occlusions.

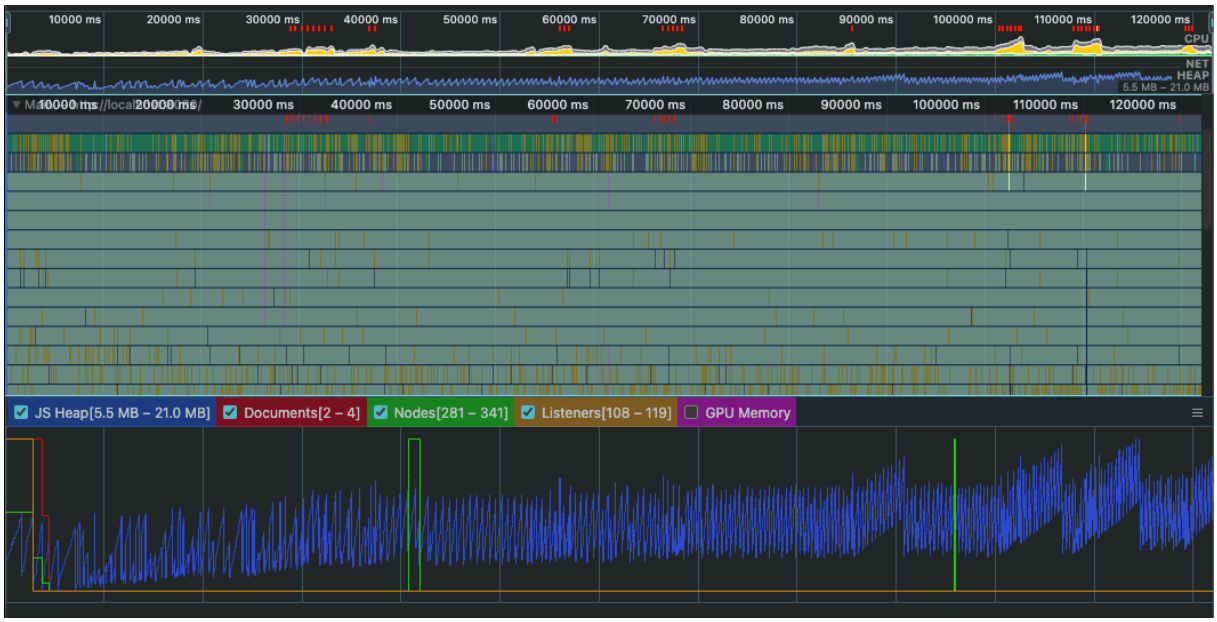

Figure 8.2: Resource Usage Profiling

8.6.2 Accuracy Testing

Accuracy of the system defines how accurately it can resolve an occlusion in a given scenario. That the robot should move to a place where it can visually see the subject when the occlusion recovery is done. A key factor determines this accuracy is the accuracy of the localisation system. The more accurate the localization system the algorithm can produce accurate estimations.

However UWB is the most accurate localisation systems in the market today, these systems are prone to errors they tend to produce output with some error margin. UWB technology is accurate to 10-30 centimeters in general.

Test is carried out to asses the accuracy of the algorithm and time taken to estimate a successful route is measured with respected to the error margin of UWB readings. The experiment setup is described in Appendix E: Functional Testing.

UWB Error Margin(%)	**Successful Recoveries**	**Average Time Taken**
5%	100%	6.08
8%	100%	6.6
10%	100%	6.64
12%	80%	7.43
15%	80%	7.54

Table 8.5: Number of Obstacles Vs. Runtime for Occlusion Recovery (Env known)

8.7 Limitations of Testing Process

The testing process in this project primarily uses a simulation environment, leading to several limitations:

- **Run on Actual Hardware**: This setup is needed to run on actual hardware devices to find the best metrics.
- **Lack of Real-World Setup**: Simulations cannot capture all real-world complexities, such as physical obstructions, environmental variations, and unpredictable human interactions.
- **Unanticipated Issues**: Practical use may reveal unique environmental challenges, unforeseen user behaviors, and hardware-software integration problems not evident in simulations.

8.8 Chapter Summary

In this chapter, the testing results of the occlusion-aware perception system are presented. The objectives and goals were clearly defined, aiming to validate the functionality, assess the ac-

curacy, optimize performance, and identify any potential bugs. Functional requirements were tested, and the results were summarized in a comprehensive table. Module and integration testing were conducted to ensure all system components worked harmoniously. Performance testing was carried out under different scenarios, and accuracy testing was performed to evaluate the system's capability to resolve occlusions accurately. The limitations of the testing process were acknowledged, noting that real-world testing is necessary to capture all potential challenges.

9 EVALUATION

9.1 Overview

This chapter provides a comprehensive project evaluation, focusing on the methodology, criteria, and results from expert evaluations. The qualitative approach is emphasized, with benchmarking against similar works to provide context and validation. Various evaluators, including technical and domain experts, have been selected to assess the project critically. The evaluation covers functional and non-functional requirements, highlighting the project's achievements and areas for future improvement. Limitations of the evaluation process are also discussed, acknowledging the challenges of modeling real-world scenarios through simulations.

9.2 Evaluation Methodology & Approach

The evaluation methodology for this project primarily utilizes a qualitative approach, which is well-suited for assessing the nuances and complexities inherent in the system. This approach enables a thorough exploration of the project's strengths and weaknesses through detailed feedback from domain and technical experts. In addition to qualitative evaluations, the project includes benchmarking and comparison with similar works.

9.3 Evaluation Criteria

Based on the features of the project a qualitative evaluation approach is well suited rather than a quantitative one. The outcomes of the project is compared with the more recent studies using qualitative indicators. Evaluators will be selected to critically assess the system using the criteria listed below (Refer table 9.1):

Criteria	Purpose of Evaluation
Project Overall Concept	To gather comments and insights on the system from the target audience related to the domain. This helps in understanding the overall reception and perceived value of the project.
Scope and Depth of the Project	To evaluate whether the project's scope and depth are adequate, particularly from the perspective of domain experts for the given time frame and Master's degree.
System Design, Architecture, and Implementation	To assess whether the system design, architecture, and implementation meet the required standards and best practices. This ensures that the system is built on a solid foundation.
Solution and Prototype	To determine if the implemented prototype effectively demonstrates the proposed solution and acts as a proof of concept.
Accuracy and Performance	To measure the accuracy and performance of the system, particularly focusing on the UWB-based localization and the system's ability to handle occlusions and navigate accurately.
Future Improvements	To identify any bugs, gaps, or potential areas for improvement in the system, providing insights into future development and enhancement opportunities.

Table 9.1: Evaluation Criteria

9.4 Self-Evaluation

The table below depicts the self evaluation of the project from the author's perspective (See table 9.2).

Criteria	Self Evaluation
Project Overall Concept	The project's overall concept is sound and addresses a real-world problem in occlusion handling for navigation systems. Integrating a localization system into a HFR can mitigate the risks of failure due to occlusion.
Scope and Depth of the Project	The project covers a broad scope, including localization, path planning, and navigation. The Depth of research and implementation demonstrates a comprehensive understanding of the domain.
System Design, Architecture, and Implementation	The system design and architecture are well-structured and adhere to best practices in OOP and Software engineering.
Solution and Prototype	The prototype serves as a proof of concept for the proposed solution. However, it would have been better to implement these codes in Python or C language as they can be directly integrated into the implementation, unlike Javascript.
Accuracy and Performance	The accuracy and performance of the system are impressive within the simulation environment. The UWB-based localization has shown high precision. Nonetheless, the true performance can only be measured through extensive real-world testing, which is currently a limitation.
Future Improvements	Future improvements should focus on transitioning from simulation to real-world testing. Integration of a primary subject tracking method (ML/DL Based Approach) is crucial.

Table 9.2: Self Evaluation

9.5 Selection of Evaluators

As this is a simulation and not a widely available customer product, only technical experts and domain experts are chosen as evaluators (Refer table 9.3).

ID	Evaluator Category
EV1	Domain expert in the domains of Unmanned Aerial Vehicle (UAV), Computer vision, Vision based navigation and Robotic systems.
EV2	Technical expert in the industry of Robotics, IoT Systems & Electronics, Autonomous Navigation and Machine Learning.

Table 9.3: Selection of the Evaluators

9.6 Evaluation Results from Experts

A summary of the experts review is included in the Appendix Table F: .1 Evaluation Result from Experts.

Qualitative Analysis

Theme	**Evaluator**	**Brief Opinion**
Concept	EV1	The concept is innovative and addresses a significant challenge in autonomous navigation, particularly in occluded environments.
Scope	EV2	The scope is comprehensive, covering essential aspects of autonomous navigation. However, it could benefit from additional real-world testing scenarios.
Solution	EV1	This solution is novel and the simulation planned is quite good for a concept and the prototype demonstrations and it conveys the proof of concept.
Solution Architecture	EV2	The system design and architecture are adhere to best practices in the industry. There is no much deviations from similar standard projects.
Implementation	EV2	Have identified the best possible solutions yet minimal approach which is good for a simulation. The simulator program captures most of the requirements.
Future Improvements	EV1	Future work should focus on integrating additional sensors get more accurate localization while reducing the number of anchor points needed.

	EV2	Consider exploring machine learning techniques. Enhance path planning with other D* or similar algorithm.

Table 9.4: Evaluation Result from Experts

9.7 Evaluation of Functional Requirements

ID	**Requirement Description**	**MoSCoW Priority**	**Status**
FR01	Implement essential sensors for human subject tracking in the simulated wearable device.	MH	Done
FR02	Identify and implement an optimal indoor localization method for tracking human subjects and robot movements.	MH	Done
FR03	Implement an algorithm to integrate data from localization and wearable trackers for constructing a map for robot guidance.	MH	Done
FR04	Implement real-time data visualization features to monitor the tracking and mapping process during simulation.	SH	Done
FR05	Implement path planning and optimal path selection algorithm to resolve occlusion when the environment is known.	SH	Done
FR06	Implement frontier-based environment mapping and resolve occlusion when the environment is unknown.	SH	Done
FR07	Integration of a vision-based tracking system as a primary tracking mechanism.	WH	Not Done

Table 9.5: Evaluation of Functional Requirements

9.8 Evaluation of Non-Functional Requirements

ID	Requirement	Priority	Status
NFR01	Accuracy	MH	Achieved
NFR02	Performance	MH	Achieved
NFR03	Usability	SH	Achieved
NFR04	Adaptability	CH	Achieved
NFR05	Scalability	WH	Achieved

Table 9.6: Evaluation of Non-Functional Requirements

9.9 Limitations of Evaluation

Several limitations are there in the evaluation process due to modeling a real-world scenario through simulation. The lack of real-world fidelity results in differences in system performance when deployed outside the simulated environment. The simplified environment needs to capture complex interactions and unforeseen events. Human factors and varying usage patterns are challenging to model. Testing and validation processes may not cover all real-world use cases, and reliance on assumptions can bias results, affecting the generalizability of the findings.

9.10 Summary

This project's evaluation methodology and approach primarily rely on qualitative feedback, which is well-suited for assessing complex systems like the one developed. Expert evaluations provided valuable insights into the project's overall concept, scope, system design, accuracy, performance, and potential improvements. The project successfully met most functional and non-functional requirements, as evidenced by the high completion percentages. However, the evaluation also identified limitations due to the reliance on simulations, which may only partially capture real-world complexities. These findings underscore the need for future work to include more extensive real-world testing and enhancements to validate and refine the system further.

10 CONCLUSION

10.1 Chapter Overview

The chapter highlights the achievement of research objectives, including problem identification, literature review, and system design, and showcases how knowledge from various course modules was applied to the project. It also discusses the use of existing and new skills, the challenges faced during the research, deviations from the initial plan, and the limitations of the study. Future enhancements are proposed to address these limitations, and the chapter concludes with a reflection on the project's contributions to the field of autonomous robot navigation and occlusion handling.

10.2 Achievements of Research Aims & Objectives

10.2.1 Achievements of Research Aims

Research Aim: *Design, develop, and evaluate occlusion-aware perception algorithms for HFR by integrating wearable devices and sensor fusion techniques to enhance accuracy and robustness in tracking human subjects prone to occlusion, facilitating seamless navigation and interaction in dynamic environments.*

The research aim is achieved by developing a simulated environment featuring obstacles and potential occlusion scenarios. A robust occlusion recovery method is proposed and demonstrated, incorporating the latest UWB indoor localization technologies. Additionally, the project successfully covered optimal path planning for known environments and implmented exploration and mapping techniques for unknown environments with the help of indoor localization.

10.2.2 Achievements of Research Objectives

The research objectives along wit the respective learning outcomes covered and the statuses are shown in the table 10.1.

Research Objectives	**Learning Outcome**	**Status**

Problem Identification: Identified problems in HFR when handling occlusions, determined the limitation of current object tracking methods.	LO1, LO2	Completed
Literature Review: Comprehensive analysis of domain and existing literature. Identified current trends and existing occlusion handling techniques and their drawbacks.	LO1, LO2, LO4, LO6	Completed
Requirement Analysis: Gathered requirements through interviews, literature review and prototyping.	LO2, LO3	Completed
Design: Designed and system architecture that can simulate robotic experiments.	LO1, LO3, LO5	Completed
Development: Developed the simulator based environment to model the problem and the solution.	LO6, LO4, LO5	Completed
Testing and Evaluation: Conducted both qualitative and quantitative tests and evaluations to validate the project findings.	LO2, LO5, LO6	Completed

Table 10.1: Achievements of Research Objectives

10.3 Utilization of the Knowledge from the Course

The table below (refer table 10.2) summarizes how the knowledge gained from course modules was utilized in the project:

Module(s)	**Utilized Knowledge**
Advanced software Design	Applied principles of advanced software design to ensure a robust and scalable system architecture.
Concurrent and Distributed Systems	Used the knowledge to run the parallel tasks and optimize the algorithms with concurrent processing.

Research Methods and Professional Practices	Employed research methodologies for systematic requirements gathering, literature review, and professional practices for project management and documentation.

Table 10.2: Utilization of Knowledge from the Course

10.4 Use of Existing Skills

This project leveraged my existing skills as a full-stack JavaScript developer, with primary expertise in React and Node.js. My extensive knowledge of these web technologies facilitated the development of the TypeScript-based web project. My longstanding interest in electronics, robotics, and IoT also provided a solid foundation for understanding and integrating various system components.

10.5 Use of New Skills

Throughout this project, I acquired and applied several new skills. I developed a deep understanding of Ultra-Wideband (UWB) technology, which is essential for indoor localization. I learned advanced path-planning techniques for navigating and overcoming occlusions in dynamic environments. Additionally, I gained proficiency in Unity 3D for simulation purposes and honed my skills in web canvas 2D graphics for real-time data visualization. My knowledge of Webpack also deepened, allowing for efficient bundling and optimization of the web application. These new skills were instrumental in achieving the project's objectives and delivering a robust solution.

10.6 Achievements of Learning Outcomes

What is being learned	**Learning Outcome**
Identified research gaps, formulated research topics, conducted literature reviews, created project proposals, defined scope, and created delivery timelines.	LO1, LO2, LO4, LO6

Applied UWB technology, path planning algorithms, and Unity 3D for simulation to solve autonomous navigation and occlusion handling issues.	LO3
Reflected on and defended research methods, design approaches, and tool choices through documentation and presentations.	LO3, LO4
Applied current standards and quality processes and addressed legal, ethical, and professional issues.	LO5
Produced project proposal, PSPD, and Thesis documenting the project, including research background, methodology, implementation, and evaluation.	LO4, LO6

Table 10.3: Achievements of Learning Outcomes

10.7 Problems and Challenges Faced

Problem/Challenge	**Solution**
Project domain is broad	When developing a solution which tackles robot navigation, there are several features that needs to develop (Obstacle avoidance, Path planning, Environment exploration). Had to develop simplified version of these.
Complex Mathematics	All of above mentioned features are working with number as calculation at the heart. Learned most of these principals from scratch.
Time Constraints, Budgeting and Resources	Initial idea was to build a hardware prototype of the solution. However it was difficult find the sensors and devices needed from the local market and importing them costs a considerable amount. Had to go for a simulated solution.
Lack of Expertise	Initially it was difficult to even find a supervisor for the project. Hard to find a personal to get a feedback or suggestion. Had to self study most of the concepts. It was difficult to find experts to evaluate this solution.

Table 10.4: Problems and Challenges

10.8 Deviations

Several changes were made to the project's initial scope based on initial experiments and insights from the requirements-gathering process. The original plan to design a custom wearable tracker equipped with IMU sensors was abandoned due to the high error accumulation and the need for complex algorithms deemed infeasible within the project's time constraints. Instead, UWB technology was adopted for tracking the human subject, with a generalized interface in the mapping module to allow future integration of a custom wearable device. Additionally, testing in different light conditions was deemed unnecessary as the simulation does not rely on visual inputs or vision-based components, making the solution inherently suitable for any lighting condition.

10.9 Limitations of The Research

The research conducted has several limitations that should be acknowledged:

- **Simulation-Based Evaluation**: The study relied heavily on simulation environments for testing and validation. While simulations provide valuable insights, they cannot fully replicate the complexities and unpredictability of real-world scenarios.
- **Not Suitable for Open Environment**: Since this solution rely on indoor localization system; it is not solely applicable for outdoor environment.
- **Limited Real-World Testing**: Due to constraints in time and resources, extensive real-world testing was not conducted. As a result, the system's actual performance in diverse and dynamic environments remains uncertain.
- **Simplified Environmental Models**: The simulation environment used simplified models that may not capture all the nuances of real-world settings, such as varying terrain, lighting conditions, and unexpected obstacles.
- **Hardware and Cost Constraints**: Implementing custom wearable devices with advanced sensors was deemed infeasible within the project's scope and budget, limiting the explo-

ration of potential hardware solutions.

10.10 Future Enhancements

Future enhancements to this research could address the limitations and expand the capabilities of the system:

- **Extensive Real-World Testing**: Re-create this prototype with real hardware and test. This will help identify practical challenges and refine the system for real-world applications.
- **Error Reduction For UWB**: Current EMA(Exponential Moving Average) for map module works well for errors in the UWB readings. However it is better to implement a Kalman filter based outlier filtering method for map construction.
- **Number Of UWB Anchors**: Experiment combining other sensors into wearable tracker so it can reduce the number of anchor points needed in the infrustructure.
- **Integration of Vision-Based Tracking**: **Incorporate computer vision technologies** Integrate a primary subject tracking mechanism based on computer vision and machine learning.
- **Advanced Sensor Fusion**: **Develop advanced sensor fusion algorithms** that integrate data from multiple sources, such as LiDAR, IMU, and cameras, to enhance the accuracy and reliability of localization and navigation.
-

10.11 Achievement of the Contribution to Body of Knowledge

This section details the study's contributions to knowledge in autonomous robot navigation and the specific problem domain of occlusion handling in Human Following Robots.

10.11.1 Achievement of Contribution to Research Domain

The research made significant strides in the **Autonomous Robot Navigation** field by tackling the challenges associated with occlusion handlng in HFR. The study provided a solid mechanism to handle occlusion more reliably and adaptive navigation solutions by leveraging Ultra-Wideband

(UWB) technology. These advancements address gaps in current autonomous navigation systems and offer a framework for future innovations in the field.

10.11.2 Achievement of Contribution to Problem Domain

The research effectively addressed the problem of occlusion handling in HFR by combining indoor localization techniques with existing human subject tracking methods.

Implementing UWB-based localization provided precise tracking and mapping algorithms, which allowed the robot to navigate and recover from occlusions efficiently, showing that the proposed approach can handle occlusions in real-time, maintaining accurate tracking and navigation.

This research fills a crucial gap in the literature, where previous studies focused on minimizing occlusions rather than addressing them post-occurrence.

10.12 Concluding Remarks

This research project has made significant strides in addressing the challenge of occlusion handling for human following robots. By integrating data from indoor localization systems, the proposed occlusion-aware perception system demonstrates the potential to accurately track human subjects even in the face of complete occlusion. The testing results reveal that the system maintains high success rates in recovering from occlusion, with average recovery times ranging from 6.08 to 6.64 seconds for UWB error margins of 5% to 10%.

The project's contributions to the research domain include the development of algorithms for handling complete occlusion, the implementation of a simulated custom wearable tracking device, and the creation of simulated environments for testing and validation. These advancements pave the way for enhanced occlusion handling capabilities in human-following robots, with potential applications in personal assistance, security, and entertainment. While the project has made significant progress, there are still areas for improvement, such as optimizing recovery times, enhancing accuracy in the presence of larger UWB error margins, and expanding the system's capabilities to handle more complex environments and scenarios. Future research should focus on refining the algorithms, exploring alternative localization technologies, and conducting real-world testing to further validate the system's performance and applicability.

REFERENCES

Algabri, Redhwan and Mun Taek Choi (2020). "Deep-learning-based indoor human following of mobile robot using color feature". In: *Sensors (Switzerland)* 20.9. ISSN: 14248220. DOI: 10.3390/s20092699.

Bai, Nan et al. (2020). "A High-Precision and Low-Cost IMU-Based Indoor Pedestrian Positioning Technique". In: *IEEE Sensors Journal* 20.12, pp. 6716–6726. ISSN: 15581748. DOI: 10.1109/JSEN.2020.2976102.

Feng, Daquan et al. (2020). "Kalman-Filter-Based Integration of IMU and UWB for High-Accuracy Indoor Positioning and Navigation". In: *IEEE Internet of Things Journal* 7.4, pp. 3133–3146. ISSN: 23274662. DOI: 10.1109/JIOT.2020.2965115.

Feng, Tao et al. (2018). "A Human-Tracking Robot Using Ultra Wideband Technology". In: *IEEE Access* 6, pp. 42541–42550. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2859754.

Hassan, Muhammad Sarmad et al. (2016). "A computationally low cost vision based tracking algorithm for human following robot". In: *Proceedings - 2016 the 2nd International Conference on Control, Automation and Robotics, ICCAR 2016*, pp. 62–65. DOI: 10.1109/ICCAR.2016.7486699.

Hoang, Minh Do, Sang Seok Yun, and Jong Suk Choi (2017). "The reliable recovery mechanism for person-following robot in case of missing target". In: *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2017*, pp. 800–803. DOI: 10.1109/URAI.2017.7992828.

Islam, Md Jahidul, Jungseok Hong, and Junaed Sattar (2019). "Person-following by autonomous robots: A categorical overview". In: *International Journal of Robotics Research* 38.14, pp. 1581–1618. ISSN: 17413176. DOI: 10.1177/0278364919881683. arXiv: 1803.08202.

Jeyatharan, Keshiha, H. E.M.H.B. Ekanayake, and K. D. Sandaruwan (2021). "Behavior Based Tracking for Human Following Robot". In: *2021 IEEE 16th International Conference on Industrial and Information Systems, ICIIS 2021 - Proceedings*, pp. 371–376. DOI: 10.1109/ICIIS53135.2021.9660756.

Kao, Wei-Wen (1991). "Integration of GPS and dead-reckoning navigation systems". In: *Vehicle Navigation and Information Systems Conference, 1991*. Vol. 2, pp. 635–643. DOI: 10.1109/VNIS.1991.205808.

Kastner, Linh et al. (2022). "Human-Following and -guiding in Crowded Environments using Semantic Deep-Reinforcement-Learning for Mobile Service Robots". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 833–839. ISSN: 10504729. DOI: 10.1109/ICRA46639.2022.9812111. arXiv: 2206.05771.

Kawarazaki, Noriyuki, Lucas Tetsuya Kuwae, and Tadashi Yoshidome (2015). "Development of Human Following Mobile Robot System Using Laser Range Scanner". In: *Procedia Computer Science* 76.Iris, pp. 455–460. ISSN: 18770509. DOI: 10.1016/j.procs.2015.12.310.

Kulkarni, J. P. and P. D. Pantawane (2022). "Person Following Robot Based on Real Time Single Object Tracking and RGB-D Image". In: *2022 International Conference on Signal and Information Processing, IConSIP 2022*, pp. 1–5. DOI: 10.1109/ICoNSIP49665.2022.10007460.

Kumar Ashe, Avijit and K. Madhava Krishna (2021). "Maneuvering Intersections Occlusions Using MPC-Based Prioritized Tracking for Differential Drive Person following Robot". In: *IEEE International Conference on Automation Science and Engineering* 2021-Augus.1, pp. 1352–1357. ISSN: 21618089. DOI: 10.1109/CASE49439.2021.9551618.

Nagumo, Y. and A. Ohya (2001). "Human following behavior of an autonomous mobile robot using light-emitting device". In: *Proceedings 10th IEEE International Workshop on Robot*

and Human Interactive Communication. ROMAN 2001 (Cat. No.01TH8591), pp. 225–230. DOI: 10.1109/ROMAN.2001.981906.

Priyandoko, Gigih et al. (2017). "Human Following on ROS Framework". In: *Sinergi* 22.2, pp. 77–82.

Shu, Guang et al. (2012). "Part-based multiple-person tracking with partial occlusion handling". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1815–1821. ISSN: 10636919. DOI: 10.1109/CVPR.2012.6247879.

Sun, Shiying et al. (2016). "Human recognition for following robots with a Kinect sensor". In: *2016 IEEE International Conference on Robotics and Biomimetics, ROBIO 2016*, pp. 1331–1336. DOI: 10.1109/ROBIO.2016.7866511.

Tai, W. W. et al. (2019). "A Study of Ultrasonic Sensor Capability in Human Following Robot System". In: *IOP Conference Series: Materials Science and Engineering* 705.1. ISSN: 1757899X. DOI: 10.1088/1757-899X/705/1/012045.

Tsun, Mark Tee Kit, Bee Theng Lau, and Hudyjaya Siswoyo Jo (2018). "An improved indoor robot human-following navigation model using depth camera, active IR marker and proximity sensors fusion". In: *Robotics* 7.1. ISSN: 22186581. DOI: 10.3390/robotics7010004.

Xue, Guang et al. (2022). "UWB-Based Adaptable Side-by-Side Following for Human-Following Robots". In: *2022 IEEE International Conference on Robotics and Biomimetics, ROBIO 2022*, pp. 333–338. DOI: 10.1109/ROBIO55434.2022.10011938.

Yang, Yuebin et al. (2013). "Using bearing-sensitive infrared sensor arrays in Motion localization for human-following robots". In: *2013 9th Asian Control Conference, ASCC 2013*, pp. 1–5. DOI: 10.1109/ASCC.2013.6606194.

Ye, Hanjing et al. (2023). "Robot Person Following Under Partial Occlusion". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2023-May, pp. 7591–7597. ISSN: 10504729. DOI: 10.1109/ICRA48891.2023.10160738. arXiv: 2302.02121.

Yuan, Jing et al. (2021). "Laser-Based Intersection-Aware Human following with a Mobile Robot in Indoor Environments". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1, pp. 354–369. ISSN: 21682232. DOI: 10.1109/TSMC.2018.2871104.

Zafari, Faheem, Athanasios Gkelias, and Kin K. Leung (2019). "A Survey of Indoor Localization Systems and Technologies". In: *IEEE Communications Surveys and Tutorials* 21.3, pp. 2568–2599. DOI: 10.1109/COMST.2019.2911558. arXiv: 1709.01015.

A: CONCEPT GRAPH

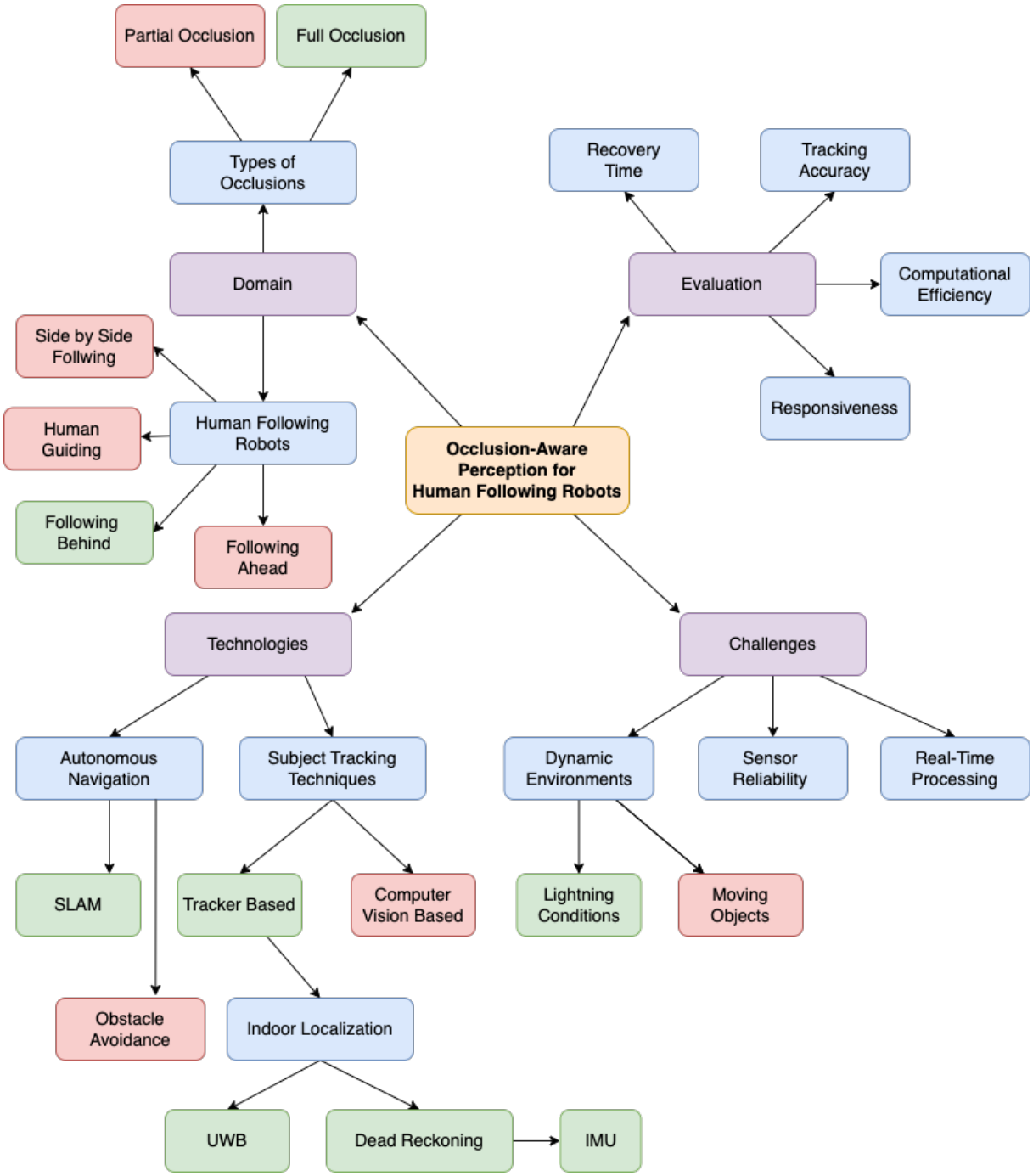

Figure A: .1: Concept Map (Self - Composed)

B: GANTT CHART

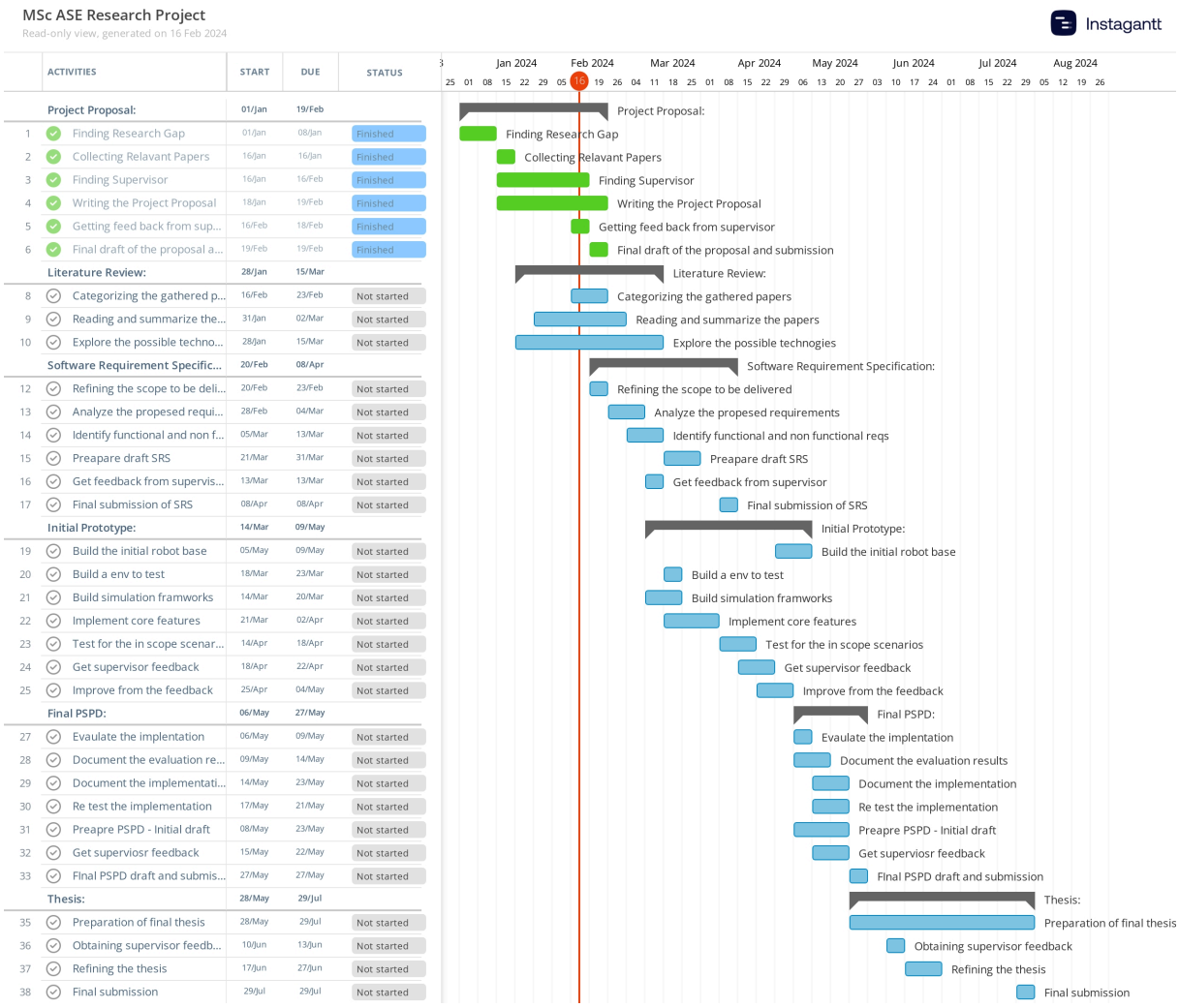

Figure B: .1: Project Timeline

C: UWB TRILATERATION FUNCTION

This function uses the mathematical principle of trilateration, where the intersection of three circles with known centers and radius determines the position of the object. The function extracts coordinates and distances from the readings, calculates intermediate values, sets up and solves linear equations to find the intersection point, and returns the estimated position.

1. **Input Validation:**
    - The function first checks if exactly three readings are provided. If not, it throws an error.
2. **Extract Coordinates and Distances:**
    - Extract the coordinates (x_1, y_1), (x_2, y_2), and (x_3, y_3) of each anchor point.
    - Extract the distances r_1, r_2, and r_3 from the object to each anchor point.
3. **Intermediate Calculations:**
    - Calculate the squares of the coordinates and distances:

$$x_1^2,\ y_1^2,\ x_2^2,\ y_2^2,\ x_3^2,\ y_3^2,\ r_1^2,\ r_2^2,\ r_3^2$$

4. **Set Up Linear Equations:**
    - Using the distances and coordinates, set up two linear equations based on the geometry of circles centered at each anchor point with radii equal to the distances:

$$A = 2(x_2 - x_1)$$
$$B = 2(y_2 - y_1)$$
$$C = r_1^2 - r_2^2 - x_1^2 + x_2^2 - y_1^2 + y_2^2$$
$$D = 2(x_3 - x_2)$$
$$E = 2(y_3 - y_2)$$
$$F = r_2^2 - r_3^2 - x_2^2 + x_3^2 - y_2^2 + y_3^2$$

5. **Solve for Intersection:**

    - Solve these linear equations to find the (x, y) coordinates where the three circles intersect:

$$x = \frac{C \cdot E - F \cdot B}{E \cdot A - B \cdot D}$$
$$y = \frac{C \cdot D - A \cdot F}{B \cdot D - A \cdot E}$$

6. **Return Position:**

    - Return the estimated position as an object containing x and y coordinates.

D: IMPLEMENTATION CODES

D: .1 Typescript Code for the Implementation

D: .1.1 Simulated Environment

```
import { Robot } from '../robot/robot';
import { Environment } from '../environment/environment';
import { Obstacle } from '../environment/obstacle';
import { Canvas, CanvasImpl, ExploredMapCanvasImpl, UWBMapCanvasImpl } from
    '../graphics/graphics';
import { UwbAnchor } from '../uwb/anchor';
import { UwbTag } from '../uwb/tag';
import { globalConfigsProvider } from '../configs';
import { Human } from '../human/human';
import { DrawingColor } from '../types';
import { getMapBuilder } from '../map_builder/map_builder';
import { getPositionFromUwbBearing } from '../utils';
import { RobotController } from '../robot/robot_controller';

const configs = {
    scale: globalConfigsProvider.getConfig("mapScale"),
    env: {
        width: 100,
        height: 100
    },
    // cords: [x, y], dims: [width, height]
    obstacles: [
        { cords: [0, 0], dims: [25, 10] },
        { cords: [10, 20], dims: [15, 15] },
        { cords: [10, 45], dims: [20, 15] },
        { cords: [0, 70], dims: [30, 20] },
        { cords: [40, 85], dims: [25, 15] },
        { cords: [40, 70], dims: [10, 15] },
        { cords: [75, 85], dims: [25, 15] },
        { cords: [60, 40], dims: [30, 35] },
        { cords: [40, 40], dims: [20, 20] },
        { cords: [70, 10], dims: [20, 20] },
        { cords: [35, 10], dims: [25, 20] },
    ]
}


function createEnvironment(configs: any, canvas: Canvas) {
    const env = new Environment(configs.env.width, configs.env.height,
    canvas);
    configs.obstacles.forEach((config: any) => {
        const obstacle = new Obstacle(config.dims[0], config.dims[1],
    config.cords[0], config.cords[1]);
        env.addObject(obstacle);
    });
    return env;
}

function initGraphicCanvases() {
    UWBMapCanvasImpl.setScale(globalConfigsProvider.getConfig("
```

```
    localizationMapScale"));
48      UWBMapCanvasImpl.drawBackdrop({ width: 100, height: 100, fillColor: "#
    FFFFF1", strokeColor: 'black' });
49
50      ExploredMapCanvasImpl.setScale(globalConfigsProvider.getConfig("
    lidarMapScale"));
51      ExploredMapCanvasImpl.drawBackdrop({ width: 100, height: 100, fillColor
    : "#FFFFFF", strokeColor: 'black' });
52
53      CanvasImpl.setScale(globalConfigsProvider.getConfig("mapScale"));
54  }
55
56  export function createSimulator(canvas: Canvas) {
57
58      initGraphicCanvases();
59
60      const env = createEnvironment(configs, canvas);
61
62      // Add robot object to environment
63      const robot = new Robot(5, 95, env);
64      robot.setID("robot-A");
65      env.addObject(robot);
66
67
68      // Add human subject to environment
69      const human = new Human(20, 95, env);
70      human.setID("human-A");
71      env.addObject(human);
72
73
74      // Add UWB anchors to environment
75      const anchor1 = new UwbAnchor(0, 0, "anchor_a");
76      const anchor2 = new UwbAnchor(99, 0, "anchor_b");
77      const anchor3 = new UwbAnchor(99, 99, "anchor_c");
78      env.addObject(anchor1);
79      env.addObject(anchor2);
80      env.addObject(anchor3);
81
82
83      // Attach UWB tag to robot and add to environment
84      const uwbTagRobot = new UwbTag(20, 30);
85      uwbTagRobot.attachAnchor(anchor1);
86      uwbTagRobot.attachAnchor(anchor2);
87      uwbTagRobot.attachAnchor(anchor3);
88      env.addObject(uwbTagRobot);
89      robot.attachUwbTag(uwbTagRobot);
90
91
92      // Attach UWB tag to human and add to environment
93      const uwbTagHuman = new UwbTag(20, 95);
94      uwbTagHuman.attachAnchor(anchor1);
95      uwbTagHuman.attachAnchor(anchor2);
96      uwbTagHuman.attachAnchor(anchor3);
97      env.addObject(uwbTagHuman);
98      human.attachUwbTag(uwbTagHuman);
99
100
101     // Create a map builder for the human
```

```
    const humanTravelMap = getMapBuilder(globalConfigsProvider.getConfig("humanTravelMapName"), human.getAveragedUwbBearing(), DrawingColor.BLUE);
    setInterval(() => {
        const position = getPositionFromUwbBearing(uwbTagHuman.getBearing());
        humanTravelMap.addPosition(position);
    }, 80);

    // Create a map builder for the robot
    const robotTravelMap = getMapBuilder(globalConfigsProvider.getConfig("robotTravelMapName"), robot.getAveragedUwbBearing(), DrawingColor.RED);
    setInterval(() => {
        const position = getPositionFromUwbBearing(uwbTagRobot.getBearing());
        robotTravelMap.addPosition(position);
    }, 80);

    const robotController = new RobotController(robot, robotTravelMap, canvas);

    return { env, robotController, human, uwbTagRobot, humanTravelMap };
}
```

D: .1.2 Map Builder

```
import { globalConfigsProvider } from "../configs";
import { Canvas, UWBMapCanvasImpl } from "../graphics/graphics";
import { DrawingColor, Position } from "../types";
import { humanizeString } from "../utils";

export interface MapEntry {
    color: DrawingColor;
    positions: Position[];
}

class MapBuilder {
    private map: Map<string, MapEntry> = new Map<string, MapEntry>();
    private canvas: Canvas;
    private emaAlpha: number;
    private emaPositions: Map<string, Position>;

    constructor(emaAlpha: number) {
        this.canvas = UWBMapCanvasImpl;
        this.emaAlpha = emaAlpha;
        this.emaPositions = new Map<string, Position>();
    }

    registerMap(mapName: string, color: DrawingColor, startingPosition: Position) {
        this.map.set(mapName, { color, positions: [] });
        this.emaPositions.set(mapName, startingPosition);
        this.drawMapLegend();
    }

    addPosition(mapName: string, position: Position) {
        const mapEntry = this.map.get(mapName);
        if (mapEntry) {
```

```
32              // Update EMA
33              const lastEmaPosition = this.emaPositions.get(mapName) || { x:
    0, y: 0 };
34              const emaPosition = {
35                  x: this.emaAlpha * position.x + (1 - this.emaAlpha) *
    lastEmaPosition.x,
36                  y: this.emaAlpha * position.y + (1 - this.emaAlpha) *
    lastEmaPosition.y
37              };
38              this.emaPositions.set(mapName, emaPosition);
39
40              // Add smoothed position
41              mapEntry.positions.push(emaPosition);
42              this.drawMapLinePolygon(mapName);
43          }
44      }
45
46      getMap(mapName: string) {
47          return this.map.get(mapName);
48      }
49
50      drawMapLegend() {
51          let index = 0;
52          this.map.forEach((map, mapName) => {
53              this.canvas.drawCircle(`${mapName}-legend`, 3, 4 + index * 4,
    1, map.color, map.color, 1);
54              this.canvas.drawText(`${mapName}-legend-text`, 5, 2.8 + index *
     4, humanizeString(mapName), map.color, 2.5);
55              index++;
56          });
57      }
58
59      drawMapLinePolygon(mapName: string) {
60          const map = this.map.get(mapName);
61          if (map) {
62              // draw map line
63              this.canvas.removeObject(mapName);
64              this.canvas.drawContiguousLine(mapName, map.positions, map.
    color, 1);
65
66              // draw start point
67              this.canvas.removeObject(`${mapName}-start`);
68              this.canvas.drawCircle(`${mapName}-start`, map.positions[0].x,
    map.positions[0].y, 0.5, map.color, map.color, 1);
69
70              // draw current point
71              this.canvas.removeObject(`${mapName}-current`);
72              this.canvas.drawCircle(`${mapName}-current`, map.positions[map.
    positions.length - 1].x,
73                  map.positions[map.positions.length - 1].y, 0.5, map.color,
    map.color, 1);
74          }
75      }
76  }
77
78  const mapBuilder = new MapBuilder(globalConfigsProvider.getConfig("emaAlpha
    "));
79
```

```
export interface MapBuilderInstance {
    addPosition: (position: Position) => void;
    getMap: () => MapEntry | undefined;
    getCurrentPosition: () => Position;
}

export const getMapBuilder = (mapName: string, startingPosition: Position, color: DrawingColor = DrawingColor.BLUE): MapBuilderInstance => {
    if (!mapBuilder.getMap(mapName)) {
        mapBuilder.registerMap(mapName, color as DrawingColor, startingPosition);
    }
    return {
        addPosition: (position: Position) => mapBuilder.addPosition(mapName, position),
        getMap: () => mapBuilder.getMap(mapName),
        getCurrentPosition: () => {
            const map = mapBuilder.getMap(mapName);
            const lastPosition = map?.positions[map.positions.length - 1];
            return lastPosition || { x: 0, y: 0 };
        }
    };
};
```

D: .1.3 Handling Occlusion - Robot Controller

```
    import { globalConfigsProvider } from "../configs";
import { Canvas, ExploredMapCanvasImpl } from "../graphics/graphics";
import { MapBuilderInstance } from "../map_builder/map_builder";
import { Direction, Position } from "../types";
import { convertPositionToInt } from "../utils";
import { AstarPathPlanner } from "./astar";
import { Robot } from "./robot";

interface PositionWithDistance extends Position {
    distance: number;
}

class FrontiersPriorityQueue {
    // this should be a priority queue
    // the frontiers should be sorted based on distance
    // minimum distance should be popped first
    private frontiers: PositionWithDistance[] = [];

    addFrontier(frontier: PositionWithDistance, endPosition: Position) {
        frontier.distance = Math.hypot(frontier.x - endPosition.x, frontier.y - endPosition.y);
        this.frontiers.push(frontier);
        this.frontiers.sort((a, b) => a.distance - b.distance);
    }

    popFrontier() {
        return this.frontiers.shift();
    }

    isEmpty() {
        return this.frontiers.length === 0;
```

```
31      }
32  }
33
34  export class RobotController {
35
36      private robot: Robot;
37      private canvas?: Canvas;
38      private discoveredMap: number[][] = [];
39      private lidarRange = 5;
40      private robotTravelMap: MapBuilderInstance;
41
42      constructor(robot: Robot, robotTravelMap: MapBuilderInstance, canvas?:
    Canvas) {
43          this.robot = robot;
44          this.canvas = canvas;
45          this.robotTravelMap = robotTravelMap;
46          this.discoveredMap = Array.from({ length: 100 }, () => Array.from({
     length: 100 }, () => 1));
47      }
48
49      private async moveAlongPath(path: Position[]) {
50          for (let i = 1; i < path.length; i += 1) {
51              const position = path[i];
52              const { direction, distance } = this.
    getDirectionCommandToTravel(position);
53              await this.moveRobot(direction, distance);
54          }
55      }
56
57      public changeRobotSpeed(speed: number) {
58          this.robot.setSpeed(speed);
59      }
60
61      private getLatestRobotPosition(): Position {
62          const position = this.robotTravelMap.getCurrentPosition()
63          return convertPositionToInt(position);
64      }
65
66      async handleOcclusionInKnownEnvironment(targetPosition: Position, map:
    number[][]) {
67          const startPosition = convertPositionToInt(this.
    getLatestRobotPosition());
68          const endPosition = convertPositionToInt(targetPosition);
69
70          const path = AstarPathPlanner.findPath(startPosition, endPosition,
    map).slice(0, -5);
71          this.drawAStarPath(path);
72          await this.moveAlongPath(path);
73      }
74
75      async handleOcclusionInUnknownEnvironment(targetPosition: Position) {
76          const endPosition = convertPositionToInt(targetPosition);
77          const frontierDistanceThreshold = 3; // Set a distance threshold
    for filtering out similar >frontiers
78          let path: Position[] = [];
79          path = AstarPathPlanner.findPath(convertPositionToInt(this.
    getLatestRobotPosition()), endPosition, this.discoveredMap);
80          if (path.length === 0) {
```

```
            console.log("No path found");
        } else {
            // draw path
            this.drawAStarPath(path);
            // move robot along the path
            await this.moveAlongPath(path);
            return;
        }
        // Perform initial scan and update the map
        await this.updateDiscoveredMap();
        const frontierPriorityQueue = new FrontiersPriorityQueue();
        const frontiers = this.identifyFrontiers();
        // Sort the frontiers based on distance
        frontiers.forEach(frontier => {
            frontierPriorityQueue.addFrontier({ ...frontier, distance: 0 },
 endPosition);
        });

        do {
            const frontier = frontierPriorityQueue.popFrontier();
            if (!frontier) {
                break;
            }
            if (this.canMoveToPosition(frontier)) {
                await this.moveToFrontierInDiscoveredMap(frontier);
                this.updateDiscoveredMap();
                const frontiers = this.identifyFrontiers();
                // draw filtered frontiers
                frontiers.forEach(({ x, y }) => {
                    ExploredMapCanvasImpl!.drawRectangle(`filtered-frontier
-${x}-${y}`, x, y, 1, 1, "blue", "blue", 1);
                });
                // Sort the frontiers based on distance
                frontiers.forEach(frontier => {
                    frontierPriorityQueue.addFrontier({ ...frontier,
distance: 0 }, endPosition);
                });
            }
            if (this.isPositionsNear(frontier, endPosition,
frontierDistanceThreshold)) {
                break;
            }
        } while (!frontierPriorityQueue.isEmpty());
        // Perform A* path planning
        path = AstarPathPlanner.findPath(convertPositionToInt(this.
getLatestRobotPosition()), endPosition, this.discoveredMap);
        if (path.length === 0) {
            console.log("No path found");
        } else {
            // draw path
            this.drawAStarPath(path);
            // move robot along the path
            await this.moveAlongPath(path);
        }
    }

    private drawAStarPath(path: Position[]) {
        if (globalConfigsProvider.getConfig("showAstarPath") && this.canvas
```

```
    ) {
134             this.canvas.removeSimilarObjects("a*path");
135             path.slice(0, path.length - 5).forEach((pos, idx) => {
136                 this.canvas!.drawRectangle("a*path" + idx, pos.x, pos.y,
    0.5, 0.5, "red", "red", 0.5);
137             });
138         }
139     }
140 
141     private isPositionsNear(position1: Position, position2: Position,
    comparingDistanceRange: number): boolean {
142         const distance = Math.hypot(position1.x - position2.x, position1.y
    - position2.y);
143         return distance <= comparingDistanceRange;
144     }
145 
146     private identifyFrontiers(): Position[] {
147         const frontiers: Position[] = [];
148         for (let y = 0; y < this.discoveredMap.length; y++) {
149             for (let x = 0; x < this.discoveredMap[y].length; x++) {
150                 if (this.discoveredMap[y][x] === 1) { // If cell is unknown
151                     const neighbors = this.getNeighbors(x, y);
152                     if (neighbors.some(([nx, ny]) => this.discoveredMap[ny
    ][nx] === 0)) {
153                         frontiers.push({ x: x, y: y });
154                     }
155                 }
156             }
157         }
158         const reachableFrontiers = frontiers.filter(frontier => this.
    canMoveToPosition(frontier));
159         return reachableFrontiers;
160     }
161 
162     private getNeighbors(x: number, y: number): [number, number][] {
163         const neighbors: [number, number][] = [];
164         if (x > 0) neighbors.push([x - 1, y]);
165         if (x < this.discoveredMap[0].length - 1) neighbors.push([x + 1, y
    ]);
166         if (y > 0) neighbors.push([x, y - 1]);
167         if (y < this.discoveredMap.length - 1) neighbors.push([x, y + 1]);
168         return neighbors;
169     }
170 
171     async moveToFrontierInDiscoveredMap(destination: Position) {
172         const map = this.discoveredMap;
173         this.discoveredMap[destination.y][destination.x] = 0; // Transposed
174 
175         const path = AstarPathPlanner.findPath({ x: this.robot.x, y: this.
    robot.y }, destination, map);
176         this.drawAStarPath(path);
177         await this.moveAlongPath(path);
178     }
179 
180     canMoveToPosition(position: Position): boolean {
181         const { direction, distance } = this.getDirectionCommandToTravel(
    position);
182         return this.robot.canMove(direction, distance);
```

```
    }

    private getDirectionCommandToTravel(position: Position): { direction:
    Direction, distance: number } {
        // get the average position
        let robotPosition = { x: this.robot.x, y: this.robot.y };

        const dx = position.x - robotPosition.x;
        const dy = position.y - robotPosition.y;

        let direction: Direction = "up";
        let distance = 1;

        if (Math.abs(dx) > Math.abs(dy)) {
            if (dx > 0) {
                direction = "right";
                distance = Math.floor(dx);
            } else {
                direction = "left";
                distance = Math.floor(-dx);
            }
        } else {
            if (dy > 0) {
                direction = "down";
                distance = Math.floor(dy);
            } else {
                direction = "up";
                distance = Math.floor(-dy);
            }
        }


        return { direction, distance };
    }

    private async updateDiscoveredMap() {
        const lidarReading = this.robot.getLidarReading(this.lidarRange);
        const position = { x: this.robot.x, y: this.robot.y };

        const x = Math.floor(position.x);
        const y = Math.floor(position.y);
        this.discoveredMap[y][x] = 0; // Note the transposition here

        const newPositions: Position[] = [{ x, y }];

        // lidar output as this { up: 1, down: 4, left: 3, right: 0 }
        // right 0 means no obstacle detected
        // up 1 means 1 unit distance to obstacle
        // left 3 means 3 unit distance to obstacle

        if (lidarReading.up > 0) {
            for (let i = 1; i <= lidarReading.up; i++) {
                if (y - i >= 0) {
                    this.discoveredMap[y - i][x] = 0; // Transposed
                    newPositions.push({ x, y: y - i });
                }
            }
        }
```

```
240
241         if (lidarReading.down > 0) {
242             for (let i = 1; i <= lidarReading.down; i++) {
243                 if (y + i < 100) {
244                     this.discoveredMap[y + i][x] = 0; // Transposed
245                     newPositions.push({ x, y: y + i });
246                 }
247             }
248         }
249
250         if (lidarReading.left > 0) {
251             for (let i = 1; i <= lidarReading.left; i++) {
252                 if (x - i >= 0) {
253                     this.discoveredMap[y][x - i] = 0; // Transposed
254                     newPositions.push({ x: x - i, y });
255                 }
256             }
257         }
258
259         if (lidarReading.right > 0) {
260             for (let i = 1; i <= lidarReading.right; i++) {
261                 if (x + i < 100) {
262                     this.discoveredMap[y][x + i] = 0; // Transposed
263                     newPositions.push({ x: x + i, y });
264                 }
265             }
266         }
267
268         if (ExploredMapCanvasImpl) {
269             for (let i = 0; i < newPositions.length; i++) {
270                 const { x, y } = newPositions[i];
271                 ExploredMapCanvasImpl!.drawRectangle(`discovered-map-${x}-${y}`, x, y, 1, 1, "gray", "gray", 1); // Transposed
272             }
273         }
274     }
275
276     moveRobot(direction: Direction, distance?: number) {
277         return this.robot.move(direction, distance || 1);
278     }
279 }
```

D: .1.4 A* Path Planner

```
1  import { Position } from "../types";
2
3  class PathNode {
4      constructor(
5          public position: Position,
6          public parent: PathNode | null,
7          public g: number,
8          public h: number,
9          public f: number
10     ) { }
11 }
12
13 const heuristic = (pos0: Position, pos1: Position): number => {
```

```
14      return Math.abs(pos0.x - pos1.x) + Math.abs(pos0.y - pos1.y); //
    Manhattan distance
15  };
16
17  export class AstarPathPlanner {
18      public static findPath(start: Position, end: Position, grid: number
    [][]): Position[] {
19          const openList: PathNode[] = [];
20          const closedList: Set<string> = new Set();
21          const startNode = new PathNode(start, null, 0, 0, 0);
22          const endNode = new PathNode(end, null, 0, 0, 0);
23          startNode.h = heuristic(start, end);
24          startNode.f = startNode.g + startNode.h;
25          openList.push(startNode);
26          const getNeighbors = (node: PathNode): PathNode[] => {
27              const { x, y } = node.position;
28              const neighbors: PathNode[] = [];
29              const directions = [
30                  [0, -1], // left
31                  [0, 1],  // right
32                  [-1, 0], // up
33                  [1, 0] // down
34              ];
35              for (const [dx, dy] of directions) {
36                  const nx = x + dx;
37                  const ny = y + dy;
38                  if (nx >= 0 && ny >= 0 && ny < grid.length && nx < grid[0].
    length && grid[ny][nx] === 0) {
39                      neighbors.push(new PathNode({ x: nx, y: ny }, node, 0,
    0, 0));
40                  }
41              }
42              return neighbors;
43          };
44          const reconstructPath = (endNode: PathNode): Position[] => {
45              const path: Position[] = [];
46              let currentNode: PathNode | null = endNode;
47              while (currentNode !== null) {
48                  path.push(currentNode.position);
49                  currentNode = currentNode.parent;
50              }
51              return path.reverse();
52          };
53          while (openList.length > 0) {
54              let currentNode = openList.reduce((prev, curr) => (prev.f <
    curr.f ? prev : curr));
55              if (currentNode.position.x === endNode.position.x &&
    currentNode.position.y === endNode.position.y) {
56                  return reconstructPath(currentNode);
57              }
58              openList.splice(openList.indexOf(currentNode), 1);
59              closedList.add(`${currentNode.position.x},${currentNode.
    position.y}`);
60              const neighbors = getNeighbors(currentNode);
61              for (const neighbor of neighbors) {
62                  if (closedList.has(`${neighbor.position.x},${neighbor.
    position.y}`)) {
63                      continue;
```

```
64                  }
65                  neighbor.g = currentNode.g + 1;
66                  neighbor.h = heuristic(neighbor.position, endNode.position)
    ;
67                  neighbor.f = neighbor.g + neighbor.h;
68                  const existingNode = openList.find(node => node.position.x
    === neighbor.position.x
69                      && node.position.y === neighbor.position.y);
70                  if (!existingNode || neighbor.g < existingNode.g) {
71                      openList.push(neighbor);
72                  }
73              }
74          }
75          return []; // No path found
76      }
77  }
```

E: FUNCTIONAL TESTING

E: .1 Hardware Specs of Simulator PC

This hardware setup is used for all the simulation processings.

- **Device**: MacBook Pro (13-inch, 2020)
- **Processor**: 2 GHz Quad-Core Intel Core i5
- **Graphics**: Intel Iris Plus Graphics 1536MB
- **Memory**: 16GB 3733 MHz LPDDR4X
- **Operating System:**: macOS Sonoma 14.0

E: .2 Simulator Setup

The following simulator setup is used in both accuracy and performance testing.

- **Environment**: Simulated 2D space where human following robots navigate.
- **Obstacles**: Static obstacles are included.
- **Measurement**: Runtime of the occlusion recovery algorithm in seconds.
- **Map EMA Alpha**: 0.09
- **Robot Speed**: 10 (Unit is 1 square per second)
- **UWB Error Rate**: 5%

E: .3 Performance Testing

E: .3.1 Test Initialization

- Set up a 2D simulated environment with predefined dimensions (100X100).
- Place static obstacles at fixed positions within the environment.

E: .3.2 Test Run

- For each test configuration, run the occlusion recovery algorithm multiple 10 times to obtain consistent measurements.
- Record the average runtime for each configuration.

E: .3.3 Measurement

- Use a timer function to measure the runtime of the occlusion recovery algorithm (Occlusion Recovery when Envioronment is unknown) for each iteration.
- Calculate the average of runtime for each configuration.

E: .4 Accuracy Testing

Accuracy of the system defines how accurately it can resolve an occlusion in a given scenario. The robot should move to a place where it can visually see the subject when the occlusion recovery is done. A key factor determining this accuracy is the accuracy of the localization system. The more accurate the localization system, the more precise the algorithm's estimations. While UWB is one of the most accurate localization systems available, it is still prone to errors, typically producing outputs with an error margin of 10-30 centimeters.

The test is carried out to assess the accuracy of the algorithm and the time taken to estimate a successful route, considering the error margin of UWB readings. The experiment setup is described in Appendix E: Functional Testing.

E: .4.1 Simulator Setup

The following simulator setup is used for accuracy testing.

- **Environment**: Simulated 2D space where human following robots navigate.
- **Obstacles**: Fixed number of Static obstacles are included.
- **UWB Error Rate**: Varying

E: .4.2 Test Run

- For each test configuration, run the occlusion recovery algorithm multiple times (10 iterations) to obtain consistent measurements.
- Introduce varying UWB error margins (e.g.,5%, 8%, 10%) in the localization data.
- Record the estimated time taken for the robot to resolve the occlusion and visually reacquire the subject for each iteration.

E: .4.3 Measurement

- Use a timer function to measure the time taken for the occlusion recovery algorithm to estimate a successful route for each iteration.
- Calculate the average time taken for each configuration.
- Evaluate the accuracy of the system by measuring the distance between the estimated and actual positions of the human subject after occlusion recovery.

F: EVALUATION

Evaluator	Opinion
Evaluator 1 (Domain) Mr. Kalana Mihirange PHD Candidate (Computer Vision) - Western Sydney University B.Sc Honors Computer Science - University of Colombo Former Software Engineer at WS02 Research Interests: UAV, Computer Vision, Robotic Systems	”The concept is quite good and novel and addresses a problem that haven’t got enough attention in literature. Yeah this is significant challenge in autonomous navigation, particularly in occluded environments. This simulator program is good to demonstrate the project idea and the concept. Study closely how several drone systems achieve human following capabilities like in DJI. Try to combine some other movement sensors to get more accurate localization while reducing the number of anchor points needed. Consider exploring machine learning techniques to further enhance path planning and obstacle detection capabilities. Implementation is efficient, Consider building a topological map from LiDAR based exploration.”

Evaluator 2 (Technical) Mr. Pasindu Dewapriya Senior Software Engineer at Yaala Labs B.Sc Honors Computer Science - University of Colombo Research Interests: IoT Sytems, Machine Learning, Embedded Systems, Autonomous Navigation	”A well-thought-out approach that combines multiple cutting-edge technologies effectively. Minimal design with focus on the requirements are really what we need. This project has great potential for practical applications. Depth of analysis is commendable, though future iterations should include more complex environments to fully assess system robustness. The system design and architecture are adhere to best practices in the industry. Better we could try use some reinforcement learning on handling the occlusion. Accuracy and error handling in the simulation is good but when it comes to implementation those accumulated errors need to be addressed in different ways. Can’t we use D* or D*-Lite algorithm when dealing with unknown environment case.”

Table F: .1: Evaluation Result from Experts