THE HASHEMITE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

SENIOR DESIGN PROJECT-(II)

Indoor Facility Assistant Robot

STUDENTS

Dana Jihad Shaqdih	1931939
Islam Azmi Alshourman	1935680
Maryam Mohammed Shehadeh	1933756
Tasneem Eyad Barakat	2038234

Senior Design Project Advisor
Prof. Bassam Jamil

DEPARTMENT OF COMPUTER ENGINEERING

Academic Year
2023-2024

ACKNOWLEDGMENT

Firstly, we express our gratitude to Allah for providing us with the strength and dedication needed to complete this project. We extend our sincere thanks to our mentor, Dr. Bassam Jamil, for his invaluable guidance, motivation, and support throughout our journey. We also acknowledge Dr. Khalil Yousef for his advice and assistance during the challenging times. Lastly, we are deeply thankful to our parents, family, friends, and supporters for their belief in us and their continuous encouragement.

ABSTRACT

In recent years, autonomous robotics have seen significant advancements, finding diverse applications in fields such as security, navigation, and education. Our project involves the development of a sophisticated robot system designed for the faculty of engineering, integrating multiple cutting-edge technologies. The robot serves a dual purpose: ensuring campus safety.

The Peoplebot includes important parts such as a strong obstacle detection system, an effective pathfinding algorithm, and a high-level security module. The obstacle detection system uses laser and sonar sensors to autonomously maneuver around the campus, avoiding both static and moving obstacles in real-time. The pathfinding component combines A* and reinforcement learning to map out the best routes to specific destinations. Lastly, the security module uses cameras to detect people and recognize faces for identity verification, allowing authorized access to limited areas.

Students get practical experience with cutting-edge robotics, AI, and computer vision technologies, readying them for future jobs in these areas. Early assessments show that the system is very effective and useful.

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

This project focuses on advancing the capabilities of the Peoplebot robot in navigation, obstacle detection, and user interaction within a building environment. By using A* and reinforcement learning (RL) algorithms, the robot achieves optimal pathfinding, allowing it to navigate complex environments with efficiency and accuracy. The robot is equipped with laser and sonar sensors, which provide accurate obstacle detection and avoidance, ensuring safe operation even in dynamic settings. Additionally, the robot includes people recognition technology and a security system, enhancing its ability to guide students, deliver items, and perform monitoring tasks. Initially tested within a lab environment, the Peoplebot has successfully demonstrated its multipurpose capabilities. Future plans include expanding the robot's operational scope to cover the entire engineering faculty, improving its capacity to navigate and interact in more extensive and diverse environments. This progression highlights the robot's potential for practical applications in various sectors, paving the way for its deployment in industries that require advanced autonomous navigation, reliable interaction capabilities, and robust security features.

Keywords: Peoplebot (Autonomous Robotics), Obstacle Detection, Optimal path (Pathfinding), Reinforcement Learning (RL), People Detection, Face Recognition, Campus Security, Educational Tool.

1 CHAPTER 1: INTRODUCTION

1.1 Problem Statement and Purpose

In our modern world, marked by rapid technological progress, there is a growing need for autonomous robotic systems capable of interacting with their environment meaningfully and effectively.

One of the major challenges these systems face is the ability to interact and respond to dynamic environments and user needs. This challenge is particularly relevant to many sectors, including logistics, customer service, healthcare, and security. For example, in the logistics and warehousing sector, efficient product delivery and guided navigation are essential to streamline operations.

In customer service, interactive guidance can improve the user experience at large facilities such as shopping malls, colleges or airports. In healthcare environments, robots can assist healthcare professionals by ensuring timely supplies and patient guidance. Additionally, when it comes to security, facial recognition and intruder detection through advanced monitoring are crucial for maintaining security. As a result, the need for powerful and flexible robotic solutions capable of performing delivery, guidance and security tasks is constantly increasing.

Our goal is to meet the growing demand for advanced robotic systems that interact autonomously with the environment. Introduce “Peoplebot”, a

multi-functional robotic designed to integrate advanced machine learning and image processing technologies.

Peoplebot's overarching goal is to automate delivery, serve as a user guide, and improve security through facial recognition and streaming video to detect humans. By achieving these goals, Peoplebot aims to improve the functionality and flexibility of robotic systems. Consequently, this opens the door for new applications in many fields.

The Peoplebot project demonstrates the idea that robots with the ability to deliver goods, guide users, and improve safety can significantly contribute to automation, efficiency, and problem-solving in different fields. It demonstrates the potential for integrating advanced technologies in the field of robotics and paves the way for a future in which intelligent robotic platforms will play a central role in various industries.

1.2 Project and Design Objectives

This project addresses many challenges in robotics, focusing on the development of an advanced system capable of sophisticated environmental interaction. The primary objective is to *use the* Peoplebot robot that excels in guiding students, delivering items, and performing security roles within various settings.

Our project transcends conventional robotic capabilities by integrating pathfinding algorithms with machine learning techniques. This combination enables the Peoplebot to navigate the environment, guide students to specific destinations, and perform security tasks.

The practical applications of our project span multiple industries.
In educational environments, the Peoplebot will guide students to specific locations, such as classrooms or administrative offices. In logistics, it will autonomously deliver items between users, improving efficiency and convenience. Additionally, the robot's security functions will include monitoring and reporting on its surroundings, ensuring a safe environment.

This initiative also explores the Peoplebot's potential in research and development. For example, its ability to navigate and extract data from challenging environments, such as archaeological sites, opens new avenues

for exploration and discovery.

In addition to its operational capabilities, the project emphasizes user-friendliness, security, and adherence to high safety. The development process incorporates rigorous design and testing phases to ensure that the robot is not only effective but also reliable and safe to operate in diverse environments.

In summary, the Peoplebot is a pioneering step in advanced robotics. Combining machine learning and image processing in an innovative robotic form, the project seeks to set new benchmarks in automation, serving as a guide, delivery assistant, and security tool, while opening new frontiers for future advancements in the field.

1.3 Intended Outcomes and Deliverables

Intended Outcomes

1. Navigation and Safety: The robot should effectively navigate the campus using A* and reinforcement learning algorithms, avoiding obstacles detected by laser and sonar sensors to ensure smooth and safe operation.

2. Enhanced Security: By using cameras for people detection and face recognition, the robot ensures that individuals on campus are authorized, matching IDs with faces accurately.

3. Campus Monitoring: The robot should assist in monitoring campus activities, providing real-time updates and alerts for any security breaches or unusual activities.

4. People counting: This feature is essential for managing events and gatherings, attendance tracking and crowd management, and contributing to overall campus safety and efficiency.

5. Automation of Routine Tasks: The robot should handle routine tasks such as delivering messages, documents, or small packages across campus, reducing the workload on staff.

6. Data Collection and Analysis: The system should gather data on movement patterns, security incidents.

7. User Interaction and Assistance: The robot should interact with

students and teachers, providing information, guiding .

Deliverables

1. A robot equipped with laser and sonar sensors, cameras, and capable of executing A* and reinforcement learning algorithms for navigation and security tasks.

2. Navigation and Obstacle Detection System: real-time obstacle detection and path planning, ensuring the robot navigates the campus effectively.

3. Face Recognition and ID Verification System: A reliable system for detecting people and verifying their identity by matching their face with their ID photos.

4. People Recognition System: An advanced system utilizing algorithms to detect and count people in various campus environments.

5. Data Analytics and Reporting: analyzing the data collected by the robot, providing insights into security.

Additional Applications

1. Healthcare Facilities: In hospitals or clinics, the robot can navigate through wards and departments, delivering medications, documents, or equipment. It can also verify personal identities to ensure secure

access to restricted areas.

2. Corporate Environments: In office buildings, the robot can assist with internal deliveries, visitor guidance, and employee identification for enhanced security.

3. Retail and Shopping Centers: The robot can help with customer service by guiding customers to products, providing information, and ensuring security by monitoring unauthorized access.

4. Event Management: At conferences, exhibitions, or large events, the robot can guide attendees, provide information, and ensure only authorized individuals access specific areas.

5. Libraries and Archives: The robot can assist in managing book deliveries, guiding visitors, and ensuring the security of restricted sections.

6. Transportation Hubs: In airports, train stations, or bus terminals, the robot can assist passengers with directions, provide information, and enhance security through identity verification.

7. Educational Institutions: Beyond the faculty of engineering, the robot can be deployed across different departments, assisting in various administrative tasks, enhancing campus security, and providing guidance and information to students and visitors.

8. Manufacturing and Warehousing: The robot can navigate complex environments, transporting materials and verifying personal identities to enhance security and operational efficiency.

The intended outcomes and deliverables of the robot system for the faculty of engineering encompass navigation, enhanced security, efficient campus monitoring, and automation of routine tasks. With its advanced features, this robot can be adapted for various applications in healthcare, corporate environments, retail, event management, libraries, transportation hubs, educational institutions, and manufacturing, offering significant benefits in terms of efficiency, security, and user assistance.

Additional Applications in the Engineering Faculty and University

1. Laboratory Assistance:
    - Equipment Transport: The robot can transport sensitive equipment and materials between labs, ensuring timely and secure delivery.
    - Sample Collection: It can collect and deliver samples or specimens for analysis, reducing the time researchers spend on these tasks.
2. Lecture Assistance:
    - Lecture Material Delivery: The robot can deliver lecture notes, handouts, and other materials to classrooms, ensuring that professors and students receive necessary resources on

time.

- Audio-Visual Support: It can assist in setting up and
- troubleshooting audio-visual equipment in lecture halls.

3. Library Services:

    - Book Delivery: The robot can fetch and deliver books or other materials to and from the library to faculty offices or student dorms.
    - Inventory Management: It can help in managing library inventory by scanning and updating the database in real-time.

4. Campus Tours and Orientation:

    - Guided Tours: The robot can provide guided tours for new students, visitors, and prospective students, offering information about different facilities and departments.
    - Interactive Maps: It can display interactive maps and help users find their way around the campus.

5. Administrative Support:

    - Document Delivery: The robot can handle inter-departmental document delivery, reducing the need for staff to move around the campus for administrative tasks.
    - Meeting Assistance: It can help set up rooms for meetings and deliver materials or refreshments.

6. Maintenance and Inspection:

    - Facility Inspection: Equipped with cameras and sensors, the robot can inspect facilities for maintenance issues, such as leaks, damages, or security breaches.

    - Environment Monitoring: It can monitor environmental conditions like temperature, humidity, and air quality in labs and classrooms.

7. Student Assistance:

    - Study Assistance: The robot can help students by providing information on available study resources, lab schedules, and upcoming deadlines.

    - Lost and Found: It can manage a lost and found service, helping students and staff retrieve lost items.

8. Health and Safety:

    - Emergency Response: The robot can assist during emergencies by guiding people to safety, providing first aid supplies, and communicating with emergency services.

    - Sanitization: It can perform routine sanitization of common areas, especially important in maintaining hygiene standards.

9. Event Support:

    - Event Setup: The robot can assist in setting up events,

including arranging seating, counting guests, delivering materials, and providing information to attendees.

- Registration Assistance: It can help with attendee registration, checking IDs, and directing people to event locations.

10. Research and Development:

- Data Collection: The robot can collect data from various sensors placed around the campus for research purposes, providing valuable insights for engineering projects.

- Prototyping and Testing: It can serve as a platform for testing new technologies developed by students and faculty, such as new sensors, navigation algorithms, or AI applications.

11. Dining Services:

- Food Delivery: The robot can deliver meals or snacks to faculty offices, student lounges, or study areas, enhancing convenience.

- Queue Management: It can manage queues in cafeterias by taking orders and informing staff about the demand.

12. Security :

- Surveillance: The robot can patrol the campus, providing live video feed to security personnel and detecting unusual activities.

- Access Control: It can assist in managing access to restricted areas, ensuring that only authorized individuals enter.

Deploying the robot across various facets of the engineering faculty and university can significantly enhance efficiency, security, and convenience. From assisting in laboratories and libraries to supporting administrative tasks and enhancing campus safety, the robot's applications are vast and versatile, contributing to a more efficient and secure academic environment.

1.4 Summary of the Used Design Process

1. Needs Assessment and Requirement Gathering:
    - Use Case Analysis: obstacle detection, navigation, people detection, face recognition, and task automation
2. System Design:
    - Hardware Design: using laser and sonar sensors, cameras, and mobility mechanisms.
    - Software Architecture: Design the software architecture to integrate navigation algorithms (A* and reinforcement learning), obstacle detection, face recognition, and user interface.
3. Prototyping:
    - Initial Prototyping: test basic functionalities and validate design.
    - Iterative Refinement: Improve the prototype through iterative testing and feedback, addressing issues and enhancing performance.
4. Algorithm Development:
    - Navigation and Obstacle Detection: test A* and reinforcement

learning algorithms to ensure efficient and safe navigation.

- Face Recognition: Implement and train face recognition algorithms, ensuring accuracy in matching faces to the ID.
- Integration: Integrate these algorithms to operate on the robot.

5. Testing and Validation:
    - Functional Testing: testing of functionalities including navigation, obstacle detection, and face recognition .

1. Evaluation and Optimization:

    Optimization: Make necessary adjustments and improvements based on collected data to enhance overall functionality.

2. Documentation and Maintenance:

    Comprehensive Documentation: Prepare detailed documentation covering the robot's design.

1.5 Summary of Report Structure

This report is organized into eight chapters, starting with the Introduction, which outlines the project's purpose, objectives, and design process. Chapter 2 summarizes achievements from the senior design project, including initial designs and cost estimates. Chapter 3 updates background theory with relevant literature and ethical considerations. Chapter 4 details the design specifications, alternatives, and final design. Chapter 5 focuses on project realization, performance optimization, testing, and safety. Chapter 6 discusses economic, ethical, and environmental issues, including cost analysis and regional relevance. Chapter 7 covers project management, tasks, schedules, resources, and lessons learned. Chapter 8 concludes with a restatement of objectives, a summary of achievements, new skills learned, future steps, and final remarks. The report is supplemented with references and appendices for additional details and supporting materials.

2 CHAPTER 2: Summary of achievements in senior design project-I

2.1 Proposed Preliminary Design

2.1.1 General Constraints

Technical Restrictions

Hardware Compatibility: Ensuring seamless integration of FPGA, AXIS 214 PTZ camera, and PeopleBot, focusing on processor power, data transmission speeds, and interface compatibility.

Operational Range and Battery Life: Maximizing operational range by optimizing battery capacity and power management, and tuning the camera's pan, tilt, and zoom for efficient surveillance.

Processing Power and Speed: Providing sufficient processing capacity for real-time image processing, autonomous decision-making, and machine learning tasks.

Sensor Limitations: Enhancing navigation and interaction capabilities through sensitive and wide-range sensors, considering environmental complexities.

Environmental Elements

Adaptability to Different Conditions: Designing for indoor and outdoor use, including temperature variations and weather resilience.

Interaction with Diverse Materials: Ensuring reliable performance on various surfaces, around obstacles, and in different lighting conditions.

Durability and Maintenance: Building a robust external and internal structure for frequent operation, emphasizing ease of maintenance and repair.

By addressing these design constraints, the PeopleBot project aims to deliver a reliable, efficient, and versatile robot suitable for various environments and applications.

2.2 Selected Preliminary Design and Justification

2.2.1 Design alternatives

Variability of Components

Camera Options: Evaluating potential cost savings or performance gains from other camera types or brands in addition to the AXIS 214 PTZ. This can involve different characteristics like resolution, zoom capacity, and low-light performance to determine which one best meets the requirements of the PeopleBot.

Alternatives to Sensors: utilizing a variety of sensors from many producers to improve environmental interaction and navigation. To improve obstacle identification and environmental awareness, this involves taking into account variations in sonar, infrared, and other types of sensors like LIDAR or thermal imaging.

Options for Power Sources: Evaluating various battery or power system types. For example, looking into using solar panels as an additional power

source or investigating the usage of upgraded lithium-polymer batteries for longer life.

Alternative Software

Programming Language Flexibility: Taking into account a variety of programming languages for system development, including C++ for its performance efficiency or Python for its user-friendliness and extensive machine learning support. The system's performance, communication with other systems, and ease of development will all be impacted by the language selection.

Machine Learning Frameworks: Exploring different frameworks for autonomous decision-making and image processing. Depending on their suitability for real-time processing, support for FPGA integration, and the particular requirements of the PeopleBot's AI functions, alternatives to take into consideration might include TensorFlow, PyTorch, or OpenCV.

Options for Operating System: Evaluating various operating systems for the onboard computer, such Windows for its extensive support and user-friendliness or Linux for its robustness and customization. The choice will have an impact on system security, usability, and program compatibility.

The Peoplebot project may maximize performance, cost, and adaptability by studying these design options. This will guarantee that the finished product is not only technically sound but also commercially feasible and able to keep up with changing market demands and technological developments.

1. Alternatives for Routing Algorithms

Dijkstra's and A-Star in Finding the Shortest Path

Finding the shortest path between two points is a common problem in robotics, navigation, and routing. For solving this problem we have two well-known algorithms which are Dijkstra's algorithm and the A* algorithm which is an extension of Dijkstra.

Dijkstra's Algorithm:

The Dijkstra Algorithm works by exploring all possible paths from the start node, and then it selects the shortest path to carry on. This process continues until the shortest path to the target node is found. Dijkstra's algorithm guarantees finding the optimal path but can be time and power costly, especially for large graphs, due to its exhaustive nature. Its time complexity is $\Theta(n^2)$ when using a simple array and $\Theta((n + e) \log n)$ when using a priority queue, where **n** is the number of nodes in the graph and e is the number of edges in the graph.

A Star Algorithm:

This algorithm improves pathfinding by adding a heuristic that includes the remaining cost to the goal. This helps make the search process more efficient. In A*, the evaluation function combines the actual cost with this heuristic estimate, leading to faster pathfinding. Although A* doesn't always ensure the absolute shortest path when the heuristic isn't entirely accurate, it is usually faster than Dijkstra's and has a time complexity of $\Theta(m*n)$, where

m is the number of paths that can be taken from any given node, and n is the number of nodes in the graph. As a consequence, A* approach in finding the optimal path is less costly than Dijkstra knowing that both of them are very good algorithms to find the optimal path [5].

2. Reinforcement Learning

1. Classical Pathfinding Algorithms:

    - *A* Algorithm*: As already part of our system, A* is a powerful heuristic search algorithm used for finding the shortest path. It can be tuned further for improved performance in specific environments.

    - Dijkstra's Algorithm: Suitable for finding the shortest path in a graph with non-negative weights, Dijkstra's algorithm is less computationally intensive than A* but may be less efficient in complex environments.

2. Rule-Based Systems:

    - Finite State Machines (FSMs): FSMs use a set of predefined states and transitions to navigate the environment. They are straightforward to implement and maintain, making them suitable for simpler navigation tasks.

    - Behavior Trees: Used in game AI, behavior trees provide a modular and flexible way to control decision-making

processes. They can handle complex behaviors and are more scalable than FSMs.

3. Potential Field Methods:

    - Artificial Potential Fields: This approach uses attractive forces towards the goal and repulsive forces from obstacles to guide the robot. It's simple to implement but can suffer from local minima problems.

4. Graph-Based Methods:

    - Rapidly-exploring Random Trees (RRT): RRTs are used for path planning by randomly exploring the space and ensuring the robot finds a path to the goal. They are particularly useful in high-dimensional spaces.

    - Probabilistic Roadmaps (PRM): PRM involves creating a roadmap of the environment with nodes and edges, allowing the robot to navigate by following precomputed paths. This method is effective in complex environments.

5. Machine Learning Approaches:

    - Supervised Learning: Train a model using labeled data to predict the best path or action. This approach requires a large dataset of examples for effective learning.

- Imitation Learning: The robot learns by mimicking expert behavior, using demonstration data to learn navigation strategies. It combines elements of supervised learning and reinforcement learning.

6. Optimization-Based Methods:

    - Linear Programming (LP): LP can be used for optimizing the robot's path by defining constraints and an objective function. It is efficient but may not handle dynamic environments well.

    - Genetic Algorithms (GAs): GAs use evolutionary principles to find optimal paths. They are good at exploring large search spaces but can be computationally expensive.

Choosing the right alternative to reinforcement learning depends on the specific requirements and constraints of the robot's operating environment. Classical pathfinding algorithms like A* and Dijkstra's algorithms are robust and well-understood, making them reliable choices. For more dynamic or complex scenarios, methods like RRTs, PRMs, or machine learning approaches may offer better performance and flexibility.

3. Reasons to Choose Reinforcement Learning

1. Adaptability to Dynamic Environments:

    - Learning from Experience: Reinforcement learning (RL) allows the robot to learn from interactions with its

environment, enabling it to adapt to dynamic and unpredictable changes.

- Real-Time Decision Making: RL algorithms can continuously update their strategies based on new data, making them highly effective in environments where conditions frequently change.

2. Handling Complex and High-Dimensional Spaces:

    - Complex Navigation Tasks: RL is well-suited for environments with complex obstacle configurations and high-dimensional spaces, where traditional algorithms like A* or Dijkstra might struggle.

    - Scalability: RL methods can scale to handle more complex tasks and larger state-action spaces, making them ideal for sophisticated robotic applications.

3. Optimizing Long-Term Rewards:

    - Goal-Oriented Learning: RL focuses on maximizing cumulative rewards over time, which is particularly useful for tasks that require long-term planning and strategy.

    - Balancing Exploration and Exploitation: RL algorithms inherently balance the need to explore new strategies with the exploitation of known good strategies, leading to continuous improvement.

4. Learning Complex Behaviors:

    - Policy Learning: RL enables the robot to learn complex behaviors and policies that can be difficult to hard-code. This includes tasks like obstacle avoidance, path planning, and resource management.

    - Flexible Reward Structures: Designers can specify reward structures that encourage the robot to exhibit desired behaviors, such as minimizing travel time or avoiding collisions.

5. Reduced Need for Pre-Defined Rules:

    - Minimizing Human Intervention: RL reduces the need for manually defined rules and heuristics, allowing the robot to develop its own strategies based on environmental feedback.

    - Generalization: RL can generalize from specific experiences to new, unseen situations, improving the robot's ability to handle novel scenarios.

6. Integration with Other Learning Methods:

    - Hybrid Approaches: RL can be combined with other machine learning techniques, such as supervised learning or imitation learning, to leverage their strengths and enhance performance.

- Data Utilization: RL can make use of vast amounts of data collected from sensors and interactions, improving decision-making accuracy and efficiency.

7. Practical Applications and Benefits

- Autonomous Navigation: In environments like university campuses, RL can help the robot navigate complex terrains, avoid obstacles, and reach destinations efficiently.

- Personalization: The robot can learn and adapt to the specific preferences and behaviors of users, providing a more personalized interaction experience.

- Continuous Improvement: Over time, the robot can improve its performance and efficiency as it learns from its experiences, leading to better service delivery and user satisfaction.

Reinforcement learning offers significant advantages in terms of adaptability, complexity handling, and long-term optimization, making it a powerful choice for developing advanced robotic systems. Its ability to learn from experience and optimize behaviors in real-time makes it especially suitable for dynamic and complex environments, ensuring the robot can perform its tasks effectively and efficiently

2.3 Final Deliverables and Preliminary Cost

Final deliverables for the PeopleBot and Axis Camera project include documentation and hardware integrations showing the robot capabilities and applications can be made to the camera. We also consider the video showcasing the robot's capabilities and a final presentation summarizing the project's goals, methods, results, and potential future applications are also part of the deliverables.

The project was finished without any **Preliminary Cost** as the PeopleBot robot and Axis Camera were provided for free by our project supervisor and The Hashemite University. Additionally, all software needs were met using free resources and open-source code. The team used their personal laptops for data processing and work on the virtual environment. This allowed us to finish the project without obtaining any financial costs. However, for anyone else implementing the system, they should take into consideration the following expenses: The PeopleBot Mobile Platform Robot is priced at 60,000 JOD, with possible price variations. Similarly, the cost of an Axis Camera is 270 JOD, also with possible price variation.

3 CHAPTER 3: UPDATED BACKGROUND THEORY

3.1 Relevant Literature Search for GP-II

"Behavior of Delivery Robot in Human-Robot Collaborative Spaces During Navigation"

Navigation is a crucial skill for robots, especially in human-populated environments. Industry 5.0 emphasizes human-robot interaction, where robot behavior plays a key role in human acceptance, comfort, and safety. The "PI" robotic platform, equipped with proximity and vision sensors, uses the YOLO algorithm for real-time object and human detection during navigation.

Robots are expanding beyond structured environments like manufacturing to everyday settings such as hospitals, offices, homes, and hotels, where they serve as service robots. In hotels, delivering items to guest rooms is identified as a suitable application for robots, involving tasks like delivering towels, laundry, and food. Mobility in human-populated areas is critical for these service robots.

As robots increasingly enter consumer markets and share spaces with humans, it's important to study and implement socially acceptable behaviors. This includes maintaining appropriate social distances and acknowledging humans during navigation, ensuring smoother and more accepted human-robot interactions.

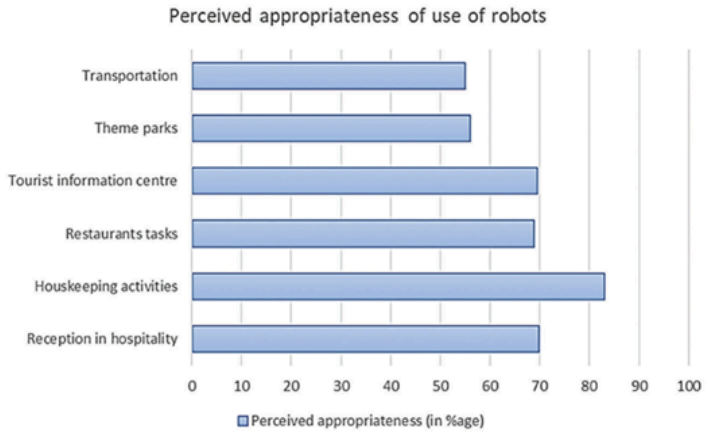


Figure 3-1

A robot “PI” acting as a delivery robot in a hotel corridor scenario with 36 participants to cater to the following objectives:

1. find the perception of a delivery robot during navigation tasks in a hotel setting by collecting responses from human participants for different robot behaviors.

2. To study the effect of audio and display capabilities of a robot on the perceived social presence of the robot.

3. To study the effect of different behavior conditions on the perceived role of a robot.

Mavrogiannis studied how different robot navigation strategies affect human behavior in a controlled lab environment. The study revealed that human acceleration decreases around autonomous robots compared to tele-operated ones.

In social robotics, vision algorithms are crucial for enhancing human-robot interactions. It is important to distinguish between human-robot interactions in industrial settings and social spaces, as they differ significantly. The hotel industry is likely to be an early adopter of this technology, making it essential to assess how robot behavior affects human acceptance and trust.

The robot discussed uses the YOLO algorithm to detect humans and obstacles during navigation. Its behavior is managed by a planning block, which executes different behavior conditions. The control block processes data from proximity and camera sensors and directs the actuators (motors, display, speakers) to perform specific behaviors. The robot navigates by incorporating proxemics constraints to ensure smooth interactions in corridor spaces.

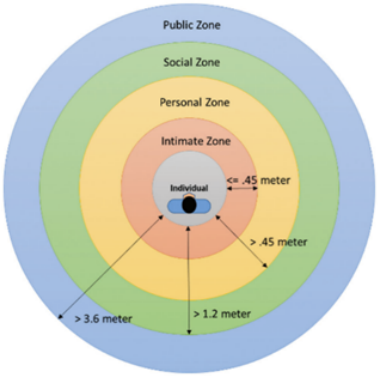


Figure 3.2 Classification of personal space [30]

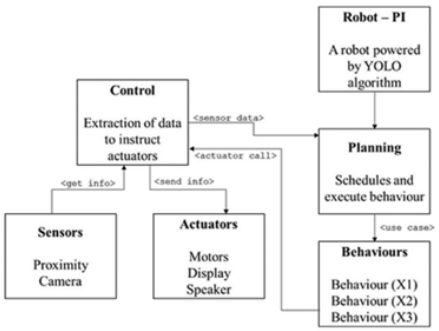


Figure 3.3 Complete system overview and approach

PI is a 150 cm tall delivery robot equipped with a 7” LCD IPS screen and inbuilt speakers, allowing it to use graphical and audio aids. It has a four-wheeled platform for mobility. PI is used to study the impact of different behavior conditions during navigation on humans. The robot employs the YOLO network for object detection, trained using the error back propagation algorithm. This method calculates the error between predicted data (PD) and ground truth (GT) using a loss function, and adjusts the network parameters to minimize the error.

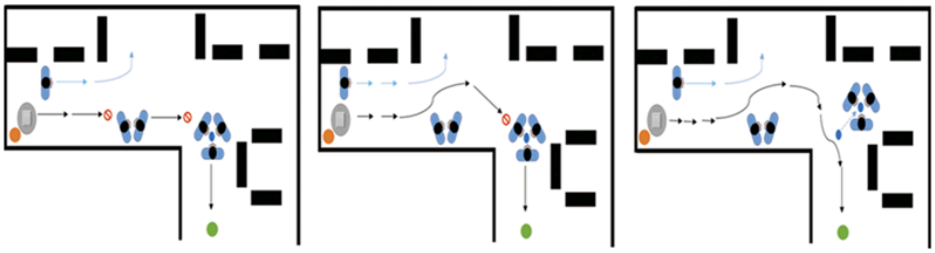


Figure 3.4 (a) Behavior condition X1 (B) Behavior condition X2 (C) Behavior condition X3

Figure 3.5 Object detection results from real-time execution of YOLO v3 algorithm

In this study, a delivery robot named "PI" was tested in different behavior conditions (X1, X2, X3) to understand its navigation in a hotel setting and how it is perceived by humans.

Behavior Conditions:

X1: PI moved at a constant speed, stopped when encountering blockages, and waited for the path to clear.

X2: PI moved alongside participants at the same speed, found alternate routes when partially blocked, and waited when completely blocked.

X3: PI moved at a slower speed, navigated around partial blockages, and used audio and visual signals to request passage when fully blocked.

Figure 3.6 Display and audio status for different conditions

Condition	State	Display graphics	Audio message
X1	Not applicable	Not applicable	Not applicable
X2	Not applicable	Not applicable	Not applicable
X3	Asking participants to pass if the path is completely blocked		Excuse me, can I pass?
X3	Thanking participants when they clear the path		Thank you!

Figure 3.7 Behaviour conditions

Condition	Movement	Obstacle	Display and audio
X1	• Moves at a constant speed	• Stop and waits for participants to clear the path if finds path blockage	• Display: OFF • Audio: OFF
X2	• Moves at the same speed as participants • Moves along with participants if moving in the same direction	• Performs obstacle avoidance if it is way-out • Stop and waits for participants to clear if the path is completely blocked	• Display: OFF • Audio: OFF
X3	• Moves at a slower speed as compared to participants • Follow the participants if moving in the same direction	• Performs obstacle avoidance if it is way-out • Ask participants to pass if the path is completely blocked	• Display: Show graphics • Audio: Turned on during asking and thanking process

Results:

- Participants rated the robot's behavior using the Godspeed questionnaire, evaluating likeability, perceived intelligence, and safety.

- The MANOVA test indicated significant differences in perceptions based on behavior conditions, with X3 scoring highest across all parameters.

- Participants preferred the robot that moved slower, used audio/visual aids, and exhibited social behaviors.
- Role perception varied, with PI seen as a machine in X1, a companion in X2, and an assistant in X3.
- Preferred robot behaviors included maintaining an appropriate distance from humans, approach speed, and body orientation.

Participants favored the third behavior condition (X3) where the robot demonstrated social behaviors such as asking for space and displaying emotions. This suggests that socially interactive robots are more acceptable in human-populated environments.

"Personal Care Robots"

Personal care robots are emerging as integral components in various sectors, significantly improving the quality of life by assisting with daily activities. These robots, unlike their industrial counterparts, are designed to interact directly with humans, often in unpredictable environments. This document explores the ISO standards that govern the safety and functionality of personal care robots, focusing primarily on ISO 13482.

Categories of Personal Care Robots

ISO 13482 classifies personal care robots into three main categories:

1. **Mobile Servant Robots**: These robots perform tasks like handling objects or exchanging information while moving around.
2. **Physical Assistant Robots**: These assist humans in performing physical tasks, enhancing the user's capabilities.
3. **Person Carrier Robots**: These transport humans to specific destinations.

Key Functions

Personal care robots are designed to:

- Navigate homes or public buildings without colliding with obstacles.

- Interact with humans, including exchanging objects.
- Handle objects of various sizes.
- Assist with mobility and reduce physical load for humans.

Market Growth

The personal care robots' market is expected to grow at a rate of 19% annually, reaching $10 billion in sales by 2024. This growth is driven by advancements in robotics technology and increasing consumer familiarity and comfort with such robots.

Safety Standards

Personal care robots must adhere to stringent safety standards due to their interaction with untrained users in diverse environments. Unlike industrial robots, which operate in controlled settings, personal care robots must function safely around infants, elderly individuals, and others. ISO 13482, harmonized under the Machinery Directive (2006/42/EC) and adopted in Japan as JIS B 8445, addresses the safe design and protective measures for these robots.

Hazard Assessments

ISO 13482 outlines various hazards and requires manufacturers to conduct risk assessments to mitigate these risks. The main hazards include:

1. Battery and charging issues.
2. Energy storage and supply hazards.
3. Startup and operation hazards.
4. Electrostatic potential issues.
5. Electromagnetic interference.
6. Stress and usage-related hazards.
7. Motion-related hazards.
8. Durability concerns.
9. Autonomous decision-making errors.
10. Moving component hazards.
11. Human awareness issues.
12. Environmental conditions.
13. Localization and navigation errors.

Safety Functions

To address these hazards, safety functions are implemented, including:

1. Emergency stops.
2. Protective stops.
3. Workspace limits.
4. Speed control.
5. Force control.
6. Collision avoidance.
7. Stability control.

The required safety performance level (PL) or safety integrity level (SIL) for these functions is determined by risk assessments in accordance with ISO 13849-1 or IEC 62061.

“The Dawning of the Ethics of Robots”

Overview

Service robots are becoming a familiar presence in our daily lives, whether they're helping with household chores, providing care to the elderly, or assisting in various industries. As we embrace the convenience and efficiency these robots bring, it's essential to consider the ethical and environmental impacts of their widespread use. This document delves into these issues, emphasizing the importance of responsible development and deployment to ensure that service robots benefit society without compromising ethical standards or the environment.

Ethical Issues

1. **Privacy Concerns**:
    - **Data Collection**: Service robots use sensors and cameras to perform their tasks, which means they can collect a lot of personal data about the people they interact with. This can include sensitive information, such as daily routines and personal preferences. It's crucial to ensure that this data is collected and stored responsibly, with clear guidelines on how it will be used and protected. Failure to do so can lead to privacy breaches and unauthorized use of personal data.
    - **Surveillance and Consent**: The ability of service robots to continuously monitor their surroundings can lead to concerns about surveillance. People might not always be aware that they are being recorded, raising questions about consent and

autonomy. Transparent policies and the option for individuals to opt out of data collection are essential to address these concerns. Without proper consent mechanisms, the deployment of service robots can infringe on individual privacy rights.

- **Data Security**: With the vast amount of data collected by service robots, ensuring its security is paramount. Robust encryption, secure data storage, and regular security audits are necessary to protect this information from unauthorized access and misuse. Data breaches can lead to significant harm, including identity theft and financial loss.

2. **Autonomy and Decision Making**:

   - **Ethical Programming**: Programming service robots to make ethical decisions is a complex challenge. These robots need to act in ways that align with societal values and ethical norms, but translating these principles into code is not straightforward. Developing comprehensive ethical guidelines for robot behavior is crucial to ensure they behave in a socially acceptable manner. This involves considering various ethical frameworks and continuously updating them to address new challenges.

   - **Accountability and Responsibility**: When a service robot causes harm or damage, determining who is responsible can be difficult. Is it the manufacturer, the programmer, or the operator? Clear legal frameworks are needed to define

accountability and ensure that the right parties are held responsible. This helps prevent legal ambiguities and ensures fair compensation for affected individuals.

- **Bias and Fairness**: Robots can unintentionally reflect the biases present in their programming or data sets. This can lead to unfair or discriminatory behavior. Regularly auditing algorithms and data for biases and implementing corrective measures is essential to ensure fairness. Addressing bias proactively can help prevent unequal treatment and enhance trust in robotic systems.

3. **Impact on Employment**:

   - **Job Displacement**: As service robots take over tasks traditionally performed by humans, there is a risk of job displacement, particularly in roles that involve repetitive or manual labor. This can have significant economic impacts on workers, highlighting the need for policies to support those affected, such as retraining and reskilling programs. Providing support can help workers transition to new roles and mitigate the negative effects of job loss.

   - **Reskilling and Upskilling**: Providing opportunities for workers to learn new skills and adapt to the changing job market is crucial. Ensuring equitable access to these training programs helps mitigate the negative impacts of job

displacement and supports a smooth transition for workers into new roles.

- Employers and governments should collaborate to create comprehensive reskilling initiatives.

4. Human-Robot Interaction:

- **Trust and Dependency**: Building trust between humans and robots is essential as these machines become more integrated into our lives. However, over-dependence on robots can diminish human skills and autonomy. It's important to design robots that complement human abilities rather than replace them entirely. Encouraging human-robot collaboration can enhance the effectiveness of both.

- **Dehumanization**: Over-reliance on robots for care and service tasks can lead to a sense of dehumanization. Maintaining the human element in these interactions is critical to ensure empathy and emotional support are not lost. Training for caregivers and service providers should emphasize the importance of human interaction and emotional connections.

Environmental Issues

1. **Energy Consumption**:

    - **Operational Energy Use**: Service robots require energy to operate, contributing to their carbon footprint. The environmental impact of this energy consumption needs to be

mitigated through the use of renewable energy sources and energy-efficient technologies. Implementing energy-saving features in robot design and promoting the use of renewable energy can help reduce their environmental footprint.

- **Battery Life and Disposal**: The batteries that power service robots pose environmental challenges when it comes to disposal. Proper recycling and disposal methods are necessary to prevent pollution and resource depletion. Manufacturers should design robots with recyclable batteries and establish take-back programs to ensure responsible disposal.

2. **Electronic Waste**:

    - **Production and Disposal**: The production of service robots involves the use of various raw materials, including rare and non-renewable resources. At the end of their lifecycle, these robots contribute to electronic waste. Sustainable manufacturing practices and effective recycling programs are essential to address this issue. Adopting circular economy principles, where components are reused and refurbished, can help reduce waste.

    - **Sustainable Manufacturing**: Manufacturers must adopt eco-friendly practices to minimize the environmental impact of robot production. This includes using sustainable materials and ensuring efficient recycling processes. Investing in research and development to create biodegradable and

sustainable materials for robot production is a critical step toward reducing their environmental footprint.

3. **Resource Depletion**:

    - **Material Extraction**: Extracting raw materials for the production of service robots can lead to environmental degradation and resource depletion. Sustainable sourcing practices are essential to mitigate these impacts. Companies should prioritize sourcing materials from environmentally responsible suppliers and invest in alternative materials that have a lower environmental impact.

    - **Lifecycle Management**: Effective lifecycle management of service robots, including maintenance, refurbishment, and recycling, is crucial to reduce their environmental footprint and promote sustainability. Implementing product life cycle assessment (LCA) methodologies can help identify areas for improvement in the environmental performance of robots. Companies should provide guidelines and support for the proper maintenance and disposal of their products.

A **“ YOLO, SSD, and Faster R-CNN for Object Detection ”**

Overview

Object detection is a critical task in computer vision, requiring models to identify and locate objects within images or video frames. Among the leading algorithms for this task are YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), and Faster R-CNN (Region-based Convolutional Neural Networks). Each of these algorithms has distinct methodologies, strengths, and weaknesses, making them suitable for different applications. This detailed comparison examines how each algorithm works, their performance metrics, advantages, and limitations.

YOLO (You Only Look Once)

How YOLO Works

YOLO treats object detection as a single regression problem, directly mapping from image pixels to bounding box coordinates and class probabilities. Here's a step-by-step breakdown of how YOLO operates:

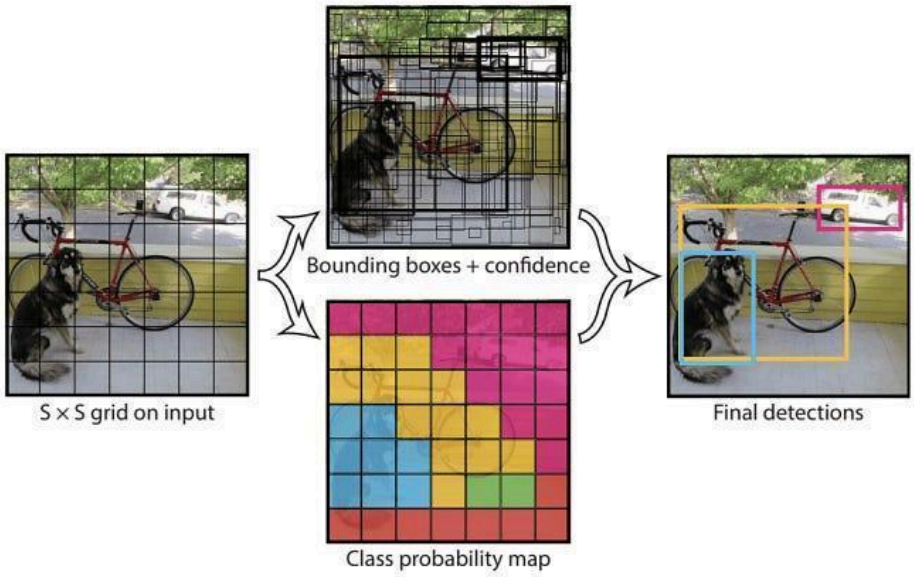


Figure 3.8: YOLO Operation

1. **Image Division**:

    - The input image is divided into an (SxS) grid.
    - Each grid cell is responsible for detecting objects whose center falls within the cell.

2. **Bounding Box Prediction**:

    - Each cell predicts B bounding boxes, each with a confidence score.
    - The confidence score represents the probability that a bounding box contains an object and the accuracy of the bounding box's location.

3. **Class Prediction**:

- Each grid cell predicts conditional class probabilities for the object.

- The final class-specific confidence score is obtained by multiplying the conditional class probability with the bounding box confidence score.

4. **Non-Maximum Suppression**:

    - YOLO uses non-maximum suppression to remove redundant bounding boxes, keeping only the most accurate ones.

Performance

- **Speed**: YOLO is known for its real-time performance, capable of processing up to 45 frames per second (fps) on a high-end GPU like the NVIDIA TITAN X.

- **Accuracy**: While YOLO achieves high accuracy, it can struggle with small objects and densely packed scenes due to its coarse grid structure.

Advantages

- **Fast Inference**: YOLO's single-stage approach allows for rapid processing, making it ideal for applications requiring real-time detection.

- **Global Context**: By considering the entire image at once, YOLO leverages global context, reducing false positives.

Limitations

- **Localization Errors**: YOLO can produce less precise bounding boxes compared to two-stage detectors like Faster R-CNN.
- **Small Object Detection**: The algorithm's grid-based approach can lead to difficulties in accurately detecting small objects.

SSD (Single Shot MultiBox Detector)

How SSD Works

SSD is a single-stage object detector that uses multiple feature maps at different scales to handle objects of various sizes. Here's how SSD operates:

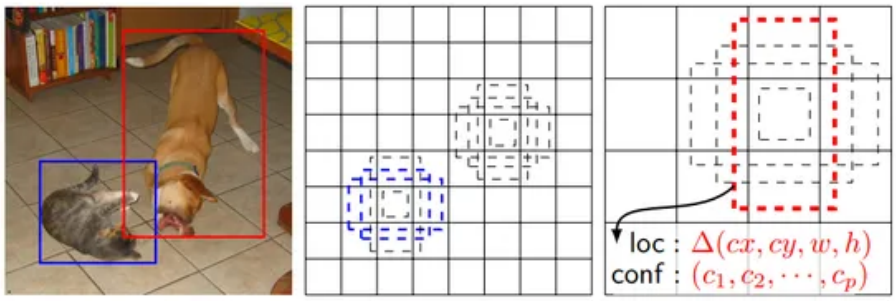


Figure 3.9 SSD multibox detector

1. **Feature Extraction**:

    - A base convolutional network (like VGG-16) is used to extract feature maps from the input image.

    - SSD adds several convolutional layers on top of the base network to create multiple feature maps of different resolutions.

2. **Bounding Box and Class Prediction**:

    - For each feature map cell, SSD predicts a fixed set of default bounding boxes (anchor boxes) with different aspect ratios and scales.

    - For each bounding box, SSD predicts the class scores and the offsets for adjusting the box coordinates.

3. **Multi-Scale Detection**:

    - The use of multiple feature maps allows SSD to detect objects at different scales, improving its ability to detect small objects.

4. **Non-Maximum Suppression**:

    - Like YOLO, SSD employs non-maximum suppression to filter out redundant bounding boxes.

Performance

- **Speed**: SSD offers a good balance between speed and accuracy, being faster than Faster R-CNN but typically slower than YOLO.
- **Accuracy**: SSD performs well, particularly for small objects, due to its multi-scale detection approach.

Advantages

- **Multi-Scale Detection**: The use of multiple feature maps enhances SSD's ability to detect objects of various sizes.
- **Speed-Accuracy Trade-off**: SSD provides a favorable compromise between inference speed and detection accuracy.

Limitations

- **Complexity**: SSD's architecture is more complex than YOLO's due to the additional feature maps and anchor boxes.
- **False Positives**: The algorithm can produce more false positives, especially in cases of overlapping objects.

Faster R-CNN (Region-based Convolutional Neural Networks)

How Faster R-CNN Works

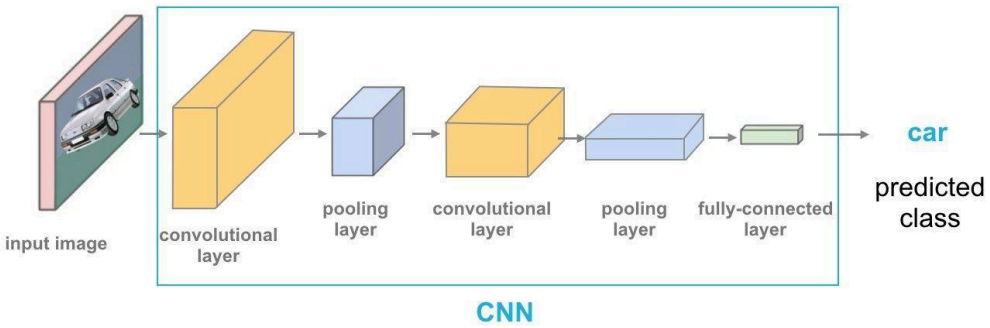


Figure 3.10: R-CNN

Faster R-CNN is a two-stage detector that first generates region proposals and then classifies these proposals. Here's a detailed look at its working:

1. **Feature Extraction**:
    - A convolutional neural network (e.g., ResNet) is used to extract feature maps from the input image.
2. **Region Proposal Network (RPN)**:
    - The RPN generates candidate object bounding boxes (region proposals) from the feature maps.
    - For each region proposal, the RPN predicts the objectness score and the bounding box coordinates.
3. **RoI Pooling**:
    - The region proposals are fed into a RoI (Region of Interest) pooling layer to extract fixed-size feature maps.

4. **Object Detection**:

    - These fixed-size feature maps are then passed through fully connected layers to perform classification and bounding box regression.

5. **Non-Maximum Suppression**:

    - Non-maximum suppression is applied to eliminate redundant bounding boxes and retain the most accurate ones.

Performance

- **Speed**: Faster R-CNN is slower than single-stage detectors like YOLO and SSD due to its two-stage process.

- **Accuracy**: It is known for its high accuracy, especially in complex scenes and for small objects, due to the precise region proposals.

Advantages

- **High Precision**: The two-stage detection process allows Faster R-CNN to achieve superior accuracy in both localization and classification.

- **Small Object Detection**: Performs well in detecting small and overlapping objects due to the accurate region proposals.

Limitations

- **Inference Time**: The two-stage nature results in slower inference, making it less suitable for real-time applications.

- **Complexity**: Faster R-CNN is more computationally intensive and complex to implement and train.

“ Face Recognition in Python - A Comprehensive Guide ”

Overview

This comprehensive guide by Basil Chacko Mathew offers an in-depth exploration of face recognition in Python. It spans the fundamental concepts, essential libraries, and step-by-step implementation processes required to build an effective face recognition system.

Key Sections and Detailed Summary

1. Introduction to Face Recognition

- **Conceptual Overview**:
    - Face recognition involves identifying or verifying an individual based on their facial features.
    - It is a critical component in numerous applications such as security systems, attendance tracking, and user authentication on personal devices.
- **Importance**:
    - Emphasizes the growing significance of face recognition technology in enhancing security and convenience in various domains.

2. Understanding the Basics

- **Face Detection vs. Face Recognition**:

Face Detection: The process of identifying and locating faces within an image or video frame. It is essentially about finding all the faces present in a given scene. Face detection is a preliminary step and serves as the foundation for subsequent face recognition processes. It involves techniques like the use of Haar cascades, HOG (Histogram of Oriented Gradients) features, or deep learning-based methods.

Key Points:

- Involves scanning the entire image to locate faces.
- Utilizes classifiers like Haar cascades in OpenCV or HOG + SVM in dlib.
- Primarily focuses on the spatial location of faces.

Face Recognition: The process of identifying or verifying a person by comparing the detected face with known faces in a database. It involves extracting features from the detected face and matching them against pre-stored features to determine identity.

Key Points:

- Goes beyond detection to determine the identity of the face.
- Uses encodings or feature vectors for comparison.

- Employs algorithms such as deep metric learning to create embeddings that represent facial features.

- **Technical Differentiation**:

    - Detection is a simpler, faster process focusing on where faces are located.

    - Recognition is more complex, involving detailed feature extraction and matching to identify the person.

3. Face Recognition Libraries

- **Overview of Key Libraries**:

    - **OpenCV**: Provides a comprehensive set of tools for computer vision tasks, including face detection.

    - **dlib**: Offers machine learning algorithms and tools for facial landmark detection, face encodings, and recognition.

    - **face_recognition**: A high-level library built on dlib, simplifying the face recognition process with easy-to-use functions.

4. Setting Up the Environment

- **Prerequisites**:

    - Detailed instructions on setting up a Python environment for face recognition tasks.

- **Installation Commands**:
    - **opencv-python**: **pip install opencv-python**
    - **dlib**: **pip install dlib**
    - **face_recognition**: **pip install face_recognition**
    - **imutils**: **pip install imutils** (for additional image processing utilities)
- **Environment Configuration**:
    - Guidance on configuring the development environment for optimal performance and compatibility.

5. Face Detection

- **Using OpenCV for Face Detection**:
    - **Loading and Processing Images**:
        - Load an image using OpenCV and convert it to grayscale for easier processing.
        - Example code for loading and displaying images.
    - **Haar Cascade Classifier**:
        - Utilizes pre-trained Haar cascade classifiers to detect faces within an image.

- Example code for applying the Haar cascade classifier and drawing rectangles around detected faces.

6. Face Recognition with face_recognition Library

- **Library Introduction**:
    - The **face_recognition** library provides a simple interface for performing face recognition tasks using dlib.
- **Step-by-Step Guide**:
    - **Loading and Encoding Known Faces**:
        - Load images of known individuals and create encodings (numerical representations) of their faces.
    - **Comparing Faces**:
        - Compare a new image (containing an unknown face) against the known face encodings to identify matches.
    - **Example Code**:

7. Improving Accuracy

- **Techniques to Enhance Recognition Accuracy**:

- **High-Quality Images**: Use high-resolution images to capture more facial details.
- **Consistent Lighting**: Ensure consistent and adequate lighting conditions to minimize shadows and highlights.
- **Multiple Images**: Use multiple images of each individual to create a robust and comprehensive face encoding.

8. Building a Complete Face Recognition System

- **Integration of Face Detection and Recognition**:
  - Combining face detection (using OpenCV) and face recognition (using **face_recognition**) into a cohesive system.

3.1.1 Industry Standards

"Personal Care Robots"

Overview

Personal care robots are emerging as integral components in various sectors, significantly improving the quality of life by assisting with daily activities. These robots, unlike their industrial counterparts, are designed to interact directly with humans, often in unpredictable environments. This document explores the ISO standards that govern the safety and functionality of personal care robots, focusing primarily on ISO 13482.

Categories of Personal Care Robots

ISO 13482 classifies personal care robots into three main categories:

1. **Mobile Servant Robots**: These robots perform tasks like handling objects or exchanging information while moving around.
2. **Physical Assistant Robots**: These assist humans in performing physical tasks, enhancing the user's capabilities.

3. **Person Carrier Robots**: These transport humans to specific destinations.

Key Functions

Personal care robots are designed to:

- Navigate homes or public buildings without colliding with obstacles.
- Interact with humans, including exchanging objects.
- Handle objects of various sizes.
- Assist with mobility and reduce physical load for humans.

Market Growth

The personal care robots' market is expected to grow at a rate of 19% annually, reaching $10 billion in sales by 2024. This growth is driven by advancements in robotics technology and increasing consumer familiarity and comfort with such robots.

Safety Standards

Personal care robots must adhere to stringent safety standards due to their interaction with untrained users in diverse environments. Unlike industrial robots, which operate in controlled settings, personal care robots must function safely around infants, elderly individuals, and others. ISO 13482, harmonized under the Machinery Directive (2006/42/EC) and adopted in Japan as JIS B 8445, addresses the safe design and protective measures for these robots.

Hazard Assessments

ISO 13482 outlines various hazards and requires manufacturers to conduct risk assessments to mitigate these risks. The main hazards include:

1. Battery and charging issues.
2. Energy storage and supply hazards.
3. Startup and operation hazards.
4. Electrostatic potential issues.
5. Electromagnetic interference.
6. Stress and usage-related hazards.
7. Motion-related hazards.
8. Durability concerns.
9. Autonomous decision-making errors.
10. Moving component hazards.
11. Human awareness issues.
12. Environmental conditions.
13. Localization and navigation errors.

Safety Functions

To address these hazards, safety functions are implemented, including:

1. Emergency stops.
2. Protective stops.
3. Workspace limits.
4. Speed control.
5. Force control.
6. Collision avoidance.
7. Stability control.

The required safety performance level (PL) or safety integrity level (SIL) for these functions is determined by risk assessments in accordance with ISO 13849-1 or IEC 62061.

3.2 Literature on Potential Ethical and/or Environmental Issue

“Potential Ethical and Environmental Issues for Service Robots”

Overview

Service robots are becoming a familiar presence in our daily lives, whether they're helping with household chores, providing care to the elderly, or assisting in various industries. As we embrace the convenience and efficiency these robots bring, it's essential to consider the ethical and environmental impacts of their widespread use. This document delves into these issues, emphasizing the importance of responsible development and deployment to ensure that service robots benefit society without compromising ethical standards or the environment.

Ethical Issues

1. **Privacy Concerns**:
    - **Data Collection**: Service robots use sensors and cameras to perform their tasks, which means they can collect a lot of personal data about the people they interact with. This can include sensitive information, such as daily routines and personal preferences. It's crucial to ensure that this data is

collected and stored responsibly, with clear guidelines on how it will be used and protected. Failure to do so can lead to privacy breaches and unauthorized use of personal data.

- **Surveillance and Consent**: The ability of service robots to continuously monitor their surroundings can lead to concerns about surveillance. People might not always be aware that they are being recorded, raising questions about consent and autonomy. Transparent policies and the option for individuals to opt out of data collection are essential to address these concerns. Without proper consent mechanisms, the deployment of service robots can infringe on individual privacy rights.

- **Data Security**: With the vast amount of data collected by service robots, ensuring its security is paramount. Robust encryption, secure data storage, and regular security audits are necessary to protect this information from unauthorized access and misuse. Data breaches can lead to significant harm, including identity theft and financial loss.

2. **Autonomy and Decision Making**:

    - **Ethical Programming**: Programming service robots to make ethical decisions is a complex challenge. These robots need to act in ways that align with societal values and ethical norms, but translating these principles into code is not straightforward. Developing comprehensive ethical guidelines for robot behavior is crucial to ensure they behave in a

socially acceptable manner. This involves considering various ethical frameworks and continuously updating them to address new challenges.

- **Accountability and Responsibility**: When a service robot causes harm or damage, determining who is responsible can be difficult. Is it the manufacturer, the programmer, or the operator? Clear legal frameworks are needed to define accountability and ensure that the right parties are held responsible. This helps prevent legal ambiguities and ensures fair compensation for affected individuals.

- **Bias and Fairness**: Robots can unintentionally reflect the biases present in their programming or data sets. This can lead to unfair or discriminatory behavior. Regularly auditing algorithms and data for biases and implementing corrective measures is essential to ensure fairness. Addressing bias proactively can help prevent unequal treatment and enhance trust in robotic systems.

3. **Impact on Employment**:

    - **Job Displacement**: As service robots take over tasks traditionally performed by humans, there is a risk of job displacement, particularly in roles that involve repetitive or manual labor. This can have significant economic impacts on workers, highlighting the need for policies to support those affected, such as retraining and reskilling programs. Providing

support can help workers transition to new roles and mitigate the negative effects of job loss.

- **Reskilling and Upskilling**: Providing opportunities for workers to learn new skills and adapt to the changing job market is crucial. Ensuring equitable access to these training programs helps mitigate the negative impacts of job displacement and supports a smooth transition for workers into new roles. Employers and governments should collaborate to create comprehensive reskilling initiatives.

4. **Human-Robot Interaction**:

    - **Trust and Dependency**: Building trust between humans and robots is essential as these machines become more integrated into our lives. However, over-dependence on robots can diminish human skills and autonomy. It's important to design robots that complement human abilities rather than replace them entirely. Encouraging human-robot collaboration can enhance the effectiveness of both.

    - **Dehumanization**: Over-reliance on robots for care and service tasks can lead to a sense of dehumanization. Maintaining the human element in these interactions is critical to ensure empathy and emotional support are not lost. Training for caregivers and service providers should emphasize the importance of human interaction and emotional connections.

Environmental Issues

1. **Energy Consumption**:

    - **Operational Energy Use**: Service robots require energy to operate, contributing to their carbon footprint. The environmental impact of this energy consumption needs to be mitigated through the use of renewable energy sources and energy-efficient technologies. Implementing energy-saving features in robot design and promoting the use of renewable energy can help reduce their environmental footprint.

    - **Battery Life and Disposal**: The batteries that power service robots pose environmental challenges when it comes to disposal. Proper recycling and disposal methods are necessary to prevent pollution and resource depletion. Manufacturers should design robots with recyclable batteries and establish take-back programs to ensure responsible disposal.

2. **Electronic Waste**:

    - **Production and Disposal**: The production of service robots involves the use of various raw materials, including rare and non-renewable resources. At the end of their lifecycle, these robots contribute to electronic waste. Sustainable manufacturing practices and effective recycling programs are essential to address this issue. Adopting circular economy principles, where components are reused and refurbished, can help reduce waste.

    - **Sustainable Manufacturing**: Manufacturers must adopt eco-friendly practices to minimize the environmental impact

of robot production. This includes using sustainable materials and ensuring efficient recycling processes. Investing in research and development to create biodegradable and sustainable materials for robot production is a critical step toward reducing their environmental footprint.

3. **Resource Depletion**:

- **Material Extraction**: Extracting raw materials for the production of service robots can lead to environmental degradation and resource depletion. Sustainable sourcing practices are essential to mitigate these impacts. Companies should prioritize sourcing materials from environmentally responsible suppliers and invest in alternative materials that have a lower environmental impact.

- **Lifecycle Management**: Effective lifecycle management of service robots, including maintenance, refurbishment, and recycling, is crucial to reduce their environmental footprint and promote sustainability. Implementing product lifecycle assessment (LCA) methodologies can help identify areas for improvement in the environmental performance of robots. Companies should provide guidelines and support for the proper maintenance and disposal of their products.

4 CHAPTER 4:DETAILED DESIGN

4.1 Detailed Specifications

4.1.1 Routing Algorithms:

In our development for the delivery and user guide autonomous PeopleBot robot, efficient and reliable routing algorithms are essential for enabling the robot to navigate effectively through the environment. PeopleBot relies on routing techniques to perform tasks. What does really ensure the success of accomplishing these tasks is the robot ability to path planning by finding the optimal and least cost path, and avoiding static and dynamic obstacles.

The importance of the routing algorithms for PeopleBot is represented in the efficiency of finding the shortest and fastest paths to its destination, reducing travel time and saving energy. In addition, these algorithms enable the robot to navigate around obstacles, static or dynamic, by the integration with the robot laser.

All of this is achieved by the Integration has made on the PeopleBot between the A* (A Star) Search Algorithm and Reinforcement Learning (RL).

A (A Star) Search Algorithm: A* is a well-known pathfinding algorithm that is widely used due to its efficiency and accuracy. It combines the advantages of Dijkstra's algorithm and Best-First Search to find the shortest path to the target by using a combination of heuristic searching and searching based on the shortest path. A* algorithm is defined as best-first algorithm, because each cell in the configuration space is evaluated [12].

Reinforcement Learning (RL): RL is a type of machine learning where the robot learns to make decisions by interacting with its environment and receiving rewards for its actions. By using RL, PeopleBot can improve its routing decisions over time based on past experiences, leading to more efficient and intelligent navigation.

These algorithms provide a robust framework for PeopleBot's navigation system, combining the deterministic approach of A* with the adaptive learning capabilities of RL. This integrated approach ensures that PeopleBot can effectively fulfill its roles as a delivery and user guide robot, navigating complex environments with adaptability.

A* (A Star) Search Algorithm

A* is highly effective in various applications, including robotics, gaming, and geographic information systems, due to its ability to guarantee the shortest path while optimizing computational efficiency. Among graph-based searching algorithms like Dijkstra, depth-first search (DFS), and breadth-first search (BFS), A* is particularly notable for its completeness, efficient search area, and ability to use heuristics [7].

The A* algorithm's efficiency and effectiveness in finding the shortest path rely heavily on its core components: the **heuristic function**, the **cost function**, giving the **evaluation function**. These components work together to guide the search process, ensuring that the algorithm prioritizes paths that are likely to lead to the goal in the most efficient manner. By maintaining a balance between the actual costs incurred and the estimated costs to the goal,

A* can dynamically adjust its search strategy to navigate through complex environments effectively.

Heuristic Function (h(n))

The heuristic function (h(n)) is a key component of the A* algorithm, providing an estimated cost to reach the goal from the current node (n). It guides the search process by prioritizing nodes that appear closer to the goal. Common heuristics include:

- **Manhattan Distance:** Suitable for grid-based environments where movement is restricted to horizontal and vertical steps.

- **Euclidean Distance:** Used in environments where diagonal movement is allowed, calculating the straight-line distance to the goal.

- **Custom Heuristics:** In PeopleBot, the heuristic function is dynamically adjusted using Q-values from Reinforcement Learning, enhancing its adaptability and precision.

Cost Function (g(n))

The cost function (g(n)) represents the exact cost incurred to reach the current node (n) from the start node. This function accumulates the total distance or time traveled and is crucial for determining the efficiency of the path taken.

Evaluation Function (f(n))

The evaluation function (f(n)) combines both the actual cost to reach the current node and the estimated cost to reach the goal:

f(n) = g(n) + h(n)

- g(n): Actual cost from the start node to the current node.

- h(n): Estimated cost from the current node to the goal node.

The node with the lowest **(f(n))** value is selected for expansion, ensuring an optimal and efficient pathfinding process. As A* traverses the graph, it follows a path of the lowest known cost, keeping a sorted priority queue of alternate path segments along the way. If, at any point, a segment of the path being traversed has a higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment instead. This process continues until the goal is reached.

A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). It uses a distance-plus-cost heuristic function (usually denoted (f(x)) to determine the order in which the search visits nodes in the tree. The distance-plus-cost heuristic is a sum of two functions: the path-cost function, which is the cost from the starting node to the current node (usually denoted (g(x))), and an admissible "heuristic estimate" of the distance to the goal (usually denoted (h(x))).

The A* algorithm is renowned for its effectiveness in finding the shortest path through its heuristic search technique. However, the conventional A*

algorithm often encounters challenges in efficiently searching and evaluating numerous cells when seeking a goal. To ensure a robot reaches its destination swiftly, it is crucial to minimize the number of nodes checked while maintaining low time complexity. The proposed model, shown in Figure **4.1**, addresses these concerns by streamlining the path-planning process. By reducing unnecessary computations, the proposed model enhances the overall performance of the A* path-planning algorithm, particularly in navigating around obstacles. Illustrated in Figure 4.1, the proposed model comprises a robot path-planning algorithm optimized for real-time usability. Through this model, the robot efficiently explores neighboring grid cells, improving the path-planning algorithm's performance across various grid maps with different obstacle environments and starting and goal positions [8].

Figure 4.1, Block diagram of our proposed model, Reference.

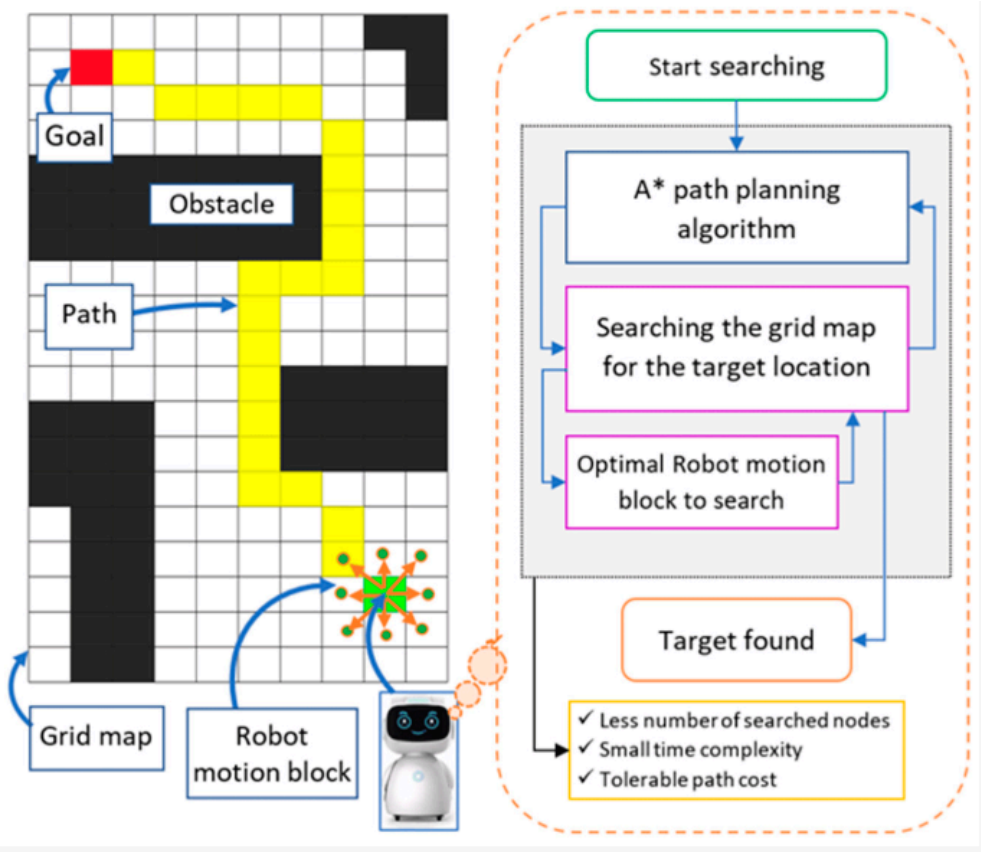


Figure4.1: A* Grid Map and flowchart

A Star Limitations:

A* algorithm is proficient in finding the shortest path but has limitations. It can be computationally inefficient in complex or dynamic environments, requiring exploration of many nodes, leading to high time and space complexity. A* heavily depends on the accuracy of its heuristic function and may not always consider real-world uncertainties or environmental changes, resulting in inefficient pathfinding or increased computational load. To overcome these limitations, reinforcement learning (RL) was integrated with A*. RL enhances adaptability and efficiency of pathfinding by continuously

learning from the environment and adjusting the heuristic function dynamically. Q-values from RL help the algorithm better estimate the true cost to the goal, considering past experiences and real-time data. The adaptive nature of RL allows the robot to quickly respond to changes and obstacles, maintaining optimal performance in various scenarios.

4.1.2 Reinforcement Learning Algorithms in Routing for Autonomous Robots

Reinforcement Learning (RL) is a machine learning method in which an agent learns to make decisions by taking actions in an environment to maximize total rewards. Unlike supervised learning, where the model learns from a dataset of correct answers, RL focuses on learning from the consequences of actions, emphasizing exploration and exploitation of the environment.

In our project, the RL algorithm is essential for optimizing the navigation and pathfinding capabilities of the system. The robot engages with various obstacles and pathways within the physical campus environment. Through the RL framework, the robot learns to decide on the best paths to reach its goal destination while efficiently avoiding obstacles. In the context of routing, the environment comprises the physical space through which the robot navigates, and the agent is the robot itself.

RL algorithms like Q-learning, Deep Q-Networks (DQN), or Policy Gradient methods may be utilized. These algorithms progressively update the value function and policy based on the robot's experiences, enhancing its ability to navigate the environment effectively.

RL empowers robots to learn optimal routes through trial and error, adapting to changing conditions and improving their performance over time. Using a Reinforcement Learning (RL) algorithm in a Peoplebot robot to enable it to discover new places and store them for future reference.

One of the key advantages of RL algorithms in routing is their ability to handle complex and dynamic environments like a university campus, where

new obstacles or paths may frequently emerge. As the robot interacts more with its surroundings, it continually enhances its navigation strategy, leading to more efficient and reliable operation.

Unlike traditional algorithms that rely on predefined maps or graphs, RL algorithms learn directly from experience. Through repeated interactions with the environment, the robot learns which actions lead to favorable outcomes (rewards) and adjusts its behavior accordingly by using this equation:

Q-learning Update Rule for Dynamic Routing

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- $Q(s,a)$ is the current Q-value of taking action a in state s.
- α is the learning rate ($0 < \alpha \leq 1$).
- r is the reward received after taking action a in state s.
- γ is the discount factor ($0 \leq \gamma < 1$).
- s' is the new state after taking action a.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279-292. doi:10.1007/BF00992698.

In the context of routing:

State (s) : the current position or condition of the robot in the environment.

Action (a): the possible directions or moves the robot can take from its current state.

Reward (r): the feedback received after taking an action, which could be based on factors like distance traveled, obstacles avoided, or progress towards the destination.

Next State (s'): the new position or condition after the action is taken.

The policy is the strategy the robot follows to determine its actions based on the current state. In RL, the policy can be deterministic or stochastic and is often represented as a function or a lookup table.

Value Function: The value function estimates the expected total reward that can be obtained from each state, helping the robot evaluate the long-term benefits of its actions rather than just immediate rewards.

By continuously updating the Q-values using the above equation, the robot develops a policy that maximizes its cumulative reward over time. This process allows the robot to adapt to changes in the environment, such as dynamic obstacles or varying terrain, making RL particularly suited for real-world routing tasks.

In the context of discovering new places, RL algorithms enable robots to explore their surroundings and identify previously unknown locations. This is achieved through a process of exploration-exploitation trade-off, where the robot balances between trying new paths and exploiting known routes. Initially, the robot may follow a random or heuristic-based exploration strategy to gather information about its environment. As it collects data and receives feedback (rewards) from the environment, the robot refines its understanding of space and learns to navigate more efficiently.

Storage and Memory

Once a new place is discovered, the robot must store this information for future reference. RL algorithms facilitate this by enabling the robot to encode spatial representations of the environment and associate them with specific locations. This can be achieved through techniques such as memory-based learning or neural network-based embeddings. By storing representations of places along with their corresponding features (e.g., landmarks, distances), the robot can recall and navigate to these locations when needed. In our graduation project the robot will keep updating the map. When it discovers obstacles, it will represent it as dots on the map, walls and windows will be represented as continuous lines on the map.

Practical Implementation

Implementing RL algorithms for routing in autonomous robots requires careful consideration of several factors, including sensor capabilities, computational resources, and real-time constraints. Sensor data, such as lidar scans or camera images, provide crucial input for the robot to perceive its surroundings. Computational resources are needed for training and inference, especially if deep reinforcement learning techniques are employed. Real-time constraints impose limitations on the speed and efficiency of decision-making, requiring algorithms that can operate within tight timeframes.

In conclusion, reinforcement learning algorithms offer a promising approach to routing in autonomous robots, enabling them to discover new places and store them for future reference. By learning directly from experience, robots can adapt to dynamic environments and improve their navigation capabilities

over time. However, practical challenges such as sensor limitations and real-time constraints must be addressed to realize the full potential of RL in robotic routing. With ongoing advancements in machine learning and robotics, the integration of RL algorithms holds great promise for enabling autonomous robots to navigate and explore the world with greater efficiency and autonomy.

4.1.3 Obstacles detection and avoidance

The obstacle detection and avoidance system of our project is a critical component that ensures the Peoplebot can navigate safely and effectively in dynamic environments. This system relies on both static and dynamic obstacle detection, utilizing advanced sensors and sophisticated algorithms to enhance the robot's operational efficiency and safety.

Laser and Sonar Sensors

The Peoplebot is equipped with a combination of laser rangefinders and sonar sensors to detect obstacles in its path. The laser rangefinders, typically using Light Detection and Ranging (LiDAR) technology, provide high-resolution data about the robot's surroundings by emitting laser beams and measuring the time it takes for the reflections to return. This allows the robot to create a detailed map of the environment, identifying both static and dynamic obstacles with high accuracy.

Sonar sensors, on the other hand, operate by emitting sound waves and measuring the time it takes for the echoes to return after hitting an object. These sensors are particularly effective in detecting obstacles that are closer to the robot and can operate in various lighting conditions, making them a robust complement to the laser sensors.

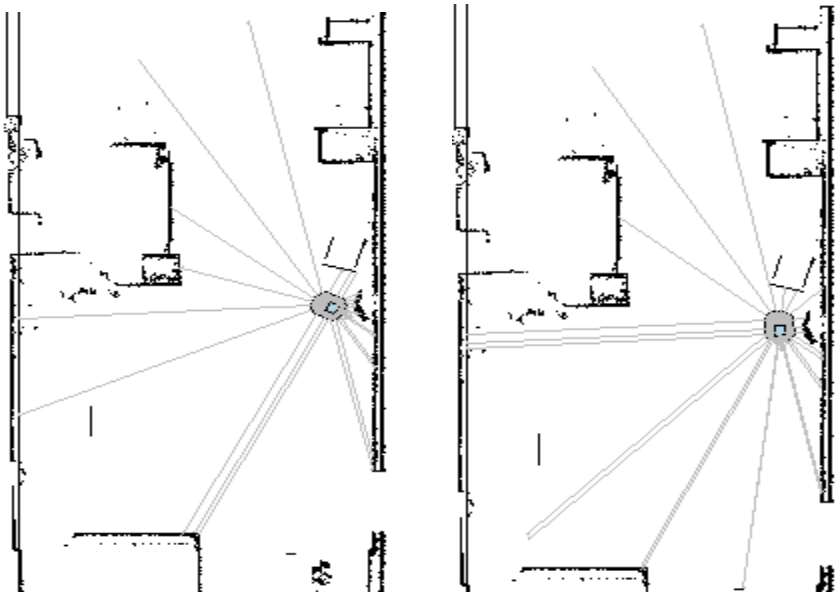


Figure 4.2: Sonar detection

Here, we can observe that the sonar detects the obstacle (the wall) and avoids it when it gets too close.

*This map is the default map(columbia) on MobileSim.

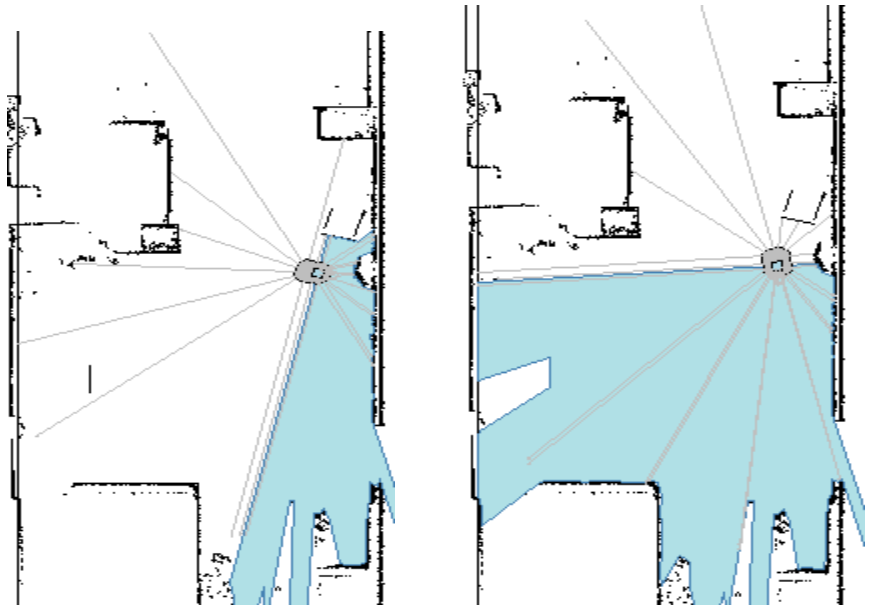


Figure 4.3: Laser detection

Here, we can observe that the Laser detects the obstacle (the wall) and avoids it when it is far away.

*This map is the default map(columbia) on MobileSim.

Obstacle Detection Algorithms

The data collected from the laser and sonar sensors is processed using sophisticated algorithms to detect and classify obstacles. For static obstacles, the robot continuously updates its map of the environment, marking areas that are consistently occupied. For dynamic obstacles, the robot tracks moving objects in real-time, predicting their path to avoid collisions.

Integration with Control System

The obstacle detection and avoidance system is tightly integrated with the Peoplebot's control system. This integration ensures that the robot's movements are continuously adjusted based on real-time sensor data, enabling seamless and efficient navigation. The control system prioritizes safety, ensuring that the robot maintains a safe distance from obstacles while still progressing toward its goal.

Use Cases and Applications

In practical applications, these capabilities allow the Peoplebot to perform tasks such as guiding students through a building, delivering items between users, and executing security patrols. The advanced obstacle detection and avoidance system ensures that the robot can operate autonomously and reliably in complex and dynamic environments, providing valuable assistance in various scenarios.

Overall, the integration of laser and sonar sensors with advanced algorithms and control strategies makes the Peoplebot a robust and versatile platform for autonomous navigation, capable of performing a wide range of tasks while ensuring safety and efficiency.

4.1.4 People Recognition System

The people recognition feature of PeopleBot is essential for its functionality in real-time applications, such as counting people in a crowd or monitoring environments. The system uses the YOLO (You Only Look Once) object detection model to identify and locate people in the video feed. Below are the detailed specifications:

- **Operating System**: Windows, macOS, or Linux.
- **Python**: Python 3.7 or later.
- **Libraries and Dependencies**:
    - **OpenCV**: For video capture and image processing.
    - **NumPy**: For numerical operations.
    - **YOLO**: For object detection.

System Architecture

- **Camera (Video Feed)**: Provides real-time input to the system.
- **People Detection (YOLO)**: Processes the video feed to detect people using the YOLO.v3 model.
- **Display and Count**: Draws bounding boxes around detected people in the video feed and displays the count of detected people on the screen.

- **Output/Action**: Real-time visualization of detected people and the number of people displayed on the screen.

4.1.5 Face Recognition Security System

The security feature of PeopleBot is crucial for maintaining safety within the engineering faculty after hours. It is designed to ensure that only authorized individuals can access the premises by verifying identities through a combination of barcode detection and facial recognition. Here are the specifications:

- **Database**: SQLite for storing and retrieving member information.
- **Libraries and Dependencies**:
    - **OpenCV**: For video capture and image processing.
    - **Pyzbar**: For barcode detection and decoding.
    - **face_recognition**: For face detection and recognition.
    - **SQLite**: For database operations, integrated with Python's standard library.

System Architecture

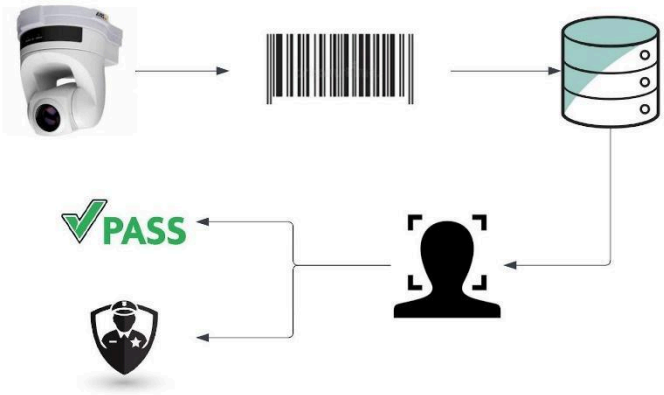


Figure 4.4

- **Camera (Video Feed)**: Provides real-time input to the system.
- **Barcode Detection (Pyzbar)**: Scans the video feed for barcodes (ID cards). It also decodes the barcode to retrieve the identification number; if a barcode is detected, it sends the ID number to the database query component.
- **Database Query (SQLite)**: Uses the ID number as the primary key to query the SQLite database, it retrieves the corresponding member's information, including the path to their photo: if a member is found, it sends the photo path to the face recognition component.
- **Face Recognition (face_recognition)**: Compare the live video feed's captured face with the known face from the database, it verifies the identity of the individual; if there is a match, it confirms the identity. If there is no match, it captures and saves the photo for security purposes.

- **Output/Action**: Displays the results of the recognition process, it saves images of mismatched identities for security purposes.

4.2 Discussion of Detailed Design Alternatives

Camera Options

Evaluating alternative models or brands of cameras besides the AXIS camera for potential cost benefits or performance improvements. This includes comparing specifications like resolution, zoom capabilities, low-light performance, and compatibility with facial recognition and video streaming requirements, to find the most suitable option for the PeopleBot's needs.

Sensor Alternatives

Exploring a range of sensors from different manufacturers for enhanced navigation, environmental interaction, and people detection. This includes considering variations in sonar, infrared, LIDAR, thermal imaging, and other advanced sensors to enhance obstacle detection, environmental awareness, and security capabilities.

Power Source Options

Assessing different types of batteries or power systems. For instance, considering the use of advanced lithium-polymer batteries for longer operational life, or exploring renewable energy sources like solar panels for auxiliary power to enhance the sustainability and efficiency of the PeopleBot.

Software Alternatives

Programming Language Flexibility

Considering a range of programming languages for system development. Options include Python for its ease of use and wide support in machine learning, or C++ for its performance efficiency. The choice of language will impact the ease of development, system performance, and compatibility with existing systems and AI functionalities.

Machine Learning Frameworks

Exploring various machine learning frameworks for image processing, facial recognition, and autonomous decision-making. Alternatives to consider may include TensorFlow, PyTorch, or OpenCV, depending on their suitability for real-time processing, support for the specific requirements of the PeopleBot's AI functionalities. Also, rioting algorithms for the robot path planning such as A Star Search and Reinforcement Learning.

Operating System Choices

Assessing different operating systems for the onboard computer, such as Linux for its customization and robustness, or Windows for its user-friendly interface and widespread support. The decision will affect software compatibility, ease of use, and system security.

Mechanical Design Variations

Modular Design Adaptations

Considering different approaches to the PeopleBot's modular design, such as varying the configuration or the ease of attaching and detaching modules. This could affect the robot's adaptability to different tasks, ease of maintenance, and customization based on user requirements.

Material Selection

Evaluating alternative materials for the robot’s body and structure, aiming to balance durability, weight, and cost. Materials such as carbon fiber composites could offer weight reduction, whereas aluminum or plastics might provide cost savings and ease of manufacturing.

Networking and Connectivity

Communication Systems

Looking into alternative communication technologies, such as 5G for faster data transmission or dedicated short-range communications (DSRC) for enhanced reliability in certain environments. This ensures the PeopleBot maintains robust and efficient connectivity for real-time data processing and interaction with its surroundings.

By exploring these design alternatives, the PeopleBot project can optimize its performance, cost, and adaptability, ensuring the final product is not only functionally efficient but also commercially viable and responsive to various market demands and technological advancements.

Object Detection Models Considered:

1. **YOLO (You Only Look Once)**:

    - **Architecture**: Single-shot detector that predicts bounding boxes and class probabilities directly from full images in a single evaluation.

    - **Advantages**: Fast, real-time object detection, end-to-end training.

    - **Disadvantages**: Trade-off between speed and accuracy, especially for smaller objects.

2. **SSD (Single Shot MultiBox Detector)**:

    - **Architecture**: Single-shot detector that uses different scales and aspect ratios for detecting objects.

    - **Advantages**: Good balance between speed and accuracy.

    - **Disadvantages**: Less accurate for small objects compared to Faster R-CNN.

3. **Faster R-CNN (Region-based Convolutional Neural Networks)**:

    - **Architecture**: Two-stage detector; the first stage proposes regions and the second stage classifies and refines these regions.

    - **Advantages**: High accuracy, particularly for detecting small objects.

- **Disadvantages**: Slower than YOLO and SSD, more computationally intensive.

Comparison of YOLO with SSD and Faster R-CNN

- **Speed**:
  - **YOLO**: Designed for real-time detection, processing images at around 45-155 FPS.
  - **SSD**: Processes images at around 20-60 FPS, making it faster than Faster R-CNN but slower than YOLO.
  - **Faster R-CNN**: Processes images at around 5-7 FPS, making it the slowest among the three.

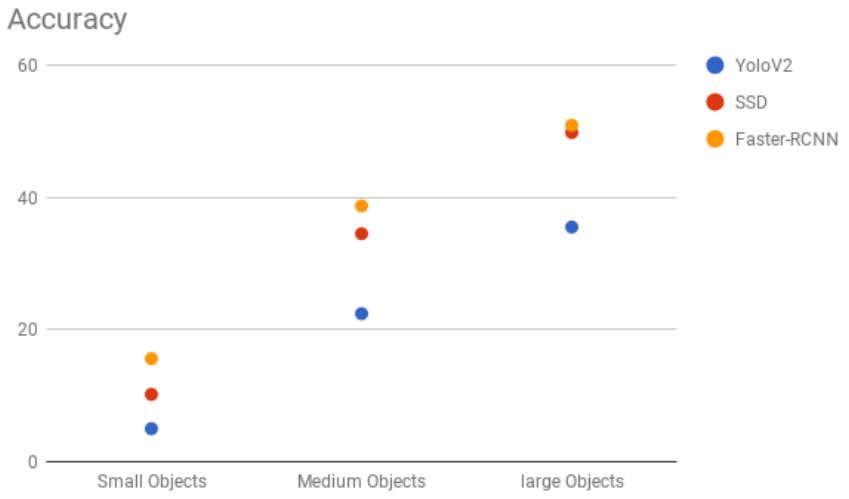


Figure 4.5: Accuracy

- **YOLO**: While fast, it may struggle with detecting smaller objects and has lower accuracy compared to Faster R-CNN.
- **SSD**: Balances speed and accuracy well but falls short compared to Faster R-CNN in terms of precision.
- **Faster R-CNN**: Offers the highest accuracy, particularly for smaller objects, but at the cost of speed.
- **Use Case Fit**:
    - **YOLO**: Ideal for applications requiring real-time detection, such as surveillance, robotics, and real-time monitoring systems.
    - **SSD**: Suitable for applications needing a balance between speed and accuracy.
    - **Faster R-CNN**: Best for applications where accuracy is critical, and processing time is not a constraint, such as detailed image analysis.

Justification for Choosing YOLO:

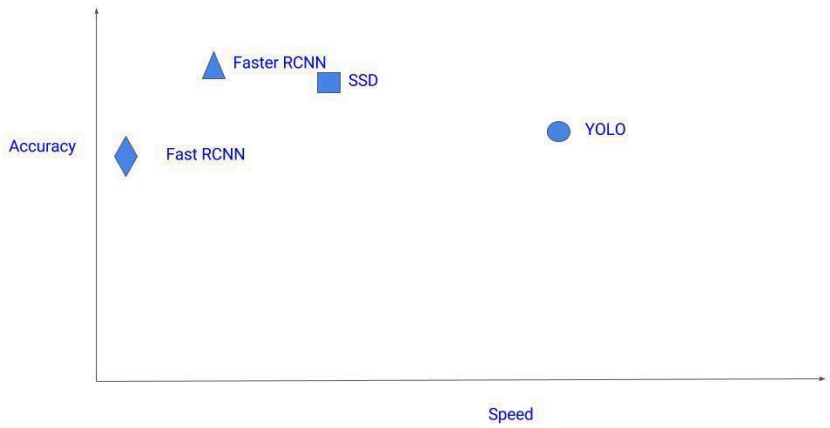


Figure 4.6: YOLO Justification

- **Real-Time Performance**: Essential for the robot's functionality, ensuring it can process video frames quickly and efficiently.
- **Simplicity and End-to-End Training**: YOLO's architecture allows for simpler integration and faster deployment.
- **Accuracy**: While not the highest, it provides a sufficient balance for the use case where real-time detection is more critical than the highest possible accuracy.

Why YOLOv3 is the Best Fit for Our Model

In our evaluation of various YOLO versions for the people recognition system, YOLOv3 emerged as the optimal choice due to several key factors. While newer versions like YOLOv4 and YOLOv5 offer improvements, YOLOv3 strikes the best balance for our specific needs.

1. Speed vs. Accuracy:

YOLOv3 provides an excellent trade-off between speed and accuracy. It is fast enough for real-time detection, crucial for our PeopleBot's dynamic environment, and accurate enough to detect people reliably.

YOLOv4 and YOLOv5, while offering marginally better accuracy and additional features, introduce increased complexity and computational requirements that are not essential for our use case. The slight improvements in detection accuracy do not justify the significant increase in resource demands and potential latency.

2. Robustness and Flexibility:

YOLOv3 is known for its robustness in various conditions. It performs well in detecting people in different lighting environments, handling occlusions, and recognizing various poses. These capabilities are vital for our system to function effectively in diverse campus settings.

Although YOLOv4 and YOLOv5 have advanced features like cross-stage partial networks (CSPNet) and optimized detection heads, these enhancements mainly benefit more complex and varied object detection tasks. For our focused application of people detection, YOLOv3's architecture is sufficiently robust and simpler to implement.

3. Community Support and Documentation:

YOLOv3 benefits from extensive community support and comprehensive documentation. This wide adoption means more available resources, tutorials, and troubleshooting guides, facilitating smoother integration and faster problem-solving during development.

YOLOv4 and YOLOv5, while gaining popularity, still lack the extensive community-backed resources that YOLOv3 enjoys. This can pose challenges, especially in debugging and optimizing the system in real-world scenarios.

4. Computational Efficiency:

Our hardware setup, including the processing units, aligns well with YOLOv3's computational demands. YOLOv4 and YOLOv5 require more

powerful GPUs and higher memory bandwidth, which could necessitate hardware upgrades and increase operational costs.

By using YOLOv3, we ensure that our system remains efficient and cost-effective without compromising on performance.

In conclusion, YOLOv3 provides the ideal balance of speed, accuracy, robustness, and ease of integration for our PeopleBot project. Its suitability for real-time applications and the extensive community support make it the best fit for our people recognition system.

Security Methods Considered:

- **RFID technology:** offers a straightforward and quick method for security verification. The primary advantage of RFID is its simplicity and ease of use, as it does not require a camera and can quickly verify identity through a physical card. However, RFID has notable disadvantages, such as the risk of the card being lost or stolen. This necessitates users to carry a physical card, which can be a security vulnerability if not managed properly.

- **Fingerprint recognition:** provides high accuracy and security, making it a reliable method for identity verification. The technology's primary advantage lies in its precision and difficulty to forge, ensuring a secure authentication process. However, fingerprint recognition requires physical contact, which can be less hygienic, especially in shared or public spaces. Additionally, it demands special hardware to capture and analyze fingerprints, which can increase the overall cost and complexity of the system.

- **Face recognition:** stands out as a non-intrusive method that can be performed at a distance, making it highly convenient and user-friendly. It integrates seamlessly with video surveillance

systems, allowing for continuous and unobtrusive monitoring. However, face recognition is susceptible to various factors that can affect its accuracy, such as changes in lighting, facial expressions, and occlusions like masks or glasses. These challenges require robust algorithms and high-quality cameras to ensure reliable performance under diverse conditions.

Justification for Choosing Face Recognition:

- Face recognition offers a non-intrusive and user-friendly method for verifying identities.
- It can be integrated with existing camera systems, reducing the need for additional hardware.
- Provides high security by accurately verifying individuals based on unique facial features.

Laser and sonar alternatives

Laser Sensor (LiDAR)

Precision and Accuracy: Highly accurate distance measurements, detailed maps with high resolution.

Real-Time Performance: Minimal latency, quick data processing for immediate responses.

Environmental Adaptability: Operates in various lighting and weather conditions, including fog and rain.

Complexity and Cost: Straightforward integration, relatively expensive but justified by reliability and accuracy.

Sonar Sensor

Precision and Accuracy: Uses ultrasonic waves, reliable in dark or foggy conditions.

Real-Time Performance: Quick updates and rapid processing of obstacle data.

Environmental Adaptability: Unaffected by dust, dirt, or lighting changes, suitable for various environments.

Complexity and Cost: Simple installation, minimal processing power required, cost-effective.

PTZ Camera

Precision and Accuracy: Affected by lighting conditions, shadows, and reflections, reducing accuracy.

Real-Time Performance: Relies on advanced, computationally demanding algorithms, slower response times.

Environmental Adaptability: Requires sufficient lighting, affected by weather conditions.

Complexity and Cost: Challenging setup and calibration, costly due to required image processing hardware and software.

Laser and sonar sensors are more suitable than PTZ cameras for reliable and efficient obstacle detection in autonomous robots, ensuring safe and effective navigation.

4.3 Relevant Engineering Applications and Calculations

The Peoplebot robot which integrates laser and sonar for obstacle detection, A* and reinforcement learning for routing, and camera-based people detection and face recognition, has a wide range of relevant engineering applications and calculations.

Firstly, the obstacle detection system requires precise measurements of distances using both laser and sonar sensors. We as engineers must consider factors such as sensor resolution, accuracy, and range to ensure reliable obstacle detection in various environmental conditions. Calculations involving signal processing techniques, such as time-of-flight or phase-shift measurements, are crucial for accurately determining obstacle distances. Additionally, engineers may need to perform calibration procedures to account for sensor inaccuracies and environmental factors like temperature and humidity.

For the routing algorithm, engineers utilize mathematical optimization techniques to find the most efficient path from the robot's current location to the goal destination. A* algorithm, for instance, involves heuristic evaluations of potential routes to guide the robot towards the goal while considering obstacles and terrain characteristics. Engineers must calculate heuristic values based on factors like distance to the goal, terrain elevation, and obstacle density to ensure optimal path planning. Similarly, reinforcement learning algorithms require mathematical models to estimate rewards and update the robot's policy over time based on its interactions with the environment. Calculations involving probability distributions, temporal difference learning, and reward functions play a crucial role in training the robot to navigate efficiently.

In terms of the camera-based security features, engineers employ image processing techniques and machine learning algorithms for people detection

and face recognition. Calculations involving image segmentation, feature extraction, and pattern recognition are essential for accurately identifying and tracking individuals in real-time. Moreover, engineers may need to optimize computational resources and algorithms to achieve fast and reliable performance, especially in resource-constrained environments like onboard processing on the robot. Additionally, calculations related to biometric authentication, such as face matching algorithms and similarity metrics, ensure robust security measures to verify the identity of individuals based on their ID photos. Overall, these engineering applications and calculations form the foundation of the Peoplebot robot, enabling it to perform its tasks effectively and efficiently in various real-world scenarios.

4.4 Description of the Final Design

Our final design integrates cutting-edge technologies such as A* algorithm, reinforcement learning (RL), routing, navigation, obstacle avoidance, and security features powered by face recognition using CV2 (OpenCV). This comprehensive approach ensures efficient and secure campus navigation within the engineering faculty.

Physical Design:

The robot boasts a sturdy chassis equipped with omnidirectional wheels, enabling smooth movement across various surfaces within the campus environment.

Sensors and Hardware:

- Laser and Sonar Sensors: These sensors enable precise obstacle detection and mapping, allowing the robot to navigate around obstacles in real-time
- Cameras: High-resolution cameras, integrated with CV2 (OpenCV), facilitate people detection and face recognition, enhancing campus security.

- Processing Unit: A powerful onboard processor processes sensor data and executes navigation algorithms efficiently, ensuring real-time decision-making capabilities.

Navigation and Routing:

- A* Algorithm: The robot utilizes the A* algorithm for optimal path planning, enabling it to find the shortest and safest routes to its destination while avoiding obstacles efficiently.

- Reinforcement Learning: Complementing A*, reinforcement learning algorithms continuously learn and adapt to environmental changes, optimizing navigation strategies over time for improved efficiency and adaptability.

Obstacle Avoidance:

The robot utilizes data from laser and sonar sensors to detect obstacles in its path and employs sophisticated algorithms to navigate around them, ensuring safe traversal through the campus environment.

Security Features:

- People Detection: The robot employs computer vision techniques to detect and classify individuals within its vicinity, ensuring constant monitoring of campus activity.
- Face Recognition: By comparing captured images with a database of authorized individuals, the robot verifies identities and alerts security personnel in case of unauthorized access or suspicious behavior.
- Real-time Alerts: Upon detecting suspicious individuals or unauthorized access attempts, the robot sends real-time alerts to security personnel, enabling prompt intervention and response.

Integration and Scalability:

- Modular Design: The robot's architecture is modular, allowing for easy integration of additional sensors, features, and functionalities to meet evolving campus needs.
- Scalable Deployment: Multiple robots can be deployed across the campus, working collaboratively to cover larger areas and provide comprehensive security and navigation support.

With its advanced features and intelligent algorithms, our final robot design represents a holistic solution for campus navigation and security within the

engineering faculty. By integrating A*, reinforcement learning, routing, obstacle avoidance, and face recognition using CV2, the robot ensures efficient and secure traversal while enhancing overall campus safety and operational efficiency.

Main Code Implementation for Security and People Recognition

The main code of the PeopleBot project integrates the functionalities of people recognition and security verification, ensuring that both systems operate seamlessly based on the time of day. The following sections describe the key components and processes involved in the implementation of the main code, guided by the system flowchart provided.

To balance between people recognition and security verification by dynamically switching between these functionalities based on the time of day. The system focuses on people detection

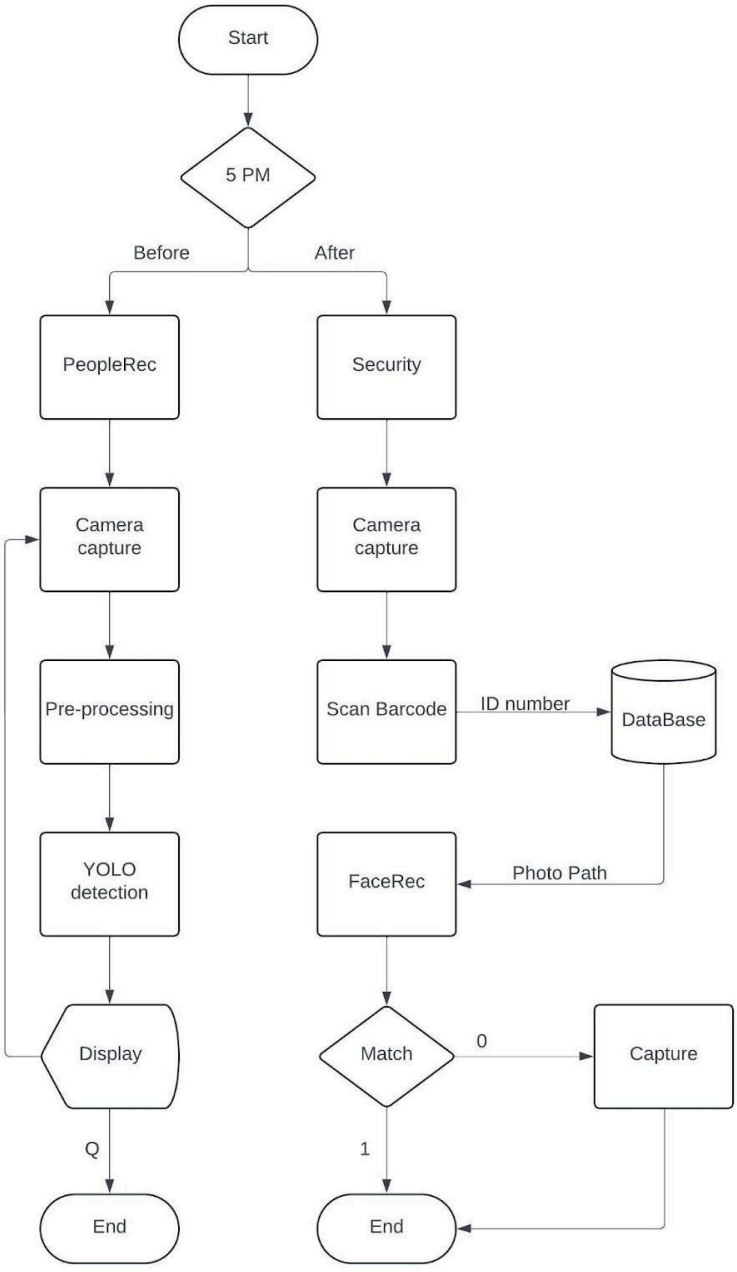


Figure 4.7: Main code flow chart

during the day and security verification in the evening.

Initialization

Load Necessary Libraries and Set Up Database Connection:

- Import libraries required for people recognition, barcode scanning, database operations, and face recognition.
- Establish a connection to the SQLite database containing member information.

Determine Time of Day

Switch Between People Detection and Security Verification:

- Get the current time and determine the appropriate functionality to run based on the time of day.
- People detection is prioritized before 5 PM, and security verification is prioritized after 5 PM.

People Recognition System Implementation

1. Initialization:

- **Load YOLO Model**:
    - Load the pre-trained YOLO model with its configuration and weights.
    - Set the model's input parameters and layer names.

2. Video Capture:

- **Capture Video Feed**:
    - Use the robot’s camera to capture a live video feed.
    - Resize the frame for processing and display.

3. YOLO Detection Preparation:

- **Prepare Frame for YOLO**:
    - Convert the frame into a blob suitable for neural network input.
    - Set the input to the YOLO network and perform a forward pass.

4. Processing Detections:

- **Extract and Filter Detections**:
    - Extract class IDs, confidence scores, and bounding box coordinates from the detection outputs.
    - Filter out weak detections and calculate bounding box coordinates.

5. Non-Maximum Suppression:

- **Apply Non-Maximum Suppression (NMS)**:
    - Eliminate redundant and overlapping bounding boxes using NMS.
    - Draw bounding boxes around detected people in the frame.

Security System Implementation

1. Initialization:

- **Establish Database Connection**:
    - Connect to the SQLite database containing member information.
    - Ensure the table for storing member details exists.

2. Barcode Detection:

- **Capture Video Feed**:
    - Use the robot's camera to capture a live video feed.
    - Overlay text on the video feed instructing users to show their ID card.
    - Continuously scan the video feed for barcodes using the Pyzbar library.
    - Decode the detected barcode to extract the ID number.

3. Database Query:

- **Query Member Information**:
    - Use the detected ID number to query the SQLite database.
    - Retrieve the member's information, including their photo path.

4. Face Recognition:

- **Load Known Image**:
    - Load the stored image of the member from the retrieved photo path.
    - Encode the known image using the face_recognition library.
- **Capture Video for Face Recognition**:
    - Continuously capture frames from the video feed for face recognition.
- **Detect and Encode Faces**:
    - Detect faces in the video frames and encode them.
- **Compare Faces**:
    - Compare the encoded face from the video feed with the known face encoding.
    - Determine if there is a match based on the similarity of the encodings.
- **Handle Match/No Match**:
    - If a match is found, display a confirmation message.
    - If no match is found within a specified duration, save the frame as an image for security purposes.

New ID design

The new ID card design was implemented to address the legal restrictions on collecting and using actual student IDs. This custom solution ensures compliance while providing a secure and efficient means of identity verification within the PeopleBot project.

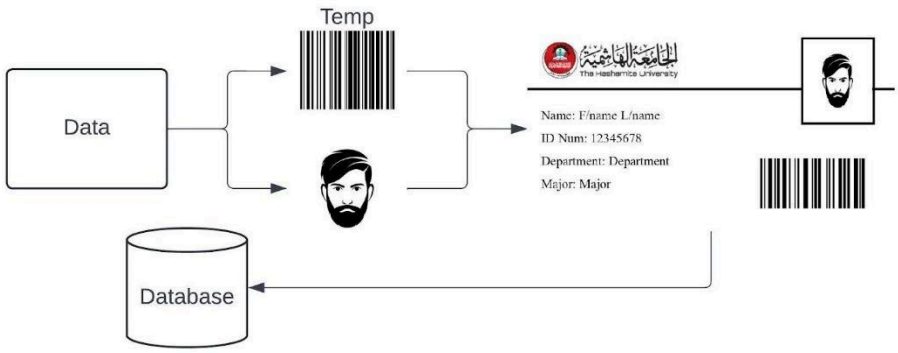


Figure 4.8: ID Design

Design Features

- **Unique Barcode System:** Each ID card is equipped with a unique barcode that encodes the student's ID number. This allows for quick and accurate scanning and retrieval of the student's information from the database.

- **Enhanced Security:** The barcode system minimizes the risk of unauthorized access by ensuring that only IDs with valid barcodes can be used for verification. This adds an extra layer of security to the overall system.

- **Consistency and Reliability:** The custom-designed ID cards ensure consistency in the format and placement of barcodes, reducing the chances of errors during scanning. This reliability is crucial for maintaining the system's efficiency and accuracy.

- **Ease of Use:** The ID cards are designed to be user-friendly, allowing students to easily present their cards for scanning. The process is quick, reducing wait times and improving the overall user experience.

- **Implementation:** A custom Python script was developed to automate the generation of these ID cards. The script creates a unique barcode for each student ID number and places it on the ID card template.

The process involves:

1. **Barcode Generation:** The script uses the python-barcode library to generate a unique barcode based on the student’s ID number.

2. **Card Template:** A standardized ID card template is used, which includes spaces for the student's name, photo, and the generated barcode.

3. **Automated Process:** The script automates the placement of the barcode onto the ID card template, ensuring that each card is generated consistently and accurately.

4. Storing the ID Data into the Database: The script stores the student’s information, including their ID number, name, and photo path, into a SQLite database. This ensures that each ID card's data is securely saved and can be retrieved efficiently for verification purposes.

The new ID card design offers several key benefits, ensuring compliance, security, efficiency, and scalability within the PeopleBot project. By designing our own ID cards, we ensure compliance with legal restrictions, avoiding potential issues related to the collection and use of actual student IDs. The unique barcode system enhances security by making it difficult to replicate or forge ID cards. Additionally, the automated generation process streamlines the production of ID cards, making it quick and easy to produce cards for a large number of students. Furthermore, the system is scalable, allowing for easy adaptation to different sizes of student populations without significant changes to the process.

id	first_name	last_name	id_number	photo_path
Filter	Filter	Filter	Filter	Filter
1	Islam	Alshourman	1935680	D:\Spring 2024\GP2\ID ...
2	Maryam	Shehadeh	1933756	D:\Spring 2024\GP2\ID ...
3	Tasneem	Barakat	2038234	D:\Spring 2024\GP2\ID ...
4	Dana	Shaqdih	1931939	D:\Spring 2024\GP2\ID ...
5	Bassam	Jamil	1178523	D:\Spring 2024\GP2\ID ...

Figure 4.9: IDs Dataset

5 CHAPTER 5: PROJECT REALIZATION AND PERFORMANCE OPTIMIZATION

5.1 Analysis and Optimization

5.1.1 A Star Integration with Reinforcement Learning

The combination of A* and Reinforcement Learning results in an adaptable navigation solution, merging the strengths of both methods to create a strong system suitable for real-world navigation challenges.

5.1.2 Testing the RL (Reinforcement Learning)

```python
def test_rl_agent(env, q_values, epsilon):
    total_rewards = 0
    num_episodes = 10 #1000

    for _ in range(num_episodes):
        state = env.reset()
        done = False

        while not done:
            action = env.choose_action(state, epsilon)  # Pass epsilon value
            print("Chosen action:", action)  # Print chosen action for debugging
            next_state, reward, done, _ = env.step(action)
            total_rewards += reward
            state = next_state

    avg_reward = total_rewards / num_episodes
    print("Average reward over {} episodes: {:.2f}".format(num_episodes, avg_reward))
```

Figure 5.1: test_rl code

For the RL code in the test function we measure the accuracy of the algorithm by measuring and printing the average reward.

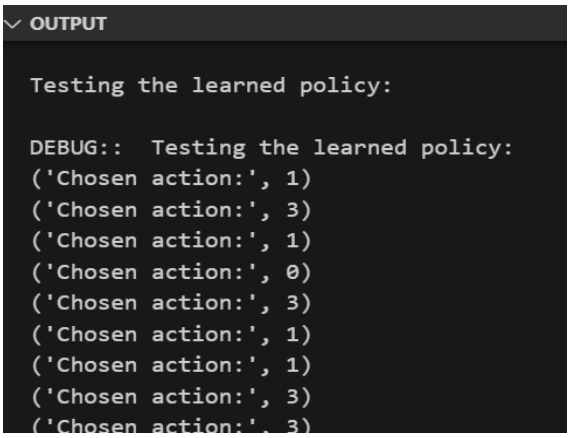


Figure 5.2: action output

Figure 5.3: Average reward

After the code finishes testing and choosing the write action then it prints the average reward.

5.1.3 Algorithms Integration with Laser and Sonar

The PeopleBot robot employs advanced routing algorithms along with laser and sonar sensors to effectively navigate through its surroundings. This integration is essential for avoiding obstacles and enabling the robot to discover optimal routes while avoiding collisions. Our project uses the ARIA library to facilitate seamless interaction between the robot's sensors and its pathfinding algorithms.

Obstacle Avoidance with ARIA and Laser

The ARIA library provides extensive support for integrating various sensors, including laser rangefinders, which are essential for precise obstacle detection and avoidance.

Laser Rangefinder Integration: The laser rangefinder provides accurate distance measurements, allowing the robot to identify obstacles at greater distances, which is essential for path planning and real-time adjustments.

ARIA Laser Connection: The laser is connected and interfaced with the robot using the ArLaserConnector class, enabling the robot to receive continuous updates about its surroundings, which are then used to inform its pathfinding decisions.

Real-Time Obstacle Avoidance: The laser data is processed to identify obstacles in the robot's path, and ARIA offers built-in actions such as ArActionAvoidFront, which apply this data to prevent collisions by adjusting the robot's course.

Obstacle Avoidance with ARIA and Sonar

Sonar sensors complement the laser rangefinder by providing additional proximity data, particularly useful for detecting nearby obstacles.

Sonar Sensor Integration: The **ArSonarDevice** class is employed to integrate sonar data into the robot's navigation system, as sonar sensors are adept at detecting objects at close range, complementing the laser's capabilities.

Together, laser and sonar sensors offer a comprehensive view of the robot's environment, ensuring more reliable obstacle detection and avoidance.

The foundation of our navigation system combines the A* algorithm with Reinforcement Learning (RL) to create a flexible and efficient pathfinding solution. In our implementation, the heuristic function $h(n)$ in the A* algorithm dynamically adjusts using Q-values derived from RL, enabling the

algorithm to make more informed decisions based on learned experiences, these experience takes into consideration the data received by the laser and sonar for obstacle detection, this is because of the obstacles play an essential role in determining the path in which the robot will navigate.

Obstacles are represented as cells with a value of one in the grid, ensuring consistency in how obstacles are handled both in the RL training phase and the A* algorithm.

The sensor data from laser and sonar are continuously integrated, enabling ARIA's obstacle avoidance actions to make real-time adjustments, ensuring the robot's safe navigation and routing in dynamic environments.

The integration of A* and RL with laser and sonar sensors through the ARIA library creates a powerful and adaptive navigation system for the PeopleBot robot. This setup ensures that the robot can find optimal paths while dynamically avoiding obstacles, making it well-suited for complex and dynamic environments. The combination of actual sensor data and flexible learning algorithms enhances the robot's efficiency and reliability, opening the door for advanced autonomous navigation applications.

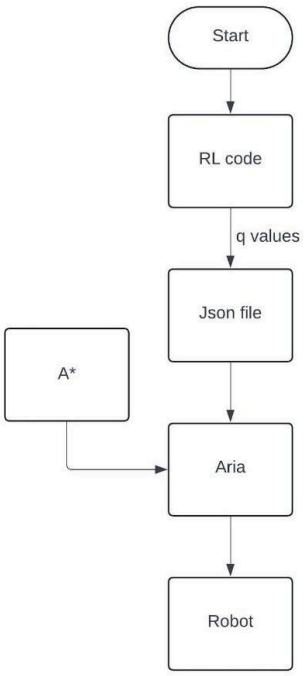


Figure 5.4: robot flowchart

5.1.4 People Recognition System

To ensure the system operates efficiently, extensive analysis and optimization processes were undertaken. Performance testing measured the system's latency and accuracy under different conditions, including varying lighting, background complexities, and crowd sizes. Initial tests revealed areas for improvement, such as optimizing image pre-processing and adjusting the confidence threshold for detections. For instance, brightness and contrast adjustments were implemented to improve detection accuracy in low-light conditions. Additionally, non-maximum suppression parameters were fine-tuned to better handle overlapping bounding boxes in crowded environments.

Integration testing involves end-to-end tests simulating real-world scenarios where people are detected, and their identities are verified. The system performed seamlessly, accurately detecting people without delays. Scalability testing evaluated the system's performance as the number of people increased. The system effectively handled larger crowds, maintaining accuracy and performance without significant degradation.

Results

Accuracy: High accuracy (>90%) in detecting and counting people.

Latency: Average latency of 300-500 ms per frame, depending on the environment.

Scalability: Effective in environments with up to 50 people without significant performance degradation.

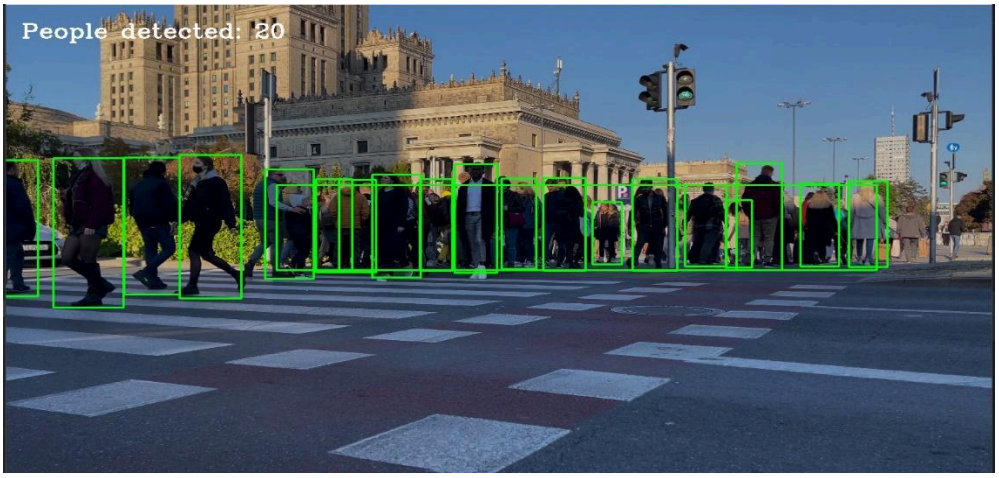


Figure 5.5: people detection

5.1.5 Security Verification System

To ensure the integrated system operates efficiently and reliably, extensive analysis and optimization processes were undertaken. Performance testing measured the system's latency and accuracy under different conditions, including varying lighting, background complexities, and face angles. Initial tests revealed areas for improvement, such as optimizing the face recognition parameters and improving the handling of edge cases where multiple faces are detected. Adjustments were made to enhance the system's ability to distinguish between authorized and unauthorized individuals, ensuring reliable identity verification.

The security system involves verifying identities through barcode scanning and face recognition. One significant challenge was the legal restriction on collecting student IDs, which led us to design our own ID card with a unique barcode system. The barcode encodes the student's ID number, which is used to retrieve the corresponding photo from the database for face recognition.

A custom Python script was developed to generate these ID cards. The script creates a unique barcode for each student ID number and places it on the ID card template, ensuring consistency and quick identification.

Integration testing involves end-to-end tests simulating real-world scenarios where people are detected, and their identities are verified. The system performed seamlessly, accurately verifying identities without delays. Scalability testing evaluated the system's performance as the number of verifications increased. The system effectively handled multiple verifications simultaneously, maintaining accuracy and performance. Security testing ensured the system reliably distinguished between authorized and

unauthorized individuals, achieving a match rate greater than 95% and a no-match rate less than 5%, by testing multiple thresholds.

Threshold = .5

Training Set			
TARGET / OUTPUT	Match	No match	SUM
Match	9 45.00%	6 30.00%	15 60.00% 40.00%
No match	0 0.00%	5 25.00%	5 100.00% 0.00%
SUM	9 100.00% 0.00%	11 45.45% 54.55%	14 / 20 70.00% 30.00%

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Match	0.6000	0.4000	1.0000	0.0000	0.7500
No match	1.0000	0.0000	0.4545	0.5455	0.6250
Accuracy	0.7000				

Threshold = .6

* The mismatch case was a test for side view face

Training Set			
TARGET / OUTPUT	Match	No match	SUM
Match	10 47.62%	0 0.00%	10 100.00% 0.00%
No match	1 4.76%	10 47.62%	11 90.91% 9.09%
SUM	11 90.91% 9.09%	10 100.00% 0.00%	20 / 21 95.24% 4.76%

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Match	1.0000	0.0000	0.8000	0.2000	0.8889
No match	0.8333	0.1667	1.0000	0.0000	0.9091
Accuracy	0.9000				

Threshold = .65

Training Set			
TARGET / OUTPUT	Match	No match	SUM
Match	8 40.00%	0 0.00%	8 100.00% 0.00%
No match	2 10.00%	10 50.00%	12 83.33% 16.67%
SUM	10 80.00% 20.00%	10 100.00% 0.00%	18 / 20 90.00% 10.00%

Figure 5.6: Threshold Test

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Match	1.0000	0.0000	0.9091	0.0909	0.9524
No match	0.9091	0.0909	1.0000	0.0000	0.9524
Accuracy	0.9524				

5.2 Methods of Realizing Final Design

Hardware Selection

Cameras: The PTZ cameras were chosen for their ability to cover wide areas with their panning, tilting, and zooming capabilities, enabling detailed and dynamic monitoring in real-time.

Sensors: Sensors, such as ultrasonic sensors, were integrated to aid in navigation and obstacle detection. These sensors provided accurate distance measurements and environmental mapping capabilities.

Software Development

Algorithm Implementation: Key algorithms for people recognition, security verification, and navigation were implemented using Python and integrated libraries such as OpenCV, Dlib, face_recognition, and Pyzbar. These libraries provided robust and efficient implementations of the required functionalities.

Data Management: An SQLite database was used to store member information, including ID numbers and corresponding face images. The database design focused on quick access and retrieval, ensuring minimal delays during the verification process.

System Integration: Software components were integrated to work seamlessly together. This involved developing a central control module that managed the workflow from people detection to security verification and

navigation. The control module coordinated the capture, processing, and display of video frames, as well as database queries and identity verification.

Navigation and Interaction

To enable the PeopleBot to navigate autonomously and interact with the environment effectively we have integrated:

Path Planning Algorithms:

Implemented path planning algorithms such as A* (A-star) to determine optimal routes. These algorithms were chosen for their efficiency in finding the shortest path in graph-based navigation scenarios.

Sensor Integration:

Ultrasonic sensors were used to detect obstacles and assist in navigating the environment in real-time.

Data from these sensors were processed to create an accurate representation of the surroundings, essential for effective path planning and obstacle avoidance.

Movement Control:

Movement control algorithms translate path planning decisions into smooth and precise movements.

These algorithms were integrated with motor controllers to ensure accurate execution of planned paths, maintaining stability in dynamic environments.

People Recognition System

To accurately detect and count people in various environments using the YOLO algorithm we have used:

Algorithm Selection and Implementation:

The YOLO (You Only Look Once) algorithm was chosen for its balance between accuracy and real-time performance. Implemented using the OpenCV library in Python, leveraging pre-trained YOLO models to avoid extensive training.

Pre-processing Optimization:

Frames captured from the PTZ cameras were resized and normalized to enhance detection accuracy. Adjustments to brightness and contrast were made dynamically based on real-time conditions.

Real-time Processing:

The YOLO model was kept in memory to minimize loading times, ensuring efficient frame processing in a continuous loop.

Detection results, including bounding box coordinates, were processed to identify and count people.

Post-processing and Display:

Bounding boxes were drawn around detected people, and the total count was displayed on the frame. The processed video frames were shown on the screen in real-time, providing immediate feedback.

Security Verification System

To ensure only authorized individuals gain access by verifying their identities through barcode scanning and face recognition we did:

Barcode Scanning Implementation:

Integrated the Pyzbar library for efficient barcode detection and decoding from video frames. Scanned ID cards to retrieve barcode data, which was then used to query a database.

Face Recognition Implementation:

Utilized the face_recognition library to compare the live video feed's captured face with stored photos. Implemented a mechanism to handle multiple face detections and ensure reliable matching.

Integration and Workflow Coordination:

Developed a central control module to manage the workflow from barcode scanning to face recognition and verification. Ensured seamless integration between capturing, processing, and displaying results, maintaining high accuracy and low latency.

ID Card Design:

Due to legal restrictions on collecting students' IDs, a custom ID card was designed specifically for this project. The custom ID card included a barcode that could be quickly and accurately scanned for security verification purposes

5.3 Sustainability

Sustainability is a critical aspect of maintaining a robot system. Continuous maintenance ensures the efficient functioning of the PeopleBot robot over its lifespan. Regular checks on mechanical components, such as motors, sensors and its camera, help identify issues early, preventing costly breakdowns and reducing the need for replacement parts.

Additionally, implementing predictive maintenance techniques, enabled by data collected from the robot's sensors, can further optimize maintenance schedules and minimize downtime. By prioritizing preventive measures, the system maximizes operational efficiency while reducing resource consumption and waste generation.

Another sustainability consideration lies in updating the mapping data used by the robot for navigation. As the campus environment evolves with new construction projects or changes in infrastructure, the accuracy of the mapping information becomes crucial for efficient route planning. Regularly updating the map data ensures that the robot can adapt to these changes and navigate the campus effectively.

Furthermore, incorporating crowdsourcing or collaborative mapping efforts involving students and faculty can enhance the accuracy and timeliness of map updates while fostering a sense of ownership and engagement within the campus community. By maintaining up-to-date mapping data, the system optimizes energy usage and reduces unnecessary detours, contributing to overall sustainability efforts.

Furthermore, integrating energy-efficient components and implementing power-saving strategies can reduce the environmental footprint of the robot system. Utilizing low-power sensors and processors, optimizing algorithms for energy efficiency, and implementing sleep modes during idle periods are effective ways to minimize energy consumption.

Additionally, exploring renewable energy sources such as solar panels for charging can further reduce the system's reliance on conventional energy sources. By prioritizing energy conservation measures, the robot system not only reduces its environmental impact but also lowers operating costs, making it more economically sustainable in the long run.

5.4 Testing and Improvement

5.4.1 Robot Testing

The experiments done on the PeopleBot robot are simulated using MobileSim, a virtual platform that allows control algorithms, behaviors, and software to be tested and improved without requiring physical hardware. Furthermore, MobileSim simulates the actions of a PeopleBot within an adaptable and dynamic virtual environment.

MobileSim's smooth integration with the ARIA library is a key component that significantly improves the PeopleBot robot's development and testing process. The connection with the ARIA library acts as a vital link and guarantees reliable performance when applied to the actual robot. The development process is greatly streamlined by this capability, which enhances the overall dependability and efficiency of robotic applications that use the ARIA library.

ARIA and ARNL: Robotics Libraries for Mobile Robots

One important feature of this documentation is the way that ARIA and ARNL are seamlessly integrated with MobileSim. This integration empowers developers by providing an ideal environment to test and refine their algorithms in a simulated setting before deploying them onto physical robots. This integration serves as an essential link as we will explore the features and capabilities of ARIA and ARNL, allowing us to thoroughly investigate these libraries in the context of simulated robotic environments.

ARIA (Advanced Robot Interface for Applications) and ARNL (Advanced Robot Navigation and Localization) are comprehensive robotics libraries developed by MobileRobots Inc. These libraries provide a powerful set of tools and functionalities for the control, navigation, and localization of mobile robots, including the PeopleBot platform. ARIA serves as the core interface for robot control, while ARNL extends its capabilities with advanced navigation and localization features.

ARIA (Advanced Robot Interface for Applications) is a comprehensive robotics library known for its key features designed for mobile robot control. First of all, ARIA gives developers access to high-level control features that make it easier to direct motions, examine sensor data in real-time, and interface with robot peripherals. The library's skill set also includes sensor integration, where it combines a variety of sensors—including sonar, laser rangefinders, and cameras—to create a strong basis for all-encompassing perception abilities. A notable benefit of ARIA is its adaptability, which facilitates the inclusion of unique modules and behaviors. Because of the design's flexibility, developers can modify the robot's functionality to suit particular needs.

ARNL (Advanced Robot Navigation and Localization) stands out with its additional features that increase robotic navigation capabilities. ARNL primarily uses advanced path-planning algorithms, which enable robots to navigate effectively in dynamic settings and skillfully avoid obstacles. Particle filters are one of the advanced localization techniques that help the robot become more spatially aware and accurately estimate its position in the surroundings. In addition, ARNL facilitates the generation and maintenance of maps, allowing robots to develop a thorough spatial awareness of their environment. Furthermore, the library makes it easier to complete tasks by

creating and carrying out movement and action sequences. Collectively, these advanced characteristics establish ARNL as a reliable option for improving localization, navigation, and general autonomous in mobile robotic systems.

In our project, ARIA and ARNL play an essential part in developing a smart control system for mobile robots. These libraries are essential to research and development since they are building blocks for developing, evaluating, and verifying innovative robotic algorithms. Furthermore, ARIA and ARNL are crucial resources for teaching and mastering complex robotics ideas in an educational setting. Through the use of mobile robots outfitted with ARIA and ARNL for tasks like autonomous material handling, tracking, and inspection, the project expands its influence into industrial applications. This demonstrates the created smart control system's versatility and industry relevance, making it a flexible solution for handling problems in mobile robots across the research and practical application domains.

ARIA and ARNL play a pivotal role in the development and deployment of mobile robotic systems. Their rich feature set, extensibility, and compatibility with the PeopleBot platform make them essential tools for researchers, educators, and engineers in the field of robotics. The integration with MobileSim further enhances their utility by providing a realistic simulation environment for testing and experimentation.

To seamlessly integrate MobileSim with the ARIA library, a systematic setup process ensures a conducive environment for developing and testing control algorithms for the PeopleBot robot. Begin by installing MobileSim following

the guidelines provided in the ARIA documentation. Once installation is complete, proceed to configure MobileSim by adjusting parameters related to robot specifications, sensor characteristics, and simulation environment properties. This step ensures that the simulation aligns precisely with the specific testing conditions required for the development and evaluation of control algorithms.

Subsequently, initiate simulations by executing MobileSim, providing a dynamic and virtual environment for testing control algorithms and behaviors. The versatility of MobileSim is further accentuated by its emulation of various sensors equipped on the PeopleBot, including sonar and laser range finders. This emulation facilitates realistic testing scenarios, particularly in sensor-based navigation and obstacle avoidance. As you embark on this setup journey, the integration of MobileSim with the ARIA library emerges as a vital component, offering developers a robust platform to refine and optimize control strategies before deploying them on the physical PeopleBot.

To simulate a robot, we need a map file containing lines for walls. A map file with lines can be created using a recent version of Mapper3 or Mapper3-Basic

Run MobileSim from a command line prompt by specifying a map file and robot model like this:

```
.\MobileSim.exe -m .\columbia.map -r peoplebot
```

to skip the initial dialog by specifying a map file and robot model to create on the command line

MobileSim -m columbia.map -r p3dx

The default robot model created, if -r is not given, is "p3dx".

Some additional robot models include:

"p3at", "powerbot", "peoplebot", "patrolbot-sh", "seekur", and "amigo".

To create multiple simulated robots, use -r multiple times, naming the robots after the model type, separated by a colon:

MobileSim -m columbia.map -r p3dx:robot1 -r p3dx:robot2 -r amigo:robot3

If you use -R instead of -r, then a new robot will be created on demand for each client that connects, and destroyed when the client exits. (So, the robot's state is not preserved.)

If you run MobileSim with no command line options, you may select the map and one robot model from a dialog box. Or you may choose to use no map (but note that in this case, the usable universe is limited to 200 x 200 meters -- gray area indicates the edge.)

Example maps are included with MobileSim.

On Linux, you can find these files by navigating from the root file system to

/user/local/MobileSim.

On Windows, these maps are in

C:\ProgramFiles\MobileRobots\MobileSim. columbia.map

Included with MobileSim is equivalent to columbia.map included with ARNL.

You can pan in the window by holding down the right mouse button, or mouse button with the Shift or Control key and dragging.

You can zoom with the mouse scroll wheel, or by holding down the middle mouse button or left mouse button with the Alt or Option key and dragging towards or away from the center of the circle that appears.

The robot may be moved by dragging it with the left mouse button and rotated by dragging it with the right mouse button, or while holding down the Shift or Control key. This will update the robot's true pose, but not its odometry. It is like picking up the robot off the ground and carrying it to a new location.

Once running, a program can connect to the simulator via TCP port 8101 and use the same protocol on that TCP port as it does with a real robot over its serial port link.

Any ARIA-based program that uses ArRobotConnector or ArTcpConnection will do this automatically. You can specify an alternate port number with the -p option. If multiple robots were requested with multiple -r options, each additional robot will use the next port number (i.e. 8102, 8103, etc.). For these additional robots, you will have to specify the port number when

running its ARIA program, usually using the -rrtp (-remoteRobotTcpPort) command-line option.

Mobilesim Integration with Python Script

MobileSim achieves seamless integration with Python scripts, empowering us to programmatically control and interact with the simulated PeopleBot. This integration is a fundamental aspect of the documentation, providing a streamlined pathway to implement, test, and refine robotic control logic within the simulated environment. Moreover, this integration is a cornerstone in the development and testing workflow, offering a versatile and programmable interface to explore and optimize control algorithms before transitioning to real-world implementation on the PeopleBot robot.

Testing the Reinforcement Learning Algorithm on the MobileSim

First, open the MobileSim simulator by navigating to the MobileSim file in the Program Files. Then, **open the Terminal** and enter the following command to run the PeopleBot Simulator on the specified map (in this example, we will use the Columbia map):

```
.\MobileSim.exe -m .\columbia.map -r peoplebot
```

Next, run the RL.py code, which contains reinforcement learning algorithm code. Ensure that the q-values are stored in the q_values.json file.

After that, run the Robot.py code, which will read the q_values.

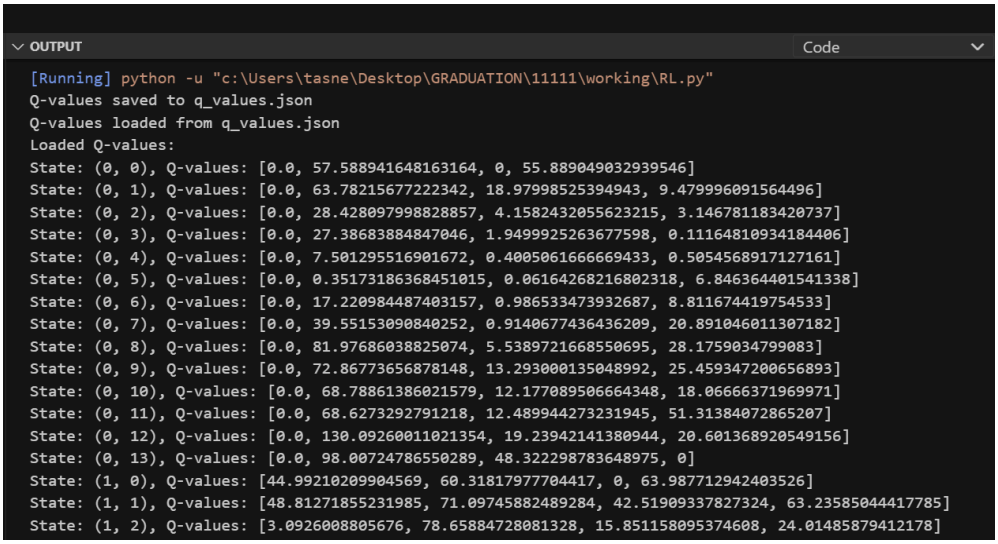


Figure 5.7: q-values result

This must be the output of the RL.py code, which indicates that the q_values are being calculated, saved to the json file and loaded to make a double check. These values are used to let the robot take the action based on the highest value of the four actions of movement.

The Robot.py code will connect to the A_Star.py code to select the optimal path for the robot and to reach the goal node.

Once the Robot.py code is running, it will connect to the simulator and send the correct actions to the robot to guid it to the specified goal node.

Additionally, the robot will use both laser and sonar for obstacle detection and collision avoidance.

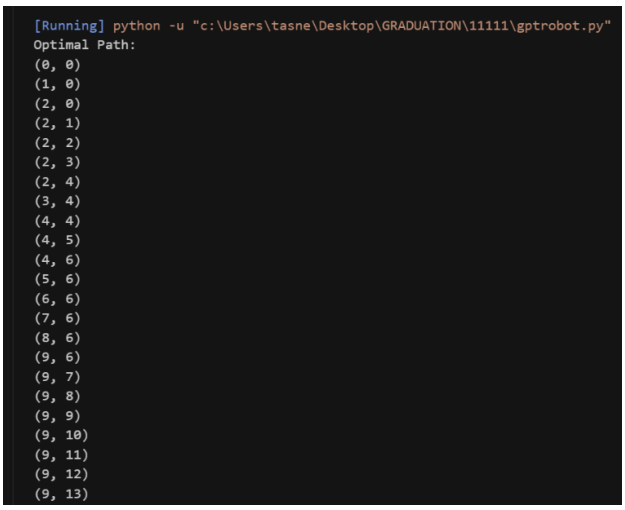


Figure 5.8: A* optimal path

This picture indicates that the A_Star.py code is working well and the A Star algorithm is finding the optimal path and printing the points in order to reach the goal node.

```
OUTPUT                                                    Code
(9, 12)
(9, 13)
Hey There...
Connecting to simulator through tcp.

Syncing 0
Syncing 1
Syncing 2
Connected to robot.
Name: MobileSim
Type: Pioneer
Subtype: peoplebot-sh
ArConfig: Config version: 2.0
Loaded robot parameters from C:\Program Files\MobileRobots\Aria\params/peoplebot-sh.p
ArRobotConnector: Connected to simulator, not connecting to additional hardware components.
Running...
ArLaserConnector: Auto parsing args for lasers
Using new style simulated laser for lms2xx_1
sim_lms2xx_1::setConnectionTimeoutSeconds: Setting timeout to 8 secs
sim_lms2xx_1::blockingConnect: Robot isn't running so can't wait for data
sim_lms2xx_1: Connected to simulated laser.
Connected to the robot. (Press Ctrl-C to exit)
x    0.0  y    0.0  th     0.0 vel     0.0 mpacs   0
x    0.0  y    0.0  th     0.0 vel     0.0 mpacs   0
x    0.0  y    0.0  th     0.0 vel     0.0 mpacs   0
x    3.0  y    0.0  th    -1.0 vel    34.5 mpacs   0
```

Figure 5.9: connecting to robot

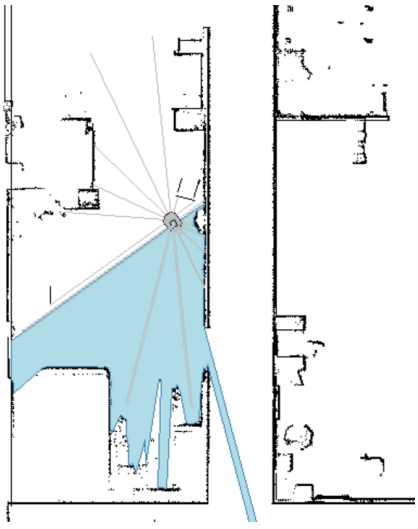


Figure 5.10: obstacle detection

These pictures show that the code is correctly connected to the MobileSim simulator. The robot is using its laser and sonar sensors for obstacle detection and avoidance, and it is moving towards the specified location.

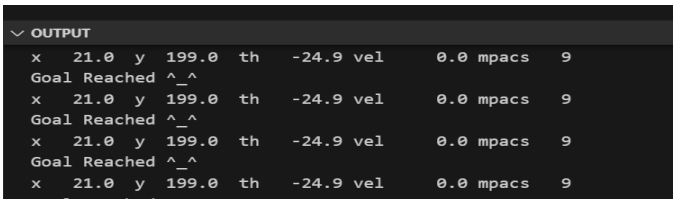


Figure 5.11: Goal Reached

Once the robot reaches its destination, it will stop and display the message

"**Goal Reached ^_^**".

5.4.2 More experiments

In this section, we view some of our most effective experiments that have been extremely beneficial for optimizing the robot movement, obstacle avoidance, and routing.

Navigation with sonar sensor Experiment:

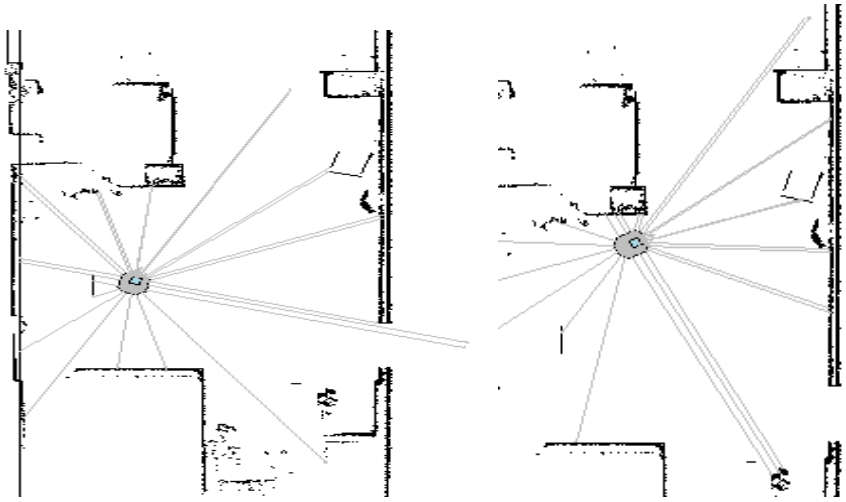
In this experiment, we evaluated the Peoplebot's navigation capabilities using the sonar sensors. The robot was equipped with an ArSonarDevice() to continuously scan the environment and detect obstacles, and it works by emitting ultrasonic pulses, measuring the time taken for the pulses to return, calculating the distance to the detected objects, and then providing real-time distance data to the robot's navigation system. To ensure effective obstacle avoidance, we employed several predefined actions. The ArActionStallRecover() was used to automatically resume when possible stalls occur and it works by detecting when the robot has stalled, initiating recovery actions, such as reversing or turning to avoid the obstacle, then continue normal operation once the stall is resolved, while the ArActionAvoidFront() enabled the robot to navigate around detected obstacles , and it works by continuously monitoring data from the front

sensors, detecting obstacles within a certain range, calculating a safe path around the obstacles, and then adjusting the robot's path to avoid collisions . To maintain consistent movement, we implemented the ArActionConstantVelocity(), which commanded the robot to move in a constant velocity, and it works by receiving a target velocity command, controlling the motor speed to achieve and maintain the specified velocity, then monitoring and adjusting the speed to ensure consistency.

This approach allowed the robot to navigate within the environment, relying on real-time sonar data for obstacle detection and avoidance, ensuring a balance between maintaining speed and avoiding collisions.

navigating :

*This map is the default map(columbia) on MobileSim.



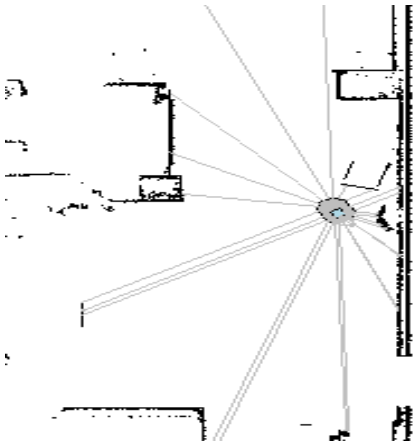


Figure 5.12b: Robot Navigation

output the exact position of the robot:

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
x  963.0  y  817.0  th   -48.0 vel   173.5 mpacs   9
x  977.0  y  801.0  th   -48.0 vel   207.5 mpacs   9
x  994.0  y  783.0  th   -48.0 vel   243.0 mpacs   9
x 1012.0  y  762.0  th   -48.0 vel   274.0 mpacs   9
x 1032.0  y  740.0  th   -48.0 vel   304.0 mpacs   9
x 1055.0  y  714.0  th   -48.0 vel   338.0 mpacs   9
x 1080.0  y  686.0  th   -48.0 vel   373.0 mpacs   9
x 1107.0  y  656.0  th   -48.0 vel   400.0 mpacs   9
x 1134.0  y  626.0  th   -48.0 vel   400.0 mpacs   9
x 1161.0  y  596.0  th   -48.0 vel   400.0 mpacs   9
x 1188.0  y  566.0  th   -48.0 vel   400.0 mpacs   9
x 1214.0  y  536.0  th   -48.0 vel   400.0 mpacs   9
x 1241.0  y  506.0  th   -48.0 vel   400.0 mpacs   9
x 1268.0  y  476.0  th   -48.0 vel   400.0 mpacs   9
x 1295.0  y  446.0  th   -48.0 vel   400.0 mpacs   9
x 1322.0  y  415.0  th   -48.0 vel   400.0 mpacs   9
x 1349.0  y  385.0  th   -48.0 vel   400.0 mpacs   9
x 1375.0  y  355.0  th   -48.0 vel   400.0 mpacs   9
x 1402.0  y  325.0  th   -48.0 vel   400.0 mpacs   9
x 1429.0  y  295.0  th   -48.0 vel   400.0 mpacs   9
x 1456.0  y  265.0  th   -48.0 vel   400.0 mpacs   9
x 1483.0  y  235.0  th   -48.0 vel   400.0 mpacs   9
x 1510.0  y  205.0  th   -48.0 vel   400.0 mpacs   9
x 1536.0  y  175.0  th   -48.0 vel   400.0 mpacs   9
x 1563.0  y  145.0  th   -48.0 vel   400.0 mpacs   9
x 1590.0  y  115.0  th   -48.0 vel   400.0 mpacs   9
x 1617.0  y   85.0  th   -48.0 vel   400.0 mpacs   9
x 1644.0  y   55.0  th   -48.0 vel   400.0 mpacs   9
x 1671.0  y   25.0  th   -48.0 vel   400.0 mpacs   9
x 1697.0  y   -5.0  th   -48.0 vel   400.0 mpacs   9
x 1724.0  y  -36.0  th   -48.0 vel   400.0 mpacs   9
x 1751.0  y  -66.0  th   -48.0 vel   400.0 mpacs   9
x 1778.0  y  -96.0  th   -48.0 vel   400.0 mpacs   9
x 1805.0  y -126.0  th   -48.0 vel   400.0 mpacs   9
x 1831.0  y -156.0  th   -48.0 vel   400.0 mpacs   9
x 1858.0  y -186.0  th   -48.0 vel   400.0 mpacs   9
```

Figure 5.13: X-Y position

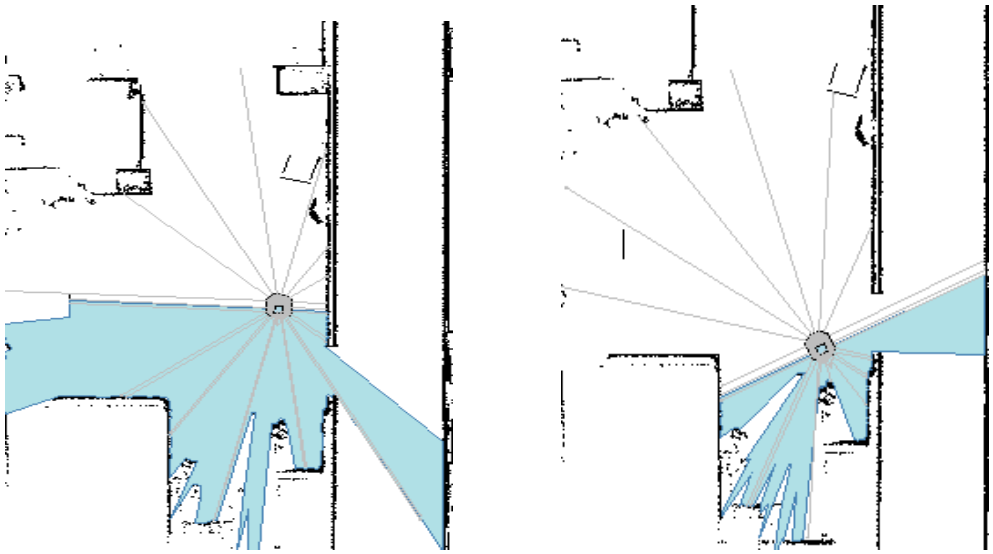
Navigation with Obstacle Detection Using Sonar and Laser Experiment

In this experiment, we focused on evaluating the Peoplebot's obstacle detection and avoidance capabilities by using both sonar and laser sensors. The robot was equipped with an ArSonarDevice(), We've already covered how it works . And an ArLaserConnector(), and it works by emitting a laser beam, detecting the reflection of the beams from obstacles, measuring the time of flight of the laser beams, calculating the precise distance to the obstacles, and then providing detailed and accurate distance data to the robot's navigation system. These two actions provide comprehensive environmental sensing. Then to navigate safely, we implemented several actions. The ArActionStallRecover() , allows the robot to recover from stalls, and We've already covered how it works, while the ArActionAvoidFront() enables dynamic avoidance of obstacles detected in its path, and We've already covered how it works. Then to maintain a consistent speed, we used ArActionConstantVelocity and We've already covered how it works. For additional safety, ArActionLimiterForwards() , and it works by continuously monitoring the distance to obstacles in front of the robot, adjusting the maximum forward speed based on the proximity of obstacles, to ensure that the robot slows down or stops if it gets too close to an obstacle. And ArActionLimiterBackwards() , and it works by continuously monitoring the distance to obstacles behind the robot, adjusting the maximum backward speed based on the proximity of obstacles, to ensure that the robot slows down or stops if it gets too close to an obstacle while reversing. So these actions were employed to limit the robot's forward and backward speeds, respectively.

This combination of sonar and laser sensors, along with the predefined actions, enabled the Peoplebot to effectively detect and navigate around obstacles in real-time, ensuring smooth and safe operation within the environment.

starting the robot :

*This map is the default map(columbia) on MobileSim.



Detecting an obstacle and avoiding it:

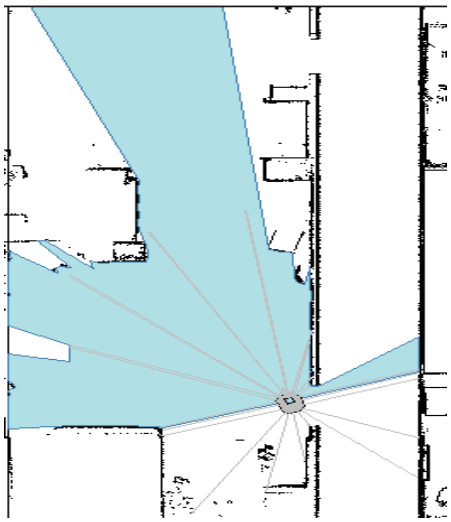


Figure 5.14: Obstacle Avoidance

Output the exact position of the robot:

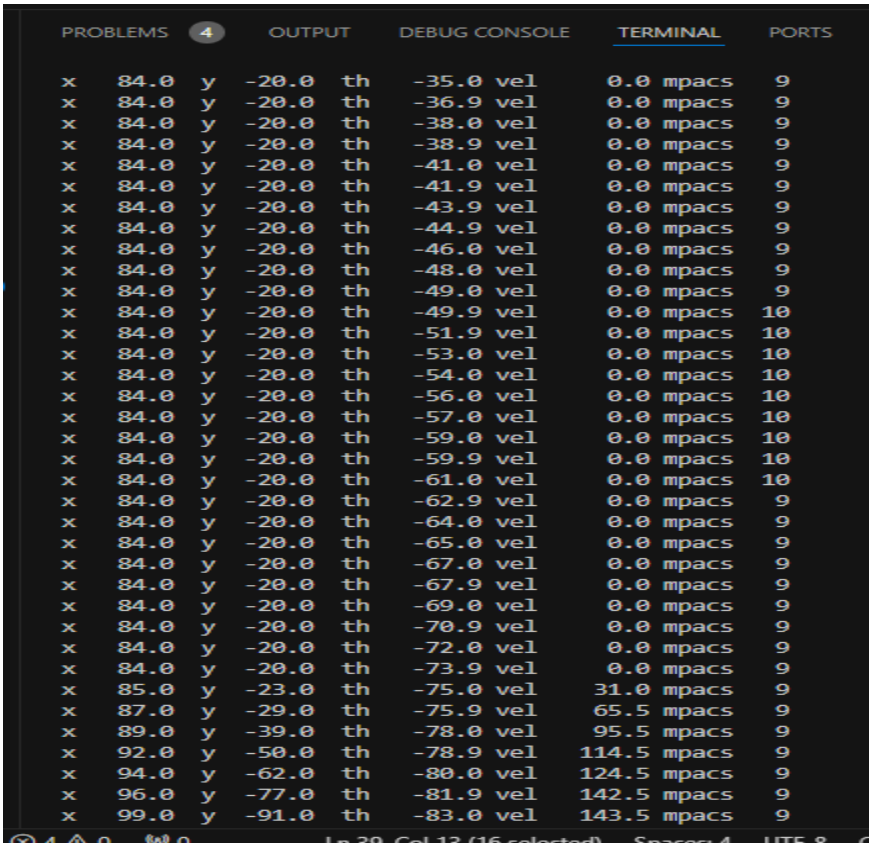


Figure 5.15: robot position

User Goal Input Experiment

This experiment shows the integration of the AriaPy library, a ready library in the robot, to navigate a robot toward a specific goal node, the goal node in which the robot navigates through the ArPose which defines the goal coordinates and the ArActionGoto that directs the robot to move towards the specified goal, with its status repeatedly monitored by the PrintingTask class where robot's status, position, orientation, velocity, is shown. The robot successfully reaches the goal, showing accurate performance with minimal

error. Real-time feedback provided by PrintingTask is crucial for debugging, and the simulated environment (MobileSim) offers a safe testing ground before real-world application.

Starting Position of The Robot:

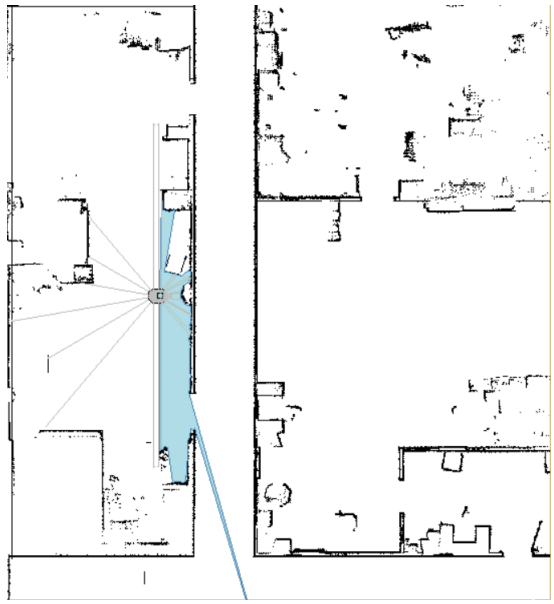


Figure 5.16

First Time Goal Reached:

Second Time Goal Reached:

Figure 5.17

```
Goal Reached ^_^
x  249.0  y  520.0  th   137.6 vel   165.5 mpacs   9
```

In this experiment, the code successfully navigates the robot to the specified goal node (100, 500) in the simulated environment, with output elements detailing the robot's position (x, y), orientation (th), velocity (vel), and motor packet count (mpacs). In multiple runs, the robot reaches positions such as (158, 590) and (249, 520), indicating alteration from the exact goal coordinates of (100, 500). These alterations, resulting in the robot stopping approximately 20 to 149 units away from the goal, highlight potential influences of the absence of routing algorithm, environmental factors, or control movement limitations. However, the robot's consistent ability to approach the goal closely while avoiding obstacles shows reliable performance and suggests that further accuracy could enhance precision.

The robot successfully avoids obstacles and reaches the goal, demonstrating reliable navigation. The provided logs are useful for monitoring the robot's status and performance, ensuring accurate goal-reaching behavior. Further analysis can include more complex environments and real-world testing to validate the system's robustness, and these results help in finding ways for optimizing and enhancing the accuracy of the robot.

A* Routing Experiment

In this experiment, the peoplebot is meant to route from the start node to the goal node shown in figure (5.18). The figure shows the map of the Robotics Lab where we have held our experiments, the map was previously built in the robot. Every node represents a tile, so in the A Star Routing code a 10x14 grid map was built, every cell in the grid map represents a tail in the actual map of the lab where the robot is supposed to navigate.

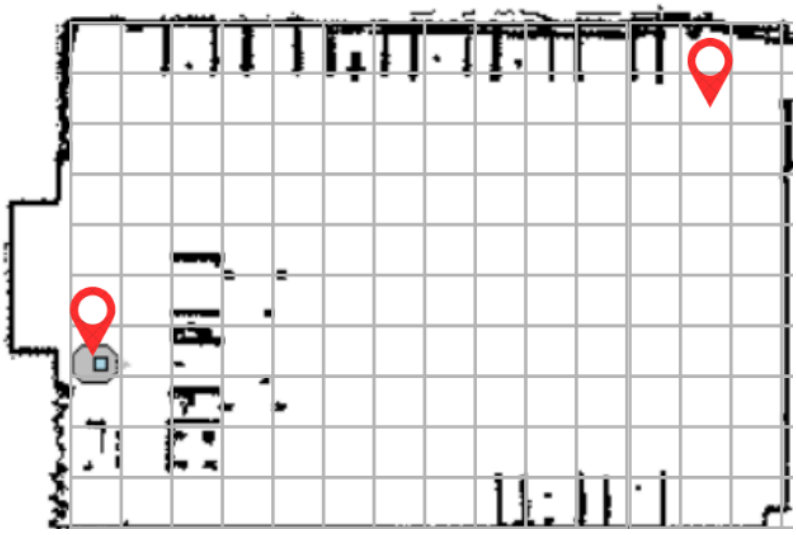


Figure 5.18: Robotics Lab

The A Star is supposed to deliver the robot to the goal node by giving the optimal path, and then control the robot movement in each node in order to reach the goal node. Figure (5.19) shows the optimal path and the movements taken to reach the goal node.

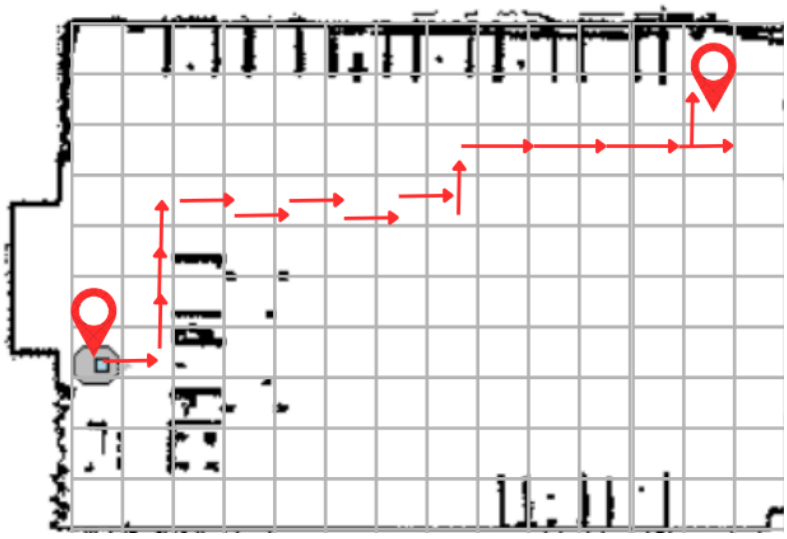


Figure 5.19: A Star Path Planning

Updating The Map Experiment

Using the drawing code on the MobileSim, we see the result on MobileEyes Simulator, where the drawing code will update the map by adding dots to the map when the robot detects an obstacle.

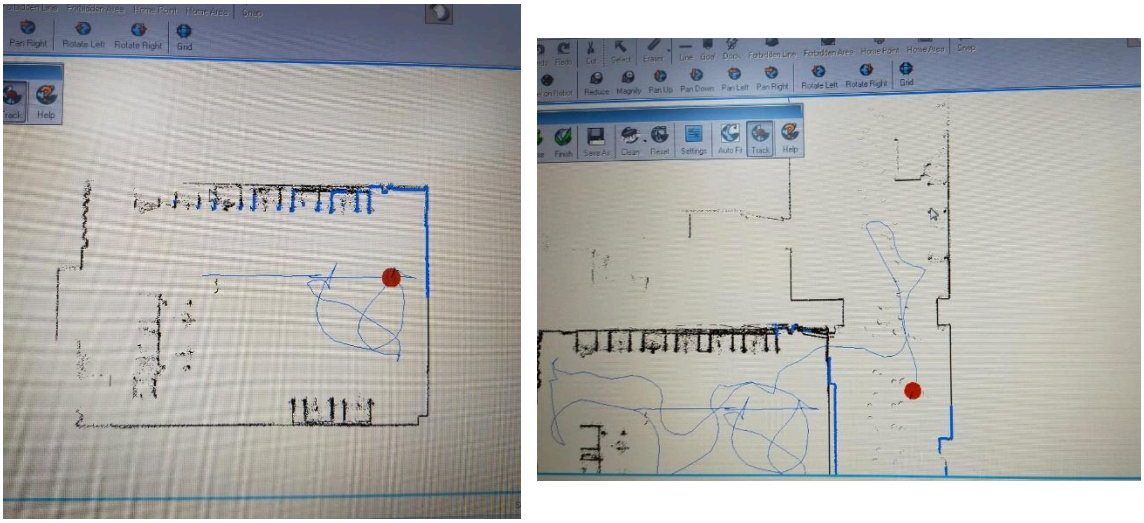


Figure 5.20: Updating Map

In these pictures we ran the mapper on the Peoplebot using the lab's map, we detected the robot movement on the simulator for a static environment. It was useful to understand the robot movement during exploration.

5.4.2 People Recognition System Testing

Experimental Setup and Results

Methodology

1. **Data Collection**: Collected video clips with various numbers of people under different lighting conditions, angles, and backgrounds.

Figure 5.21: video clips

2. **Experimental Setup**: Set up the YOLO model with specific parameters for detection. This part included removing the detection of objects other than people to focus mainly on people, optimizing computational overhead as much as possible.

Figure 5.22 people detection

3. **Testing Procedure**: Conducted tests with different crowd densities, lighting conditions, and background complexities.

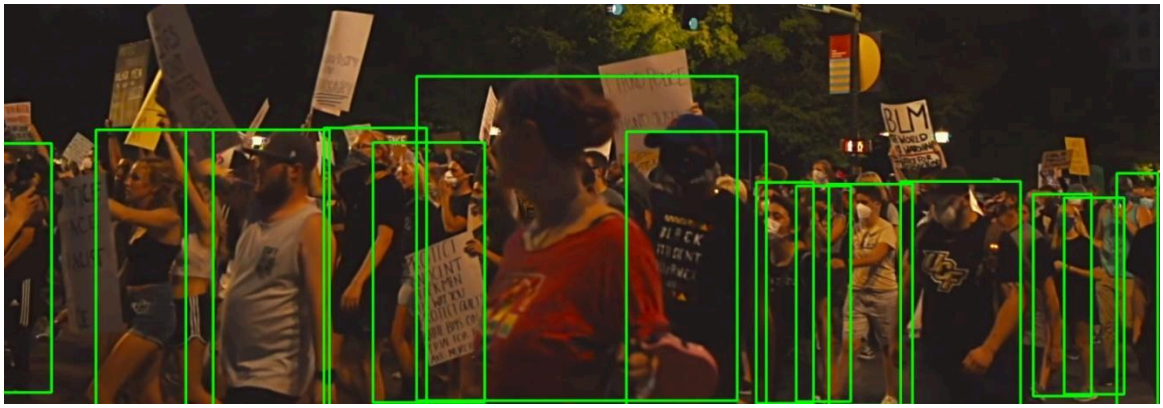


Figure 5.23: Testing Procedure

- **Functional Testing**:
    - **Objective**: Ensure the YOLO-based people recognition system detects people accurately.
    - **Method**: Tested the system with various video feeds containing different numbers of people and varying conditions.
    - **Outcome**: The system consistently detected people with high accuracy, but struggled slightly in low-light conditions.
- **Integration Testing**:
    - **Objective**: Verify the integration of the people recognition system with the rest of the robot's functionalities.

- **Method**: Combined the people recognition system with other components like video capture and display functions.
- **Outcome**: The system operated seamlessly, providing real-time detection and display.

- **Stress Testing**:
  - **Objective**: Evaluate the system's performance under continuous and high-load conditions.
  - **Method**: Ran the system continuously for extended periods and with multiple people in the frame.
  - **Outcome**: Ensured system stability and identified potential performance bottlenecks.

Errors and Solutions

1. **Low Detection Accuracy in Low Light**:

    - **Description**: The system struggled to detect people accurately in low-light conditions.

    - **Resolution**: Improved image pre-processing by adjusting brightness and contrast before detection.

frame = cv2.convertScaleAbs(frame, alpha=1.2, beta=50)

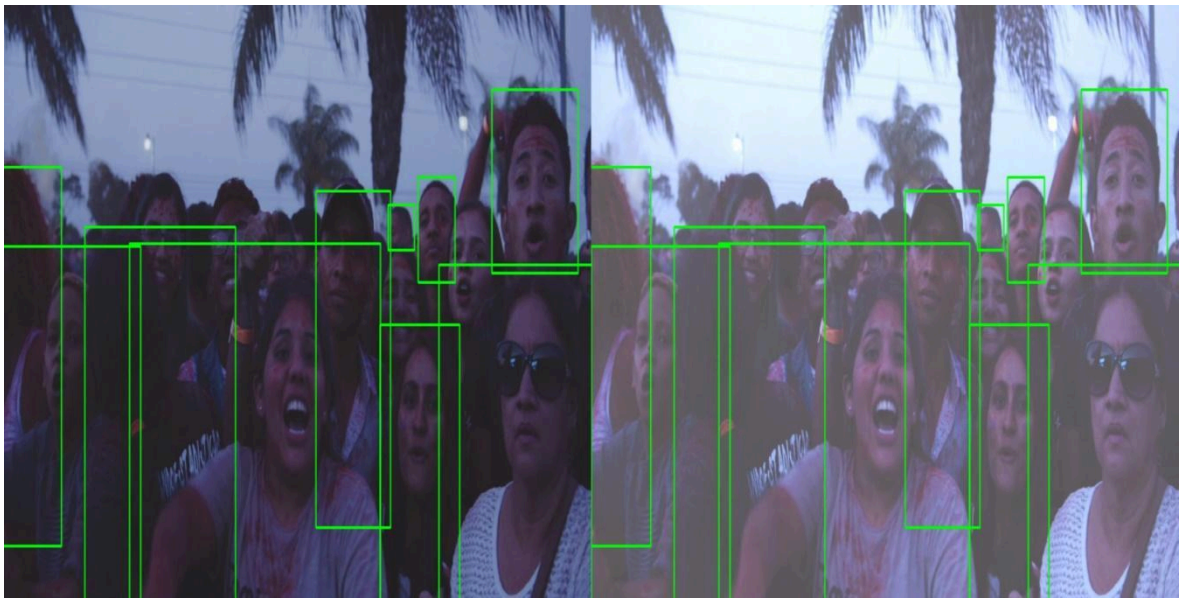


Figure 5.24: Detection in low light

2. **Multiple People Overlapping**:

   - **Description**: Overlapping people in the video feed caused detection errors.

   - **Resolution**: Adjusted non-maximum suppression parameters to better handle overlapping bounding boxes.

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.3, 0.4)

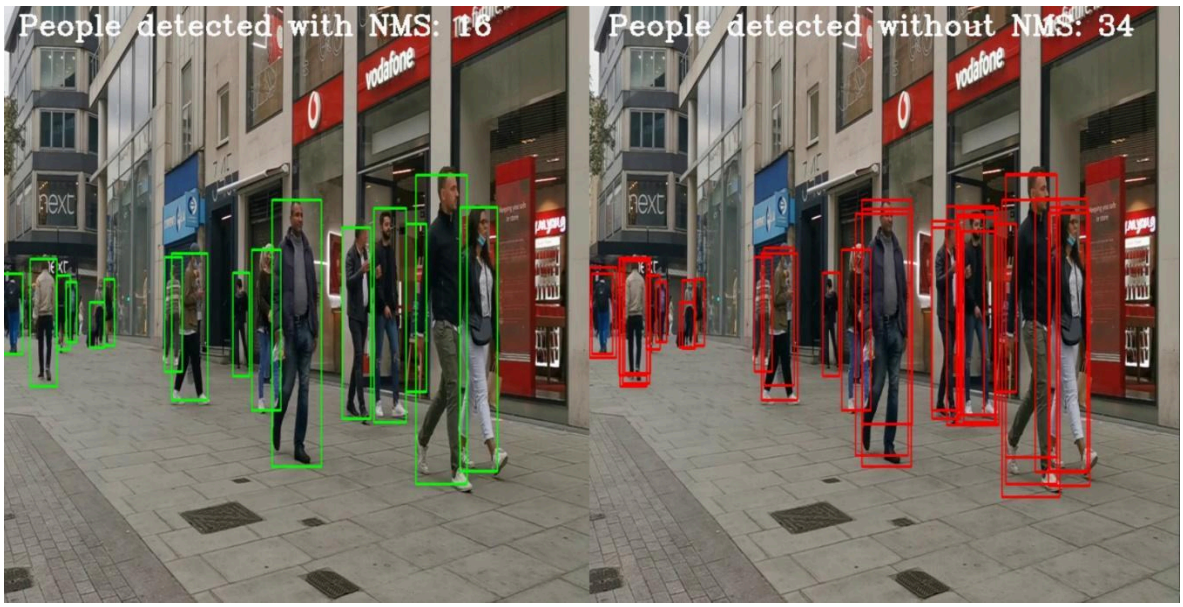


Figure 5.25: Detection in low light

3. **False Positives**:

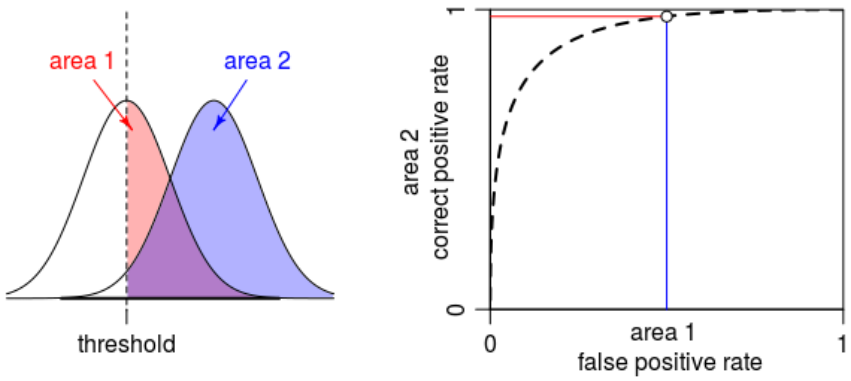


Figure 5.26: Threshold and false positive

- **Description**: The system occasionally detects non-human objects as people.

- **Resolution**: Increased the confidence threshold for detections.

if confidence > 0.6

Choosing a threshold of 0.6 provides the optimal balance between accuracy and minimizing false positives. This threshold effectively identifies true positive detections while significantly reducing the number of incorrect identifications. As a result, it enhances the reliability and precision of the object detection system, ensuring that the detections are both accurate and meaningful. By carefully selecting this threshold, we achieve a robust performance that excels in practical applications, reducing the likelihood of false alarms and improving overall detection quality.

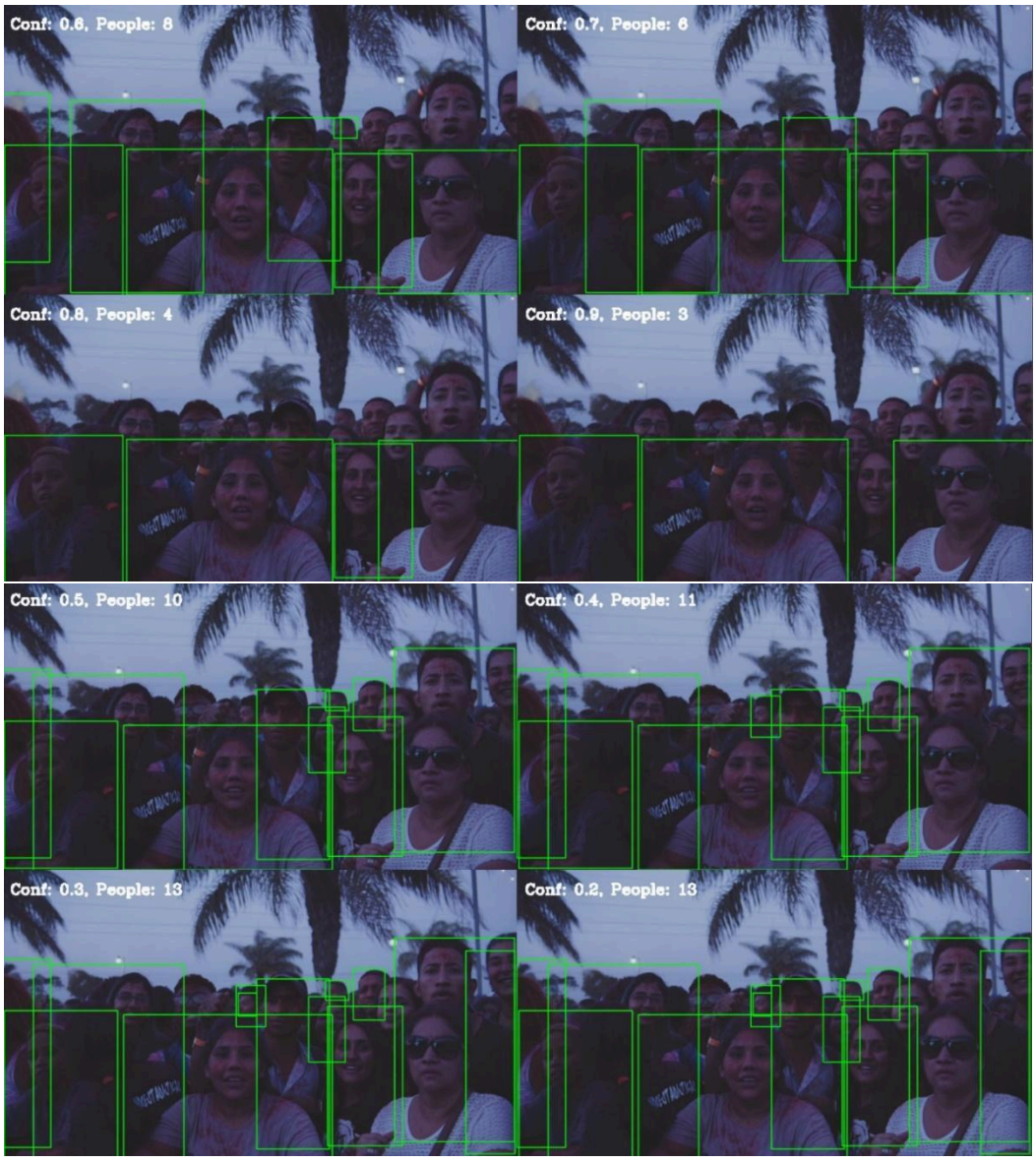


Figure 5.27: Testing on different confidence

5.4.3 Security System Testing

Functional Testing

Objective: Ensure all functions work as intended from face detection to encoding and comparison.

Method: Tested individual functions with a variety of images, including edge cases such as:

- Different lighting conditions

Figure 5.28

- A person aging over time

Figure 5.29

- With and without glasses

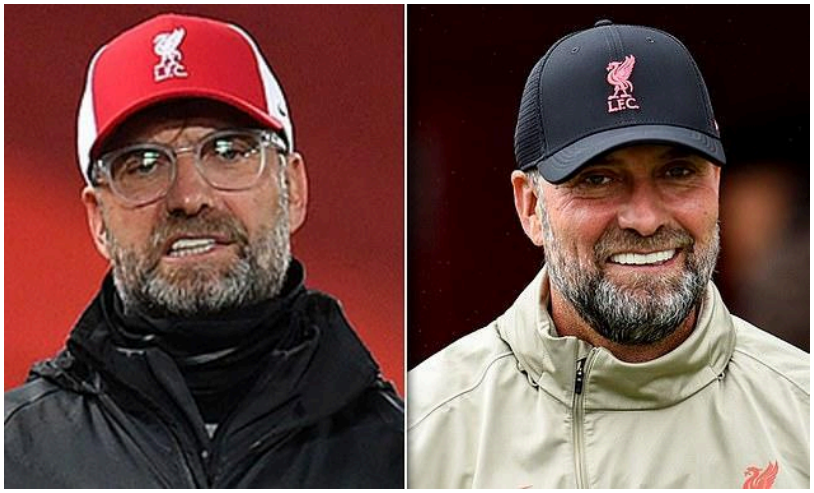


Figure 5.30

- With and without a beard

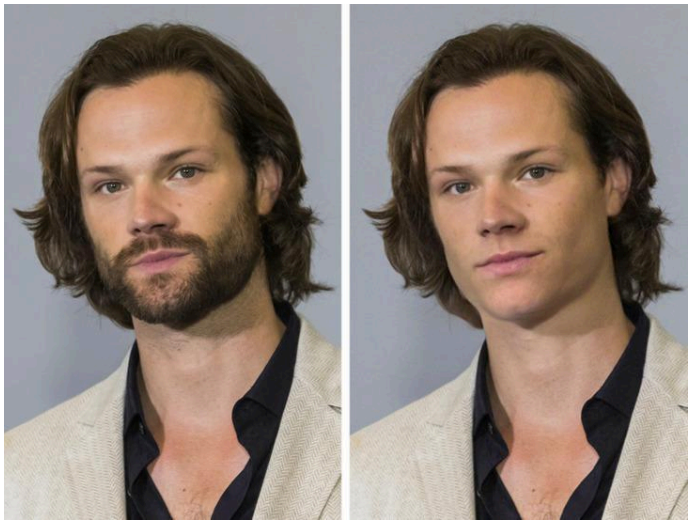


Figure 5.31

Outcome: The functions were refined to handle various scenarios, ensuring robust performance. However, they still struggled with significant age differences.

Integration Testing

Objective: Verify that all components of the face recognition system work seamlessly together.

Figure 5.32: Integration Testing

Method: Combined face detection, encoding, and comparison functions, and tested with live video feeds.

Outcome: Identified and resolved issues related to real-time processing and integration.

Performance Testing

Objective: Measure the system's response time and accuracy under different conditions.

Method: Conducted tests with varying lighting conditions, background complexity, and face angles.

Outcome: Optimized processing speed and improved accuracy by fine-tuning parameters (e.g., tolerance levels).

Security Testing

Objective: Ensure the system reliably distinguishes between authorized and unauthorized individuals.
Method: Tested with images of authorized personnel and various distractor images to check for false positives and negatives.
Outcome: Achieved high accuracy by setting an appropriate tolerance threshold and implementing duration-based decision logic.

Errors and Solutions

1. No Faces Found in Known Image

- **Description:** The system sometimes failed to detect faces in the known image.
- **Resolution:** Added a check to ensure at least one face is found before proceeding with encoding.

2. Multiple Faces Detected

- **Description:** In some instances, multiple faces were detected in the known image, causing ambiguity in encoding.
- **Resolution:** Ensured only one face is used by selecting the first face detected and prompting users to provide a single-face image if necessary.

3. Video Frame Not Captured

- **Description:** Occasional failures in capturing video frames led to issues in real-time face detection.
- **Resolution:** Added a check to skip processing if a frame is not successfully captured.

4. No Match Found Consistently

- **Description:** The system sometimes failed to find a match consistently, causing false rejections.
- **Resolution:** Introduced a mechanism to accumulate match results over a period before deciding on a match or no match.

5.5 Health, Safety, Quality and Reliability

Operational Safety

Fail-Safe Mechanisms

Incorporating robust fail-safe systems in the PeopleBot's programming to automatically halt operations in case of a malfunction or a critical error. This includes emergency stop functions, fault detection algorithms, and safe power cut-off mechanisms, ensuring the robot can safely handle unexpected issues without causing harm or damage.

Collision Avoidance Systems

Implementing advanced collision detection and avoidance technologies to prevent accidents, particularly in environments with human interaction. Utilizing the PeopleBot's array of sensors, such as LIDAR, infrared, and ultrasonic sensors, to continuously monitor its surroundings and make real-time adjustments to its path, significantly reducing the risk of collisions.

Regular Maintenance and Testing

Establishing a rigorous maintenance schedule to ensure all components are functioning correctly. This includes routine inspections, software updates, and hardware testing to identify and rectify potential safety hazards before they lead to operational failures. Regular diagnostics and calibration of sensors and cameras are also essential to maintain optimal performance.

User Training and Guidelines

Providing comprehensive training for users and operators, focusing on safe operational practices, understanding of the Peoplebot's functionalities, and actions to take in case of emergencies. Clear user manuals, training sessions, and ongoing support will enhance user competence and safety awareness.

Data Security

Encryption and Secure Transmission

Ensuring that all data transmitted by the PeopleBot, especially visual data captured by the camera, is encrypted. Implementing secure transmission protocols, such as SSL/TLS, to protect against unauthorized access and data breaches, ensuring the privacy and security of sensitive information.

Access Control and Authentication

Establishing strict access controls for the system, including multi-factor authentication and role-based access permissions, to prevent unauthorized use or tampering with the PeopleBot. Only authorized personnel should be able to access and control the robot, ensuring secure and responsible operation.

Data Storage and Retention Policies

Adhering to best practices for data storage, including secure handling of sensitive information, implementing data retention policies in compliance with legal requirements, and ensuring that data is only stored as long as necessary. Regular audits and data purging protocols help maintain data integrity and privacy.

Regular Cybersecurity Audits

Conducting periodic security assessments and audits to identify and mitigate vulnerabilities in the system, including software, firmware, and communication channels. Staying up-to-date with the latest security threats and updates ensures the PeopleBot remains resilient against cyber-attacks.

Compliance with Safety Regulations

Adherence to Regulatory Standards

Ensuring compliance with all relevant safety standards and regulations specific to robotics and electronic devices. This includes meeting industry-specific safety certifications, such as ISO 13849 for safety of machinery and IEC 61508 for functional safety, and conducting regular compliance reviews. Adhering to these standards not only ensures the safety and reliability of the PeopleBot but also fosters trust and acceptance among users and stakeholders.

By prioritizing these health, safety, quality, and reliability considerations, the PeopleBot project aims to create a robot that is not only functionally effective but also safe to operate, secure in data handling, and responsible in its environmental impact. These measures are crucial in fostering trust and acceptance among users and stakeholders, ensuring the PeopleBot meets the highest standards of safety and performance.

5.6 Performance Evaluation

5.6.1 Routing Algorithms performance

A* Algorithm

The A* algorithm is known for its efficiency and accuracy in pathfinding tasks. Its performance in this project was evaluated based on its ability to find the shortest path from a start node to a goal node within a predefined grid environment. Key metrics include:

Path Optimality: The A* algorithm consistently found the shortest path, validating its heuristic-based approach.

Computation Time: The algorithm demonstrated quick computation times, essential for real-time navigation.

Reinforcement Learning (RL)

The RL algorithm was trained to navigate the Peoplebot through a grid world, learning optimal paths through repeated interaction with the environment. The evaluation focused on:

Learning Efficiency: The RL agent showed progressive improvement in finding optimal paths, with a significant reduction in exploration time after training.

Adaptability: The RL algorithm adapted well to dynamic changes in the environment, showcasing its potential for real-world applications where conditions are not static.

5.6.2 Obstacle Detection performance

The obstacle detection system utilizes laser and sonar sensors to identify and avoid obstacles. Performance metrics for this component include:

Detection Accuracy: Both laser and sonar sensors provided high accuracy in detecting static and dynamic obstacles, with minimal false positives.

Real-time Processing: The algorithms processed sensor data in real-time, allowing the Peoplebot to navigate smoothly and avoid collisions effectively.

Robustness: The system proved robust in various environmental conditions, maintaining reliable performance indoors and outdoors.

5.6.3 People Recognition & security System Performance

Accuracy: Achieved high accuracy in detecting people and face recognition by fine-tuning the model parameters and improving pre-processing steps. Reduced false positives and improved detection in low-light conditions.

Processing Time: Optimized the processing speed by resizing frames and adjusting processing frequency.

Reliability: Ensured system reliability through rigorous testing under various conditions and edge cases.

6 CHAPTER 6: ECONOMIC, ETHICAL, AND CONTEMPORARY ISSUES

6.1 Final Cost Analysis

The Final cost for the PeopleBot project is effectively **zero** because of the Robot Vision Research Lab in our Computer Engineering Department.

Robot Vision Research Lab

Supported by a 110,000 JD grant from the Deanship of Scientific Research at the Hashemite University, Zarqa, Jordan, the Computer Engineering (CPE) department, through its professors **Dr. Khalil Yousef** (principal investigator) and **Dr. Bassam Mohd**, has established an advanced robot vision research lab. This lab, unique among Jordanian universities, is equipped with state-of-the-art mobile and humanoid robots and a variety of sensors, including PTZ cameras, stereo cameras, RGB-D sensors, laser range finders, and a 7-DOF manipulator arm.

Objectives of the Robot Vision Research Lab:

1. Map Creation: Using an indoor mobile robot platform to create detailed 2D/3D maps of interior environments for robot navigation and place recognition.

2. Position Tracking: Developing a computer vision system to accurately determine the position of moving objects within indoor environments using constructed maps.

3. FPGA-Based Architecture: Designing a novel FPGA-based system for extracting map features used in the localization of moving objects.

4. Exploratory Research: Investigating mobile robotics applications in various domains such as agriculture, military, and healthcare.

This grant and the resources of the robot vision research lab significantly reduce the project's costs, covering equipment, software development, and research expenses. As a result, the project's financial requirements are met without additional funding, emphasizing the value and impact of the grant and the support provided by Hashemite University.

This cost-effective approach not only highlights the project's sustainability but also demonstrates efficient use of available resources to advance robotics research and development. Below are some images of the research lab equipment that facilitate this project.

By jumping on the advanced capabilities and resources provided by the robot vision research lab, the PeopleBot project is able to deliver robust and meaningful robotic solutions without incurring additional costs, showcasing the effectiveness of the university's investment in cutting-edge research infrastructure and the department responsibility of providing a high- tech equipment for the students to work on.

6.2 Commercializing the Project and Relevance to JORDAN and the Region

The PeopleBot project has great commercial potential due to its flexibility as a delivery, security, and user guide robot, enhanced by advanced navigation algorithms, sensor integrations, and Artificial Intelligence algorithms.

Key elements of taking this project to the market and its essential to Jordan and the surrounding area:

The PeopleBot project has meaningful market potential in different industries. For example, in healthcare, it can decrease staff workload and improve efficiency by delivering medications, medical supplies, and documents. In educational institutions, PeopleBot can help new students navigate large campuses and transport materials between departments. In hospitality, the robot can improve customer service by guiding visitors and delivering goods in malls and hotels. Moreover, when fitted with an Axis camera, PeopleBot can perform security guard monitoring areas, and report suspicious activities or people, offering an extra level of security.
In addition, the PeopleBot project has the potential to have a major economic influence by creating employment opportunities in engineering, programming, and technical support. This will promote technological progress, establishing Jordan as a pioneer in the Middle East's technology industry. Moreover, effective commercialization has the potential to draw investments and collaborations from global tech companies supporting Jordan’s international economic position

6.3 Relevant Code of Ethics and Moral Framework

The development and usage of PeopleBot must follow an excellent ethical framework to ensure responsible and beneficial use. This could be achieved by applying key elements that include safety and reliability, so the robot operates safely around humans without causing harm, especially with the Axis camera necessitating strong data protection measures. It is essential to be clear about how data is collected, stored, and processed.

Accountability guarantees that developers and operators take responsibility for the robot's actions. integrating ensures that the robot is available and beneficial to everyone, including people with disabilities.

6.4 Environmental Analysis and Discussion

PeopleBot should be equipped with energy-efficient batteries that provide long-lasting power while minimizing environmental harm. Using rechargeable batteries reduces waste, and incorporating renewable energy sources for recharging, such as solar panels, can further minimize its carbon footprint. Ensuring the robot's charging system is efficient and uses minimal energy helps in reducing overall energy consumption, making PeopleBot more environmentally friendly.

This would carefully consider the battery and power systems to ensure sustainability and a green footprint.

7 CHAPTER 7:PROJECT MANAGEMENT

7.1 Task and Schedule

Phase 1: Conceptualization and Research

• Ideation and conceptualizing the basic system design.

• Conducting literature review and studying similar projects.

• Familiarization with the basic components of the Peoplebot robot.

• Know and understand the routing algorithms such as RL

(Reinforcement Learning) and A* (A-star) to find the optimal path, do path planning, localization and navigation.

•Research about obstacle detection and avoidance, face recognition, people detection, Bar code scanning , NFC, Databases, Laser and Sonar..

Phase 2: Understanding and Preparing Tools

• In-depth study and analysis of the Peoplebot robot, AXIS camera, and

their documentation.

• Installation and setup of MobileSim and MobileEyes, the simulator of the Peoplebot robot to do tests on it.

• Understand the libraries used with the Peoplebot robot to control it such as AriaPy, ArNetworking, Arnl and BaseArnl.

• Know the system commands used to control the robot's action, camera and laser.

Phase 3: Development of Individual Components

• Development of the initial Python script to control the Peoplebot in

MobileSim.

• Writing and testing the code for camera connectivity, obstacle detection, routing algorithms,face recognition and people detection for security applications.

• Testing the robot using the engineering faculty map in both a MobileSim simulator and Peoplebot robot.

Phase 4: Integration

• Initial testing of the integrated system in a simulated environment.

• Integration of the robot control, obstacle detection using the robot's laser, combine it with routing algorithms and its camera to build a full robot system.

Phase 5: Debugging and Refinement

• Debugging of the integrated system and code optimization.

• Conducting extensive testing to ensure system reliability and efficiency.

•Do tests in both MobileSim Simulator and peoplebot robot in the lab.

•Make sure the integrated system is running well, free from errors and its working as expected.

Phase 6: Final Testing and Implementation

• Final testing of the system in real-world scenarios.

• Implementation of the system and final adjustments based on testing

feedback.

•Most of our code must run in python 2.7 version, because it's compatible with the libraries used to control the robot's.

Phase 7: Documentation and Reporting

• Finalizing project documentation, including technical and user manuals.

• Preparation and submission of the final project report and presentation.

Phase 8: Review and Closure

• Project review, lessons learned, and project closure.

7.2 Resources and Cost Management

Hardware Resources

PeopleBot Robot, AXIS Camera: Provided by Hashemite University.

No direct cost incurred, with efficient use of lab time effectively to maximize use of the provided hardware.

Software Resources

Development Tools: All necessary software tools, including operating systems, programming environments, machine learning libraries, and simulation tools are mostly free or open-source.

Human Resources

Team Members: Utilization of the skill sets of team members:

➢ Dana Jihad Shaqdih

➢ Islam Azmi Alshourman

➢ Maryam Mohammed Shehadeh

➢ Tasneem Eyad Barakat

Project Advisor:

Guidance and expertise from:

➢ Dr. Bassam Jamil

➢ Dr. Khaleel Mustafa

Miscellaneous Costs

Funding Source: Supported by Hashemite University's resources, along with any additional funding or grants that can be procured.

Resource Utilization Plan Software Development:

Utilize open-source software and tools to minimize costs.

Task Delegation: Assign specific roles and responsibilities to team members based on expertise to optimize productivity.

Regular Review Meetings: Conduct weekly or bi-weekly meetings to track

progress, resource usage, and address any issues promptly.

7.3 Quality and Risk Management

Quality Planning

Simple Documentation Standards:

Maintain straightforward and clear documentation for coding, testing, and design changes. This ensures everyone is on the same page and can track progress efficiently.

Quality Assurance

Regular Team Meetings (online and on campus):

Hold brief, regular meetings to discuss quality related issues, progress, and any concerns that may arise.

Training Sessions: Conduct short, focused training sessions to ensure all team members understand the use of tools like MobileSim, and the camera.

Quality Control

Basic Testing Procedures: Implement basic yet effective testing procedures like unit tests for code and functional tests for hardware integration.

Feedback Collection:

Collect feedback from team members testing the system, and use this feedback for immediate improvements.

7.4 Lessons Learned

Importance of Comprehensive Documentation:

The significance of maintaining detailed documentation for each phase of the project, including design specifications, legal aspects, and design standards.

Value of Interdisciplinary Collaboration:

This project illustrated the necessity of interdisciplinary collaboration, combining knowledge in robotics, software engineering, and machine learning.

Challenges in Integrating Multiple Technologies:

The integration of the PeopleBot robot, and AXIS camera presented challenges, underscoring the complexities of combining multiple technologies, different algorithms and different parts of the system.

Regular Testing and Debugging:

Continuous testing and debugging were essential, particularly in software development and machine learning model training.

Preparing for Future Scalability and Upgrades:

Considering future scalability and potential upgrades during the design process proved beneficial for applicability.

8 CHAPTER 8: CONCLUSION AND WAY FORWARD

8.1 Restatement of Purpose of Report and Objectives

Our purpose of this report is to comprehensively outline the development and deployment of an advanced robot system designed for our faculty of engineering. This system integrates cutting-edge technologies such as laser and sonar sensors for obstacle detection, A* and reinforcement learning algorithms for optimal routing, and camera-based systems for people detection and face recognition. The primary objective is to enhance campus safety and security, facilitate efficient navigation, and provide valuable educational and practical experiences for students and faculty.

The specific objectives of this project are as follows:

- Enhance Campus Safety and Security: Implement a robust people detection and face recognition system to verify identities and ensure that only authorized individuals can access certain areas. This aims to improve overall campus security.

- Optimize Navigation and Efficiency: Utilize laser and sonar sensors in conjunction with A* and reinforcement learning algorithms to enable the robot to navigate autonomously and efficiently across the campus, avoiding obstacles and reaching designated destinations with minimal energy consumption.

- Provide Educational Value: Offer students hands-on experience with state-of-the-art robotics technologies, fostering practical knowledge and skills in fields such as artificial intelligence, computer vision, and autonomous systems. This includes integrating the robot system into the curriculum for teaching and research purposes.

- Maintain System Sustainability: Ensure the long-term operational sustainability of the robot through continuous maintenance, regular data updates for accurate mapping, and the adoption of energy-efficient components and practices.

By achieving these objectives, our project aims to not only improve the operational dynamics of the campus but also to serve as a model for integrating advanced robotics into educational environments, promoting innovation and technological advancement within the faculty of engineering.

8.2 Restatement of Proposed Deliverables

In this section, we outline the key deliverables of our project, providing a clear roadmap of the project's outcomes and ensuring alignment with the initial objectives. These deliverables are designed to demonstrate the functionality, utility, and potential of the Peoplebot in real-world applications, particularly within the educational and research environment of an engineering faculty.

1. Autonomous Navigation System

Objective: Develop an autonomous navigation system using a combination of A* routing and reinforcement learning (RL) algorithms.

Deliverable: A fully operational Peoplebot capable of navigating from a given starting point to a specific destination within the lab environment, avoiding both static and dynamic obstacles.

Description: The robot utilizes A* and RL algorithms to calculate optimal paths and adapt to changes in its environment, ensuring efficient and safe navigation.

2. Obstacle Detection and Avoidance

Objective: Implement robust obstacle detection and avoidance mechanisms.

Deliverable: Integrated laser and sonar sensor systems for real-time obstacle detection and avoidance.

Description: The robot employs laser and sonar sensors to continuously scan its surroundings, updating its map to reflect both static and dynamic obstacles and adjusting its path to prevent collisions.

3. Guidance and Delivery System

Objective: Develop a system to guide students and deliver items within the engineering faculty.

Deliverable: A system that allows the Peoplebot to escort students to various rooms and deliver items between users or rooms.

Description: The robot will be equipped with a database of room locations and user details, enabling it to navigate the faculty and perform specific tasks as requested via the GUI.

4. People Recognition System

Objective: Develop a system to accurately detect and count people in various campus environments.

Deliverable: A people recognition system that operates effectively under diverse conditions.

Description: Utilizing advanced algorithms, the system will detect and count people, helping manage events and gatherings by providing accurate attendance tracking and crowd management.

5. Security Verification System

Objective: Implement a reliable system for identity verification through face recognition and ID matching.

Deliverable: A system that ensures only authorized individuals are verified, using cameras to detect faces and match them with ID photos.

Description: The system integrates barcode scanning for ID cards and face recognition technology, enhancing campus security by verifying identities accurately and efficiently.

6. Advanced monitoring Capabilities

Objective: Explore the potential of the robot as an exam observer.

Deliverable: Initial execution of monitoring capabilities using the robot's camera in collaboration with its laser and sonar sensors.

Description: The robot will be able to monitor exam rooms, ensuring academic integrity by observing and recording activities without disrupting the exam environment.

7. Testing and Validation

Objective: Verify and test every system that has been designed in-depth.

Deliverable: A complete set of test results demonstrating the performance and reliability of the robot's systems.

Description: The testing phase will involve evaluations of the navigation system, obstacle detection and avoidance mechanisms, people recognition,

security system, and overall system integration. The results will ensure that the Peoplebot operates efficiently and safely within its intended environment, meeting the project's performance standards.

These deliverables are designed to ensure that the Peoplebot not only meets the immediate project goals but also lays the foundation for future enhancements and applications, extending its utility and impact within the engineering faculty and beyond.

8.3 Summary of How Each Objective has been Met

Obstacle Detection

The obstacle detection objective has been successfully achieved through the integration of laser and sonar

sensors. These sensors provide accurate and complementary data, allowing the robot to detect and classify both static and dynamic obstacles.

1. **Laser and Sonar Integration**: The laser sensors offer accurate distance measurements, while sonar sensors provide reliable data in challenging conditions.

2. **Algorithm Implementation**: Advanced algorithms process the sensor data to differentiate between static and dynamic obstacles, enabling the robot to navigate safely by updating its environment map and predicting the paths of moving objects.

3. **Testing**: The system was rigorously tested in various lab scenarios, confirming its ability to detect obstacles and ensure smooth, safe navigation.

A Star Algorithm

The implementation of the A* algorithm for robot navigation has been a great success, optimal pathfinding ensures that the robot can reach the goal node while considering various cost functions and heuristic values.

1. **A Algorithm Integration***: The A* algorithm effectively integrates the cost to reach a node (g-cost) and the estimated cost to the goal

(h-cost) to determine the most optimal path. This balance ensures the robot navigates efficiently towards the goal node.

2. **Cost Function and Heuristic Value:** The g-cost represents the path cost from the start node to the current node, while the h-cost estimates the cost from the current node to the goal. The sum of these costs (f-cost) is minimized to find the shortest path, ensuring optimal navigation.

3. **Optimal Path**: The A* algorithm guarantees the discovery of the optimal path by exploring nodes with the lowest f-cost. This method ensures that the robot reaches the goal node with the least possible travel distance and time.

Moreover, integrating reinforcement learning (RL) with the A* algorithm by generating the heuristic values from the Q table ensures the robot's efficient autonomous navigation.

Reinforcement Learning

Using the Reinforcement learning (RL) in the robot has achieved its objective through various key implementations, enhancing the robot's functionality, efficiency, and adaptability.

1. Enhanced Navigation and Obstacle Avoidance: Learning Optimal Paths: The RL algorithm enables the robot to improve its navigation strategies over time by interacting with the environment and receiving feedback on its actions.

Dynamic Adaptation: RL allows the robot to adapt to changes in the environment, ensuring effective navigation around unforeseen obstructions.

2. Improved Security Monitoring:

RL helps the robot learn and optimize its routes based on observed patterns of human activity, enhancing its effectiveness in monitoring and identifying individuals.

RL enables the robot to determine the best times and locations for people detection and face recognition, leading to more efficient surveillance.

3.Sustainability and Maintenance:

Resource Efficiency: it optimizes the robot's actions to minimize energy consumption and maximize operational efficiency, contributing to sustainability goals.

Facial Recognition and Security

The facial recognition and security objectives of the Peoplebot project have been successfully achieved through the implementation of advanced machine learning algorithms and the integration of high-resolution cameras. The facial recognition system was designed to identify and authenticate individuals in real-time, utilizing a robust dataset for training the recognition models. The accuracy and efficiency of the system were validated through extensive testing, ensuring reliable performance in various lighting conditions and environments.

For the security aspect, the Peoplebot was equipped with additional sensors and software to monitor and respond to potential security threats. The robot's

capability to recognize unauthorized individuals and alert security personnel was thoroughly tested. The integration of facial recognition with the robot's navigation system allowed it to monitor predefined areas, making sure there is sufficient monitoring.

Overall, the facial recognition and security objectives have been met by combining cutting-edge technologies and exacting testing protocols, resulting in a reliable and efficient system capable of enhancing safety and security within the operational environment.

People Recognition

We implemented YOLO (You Only Look Once) object detection for real-time people recognition. Using a camera for video input, the YOLO.v3 model processes the feed to detect people. The system draws bounding boxes around detected individuals and displays the count of detected people on the screen, providing real-time visualization and accurate performance under varying conditions.

8.4 New Skills Learnt

Throughout the development and implementation of the PeopleBot project, several new skills were acquired, contributing to both the technical and practical aspects of the project. These skills span across various domains, from advanced algorithm design to practical hardware integration, and are vital for the successful execution and future enhancement of the project.

1. **Routing Algorithm Design**:
    - *A * Pathfinding Algorithm*: Gained a deep understanding of the A* algorithm, learning to implement it for efficient and optimal route calculation in dynamic environments.
    - **Reinforcement Learning**: Developed skills in reinforcement learning to create adaptive navigation strategies that allow the robot to learn from its environment and improve its path planning over time.
2. **Obstacle Detection and Avoidance**:
    - **Obstacle Detection Algorithms**: Acquired knowledge in developing and implementing obstacle detection algorithms that utilize data from laser and sonar sensors to identify and avoid obstacles in real time.
    - **Real-Time Path Planning**: Learned to design and implement real-time path planning algorithms that allow the robot to dynamically adjust its route based on sensor inputs and environmental changes.

- **Sensor Calibration and Data Processing**: Developed skills in calibrating laser and sonar sensors and processing the raw data to obtain accurate measurements for obstacle detection and navigation.

- **Multi-Sensor Integration**: Learned to integrate multiple sensors to create a comprehensive perception system, enhancing the robot's ability to detect and respond to obstacles accurately.

3. **Advanced Algorithm Design**:

    - **YOLO Object Detection**: Learned the implementation and fine-tuning of (You Only Look Once) models for real-time people recognition, ensuring high accuracy and performance under varying conditions.

4. **Computer Vision and Image Processing**:

    - **OpenCV**: Developed proficiency in using OpenCV for image processing tasks such as video capture, frame pre-processing, and object detection.

    - **Face Recognition**: Learned to implement and optimize face recognition algorithms using libraries like Dlib and face_recognition, crucial for the security verification system.

5. **Sensor Integration and Data Fusion**:

    - **Laser and Sonar Sensors**: Acquired skills in integrating laser and sonar sensors for real-time obstacle detection and avoidance, essential for safe navigation.

    - **Sensor Data Fusion**: Learned techniques to fuse data from multiple sensors, enhancing the robot's ability to perceive and react to its environment accurately.

6. **Database Management**:

    - **SQLite**: Gained expertise in designing and managing a SQLite database to store and retrieve member information efficiently, supporting the security verification process.

    - **Barcode Generation and Scanning**: Developed skills in generating and scanning barcodes for ID cards using Python libraries such as python-barcode and Pyzbar.

7. **Software Development and Integration**:

    - **Python Programming**: Enhanced proficiency in Python, focusing on developing and integrating various modules for people recognition, security verification, navigation, and user interaction.

    - **System Integration**: Learned to integrate multiple software components seamlessly, ensuring smooth communication and operation of the entire PeopleBot system.

8. **Hardware Implementation**:

    - **PTZ Cameras**: Developed skills in setting up and using PTZ (Pan-Tilt-Zoom) cameras for capturing high-resolution video feeds, crucial for both navigation and surveillance tasks.

    - **Robot Hardware**: Gained practical experience in working with robot hardware, including motors, sensors, and processing units, ensuring reliable and efficient operation.

9. **Testing and Validation**:

    - **Performance Testing**: Learned to design and execute comprehensive performance tests to evaluate system accuracy, latency, and scalability under various conditions.

    - **Error Handling and Debugging**: Gained experience in identifying, troubleshooting, and resolving errors, ensuring robust and reliable system performance.

10. **User Interface Development**:

    - **GUI Design**: Developed skills in designing and implementing user-friendly graphical user interfaces (GUIs) for controlling the robot and monitoring its status, enhancing user interaction and experience.

These new skills not only contributed to the successful implementation of the PeopleBot project but also provided valuable expertise that can be applied to

future projects and endeavors in robotics, computer vision, artificial intelligence, and autonomous systems.

8.5 Way Forward

The current implementation of the Peoplebot project has demonstrated significant success in navigating within the confines of a laboratory environment. The robot efficiently uses advanced routing algorithms, obstacle detection mechanisms, and a user-friendly interface to guide movements, avoid obstacles, and respond to user commands. However, the potential applications of this project extend far beyond the lab.

8.5.1 Expanding to the Engineering Faculty

To broaden the scope of the Peoplebot's functionalities, we aim to deploy the robot throughout the entire engineering faculty. This expansion will enable the robot to:

Guide Students: Assist students in navigating the faculty by directing them to specific rooms, labs, or offices.

Deliver Items: Facilitate the delivery of items between users or between different rooms, improving efficiency and performance.

Required Enhancements

1. **User Interface (GUI) Upgrades:**

- **Room Number Input:** Modify the GUI to include an input field where users can enter the destination room number. This will allow for more precise and user-specific guidance.

- **Enhanced Navigation Options:** Add functionalities for selecting various tasks such as guiding to a room, delivering items, or general navigation assistance.

2. **Update the maze matrix for A *Algorithm:**

- **Extended Environment Mapping:** Update the maze matrix used in the A* algorithm to reflect the entire layout of the engineering faculty, not just the lab. This will involve:

- **Mapping All Rooms and Corridors:** Creating a comprehensive map that includes all rooms, corridors, and key locations within the faculty.

- **Defining Obstacles and Pathways:** Clearly defining static obstacles and permissible pathways to ensure accurate navigation.

3. Integration of New Features:

For enhanced user involvement, we could include extra interaction features like voice commands or touchscreen inputs.

Implementation Plan

1. **Mapping and Data Collection:**

- Conduct a thorough survey of the engineering faculty to create an accurate and detailed map.

- Gather data on room locations, corridors, and potential obstacles.

2. **Software Development:**

- Update the existing GUI to include new features and inputs.
- Modify the A* algorithm to incorporate the expanded maze matrix.
- Integrate real-time dynamic adjustment capabilities into the navigation system.

3. **Testing and Validation:**

- Conduct extensive testing within the new environment to validate the accuracy and reliability of the navigation system.
- Collect user feedback to refine and improve the system's performance.

4. **Deployment and Training:**

- Gradually deploy the updated Peoplebot system across the engineering faculty.
- Provide training sessions for students and faculty members to familiarize them with the robot's functionalities and usage.

An interesting potential to use advanced robotics for useful, real-world applications is presented by extending the Peoplebot's operational scope from the lab to the full engineering faculty. By enhancing the GUI, updating the maze matrix, and integrating new features, we can ensure that the Peoplebot becomes an essential tool for navigation, item delivery, and student assistance. This forward-thinking approach will not only improve

efficiency and convenience within the faculty but also pave the way for future advancements in autonomous robotics.

8.5.2 Using the Robot as an Exam Observer

In future work, we imagine utilizing the Peoplebot as an autonomous exam observer, utilizing the collaboration of its camera, laser, and sonar sensors. By integrating the camera's real-time video feed with the laser and sonar's precise obstacle detection and localization capabilities, the robot can monitor exam rooms effectively. The camera will provide a comprehensive view of the exam environment, ensuring academic integrity by identifying unusual behavior. Meanwhile, the laser and sonar sensors will enable the robot to navigate the exam room without causing disruptions, avoiding static and dynamic obstacles such as desks, chairs, and moving students. This collaboration of technologies ensures a smooth and effective monitoring system, enhancing the exam monitoring process while maintaining a non-intrusive presence.

8.5.3 Camera, laser, and sonar collaboration

Autonomous robots like the Peoplebot can detect obstacles much better in the future if sonar, camera, and laser sensors are integrated. Whereas sonar sensors enable dependable detection in a range of environmental situations, laser sensors give accurate distance measurements and high-resolution mapping functionality. Even with their drawbacks, PTZ cameras provide important visual information for complete scene analysis and contextual understanding. We can build a reliable multi-sensor system that makes use of the advantages of each kind of sensor by combining these complementary technologies. More complex and reliable autonomous navigation in complex

and dynamic situations could be possible because of this collaboration, which aims to increase obstacle detection accuracy, real-time performance, and environmental adaptability.

8.6 Final Discussion and remarks

In this project, the integration of A* algorithm with Reinforcement Learning (RL) has enhanced PeopleBot's navigation capabilities. This merged approach integrated A*'s efficient pathfinding and RL's strong learning to create a robust navigation system that succeeds in dynamic environments. The A* algorithm provides a structured and efficient pathfinding framework, while RL continuously refines the heuristic function based on real-time feedback. All of this successfully enables the PeopleBot to navigate more effectively by prioritizing historical paths and adjusting to new obstacles or changes in the environment.

In addition, the integration of laser and sonar sensors through the ARIA library enhances PeopleBot's obstacle detection and avoidance. The laser rangefinders offer accurate measurements for long-range detection, supported by sonar sensors for short-range data, securing complete environmental recognition and effective collision avoidance.

In terms of people recognition and security verification, the implementation of advanced algorithms for detection and face recognition for security purposes has shown high effectiveness. The system's ability to detect and count individuals in various environments, integrated with the face recognition capabilities, ensures reliable identification and security. Further optimization by the use of custom ID cards with barcodes and face images enables the system ability to handle more users and verifications.

In conclusion, the integration of advanced pathfinding algorithms, self and machine-learning techniques, and practical sensor and laser data processing has resulted in an adaptive and reliable navigation system for the PeopleBot. The project's success in merging A* with RL, merging laser and sonar

sensors, and implementing robust people recognition and security verification systems using professional algorithms shows the potential of integrating multiple technologies to address complex navigation challenges. After all, this project represents an extremely great advancement in autonomous navigation technology, opening the door for future innovations in robotic systems.

REFERENCES

1. Ahmad Yousef, Khalil M., Bassam J. Mohd, Khalid Al-Widyan, and Thaier Hayajneh. "Extrinsic Calibration of Camera and 2D Laser Sensors without Overlap." *Sensors* 17, no. 10 (2017): 2346.

2. ActivMedia Robotics, LLC. *PeopleBot™ System Manual, Version 8.2*. ActivMedia Robotics, LLC, 2004.

3. AXIS Communications. *AXIS 214 PTZ Network Camera User's Manual*. AXIS Communications AB, 2007.

4. Brownlee, Jason. "A Gentle Introduction to Object Recognition with Deep Learning." *Machine Learning Mastery* 5 (2019): 10.

5. Candra, Ade, Mohammad Andri Budiman, and Kevin Hartanto. "Dijkstra's and A-star in Finding the Shortest Path: A Tutorial." In *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, 28-32. IEEE, 2020.

6. Duchoň, František, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. "Path Planning with Modified A Star Algorithm for a Mobile Robot." *Procedia Engineering* 96 (2014): 59-69.

7. Fu, Bing, Lin Chen, Yuntao Zhou, Dong Zheng, Zhiqi Wei, Jun Dai, and Haihong Pan. "An Improved A* Algorithm for the Industrial Robot Path Planning with High Success Rate and Short Length." *Robotics and Autonomous Systems* 106 (2018): 26-37.

8. Kabir, Raihan, Yutaka Watanobe, Md Rashedul Islam, and Keitaro Naruse. "Enhanced Robot Motion Block of A-Star Algorithm for Robotic Path Planning." Sensors 24, no. 5 (2024): 1422.

9. Madhavan, T. R., and M. Adharsh. "Obstacle Detection and Obstacle Avoidance Algorithm Based on 2-D RPLiDAR." In *2019 International Conference on Computer Communication and Informatics (ICCCI)*, 1-4. IEEE, 2019.

10. Singh, Kiran Jot, Divneet Singh Kapoor, Mohamed Abouhawwash, Jehad F. Al-Amri, Shubham Mahajan, and Amit Kant Pandit. "Behavior of Delivery Robot in Human-Robot Collaborative Spaces During Navigation." *Intelligent Automation & Soft Computing* 35, no. 1 (2023).

11. Sullins, John P. "Introduction: Open Questions in Roboethics." *Philosophy & Technology* 24, no. 3 (2011): 233-238.

12. Van Wynsberghe, Aimee, and Justin Donhauser. "The Dawning of the Ethics of Environmental Robots." *Science and Engineering Ethics* 24 (2018): 1777-1800.

13. Wang, Huanwei, Shangjie Lou, Jing Jing, Yisen Wang, Wei Liu, and Tieming Liu. "The EBS-A* algorithm: An improved A* algorithm for path planning." PloS one 17, no. 2 (2022): e0263841.

Web Sources:

1. Basilchackomathew. "Face Recognition in Python: A Comprehensive Guide." *Medium*. Accessed May 20, 2024. https://basilchackomathew.medium.com/face-recognition-in-python-a-comprehensive-guide-960a48436d0f.

2. Machine Learning Mastery. "Object Recognition with Deep Learning." Accessed May 20, 2024. https://machinelearningmastery.com/object-recognition-with-deep-learning/.

3. Manalelaidouni. "Single Shot Object Detection." Accessed May 20, 2024. https://manalelaidouni.github.io/Single%20shot%20object%20detection.html.

4. Webscale. "Introduction to YOLO Algorithm for Object Detection." Accessed May 20, 2024. https://www.webscale.com/engineering-education/introduction-to-yolo-algorithm-for-object-detection/.

APPENDICES

Appendix A: Robot

```
from AriaPy import *
import sys
from Rein_Learn import SimpleGridWorld, train_rl_agent,
load_q_values
import A_Star_Search as A_Star_Search

# Define your maze size and create a GridWorld object
width = 14
height = 10
grid_world = SimpleGridWorld(width, height)

# Train the RL agent to learn Q-values
q_values = train_rl_agent(grid_world, num_episodes=1000,
initial_epsilon=0.1, alpha=0.1, gamma=0.9)

# Use the learned Q-values as the heuristic function in
the A* algorithm
def h_q_values(cell1, cell2):
    row1, col1 = cell1
    row2, col2 = cell2
    key1 = "({}, {})".format(row1, col1)
    key2 = "({}, {})".format(row2, col2)
    return max(q_values[row1][col1]) +
max(q_values[row2][col2])

# Define the start and end nodes for A*
start_node = (0, 0)
end_node = (9, 13)

# Find the optimal path using A* algorithm with Q-values
as the heuristic
path = A_Star_Search.a_star(A_Star_Search.my_maze,
start_node, end_node, heuristic=h_q_values)
```

```
# Display the optimal path
print("Optimal Path:")
for cell in path:
    print(cell)


class PrintingTask:
    def __init__(self, robot, goal_pose):
        self.myRobot = robot
        self.myGoal = goal_pose
        robot.addSensorInterpTask("PrintingTask", 50,
self.doTask)

    def doTask(self):
        print("x %6.1f  y %6.1f  th  %6.1f vel %7.1f
mpacs %3d" % (self.myRobot.getX(), self.myRobot.getY(),
self.myRobot.getTh(), self.myRobot.getVel(),
self.myRobot.getMotorPacCount()))
        if self.myRobot.getX() >= self.myGoal.getX() and
self.myRobot.getY() >= self.myGoal.getY():
            print("Goal Reached ^_^")
            self.myRobot.stop()

Aria_init()

parser = ArArgumentParser(sys.argv)
parser.loadDefaultArguments()

# Create a robot object:
robot = ArRobot()

# Create a "simple connector" object and connect to
either the simulator or the robot. Unlike the C++ API
which takes int and char* pointers, the Python
constructor just takes argv as a list.
print("Hey There...")

con = ArRobotConnector(parser, robot)
if not Aria_parseArgs():
    Aria_logOptions()
```

```
    Aria_exit(1)

if not con.connectRobot():
    print("Could not connect to robot, exiting")
    Aria_exit(1)

# Define actions for obstacle avoidance
stallRecover = ArActionStallRecover()
avoidFront = ArActionAvoidFront()
limitFront = ArActionLimiterForwards("limitFront", 300,
600, 250)
limitBack = ArActionLimiterBackwards()

# Add actions to ArRobot
robot.addAction(stallRecover, 100)
robot.addAction(avoidFront, 50)
robot.addAction(limitFront, 30)
robot.addAction(limitBack, 20)

# Run the robot threads in the background:
print("Running...")
sonar = ArSonarDevice()
robot.addRangeDevice(sonar)

# Some robots have laser rangefinders (enabled in
robot's parameter .p file or with -connectLaser command
line argument):
laserConn = ArLaserConnector(parser, robot, con)
if not laserConn.connectLasers():
    print("Warning: could not connect to laser(s).")

print("Connected to the robot. (Press Ctrl-C to exit)")

# Get goal coordinates from the start and end nodes
goal_x_start, goal_y_start = start_node
goal_x_end, goal_y_end = end_node

# Create ArPose objects for start and end nodes
goal_pose_start = ArPose(goal_x_start, goal_y_start)
goal_pose_end = ArPose(goal_x_end, goal_y_end)
```

```
# Set up action for going to the goal
goto_action = ArActionGoto("goto", goal_pose_end, 0)
robot.addAction(goto_action, 0)

# Initialize PrintingTask
printTask = PrintingTask(robot, goal_pose_end)

# Enable motors and run the robot
robot.enableMotors()
robot.run(1)
print("Disconnected. Goodbye.")

Aria_exit(0)
```

A Star Search:

```
import heapq

#Lab 2048 maze
my_maze = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 1, 1, 1, 0, 0, 1],
    [1, 0, 0, 0, 1, 1, 1, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 1, 1]
]

def heuristic(cell1, cell2):
    (x1, y1) = cell1
    (x2, y2) = cell2
    return abs(x1 - x2) + abs(y1 - y2)
```

```
def a_star(maze, start, end, heuristic):
    rows = len(maze)
    cols = len(maze[0])
    open_list = []
    heapq.heappush(open_list, (0, start))
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, end)}
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == end:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        for direction in directions:
            neighbor = (current[0] + direction[0],
current[1] + direction[1])

            if 0 <= neighbor[0] < rows and 0 <=
neighbor[1] < cols and maze[neighbor[0]][neighbor[1]] ==
0:
                tentative_g_score = g_score[current] + 1

                if neighbor not in g_score or
tentative_g_score < g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] =
tentative_g_score
                    f_score[neighbor] =
tentative_g_score + heuristic(neighbor, end)
                    heapq.heappush(open_list,
(f_score[neighbor], neighbor))
```

```
    return None


Rein_Learn:
import random
import json

class SimpleGridWorld:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.agent_position = (0, 0)  # Agent starts at
top-left corner
        self.goal_position = (width - 1, height - 1)  #
Goal position at bottom-right corner
        self.rewards = [[0] * width for _ in
range(height)]

self.rewards[self.goal_position[1]][self.goal_position[0
]] = 100  # Reward at goal state
        self.q_values = [[[0, 0, 0, 0] for _ in
range(width)] for _ in range(height)]  # Initialize
Q-values

    def reset(self):
        self.agent_position = (0, 0)  # Reset agent
position to top-left corner
        return self.agent_position

    def step(self, action):
        row, col = self.agent_position

        # Define action effects
        if action == 0:  # move up
            row -= 1
        elif action == 1:  # move down
            row += 1
        elif action == 2:  # move left
            col -= 1
```

```
        elif action == 3:  # move right
            col += 1

        # Ensure new position is within the grid
        row = max(0, min(row, self.height - 1))
        col = max(0, min(col, self.width - 1))

        # Update agent position
        self.agent_position = (row, col)

        # Check if the agent reached the goal
        if self.agent_position == self.goal_position:
            done = True
            reward = self.rewards[row][col]
        else:
            done = False
            reward = self.rewards[row][col]

        return self.agent_position, reward, done, {}

    def render(self):
        grid = [['O' for _ in range(self.width)] for _
in range(self.height)]

grid[self.agent_position[1]][self.agent_position[0]] =
'A'  # Agent

grid[self.goal_position[1]][self.goal_position[0]] = 'G'
# Goal

        for row in grid:
            print(' '.join(row))

    def choose_action(self, state, epsilon):
        row, col = state
        if random.uniform(0, 1) < epsilon:
            return
random.choice(self.available_actions(state))
        else:
```

```
            return max(range(4), key=lambda x:
self.q_values[row][col][x])

    def available_actions(self, state):
        row, col = state
        actions = []
        if row > 0:
            actions.append(0)  # Up
        if row < self.height - 1:
            actions.append(1)  # Down
        if col > 0:
            actions.append(2)  # Left
        if col < self.width - 1:
            actions.append(3)  # Right
        return actions

def train_rl_agent(env, num_episodes, initial_epsilon,
alpha, gamma, max_steps=1000, epsilon_decay=0.995,
min_epsilon=0.01):
    epsilon = initial_epsilon

    try:
        for episode in range(num_episodes):
            state = env.reset()
            done = False
            step = 0

            while not done and step < max_steps:
                row, col = state

                if random.random() < epsilon:
                    action =
random.choice(env.available_actions(state))  # Explore
                else:
                    action = env.choose_action(state,
epsilon)  # Exploit

                next_state, reward, done, _ =
env.step(action)
                next_row, next_col = next_state
```

```
                # Update Q-values using the Q-learning
update rule
                old_value =
env.q_values[row][col][action]
                next_max =
max(env.q_values[next_row][next_col])
                env.q_values[row][col][action] = (1 -
alpha) * old_value + alpha * (reward + gamma * next_max)

                state = next_state
                step += 1

            # Decay epsilon
            epsilon = max(min_epsilon, epsilon *
epsilon_decay)

        return env.q_values

    except Exception as e:
        print("An error occurred during training:", e)

def test_rl_agent(env, q_values, epsilon):
    total_rewards = 0
    num_episodes = 10

    for _ in range(num_episodes):
        state = env.reset()
        done = False

        while not done:
            action = env.choose_action(state, epsilon)
            next_state, reward, done, _ =
env.step(action)
            total_rewards += reward
            state = next_state

    return total_rewards / num_episodes

def save_q_values(q_values, filename):
```

```
    with open(filename, 'w') as f:
        json.dump(q_values, f)

def load_q_values(filename):
    with open(filename, 'r') as f:
        return json.load(f)


def main():
    try:
        env = SimpleGridWorld(width=14, height=10)

        # Define training parameters
        num_episodes = 1000
        epsilon = 0.7
        alpha = 0.1
        gamma = 0.9

        q_values = train_rl_agent(env, num_episodes,
epsilon, alpha, gamma)

        # Save Q-values to a JSON file
        save_q_values(q_values, 'q_values.json')

        # Load Q-values from file
        loaded_q_values = load_q_values('q_values.json')

        # Print the loaded Q-values to verify
        if loaded_q_values:
            print("Loaded Q-values:")
            for row, row_values in
enumerate(loaded_q_values):
                for col, col_values in
enumerate(row_values):
                    print("State: ({}, {}), Q-values:
{}".format(row, col, col_values))
        else:
            print("No Q-values loaded.")
```

```
        # Print the learned Q-values in a structured
format
        print("DEBUG ::: Learned Q-values:")
        for row in range(env.height):
            for col in range(env.width):
                state = (row, col)
                q_values_str = ",
".join("{:.2f}".format(q) for q in
env.q_values[row][col])
                print("State: {}, Q-values:
[{}]".format(state, q_values_str))


    except Exception as e:
        print("An error occurred:", e)

if __name__ == "__main__":
    main()
```

Appendix B: **wandering and navigation**

```
from AriaPy import *
import sys


class PrintingTask:


  # Constructor. Adds a sensor interpretation task to
the given robot object.
  def __init__(self, robot):
    self.myRobot = robot
    robot.addSensorInterpTask("PrintingTask", 50,
self.doTask)
```

```
  # This method will be called by ArRobot as a sensor
interpretation task
  def doTask(self):
    print "x %6.1f  y %6.1f  th  %6.1f vel %7.1f mpacs
%3d"  % (self.myRobot.getX(), self.myRobot.getY(),
self.myRobot.getTh(), self.myRobot.getVel(),
self.myRobot.getMotorPacCount())


Aria_init()

parser = ArArgumentParser(sys.argv)
robot = ArRobot()
con = ArRobotConnector(parser, robot)


if not con.connectRobot():
  print "Could not connect to robot, exiting"
  Aria_logOptions()
  Aria_exit(1)



sonar = ArSonarDevice()


# This object encapsulates the task we want to do every
cycle.
# Upon creation, it puts a callback functor in the
ArRobot object
# as a 'user task'.
pt = PrintingTask(robot)


# actions used to wander
recover = ArActionStallRecover()
aFront = ArActionAvoidFront()
constantVelocity = ArActionConstantVelocity("Constant
Velocity", 400)
```

```
robot.addRangeDevice(sonar)


print "Connected to the robot. (Press Ctrl-C to exit)"


# add the wander actions
robot.addAction(recover, 100)
robot.addAction(aFront, 50)
robot.addAction(constantVelocity, 25)


robot.enableMotors()


robot.run(1)


print "Disconnected. Goodbye."


Aria.exit(0)
```

Appendix C: **obstacles detection**

```
from AriaPy import *
import sys


# Define a custom task for printing robot position
information
class PrintingTask:
    def __init__(self, robot):
        self.myRobot = robot
        robot.addSensorInterpTask("PrintingTask", 50,
self.doTask)


    def doTask(self):
        print("x %6.1f  y %6.1f  th  %6.1f vel %7.1f
mpacs %3d" % (self.myRobot.getX(), self.myRobot.getY(),
self.myRobot.getTh(), self.myRobot.getVel(),
self.myRobot.getMotorPacCount()))




# Initialize ARIA
Aria_init()


# Create argument parser and load default arguments
parser = ArArgumentParser(sys.argv)
robot = ArRobot()
conn = ArRobotConnector(parser, robot)


if not conn.connectRobot():
    print("Could not connect to robot, exiting")
    Aria_exit(1)
```

```
# Parse arguments
if not Aria_parseArgs():
    Aria_logOptions()
    Aria_exit(1)


# Most robots have sonar:
print("Creating sonar object...")
sonar = ArSonarDevice()
robot.addRangeDevice(sonar)




# Some robots have laser rangefinders (enabled in
robot's parameter .p file or
# with -connectLaser command line argument):
laserConn = ArLaserConnector(parser, robot, conn)
if not laserConn.connectLasers():
    print ("Warning: could not connect to laser(s).")


# Add custom printing task for position information
printTask = PrintingTask(robot)


# Define actions for obstacle avoidance
stallRecover = ArActionStallRecover()
avoidFront = ArActionAvoidFront()
limitFront = ArActionLimiterForwards("limitFront", 300,
600, 250)
limitBack = ArActionLimiterBackwards()
constVel = ArActionConstantVelocity()


# Add actions to ArRobot
robot.addAction(stallRecover, 100)
robot.addAction(avoidFront, 50)
```

```
robot.addAction(limitFront, 30)  # Increase priority to
allow more aggressive avoidance
robot.addAction(limitBack, 20)
robot.addAction(constVel, 10)


# Enable robot motors
print("Enabling motors...")
robot.enableMotors()


# Run robot thread here in the main thread
print("Running robot...")
robot.run(1)


print("Goodbye.")
Aria_exit(0)
```

Appendix D: Rein Learn

```
import random
import json

class SimpleGridWorld:
    def __init__(self, width, height):
        self.width = width
        self.height = height
         self.agent_position = (0, 0)  # Agent starts at
top-left corner
         self.goal_position = (width - 1, height - 1)  #
Goal position at bottom-right corner
              self.rewards = [[0] * width for _ in
range(height)]

self.rewards[self.goal_position[1]][self.goal_position[0
]] = 100  # Reward at goal state
            self.q_values = [[[0, 0, 0, 0] for _ in
range(width)] for _ in range(height)]   # Initialize
Q-values

    def reset(self):
           self.agent_position = (0, 0)  # Reset agent
position to top-left corner
        return self.agent_position

    def step(self, action):
        row, col = self.agent_position

        # Define action effects
        if action == 0:  # move up
            row -= 1
        elif action == 1:  # move down
            row += 1
        elif action == 2:  # move left
            col -= 1
        elif action == 3:  # move right
            col += 1
```

```
        # Ensure new position is within the grid
        row = max(0, min(row, self.height - 1))
        col = max(0, min(col, self.width - 1))

        # Update agent position
        self.agent_position = (row, col)

        # Check if the agent reached the goal
        if self.agent_position == self.goal_position:
            done = True
            reward = self.rewards[row][col]
        else:
            done = False
            reward = self.rewards[row][col]

        return self.agent_position, reward, done, {}

    def render(self):
         grid = [['O' for _ in range(self.width)] for _
in range(self.height)]

grid[self.agent_position[1]][self.agent_position[0]]   =
'A'  # Agent

grid[self.goal_position[1]][self.goal_position[0]] = 'G'
# Goal
        for row in grid:
            print(' '.join(row))

    def choose_action(self, state, epsilon):
        row, col = state
        if random.uniform(0, 1) < epsilon:
                                                  return
random.choice(self.available_actions(state))
        else:
                    return  max(range(4),  key=lambda  x:
self.q_values[row][col][x])

    def available_actions(self, state):
        row, col = state
```

```
        actions = []
        if row > 0:
            actions.append(0)  # Up
        if row < self.height - 1:
            actions.append(1)  # Down
        if col > 0:
            actions.append(2)  # Left
        if col < self.width - 1:
            actions.append(3)  # Right
        return actions

def train_rl_agent(env, num_episodes, initial_epsilon, alpha, gamma, max_steps=1000, epsilon_decay=0.995, min_epsilon=0.01):
    epsilon = initial_epsilon

    try:
        for episode in range(num_episodes):
            state = env.reset()
            done = False
            step = 0

            while not done and step < max_steps:
                row, col = state

                if random.random() < epsilon:
                                                action = random.choice(env.available_actions(state))  # Explore
                else:
                                    action = env.choose_action(state, epsilon)  # Exploit

                                next_state, reward, done, _ = env.step(action)
                next_row, next_col = next_state

                  # Update Q-values using the Q-learning update rule
                                                old_value = env.q_values[row][col][action]
```

```
                                                    next_max   =
max(env.q_values[next_row][next_col])
                    env.q_values[row][col][action] = (1 -
alpha) * old_value + alpha * (reward + gamma * next_max)

                state = next_state
                step += 1

            # Decay epsilon
                  epsilon = max(min_epsilon, epsilon *
epsilon_decay)

        return env.q_values

    except Exception as e:
        print("An error occurred during training:", e)

def test_rl_agent(env, q_values, epsilon):
    total_rewards = 0
    num_episodes = 10

    for _ in range(num_episodes):
        state = env.reset()
        done = False

        while not done:
            action = env.choose_action(state, epsilon)
                        next_state,  reward,  done,  _  =
env.step(action)
            total_rewards += reward
            state = next_state

    return total_rewards / num_episodes

def save_q_values(q_values, filename):
    with open(filename, 'w') as f:
        json.dump(q_values, f)

def load_q_values(filename):
    with open(filename, 'r') as f:
```

```
        return json.load(f)


def main():
    try:
        env = SimpleGridWorld(width=14, height=10)

        # Define training parameters
        num_episodes = 1000
        epsilon = 0.7
        alpha = 0.1
        gamma = 0.9

           q_values = train_rl_agent(env, num_episodes,
epsilon, alpha, gamma)

        # Save Q-values to a JSON file
        save_q_values(q_values, 'q_values.json')

        # Load Q-values from file
        loaded_q_values = load_q_values('q_values.json')

        # Print the loaded Q-values to verify
        if loaded_q_values:
            print("Loaded Q-values:")
                                for  row,  row_values  in
enumerate(loaded_q_values):
                                  for  col,  col_values  in
enumerate(row_values):
                        print("State: ({}, {}), Q-values:
{}".format(row, col, col_values))
        else:
            print("No Q-values loaded.")

          # Print the learned Q-values in a structured
format
        print("DEBUG ::: Learned Q-values:")
        for row in range(env.height):
            for col in range(env.width):
                state = (row, col)
```

```
                                    q_values_str = ",
".join("{:.2f}".format(q) for q in
env.q_values[row][col])
                            print("State: {}, Q-values:
[{}]".format(state, q_values_str))

    except Exception as e:
        print("An error occurred:", e)

if __name__ == "__main__":

        main()
```

Appendix E: Main code PeopleRec & Security

```
import datetime
from PeopleRec import detect_people
from BarCode import detect_barcode
from DataBase import create_connection,
get_member_by_id, create_table
from FaceRec2 import perform_face_recognition


def main():
   database_path = r"D:\Spring
2024\GP2\ReadyCodes\members.db"
   conn = create_connection(database_path)
   create_table(conn)


   current_time = datetime.datetime.now()
   print("Current time:",
current_time.strftime("%H:%M:%S"))


   if current_time.hour < 11:
       print("Starting people detection...")
       detect_people()
   else:
       print("Starting ID detection...")
       barcode = detect_barcode()
       if barcode:
           print(f""
                 f"ID detected: {barcode}")
           member_info = get_member_by_id(conn, barcode)
           if member_info:
               photo_path = member_info[4]
               print(f"Photo path for member:
{photo_path}")
               perform_face_recognition(photo_path)
           else:
               print("No member found for this ID")
       else:
```

```
            print("No ID detected.")


    if conn:
        conn.close()


if __name__ == "__main__":
    main(
```

Appendix F: PeopleRec

```
import cv2
import numpy as np


def detect_people(display=True):
    # Loading YOLO Model
    net = cv2.dnn.readNet(r'D:\Spring
2024\GP2\yolo-cfg\weights.weights', r'D:\Spring
2024\GP2\yolo-cfg\config.cfg')
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers().flatten()]


    # Load classes
    classes = []
    with open(r'D:\Spring 2024\GP2\yolo-cfg\names.names',
'r') as f:
        classes = [line.strip() for line in
f.readlines()]


    # Video capture
    cap = cv2.VideoCapture(0)
    font = cv2.FONT_HERSHEY_COMPLEX
    new_width, new_height = 1280, 720
```

```
   while True:
        ret, frame = cap.read()
        if not ret:
            break


        # Resize the frame for processing
        frame = cv2.resize(frame, (new_width,
new_height))
        height, width, _ = frame.shape


        # Prepare the frame for YOLO detection
        blob = cv2.dnn.blobFromImage(frame, 0.00392,
(416, 416), (0, 0, 0), True, crop=False)
        net.setInput(blob)
        outs = net.forward(output_layers)


        class_ids, confidences, boxes = [], [], []
        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > 0.5:
                    center_x, center_y = int(detection[0]
* width), int(detection[1] * height)
                    w, h = int(detection[2] * width),
int(detection[3] * height)
                    x, y = int(center_x - w / 2),
int(center_y - h / 2)
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)


        indexes = cv2.dnn.NMSBoxes(boxes, confidences,
0.5, 0.4)
```

```
        people_count = 0


        for i in range(len(boxes)):
            if i in indexes and class_ids[i] == 0:  #
Class ID 0 is a person
                x, y, w, h = boxes[i]
                cv2.rectangle(frame, (x, y), (x + w, y +
h), (0, 255, 0), 2)
                people_count += 1


        if display:
            cv2.putText(frame, f'People detected:
{people_count}', (20, 50), font, 1, (255, 255, 255), 2)
            cv2.imshow("People Detection", frame)


        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


    cap.release()
    cv2.destroyAllWindows()


if __name__ == "__main__":
    detect_people()
```

Appendix G: **BarCode scanner**

```
import cv2
from pyzbar import pyzbar


def detect_barcode():
   video_capture = cv2.VideoCapture(0)
   barcode_data = None


   try:
       while True:
           ret, frame = video_capture.read()
           if not ret:
               continue


           # Displaying "Show your ID" text on the
screen
           cv2.putText(frame, "Show your ID", (50, 50),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 2)


           barcodes = pyzbar.decode(frame)
           for barcode in barcodes:
               (x, y, w, h) = barcode.rect
               cv2.rectangle(frame, (x, y), (x + w, y +
h), (0, 255, 0), 2)
               barcode_data =
barcode.data.decode("utf-8")
               barcode_type = barcode.type
               text = f"{barcode_data} ({barcode_type})"
               cv2.putText(frame, text, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
               print(f"ID detected: {barcode_data}
[{barcode_type}]")
```

```
 # Return the detected barcode data to the main script
                return barcode_data


            cv2.imshow('Barcode Scanner', frame)


            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
    finally:
        video_capture.release()
        cv2.destroyAllWindows()
```

Appendix H:Database & ID generator

```
import sqlite3
from sqlite3 import Error
from PIL import Image, ImageDraw, ImageFont
import barcode
from barcode.writer import ImageWriter
import os


def create_connection(db_file):
    """Create a database connection to the SQLite
database specified by db_file."""
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        print("Connection is established: Database is
created.")
    except Error as e:
        print(f"Error connecting to database: {e}")
    return conn


def create_table(conn):
    """Create a table if not exists."""
    sql_create_members_table = """
    CREATE TABLE IF NOT EXISTS members (
```

```
        id integer PRIMARY KEY,
        first_name text NOT NULL,
        last_name text NOT NULL,
        id_number text NOT NULL UNIQUE,
        photo_path text NOT NULL
    );"""
    try:
        cursor = conn.cursor()
        cursor.execute(sql_create_members_table)
    except Error as e:
        print(f"Error creating table: {e}")


def add_member(conn, member, department, major,
logo_path, output_dir):
    """Add a new member to the members table and create
an ID card for them."""
    sql = ''' INSERT INTO members(first_name, last_name,
id_number, photo_path)
              VALUES(?,?,?,?) '''
    try:
        cur = conn.cursor()
        cur.execute(sql, member)
        conn.commit()
        print(f"New member added with id:
{cur.lastrowid}")


        # Generate ID card
        first_name, last_name, id_number, photo_path =
member
        name = f"{first_name} {last_name}"
        output_path = os.path.join(output_dir,
f"{id_number}.png")
        generate_id_card(name, id_number, department,
major, photo_path, logo_path, output_path)
    except Error as e:
        print(f"Error adding new member: {e}")
```

```
def get_member_by_id(conn, id_number):
    """Query member by ID number."""
    sql = 'SELECT * FROM members WHERE id_number=?'
    try:
        cur = conn.cursor()
        cur.execute(sql, (id_number,))
        member = cur.fetchone()
        return member
    except Error as e:
        print(f"Error querying member by ID: {e}")
        return None


def generate_id_card(name, id_number, department, major,
photo_path, logo_path, output_path):
    try:
        # Create a blank white image
        width, height = 800, 450
        card = Image.new('RGB', (width, height), 'white')
        draw = ImageDraw.Draw(card)


        # Define fonts
        font_path = "C:\\Windows\\Fonts\\times.ttf"  #
Update path if necessary
        font_large = ImageFont.truetype(font_path, 36)
        font_medium = ImageFont.truetype(font_path, 28)
        font_small = ImageFont.truetype(font_path, 24)


        # Load and paste the new university logo with
transparency handling
        if not os.path.exists(logo_path):
            raise FileNotFoundError(f"Logo file not found
at {logo_path}")
        logo =
Image.open(logo_path).convert("RGBA").resize((300, 120),
Image.LANCZOS)  # Increased size
        logo_bg = Image.new('RGBA', logo.size, (255, 255,
255, 0))
```

```
        logo_combined = Image.alpha_composite(logo_bg,
logo)
        card.paste(logo_combined, (30, 15),
logo_combined)


        # Draw a thick line below the logo
        line_y = 135  # Y-coordinate for the line
        draw.rectangle([0, line_y, 800, line_y + 5],
fill="black")


        # Load and paste the photo
        if not os.path.exists(photo_path):
            raise FileNotFoundError(f"Photo file not
found at {photo_path}")
        photo = Image.open(photo_path).resize((150, 180),
Image.LANCZOS)  # Increased size
        photo_position = (600, 30)
        card.paste(photo, photo_position)


        # Draw the border around the photo
        border_thickness = 5
        photo_border_position = [photo_position[0] -
border_thickness,
                                 photo_position[1] -
border_thickness,
                                 photo_position[0] +
photo.size[0] + border_thickness,
                                 photo_position[1] +
photo.size[1] + border_thickness]
        draw.rectangle(photo_border_position,
outline="black", width=border_thickness)


        # Draw the details text
        draw.text((30, 180), f"Name: {name}",
font=font_medium, fill="black")
```

```
        draw.text((30, 230), f"ID Num: {id_number}",
font=font_medium, fill="black")
        draw.text((30, 280), f"Department: {department}",
font=font_medium, fill="black")
        draw.text((30, 330), f"Major: {major}",
font=font_medium, fill="black")


        # Generate and save the barcode using
python-barcode
        EAN = barcode.get_barcode_class('code128')
        ean = EAN(id_number, writer=ImageWriter())
        barcode_path =
os.path.join(os.path.dirname(output_path), 'barcode')
        ean.save(barcode_path)
        barcode_path += '.png'
        print(f"Barcode saved at {barcode_path}")


        # Check if the barcode file was created
successfully
        if not os.path.exists(barcode_path):
            raise FileNotFoundError(f"Barcode file not
created at {barcode_path}")


        # Load and crop the barcode image to fit into a
rectangle
        barcode_image = Image.open(barcode_path)
        barcode_width, barcode_height =
barcode_image.size
        crop_height = int(barcode_height / 2)
        barcode_cropped = barcode_image.crop((0, 0,
barcode_width, crop_height))


        # Resize the cropped barcode image to fit the
desired dimensions
        desired_width, desired_height = 300, 120  #
Adjust dimensions as needed
```

```
        barcode_resized =
barcode_cropped.resize((desired_width, desired_height),
Image.LANCZOS)
        barcode_position = (470, 280)  # Adjust position
as needed
        card.paste(barcode_resized, barcode_position)


        # Save the ID card
        card.save(output_path)
        print(f"ID card saved at {output_path}")
    except Exception as e:
        print(f"An error occurred: {e}")


# Usage example
database = r"D:\Spring 2024\GP2\ReadyCodes\members.db"
conn = create_connection(database)
create_table(conn)
logo_path = r'D:\Spring 2024\GP2\ID
template\LOGO_FOOTER.png'
first_name = "Name"
last_name = "Name"
id_number = "1111111"
photo_path = r'path to the face photo'
department = "Engineering"
major = "Computer engineering"
output_dir = r'D:\Spring 2024\GP2\ReadyCodes\IDs'  #
Directory where ID cards will be saved


member = (first_name, last_name, id_number, photo_path)
add_member(conn, member, department, major, logo_path,
output_dir)


# Close the database connection
if conn:
    conn.close()
```

Appendix I: **FaceRec**

```
import face_recognition
import cv2
import time
import os


def perform_face_recognition(photo_path):
    # Load the known image and encode it
    known_image =
face_recognition.load_image_file(photo_path)
    known_encodings =
face_recognition.face_encodings(known_image)


    if not known_encodings:
        print(f"Error: No faces found in the image
{photo_path}")
        return None, None


    known_encoding = known_encodings[0]


    # Capture a video frame
    video_capture = cv2.VideoCapture(0)
    match_start_time = None
    no_match_start_time = None
    match_duration = 3  # Match duration in seconds
    no_match_duration = 3  # No match duration in seconds


    while True:
        ret, frame = video_capture.read()
        if not ret:
            continue
```

```
        # Convert the frame to RGB
        rgb_frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)


        # Find all faces and face encodings in the frame
        face_locations =
face_recognition.face_locations(rgb_frame)
        face_encodings =
face_recognition.face_encodings(rgb_frame,
face_locations)


        match_found = False


        # Check each face in the frame
        for face_encoding, face_location in
zip(face_encodings, face_locations):
            match =
face_recognition.compare_faces([known_encoding],
face_encoding, tolerance=0.6)
            distance =
face_recognition.face_distance([known_encoding],
face_encoding)


            top, right, bottom, left = face_location
            match_percentage = (1 - distance[0]) * 100


            if match[0]:
                if match_start_time is None:
                    match_start_time = time.time()
                elapsed_time = time.time() -
match_start_time
                if elapsed_time >= match_duration:  #
Match found for at least match_duration seconds
```

```
                    cv2.rectangle(frame, (left, top),
(right, bottom), (0, 255, 0), 2)
                    cv2.putText(frame, f'Match -
{match_percentage:.2f}%', (left, top - 10),
cv2.FONT_HERSHEY_SIMPLEX,
                                0.5, (0, 255, 0), 2)
                    print("Match found, stopping
recognition.")
                    video_capture.release()
                    cv2.destroyAllWindows()
                    return
                match_found = True
                no_match_start_time = None  # Reset no
match timer
            else:
                cv2.rectangle(frame, (left, top), (right,
bottom), (0, 0, 255), 2)
                cv2.putText(frame, f'No Match -
{match_percentage:.2f}%', (left, top - 10),
cv2.FONT_HERSHEY_SIMPLEX,
                            0.5, (0, 0, 255), 2)
                match_start_time = None  # Reset match
timer


        if match_found:
            no_match_start_time = None
        else:
            if no_match_start_time is None:
                no_match_start_time = time.time()
            elapsed_time = time.time() -
no_match_start_time
            if elapsed_time >= no_match_duration:
                # Save the frame as no match found for
the specified duration
                timestamp =
time.strftime("%Y%m%d-%H%M%S")
                output_path = os.path.join(r"D:\Spring
2024\GP2\ID template", f"NoMatch_{timestamp}.jpg")
                cv2.imwrite(output_path, frame)
```

```
                print(f"No match found, image saved to
{output_path}")
                no_match_start_time = None  # Reset timer
after saving
                # Optionally break the loop after saving
the image
                video_capture.release()
                cv2.destroyAllWindows()
                return


        cv2.imshow('Video', frame)


        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


    video_capture.release()
    cv2.destroyAllWindows()
```