ESTABLISHING CONSENSUS IN MULTI-AGENT SYSTEMS VIA REINFORCEMENT LEARNING TECHNIQUE

Dissertation Phase-2 Report

Submitted by

IRFAN E

REG NO : TKM22MEAI08

to

the APJ Abdul Kalam Technological University in partial fulfillment for the award of the degree of

MASTER OF TECHNOLOGY
IN
Artificial Intelligence

Under the guidance of
Dr. Resmi R

Centre for Artificial Intelligence

TKM College of Engineering Kollam

JUNE 2024

Thangal Kunju Musaliar College of Engineering
Centre for Artificial Intelligence

C E R T I F I C A T E

This is to certify that, this report titled ***ESTABLISHING CONSENSUS IN MULTI-AGENT SYSTEMS VIA REINFORCEMENT LEARNING TECHNIQUE*** is a bonafide record of the **Dissertation Phase-2** presented by **Irfan E (TKM22MEAI08)**, under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, **M. Tech in Artificial Intelligence** in **APJ Abdul Kalam Technological University**.

Internal Supervisor	Project Coordinator	Head of the Department
Dr. Resmi R Assistant Professor Department of EEE TKMCE	Dr. Sumod Sundar Associate Professor Centre for AI TKMCE	Dr. Imthias Ahamed T P Professor Centre for AI TKMCE

ACKNOWLEDGEMENT

A successful dissertation is a fruitful culmination of efforts by many people, some directly involved and some others indirectly, by providing support and encouragement. Firstly I would like to thank the almighty for giving me the wisdom and grace for making my project a memorable one. I thank him for steering me to the shore of fulfillment under his protective wings.

I extend my heartfelt appreciation to **Dr. T. A. Shahul Hameed**, Principal of T.K.M College of Engineering, for granting me the opportunity to showcase my dissertation phase-2. I would like to thank **Dr. Imthias Ahamed T P**, Professor and Head of the Department, Centre for Artificial Intelligence, TKM College of Engineering, Kollam, for his constant support and encouragement throughout the work.

With a profound sense of gratitude, I would like to express my heartfelt thanks to my internal supervisor **Dr. Resmi R**, Assistant Professor, Department of Electrical and Electronics Engineering and Project Coordinator, **Dr. Sumod Sundar**, Associate Professor, Centre for Artificial Intelligence(AI), TKM College of Engineering, Kollam for their expert guidance, cooperation, and immense encouragement. I also extend my thanks to the entire faculty and staff members of the Centre for AI, TKMCE, who have encouraged me throughout this work.

I also express my thanks to my loving parents and friends, for their support and encouragement in the successful completion of this work.

Irfan E

Abstract

A multi agent system(MAS) in the context of artificial intelligence refers to a system composed of multiple interacting agents, which can be entities or robots. These agents work together or compete to achieve individual or collective goals. Consensus is a decision making process that aims to achieve the agreement of the majority of entities involved, it involves collaborative efforts to reach a decision that is mutually acceptable to all involved. Reinforcement Learning is an AI algorithm utilized with robots and autonomous vehicles to learn optimal behaviors by interacting with simulated environments, thereby facilitating safer and more efficient real world decision making. In this work, the focus is on the application of RL technique Q-learning using E-puck robots. The E-puck employs Q-learning to determine the most suitable actions to take within its environment. The concept of consensus is examined within the context of multi agent systems, where E-puck robots constitute the agents. Consensus, in this scenario, pertains to the agreement reached among the E-puck agents regarding their relative positions or distances. Establishing consensus among MAS and ensuring efficient communication and coordination among E-puck robots pose significant challenges, yet achieving it holds immense potential for enhancing MAS and optimizing automation and decision making processes. Using RL algorithm along with webots simulation and E-puck robots the frameworks can help different agents to navigate toward a common destination while avoiding obstacles and maintaining consensus. This approach will be beneficial for various sectors, including healthcare assistance, defense, warehouse management, and even agricultural automation.

Contents

List of Figures

Chapter 1

Introduction

Multi Agent Systems(MAS) represent an approach to problem solving by using the collective intelligence and capabilities of multiple interacting entities or agents. These systems are deployed across various domains, ranging from robotics and autonomous vehicles to complex networked environments like social networks and distributed computing systems. Within MAS, agents are autonomous entities capable of perceiving their environment, making decisions, and taking actions to achieve their objectives. The signature of MAS lies in the interconnections and interdependence of these agents, as they collaborate and compete within their environment. Each agent typically operates based on its own goals, constraints, and information, which may often be incomplete or conflicting with those of other agents. Consequently, the coordination and synchronization of these diverse agents pose significant challenges, involve mechanisms for establishing consensus and aligning individual decisions or states toward common objectives. Consensus within MAS involves reaching an agreement or convergence among the participating agents despite their differences in goals, preferences, or information. This consensus is not merely about compromising but rather about promoting cooperation and synergy among the agents to enhance overall system performance. Achieving consensus typically involves iterative processes of communication, negotiation, and possibly compromise, where agents exchange information, update their beliefs, and adjust their behaviors to converge towards a shared understanding or course of action.

There are several control systems for Multi Agent Systems like Centralized Control, Decentralized Control, Distributed Control, Hierarchical Control, Market-Based Control, Learning-Based Control, Swarm Intelligence, Game-Theoretic Control; these control systems play crucial roles in orchestrating agent behaviors and interactions within MAS. These control systems have a spectrum of approaches, from centralized to decentralized paradigms, each influencing how agents communicate, make decisions, and synchronize their actions towards shared objectives. Centralized control entails a single entity, often referred to as a controller or coordinator, dictating actions or decisions for the entire system. This approach offers the advantage of efficiency by enabling coordinated and coherent actions across all agents. However, centralized control faces scalability challenges, particularly as the number of agents or complexity of interactions increases. The overhead of processing information and issuing commands for numerous agents can limit the applicability in large scale MAS scenarios. In contrast, decentralized control distributes decision making authority among individual agents, empowering them with autonomy to make local decisions based on their

observations and objectives. Decentralization promotes scalability and robustness by reducing the burden on any single entity and enabling agents to adapt flexibly to changing conditions. However, decentralized control presents challenges in achieving consensus among agents with diverse goals, preferences, and information, as each agent pursues its own objectives independently. Amidst the spectrum of control paradigms in MAS, Reinforcement Learning (RL) emerges as a dynamic and adaptive approach to decision-making and control. RL integrates learning mechanisms with decision-making processes, allowing agents to improve their behaviors over time through iterative experiences and feedback from their environment. By optimizing decision policies to maximize cumulative rewards, RL enables agents to adapt to complex and uncertain environments, making it well suited for MAS applications.

RL represents a subset of machine learning focused on making optimal decisions to maximize rewards within a given context or environment. It revolves around the idea of learning through trial and error, where an agent interacts with its environment, taking actions and receiving feedback in the form of rewards or penalties. The goal is to learn a policy that guides the agent to select actions that lead to the highest possible cumulative reward over time. In scenarios where consensus needs to be established among multiple agents within an environment, RL. By iteratively interacting with the environment and learning from the resulting rewards, agents can adapt their behaviors to align with shared objectives and achieve consensus. Some of the RL algorithms include Q-Learning, SARSA, Deep Q-Networks (DQN), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO). One commonly used reinforcement learning algorithm for such scenarios is Q-learning. In Q-learning, each state-action pair is associated with a Q-value, which represents the expected cumulative reward that the agent can obtain by taking a particular action in a specific state. Initially, these Q-values are often set to zero or some arbitrary value. During the learning process, the agent explores different actions and observes the rewards obtained. Based on these experiences, it updates its Q-values using a formula that incorporates the observed rewards and the estimated future rewards from subsequent states. This process continues iteratively, with the agent gradually refining its Q-values through repeated interactions with the environment. By learning and updating Q-values over time, the agent can converge towards optimal policies that maximize cumulative rewards, thereby guiding its decision making towards consensus building with other agents. Through this iterative process of exploration and exploitation, reinforcement learning enables agents to navigate complex environments and coordinate their actions effectively to achieve common goals.

When multiple agents interact with distinct or common goals, achieving the consensus among them can become a complex challenge. The challenge is to converge the decisions of the multiple agents towards a shared decision, value or state not considering the initial differences in information or objectives.

The implementation of Q-learning within the Webots software platform provides a versatile environment for simulating multi-agent interactions. Webots allows users to create customized simulation environments tailored to their specific needs, including the addition of obstacles, control of geometry, and integration of various elements and properties. In this setup, the controller design is facilitated through Python scripting, offering flexibility and ease of development. Python scripts can be used to define the behavior of agents, including

how they perceive their environment, make decisions, and interact with other agents. The Q-learning algorithm, a fundament of reinforcement learning, is employed within this framework to enable agents to learn optimal policies for decision making. Bellman's equation, a fundamental principle in dynamic programming, serves as the basis for updating Q-values in the controller. By iteratively applying Bellman's equation to the observed rewards and estimated future rewards, the Q-values are adjusted, allowing agents to gradually improve their decision making strategies over time. The choice of agent for the simulation involves using E-puck robots, which are open-source robots equipped with proximity sensors. These sensors provide readings that enable agents to perceive their surroundings and make informed decisions based on the proximity of obstacles or other agents. By utilizing the sensor data, agents can set thresholds for obstruction and navigate the environment effectively. Overall, the combination of Webots simulation software, Python scripting for controller design, and the integration of E-puck robots with proximity sensors offers a robust platform for implementing and experimenting with Q-learning algorithms in simulated multi agent environments. This setup enables to explore and optimize reinforcement learning techniques for consensus building and cooperative behavior among autonomous agents.

Chapter 2

Literature Survey

MASs are intricate frameworks comprising multiple autonomous agents interacting within dynamic environments, ranging from robotic systems to complex networks. The fundamental challenge within MAS lies in achieving consensus among these agents, where they must coordinate their actions to achieve mutual agreement or shared objectives. The imperatives for consensus are effective communication, coordination, and decision making strategies among the agents. Reinforcement learning techniques, such as Q-learning and policy gradient methods, serve as pivotal tools in refining control strategies to facilitate consensus within MAS [2][4].

The integration of neural networks enhances the capabilities of MAS, particularly in domains such as mobile robotics, where obstacle avoidance is crucial. Neural networks enable agents to perceive and navigate complex environments, contributing to consensus building by facilitating seamless coordination and cooperation. Moreover, RL techniques have proven effective in managing intricate tasks within power distribution networks, ensuring consensus among agents while optimizing energy flow and system stability [5].

Reinforcement learning emerges as a transformative approach in addressing the complexities within MAS. The work by Wang and Li [1] showcases the efficiency of model free RL in achieving fully cooperative consensus among MASs. This highlights the significance of RL techniques in optimizing cooperative behaviors without explicit models. Similarly, Mu et al. [2] delve into Q-learning methodologies for optimal consensus control in discrete time MASs, emphasizing RL's role in attaining optimal control strategies for achieving consensus among discrete time agents.

Reinforcement learning acts as a powerful mechanism for addressing complex challenges within MAS, enabling agents to learn and adapt to dynamic environments through interaction and feedback mechanisms. The studies by Wang and Li [1] and Mu et al. [2] underscore the versatility of RL techniques in optimizing cooperative behaviors and control strategies among multiple agents. This adaptability extends beyond consensus problems, as showcased by Duguleana and Mogan [3], emphasizing RL's application in mobile robot navigation by combining neural networks with RL for obstacle avoidance, crucial for autonomous movement in uncertain environments. Similarly, Yang, Zhang, and Wang [4] emphasize the data driven nature of RL employing policy gradient methods to optimize control policies for MAS,

further solidifying RL's significance by introducing a consensus multi-agent RL framework tailored to manage volt-var control in power distribution networks, highlighting its potential in improving critical infrastructure operations.

Recent advancements in MAS have seen the emergence of novel trends that further underscore the importance of reinforcement learning. One such trend is the integration of machine learning techniques, including deep reinforcement learning, which enables agents to learn more complex and abstract behaviors from high-dimensional sensory inputs. This integration has led to significant breakthroughs in domains such as autonomous driving, where agents must navigate dynamic environments with diverse and unpredictable scenarios. Additionally, the rise of swarm robotics, where large numbers of simple agents cooperate to achieve complex tasks, presents new challenges and opportunities for reinforcement learning-based approaches to consensus-building and coordination. As MAS continue to evolve and grow in complexity, scalability becomes a concern. Traditional approaches to consensus and coordination may struggle to scale to systems comprising thousands or even millions of agents. Reinforcement learning offers promising avenues for addressing scalability challenges by enabling agents to learn and adapt autonomously, reducing the need for centralized coordination and control. Techniques such as hierarchical reinforcement learning and multi agent reinforcement learning provide mechanisms for decomposing complex tasks into smaller, more manageable sub tasks, allowing agents to collaborate effectively while scaling to larger systems.

The deployment of MAS in real-world scenarios raises important ethical and societal considerations. As agents become more autonomous and capable of making decisions that impact human lives, ensuring fairness, transparency, and accountability in their behaviors becomes crucial. Reinforcement learning algorithms must be carefully designed and regulated to prevent unintended consequences, biases, or unethical behaviors. Moreover, fostering trust and collaboration between humans and autonomous agents is essential for the successful integration of MAS into society, highlighting the need for interdisciplinary research and collaboration between experts in fields such as ethics, law, and computer science.

In conclusion, the adaptability of reinforcement learning across diverse domains within MAS significantly contributes to advancing MAS capabilities in real-world scenarios. The comprehensive examination of various studies underscores the multifaceted applications and methodologies of RL within MAS. These applications range from achieving cooperative consensus among agents to addressing domain- pecific challenges such as obstacle avoidance in robotics and control optimization in power distribution networks. Similarly, data driven RL methods optimize control policies for MAS, while a consensus multi agent RL framework tailored for power distribution networks highlights RL's potential in critical infrastructure operations. Moreover, the integration of RL with machine learning techniques like deep RL enables agents to learn complex behaviors from high dimensional inputs. Additionally, the rise of swarm robotics presents new challenges and opportunities for RL based approaches to consensus building and coordination. Scalability concerns in MAS are addressed through techniques like hierarchical RL and multi-agent RL, which decompose complex tasks into manageable subtasks. The deployment of MAS in real-world scenarios raises important ethical and societal considerations. Ensuring fairness, transparency, and accountability in

autonomous agents' behaviors is crucial, requiring careful design and regulation of RL algorithms. RL's adaptability across various MAS domains is pivotal in advancing their capabilities in real world scenarios. The diverse applications and methodologies of RL within MAS underscore its versatility and potential in addressing real world complexities.

Chapter 3

Methodology

3.1 Objectives

- To develop a webots simulation world where E-puck robots can interact with the environment.
- To develop suitable algorithm to attain consensus of Multi Agent System.
- To implement Q-learning approach with the E-puck robots in the simulated world.
- To extend the control algorithm to achieve obstacle avoidance by the multi-agents.

3.2 Proposed Framework

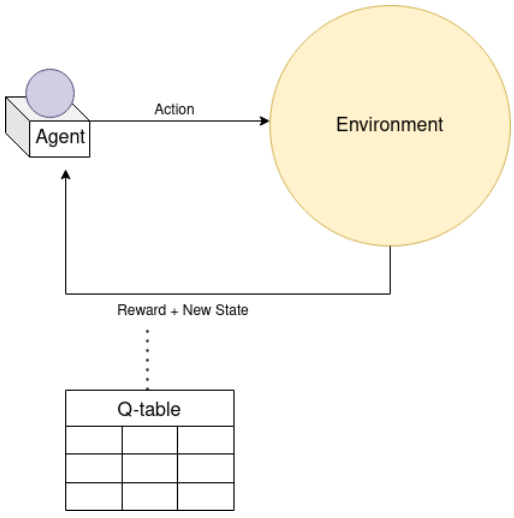

Figure 3.1: Proposed Framework for Q-learning

The proposed framework for Q-learning, as depicted in Figure 3.1, employs an agent and an environment within a simulation setup to implement reinforcement learning. In this framework, the agent utilizes E-puck open source robot as a tool to interact with the simulated environment. During these interactions, the agent executes actions and receives feedback in the form of rewards, which are used to calculate the effectiveness of the actions taken. The agent observes changes in the environment and transitions to new states accordingly.

The core of Q-learning is integrated into the controller using Python scripts. Within this setup, Q-values are assigned to each possible state action pair, representing the expected future rewards associated with taking a specific action in a given state. Initially, these Q-values are initialized to zeroes. As the agent continues to interact with the environment and accumulate experience, the Q-values are updated based on the observed rewards and transitions between states. Over multiple iterations of interaction and learning, the Q-values gradually converge towards their optimal values, reflecting the best possible decisions to make in each state. Convergence implies that the agent's policy, or strategy for selecting actions, approaches the optimal policy that maximizes the cumulative reward over time.

In summary, the proposed framework for Q-learning combines the use of an agent, environment simulation, and Python scripts to implement and optimize the agent's decision making process, ultimately aiming to achieve optimal performance in the given task or environment.

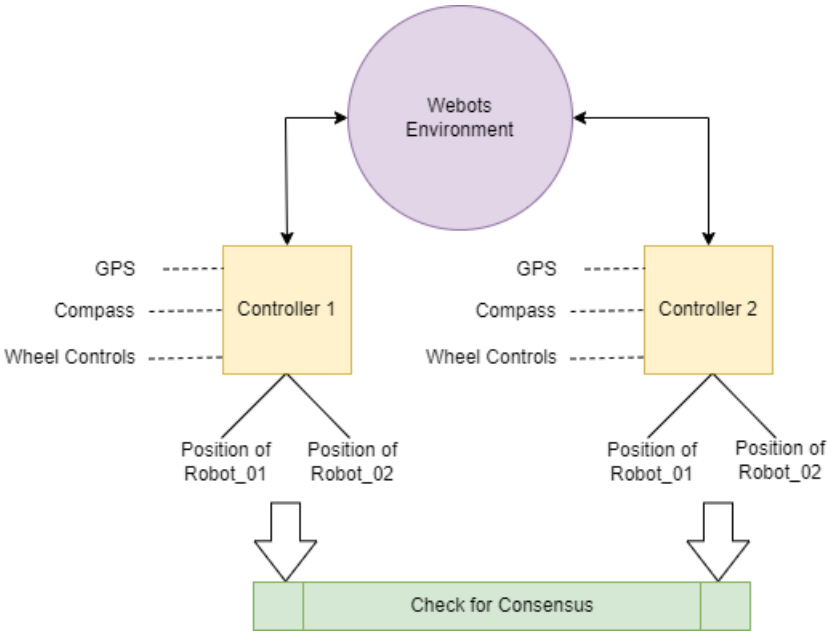

Figure 3.2: Proposed Framework for muti-agent consensus

Figure 3.2 illustrates the proposed framework for multi-agent consensus, where multiple agents collaborate to achieve a common goal, specifically maintaining a desired distance between them. Consensus, in this context, refers to the agreement among the agents regarding their relative positions, with the goal of ensuring that they remain within a predefined threshold distance from each other while moving towards their destination.

The agents are equipped with various sensors and capabilities to facilitate this task. The agents utilize GPS (Global Positioning Sensor) to determine their absolute positions, compasses to assess their orientations to evaluate the consensus status at each time step. These sensors enable the agents to continuously monitor their proximity to one another and adjust their movements accordingly to maintain the desired distance.

Throughout the collaboration, the agents strive to minimize the distance between them while ensuring that it remains within the specified threshold. This process involves constant communication and coordination among the agents to synchronize their movements and achieve consensus effectively. By leveraging their sensing capabilities and communication protocols, the agents can dynamically adapt their behaviors in response to changes in the environment and the positions of other agents. This adaptability allows them to navigate complex scenarios and maintain consensus even in challenging conditions, ultimately facilitating the successful achievement of the collective objectives.

3.3 Multi Agent Systems

MAS consist of multiple interacting agents, each capable of autonomous action in a shared environment to achieve individual or collective goals. These agents are typically autonomous entities with their own goals, abilities, and decision-making processes. They can perceive their environment, make decisions, and take actions based on their internal state and external stimuli. The environment in which these agents operate can be static or dynamic, discrete or continuous, and deterministic or stochastic. Interaction among agents is a fundamental aspect of MAS, encompassing communication, coordination, cooperation, and competition. Agents communicate to exchange information, coordinate their actions to avoid conflicts, cooperate to achieve common goals, and sometimes compete when their individual goals conflict. The organizational structure of MAS can vary from hierarchical to flat, or even hybrid, where agents may have equal status or follow a layered structure with higher-level agents overseeing lower-level ones. These organizational structures help in managing the complexity and scalability of MAS, making them suitable for solving problems that are difficult or impossible for a single agent to handle.

There are several types of MAS, each tailored to specific applications and challenges. Homogeneous MAS consist of agents that are identical in terms of capabilities and objectives. These systems are often simpler to design and manage, as all agents follow the same protocols and behaviors. This uniformity allows for easier implementation of coordination

strategies, but it may limit the system's flexibility and adaptability. Heterogeneous MAS, in contrast, include agents with different capabilities, roles, and objectives. This diversity can enhance the system's robustness and ability to handle complex tasks, as different agents can specialize in various aspects of the problem. However, it also requires more sophisticated coordination and communication mechanisms to ensure that the agents can work together effectively. Cooperative MAS focus on agents working together to achieve shared goals, emphasizing collaboration and joint actions. In such systems, agents may share information and resources, coordinate their efforts, and form alliances to maximize the overall benefit. Competitive MAS, on the other hand, involve agents with conflicting goals, leading to competition and strategic behaviors. These systems often require the development of sophisticated negotiation and conflict resolution techniques to manage the interactions between competing agents. Hybrid MAS combine elements of both cooperation and competition, where agents may need to cooperate in some scenarios and compete in others. This type of system reflects many real-world situations where agents must balance cooperative and competitive behaviors to achieve their objectives.

The design and implementation of MAS involve addressing several challenges, such as scalability, robustness, coordination, cooperation, and conflict resolution. Scalability ensures that the system can handle a large number of agents without performance degradation. This is particularly important in applications like smart grids and large-scale robotics, where the number of agents can be very high. Robustness ensures that the system can continue functioning effectively despite failures or changes in the environment. This is crucial for applications in dynamic and unpredictable environments, such as disaster response and autonomous vehicle coordination. Coordination and cooperation mechanisms are essential for enabling agents to work together effectively, particularly in heterogeneous and cooperative MAS. Techniques like market-based approaches, contract nets, and shared plans can be used to manage the coordination of multiple agents. Conflict resolution strategies are crucial in competitive and hybrid MAS to manage and resolve conflicts arising from competing goals or limited resources. These strategies might include negotiation, arbitration, and game-theoretic approaches. Learning mechanisms, such as reinforcement learning and social learning, are often integrated into MAS to enable agents to adapt and improve their performance over time. Through these learning processes, agents can optimize their behaviors, improve their strategies, and enhance their overall effectiveness in achieving their goals. By leveraging these capabilities, MAS provide powerful solutions for complex, distributed, and dynamic problems, making them an essential component of modern intelligent systems.

3.4 Consensus

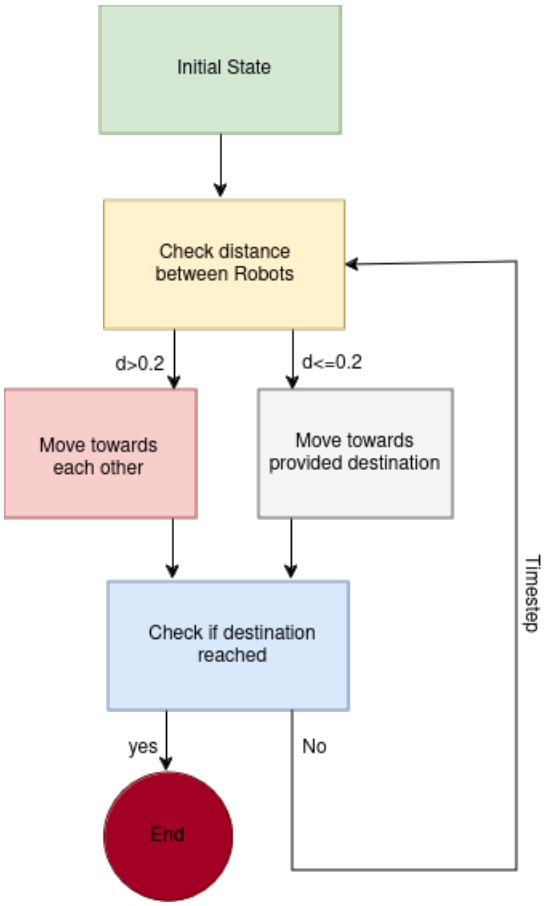

Figure 3.3: Consensus

Figure 3.3 shows how the consensus framework works. The agents are equipped with a coordination mechanism designed to facilitate their collective movement towards a predetermined destination. Each agent, initially dispersed across separate locations, operates autonomously to engage in collaborative behavior. Once consensus is achieved. the agents aligns their trajectories towards the shared destination. To ensure effective coordination, the agents continuously monitor their spatial relationships with one another. If the distance between any pair of agents exceeds a predefined threshold of 0.2 meters, they enact corrective actions to reduce this gap. This mechanism prompts the agents to converge towards each other, dynamically adjusting their positions and trajectories to maintain a close proximity. Once the agents are within the prescribed vicinity of 0.2 meters, they transition into a synchronized mode of operation. In this state, they harmonize their movements and advance towards the common destination as a cohesive unit. By moving in unison, the agents opti-

mize their collective progress while minimizing the risk of losing the desired path or losing cohesion. Throughout the journey, the synchronization mechanism remains active, continually monitoring and adjusting the agents' movements to uphold the prescribed proximity. This iterative process ensures that the agents remain tightly coordinated, effectively navigating obstacles and dynamic environments as they progress towards their goal. In essence, the synchronization mechanism fosters a synergistic relationship among the agents, enabling them to function as a unified entity.

In the journey towards the shared destination, the robots encounter various obstacles that pose potential challenges to their progress. To navigate their environment safely and efficiently, the robots employ obstacle avoidance strategies. These strategies are grounded in the utilization of a Q-table, a key component of their decision making framework. The Q-table serves as a repository of mapped actions derived from training in the simulated environment. Each entry in the Q-table corresponds to a specific state of the environment and the optimal action to undertake in that state. When confronted with an obstacle, the robots consult the Q-table to determine the most appropriate course of action. By analyzing the current state of the environment and referencing the Q-table, the robots can make informed decisions on how to navigate around obstacles while still progressing towards their destination. This process allows the robots to dynamically adapt their movements, selecting actions that prioritize both obstacle avoidance and maintaining consensus with other robots. The Q-table framework enables the robots to learn from their interactions with the environment over time. Through iterative exploration and reinforcement learning, the robots continually refine their understanding of optimal strategies for obstacle avoidance in diverse scenarios. This adaptive learning process enhances the robots' ability to respond effectively to novel challenges and evolving environmental conditions. This framework empowers the robots to collaborate effectively and coordinate their movements towards a shared goal. By leveraging the Q-table to navigate obstacles while maintaining consensus with their peers, the robots can traverse complex environments with efficiency. This combination of collaborative coordination and dynamic adaptation ensures that the robots can safely navigate their surroundings while making progress towards their ultimate objective.

3.5 Tools Used

3.5.1 Webots Software

Webots is an open source, multi platform desktop application utilized for simulating robots. This comprehensive tool provides a complete development environment tailored for modeling, programming, and simulation within a 3D environment. Originally developed in 1998 by Cyberbotics Ltd., Webots has since been continuously maintained as their primary product.

Webots serves a wide range of users across industries, educational institutions, and re-

search settings due to its professional grade features and versatility. In robotics research, it facilitates the creation of complex robotic systems and enables researchers to simulate sensors, test algorithms, and evaluate behaviors within a virtual environment. This virtual testing environment is invaluable as it allows developers to iterate and refine their designs before deploying them onto physical robots. Webots includes a large collection of freely modifiable models of robots, sensors, actuators and objects. In addition, it is also possible to build new models from scratch or import them from 3D CAD software. When designing a robot model, the user specifies both the graphical and the physical properties of the objects. The graphical properties include the shape, dimensions, position and orientation, colors, and texture of the object. The physical properties include the mass, friction factor, as well as the spring and damping constants. Simple fluid dynamics is present in the software.

Key features of Webots include its support for various robot models, sensors, actuators, and programming languages. This flexibility enables users to simulate a diverse range of robotic platforms and experiment with different hardware configurations and control strategies. Moreover, its support for multiple programming languages allows developers to work with the language of their choice, enhancing accessibility and flexibility in development workflows. In educational settings, Webots serves as a valuable tool for teaching robotics concepts and facilitating hands on learning experiences. Its intuitive interface and comprehensive documentation make it accessible to students and educators alike, enabling them to explore robotics principles and experiment with different robotics scenarios in a virtual environment.

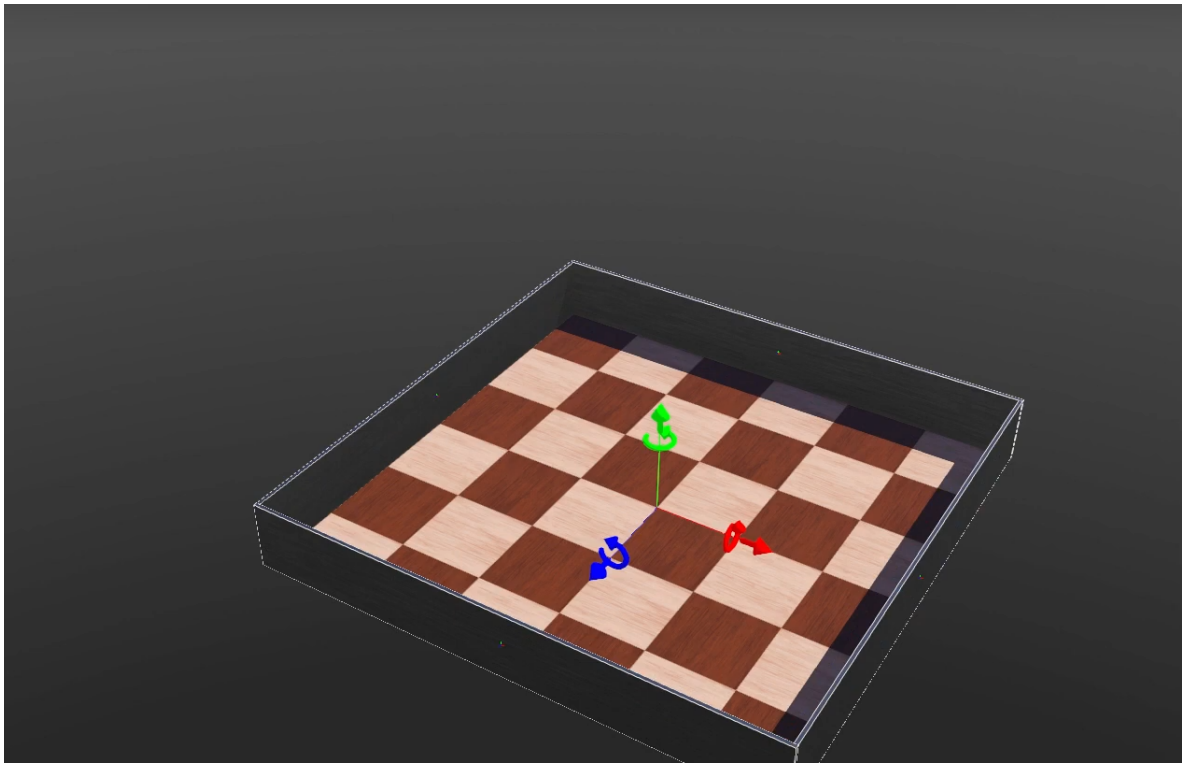

Figure 3.4: A sample environment from Webots

Figure 3.4 illustrates a sample floor within the Webots simulation environment, measuring 10 meters by 10 meters. Within this simulated space, users have the capability to add various elements such as nodes, robots, sensors, actuators, and other objects to create a dynamic and interactive simulation environment.

- Nodes: In Webots, nodes represent various elements within the simulation environment, including robots, sensors, controllers, and objects. These nodes can be added, configured, and interconnected to create complex robotic systems and environments.
- Robots: Users can add different robot models to the simulation environment, each with its own set of sensors, actuators, and control mechanisms. These robots can be programmed to perform specific tasks, navigate the environment, interact with objects, and communicate with other robots or external systems.
- Controller: The controller serves as the brain of the simulation, responsible for orchestrating the behavior of the robots and other elements within the environment. It can be programmed using languages such as C, C++, Python, or MATLAB to implement various control algorithms, decision-making processes, and interaction protocols.
- Collective Control: Webots provides tools for collectively controlling the simulation, allowing users to start, pause, resume, and stop the simulation as well as adjusting simulation parameters and settings. This collective control enables users to observe and analyze the behavior of the simulated robots and environment in real-time.

GUI

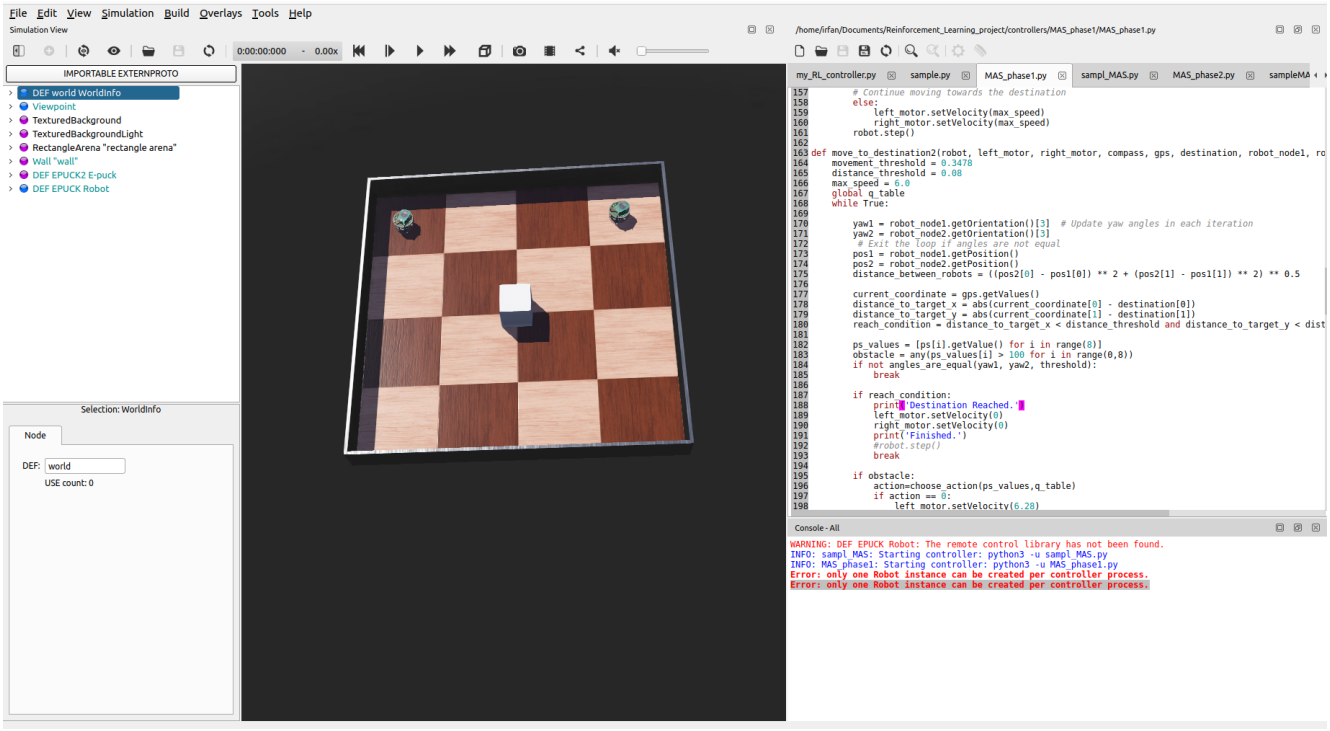

Figure 3.5: Webots GUI

Figure 3.5 shows the GUI window of the webots software where we can add different nodes, control the robots and run various scripts with a dedicated console. The main components of the GUI are:

- 3D Simulation Window: This is the primary workspace where the 3D simulation takes place. Users can visualize the robot and its environment from different angles and perspectives. The window supports real-time rendering and interaction, allowing users to observe and manipulate the simulation dynamically.
- Scene Tree: The Scene Tree is a hierarchical representation of all the objects in the simulation, including robots, sensors, actuators, and environmental elements. Users can select, modify, and configure properties of each object directly from the Scene Tree.
- Text Editor: Webots includes an integrated text editor for writing and editing controller programs that define the behavior of the robots. It supports multiple programming languages, including C, C++, Python, Java, and MATLAB.
- Console Window: This window displays output messages, error logs, and debugging information from the simulation and the controller programs. It helps users monitor the execution of their code and troubleshoot issues.

- 3D View Manipulation Tools: Tools for panning, zooming, and rotating the 3D view to inspect the simulation environment from various angles. Users can also switch between different camera perspectives.
- Simulation Control Buttons: Buttons to control the simulation execution, such as play, pause, step, and reset. These controls allow users to manage the simulation flow and analyze specific moments in detail.
- Robot Window: A customizable window that can display real time data from the robot's sensors and actuators. It provides an interface for monitoring the internal states of the robot and interacting with it during the simulation.

3.5.2 E-puck Robot

The e-puck (short for "evolutionary puck") is a small, differential drive robot designed for research and education in the field of robotics. Developed by EPFL (École Polytechnique Fédérale de Lausanne).

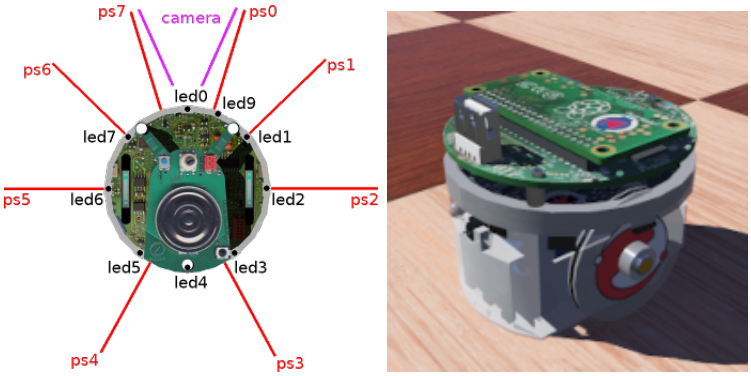

Figure 3.6: E-puck mobile robot

The figure 3.6 shows the E-puck robot. The E-puck mobile robot has 8 proximity sensors attached to the structure which helps to calculate proximity values. The proximity values are computed using infrared sensors. Higher proximity values implies that the structure is close to the obstruction. The differential wheel controls of the mobile robot allows to rotate in its axis. The maximum speed with which the robot can move is 6.28m/s.

- Diameter: 70 mm
- Height: 50 mm
- Weight: 200 g
- Max speed: 6.28 cm/s
- Autonomy: 2 hours moving
- dsPIC 30 CPU @ 30 MHz (15 MIPS)

- 8 KB RAM
- 144 KB Flash
- 2 step motors
- 8 infrared proximity and light (TCRT1000)
- color camera, 640x480
- 8 LEDs in ring + one body LED + one front LED
- 3D accelerometers
- 3 microphones
- 1 loudspeaker

3.5.3 Peripherals of E-puck

The following peripherals are attached to the robot for the experiment;

- GPS (Global Position Sensor): The GPS sensor is an essential component that enables the robot to determine its precise location coordinates within the environment. By receiving signals from orbiting satellites, the GPS sensor calculates the robot's latitude, longitude, and sometimes altitude. These coordinates provide the robot with accurate information about its current position relative to a global reference frame. This data is crucial for various robotic tasks, such as navigation, localization, mapping, and path planning. By leveraging GPS information, the robot can effectively navigate through the environment, avoid obstacles, and reach designated destinations with precision.
- Compass: The compass is a sensor installed on the robot that serves to determine its heading or orientation relative to the Earth's magnetic field. By measuring the Earth's magnetic field's direction, the compass provides the robot with information about its current heading, typically expressed in degrees relative to the magnetic north. This heading information is crucial for the robot to understand its orientation and make appropriate navigation decisions. For example, when navigating towards a specific direction or following a predefined path, the robot can use the compass data to ensure it maintains the correct heading. Additionally, the compass data can be utilized to adjust the robot's orientation, such as rotating or aligning itself in a particular direction to achieve its objectives accurately. Overall, the compass sensor enhances the robot's ability to navigate autonomously and perform tasks that require precise orientation control within the environment.

3.5.4 Reinforcement Learning

Reinforcement Learning is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative reward. Unlike supervised learning, where the model is trained on a fixed dataset, RL involves learning through interaction with the environment. The agent takes actions based on a policy, receives feedback in the form of rewards, and uses this feedback to improve its policy over time. The goal is to

find the optimal policy that maximizes the expected cumulative reward over the long run. RL is inspired by behavioral psychology and involves key concepts such as states, actions, rewards, and policies, typically formalized through Markov Decision Processes.

RL algorithms can be broadly classified into Model-Based and Model-Free categories. Model-based algorithms involve creating a model of the environment, which is then used to simulate and plan actions. Examples include Dynamic Programming (DP), which uses a known model of the environment and includes algorithms like Policy Iteration and Value Iteration, suitable for small-scale problems with a fully known model, and Monte Carlo Methods, which learn from complete episodes of interaction with the environment, estimating the value of states or state-action pairs based on average returns. Another model-based approach is Dyna-Q, which combines model-free learning with model-based planning, learning a model from interactions and using it to simulate additional experiences to speed up learning. On the other hand, model-free algorithms do not require a model of the environment and learn directly from interactions. These can be further divided into Value-Based and Policy-Based algorithms. Value-Based algorithms, such as Q-learning and SARSA (State-Action-Reward-State-Action), focus on learning value functions that estimate the expected return of states or state-action pairs. Q-learning is an off-policy algorithm that learns the value of action-value pairs (Q-values), updating them using the Bellman equation and the maximum expected future reward, and is known for its convergence to the optimal policy given sufficient exploration. SARSA is an on-policy algorithm that updates Q-values based on the action actually taken by the agent, learning from the current policy. Policy-Based algorithms, like REINFORCE and Actor-Critic Methods, focus on learning the policy directly. REINFORCE is a Monte Carlo policy gradient method that uses the gradient of expected return to adjust policy parameters, suitable for continuous action spaces, while Actor-Critic Methods combine value-based and policy-based approaches, consisting of an actor that updates the policy and a critic that evaluates the value function. Deep Reinforcement Learning extends traditional RL algorithms using deep learning techniques, with methods such as Deep Q-Networks (DQN) combining Q-learning with deep neural networks to handle high-dimensional state spaces, and Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO) providing advanced policy-based algorithms that stabilize training using constraints on policy updates.

Q-learning, a model-free algorithm, was chosen for several reasons. Its simplicity and ease of implementation make it relatively straightforward to understand and apply, as it does not require a model of the environment, making it suitable for complex or unknown dynamics. Being an off-policy algorithm, Q-learning learns the optimal policy independently of the agent's current behavior, allowing the use of exploratory policies during training while still converging to the optimal policy. This proven convergence to the optimal policy, given sufficient exploration and a suitable learning rate, provides theoretical assurances of optimality. Additionally, Q-learning is scalable, capable of handling large state-action spaces through function approximation techniques like deep Q-networks (DQN), and is adaptable to various problem domains. It is particularly effective for discrete action spaces, making it well-suited for tasks such as navigation and control in robotics where actions can be discretized. Furthermore, its robustness to suboptimal exploration strategies allows Q-learning to learn optimal behavior even with noisy or exploratory actions. These advantages make Q-learning

a practical and effective choice for training agents in complex, dynamic environments, making it a suitable algorithm for reinforcement learning tasks, including robotics and path planning.

Q-learning

Q-learning is a reinforcement learning algorithm, focusing on making decisions in an environment to maximize cumulative rewards. The algorithm uses Q-table to store Q-values which represent the expected cumulative rewards of taking a particular action in a specific state. Q-learning involves an agent interacting with an environment, receiving rewards for its actions, and updating its Q-table based on these experiences. These values guide the agent in making decisions by helping it choose the most promising actions to maximize its total reward over time. The Q-table is updated using the Bellman equation, that reflects the relationship between current and future rewards. Through exploration and exploitation the agent refines its Q-table, converging toward an optimal policy for decision making in the given environment.

The aspects of Q-learning include:

- State Representation: The e-puck robot gathers information about its surroundings through proximity sensors. The sensors values are stored as a tuple representing distances to nearby objects. For instance, the robot has eight proximity sensors, the state might be represented as (ps0, ps1, ps2, ps4, ps5, ps6, ps7), where each value 'ps' represents the distance measured by a sensor.
- Action Space: Actions available to the e-puck robot includes rotating left and right in different angles.
- Q-table: The Q-table is a data structure where the e-puck robot stores Q-values associated with state action pairs. For every possible combination of sensor reading and available actions, the Q-table contains a Q-value indicating the expected cumulative reward if that action is executed in that state. Initially, the Q-table is initialized with zeroes. As the robot interacts with the environment, the Q-values get updated.
- Exploration and Exploitation: The robot employs an exploration exploitation strategy. Initially, it might explore different actions randomly to gather information about the environment and update its Q-table. Over time, it starts exploiting the learned Q-values to make better decisions.
- Action Set:The action state contains six actions from which the robot can choose one whilst avoiding obstacles. The set contains 3 different angles to rotate left and 3 different angles to rotate right.
  - Rotate left 35 degree
  - Rotate right 35 degree
  - Rotate left 55 degree
  - Rotate right 55 degree
  - Rotate left 100 degree

– Rotate right 100 degree

Bellman's equation is used to update the Q-value on Q-table. Bellman's equation is a fundamental concept in reinforcement learning, expressing how the value of a state is related to the immediate reward and the expected value of the next state. It mathematically defines how an agent's action choice affects its long term cumulative reward. Rewards signify the immediate benefit or penalty an agent receives upon taking an action in a specific state. Cumulative rewards, on the other hand, represent the total sum of rewards obtained over a sequence of actions, guiding the agent to make decisions that maximize its overall long term gain in an environment. Bellman's equation acts as a framework, enabling agents to estimate the value of actions by considering both immediate rewards and future expected rewards for optimal decision making. Mathematically, it defines how an agent's action choice influences its long term cumulative reward, guiding the agent to make decisions that maximize its overall gain in the environment.

Rewards represent the immediate benefit or penalty an agent receives upon taking an action in a specific state. Cumulative rewards, on the other hand, denote the total sum of rewards obtained over a sequence of actions, guiding the agent's decision-making to optimize its long term performance.

Bellman's equation is expressed as:

$$Q(s,a) = (1-\alpha) \cdot Q(s,a) + \alpha \cdot [r + \gamma \cdot \max_a Q(s',a')] \tag{3.1}$$

Where:

- Q(s,a) is the Q-value for the state-action pair (s,a).
- α is the learning rate, determining the extent to which new information overrides old information.
- r is the observed reward obtained after taking action a in state s.
- γ is the discount factor, representing the importance of future rewards relative to immediate rewards.
- $\max_a Q(s',a')$ denotes the maximum Q-value for the next state s' over all possible actions a'.

By iteratively updating Q-values using Bellman's equation, agents can learn optimal policies for navigating their environment and achieving their objectives effectively. This framework enables agents to balance exploration and exploitation, gradually improving their decision making capabilities and maximizing their cumulative rewards over time. In robotics, Q-learning is extensively used for various tasks, including navigation, manipulation, and interaction with the environment. Its ability to learn optimal actions through trial and error without requiring a model of the environment makes it particularly suited for robotic applications where modeling the environment accurately can be challenging. Robots can use Q-learning to adapt to dynamic environments. For instance, in a warehouse, a robot can learn to navigate efficiently even as the layout of obstacles changes over time. Q-learning

allows robots to learn and automate complex tasks, such as picking and placing objects, by learning from rewards associated with successful task completion.

3.6 Discretization

The e-puck robot is equipped with proximity sensors that continuously measure the distance to nearby objects. However, Q-learning typically operates on discrete state action spaces, which presents a challenge when dealing with continuous sensor readings. Continuous sensor readings make it difficult to define a discrete state space directly applicable to Q-learning. Without discretization, the Q-learning algorithm would struggle to generalize effectively across the continuous sensor values, leading to suboptimal decision making and potentially poor performance in obstacle avoidance tasks. To overcome this challenge, a discretization approach was implemented. Discretization involves partitioning the continuous sensor readings into a finite number of discrete states, each representing a range of sensor values. This transformation enables the application of Q-learning by converting the continuous sensor space into a manageable discrete representation.

Implementation:

- Sensor Value Ranges: Initially, the range of sensor values is divided into intervals or bins. The number and size of these bins can be adjusted based on the specific characteristics of the sensor readings and the requirements of the task.
- State Definition: Each bin corresponds to a discrete state, representing a particular range of sensor values. The state space is defined by the combination of these discrete states across all proximity sensors.
- State Encoding: During operation, the current sensor readings are mapped to the appropriate discrete states based on their values falling within the predefined intervals. This mapping effectively discretizes the continuous sensor inputs into discrete states that can be used by the Q-learning algorithm.
- Q-Value Updates: The Q-learning algorithm operates on the discretized state space, updating Q-values for state-action pairs accordingly. By learning from discretized states, the robot can effectively generalize its actions across similar sensor readings, leading to more robust decision making in obstacle avoidance tasks.

Table 3.1: Discretization of proximity values.

Proximity Values	Discretized Values
0-25	25
26-50	50
51-75	75
76+	100

The table 3.1 shows the discretized values. The Q-table consists of all the possible permutations of proximity values which are discretized. There are eight values and there will be (4x4x4x4x4x4x4x4) possible states and corresponding action values. The action values for each state will be a list of 6 values initialized to zeroes and after interaction of the agent with the environment the values get updated. The 6 values represent the action to rotate left and right in various angles. The implementation of discretization addresses the challenge posed by the continuous nature of proximity sensor readings in the context of Q-learning with the e-puck robot in the Webots environment. By transforming continuous sensor values into discrete states, discretization enables effective decision making and improved performance in obstacle avoidance tasks, facilitating the application of reinforcement learning techniques in robotic systems.

Chapter 4

Experimental Results and Analysis

4.1 Simulation Environment

Webots software is used to create the simulation environment to work on. A 10mx10m floor is created in the simulated world where different obstacles are added and an agent is set to interact with the environment. The E-puck robot is available in the simulation environment which was used as the agent for interaction. In the first phase of the work, a single E-puck robot is controlled in the simulated environment avoiding obstacles and Q-learning algorithm is implemented. In the subsequent phase, two agents were introduced, tasked with simultaneously progressing towards a destination while ensuring consensus and avoiding obstacles. Obstacle avoidance was implemented by utilizing the Q-values derived from the Q-table established in the initial phase.

Figure 4.1: Simulated environment with robots and obstacles

The figure 4.1 shows a sample simulation environment with a multiple agents and obsta-

cles. Three blocks are placed on the ground level of the simulated floor where the blocks act as obstacles. The proximity sensors attached to the robot reads the proximity values while encountering with the rocks and performs the exploration exploitation strategy whilst iterating Q-table.

4.2 Hardware and Environmental Setup

The experiments were conducted on an MSI GF63 Thin 11SC-853IN laptop, which features the following specifications:

- Processor: 11th Gen Intel Core i5 processor
- Memory: 8 GB of DDR4 memory
- Storage: 512 GB NVMe PCIe M.2 SSD
- Graphics Card: NVIDIA GeForce GTX 1650 Max-Q graphics card with 4 GB VRAM

This laptop provides sufficient computational power and memory capacity to perform the experiments effectively. The 11th Gen Intel Core i5 processor offers reliable performance for various computational tasks, while the NVIDIA GeForce GTX 1650 Max-Q graphics card accelerates graphics processing, benefiting visualization tasks or simulations. The laptop's 8 GB of DDR4 memory and 512 GB NVMe PCIe M.2 SSD ensure smooth operation and provide ample storage space for datasets and experiment files.

The experiments were conducted within a Python environment set up using Anaconda, a popular platform for managing Python packages and environments. Anaconda simplifies the process of setting up reproducible research environments and was used to establish a Python environment with Python version 3.7. Additionally, Anaconda facilitated the installation of other necessary dependencies required for conducting the experiments, including machine learning libraries, simulation tools, and data analysis packages. The combination of the specified hardware components and software environment on the MSI GF63 Thin 11SC-853IN laptop provided a suitable platform for conducting experiments involving computational tasks, machine learning, and data analysis. These resources enabled efficient execution of experiments, analysis of results, and iteration on methodologies to advance research objectives.

4.3 Transfer learning

Transfer learning in this context involves leveraging the knowledge gained from training a single robot to enable multiple robots to perform similar tasks more effectively and efficiently. This approach enhances the scalability and adaptability of robotic systems in navigating complex environments. Transfer learning is evident in the process of applying knowledge gained from training one robot to avoid obstacles to the task of enabling multiple robots to do the same.

- Initial Learning Phase: Initially, a single robot undergoes a learning phase where it learns to avoid obstacles using reinforcement learning techniques. During this phase, the robot is trained to associate proximity values with actions that lead to successful navigation without collisions.
- Knowledge Representation: The learned knowledge is then encoded in the form of Q-values, which represent the expected rewards for taking certain actions in specific states (proximity values in this case). These Q-values are stored in a separate .txt file.
- Transfer to Multiple Robots: Transfer learning occurs when the knowledge gained from training the single robot is applied to multiple robots. Instead of training each robot from scratch, the pre-existing knowledge in the form of Q-values is utilized to bootstrap the learning process for the additional robots.
- Consensus Design: In the consensus design, when multiple robots encounter obstacles, they refer to the stored Q-values to inform their actions. This enables them to make decisions based on the learned knowledge rather than starting the learning process from scratch.
- Adaptation and Fine Tuning: While the initial learning phase provides a foundation, the transferred knowledge may need to be adapted or fine tuned to suit the specific dynamics or environments encountered by each robot. However, the bulk of the learning effort is saved due to the transfer of knowledge from the single robot.
- Efficiency and Scalability: By leveraging transfer learning, the overall learning process becomes more efficient and scalable. Instead of individually training each robot extensively, knowledge is shared among them, leading to quicker deployment and adaptation in real world scenarios.

4.4 Navigation Algorithm

The algorithm outlines the steps for coordinating the movement of multiple robots towards a common destination in a simulated environment. It begins by initializing the necessary components such as robot instances, sensor devices (GPS, compass, proximity sensors), and a Q-table with pre-trained values. Timestep parameters are also set to ensure synchronization among the robots.

During each iteration, the algorithm iterates through each robot to gather sensor data, including its position relative to the destination and proximity to other robots. If a rotation towards the destination is required based on the current heading, the robot rotates accordingly. In case an

obstacle is detected, the robot selects an action from the Q-table based on the sensor data. If a Q-value exists for the current state, the action with the highest Q-value is chosen; otherwise, predefined actions or default behavior are employed to avoid the obstacle. If no obstacles are detected and the robot is not too close to other robots, it moves towards the destination with a predefined speed. Timestep completion is checked to maintain synchronization among the robots.
Algorithm

Algorithm 1 Q-table generation

```
Initialize Q-table Q with random values or zeros
Initialize state representation function get_state()
Initialize exploration rate ϵ
Initialize action space NUM_ACTIONS
Initialize constants: TIMESTEP, MAX_SPEED, OBSTACLE_THRESHOLD, STUCK_DURATION, RESET_POSITION
Initialize actions history actions_history
Initialize previous state prev_state
while robot is active do
    Read sensor values ps_values
    Get current state state ← get_state(ps_values)
    if prev_state is not null then
        Choose action action based on ϵ-greedy policy and Q
        Execute action on robot
        if obstacle detected then
            Update exploration rate ϵ
            Handle obstacle avoidance actions
        else
            Move robot forward with maximum speed
        end if
        Update actions history
    end if
    Update previous state prev_state ← state
    if robot is stuck then
        Reset robot's position
    end if
    if time to update Q-table then
        Select top states top_states based on actions_history
    end if
end while
```

Algorithm 2 Robot navigation control

```
Initialize the robot, sensors, motors, and other necessary components.
Load the Q-table from a file.
while the robot is active do
    Step the simulation.
    Get positions and orientations of robots.
    Check distance between robots.
    if distance between robots is less than a threshold then
        Calculate cutoff angle to target.
        if cutoff angle is greater than a threshold then
            Rotate the robot to face the destination.
        else
            Do nothing.
        end if
        Check for obstacles.
        Move the robot to the destination with obstacle avoidance.
    else
        Calculate cutoff angle to other robot.
        if cutoff angle is greater than a threshold then
            Rotate the robot to face the other robot.
        else
            Do nothing.
        end if
        Check for obstacles.
        Move the robot to the other robot with obstacle avoidance.
    end if
    Check proximity sensor values.
    if proximity values meet a threshold then
        Choose an action from the Q-table based on the current state.
        Execute the chosen action.
    end if
end while
```

4.5 Results

The movements of the E-puck robot placed on the environment floor is controlled using Q-learning algorithm. Three obstacles are introduced in the simulated floor for the agent to interact and identify it as obstructions. While interacting, the Q-values for each movement are iterated and stored using Bellman's equation and these values are stored in a .txt file within the same folder as the python controller. This table of Q-vlaues can be later used when the simulation is run again and there will not be a loss of progress. This Q-value table helps to choose the right action in a provided state which is the tuple of values of all the proximity sensor values.

Table 4.1: Initial Q-values.

States	**TR.35**	**TL.35**	**....**	**TL.100**
(25,25,25,25,25,25,25,25)	0	0	0	0
(50,25,25,25,25,25,25,25)	0	0	0	0
.....	0	0	0	0
.....	0	0	0	0
.....	0	0	0	0
(100,100,100,100,100,100,100,75)	0	0	0	0
(100,100,100,100,100,100,100,100)	0	0	0	0

The table 4.1 shows the initial Q-table with all the action values initialized to zeroes. After the interaction of robot with the environment and obstacles the Q-value get updated using Bellman's equation.

```
State: (75, 75, 75, 75, 75, 75, 75, 75)
Actions: [-0.5114435934108611, 0.027249321084372988, -0.11027326763465234, 0.08134456583351032, 2.75536449973883, 0.21909031216626]
State: (75, 75, 75, 75, 75, 75, 75, 100)
Actions: [-0.8537695682953449, -0.605076697448643, -0.5423189053837938, -0.5094468077868086, 2.805317969171606, -0.2483068487535232]
State: (75, 75, 75, 75, 75, 100, 75, 75)
Actions: [2.129901589555487, -0.6593096190568111, -0.17216694316060732, -0.606271308645471, 0.10988269210942886, -0.5922315427038278]
State: (75, 75, 75, 75, 75, 75, 100, 75)
Actions: [0.5450793604204381, -0.9698511021229811, 0.8641189156846837, -1.3526366876154923, 3.0584455807563433, -0.37777763735220726]
State: (75, 75, 100, 75, 75, 75, 75, 75)
Actions: [-0.49247176065283604, -0.13042225546312236, -0.9803260144760838, -0.21175266308127516, -0.7777232864230239, 2.161410885134065]
State: (75, 75, 75, 100, 75, 75, 75, 75)
Actions: [-0.7516583351433281, -0.7788461689991023, -0.8968734378762206, 0.3383499127865814, -0.8462274855051024, -0.7308066315429783]
State: (100, 75, 75, 75, 75, 75, 75, 75)
Actions: [-0.7436536824983715, -1.2627926897860309, -0.9977093022937935, -0.8516871301118702, -0.36175641593452096, 3.3036932382622965]
State: (100, 75, 75, 75, 75, 75, 100, 100)
Actions: [-1.8293470997860415, -1.863016049565744, -1.8875193704432174, -1.8487900301483493, -1.889763498141516, -1.470644639324159]
State: (100, 100, 75, 75, 75, 75, 75, 100)
Actions: [-1.7604900010550582, -1.5239358903742668, -1.7727599889943564, -1.797366017645416, -1.9095955232928201, -1.779124640587158]
State: (75, 75, 75, 75, 75, 75, 100, 100)
Actions: [-0.8743844340726736, -1.768566260431399, -0.686703649735136, -0.8553846533606564, 2.288563011773962, -1.1233520590700952]
State: (75, 75, 100, 100, 75, 75, 75, 75)
Actions: [-1.0669616707358347, -0.6220811294740815, -1.340013084245653, -1.2690065052213768, -1.2805474142788658, -1.3219369097975349]
State: (75, 75, 75, 75, 75, 100, 100, 75)
Actions: [1.840326191804122, -1.2172773609048688, -0.9781517120580471, -1.3040769960498788, -0.9718893142316455, -1.0961672914197906]
State: (100, 75, 75, 75, 75, 100, 75, 75)
Actions: [0.48211084132255405, -0.20696811262781056, -0.1, -0.2157655959372785, -0.30639667451123975, -0.25258084869793374]
State: (75, 75, 75, 75, 100, 75, 75, 75)
Actions: [-0.5183769012847307, -0.221972067377265 77, -0.5439023346069308, -0.9481197203623284, -0.40892387642430295, -0.5636418176431071]
State: (75, 75, 75, 75, 100, 75, 100, 75)
Actions: [-0.09180470249652685, 0, 1.0192499952369403, 0.0067936889928075506, 0.016900487703159786, 0]
State: (100, 100, 75, 75, 75, 75, 75, 75)
Actions: [-1.3746414676986205, -1.315128183700867, -0.8287697318316253, -0.10069474382327137, -0.6838419406165465, 2.105406954685673]
```

Figure 4.2: Q-table instance

The fig 4.2 shows the instance of the Q-table where the Q-values for every state is updated when the agent continuously interacts with the environment and rewards are calculated. The obtained Q-table is integrated into the controller. When the proximity sensor detects values exceeding the obstacle threshold, the robot selects values from the Q-table and executes various movements to navigate around obstacles. While engaged in obstacle avoidance, multiple agents collaborate to achieve consensus; they converge towards each other if their distance exceeds 0.2m, otherwise they proceed towards the provided destination. Changes in rotations and heading are monitored and adjusted using compass and GPS readings.

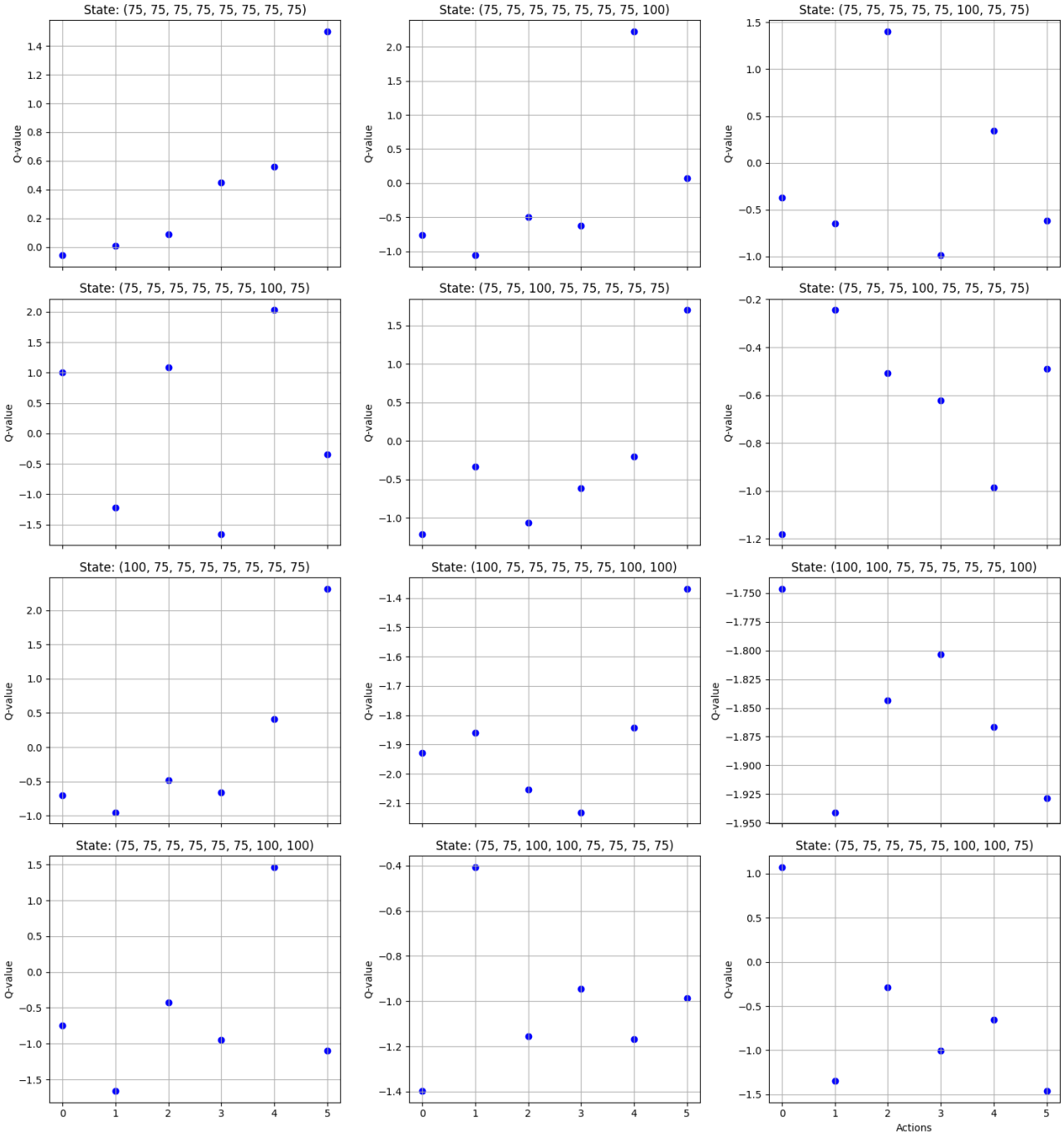

Figure 4.3: Convergence of action values

Figure 4.3 illustrates the action value convergence across epochs throughout the training process. A set of 12 states is randomly chosen, and the plot is stored once the epoch reaches 1.

4.5.1 State Transition Function

The state transition function in a reinforcement learning environment describes how an agent moves from one state to another based on the actions it takes. In the context described, the agent's state S is represented by sensor readings, denoted as ps0 to ps7. These sensor readings provide information about the agent's surroundings, such as proximity to obstacles, landmarks, or other agents. When the agent selects an action A from the action space, which includes options like moving forward, moving backward, rotating left, or rotating right, the environment transitions the agent to a new state S based on both the chosen action and the resulting sensor readings.

For example, if the agent selects the action to move forward, the environment updates the agent's position based on its current orientation and any obstacles or boundaries in the environment. Subsequently, the agent's sensors gather new readings reflecting its updated surroundings, resulting in a new state S'. Similarly, if the agent chooses to rotate left or right, the environment adjusts the agent's orientation accordingly, leading to a change in sensor readings and consequently a new state S. The transition from state S to state S' is influenced by both the agent's actions and the dynamics of the environment, including factors such as obstacles, terrain, or other agents' actions. This process of state transition forms the basis of the agent's interaction with its environment in a reinforcement learning setting, enabling it to learn and adapt its behavior over time to achieve its goals more effectively.

- State: S(ps0,ps1,ps2,ps3,ps4,ps5,ps6,ps7)
- A=move forward, move backward, move left, move right
- Transition Function: S, A$\rightarrow$T(S, A)$\rightarrow$S'

4.5.2 Reward Function

The reward function in a reinforcement learning environment provides immediate feedback to the agent after it takes an action in a particular state of the environment. This function quantifies the desirability of a state-action pair by assigning a numerical value, known as the reward, to that pair.

The reward function operates as follows:

- If the obstacle threshold exceeds, indicating that the agent is in a state where obstacles are too close for safe navigation, actions derived from the Q-table are initiated. This suggests that the agent needs to rely on its learned policy to navigate around obstacles.
- Upon executing an action, if the obstacle threshold persists, indicating that the agent's action did not effectively mitigate the obstacle or improve its situation, a negative reward of -1 is given. This negative reward penalizes the agent for taking actions that fail to resolve the obstacle or improve its state.
- Conversely, if the obstacle threshold does not persist after executing the action, indicating that the agent successfully navigated around the obstacle or improved its situation, a positive reward of +1 is assigned. This positive reward reinforces the agent's behavior, encouraging it to take actions that lead to desirable outcomes, such as avoiding obstacles and progressing towards its goal.

By assigning rewards based on the agent's actions and their outcomes, the reward function guides the agent's learning process, helping it to learn optimal behaviors that maximize cumulative rewards over time. This feedback mechanism is crucial for shaping the agent's behavior and guiding it towards achieving its objectives effectively within the environment.

4.5.3 Robot Control

The robot iterates through time steps, following a process that involves reading sensor values, selecting actions based on Q-values, and updating the Q-table accordingly. With its differential wheel control mechanism, the robot has the capability to perform various movements, including driving forward, and rotating.

The differential wheel control works as follows:

- Driving Forward: When both wheels are provided with the same velocity in the forward direction, the robot moves forward. This ensures that both wheels rotate in the same direction at the same speed, propelling the robot forward.
- Rotating Left or Right: To rotate left or right, the robot adjusts the velocities of its wheels in opposite directions. Specifically, to rotate left, the velocity of the left wheel is decreased or reversed while the velocity of the right wheel is increased. Conversely, to rotate right, the velocity of the right wheel is decreased or reversed while the velocity of the left wheel is increased. This differential in wheel speeds causes the robot to rotate about its center axis.

The maximum speed of the robot's wheels is 6.28 meters per second (m/s). This maximum speed limit imposes a constraint on the robot's velocity, ensuring that its movements remain within a predefined range and promoting safe and controlled navigation within its environment.

By leveraging its differential wheel control mechanism and adhering to the maximum speed limit, the robot can execute various maneuvers effectively, enabling it to navigate through its environment and perform tasks autonomously based on the actions derived from its Q-learning algorithm.

4.5.4 Robot Heading

The robot's heading is determined by analyzing data from the compass sensor, which provides information about the robot's orientation relative to magnetic North. This orientation is essential for navigation, as it guides the robot in determining its direction of movement within its environment. The compass sensor furnishes a vector indicating the direction of the magnetic field surrounding the robot. To derive the robot's heading angle from this sensor data, the math.atan2 function is commonly utilized. This function calculates the arctangent of the ratio of two specified numbers, yielding the angle in radians.

After obtaining the angle in radians from the math.atan2 function, the data is then converted into degrees using appropriate mathematical operations. This transformation ensures that the robot's heading angle is expressed in degrees, making it more intuitive and easier to interpret. By utilizing data from the compass sensor and performing the necessary mathematical operations, the robot can accurately determine its heading angle, which serves as a crucial piece of information for navigation and guiding its movements within its environment.

Algorithm 3 Movement Function with inclusion of Q-learning

```
1: procedure MoveToDestination(robot, left_motor, right_motor, compass, gps, destination,
   robot_node1, robot_node2, yaw1, yaw2, threshold, proximity sensorss)
2:     Initialize movement_threshold, distance_threshold, and max_speed
3:     while True do
4:         Update yaw1 and yaw2 (headings of the robots)
5:         Update positions pos1 and pos2
6:         Compute distance_between_robots
7:         Compute distance_to_target_x, distance_to_target_y, and reach_condition
8:         Get ps_values and check for obstacle
9:         if not angles_are_equal(yaw1, yaw2, threshold) then
10:            break
11:        end if
12:        if reach_condition then
13:            Stop motors and print('Destination Reached. Finished.')
14:            break
15:        end if
16:        if obstacle then
17:            action ← choose_action(ps_values, q_table)
18:            if action indicates turning left or right then
19:                Execute turning action to avoid obstacle
20:            end if
21:        else
22:            Move forward at max_speed
23:        end if
24:        robot.step()
25:    end while
26: end procedure
```

Algorithm 4 Algorithm for rotating towards a destination coordinate

```
1: procedure RotateToDestination(robot, left_motor, right_motor, compass, gps, destination, robot_node1, robot_node2)
2:     rotate_right ← True if calculate_degree_to_target(robot, destination) ≥ 180 else False
3:     max_speed ← 6.0
4:     angle_threshold ← 4
5:     initial_degree ← round(get_robot_heading(compass.getValues()))
6:     while True do
7:         current_degree ← get_robot_heading(compass.getValues())
8:         diff_current_vs_initial_degree ← |current_degree − initial_degree|
9:         if rotate_right then
10:            set_motor_speed(left_motor, max_speed)
11:            set_motor_speed(right_motor, -max_speed)
12:        else
13:            set_motor_speed(left_motor, -max_speed)
14:            set_motor_speed(right_motor, max_speed)
15:        end if
16:        robot.step()
17:        if diff_current_vs_initial_degree < angle_threshold then
18:            stop_motors(left_motor, right_motor)
19:            break
20:        end if
21:    end while
22: end procedure
```

4.5.5 Q-learning Approach

- State Representation:
  - The state is represented based on the readings of proximity sensors.
  - A function discretizes sensor readings into four levels: 25, 50, 75, and 100.
  - The state is formed as a tuple, where each element represents the discretized reading of a proximity sensor.
  - The threshold for proximity sensors is set as 100. If the proximity value exceeds 100, the agent considers it a collision, where 0-99 is considered safe.
- Q-table Initialization:
  - The Q-table is represented as a dictionary, where keys are state tuples and values are lists representing action values.
  - The Q-table is initialized with all possible states and action values as lists of zeroes.
- Action Selection:
  - The agent determines the action to take based on the current state.
  - It employs the epsilon-greedy strategy for action selection, balancing exploration and exploitation.
  - With a certain probability (determined by epsilon), the agent selects a random action (exploration). Otherwise, it selects the action with the highest Q-value for that state (exploitation).
- Q-value Update:
  - After taking an action and observing the new state and reward, the Q-value for the previous state-action pair is updated.
  - The controller uses Bellman's equation to adjust the Q-value, incorporating the observed reward and the maximum expected future reward.
  - The new Q-value is calculated using the current Q-value, learning rate (alpha), reward, discount rate (gamma), and the maximum Q-value for the next state.

Figure 4.4: Robots without obstacles working for consensus

Figure 4.5: Robots with obstacle avoidance working for consensus

Figure 4.4 depicts agents initially positioned at two distant locations, converging towards each other and subsequently moving towards a predefined destination in an obstacle free environment, all while ensuring a safe distance between them. In contrast, Figure 4.5 illustrates robots maneuvering around obstacles as they advance towards the same destination, particularly when they are within a specified threshold distance.

Chapter 5

Challenges and Limitations

Implementing a multi-robot system for coordinated navigation presents several challenges, particularly when transitioning from simulation to real-world environments. One significant challenge is the inherent limitations of simulation environments. Simulations provide a controlled and repeatable setting for developing and testing algorithms, but they cannot fully replicate the complexities and unpredictability of real-world conditions. For instance, in a simulated environment like Webots, the e-puck robots operate on an idealized, even ground. This assumption simplifies the problem and allows the robots to focus on navigation and coordination without dealing with physical irregularities. However, in real-life scenarios, the robots may encounter uneven terrain, slopes, and various types of surfaces that can significantly impact their movement and stability. These physical factors can introduce noise and variability into the system, making it difficult to predict how the robots will behave when deployed outside the simulation.

Another major challenge is scalability. While the project currently involves only two robots, scaling up to a larger number of robots introduces additional complexity. As the number of robots increases, the system must manage more interactions and communications, which can lead to issues such as network congestion and increased computational load. Effective communication is crucial for maintaining coordination among the robots, especially in dynamic environments where real-time updates are necessary. In the simulation, communication is often simplified and assumes perfect transmission without delays or losses. In contrast, real-world communication systems are subject to interference, signal attenuation, and bandwidth limitations. Ensuring reliable and efficient communication among a larger group of robots requires robust network protocols and possibly redundant communication channels to mitigate potential failures. The physical environment also poses challenges related to the robots' ability to maintain coordination and achieve their goals. In the simulation, the robots are programmed to move towards each other when they are separated beyond a certain threshold and to proceed towards the common destination once they are together. This behavior assumes that the robots can accurately sense each other's positions and navigate without obstacles. In real-world applications, obstacles such as walls, furniture, or other objects can obstruct the robots' paths and interfere with their sensors. Additionally, factors like wheel slippage, battery life, and mechanical wear can affect the robots' performance. These issues necessitate more sophisticated sensing, planning, and control algorithms to ensure that the robots can adapt to changing conditions and still perform their tasks effectively.

To address these challenges, several implementation strategies can be employed. First, integrating more advanced sensor systems can help the robots better perceive and adapt to their

environment. This might include using LIDAR, cameras, or ultrasonic sensors to detect obstacles and terrain variations. Additionally, implementing robust path-planning algorithms that can account for dynamic obstacles and varying terrain can improve the robots' ability to navigate complex environments. Machine learning techniques, such as reinforcement learning, can be used to train the robots to handle a wider range of scenarios by exposing them to diverse and challenging conditions during the training phase. Scalability and communication can be managed through distributed algorithms and robust communication protocols. Distributed algorithms can allow robots to make local decisions based on their immediate surroundings and limited information, which can then be aggregated to achieve global coordination. Implementing robust communication protocols, such as those based on mesh networking, can enhance reliability by allowing robots to relay messages through multiple paths, ensuring that communication is maintained even if some links fail. Moreover, fault-tolerant algorithms can be designed to handle communication breakdowns, ensuring that the robots can still operate effectively even in the presence of network issues.

Chapter 6

Real World Applications

There are potential real world applications of E-puck robots equipped with algorithms for avoiding obstacles and moving towards common destinations or each other. These applications span across various fields, including search and rescue, warehouse management, autonomous delivery, agriculture, environmental monitoring, healthcare, military, and education.

- Search and Rescue Operations: In disaster stricken areas such as those affected by earthquakes, floods, or building collapses, it is crucial to locate and rescue survivors quickly. Search and rescue teams often face challenges due to the hazardous and unpredictable nature of these environments. E-puck robots can be deployed in such scenarios to assist human rescuers. Equipped with sensors for detecting humans and navigating debris, these robots can move through rubble and tight spaces that may be dangerous or inaccessible to humans. The robots can navigate through complex environments, avoiding debris and other obstacles using their obstacle avoidance algorithms. Multiple robots can work together to cover larger areas. If a robot detects a survivor, it can signal other robots to converge on that location to provide assistance or relay information to human rescuers. This application reduces the risk to human rescuers, increases the speed of locating survivors with coordinated search patterns, and allows for scalability by deploying multiple robots to cover extensive areas more effectively.

- Warehouse and Inventory Management: Modern warehouses are vast, with complex layouts requiring efficient systems to manage and transport goods. Automation in warehouses can significantly enhance operational efficiency and reduce human labor. E-puck robots can be used in warehouses for transporting items between different sections. They can navigate around shelves, other robots, and obstacles, moving goods from storage to packaging or loading areas. The robots can use their algorithms to find optimal paths to their destinations, avoiding collisions and traffic jams. When distant from each other, robots can move towards each other to transfer items or collaborate on larger tasks. This application streamlines the movement of goods, reduces processing times, lowers labor costs by automating repetitive tasks, and minimizes errors in item handling and placement.

- Agricultural Automation: Agriculture is increasingly relying on automation to enhance productivity and reduce labor costs. Robots can perform tasks such as planting, monitoring, and harvesting crops. E-puck robots can be used in agricultural settings to monitor crops, apply pesticides, or harvest produce. They can navigate through fields, avoiding plants and obstacles. Equipped with sensors, robots can collect data on soil conditions, plant health, and

growth rates. Robots can work together on tasks like harvesting, moving towards each other to transport produce to collection points. This application increases efficiency and speed of agricultural tasks, provides accurate data collection and targeted application of resources, and minimizes the need for manual labor.

- Environmental Monitoring and Data Collection: Continuous monitoring of environmental parameters is crucial for research and conservation efforts. This includes monitoring wildlife, air and water quality, and vegetation health. E-puck robots can be deployed in various environments to collect data. They can navigate through forests, wetlands, or urban areas, avoiding obstacles and moving towards specific monitoring stations or each other to relay data. Equipped with sensors, robots can collect and transmit environmental data. Algorithms allow robots to move efficiently through diverse terrains, avoiding obstacles. This application provides high-quality data for research and conservation, enables monitoring of large and remote areas, and reduces the need for human presence in potentially hazardous environments.

- Healthcare Assistance: Hospitals and care facilities require efficient systems to assist with the transportation of medical supplies and equipment. Automation can help streamline these processes. E-puck robots can be used to transport medical supplies within healthcare facilities. They can navigate around furniture, people, and other obstacles, coordinating with each other to ensure timely delivery. Robots can find optimal paths to deliver supplies quickly and safely. Multiple robots can work together to distribute supplies to different parts of the facility. This application enhances the speed and reliability of supply distribution, reduces the risk of contamination and handling errors, and allows healthcare staff to focus on patient care.

- Military and Defense: In military operations, robots can be used for reconnaissance, surveillance, and support tasks, reducing the risk to human soldiers. E-puck robots can be used for scouting missions, navigating through difficult terrains while avoiding obstacles. They can work in teams, moving towards each other to share data and coordinate actions. Robots can gather intelligence without putting soldiers at risk. Teams of robots can collaborate on surveillance and support missions. This application minimizes the risk to human soldiers in dangerous missions, enhances the effectiveness of reconnaissance and support operations, and can be deployed in various terrains and mission scenarios.

Chapter 7

Conclusion and Future Scope

Reinforcement learning, particularly Q-learning, is a robust methodology for enabling the E-puck robot to learn optimal strategies in the Webots environment. Through the implementation of Q-learning, the E-puck robot exhibit adaptability, gradually improving its decision making by exploring and exploiting the environment to maximize cumulative rewards. The successful storage of Q-values in a .txt file serves as a result of implementing reinforcement learning algorithms which can be extended in real-world robotic applications. The project's outcomes implies the significance of reinforcement learning in enabling autonomous learning in robotics. The E-puck robot, through its interactions and learning process within the Webots environment, demonstrated the acquisition of optimal policies and the potential of reinforcement learning techniques in enhancing the adaptability and decision making capabilities of autonomous agents.

Future research could delve into exploring advanced variations of Q-learning or other reinforcement learning algorithms to enhance the learning efficiency of the E-puck robot. Implementing methods such as deep Q-networks (DQN) or actor-critic methods presents promising avenues for improving learning rates and convergence speeds. By leveraging deep neural networks, DQN extends Q-learning to handle high-dimensional state spaces more effectively, potentially leading to more efficient and robust learning processes. Similarly, actor-critic methods offer a framework for simultaneously learning both value-based and policy-based aspects of reinforcement learning, offering potential advantages in terms of sample efficiency and stability.

Furthermore, incorporating environmental factors and external influences into the analysis can enhance the realism and applicability of the research findings. Considering environmental dynamics, such as obstacles, terrain variations, or changing conditions, can pose additional challenges for the agents and require adaptive strategies for successful navigation and task completion. By addressing these complexities, researchers can develop more robust and adaptable robotic systems capable of operating in diverse real-world environments. Future research directions for enhancing the E-puck robot's learning capabilities could involve exploring advanced reinforcement learning algorithms, extending analyses to multi-agent scenarios, and considering environmental dynamics to develop more capable and adaptive robotic systems. These efforts have the potential to advance the field of robotics and contribute to the development of intelligent and autonomous robotic agents for various applications.

Exploring the feasibility of transferring learned knowledge from simulations to real-world scenarios could be a significant advancement. Investigating methods to bridge the reality gap between

simulation and physical environments is vital for practical robotic applications. As autonomous agents like the E-puck robot gain more capabilities, ethical considerations surrounding their decision making processes become crucial. Future research should try ethical frameworks for reinforcement learning based robots to ensure responsible and safe autonomous behaviors in various settings. These future prospects not only hold promise in advancing the field of robotics but also underscore the potential impacts of autonomous agents in diverse domains.

References

[1] Wang, H. and Li, M., 2020. Model-free reinforcement learning for fully cooperative consensus problem of nonlinear multiagent systems. IEEE Transactions on Neural Networks and Learning Systems, 33(4), pp.1482-1491.

[2] Mu, C., Zhao, Q., Gao, Z. and Sun, C., 2019. Q-learning solution for optimal consensus control of discrete-time multiagent systems using reinforcement learning. Journal of the Franklin Institute, 356(13), pp.6946-6967.

[3] Duguleana, M. and Mogan, G., 2016. Neural networks based reinforcement learning for mobile robots obstacle avoidance. Expert Systems with Applications, 62, pp.104-115.

[4] Yang, X., Zhang, H. and Wang, Z., 2021. Data-based optimal consensus control for multiagent systems with policy gradient reinforcement learning. IEEE Transactions on Neural Networks and Learning Systems, 33(8), pp.3872-3883.

[5] Gao, Y., Wang, W. and Yu, N., 2021. Consensus multi-agent reinforcement learning for volt-var control in power distribution networks. IEEE Transactions on Smart Grid, 12(4), pp.3594-3604.

[6] Li, Y. and Tan, C., 2019. A survey of the consensus for multi-agent systems. Systems Science Control Engineering, 7(1), pp.468-482.

[7] Olfati-Saber, R. and Murray, R.M., 2004. Consensus problems in networks of agents with switching topology and time-delays. IEEE Transactions on automatic control, 49(9), pp.1520-1533.

[8] Qin, J., Ma, Q., Shi, Y. and Wang, L., 2016. Recent advances in consensus of multi-agent systems: A brief survey. IEEE Transactions on Industrial Electronics, 64(6), pp.4972-4983.

[9] T. C. Silva, F. O. Souza, and L. C. Pimenta, "Consensus in multi-agent systems subject to input saturation and time-varying delays," International Journal of Systems Science, vol. 52, no. 7, pp. 1479-1498, 2021.

[10] Michel, O., 2004. Cyberbotics ltd. webots™: professional mobile robot simulation. International Journal of Advanced Robotic Systems, 1(1), p.5.

[11] Adam, Y.M., Sariff, N.B. and Algeelani, N.A., 2021, June. E-puck mobile robot obstacles avoidance controller using the fuzzy logic approach. In 2021 2nd International Conference on Smart Computing and Electronic Enterprise (ICSCEE) (pp. 107-112). IEEE.

[12] Puriyanto, R.D., Wulandari, I., Ma'Arif, A., Fathurrahman, H.I.K. and Rosyady, P.A., 2022, December. Webots-Based: Design of Collision Free Trajectory of Wheeled Mobile Robot Using Artificial Potential Field. In 2022 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET) (pp. 56-61). IEEE.

[13] Xinchi, T., Huajun, Z., Wenwen, C., Peimin, Z., Zhiwen, L. and Kai, C., 2018, November. A research on intelligent obstacle avoidance for unmanned surface vehicles. In 2018 Chinese Automation Congress (CAC) (pp. 1431-1435). IEEE.