EXPLORING THE FEASIBILITY OF SIM2REAL TRANSFER IN REINFORCEMENT LEARNING
APPLICATION IN MAZE NAVIGATION

INTERNAL PROMOTOR: GEVAERT WOUTER
EXTERNAL PROMOTOR: SAM DE BEUF

RESEARCH CONDUCTED BY
LUCAS DRIESSENS

FOR OBTAINING A BACHELOR'S DEGREE IN
MULTIMEDIA & CREATIVE TECHNOLOGIES

HOWEST | 2023-2024

Abstract

This research explores the feasibility of transferring a trained reinforcement learning (RL) agent from a simulation to the real world, focusing on maze navigation. The primary objective is to determine if and how an RL agent, specifically a Double Deep Q-Network (DDQN), can successfully navigate a physical maze after being trained in a virtual environment.

First, we explore suitable virtual environments for training an RL agent and evaluate the most effective reinforcement learning techniques for this application. The study then addresses the challenges in translating simulation-trained behaviors to real-world performance, such as sensor data interpretation and movement replication.

Results show that the DDQN agent, trained in a simulated maze, can navigate a physical maze with some challenges in sensor data interpretation and more importantly movement replication. Practical solutions, including sensor calibration and algorithm adjustments, were implemented to improve real-world performanceResults show that the DDQN agent, trained in a simulated maze, can navigate a physical maze with some challenges in sensor data interpretation and, more importantly, movement replication. Practical solutions, including sensor calibration and algorithm adjustments, were implemented to improve real-world performance.

This study contributes to AI and robotics by providing insights and methodologies for sim-to-real transfer in RL, with potential applications extending beyond robotics to other fields where simulation-based training is essential.This study contributes to AI and robotics by providing insights and methodologies for sim-to-real transfer in RL, with potential applications extending beyond robotics to other fields where simulation-based training is essential.

Preface

This bachelor thesis, titled “Exploring the Feasibility of Sim2Real Transfer in Reinforcement Learning,” is the final project of my studies in Multimedia & Creative Technology at Howest, University of Applied Sciences. The main question it tackles is: “Can a trained RL agent be successfully transferred from a simulation to the real world?” This question highlights my curiosity about how virtual simulations can be used in real-life applications and addresses a significant challenge in AI: making systems that can work in real-world conditions.

My interest in Sim2Real transfer started during the ‘Advanced AI’ classes and the ‘Research Project’ module. Learning about reinforcement learning and seeing how simulated environments can mimic complex real-world behaviors got me excited to explore their practical uses. This thesis delves into the theory behind Sim2Real transfer and tests its feasibility through various experiments aimed at improving the process and making it more reliable.

The research combines theoretical studies with practical experiments. The theoretical part provides a solid background, while the experiments test how well RL agents perform in different controlled scenarios. By evaluating these agents, the research aims to find the best strategies for successfully transferring them from simulations to real-world applications.

Lucas Driessens

01-06-2024

Acknowledgements

I extend heartfelt thanks to my coach and supervisor, Gevaert Wouter, for his invaluable guidance and insights throughout this research. His expertise and mentorship have been fundamental to my scholarly and personal growth. Gratitude is also due to Amaury Van Naemen for his technical support with 3D printing components crucial for my experiments. His assistance was pivotal in navigating the practical challenges of my research.

I would also like to thank Sam De Beuf for not only being my internship supervisor but also my external promotor for this thesis, and for allowing me the freedom to conduct research within my internship. This opportunity helped me grow significantly as a person and gain valuable soft skills, such as learning to accept feedback, considering different solutions, and being flexible in problem-solving.

Additional appreciation goes to the faculty and staff at Howest, whose commitment to fostering an innovative educational environment has profoundly influenced my development. Their dedication to excellence and support for student initiatives have been instrumental in shaping my academic journey.

Table of Contents

List of Figures

1 Glossary of Terms

1. Artificial Intelligence (AI): The simulation of human intelligence processes by machines, especially computer systems, enabling them to perform tasks that typically require human intelligence.
2. Double Deep Q-Network (DDQN): An enhancement of the Deep Q-Network (DQN) algorithm that addresses the overestimation of action values, thus improving learning stability and performance.
3. Epsilon Decay: A technique in reinforcement learning that gradually decreases the rate of exploration over time, allowing the agent to transition from exploring the environment to exploiting known actions for better outcomes.
4. Mean Squared Error (MSE): A loss function used in regression models to measure the average squared difference between the estimated values and the actual value, useful for training models by minimizing error.
5. Motion Processing Unit (MPU6050): A sensor device combining a MEMS (Micro-Electro-Mechanical Systems) gyroscope and a MEMS accelerometer, providing comprehensive motion processing capabilities.
6. Policy Network: In reinforcement learning, a neural network model that directly maps observed environment states to actions, guiding the agent's decisions based on the current policy.
7. Raspberry Pi (RPI): A small, affordable computer used for various programming projects, including robotics and educational applications.
8. RC Car: A remote-controlled car used as a practical application platform in reinforcement learning experiments, demonstrating how algorithms can control real-world vehicles.
9. Reinforcement Learning (RL): A subset of machine learning where an agent learns to make decisions by taking actions within an environment to achieve specified goals, guided by a system of rewards and penalties.
10. Sim2Real Transfer: The practice of applying models and strategies developed within a simulated environment to real-world situations, crucial for bridging the gap between theoretical research and practical application.
11. Target Network: Utilized in the DDQN framework, a neural network that helps stabilize training by providing consistent targets for the duration of the update interval.
12. Virtual Environment: A simulated setting designed for training reinforcement learning agents, offering a controlled, risk-free platform for experimentation and learning.
13. Wheel Slippage: A phenomenon where the wheels of a vehicle lose traction, causing them to spin without propelling the vehicle forward, often encountered in real-world scenarios with uneven terrain.
14. Ultrasonic Distance Sensor (HC-SR04): A sensor that uses ultrasonic waves to measure distance, commonly employed in robotics for obstacle detection and navigation.
15. Over the air updates (OTA): A method of remotely updating software or firmware on devices, allowing for seamless upgrades and maintenance without physical access to the device.
16. autonomous vehicle (AV): A self-driving vehicle capable of navigating and operating without human intervention, relying on sensors, algorithms, and AI to perceive and interact with the environment.

2 List of Abbreviations

1. AI - Artificial Intelligence
2. DDQN - Double Deep Q-Network
3. DQN - Deep Q-Network
4. ESP32 - Espressif Systems 32-bit Microcontroller
5. HC-SR04 - Ultrasonic Distance Sensor
6. MSE - Mean Squared Error
7. MPU6050 - Motion Processing Unit (Gyroscope + Accelerometer)
8. PPO - Proximal Policy Optimization
9. RC - Remote Controlled
10. RPI - Raspberry Pi
11. RL - Reinforcement Learning
12. RCMazeEnv - RC Maze Environment (Custom Virtual Environment for RL Training)
13. Sim2Real - Simulation to Reality Transfer
14. OTA Over the air updates
15. AV - autonomous vehicle

3 Introduction

3.1 Navigating the Maze: Sim-to-Real Transfer in Reinforcement Learning

In our increasingly interconnected world, the boundaries between virtual simulations and tangible reality are blurring. Consider the task of training a robot in a computer simulation to navigate a physical maze. This challenge raises critical questions about the transferability of learned behaviors from the digital realm to the real world. This thesis explores the captivating intersection of Reinforcement Learning (RL) and sim-to-real transfer.

3.2 Sim-to-Real Transfer: Bridging the Gap

Sim-to-real transfer involves translating learned behaviors from simulated environments to effective actions in the real world. While simulations provide a safe and controlled space for training, they often differ significantly from reality due to factors like sensor noise, friction, lighting conditions, and unexpected obstacles. This research aims to address these discrepancies.

3.3 The Maze Navigation Challenge: RC Cars and Algorithms

This study focuses on maze navigation using a remote-controlled (RC) car equipped with sensors. The car learns optimal paths, avoids dead ends, and optimizes its trajectory in a simulated maze. The key question is whether this digital training can translate seamlessly to navigating a physical maze, where real-world challenges like friction and uneven terrain await.

3.4 The Expedition: Four Key Steps

1. Simulator Design: Creating a realistic maze simulator that captures physical nuances such as wheel slippage, sensor noise, and limited field of view. This simulator allows the virtual car to explore and learn through trial and error, ensuring the RL agent is well-prepared for real-world conditions.

2. Transfer Learning Strategies: Employing techniques like domain adaptation, fine-tuning, and meta-learning to bridge the gap between simulation and reality. These strategies ensure the RL agent can adapt its learned behaviors to varied and unpredictable real-world conditions.

3. Sensor Calibration: Calibrating the RC car's sensors, and motors to match their virtual counterparts. This involves exploring sensor fusion and adaptation methods to maintain the integrity of the agent's learned behaviors and ensure accurate real-world operation.

4. Robust Policies: Developing policies resilient to noisy data and unexpected scenarios. This involves training the RL agent with a variety of disturbances in the simulation to handle real-world complexities, such as dynamic obstacles and environmental changes.

3.5 Beyond Mazes: A Broader Canvas

While this research focuses on maze navigation, its implications extend far beyond. The principles of sim-to-real transfer can be applied to autonomous drones in urban landscapes, self-driving cars avoiding pedestrians, or medical robots operating in cluttered hospital rooms. Sim-to-real transfer is the key to making these scenarios feasible.

So buckle up (or tighten your wheel nuts), as we embark on this thrilling expedition. The RC car awaits, ready to unravel the mysteries of both simulation and reality.

4 Background on Reinforcement Learning

The challenge of sim-to-real transfer is pivotal in the deployment of autonomous systems, influencing applications ranging from robotic navigation to self-driving vehicles (Sutton & Barto, 2018; van Hasselt et al., 2016). Recent advancements in RL, such as the introduction of Proximal Policy Optimization (Schulman et al., 2017) and Soft Actor-Critic algorithms (Haarnoja et al., 2018), have shown promise in various domains. However, the discrepancy between simulated and real environments, often referred to as the 'reality gap' (Koos et al., 2013), poses a major hurdle.

Several approaches have been proposed to bridge this gap. Domain randomization, for instance, involves training models on a variety of simulated environments with different parameters to improve their robustness (Tobin et al., 2017). Another promising technique is domain adaptation, which seeks to align the simulated and real-world data distributions (Ganin et al., 2016). Despite these advancements, challenges remain, particularly in ensuring the transferability of learned behaviors in complex, dynamic environments (Zhao et al., 2020).

This thesis builds on these foundations by exploring the feasibility of transferring RL agents trained in a simulated maze environment to a real-world RC car setup. By leveraging the Double Deep Q-Network (DDQN) architecture, known for its reduced overestimation bias (van Hasselt et al., 2016), this study aims to enhance the reliability of sim-to-real transfer in maze navigation tasks. The chosen approach addresses the limitations of prior methods by integrating robust policy development and comprehensive sensor calibration, providing a novel contribution to the field.

Reinforcement Learning (RL) employs a computational approach where agents learn to optimize their action sequences through trials and errors, engaging with their environment to maximize rewards over time. This learning framework is built upon the foundation of Markov Decision Processes (MDP), which includes:

- States (S): A definitive set of environmental conditions.
- Actions (A): A comprehensive set of possible actions for the agent.
- Transition Probabilities ($P(s_{t+1}|s_t, a_t)$): The likelihood of moving from state s_t to state s_{t+1} after the agent takes action a_t at time t.
- Rewards ($R(s_t, a_t)$): The reward received when transitioning from state s_t to state s_{t+1} due to action a_t.

The principles of Reinforcement Learning, particularly the dynamics of Markov Decision Processes involving states S, actions A, transition probabilities $P(s_{t+1}|s_t, a_t)$, and rewards $R(s_t, a_t)$, form the foundation of how agents learn from and interact with their environment to optimize decision-making over time. This understanding is crucial in the development of autonomous vehicles, improving navigational strategies, decision-making capabilities, and adaptation to real-time environmental changes. The seminal work by R.S. Sutton and A.G. Barto significantly elucidates these principles and complexities of RL algorithms [18].

5 Research Questions

This investigation centers around the question: “Is it possible to transfer a trained RL-agent from a simulation to the real world? (case: maze)” To address this question, we’ll explore various aspects of RL training and implementation:

1. Which virtual environments exist to train a virtual RC-car?: determine which virtual environments are most effective for RL training.
2. Which reinforcement learning techniques can I best use in this application?: Identifying RL techniques suitable for autonomous navigation.
3. Can the simulation be transferred to the real world? Explore the difference between how the car moves in the simulation and in the real world.: Assessing how well the agent adapts to real-world dynamics.
4. Does the simulation have any useful contributions? In terms of training time or performance.: Evaluating training effectiveness and optimizing performance through simulation.
5. How can the trained model be transferred to the real RC car? How do we need to adjust the agent and the environment for it to translate to the real world?: Discussing necessary adjustments for real-world application.

6 Methodology

This section explores the Reinforcement Learning Maze Navigation (RCMazeEnv) method, which utilizes a Double Deep Q-Network (DDQNAgent) architecture. We will delve into the maze environment setup, the design of the DDQN agent, and the comprehensive training algorithm, incorporating mathematical functions to describe the system’s mechanics.

6.1 Environment Setup (RCMazeEnv)

RCMazeEnv, a custom 12x12 cell maze built on OpenAI Gym, features walls ('1') and paths ('0'). The agent starts at position (1,1) aiming to reach (10,10), equipped with forward, left, and right sensors.

To aid with navigation, the agent has sensors providing readings in three directions: front, left, and right. These sensors measure the distance to the nearest wall in their respective directions, crucial for decision-making. The environment's state space ($\mathcal{S}$) includes the agent's current position (x, y), orientation θ (north, east, south, or west), and sensor readings $\{s_{\text{front}}, s_{\text{left}}, s_{\text{right}}\}$. The agent's goal is efficient maze navigation, reaching the goal while avoiding collisions with walls or getting stuck in corners all while optimizing its path based on sensor inputs and past experiences.

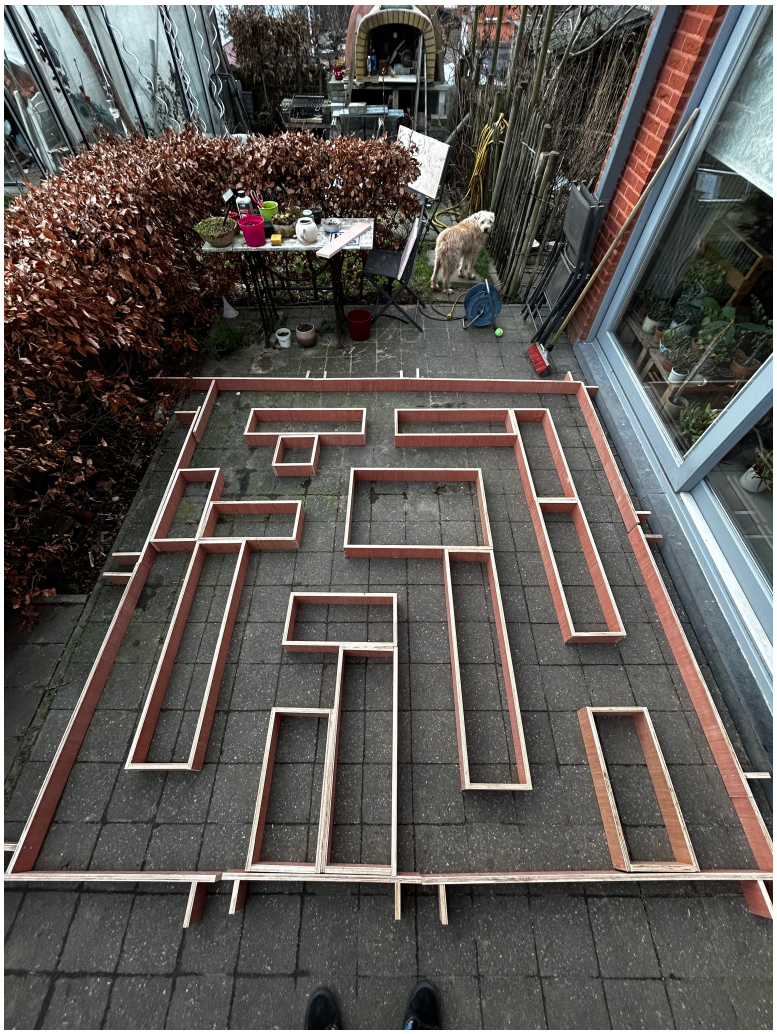


Figure 1: Real life Maze Build (Image created by author)

6.1.1 Web Application Interface

A web application was developed to serve as a control interface for the RC car, allowing real-time monitoring and control of the car's movements. The interface displays sensor readings and includes an emergency stop feature.

6.2 Agent Design (DDQNAgent)

The agent uses a Double Deep Q-Network (DDQN) architecture to learn the optimal policy π^*. DDQN is an enhancement over the standard DQN, aiming to reduce overestimation of Q-values by separating action selection from evaluation [19].

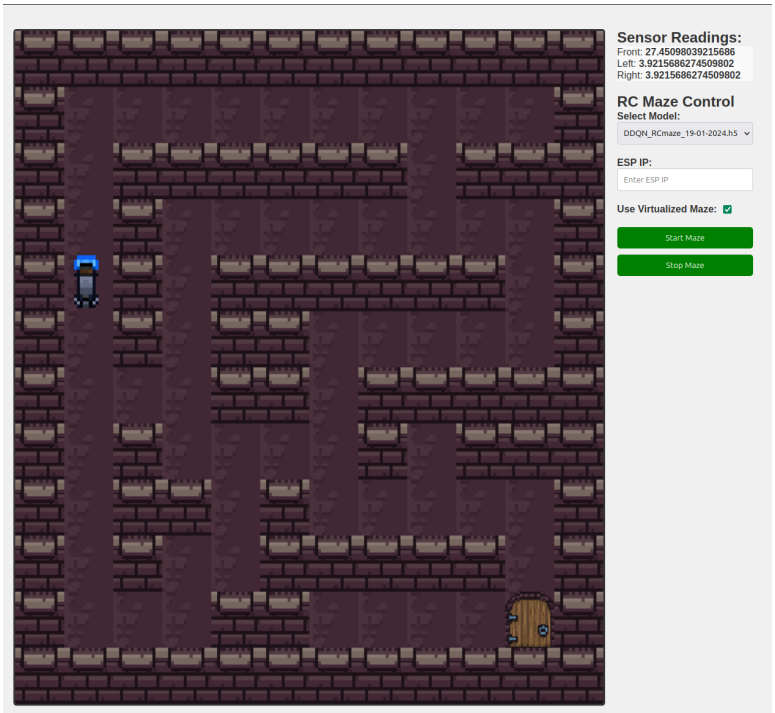


Figure 2: Web App (Image created by author)

- Policy Network: Estimates the Q-value $Q(s, a; \theta)$ for taking action a in state s, with weights θ. This network selects actions based on the current policy.
- Target Network: Independently parameterized by weights θ^-, it estimates the target Q-value for updating the policy network. The target network mirrors the policy network's architecture but updates less frequently to provide stable target values.

The DDQN update equation modifies the Q-function:

$$Y_t^{DDQN} = R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \theta); \theta^-\right)$$

Where:

- R_{t+1} is the reward received after taking action a in state s.
- γ is the discount factor.
- $\underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \theta)$ selects the action using the policy network.
- $Q\left(S_{t+1}, a; \theta^-\right)$ evaluates the action using the target network.

This approach reduces overestimation by separating the max operation in the target, mitigating overoptimism observed in Q-learning [20].

The action space $\mathcal{A}$ and other agent setup details remain consistent. DDQN significantly improves stability and performance by addressing Q-value overestimation, although its effectiveness varies depending on the task compared to traditional DQN approaches [21].

6.3 Training Process

The training process involves using experience replay, where transitions (s, a, r, s') are stored in a replay buffer denoted as D. Our objective is to train a Double Deep Q-Network (DDQN) by minimizing the loss function $L(\theta)$. This loss function quantifies the discrepancy between the current Q-values and the target Q-values:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma Q(s', \underset{a'}{\operatorname{argmax}}\, Q(s', a'; \theta); \theta^-) - Q(s, a; \theta)\right)^2\right]$$

Where:

- s represents the current state.
- a corresponds to the action taken.
- r denotes the received reward.
- s' signifies the subsequent state.
- θ^- refers to the weights of the target network.
- γ represents the discount factor.

To enhance training stability, we periodically update the target network's weights with those of the policy network. Additionally, we employ an epsilon-greedy strategy for action selection. Initially, we prioritize exploration (with ϵ set to 0.99), gradually reducing exploration as training progresses with a decay rate of 0.99973 . This balance between exploration and exploitation contributes to the DDQN's overall performance.

7 Reward Function and completion components

In the context of maze navigation, designing an effective reward function is crucial for guiding an agent's learning process. Below, I outline the key components of the reward function used in our framework:

1. Goal Achievement Bonus (R_{goal}):
   - Reaching the goal is the primary objective of the maze navigation task.
   - Upon achieving this objective, the agent receives a substantial reward: $R_{\text{goal}} = +500$.
   - However, if the agent takes an excessively long route to reach the goal (more than 1000 steps), it gets a penalty: $R_{\text{goal}} = -200$.
   - This mechanism encourages efficient navigation while still rewarding successfully reaching the goal
2. Proximity Reward ($R_{\text{proximity}}$):
   - Encourages the agent to minimize its distance to the goal over time.
   - The reward decreases as the distance to the goal increases: $R_{\text{proximity}} = \frac{50}{d_{\text{goal}}+1}$.
   - Here, d_{goal} represents the Euclidean distance to the goal.
3. Progress Reward (R_{progress}):
   - Provides immediate feedback based on the agent's movement relative to the goal.
   - If the distance to the goal decreases, the agent receives a positive reward: $R_{\text{progress}} = +50$.
   - if the distance increases, it gets a penalty: $R_{\text{progress}} = -25$.
   - This encourages smarter navigation decisions.
4. Exploration Penalty (R_{revisit}):
   - Discourages repetitive exploration of the same areas.
   - The agent receives a penalty for re-entering previously visited cells: $R_{\text{revisit}} = -10$.
   - This promotes exploration of new paths and prevents the agent from getting stuck.
5. Efficiency Penalty ($R_{\text{efficiency}}$):
   - Every step the agent takes incurs a small penalty: $R_{\text{efficiency}} = -5$.
   - Balances the need for exploration with the goal of reaching the destination efficiently.

7.1 Termination conditions

To determine whether the environment has reached a "done" or "ended" state, 2 conditions have been established. These conditions include: surpassing 3000 steps and the RC car reaching the goal position of $(10, 10)$.

The termination condition can be expressed as:

$$\text{terminate}(steps, position) = \begin{cases} \text{true, "Exceeded max steps"} & \text{if } steps > 3000 \\ \text{true, "Goal reached"} & \text{if } position = (10, 10) \\ \text{false, "Continue"} & \text{otherwise} \end{cases}$$

7.2 Expanding Real-World Testing

In this study, I conducted experiments indoors to closely replicate theoretical conditions. The tests were performed on a hard cloth surface to minimize ground-related issues and ensure a consistent testing environment. This step was crucial because during real-world testing, the RC car encountered challenges on uneven surfaces.

However, the exploration wasn't limited to indoor setups alone. I also aimed to assess the adaptability and resilience of my proposed solutions in outdoor environments. Taking the experiments outdoors posed significant challenges due to the differences in ground conditions. Outdoor landscapes are diverse and unpredictable, which exposed limitations in my current method's ability to handle such variations. This highlighted the need for further research and improvements in the methods used, such as the hardware limitations.

8 Experimental Outcomes and Implementation Details

8.1 Simulation Design and Agent Framework

- RCMazeEnv: This environment was specifically designed for the study. It simulates a robotic car navigating through a maze, aiming to replicate real-world physics and limitations. The maze layout, the robotic car's movement capabilities, and its sensor configurations all contribute to creating an authentic testbed for reinforcement learning algorithms.
- Double Deep Q-Network (DDQN): Instead of relying on a single neural network, DDQN employs two networks to enhance standard reinforcement learning techniques. By doing so, it mitigates the overvaluation of Q-values. The policy network and the target network collaborate, continuously interpreting sensor data and improving the learning process.

8.2 Implementation Insights

- Interaction Between Environment and Agent: The DDQN agent adapts dynamically to the environment. It relies on sensor feedback to make decisions and fine-tune its path through the maze. This ongoing learning cycle is demonstrated on a simulation platform, allowing us to observe how the agent's tactics evolve over time.
- Application in the Real World: Transitioning from virtual training to a physical RC robot involved significant hardware adjustments and fine-tuning. Challenges like aligning sensor data and achieving precise movement control were addressed to ensure a seamless transfer from the virtual model to practical use.

8.3 Performance Evaluation

To evaluate the agent's effectiveness in navigating the maze, I used several metrics such as manually looking at the decision and seeing if the agent would get the car stuck, Reward, loss and epsilon history per episode during training. These metrics provided insights into the agent's learning progress and performance and allowed for quick adjustments to reward functions or hyperparameters.

8.4 Distinctive Elements

- Physical Maze and Digital Interface: A real maze was built to mirror the virtual `RCMazeEnv`, which is one of the requirements of ensuring the RC robot's ability to navigate the real maze. In addition, a web application was made to not only control the RC car (starting and stopping movement) but also as a virtual twin. Allowing us to see the sensor readings and decision making in real time.

9 Addressing Research Questions

9.1 1. Which Virtual Environments Exist to Train a Virtual RC-Car?

Selecting the right virtual environment is crucial for effective RL training of a virtual RC car. Several platforms are available, including Unity 3D, AirSim, CARLA, OpenAI Gym, and ISAAC Gym. For this

project, I chose OpenAI Gym due to its flexibility in creating custom environments and compatibility with Python. This choice supports seamless integration with advanced AI coursework and facilitates effective sim-to-real transfer practices [6].

Unity 3D and AirSim offer highly realistic simulations but their complexity and limited Python support pose accessibility issues. CARLA, while excellent for traditional vehicle models, is less suited for RF-cars. ISAAC Gym, primarily focused on robotics, does not align perfectly with the goals of this project. OpenAI Gym's simplicity and reinforcement learning focus make it the ideal fit for this application [6].

9.2 2. Which Reinforcement Learning Techniques Can I Best Use in This Application?

For autonomous navigation of a virtual RC car in a maze, several reinforcement learning techniques were considered, including Deep Q-Network (DQN), Double Deep Q-Network (DDQN), and Proximal Policy Optimization (PPO). DDQN was chosen as the most suitable technique for this project. DDQN's architecture addresses the overestimation bias present in DQN, enhancing the accuracy of Q-value approximations, which is crucial for navigating complex, sensor-driven RF-car environments [6].

DQN, although effective in processing high-dimensional sensory inputs, struggles in unpredictable dynamic environments. PPO, which focuses on direct policy optimization, lacks the precision in value estimation needed for RF-car training. Empirical trials confirmed that DDQN outperforms these methods in intricate maze-like virtual RF-car scenarios [6].

9.3 3. Can the Simulation be Transferred to the Real World? Explore the Difference Between How the Car Moves in the Simulation and in the Real World.

Transferring simulation-trained models to real-world applications involves addressing discrepancies in sensor data interpretation, action synchronization, and physical dynamics. These differences can affect how well the agent adapts to real-world dynamics. For instance, real-world sensors may introduce noise and inaccuracies not present in the simulation, and the physical dynamics of the car, such as friction and wheel slippage, can differ significantly [6].

To mitigate these issues, techniques such as sensor data normalization and action synchronization mechanisms were implemented to align simulation outcomes with real-world performance. Introducing failsafe mechanisms and adjusting motor control timings proved essential in reducing collision risks and movement inaccuracies during sim-to-real transfer. Iterative testing and adaptation played a vital role in this process [6].

9.4 4. Does the Simulation Have Any Useful Contributions? In Terms of Training Time or Performance.

Simulation training offers significant advantages, including efficiency, safety, and computational power. It allows for uninterrupted, automated training sessions, eliminating the risks associated with real-world testing. Leveraging powerful computing resources accelerates the training process, making simulation indispensable for RF-car development [6].

Comparing simulation and real-world training outcomes highlights the practicality and effectiveness of simulation in developing autonomous driving models. The ability to conduct extensive training in a controlled environment significantly enhances the model's performance and robustness before real-world deployment [6].

9.5 5. How Can the Trained Model be Transferred to the Real RC Car? How Do We Need to Adjust the Agent and the Environment for It to Translate to the Real World?

Applying a trained model to a physical RC car requires several adjustments. Effective sim-to-real adaptation involves fine-tuning sensor interpretations, implementing action synchronization measures, and adjusting physical dynamics to mirror the simulation [6]. These steps include:

- Sensor Calibration: Ensuring the sensors used in the real RC car provide data in a format compatible with the trained model.
- Motor Control Adjustments: Adjusting motor control timings to match the physical dynamics of the real car.
- Failsafe Mechanisms: Introducing mechanisms to handle unexpected scenarios and reduce collision risks.
- Incremental Testing: Conducting iterative tests in real environments to validate and refine the agent's performance.

These adjustments are essential to ensure the successful application of the model in real-world scenarios, facilitating robust and reliable autonomous driving systems.

10 Model Architecture and Training Insights

To understand how our Double DQN model learns and makes decisions, let's dive into its architecture. The model consists of four dense layers, which output three actions tailored to the RC car's movement, enabling it to navigate the maze with ease.

The Double DQN architecture addresses the issue of overestimation in action-value functions, a common problem in reinforcement learning. By decoupling the action selection and action evaluation processes during the target value calculation, the Double DQN mitigates the propagation of overestimation errors. This leads to more accurate value estimates and improved policy performance [19] [20].

In terms of model design, Double DQN utilizes two networks: the primary network and the target network. The primary network is used to select the best action, while the target network evaluates the value of that action. This method effectively reduces the bias introduced by the max operator used in standard DQN updates [19] [21]. Empirical studies have shown that Double DQN not only improves stability and performance but also requires minimal changes to the original DQN architecture, making it a practical and efficient enhancement [20].

Research has shown that, all things being equal, simpler models are often preferred in reinforcement learning. This is because they can lead to better performance, faster learning, and improved generalization. However, finding the right balance of model complexity is crucial. Simplicity is not just about the number of layers or parameters but also about capturing temporal regularities, such as repetitions, in sequential strategies [27] [19].

With these insights in mind, we designed the Double DQN model to strike a balance between simplicity and effectiveness, ensuring optimal performance in maze navigation tasks. By leveraging the strengths of simpler models while addressing critical performance issues, the Double DQN maintains a robust and efficient architecture for reinforcement learning applications.

Model Architecture:

```
# Model: "sequential_52"

# Layer (type) Output Shape Param

=================================================================
dense_200 (Dense) (None, 32) 224
dense_201 (Dense) (None, 64) 2112
dense_202 (Dense) (None, 32) 2080
dense_203 (Dense) (None, 3) 99
=================================================================
Total params: 4515 (17.64 KB)
Trainable params: 4515 (17.64 KB)
Non-trainable params: 0 (0.00 Byte)

---
```

10.1 Training Parameters

The training of the Double DQN agent was governed by the following parameters:

Figure 3: Model Architecture (Image created by author)

- Discount Factor (`DISCOUNT`): 0.90
- Batch Size: 128
  - Number of steps (samples) used for training at a time.
- Update Target Interval (`UPDATE_TARGET_INTERVAL`): 2
  - Frequency of updating the target network.
- Epsilon (`EPSILON`): 0.99
  - Initial exploration rate.
- Minimum Epsilon (`MIN_EPSILON`): 0.01
  - Minimum value for exploration rate.
- Epsilon Decay Rate (`DECAY`): 0.99973
  - Rate at which exploration probability decreases.
- Number of Episodes (`EPISODE_AMOUNT`): 170
  - Total episodes for training the agent.
- Replay Memory Capacity (`REPLAY_MEMORY_CAPACITY`): 2,000,000
  - Maximum size of the replay buffer.
- Learning Rate: 0.001
  - The rate at which the model learns from new observations.

10.2 Training Procedure

1. Initialization: Start with a high exploration rate (`EPSILON`) allowing the agent to explore the environment extensively.
2. Episodic Training: For each episode, the agent interacts with the environment, collecting state, action, reward, and next state data.
3. Replay Buffer: Store these experiences in a replay memory, which helps in breaking the correlation between sequential experiences.
4. Batch Learning: Randomly sample a batch of experiences from the replay buffer to train the network.
5. Target Network Update: Every `UPDATE_TARGET_INTERVAL` episodes, update the weights of the target network with those of the policy network.
6. Epsilon Decay: Gradually decrease the exploration rate (`EPSILON`) following the decay rate (`DECAY`), shifting the strategy from exploration to exploitation.
7. Performance Monitoring: Continuously monitor the agent's performance in terms of rewards and success rate in navigating the maze.

10.3 Evaluation Metrics Overview

10.3.1 Simulation Metrics

Episodic Performance

- Objective and Goal: The aim of this metric is to monitor the agent's progress in mastering the maze. By evaluating the learning curve, I see how efficiently the agent can navigate to the end of the maze over successive trials. This gives us insights into its ability to optimize strategies and adapt over time.

- How it's Assessed: I measure the number of episodes the agent needs before it can consistently complete the maze. A reduction in episodes over time is a good indicator that the agent is learning and adapting well.
- Analytical Techniques: To examine episodic performance, we either conduct statistical analyses or create visual plots, such as learning curves. These tools help us track and visualize changes in performance throughout the training period.
- Accuracy and Consistency Measures: To maintain accuracy and consistency, I ensure data integrity and control experimental conditions. Averaging results across multiple trials helps smooth out any randomness in the learning process, providing a clearer picture of the agent's performance.

Step Efficiency

- Objective and Goal: This metric evaluates the agent's decision-making efficiency and ability to optimize its path through the maze. By measuring the steps the agent takes to solve the maze, fewer steps indicate a more efficient and smarter learning process.
- How it's Assessed: I keep track of the steps required to reach the maze's endpoint in each episode and analyze the reduction in steps over time.
- Analytical Techniques: I use quantitative analysis to examine trends in step count. Smoothing techniques may be applied to provide a clearer view of the overarching trends amidst episode-to-episode variability.
- Accuracy and Consistency Measures: To ensure reliable metrics, I replicate tests and average results, maintaining the same maze configuration for all experiments.

MSE Loss Measurement

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2$$

where:

y_i represents the actual value.
$\hat{y}_i$ represents the predicted value.
N is the total number of observations.

- Objective and Goal: This metric quantifies the accuracy of the agent's predictions by measuring the squared discrepancies between predicted values and actual outcomes, providing a clear gauge of learning precision.
- How it's Assessed: Using the provided mathematical formula, I average the squared differences across all predictions for an episode or series of episodes.
- Analytical Techniques: Calculating MSE is straightforward, but understanding its trend requires examining how it correlates with different stages of the agent's learning, such as initial acquisition of knowledge versus later strategy refinement.
- Accuracy and Consistency Measures: Regularly evaluating against a validation set or maintaining a consistent testing framework ensures reliable insights into the agent's predictive accuracy and learning trajectory.

Reward Trend Analysis

- Objective and Goal: This analysis helps determine how effectively the agent's actions lead to positive outcomes, which are indicative of its learning and strategy development.
- How it's Assessed: By tracking and analyzing the rewards the agent accumulates over time, looking for trends that show an increase in reward collection.
- Analytical Techniques: Employing time series analysis or plotting cumulative rewards can vividly illustrate improvements in the agent's decision-making and learning.
- Accuracy and Consistency Measures: Averaging trends over several runs and keeping the reward structures consistent throughout the experiments to ensure comparability.

Epsilon Decay Tracking

- Objective and Goal: This metric monitors how well the agent balances exploration of new paths with exploitation of known successful strategies, key for adapting learning methods effectively.

- How it's Assessed: By observing the decline in the epsilon parameter over episodes, which indicates the agent's shift from exploring to exploiting.
- Analytical Techniques: Plotting epsilon values across episodes helps visualize how the agent's learning strategy evolves over time.
- Accuracy and Consistency Measures: Applying the epsilon decay strategy uniformly across all training sessions and maintaining consistent experimental conditions to ensure comparability of results.

10.3.2 Real-World Metrics

Transitioning to real-world application involved assessing how well the strategies developed in simulation held up when the agent faced a physical maze with real obstacles and constraints.

- Maze Navigation: Observing the RC car as it maneuvered through a real-world maze served as direct proof of how effectively the training translated from simulation to reality. This hands-on test demonstrated the practical utility of the trained agent in navigating complex paths.
- Sensor Data Analysis: By examining the real-time sensor data during navigation trials, I gained a deeper insight into how the agent interacts with its physical environment. This analysis was crucial for evaluating the agent's ability to avoid obstacles and optimize its pathfinding strategies efficiently.

11 Background on Reinforcement Learning Algorithms

11.1 Background on Double Deep Q-Network (DDQN)

The Double Deep Q-Network (DDQN) is an enhancement of the Deep Q-Network (DQN), a pivotal algorithm in the field of deep reinforcement learning that integrates deep neural networks with Q-learning. DQN itself was a significant advancement as it demonstrated the capability to approximate the Q-value function, which represents the expected reward for taking an action in a given state, using high-capacity neural networks.

11.1.1 Evolution from DQN to DDQN

DQN Challenges: While DQN substantially improved the stability and performance of Q-learning, it was susceptible to significant overestimations of Q-values due to the noise inherent in the approximation of complex functions by deep neural networks. This overestimation could lead to suboptimal policies and slower convergence during training.

DDQN Solution: Introduced by Hado van Hasselt et al., DDQN addresses the overestimation problem of DQN by decoupling the action selection from the target Q-value generation—a technique termed "double learning." In traditional DQN, a single neural network is used both to select the best action and to evaluate its value. DDQN modifies this by employing two networks:

- The current network determines the action with the highest Q-value for the current state.
- A separate target network, which is a delayed copy of the current network, is used to estimate the Q-value of taking that action at the next state [22].

11.1.2 The Decoupling Effect

This separation ensures that the selection of the best action is less likely to overestimate Q-values, as the estimation is made using a different set of weights, thus reducing bias in the learning process. The target network's parameters are updated less frequently (often after a set number of steps), which further enhances the algorithm's stability.

11.1.3 Impact and Applications

DDQN has been shown to achieve better performance and faster convergence in complex environments compared to DQN. It is particularly effective in scenarios where precise action evaluation is crucial, such as in video games and robotic navigation tasks. The improved reliability and accuracy of DDQN make it a valuable model for studying reinforcement learning in controlled environments where stability and efficiency are critical.

11.2 Background on Deep Q-Network (DQN)

The Deep Q-Network (DQN) algorithm represents a significant breakthrough in reinforcement learning by combining traditional Q-learning with deep neural networks. This approach was popularized by researchers at DeepMind with their notable success in training agents that could perform at human levels across various Atari games [23].

Core Mechanism: DQN uses a deep neural network to approximate the Q-value function, which is the expected reward obtainable after taking an action in a given state and following a certain policy thereafter. The neural network inputs the state of the environment and outputs Q-values for each possible action, guiding the agent's decisions.

Innovations Introduced:

- Experience Replay: DQN utilizes a technique called experience replay, where experiences collected during training are stored in a replay buffer. This allows the network to learn from past experiences, reducing the correlations between sequential observations and smoothing over changes in the data distribution.
- Fixed Q-Targets: To further stabilize training, DQN employs a separate target network, whose weights are fixed for a number of steps and only periodically updated with the weights from the training network [23].

11.2.1 DQN Advantages and Applications

DQN's ability to handle high-dimensional sensory inputs directly with minimal domain knowledge makes it highly versatile and effective in complex environments such as video games, where it can learn directly from pixels.

11.3 Background on Q-agent (Q-learning)

Q-agent, based on the Q-learning algorithm, is one of the most fundamental types of reinforcement learning methods. It is a model-free algorithm that learns to estimate the values of actions at each state without requiring a model of the environment [24].

Simplicity and Versatility: Q-learning works by updating an action-value lookup table called the Q-table, which stores Q-values for each state-action pair. These values are updated using the Bellman equation during each step of training based on the reward received and the maximum predicted reward for the next state.

Challenges: While simple and effective for smaller state spaces, Q-learning's reliance on a Q-table becomes impractical in environments with large or continuous state spaces, where the table size would become infeasibly large.

11.3.1 Q-learning Applications

Q-learning has been foundational in teaching agents in environments with discrete, limited state spaces, such as simple mazes or decision-making scenarios with clear, defined states and actions.

11.4 Background on Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a policy gradient method for reinforcement learning that simplifies and improves upon the Trust Region Policy Optimization (TRPO) approach. PPO has become popular due to its effectiveness and ease of use [25].

Optimization Technique: PPO seeks to take the largest possible improvement step on a policy while avoiding

too large updates that might lead to performance collapse. It achieves this through an objective function that includes a clipped term, penalizing changes to the policy that move it too far from the previous policy.

Advantages: PPO is robust to a variety of hyperparameters and can be used in both continuous and discrete action spaces. It has shown great success in environments ranging from simulated robotics to complex game environments.

11.4.1 PPO Applications

PPO is favored in many modern RL applications due to its balance between efficiency, ease of implementation, and strong empirical performance.

11.5 Background on Actor-Critic (AC)

Actor-Critic methods form a broad class of algorithms in reinforcement learning that combine both policy-based (actor) and value-based (critic) approaches [26].

Dual Components:

- Actor: Responsible for selecting actions based on a policy.
- Critic: Estimates the value function (or Q-value), which is used to evaluate how good the action taken by the actor is.

Advantages: By separating the action selection and evaluation, actor-critic methods can be more efficient than conventional policy-gradient methods. They reduce the variance of the updates and typically converge faster.

11.5.1 Actor-Critic Applications

Actor-Critic algorithms are versatile and can be applied to both discrete and continuous action spaces. They have been effectively used in applications that require balancing exploration of the environment with the exploitation of known rewards, such as in robotics and complex game environments.

12 Comparative Analysis of Reinforcement Learning Algorithms in Maze Navigation

In this analysis, I compare various reinforcement learning algorithms, namely Double Deep Q-Network (DDQN), Deep Q-Network (DQN), Q-agent, Actor-Critic (AC), and Proximal Policy Optimization (PPO). This comparison is based on their performance in navigating a complex maze, focusing on efficiency, learning rate, and adaptability.

12.1 Performance Metrics and Visualization

12.1.1 1. Visit Heatmaps

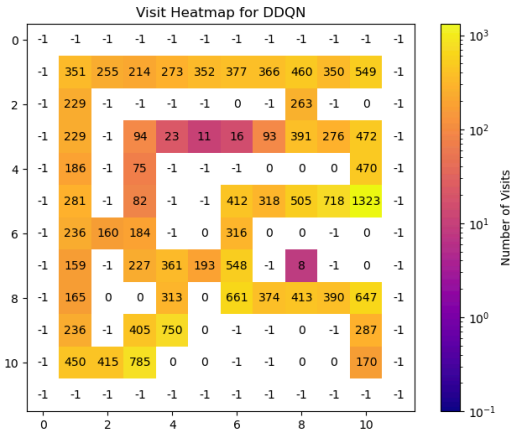


Figure 4: DDQN Heatmap (Image created by author)

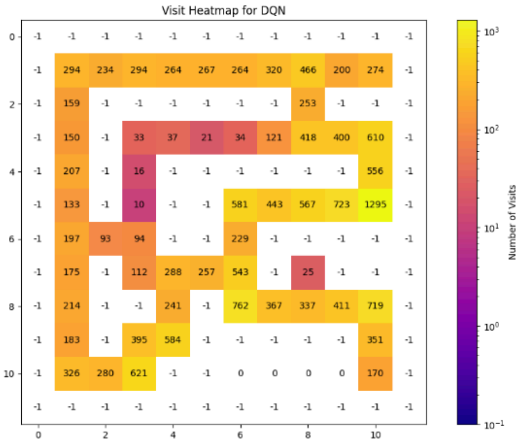


Figure 5: DQN Heatmap (Image created by author)

Commentary: The visit heatmaps provide a visual representation of the exploration patterns for different algorithms. The heatmap for DDQN shows a concentrated path, indicating the agent's ability

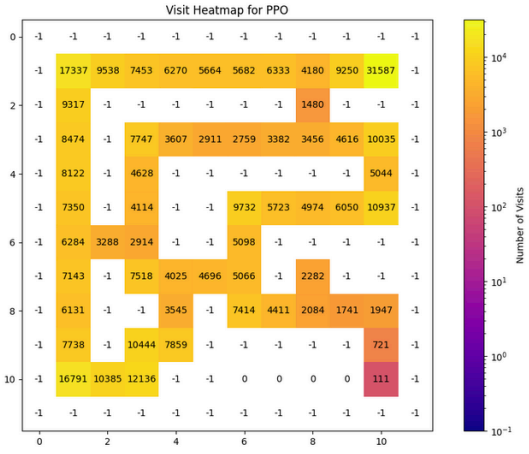
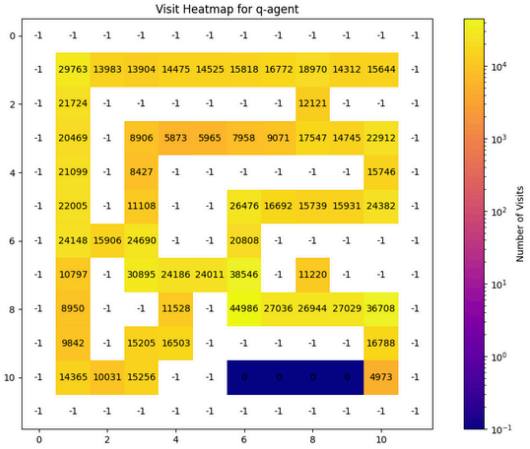


Figure 6: PPO Heatmap (Image created by author)

* 1.2*



1.2

Figure 7: Q-agent Heatmap (Image created by author)

to efficiently learn and focus on the optimal routes through the maze. Similarly, DQN exhibits focused exploration but with slightly more dispersion compared to DDQN, suggesting a robust learning process. On the other hand, PPO and Q-agent demonstrate widespread exploration across the maze, which indicates less efficient learning and decision-making. These dispersed patterns reflect the algorithms' struggle to consistently identify and follow optimal paths, resulting in suboptimal navigation strategies.

12.1.2 2. Maze Solution Efficiency

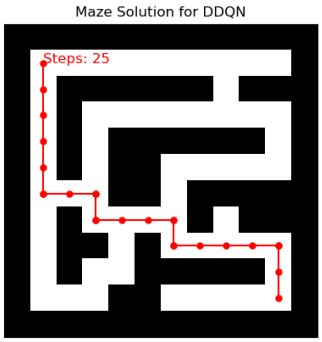


Figure 8: DDQN Maze Path (Image created by author)

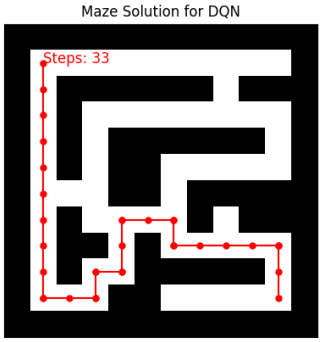


Figure 9: DQN Maze Path (Image created by author)

Commentary: The maze solution paths highlight the efficiency of each algorithm in navigating the maze. DDQN demonstrates the shortest and most direct path, indicating superior learning and optimization in its decision-making process. DQN also performs well, albeit with a slightly longer path, reflecting its effective yet slightly less optimal strategy. The Q-agent, while successful in completing the maze, takes a more winding route, suggesting a less efficient approach and a greater number of steps to reach the goal. This comparison underscores the superior efficiency of DDQN in solving the maze with minimal steps, followed closely by DQN, and identifies areas where Q-agent can improve its path optimization.

For the PPO and AC algorithms, their paths were more complex and less direct, indicating a need for further optimization and learning to enhance their maze navigation efficiency (not shown in the figures).

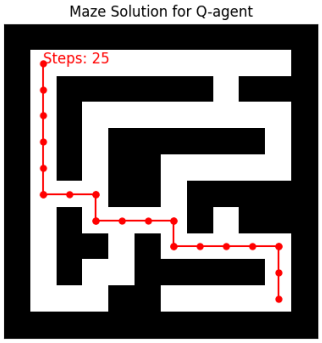


Figure 10: Q-agent Maze Path (Image created by author)

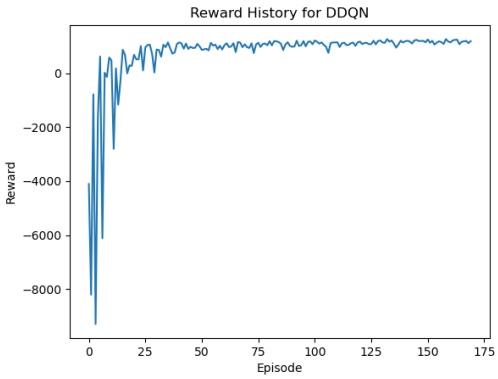


Figure 11: DDQN Reward History (Image created by author)

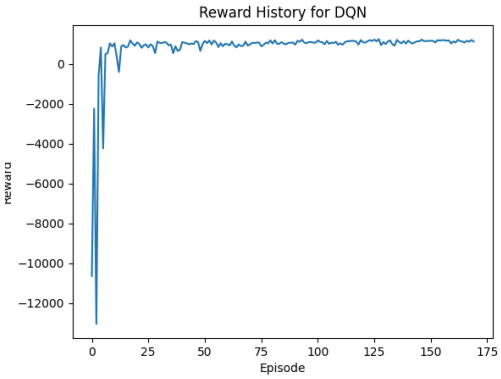


Figure 12: DQN Reward History (Image created by author)

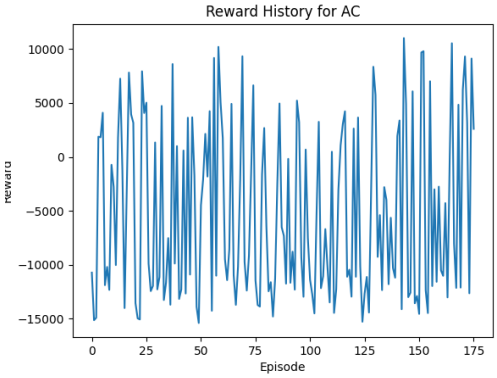


Figure 13: AC Reward History (Image created by author)

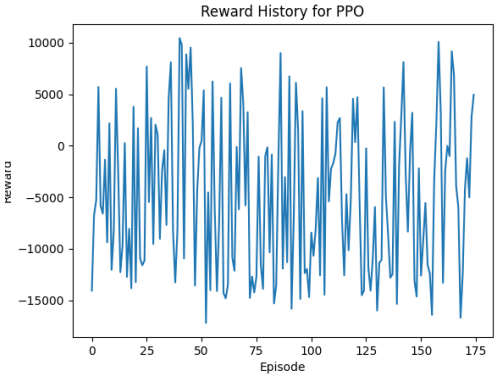


Figure 14: PPO Reward History (Image created by author)

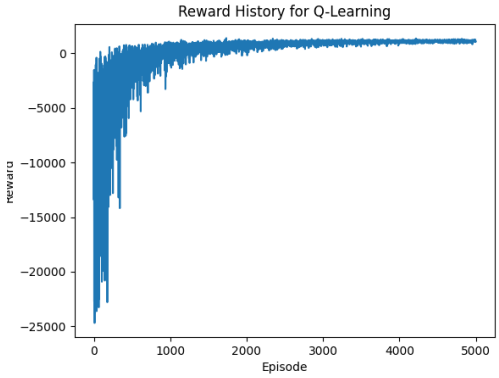


Figure 15: Q-agent Reward History (Image created by author)

12.1.3 3. Reward History and Distribution

Commentary: The reward history graphs offer insights into the learning stability and efficiency of each algorithm. DDQN and DQN display steady and consistent increases in reward accumulation, indicating stable and reliable learning processes. In contrast, AC and PPO exhibit significant fluctuations in reward history, reflecting instability and inconsistency in their learning curves. This variability suggests these algorithms encounter challenges in maintaining a steady learning trajectory. Q-agent, although stable, shows a slower rate of reward accumulation, indicating a more gradual and less efficient learning process compared to DDQN and DQN. Overall, DDQN and DQN stand out for their consistent and effective reward acquisition, while AC, PPO, and Q-agent highlight areas for potential improvement in learning stability and efficiency.

12.1.4 4. Mean Squared Error (MSE) Over Time

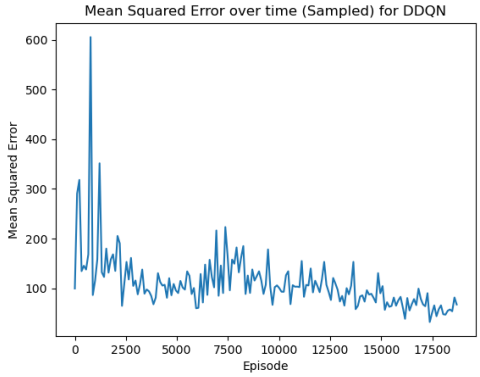


Figure 16: DDQN MSE (Image created by author)

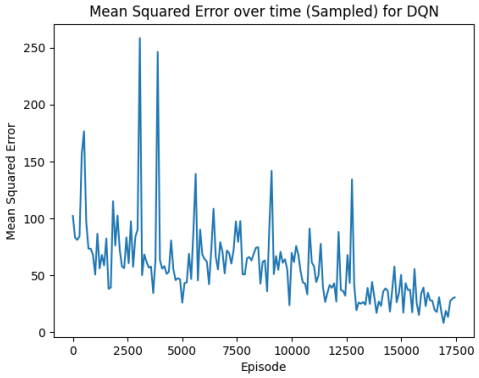


Figure 17: DQN MSE (Image created by author)

Commentary: The MSE graphs track the learning accuracy and error management of each algorithm over time. DDQN achieves the lowest and most stable MSE values, signifying its effective learning and strong capability in minimizing prediction errors. DQN also performs well with relatively stable MSE, though slightly higher than DDQN, indicating competent but less optimal error reduction. In contrast, AC and PPO show higher and more variable MSE values, pointing to less effective learning and greater difficulties in managing prediction errors. These fluctuations suggest these algorithms struggle to consistently reduce errors, impacting their overall performance. The comparison highlights DDQN's superior accuracy and error management, with DQN as a strong contender, while AC and PPO require further optimization to enhance their learning precision.

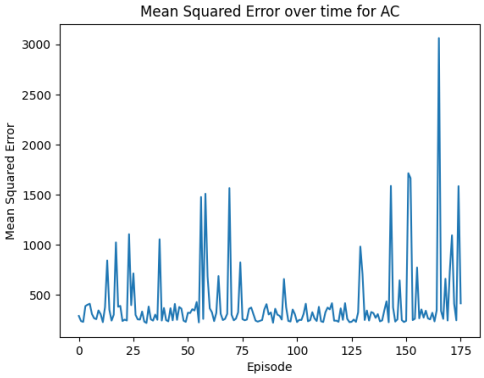


Figure 18: AC MSE (Image created by author)

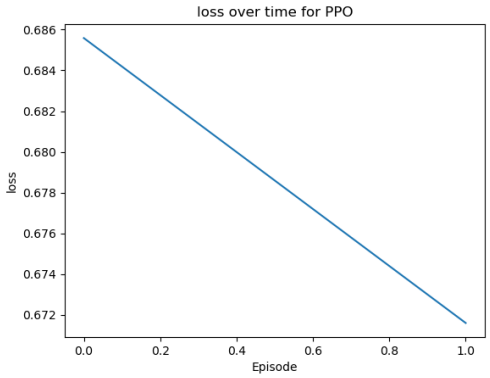


Figure 19: PPO Loss (Image created by author)

12.1.5 5. Moving Average of Rewards

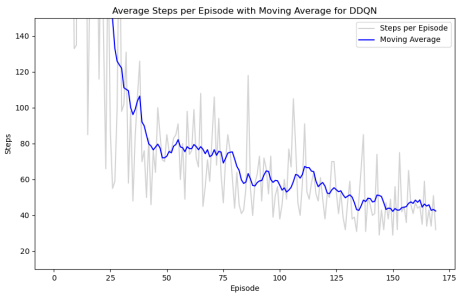


Figure 20: DDQN Moving Average (Image created by author)

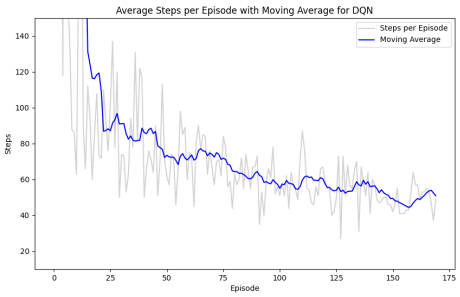


Figure 21: DQN Moving Average (Image created by author)

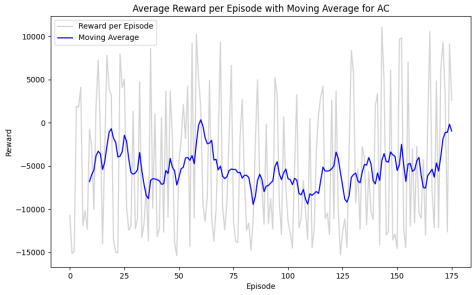


Figure 22: AC Moving Average (Image created by author)

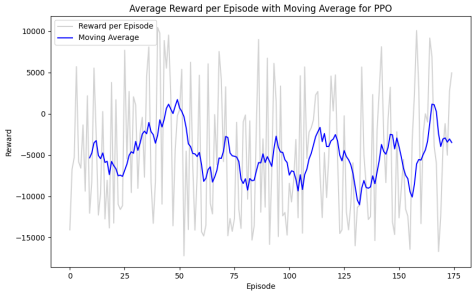


Figure 23: PPO Moving Average (Image created by author)

Commentary: The moving average of rewards provides a clear view of the long-term performance trends of each algorithm. DDQN and DQN show consistent and progressive improvement in reward accumulation, reflecting their effective learning and adaptation over time. This steady upward trend indicates robust performance enhancement. In contrast, AC and PPO display significant variability in their moving averages, suggesting inconsistent learning and slower performance improvements. The Q-agent, while showing a stable improvement trend, progresses at a slower pace compared to DDQN and DQN, highlighting a need for more efficient learning strategies. Overall, the moving averages affirm DDQN and DQN's effectiveness in continuously enhancing performance, while AC, PPO, and Q-agent demonstrate areas where more consistent and rapid improvements are needed.

12.2 Conclusion and Insights

This comprehensive analysis reveals distinct performance characteristics and efficiencies of various reinforcement learning algorithms in maze navigation. DDQN stands out for its balanced approach, efficiently solving the maze with the fewest steps while demonstrating superior stability and effective error management. DQN, though slightly less efficient in navigation, showcases robust learning stability, making it a reliable choice. Q-agent, despite its simpler approach, competes closely with DDQN in terms of steps required to solve the maze but struggles during the initial learning phases.

AC and PPO exhibit higher fluctuations in their performance metrics, indicating the need for further optimization to achieve better consistency and efficiency. These algorithms show potential but require more refinement to handle the complexities of maze navigation effectively.

Ultimately, this analysis aids in selecting the most suitable reinforcement learning algorithm based on specific task requirements and environmental complexities. It enhances our understanding of their practical applications and helps in optimizing learning outcomes for real-world scenarios.

13 Implementation of Real-World Control Algorithms

13.1 Introduction to Real-World Implementation

In this section, I delve into the practical application of control algorithms developed through simulations, now being adapted to control a physical robot. This transition is pivotal for evaluating how simulated behaviors translate into real-world scenarios, thereby assessing the effectiveness and limitations of sim-to-real transfer.

13.2 System Overview

The experimental setup uses an ESP32 microcontroller combined with MPU6050 gyroscopic sensors and ultrasonic sensors for distance measurement, connected to a motor control system. These components enable the robot to perform maneuvers like moving forward, turning left, and turning right. The system's architecture is designed to replicate the simulated environment's dynamics.

13.3 Code Architecture and Integration

System Initialization

Understanding the system's initial setup is crucial for ensuring robust and reliable operation. This phase involves preparing the robot by configuring its hardware interfaces, establishing network connectivity, and setting up sensors and actuators.

- WiFi and OTA Configuration: This sets up a network connection and facilitates Over-The-Air (OTA) updates, crucial for remote debugging and iterative improvements.
- Sensor and Display Setup: Activates ultrasonic sensors for distance monitoring and initializes a display to provide real-time feedback on the robot's status and IP address, enhancing user interaction and debugging capabilities.
- MPU6050 Setup and Calibration: Calibrates the gyroscopic sensor for accurate angle measurements, essential for precise navigation.
- Motor Setup: Configures motor drivers and establishes initial motor states, prepping the robot for subsequent movement commands.

Motor Control Mechanism

This subsection elaborates on how movement functions are implemented, translating simulated navigation algorithms into the real-world robotic system.

- Variables for Motor Control

```
int initialSpeed = 125; // Higher initial speed for robust movement
int minSpeed = 40;      // Minimum speed to maintain control
int speed = initialSpeed;
constexpr int TURN_DURATION = 245;
```

These variables dictate the motors' initial and minimum speeds, and the duration for turning, facilitating precise and controlled movements by adjusting the speed dynamically based on the robot's turning angle.

- Forward Movement

  The `move_forward` function initiates rapid forward motion, with real-time checks for obstacles to ensure safe stops—mimicking the real-world need for dynamic responsiveness.

- Left Turn

  The `move_left` function adjusts motor speeds dynamically, a strategy refined in simulations to accommodate physical and inertia effects during turns, ensuring smooth and controlled navigation.

- Right Turn

  The `move_right` function applies similar adjustments and sensor feedback to execute precise right turns. Incorporating `calibrateSensors()` before each movement guarantees accurate gyroscopic data, vital for the precise execution of turns.

- Stopping Movement

  The `stop_moving` function is designed to immediately halt all motions, crucial for accident prevention and adaptation to sudden changes in dynamic environments.

Calibration and Sensor Data Interpretation

Calibration is crucial for ensuring sensor accuracy and reliability, maintaining the integrity of behaviors developed in simulations when applied in real-world settings. The `calibrateSensors` function periodically recalibrates the gyroscopic sensors to correct any data drift or inaccuracies.

```
void calibrateSensors()
{
    long gyroZAccum = 0;
    Serial.println("Calibrating...");
    for (int i = 0; i < 100; i++)
    {
        int16_t ax, ay, az, gx, gy, gz;
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        gyroZAccum += gz;
        delay(20);
    }
    mpu.setZGyroOffset(-gyroZAccum / 13100); // Calibration based on 100 readings
    Serial.println("Calibration Complete");
}
```

13.4 Real-World Application and Limitations

Transitioning from simulated environments to real-world applications introduces unique challenges, particularly in interpreting sensor data and replicating vehicle movements. This section addresses these critical aspects, highlighting both the potential benefits and limitations of applying insights from simulations to actual autonomous vehicle (AV) operations.

13.4.1 Enhanced Sensor-Based Navigation

Refining sensor-based navigation technologies through simulations can significantly enhance the functionality of autonomous vehicles. These technologies are crucial in real-world applications that demand high precision and adaptability. For instance, in busy urban areas or automated delivery systems, dynamic and accurate navigation is essential for boosting both safety and efficiency. By integrating insights from simulations, sensor-based navigation systems can be better tuned to handle the complex and ever-changing conditions of the real world.

13.4.2 Informing Autonomous Vehicle Movement

Simulated environments offer controlled settings to study vehicle dynamics and movement responses. Applying these insights to autonomous vehicle development can lead to advanced algorithms capable of managing the unpredictable nature of real-world environments. This knowledge is essential for improving the ability of autonomous systems to safely and efficiently navigate through dynamic and often chaotic traffic conditions, thereby enhancing the overall functionality of autonomous transportation.

13.4.3 Limitations

Discrepancies in Sensor Data Interpretation: A major challenge in real-world application is the variation in sensor data accuracy between simulated and actual environments. These discrepancies can directly impact the effectiveness of navigational algorithms, potentially compromising the vehicle's decision-making processes, safety, and operational efficiency.

Insufficient Data for Exact Positioning: One critical limitation encountered was the inability of sensors to provide precise data for determining the car's exact position within the real-world grid. In simulations, sensors provided precise and reliable data, enabling accurate positioning and movement control. However, in real-world conditions, sensor data was often noisy, incomplete, and affected by various environmental factors.

- Limited Sensor Resolution: Sensors like ultrasonic distance sensors had limited resolution and range, impacting their ability to detect fine details necessary for precise positioning.
- Environmental Noise: Real-world environments introduced noise from various sources, such as electronic devices, surface irregularities, and ambient conditions, which interfered with sensor readings.
- Dynamic Obstacles: Unlike controlled simulations, real-world environments had dynamic obstacles that sensors could not always accurately detect or interpret, leading to positioning errors.

These limitations meant that the vehicle often had a rough estimate of its position rather than a precise one, hindering its ability to navigate accurately within a defined grid. Practically, the vehicle primarily relied on distance measurements to avoid crashes and verify its movement capabilities. However, it lacked the ability to determine whether it was centered within the path, which was a significant drawback in maze navigation.

13.4.4 Challenges in Movement Replication

Replicating vehicle movements from simulations in the real world is challenging due to various factors such as different road surfaces, weather conditions, vehicle weight, and mechanical issues. These factors can cause unexpected changes in vehicle behavior. Therefore, algorithms developed in simulations need to be tweaked and recalibrated to work effectively in real-world conditions. For instance, the vehicle's weight can affect acceleration and deceleration, impacting the timing of movement commands. Similarly, the surface the vehicle moves on can affect traction, influencing its ability to turn or stop.

13.4.5 Practical Implementation Considerations

To successfully apply simulation insights to the real world, several practical aspects need to be addressed:

- Sensor Calibration: Adjust sensors to account for environmental influences.
- Algorithm Adjustment: Modify algorithms to fit hardware limitations.
- Handling Unpredictability: Ensure the system can cope with real-world unpredictability.

By addressing these considerations, we can bridge the gap between simulation-based research and real-world applications, enhancing the safety, efficiency, and reliability of autonomous vehicle technologies.

13.5 Conclusion for Real-World Application

Transitioning from simulation-based research to real-world applications in autonomous vehicle navigation presents both challenges and opportunities. While using simulation insights for sensor utilization and vehicle movement can revolutionize autonomous vehicle technologies, there is a significant gap between simulation precision and real-world variability that needs to be bridged.

A significant limitation was the imprecision in sensor data for positioning. Sensors prevented crashes by measuring the distance to obstacles so it is able to decide in which direction it should move. However, this did not try to measure the exact position of the car within the grid.

Additionally, real-world factors such as surface variations, environmental conditions, and dynamic obstacles require continuous adjustments and recalibration of algorithms. Practical considerations, including sensor calibration and algorithm adaptation, are essential for effectively deploying autonomous systems.

Ultimately, overcoming these challenges is key to successfully integrating sim2real technologies. Doing so will improve the safety, efficiency, and reliability of autonomous transportation systems, paving the way for broader adoption and use in various real-world scenarios.

14 Challenges and Solutions in RL Implementation

14.1 Challenge 1: Choosing the Right Virtual Environment

- Description: Picking the best virtual environment for training the RC car.
- Solution: I chose OpenAI Gym because it's simple, familiar from previous coursework, and focuses on reinforcement learning.

14.2 Challenge 2: Selecting the Optimal Reinforcement Learning Technique

- Description: Finding the most effective RL technique for training the virtual RC car.
- Solution: After testing various methods, the Double Deep Q-Network (DDQN) proved to be the best, consistently solving the maze with fewer steps and episodes than other techniques.

14.3 Challenge 3: Addressing Movement Discrepancies in Sim2Real Transfer

- Description: Bridging the gap between how the RC car moves in simulations and in the real world.
- Solution Attempt: I fine-tuned the action command frequency using an async method, waited for the motor to finish moving, and considered a queued action system. Getting precise movement in the real world turned out to be more critical than in simulations.

14.4 Challenge 4: Alignment Issues and Motor Encoder Implementation

- Description: Ensuring the RC car moves in a straight line, as there was a persistent ~3-degree offset.
- Solution Attempts:
  - Attempt 1: Used motor encoders to improve accuracy but faced precision limits.
  - Attempt 2: Switched to a more powerful motor, but the added weight brought back the alignment issue.
  - Attempt 3: Added an MPU6050 gyroscope to measure and adjust orientation, which initially helped with 90-degree turns but didn't fix the offset.
  - Attempt 4: Removed the Raspberry Pi and used only the ESP32 for all controls, resulting in a lighter, more precise robot, though it still struggled with consistent 90-degree turns.

14.5 Challenge 5: Ensuring Consistent and Effective Training

- Description: Maximizing training efficiency and performance while keeping things consistent between simulations and real-world scenarios.
- Solution: Training in a simulation was much more efficient due to the difficulties of resetting the RC car, dealing with manual interferences, and limited battery life.

14.6 Challenge 6: Accurate Sensor Data Normalization for Sim2Real Transfer

- Description: Aligning sensor data between simulated and real-world environments for accurate model performance.
- Solution: Implemented functions to ensure real-world sensor data matched the training data.

– Real-World Sensor Data Normalization:

$$\text{map\_distance}(d) = \begin{cases} d & \text{if } d < 25 \\ 25 + (d - 25) \times 0.5 & \text{otherwise} \end{cases}$$

– Simulation Sensor Data Normalization:

$$\text{normalize\_distance}(d) = \max\left(0, \min\left(\frac{d}{\text{sensor\_max\_range}}, 1\right)\right) \times 1000$$

14.7 Challenge 7: Integration of Failsafe Mechanisms

- Description: Preventing collisions and ensuring safe navigation in the real world.
- Solution: Developed a failsafe system to prevent unwanted forward movement and retrained the model with this feature, which solved the issue of the robot driving into walls and getting stuck.

14.8 Challenge 8: Training Environment and Technique Efficacy

- Description: Finding the most effective environment and RL technique for training.
- Solution: The DDQN technique was more efficient than DQN, Q-agent, PPO, and ActorCritic approaches, highlighting the importance of selecting the right technique.

14.9 Conclusion for Challenges and Solutions

This section outlines the practical challenges encountered while applying reinforcement learning (RL) techniques to autonomous RC cars. My journey began with selecting OpenAI Gym as the virtual environment due to its simplicity and relevance to RL. The Double Deep Q-Network (DDQN) emerged as the most effective RL technique for navigating complex environments.

However like discussed before , transitioning from simulations to real-world applications revealed significant discrepancies, particularly in movement control and sensor data alignment. I explored various solutions like motor encoders, power adjustments, and gyroscope integration, which partially addressed these issues. Efforts to normalize sensor data and implement failsafe mechanisms also contributed to better alignment with real-world conditions.

A significant advancement was achieved by simplifying the robot's design to use only the ESP32 module, making it lighter and more precise. This change marked a considerable step in overcoming the previous challenges.

Although I made a lot of progress in addressing these challenges, there is still room for improvement in achieving seamless sim-to-real transfer and ensuring consistent performance across different environments.

15 Integration of Practical Experiments

Throughout my research, I used various practical experiments to solve the challenges I encountered. These experiments, documented through video demonstrations, provide clear insights into my problem-solving process.

15.1 Addressing Alignment and Orientation Challenges

One of the main challenges was ensuring the RC-car's precise orientation and alignment during movement. To address this, I used the MPU6050 gyroscope to correct alignment issues and achieve accurate 90-degree turns. My efforts focused on using the gyroscope to maintain and correct the car's orientation, crucial for navigating complex mazes with high precision.

15.1.1 Using the MPU6050 Gyroscope for Precise Orientation

- Experiment E1 - Gyroscope Calibration: Testing the MPU6050 gyroscope to correct the car's orientation for accurate navigation, aiming to improve control over the vehicle's movement through maze environments (see Video E1 in the Video References section).

- Experiment E2 - Navigational Corrections: Addressing alignment issues for precise 90-degree turns and realigning the car's forward movement to fix a persistent ~3-degree offset (see Video E2 in the Video References section).

15.2 Improving Movement Precision with Encoders

To enhance the RC-car's movement precision, I experimented with rotary encoders. These devices, which accurately measure wheel rotations, were essential for improving straight-line movements and addressing hardware reliability challenges in real-world applications.

- Experiment E6 - Encoder Implementation: Adding rotary encoders to the setup to gain more precise control over the car's movements by accurately measuring wheel rotations, thus refining the vehicle's navigation capabilities (see Video E6 in the Video References section).
- Experiment E7 - Troubleshooting Encoder Malfunction: Addressing a malfunction with one of the encoders that halted further tests, highlighting the practical challenges of maintaining hardware reliability (see Video E7 in the Video References section).

15.3 Real-World Application Tests

Moving beyond controlled environments, I tested the RC-car in both outdoor and indoor settings to evaluate its performance in real-world conditions. These tests were crucial for assessing the practical application of my research findings and understanding the challenge of accurately translating simulation models to real-world applications.

- Experiment E9 - Outdoor Navigation Test: Navigating the RC-car on uneven outdoor surfaces, where variations greatly affected performance, underscoring the importance of environmental factors in autonomous navigation (see Video E9 in the Video References section).
- Experiment E11 - Indoor Controlled Test: Conducting controlled indoor tests to closely monitor and adjust the RC-car's navigation strategies, reflecting on the complexities of sim-to-real transfer (see Video E11 in the Video References section).

16 Reflection

Reflecting on this research project, I have gained invaluable insights that have significantly influenced my approach and perspective.

16.1 Openness to New Ideas

One of the most important lessons I learned is the value of being open to new ideas. Embracing unfamiliar concepts and stepping out of my comfort zone were crucial for fostering creativity and discovering innovative solutions. For instance, the transition from simulation to real-world testing required novel problem-solving approaches to address discrepancies in sensor data and movement control. This openness not only enriched my technical skills but also enhanced my ability to adapt to new challenges and environments.

16.2 Bridging Theory and Practice

Transitioning from theory to practice proved more challenging than I expected. While controlled virtual environments were manageable, the real world presented unforeseen complexities. This experience sharpened my ability to balance theoretical knowledge with practical adaptability. For example, integrating sensor calibration and adjusting control algorithms were essential steps that highlighted the gap between simulated and real-world conditions. This hands-on experience has prepared me to handle the unpredictability of real-world applications more effectively. Additionally, it underscored the importance of iterative testing and continuous learning in the development process.

16.3 Anticipatory Thinking

Overcoming various obstacles highlighted the importance of anticipatory thinking. Navigating challenges and adapting to evolving conditions made me realize the need to think ahead. This proactive mindset has become integral to my approach, turning potential challenges into opportunities for innovation. For instance, anticipating sensor noise and environmental variability led to the development of robust policies and fine-tuning strategies that improved the system's performance in real-world scenarios. This foresight is crucial for anticipating and mitigating risks, ensuring the robustness and reliability of autonomous systems.

16.4 Policy and Regulation

Engaging with policy and regulatory frameworks was enlightening. It highlighted the delicate balance between fostering innovation and ensuring public safety. Collaborating with policymakers and industry leaders taught me the importance of creating regulations that encourage technological advancement while maintaining accountability. This experience underscored the need for ethical considerations and compliance in the deployment of autonomous systems. Understanding these regulatory landscapes will be essential as I continue to develop and implement AI and robotics solutions in the future.

16.5 Feedback from the Jury

The feedback from the jury was predominantly positive, and their insights were incredibly valuable. They commended the practical aspects of my project, particularly the real-world testing of the RC car. Specific suggestions, such as increasing the distance to the walls to reduce navigation errors and integrating a camera system for real-time visual feedback, were implemented to enhance the accuracy and reliability of the navigation algorithms. This feedback loop was instrumental in refining my approach and improving the overall quality of the project.

16.6 Methodological Insights

Creating the custom maze navigation environment (RCMazeEnv) and using the DDQN architecture provided significant insights into the complexities of sim-to-real transfer. The iterative process of testing, calibration, and adjustment refined the system and improved its performance in navigating the physical maze. Designing an effective reward function was key to guiding the agent's learning process, balancing immediate rewards and penalties for inefficient actions to optimize performance. This methodological rigor will serve as a foundation for future projects, ensuring a structured and systematic approach to research and development.

16.7 Societal Implications of Autonomous Systems

The potential applications of autonomous systems extend far beyond maze navigation. Insights from this research could contribute to developing autonomous vehicles, drones, and other robotic systems navigating complex environments. Ensuring these technologies are accessible and beneficial to society requires careful consideration of ethical, regulatory, and practical factors. For instance, autonomous systems must be designed to prioritize safety and reliability, minimizing risks to human users and the environment. Moreover, the deployment of such technologies should be accompanied by public education and engagement to build trust and understanding among stakeholders.

16.8 Future Directions

This project can be further developed by integrating more complex sensory systems, applying more sophisticated machine learning models, extending testing environments to include varying conditions, and fostering community collaboration. These enhancements could improve the technology and advance autonomous vehicle research and development. Future research could explore the use of multi-modal sensory inputs, such as combining visual, auditory, and tactile data to enhance the agent's perception and decision-making capabilities. Additionally, collaborations with other researchers and industry partners could provide valuable insights and resources for further advancements.

16.9 Personal Growth and Future Aspirations

This research project has been transformative, shaping my understanding of AI and robotics and guiding my future aspirations. The lessons learned, from technical challenges to broader societal implications, have equipped me with the skills and knowledge to pursue further research and innovation in this field. I am particularly interested in exploring the ethical dimensions of AI and robotics, ensuring that these technologies are developed and deployed in ways that are fair, transparent, and accountable. My future aspirations include contributing to the development of autonomous systems that can positively impact various sectors, from healthcare to transportation, enhancing efficiency and safety.

16.10 Commitment to Innovation and Societal Stewardship

As I move forward, my ethos will be defined by adaptability, responsiveness, and a commitment to societal stewardship. The lessons learned have prepared me for the next stage of my journey, where I will continue to engage with data, refine methodologies, and embrace future challenges. I am committed to fostering a culture of innovation and continuous improvement, where creative problem-solving and interdisciplinary collaboration are encouraged. Furthermore, I will strive to ensure that my work contributes to the greater good, addressing societal challenges and promoting sustainability.

16.11 Personal Note

Even though this was an incredible project to work on and I loved every second of it, I can't help but feel like I limited myself by not actively researching before starting this project. I went into this (and many other projects before this) headfirst, starting to make a maze in Pygame even before the three-week period began. However, by doing this, I limited myself by not exploring other methods of completing this maze, like giving the agents more controls than just moving forward, left, and right. For example, as Wouter suggested during one of our evaluations, the use of a camera to see where the car is in the maze. This, along with giving the agent the ability to move the left and right motors completely independent of each other, would have given the agent a lot more control over the car and maybe even eliminated the need for the many corrections I had to make while transferring the model to the real world.

I doubt I would have been able to get as far as fast as I did if I had taken this alternative route, but I do think it would have been a lot more interesting and maybe even more rewarding as I don't think this project is completely finished yet, and I look forward to continuing this journey.

17 Advice

17.1 Practical Utilization of Simulations

Simulations are invaluable in research, offering a risk-free, controllable environment for developing and refining algorithms.

- Cost-Effectiveness: Simulations allow for substantial cost savings by reducing the need for physical prototypes and extensive real-world trials in the early phases of research.

17.2 Strategies for Effective Transition from Simulation to Reality

Successfully transitioning from simulations to real-world applications is critical for validating the effectiveness of research outcomes.

- Incremental Testing: Start with simulations to refine algorithms, then gradually introduce real-world testing to confirm results and adapt to environmental variables.
- Feedback Loops: Use continuous feedback mechanisms to enhance simulation models with insights gained from real-world tests, improving their accuracy and applicability.

17.3 Overcoming Common Challenges in Simulation-to-Reality Transitions

Bridging the gap between simulations and actual conditions often requires specific adjustments, especially in terms of sensor data and mechanical operations.

- Sensor Discrepancy Adjustments: Regular calibration of real-world sensors is essential to ensure they align with simulation inputs.
- Movement and Mechanics Alignment: It's crucial to synchronize physical movements and mechanics with those anticipated by simulations to ensure smooth transitions.

17.4 Insights from My Research

- Simulation Platforms: Choosing the right simulation platform, like OpenAI Gym, is critical and may necessitate additional tools for more complex scenarios.
- DDQN Superiority: My findings show that Double Deep Q-Network (DDQN) surpasses other models, such as DQN and PPO, by reducing overestimations and enhancing learning stability.

17.5 Methodological Advice

- Comprehensive Evaluation: Utilize a mix of qualitative and quantitative approaches to thoroughly evaluate the effectiveness of both simulations and real-world applications.
- Adaptive Techniques: Stay flexible and responsive to the results and feedback, which are crucial for effectively tackling unexpected challenges.

17.6 Practical Experiment Integration

- Prototyping and Iteration: Use iterative design and prototyping to progressively refine systems, effectively linking theoretical research and practical implementation.
- Continuous Feedback: Continuously seek and integrate feedback from stakeholders and peers to improve simulation models and real-world applications.

17.7 Guidelines for Future Research

17.7.1 Introduction for Future Research

This chapter provides a comprehensive methodology and advice for researchers involved in simulation-based studies, focusing on ensuring a smooth transition from theoretical models to practical applications.

17.7.2 Step-by-Step Plan

Step 1: Selection of Simulation Environments

- Research and Evaluation: Investigate and evaluate available simulation tools that fit your study, such as OpenAI Gym, Unity 3D, and CARLA.
- Criteria Development: Establish criteria based on fidelity, scalability, ease of use, and integration capabilities with your existing systems.
- Preliminary Testing: Perform initial tests to evaluate these environments against your criteria, focusing on how well they replicate real-world conditions and support your research objectives.

Step 2: Managing Expectations and Adaptability

- Expectation Setting: Set realistic expectations for what simulations can achieve. Understand that simulations can provide valuable insights but may not capture all real-world nuances.
- Adaptation Strategies: Be ready to adjust your research approach based on the outcomes of simulations. Address any data discrepancies and be flexible in modifying your methodologies as needed.

Step 3: Methodology Flexibility

- Continuous Evaluation: Regularly assess the effectiveness of your chosen methodologies. Stay open to feedback and willing to make iterative improvements.
- Integration of New Technologies: Keep an eye on emerging technologies and be prepared to incorporate them into your research to enhance your methods and results.

Step 4: Addressing Real-World Constraints

- Practical Implementation: Consider the practical constraints you'll face when applying your simulation results to the real world. These might include hardware limitations, environmental variability, and regulatory requirements.
- Pilot Testing: Conduct small-scale real-world tests to validate your simulation findings. Use these tests to identify and address any gaps between the simulated and real-world performance.

Step 5: Enhancing Data Accuracy

- Sensor Calibration: Ensure your sensors are properly calibrated both in simulations and real-world tests. Accurate sensor data is crucial for reliable results.
- Data Normalization: Implement robust data normalization techniques to align simulated sensor data with real-world conditions. This helps in maintaining the consistency and reliability of your data.

Step 6: Fostering Collaboration

- Interdisciplinary Teams: Collaborate with experts from different fields to enrich your research. For example, working with specialists in robotics, computer vision, and AI can provide new insights and approaches.
- Community Engagement: Engage with the research community through conferences, workshops, and publications to share your findings and gain valuable feedback.

Step 7: Future-Proofing Your Research

- Scalability: Design your research to be scalable, so it can be adapted to larger and more complex problems in the future.
- Long-Term Vision: Keep a long-term vision for your research, anticipating future challenges and opportunities in the field of simulation-to-reality transfer.

17.7.3 Conclusion for Future Research

Following these guidelines will help researchers navigate the complexities of simulation-based studies and ensure a successful transition from theoretical models to practical applications. By being methodical, adaptable, and collaborative, you can overcome the challenges of sim2real transfer and contribute valuable insights to the field of autonomous systems and beyond.

18 Sources of Inspiration and Conceptual Framework

The inspiration for this research comes from a mix of technical documentation, digital platforms, and academic literature. Key influences include the challenges of micro mouse competitions and the potential of reinforcement learning (RL) to navigate complex mazes. My interest was further sparked by dynamic RL applications in autonomous vehicle control showcased on YouTube and GitHub, alongside influential academic research.

18.1 Micro Mouse Competitions and Reinforcement Learning

Micro mouse competitions, where small robotic mice navigate mazes, served as a major inspiration. The use of RL in these competitions showed the potential for solving real-world problems and controlling autonomous systems. Insights from a Medium article by M. A. Dharmasiri on maze traversal algorithms and shortest path strategies provided practical algorithmic approaches relevant to this study [15].

18.2 Influential YouTube Demonstrations and GitHub Insights

YouTube videos like "Self Driving and Drifting RC Car using Reinforcement Learning" [11] and "Reinforcement Learning with Multi-Fidelity Simulators – RC Car" [16] vividly demonstrated RL's real-world applicability and the feasibility of sim-to-real transfer. GitHub repositories, such as the "Sim2Real_autonomous_vehicle" project [13], detailed the practical steps and challenges of implementing RL in physical systems.

18.3 Technical Exploration and Academic Foundation

Academic articles also significantly shaped this research. Notable works include Q. Song et al.'s article on autonomous driving decision control [12] and a survey by W. Zhao, J. P. Queralta, and T. Westerlund on sim-to-real transfer in deep RL for robotics [17]. These articles provided in-depth methodologies and highlighted the challenges of applying RL to autonomous systems.

18.4 Synthesis and Research Direction

These diverse sources collectively informed the direction of this research, focusing on the feasibility and complexities of sim-to-real transfer in autonomous navigation. The goal is to combine insights from both digital and academic realms to address the challenges of applying advanced RL models in real-world scenarios.

19 General Conclusion

This thesis has demonstrated the potential of transferring a trained reinforcement learning (RL) agent from a simulated environment to a real-world setting, focusing on navigating a maze using a remote-controlled (RC) car. The detailed experiments and analyses provide a thorough exploration of this transition.

The research shows that such a transfer is not only possible but also comes with significant challenges. The experiments highlighted in Chapter 14: Challenges and Solutions in RL Implementation emphasize the importance of normalizing sensor data and adapting control algorithms to handle the unpredictable dynamics of the real world. These adaptations were essential for aligning the simulated models with the real-world scenarios encountered during implementation.

The choice of appropriate virtual environments and reinforcement learning techniques, as discussed in Chapter 6: Methodology, was crucial in shaping the experimental approach and ensuring effective simulation training. The Double Deep Q-Network (DDQN) proved to be the most suitable technique, providing a robust framework for navigating the complexities of practical applications.

This study confirms the feasibility of sim-to-real transfers and offers a detailed examination of the intricate mechanics involved in this process. This area is of growing importance in AI and robotics research. By integrating theoretical insights with practical applications, this thesis significantly contributes to the ongoing discussion on the viability and challenges of applying reinforcement learning in real-world scenarios.

In conclusion, while transitioning a trained RL agent from simulation to a real environment is feasible, the process requires careful planning, adaptability, and continual refinement. The challenges highlighted throughout this research underscore the need for ongoing efforts to enhance the robustness and reliability of sim-to-real applications, ensuring they can meet the demands of real-world conditions.

20 Guest Speakers

20.1 Innovations and Best Practices in AI Projects by Jeroen Boeye at Faktion

Jeroen Boeye, representing Faktion, shared valuable insights into the synergy between software engineering and artificial intelligence in developing AI solutions. He emphasized the importance of not only focusing on AI technology but also integrating solid software engineering principles to create robust, scalable, and maintainable AI systems. This holistic approach ensures that AI solutions are both technically sound and viable for long-term application.

During his lecture, Jeroen highlighted various aspects of AI application, particularly Chatlayer's contributions to conversational AI. He explained how Chatlayer enhances chatbot interactions through sophisticated conversational flows, improving the accuracy and relevance of exchanges with users. Another key point was Metamaze, which he praised for its innovative methods in automating document processing, creating concise summaries from extensive documents and emails, and demonstrating the capabilities of supervised machine learning in administrative tasks.

Jeroen outlined a clear roadmap for successful AI project implementation, stressing the need to validate business cases and adopt a problem-first strategy. He discussed the crucial role of high-quality data as the foundation for any AI endeavor and offered strategies for creatively overcoming data limitations. The talk also covered the importance of viewing failures as opportunities for innovation and maintaining open communication with stakeholders about challenges and setbacks.

The lecture further presented various practical AI applications across different industries, such as solar panel detection, unauthorized pool identification, air freight container inspection, and early warning systems for wind turbine gearboxes. Jeroen demonstrated how AI could tackle complex challenges through innovative data sourcing, synthetic data generation, and anomaly detection techniques. He also explored case studies on energy analysis in brick ovens and egg incubation processes, emphasizing the importance of data preprocessing and machine learning models in improving efficiency and outcomes.

Key points from Jeroen's talk included mastering data preprocessing and treating data as a dynamic asset to better tailor AI models to specific needs. He shared practical tips on enhancing operational efficiency, such as using host mounts for code integration and Streamlit for dashboard creation, to streamline development processes.

In summary, Jeroen Boeye's lecture offered a comprehensive perspective on integrating AI technologies in real-world settings. His insights into the vital role of software engineering principles, alongside a deep understanding of AI capabilities and constraints, provided valuable guidance for developing effective and sustainable AI solutions. The lecture not only underscored current AI trends and future directions but also shared practical knowledge on navigating the complexities of AI project execution.

20.2 Pioneering AI Solutions at Noest by Toon Vanhoutte

Toon Vanhoutte, speaking on behalf of Noest from the Cronos Group, delivered an engaging lecture on the effective integration of artificial intelligence and software engineering in developing cutting-edge business solutions. With a dedicated team of 56 local experts, Noest has built a reputation for its pragmatic approach to projects, targeting global impact while valuing craftsmanship, partnership, and enjoyment as core principles. This philosophy extends to their diverse services, which include application development, cloud computing, data analytics, AI innovations, low-code platforms, ERP solutions, and comprehensive system integrations, all supported by a strong partnership with Microsoft.

Toon presented a case study on a packaging company that aimed to revolutionize image search capabilities based on product labels. The project faced various challenges, such as inconsistent PDF formats and large file sizes, which were adeptly managed using Azure Blob Storage for data handling and event-driven processing strategies for efficient, cost-effective solutions, showcasing Noest's skill in utilizing cloud technologies to address complex issues.

Another significant challenge was enhancing image searchability, which involved recognizing text and objects within images. This was tackled using Azure AI Search, supplemented by Large Language Models (LLMs) and vector search techniques. This approach allowed for nuanced search functionalities beyond simple text queries, demonstrating the advanced capabilities of AI in interpreting complex data.

Toon also explored advancements in semantic search, discussing how different search methods—keyword, vector, and hybrid—along with semantic ranking, could significantly improve the accuracy

and contextuality of search results. Practical demonstrations, including comparisons between OCR and GPT-4 vision, illustrated the potential of AI to offer deeper insights based on semantic understanding.

A key takeaway from Toon's lecture was the importance of setting realistic client expectations regarding AI's capabilities and potential inaccuracies, highlighting the experimental nature of these technologies. The discussion on AI's evolving landscape emphasized the need for prompt engineering, the challenges of navigating a developing field, and the importance of client education in managing expectations about AI technologies like GPT.

In conclusion, Toon Vanhoutte's presentation not only highlighted Noest's innovative work in AI and software engineering but also imparted crucial lessons on innovation, adaptable problem-solving, and the necessity for ongoing learning in the dynamic field of AI. This presentation showcased Noest's commitment to pushing technological boundaries to create impactful, pragmatic solutions that fully utilize AI's potential.

21 Installation Steps

This section outlines the required steps to install and set up the project environment. Following these instructions will ensure the successful deployment of the autonomous navigation system.

21.1 Prerequisites

Before starting the setup process, make sure you have the following:

- Git: For cloning the project repository.
- Docker: To containerize the web application and ensure a consistent runtime environment.
- Python 3.11 and pip: If you prefer running the project without Docker, Use python along with the dependencies listed in `/web_app/web/requirements.txt` to get the project running.

21.2 Repository Setup

To clone the repository and navigate to the project directory, use these commands:

```
git clone https://github.com/driessenslucas/researchproject.git
cd researchproject
```

21.3 Hardware Setup and Assembly

21.3.1 Introduction to Hardware Components

Here's an overview of the hardware components used in the project, including the RC car, sensors, and microcontrollers. Proper integration of these components is essential for the autonomous navigation system to function correctly.

21.4 Components List

21.4.1 Core Components

- ESP32-WROOM-32 module
  - Available at Amazon.com
- 3D printed parts
  - Available at Thingiverse.com
    - HC-SR04 holders: `https://www.thingiverse.com/thing:3436448/files`
    - Top plate + alternative for the robot kit: `https://www.thingiverse.com/thing:2544002`
- Motor Controller (L298N)
  - Available at DFRobot.com
- 2WD miniQ Robot Chassist
  - Available at DFRobot.com
- Mini OLED screen
  - Available at Amazon.com
- Sensors (HC-SR04 and MPU6050)
  - Available at Amazon.com
- 18650 Battery Shield for ESP32
  - Available at Amazon.com

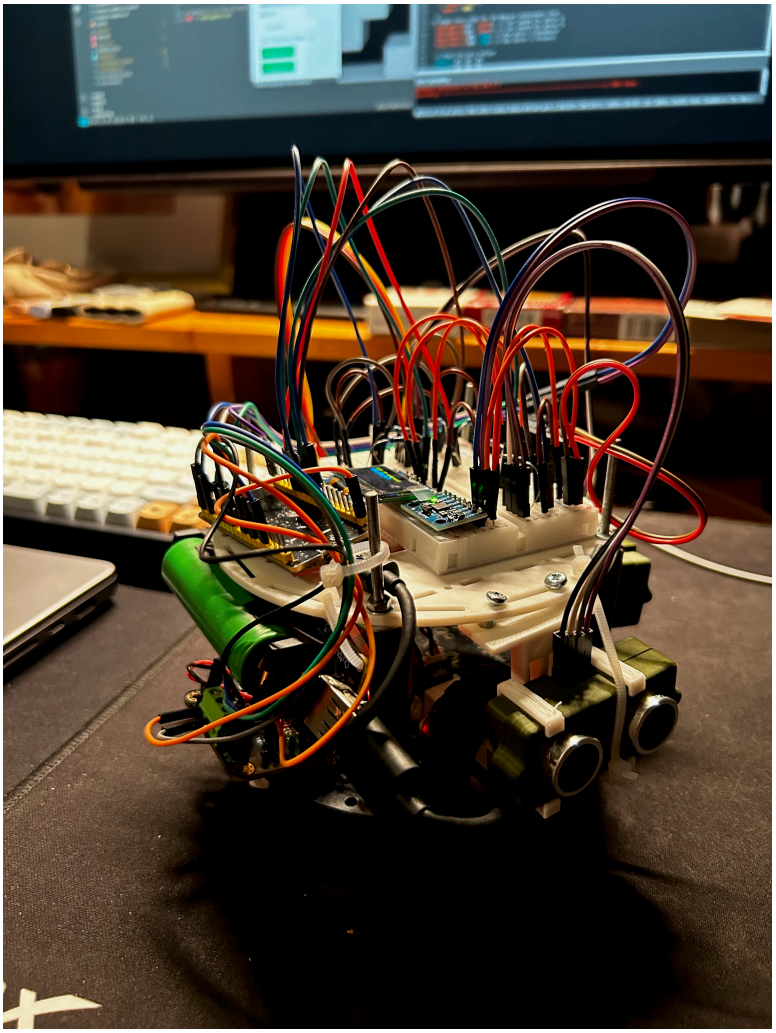


Figure 24: Final RC Car (Image created by author)

21.4.2 Supplementary Materials

- Screws, wires, and tools required for assembly
  - 4mm thick screws, 5mm long to hold the wood together
    - Available at most hardware stores
  - M3 bolts & nuts
    - Available at most hardware stores
  - Wood for the maze
    - Available at most hardware stores

21.4.3 Wiring Guide

In this section, you find how the components are connected to the ESP32 microcontroller. Proper wiring is essential for the system to function correctly.

The pin connections for the ESP32 microcontroller are as follows:

```
int E1 = 2; //PWM motor 1
int M1 = 17; //GPIO motor 1
int E2 = 19; //PWM motor 2
int M2 = 4; //GPIO motor 2

int sensor0Trig = 27; //GPIO right sensor
int sensor0Echo = 26; //GPIO right sensor

int sensor1Trig = 33; //GPIO left sensor
int sensor1Echo = 32; //GPIO left sensor

int sensor2Trig = 25; //GPIO front sensor
int sensor2Echo = 35; //GPIO front sensor

// OLED display and MPU6050 pins
#define SDA_PIN 21 // this is the default sda pin on the esp32
#define SCL_PIN 22 // this is the default scl pin on the esp32
```

ESP32 Wiring:

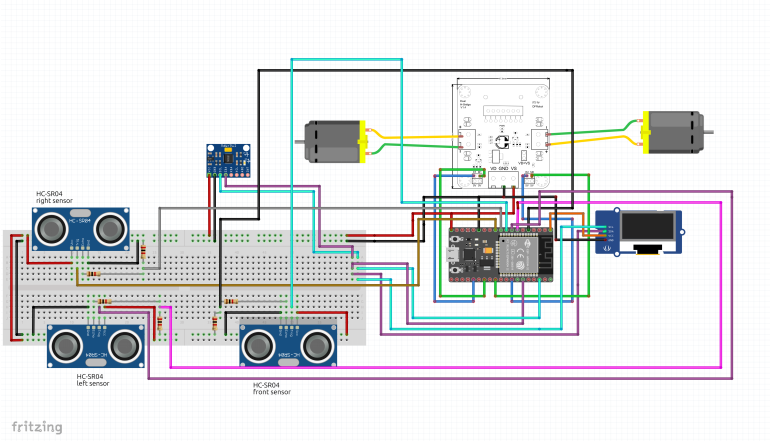


Figure 25: ESP32 Wiring (Image created by author)

In the image above, you can see the wiring connections for the ESP32 microcontroller. The pins are connected to the motor driver, sensors, OLED display, and MPU6050 gyroscope.

21.4.4 Software Configuration

1. Arduino IDE Setup: Install the Arduino IDE to program the ESP32 microcontroller. Follow Espressif Systems' instructions to add the ESP32 board to the Arduino IDE.
2. Library Installation: Install the ESP32_SSD1306 library for the OLED display functionality.
3. Code Upload: Transfer the scripts from the esp32 folder to the ESP32 device. Modify the WiFi settings in the script to match your local network configuration.

21.5 Web Application Setup

21.5.1 Note

To ensure a smooth setup of the virtual display, it's recommended to run `docker-compose down` after each session.

21.5.2 Steps

1. Navigate to the web application's source code directory:

   ```
   cd ./web_app/
   ```

2. Launch the Docker containers with:

   ```
   docker-compose up -d
   ```

21.6 Usage Instructions

1. Open your web browser and go to `http://localhost:8500` or `http://localhost:5000`.
2. Enter the ESP32's IP address in the web app and select the desired model for deployment.
3. You can also run a virtual demonstration without engaging the physical vehicle.
4. Start the maze navigation by clicking the `Start Maze` button.

A demonstration of the project is available (see Web App Demo in the Video References section).

21.7 Additional Information: Model Training

- You can use a pre-trained model or train a new model using the script in train.
- This training script is optimized for efficiency and can be run directly on the Raspberry Pi.
- After training, you will be prompted to save the new model. If saved, it will be stored in the models directory of the `web_app` folder.

By following these steps, you can successfully set up and deploy the autonomous navigation system, ensuring it runs smoothly both in simulations and real-world scenarios.

22 Video References

1. Video E1 - Gyroscope Calibration: Testing the MPU6050 gyroscope's ability to correct the car's orientation for accurate navigation, aiming to refine control over the vehicle's movement through maze environments.
   - Click here to go to the video: Video E1

Figure 26: QR code for video E1. (Video by author.)

2. Video E2 - Navigational Corrections: Addressing alignment issues when attempting precise 90-degree turns and realigning the car's forward movement to rectify a persistent ~3-degree offset.
   - Video E2

Figure 27: QR code for video E2. (Video by author.)

3. Video E6 - Encoder Implementation: Introducing rotary encoders to the setup, hoping to gain more precise control over the car's movements by accurately measuring wheel rotations, thus refining the vehicle's navigation capabilities.
   - Click here to go to the video: Video E6

Figure 28: QR code for video E6. (Video by author.)

4. Video E7 - Troubleshooting Encoder Malfunction: Addressing a malfunction with one of the encoders that halted further tests, highlighting the practical challenges of maintaining hardware reliability.
   - Click here to go to the video: Video E7
5. Video E9 - Outdoor Navigation Test: Navigating the RC-car on uneven outdoor surfaces, where variations greatly affected performance, underscoring the importance of environmental factors in autonomous navigation.
   - Click here to go to the video: Video E9
6. Video E11 - Indoor Controlled Test: Conducting controlled indoor tests to closely monitor and adjust the RC-car's navigation strategies, reflecting on the complexities of sim-to-real transfer.
   - Click here to go to the video: Video E11

Figure 29: QR code for video E7. (Video by author.)

Figure 30: QR code for video E9. (Video by author.)

7. Web App Demo: A demonstration of the web application's functionality, showcasing the user interface and the autonomous navigation system's control features.
   - Click here to go to the video: Web App Demo
8. DDQN Simulation test: A simulation test of the DDQN model navigating a maze environment, demonstrating the model's learning capabilities and decision-making processes.
   - Click here to go to the video: DDQN Simulation

Figure 31: QR code for video E11. (Video by author.)

Figure 32: QR code for Web App Demo. (Video by author.)

Figure 33: QR code for DDQN Simulation. (Video by author.)

23 References

[1] G. Brockman et al., "OpenAI Gym," arXiv preprint arXiv:1606.01540, 2016.

[2] A. Dosovitskiy et al., "CARLA: An Open Urban Driving Simulator," in Proceedings of the 1st Annual Conference on Robot Learning, 2017.

[3] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," in Proceedings of the AAAI Conference on Artificial Intelligence, 2016.

[4] J. Schulman et al., "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347, 2017.

[5] J. Tobin et al., "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.

[6] K. Bousmalis et al., "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping," in IEEE International Conference on Robotics and Automation (ICRA), 2018.

[7] Y. Pan and Q. Yang, "A Survey on Transfer Learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, Oct. 2010.

[8] A. A. Rusu et al., "Sim-to-Real Robot Learning from Pixels with Progressive Nets," in Proceedings of the Conference on Robot Learning, 2016.

[9] S. James et al., "Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks," in Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), 2019.

[10] F. Sadeghi and S. Levine, "(CAD)^2RL: Real Single-Image Flight without a Single Real Image," in Proceedings of Robotics: Science and Systems, 2016.

[11] "Self Driving and Drifting RC Car using Reinforcement Learning," YouTube, Aug. 19, 2019. [Online Video]. Available: `https://www.youtube.com/watch?v=U0-Jswwf0hw`. [Accessed: Jan. 29, 2024].

[12] Q. Song et al., "Autonomous Driving Decision Control Based on Improved Proximal Policy Optimization Algorithm," Applied Sciences, vol. 13, no. 11, Art. no. 11, Jan. 2023. [Online]. Available: `https://www.mdpi.com/2076-3417/13/11/6400`. [Accessed: Jan. 29, 2024].

[13] DailyL, "Sim2Real_autonomous_vehicle," GitHub repository, Nov. 14, 2023. [Online]. Available: `https://github.com/DailyL/Sim2Real_autonomous_vehicle`. [Accessed: Jan. 29, 2024].

[14] "OpenGL inside Docker containers, this is how I did it," Reddit, r/docker. [Online]. Available: `https://www.reddit.com/r/docker/comments/8d3qox/opengl_inside_docker_containers_this_is_how_i_did/`. [Accessed: Jan. 29, 2024].

[15] M. A. Dharmasiri, "Micromouse from scratch | Algorithm- Maze traversal | Shortest path | Floodfill," Medium, [Online]. Available: `https://medium.com/@minikiraniamayadharmasiri/micromouse-from-scratch-algorithm-maze-traversal-shortest-path-floodfill-741242e8510`. [Accessed: Jan. 29, 2024].

[16] "Reinforcement Learning with Multi-Fidelity Simulators – RC Car," YouTube, Dec. 30, 2014.

[Online Video]. Available: https://www.youtube.com/watch?v=c_d0Is3bxXA. [Accessed: Jan. 29, 2024].

[17] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey," in 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Dec. 2020, pp. 737–744. [Online]. Available: https://arxiv.org/pdf/2009.13303.pdf.

[18] R. S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA: The MIT Press, 2018.

[19] H. van Hasselt, A. Guez, D. Silver, et al., "Deep Reinforcement Learning with Double Q-learning," arXiv preprint arXiv:1509.06461, 2015.

[20] Papers With Code, "Double DQN Explained," [Online]. Available: https://paperswithcode.com/method/double-dqn.

[21] D. Jayakody, "Double Deep Q-Networks (DDQN) - A Quick Intro (with Code)," 2020. [Online]. Available: https://dilithjay.com/blog/2020/04/18/double-deep-q-networks-ddqn-a-quick-intro-with-code/.

[22] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in Proc. of AAAI Conf. on Artificial Intelligence, 2016.

[23] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529-533, 2015.

[24] C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, vol. 8, no. 3-4, pp. 279-292, 1992.

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

[26] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in Proc. of the 13th International Conf. on Neural Information Processing Systems, pp. 1008-1014, 2000.

[27] T. Saanum, "Reinforcement Learning with Simple Sequence Priors," arXiv preprint arXiv:2305.17109, 2024.