EXPLORING THE FEASIBILITY OF SIM2REAL TRANSFER IN REINFORCEMENT LEARNING

APPLICATION IN MAZE NAVIGATION

INTERNAL PROMOTOR: GEVAERT WOUTER

EXTERNAL PROMOTOR: SAM DEBEUF

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 performance.

This study contributes to AI and robotics by providing insights and methodologies for Sim2Real transfer in RL, with potential applications extending beyond robotics to other fields where simulation-based training is essential.

Preface

This 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 explores 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 supervisor, Gevaert Wouter, for his invaluable guidance and insights. His mentorship has been fundamental to my growth. I also thank Amaury Van Naemen for his technical support with 3D printing, which was crucial for my experiments.
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

List of Abbreviations

1. **AC** - Actor-Critic
2. **AI** - Artificial Intelligence
3. **AV** - Autonomous Vehicle
4. **DDQN** - Double Deep Q-Network
5. **DQN** - Deep Q-Network
6. **ESP32** - Espressif Systems 32-bit Microcontroller
7. **HC-SR04** - Ultrasonic Distance Sensor
8. **MPU6050** - Motion Processing Unit (Gyroscope + Accelerometer)
9. **MSE** - Mean Squared Error
10. **OTA** - Over the Air Updates
11. **PPO** - Proximal Policy Optimization
12. **PWM** - Pulse-Width Modulation
13. **RC** - Remote Controlled
14. **RCMazeEnv** - RC Maze Environment (Custom Virtual Environment for RL Training)
15. **RL** - Reinforcement Learning
16. **SAC** - Soft Actor-Critic
17. **Sim2Real** - Simulation to Reality Transfer
18. **TRPO** - Trust Region Policy Optimization
19. **3D** - Three-Dimensional
20. **2WD** - 2-Wheel Drive
21. **4WD** - 4-Wheel Drive

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. **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.
3. **Bellman Equation**: A fundamental recursive equation in dynamic programming and reinforcement learning that provides a way to calculate the value of a policy.
4. **Domain Adaptation**: A set of methods used to adapt a model trained in one domain (e.g., simulation) to perform well in a different but related domain (e.g., real-world).
5. **Domain Randomization**: A technique used in reinforcement learning to bridge the gap between simulation and reality by varying the parameters of the simulated environment to improve the robustness of the learned models when applied to real-world tasks.
6. **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.
7. **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.
8. **Experience Replay**: A technique in reinforcement learning where past experiences (state, action, reward, next state) are stored and randomly sampled to break the correlation between consecutive samples, improving the stability and performance of the learning algorithm.
9. **Fixed Q-Targets**: In the context of DQN, this refers to using a separate network to generate target values for training, which are held fixed for a number of steps to improve training stability.
10. **Markov Decision Process (MDP)**: A mathematical framework for modeling decision-making situations where outcomes are partly random and partly under the control of a decision-maker. It is characterized by states, actions, transition probabilities, and rewards.
11. **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.

12. **Motion Processing Unit (MPU6050)**: A sensor device combining a MEMS (Micro-Electro-Mechanical Systems) gyroscope and a MEMS accelerometer, providing comprehensive motion processing capabilities.
13. **Over the Air Updates (OTA)**: Remotely updates software or firmware, enabling seamless upgrades without physical access.
14. **Policy Gradient Method**: A class of reinforcement learning algorithms that optimize the policy directly by computing gradients of the expected reward with respect to the policy parameters.
15. **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.
16. **Proximal Policy Optimization (PPO)**: A policy gradient method for reinforcement learning that simplifies and improves upon the Trust Region Policy Optimization (TRPO) approach.
17. **Pulse-Width Modulation (PWM)**: A method used to control the amount of power delivered to a device by varying the duty cycle of the signal, commonly used in robotics to control motor speed and direction.
18. **Q-agent**: Based on the Q-learning algorithm, it is a model-free algorithm that learns to estimate the values of actions at each state without requiring a model of the environment.
19. **RC Car**: A remote-controlled car used as a practical application platform in reinforcement learning experiments, demonstrating how algorithms can control real-world vehicles.
20. **Raspberry Pi (RPI)**: A small, affordable computer used for various programming projects, including robotics and educational applications.
21. **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.
22. **Sensor Fusion**: The process of combining sensory data from multiple sources to produce a more accurate and reliable understanding of the environment.
23. **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.
24. **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.

25. **Ultrasonic Distance Sensor (HC-SR04)**: Measures distance using ultrasonic waves, used in robotics for obstacle detection and navigation.
26. **Virtual Environment**: A simulated setting designed for training reinforcement learning agents, offering a controlled, risk-free platform for experimentation and learning.
27. **Wheel Slippage**: Loss of traction causing wheels to spin without moving the vehicle forward, common on uneven terrain.
28. **3D (Three-Dimensional)**: Refers to objects or environments that have three dimensions (length, width, and height), commonly used in computer graphics and simulations.
29. **2WD (2-Wheel Drive)**: A vehicle configuration where power is delivered to two wheels, typically the front or rear wheels, providing propulsion and steering control.
30. **4WD (4-Wheel Drive)**: A vehicle configuration where power is delivered to all four wheels, enhancing traction and stability, particularly in off-road or challenging terrains.

Introduction

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

This thesis explores the intersection of reinforcement learning (RL) and Sim2Real transfer, specifically focusing on the challenges and feasibility of transferring a DDQN-trained agent from a simulated environment to a physical maze. The primary research question is: “Is it possible to transfer a trained RL-agent from a simulation to the real world? (case: maze)”

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. Rusu et al. demonstrated the effectiveness of progressive networks in bridging this gap, particularly in robot learning from pixels, highlighting the importance of adaptable architectures in overcoming environmental discrepancies [8].

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.

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.

Beyond Mazes: A Broader Canvas

While this research focuses on maze navigation, its implications extend far beyond. The principles ofSim2Real transfer can be applied to autonomous drones in urban landscapes, self-driving cars avoiding pedestrians, or medical robots operating in cluttered hospital rooms. Sim2Real transfer is the key to making these scenarios feasible.

So buckle up (or tighten your wheel nuts), as we embark on this thrilling expedition. In the following chapters, we will delve into how and why we arrived at our results. We will start with a literature review and the methodology, followed by the results and challenges encountered. Finally, we will discuss reflections and provide advice for future researchers embarking on a similar journey. Last but not least, you will find installation instructions to replicate the setup.

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 are best suited for 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.

Literature Review and Methodology.

Background on Reinforcement Learning and Reinforcement Learning Algorithms

Background on Reinforcement Learning

The challenge of Sim2Real transfer is pivotal in the deployment of autonomous systems, influencing applications ranging from robotic navigation to self-driving vehicles [18]; [3]. Recent advancements in RL, such as the introduction of Proximal Policy Optimization [4] and Soft Actor-Critic algorithms [12], have shown promise in various domains. However, the discrepancy between simulated and real environments, often referred to as the 'reality gap' [17], 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 [5]. Another promising technique is domain adaptation, which seeks to align the simulated and real-world data distributions [6]. Despite these advancements, challenges remain, particularly in ensuring the transferability of learned behaviors in complex, dynamic environments [17].

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 [3], this study aims to enhance the reliability of Sim2Real 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].

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.

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].

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.

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.

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].

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.

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.

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.

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.

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

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.

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.

Methodology

This section explores the Reinforcement Learning Maze Navigation (RCMazeEnv) method using a Double Deep Q-Network (DDQNAgent). It covers the maze environment setup, DDQN agent design, and training algorithm, including relevant mathematical functions.

Environment Setup (RCMazeEnv)

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

The agent's sensors provide readings in three directions: front, left, and right, measuring the distance to the nearest wall. The 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 goal is efficient maze navigation, reaching the end while avoiding collisions and optimizing the path based on sensor inputs and past experiences.

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

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.

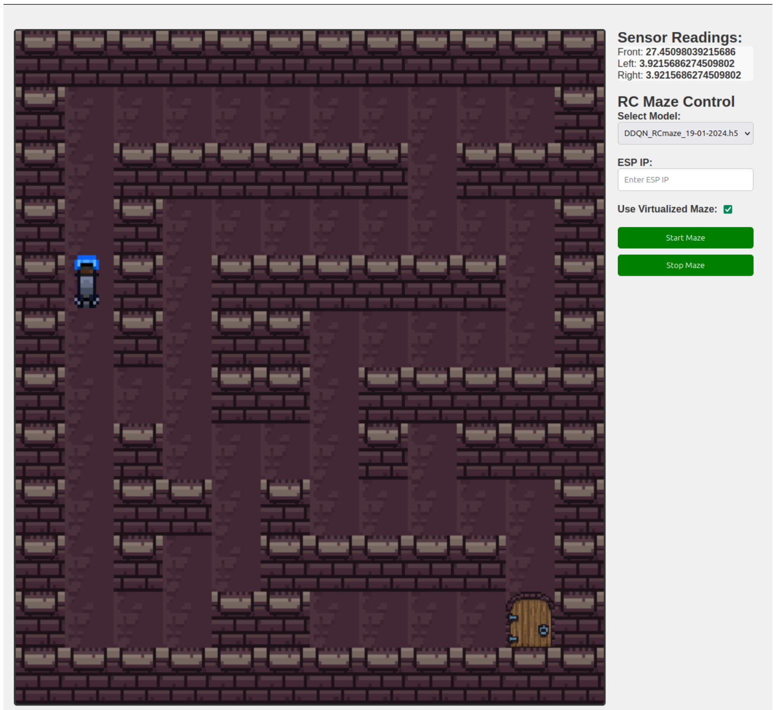

Figure 2: Web App (Image created by author)

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].

- **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:

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

Where:

- R_{t+1} is the reward received after taking action a in state s.
- γ is the discount factor.
- $\operatorname*{argmax}_a Q(S_{t+1}, a; \theta)$ 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].

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:

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

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.

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.

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:

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

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.

Addressing Research Questions

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 Sim2Real transfer practices [1].

Unity 3D, through its ML-Agents toolkit, offers highly realistic simulations and a user-friendly interface, making it a popular choice for training RL agents in various scenarios, including autonomous vehicle navigation [30]. However, its complexity and the need for substantial computing resources can pose challenges for beginners.

AirSim, developed by Microsoft, provides highly realistic environments for both drones and cars, leveraging the Unreal Engine for superior visual fidelity and physics accuracy. It supports hardware-in-the-loop simulations and offers APIs for integrating with various AI and robotics frameworks [29]. Despite its strengths, the complexity of setup and resource requirements can be a drawback for some users.

CARLA is specifically designed for autonomous driving research and offers a wide range of features for simulating urban driving scenarios. It provides realistic traffic scenarios and supports various sensors, making it a strong choice for traditional vehicle simulations [2]. However, it is less tailored for RC cars, which might limit its applicability in this context.

ISAAC Gym, developed by NVIDIA, focuses on high-fidelity physics simulations and is optimized for GPU acceleration, making it ideal for robotics simulations. It offers extensive support for reinforcement learning algorithms, though its primary focus on robotics may not align perfectly with the goals of this project [29].

OpenAI Gym's simplicity and reinforcement learning focus make it the ideal fit for this application. Additionally, OpenAI Gym's wide acceptance in the academic community and extensive documentation provide a robust foundation for developing custom environments tailored to specific research needs [1].

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

For the autonomous navigation of a virtual RC car in a maze, various reinforcement learning (RL) techniques were considered, including Deep Q-Network (DQN), Double Deep Q-Network (DDQN), Q-Learning, Proximal Policy Optimization (PPO), and Actor-Critic (AC). After careful consideration and testing, DDQN was selected as the most suitable technique for this project.

Deep Q-Network (DQN) was initially considered due to its significant breakthrough in RL, effectively handling high-dimensional sensory inputs and achieving impressive performance in many tasks. However, DQN tends to overestimate Q-values, leading to instability and slower learning. Due to these overestimation issues, DQN was not as stable or reliable as DDQN for this project [3].

Q-Learning, known for its simplicity and effectiveness in discrete and small state spaces, was also evaluated. While it is straightforward to implement and model-free, Q-Learning struggles with large or continuous state spaces, requiring a Q-table that grows exponentially, making it impractical for complex tasks. Given the complexity of maze navigation and high-dimensional sensory inputs, Q-Learning was not feasible for this application [3].

Proximal Policy Optimization (PPO) offers robustness, efficiency, and the ability to handle both continuous and discrete action spaces, maintaining stable updates with its clipped objective function. However, PPO's policy optimization approach sometimes leads to less precise value estimations compared to DDQN, which focuses on accurate Q-value approximations. Although PPO is a powerful technique, the need for precise Q-value approximations in maze navigation made DDQN a better fit [4].

Actor-Critic (AC) methods combine the strengths of policy-based and value-based methods, reducing variance in updates and generally converging faster. Despite these advantages, AC methods can be complex to implement and may not achieve the same level of stability and performance as DDQN in tasks requiring precise action evaluation. The complexity and less consistent performance of AC methods compared to DDQN led to the decision to not use AC for this project [25].

Double Deep Q-Network (DDQN) addresses the overestimation bias in DQN by decoupling action selection from value estimation, resulting in more accurate Q-value approximations and improved learning stability. It handles high-dimensional sensory inputs effectively and balances exploration and exploitation well. After testing, DDQN proved to outperform other methods in maze-like virtual RC car scenarios, making it the optimal choice for this application [3].

By selecting DDQN, the project leverages its strengths in stability, accuracy, and performance, ensuring effective navigation and learning in complex, sensor-driven environments.

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 requires addressing discrepancies in sensor data interpretation, action synchronization, and physical dynamics. Real-world sensors may introduce noise and inaccuracies not present in the simulation, and the car's physical dynamics, like friction and wheel slippage, can differ significantly [6].

To mitigate these issues, I implemented sensor data normalization and action synchronization mechanisms to align simulation outcomes with real-world performance. Introducing failsafe mechanisms and adjusting motor control timings were essential in reducing collision risks and movement inaccuracies. Iterative testing and adaptation were crucial in this process [6].

One effective approach to handle these discrepancies is domain randomization, which involves training the model on a wide range of simulated environments with varied parameters. This technique helps the model generalize better to real-world scenarios by exposing it to diverse conditions during training [5]. Another strategy is domain adaptation, where the model is fine-tuned using a small amount of real-world data to better match the target domain [9].

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 RC 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]. Studies have shown that training in simulation can drastically reduce the required real-world training time, allowing for rapid iteration and improvement of the models [10].

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 Sim2Real 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 [8]. Additionally, implementing sensor fusion techniques can improve the robustness of the real-world model by combining data from multiple sensors to provide more accurate and reliable inputs [12]. This approach helps in mitigating the effects of sensor noise and inaccuracies, further aligning the simulation-trained model with real-world conditions.

Model Architecture and Training Insights

To understand how our Double DQN model learns and makes decisions, we examine its architecture. The model has four dense layers that output three actions tailored to the RC car's movement, enabling it to navigate the maze efficiently.

The Double DQN architecture addresses overestimation in action-value functions, a common problem in RL. By decoupling action selection from action evaluation during target value calculation, Double DQN reduces overestimation errors, leading to more accurate value estimates and improved policy performance. [19] [20].

Double DQN uses two networks: the primary network selects the best action, and the target network evaluates its value. This method reduces the bias introduced by the max operator 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)

---
```

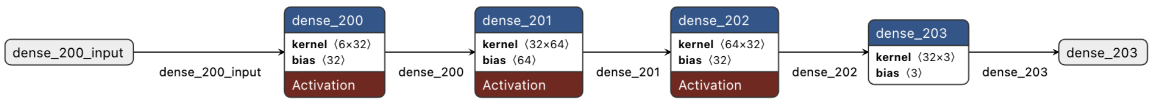

Figure 3: Model Architecture of the Double Deep Q-Network (DDQN) used in the study. (Image created by author)

Training Parameters

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

- **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`)**: 175
  - 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.

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.

Evaluation Metrics Overview

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.

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.

Experimental Outcomes and Comparative Analysis

Performance Evaluation Metrics

Several metrics were used to assess the agent's effectiveness, including reward, loss, and epsilon history per episode during training. These metrics provided insights into the learning progress and performance, allowing for quick adjustments to reward functions or hyperparameters.

Comparative Analysis of Reinforcement Learning Algorithms

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.

1. Visit Heatmaps

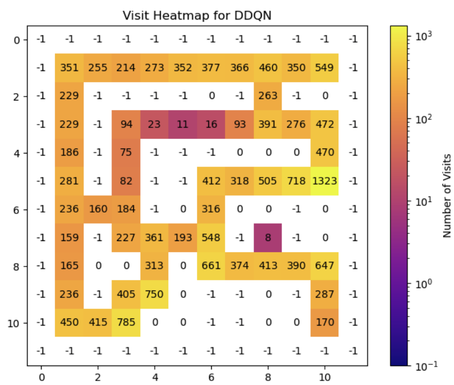

Figure 4: DDQN Heatmap (Image created by author)

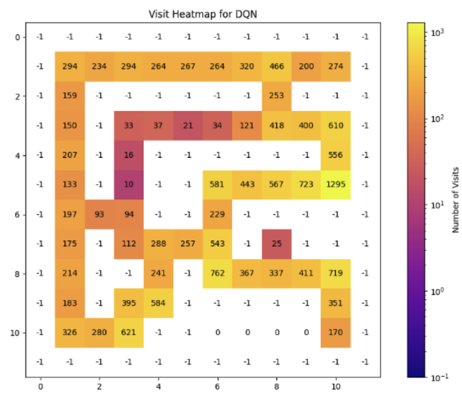

Figure 5: DQN Heatmap (Image created by author)

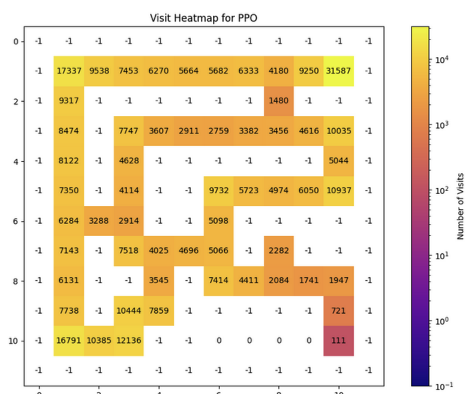

Figure 6: PPO Heatmap (Image created by author)

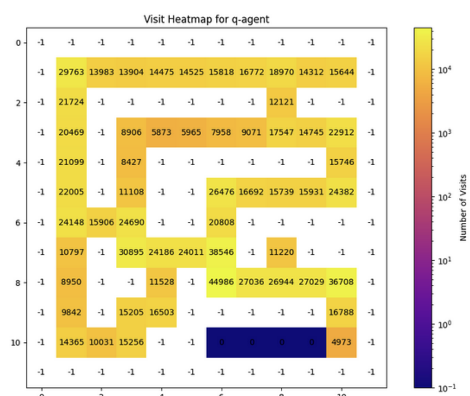

Figure 7: Q-agent Heatmap (Image created by author)

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 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.

2. Maze Solution Efficiency

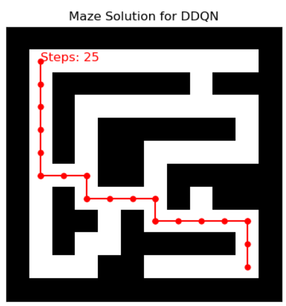

Figure 8: DDQN Maze Path (Image created by author)

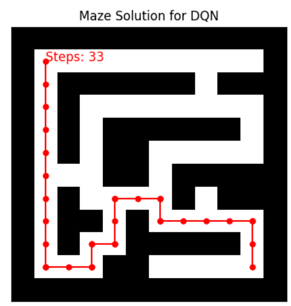

Figure 9: DQN Maze Path (Image created by author)

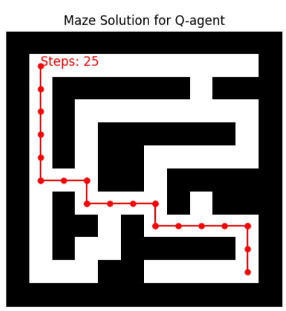

Figure 10: Q-agent Maze Path (Image created by author)

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).

3. Reward History and Distribution

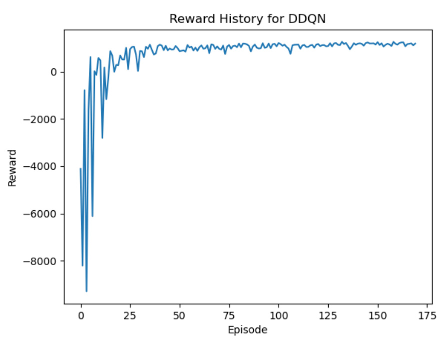

Figure 11: DDQN Reward History (Image created by author)

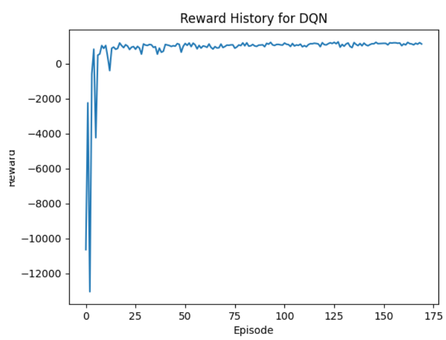

Figure 12: DQN Reward History (Image created by author)

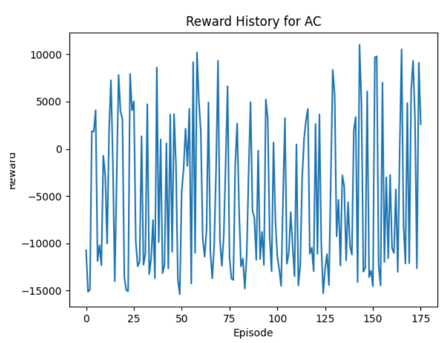

Figure 13: AC Reward History (Image created by author)

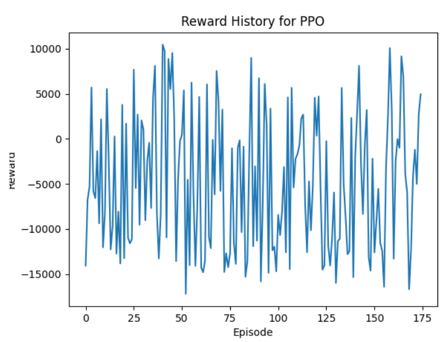

Figure 14: PPO Reward History (Image created by author)

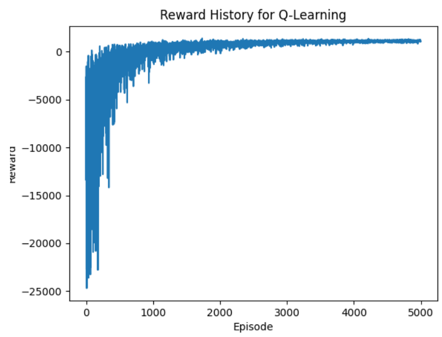

Figure 15: Q-agent Reward History (Image created by author)

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.

4. Mean Squared Error (MSE) Over Time

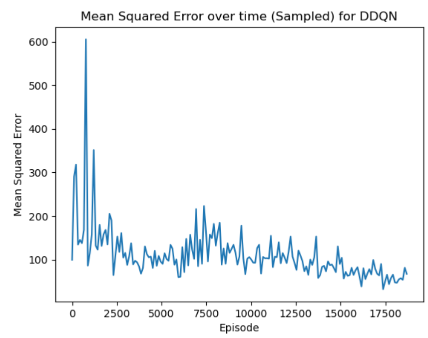

Figure 16: DDQN MSE (Image created by author)

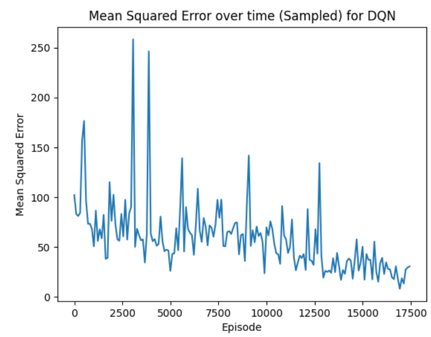

Figure 17: DQN MSE (Image created by author)

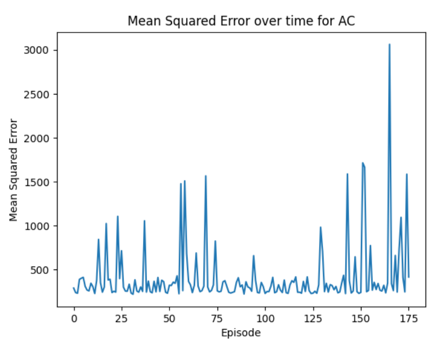

Figure 18: AC MSE (Image created by author)

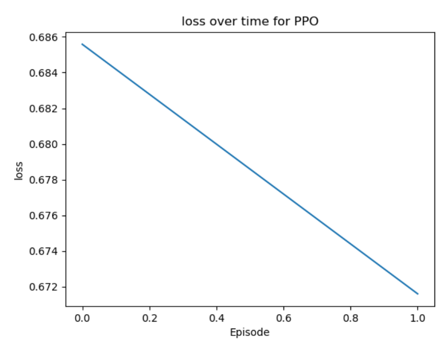

Figure 19: PPO Loss (Image created by author)

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.

5. Moving Average of Rewards

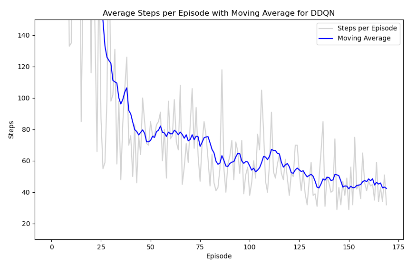

Figure 20: DDQN Moving Average (Image created by author)

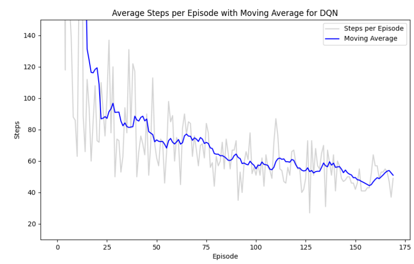

Figure 21: DQN Moving Average (Image created by author)

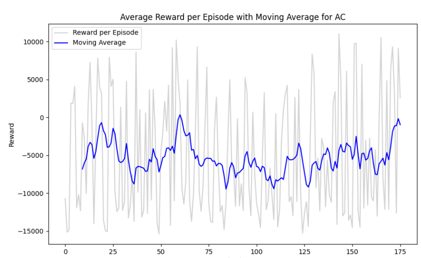

Figure 22: AC Moving Average (Image created by author)

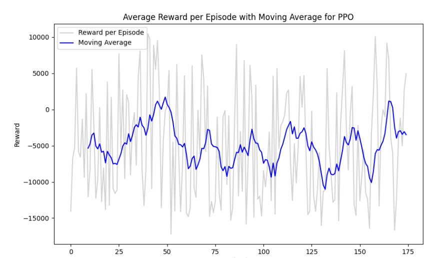

Figure 23: PPO Moving Average (Image created by author)

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.

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.

This analysis helps select the most suitable reinforcement learning algorithm based on task requirements and environmental complexities, enhancing our understanding and optimizing performance.

Implementation of Real-World Control Algorithms

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 Sim2Real transfer.

System Overview

At the core of the system is the ESP32-WROOM-32 module, a small and powerful micro-controller with integrated Wi-Fi and Bluetooth capabilitie.

The structure of the RC car is built on a 2WD miniQ robot chassis with a custom 3D printed top plate, which provides a sturdy base for mounting all components. To control the motors, an L298N dual H-Bridge motor controller was employed, allowing us to drive the motors in both directions and adjust their speed using pulse-width modulation (PWM) signals from the ESP32 microcontroller.

For sensor integration, the system utilizes HC-SR04 ultrasonic sensors for distance measurement and an MPU6050 gyroscope for orientation and stabilization. These sensors are mounted using custom 3D printed parts, specifically designed to hold the HC-SR04 sensors securely in place. Additionally, a mini OLED screen is incorporated into the setup to provide real-time feedback on the robot's status, such as its IP address and operational states, enhancing user interaction and debugging capabilities.

Powering the entire system is an 18650 battery, Providing a lightweight and portable energy source. The battary is connected to the ESP32 microcontroller using a battery shield.

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;
        # Read sensor data
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        # Accumulate gyro Z-axis readings
        gyroZAccum += gz;
        # Delay for sensor stability
        delay(20);
    }
    // Calibration based on ±100 readings
    mpu.setZGyroOffset(-gyroZAccum / 13100);
    Serial.println("Calibration Complete");
}
```

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.

Enhanced Sensor-Based Navigation

Simulated environments provide an ideal platform for refining sensor-based navigation technologies. By leveraging simulations, developers can test and optimize sensor configurations and algorithms in a risk-free setting. This is particularly beneficial in applications demanding high precision, such as urban navigation or automated delivery systems. However, real-world conditions introduce complexities like sensor noise and environmental variability that are not fully captured in simulations. Thus, while simulations can significantly enhance sensor-based navigation, the transition to real-world applications requires careful calibration and validation.

Informing Autonomous Vehicle Movement

Simulations offer controlled settings to study vehicle dynamics and movement responses. Insights gained from these controlled environments can inform the development of advanced algorithms for real-world applications. For example, vehicle dynamics in simulations can help refine control algorithms to better manage real-world unpredictability. However, real-world environments present additional challenges such as varying road surfaces, weather conditions, and mechanical issues. Algorithms that perform well in simulations may need substantial adjustments to handle these real-world factors effectively.

Challenges and Solutions in RL Implementation

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.

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.

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.

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.

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.

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**:

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

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.

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.

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.

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.

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).

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).

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 Sim2Real transfer (see Video E11 in the Video References section).

Practical Implementation Considerations

Successful application of simulation insights to real-world autonomous vehicles requires addressing several practical aspects:

- **Sensor Calibration**: Regular calibration to account for environmental influences and sensor degradation.

- **Algorithm Adjustment**: Modifications to account for hardware limitations and real-world dynamics.
- **Handling Unpredictability**: Developing algorithms that can adapt to real-world unpredictability and handle unexpected scenarios.

By focusing on these considerations, the transition from simulation to real-world applications can be more seamless, enhancing the safety, efficiency, and reliability of autonomous vehicle technologies.

Conclusion for Real-World Application

The transition from simulation-based research to real-world applications in autonomous vehicle navigation presents significant challenges. While simulations offer valuable insights and a controlled environment for testing, real-world conditions introduce variability and unpredictability that require careful handling.

Major Limitations Identified:

- **Imprecise Sensor Data**: Real-world sensors often provide noisy and incomplete data, which can affect positioning accuracy and decision-making.
- **Environmental Variability**: Surface variations, weather conditions, and dynamic obstacles necessitate continuous algorithmic adjustments and recalibration.

To bridge the gap between simulation precision and real-world variability, it is crucial to focus on:

- **Robust Sensor Calibration**: Ensuring sensors are regularly calibrated to maintain accuracy.
- **Algorithmic Adaptation**: Continuously refining algorithms to handle real-world complexities.
- **Iterative Testing and Validation**: Conducting extensive real-world testing to validate and improve system performance.

Overcoming these challenges is key to successfully integrating Sim2Real technologies into autonomous transportation systems. Doing so will enhance the safety, efficiency, and reliability of these systems, paving the way for broader adoption and practical applications in various real-world scenarios.

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 Sim2Real transfer and ensuring consistent performance across different environments.

Discussion and Reflection

Looking back on my research journey, I've learned a ton and grown a lot personally. Here's a rundown of the key insights and lessons from working on my RC car project, especially around Sim2Real transfer in reinforcement learning.

Embracing Innovation and Adaptability

One of the biggest takeaways from this project was the importance of staying open to new ideas and being flexible. Moving from simulation to real-world testing was tough, especially with sensor data discrepancies and movement control issues. These challenges pushed me to think creatively and be open to exploring new paths. My mentor, Gevaert Wouter, stressed the need to stay updated with new tech and be adaptable to changes, even though this sometimes led to more complications, like potential slippage in a 4WD setup.

Wouter pointed out that while a 4WD setup could provide better traction, it generally would not be better due to over time slippage. Additionally, switching from ultrasonic sensors to cameras might not improve performance because accurately determining the car's position in the maze could still be a challenge.

Bridging Theory and Practice

The transition from theoretical knowledge to practical application proved to be more complex than anticipated. While virtual environments were manageable and controlled, real-world conditions introduced unforeseen variables. This experience underscored the gap between simulation and reality, highlighting the necessity for continuous adjustment and iterative testing. Practical engagements, such as calibrating sensors and adjusting control algorithms, reinforced my ability to balance theoretical insights with practical needs. Feedback from interviews, including Wouter's advice on starting with a flexible virtual environment, proved invaluable in navigating these complexities.

Reddit feedback suggested expanding the action space in the virtual environment to include direct motor control. This approach could have improved the agent's learning and adaptability but would have required a longer training period and increased the risk of motor damage. Wouter advised against giving the agent direct motor control due to these potential new issues.

- **Getting Down to Specifics:**
  - **Sensor Calibration and Data Normalization:** Early on, I figured out that sensor readings between different environments were like apples to oranges. Getting them to match up meant a lot of trial and error with calibration steps and tweaking data normalization to get consistent readings.
  - **Motor Encoder Issues:** When I threw motor encoders into the mix to get precise movement, things got really techy. These gadgets were supposed to track every little move, but they ended up giving us a hard time with glitches and misreads, which meant going back to the drawing board more than once.

Anticipatory Thinking and Proactive Problem-Solving

Throughout the project, anticipating challenges and planning ahead were crucial. I had to be proactive in dealing with sensor noise and changing environments, which improved the system's performance and prepped me for future tech trends. Wouter suggested in the future a more waterfall-like technique: creating, testing, and refining the simulation with the car—to avoid last-minute surprises about the virtual setup's limitations would have been a better approach.

Feedback and Continuous Improvement

During the evaluation of the practical part of this research the jury provided positive feedback, recognizing the successful application of reinforcement learning techniques. However, they also noted specific suggestions, such as the consistency of maze navigation in real-world applications. Their suggestions, including increasing wall distances and using a camera, offered possible solutions to these problems.

Methodological Rigor and Insights

Building the custom maze navigation environment (RCMazeEnv) and using the DDQN architecture was a real eye-opener. It gave me a solid understanding of the nuts and bolts of reinforcement learning. The whole process of testing and tweaking was crucial; every round of changes made the system better and better. Feedback from interviews really drove home the point that having a well-organized setup is key, and I learned a lot about the limits of virtaul environments.

One thing that surprised me that Wouter pointed out was how my virtual twin setup didn't really add much value. I thought it would be something nice and really useful, but it turned out that a simple top-down view camera would be way more effective for real-time feedback, since the virtual twin didnt show when the real car was stuck for example (or rather why it got stuck), Additionally, it could have been used as an additional input to the agent/environment.

Educational Value

The educational value of this project is huge. By documenting the whole process and the challenges faced, this project becomes a fantastic learning tool for anyone interested in AI and robotics. It shows how to apply reinforcement learning in the real world, effectively bridging the gap between theory and practice.

This project is all about hands-on learning. Students and researchers can set up their own experiments to see how reinforcement learning, sensor calibration, and robotic control work in real life. This kind of hands-on experience is invaluable for really understanding these complex concepts.

For teachers, this project is a goldmine. The detailed steps and problem-solving approach make it a great resource for AI and robotics courses. It's perfect for showing students how theoretical concepts apply in the real world. Plus, the documented challenges and solutions make for great case studies in class, helping students develop their critical thinking and problem-solving skills.

The project's mix of successes and setbacks can also inspire others to dive into their own AI and robotics projects. It shows that hitting roadblocks is just part of the journey and can lead to major breakthroughs. This can motivate students to keep pushing forward, even when things get tough.

By making the project's code and documentation open-source, it becomes an even more powerful educational tool. Other students and researchers can build on this work, make improvements, and adapt the methods for their own projects. This creates a collaborative learning environment where knowledge and resources are shared, promoting continuous learning and innovation in AI and robotics.

Personal Growth and Aspirations

This project made me realize how much I love research. Exploring new ideas, overcoming obstacles, and seeing progress was incredibly rewarding. I'm excited about the endless possibilities in AI and robotics, and I'm determined to keep learning and exploring in this dynamic field.

Commitment to Innovation and Continuous Learning

Moving forward, I'm committed to fostering a culture of innovation and continuous improvement. This means creative problem-solving, interdisciplinary collaboration, and staying focused on the bigger picture. The lessons I've learned from this project have prepared me for the next stage of my journey, where I'll keep refining my methods and embracing new challenges.

Personal Note

Reflecting on the entire project, I recognize areas where I could have approached things differently. Initially diving into complex simulations without fully considering practical constraints limited the scope of potential solutions. However, this journey has been incredibly rewarding, and I look forward to continuing this exploration, guided by the insights and experiences gained.

Feedback from Reddit came from several people who filled in my Google form. I posted the link on the subreddit r/reinforcementlearning, a platform that has been helpful in the past. Despite receiving only three responses, the feedback was valuable and reconfirmed some of the thoughts I already had about improving the project.

Advice for Students and Researchers

Practical Utilization of Simulations

Simulations are super handy in research because they let you develop and test algorithms in a safe, controlled environment without needing physical prototypes right away. They save a lot of money by cutting down on the need for physical models and endless real-world trials during the early stages.

Simulations also let you quickly try out new ideas and see how they work without having to build and rebuild physical models. This speed and flexibility are invaluable when you're developing complex systems like autonomous RC cars.

Strategies for Effective Transition from Simulation to Reality

Moving from simulations to real-world applications isn't always easy, but with the right plan, it's totally doable. Start by refining your algorithms in simulations and then gradually introduce real-world testing. Make sure to have continuous feedback loops to tweak your simulation models based on what you learn from the real world, making them more accurate and useful.

Incremental testing is key. Start small with controlled, simple scenarios that closely match your simulations. As your models prove themselves, gradually introduce more complexity and variability. This helps you catch issues early and adapt your models step by step.

Another great strategy is to use hybrid testing environments where simulations and real-world tests are combined. This approach lets you validate your models in a controlled setting before fully transitioning to the real world.

Overcoming Common Challenges in Simulation-to-Reality Transitions

Making sure your simulations match real-world conditions can be tricky, especially with sensor data and mechanical operations. Regularly calibrating your sensors and ensuring that the physical movements match what the simulations expect is key to smooth transitions.

For example, sensor noise can be a big issue. In simulations, you can control for noise, but real-world sensors will always have some level of unpredictability. Implementing noise models in your simulations can help your algorithms learn to handle real-world data more effectively.

Mechanical discrepancies are another challenge. Simulated environments often assume perfect conditions, but real-world robots deal with friction, slippage, and wear and tear. Continuously comparing your simulation data with real-world results and adjusting accordingly can help mitigate these issues.

Insights from My Research

During my research, I found that picking the right simulation platform is super important. Tools like OpenAI Gym are great, but for more complex scenarios, you might need additional tools. I also discovered that Double Deep Q-Network (DDQN) outperforms other models like DQN and PPO by reducing overestimations and making learning more stable.

A big takeaway was the value of starting with simple simulations. Initially, I jumped into complex simulations without considering the practical constraints of my RC car. Starting simpler would have saved a lot of headaches. I also learned the importance of iterative testing and adapting based on what you learn from each step.

Methodological Advice

Use both qualitative and quantitative methods to thoroughly evaluate how well your simulations and real-world applications are working. Stay flexible and open to feedback to address any unexpected challenges effectively.

Documentation is your friend. Keep detailed logs of what works and what doesn't, including all the tweaks and adjustments you make along the way. This not only helps you track progress but also provides a valuable resource for troubleshooting and future projects.

Engage with the community. Platforms like Reddit or GitHub can provide valuable feedback and suggestions from other researchers and enthusiasts who have faced similar challenges. Their insights can be incredibly helpful in refining your approach.

Practical Experiment Integration

Iterative design and prototyping are the way to go. This approach helps you progressively refine your systems, linking theoretical research with practical implementation. Regularly seek and incorporate feedback from stakeholders and peers to improve both your simulation models and real-world applications.

Prototyping doesn't just apply to your final product. Prototype your methods, too. Try different simulation setups, test various algorithms, and iterate on your approach. This experimentation phase is crucial for finding the most effective methods and tools.

Importance of Fail-Safe Mechanisms and Duplicate Components

Having fail-safe mechanisms and duplicates of all components is crucial. In my project, I learned the hard way that mounting components can sometimes lead to accidental damage. I ended up destroying several parts, which meant I had to wait for replacements and lost a lot of time. To avoid this, always have spares on hand. This ensures that if something breaks, you can quickly replace it and keep your project moving forward.

Fail-safe mechanisms are also vital. These systems can help prevent damage to your components by automatically shutting down the system if something goes wrong. This not only protects your hardware but also saves time and money in the long run.

Guidelines for Future Research

Introduction for Future Research

This chapter outlines a comprehensive methodology for researchers involved in simulation-based studies, focusing on smoothly transitioning from theoretical models to practical applications.

Step-by-Step Plan

Step 1: Selection of Simulation Environments

Research and evaluate different simulation tools like OpenAI Gym, Unity 3D, AirSim, CARLA, and ISAAC Gym. Set up criteria focusing on fidelity, ease of use, integration capabilities, and specific needs like sensor simulation. Do some initial tests to see how well each platform replicates real-world conditions using simple test cases.

Survey the community and read reviews. Other researchers' experiences can provide insights into the strengths and weaknesses of each platform. Consider joining forums or groups focused on the tools you're interested in to gather more detailed user feedback.

Step 2: Designing the Simulation Environment

Create a custom maze environment like RCMazeEnv in OpenAI Gym. Make sure it includes realistic physical properties like friction and wheel slippage. Integrate virtual sensors that match your real-world sensors and add elements like sensor noise, dynamic obstacles, and varied lighting conditions to mimic real-world challenges.

Don't forget to iterate on your environment design. Start with a basic setup and gradually add complexity. This step-by-step increase in difficulty helps ensure that your algorithms can handle real-world unpredictability.

Step 3: Incremental Testing and Feedback Integration

Start with initial simulation testing to see how well the agent learns and performs. Gradually introduce real-world testing with controlled, simple scenarios that match your initial simulation setup. Use continuous feedback from these tests to refine the simulation environment, adjusting parameters based on what you observe.

Document each test and its results meticulously. This data will be invaluable for identifying patterns and making informed adjustments. Also, consider using automated tools to gather and analyze test data to streamline the process.

Step 4: Addressing Sensor Discrepancies and Movement Alignment

Regularly calibrate your sensors to ensure accurate data collection in both simulations and real-world tests. Make sure the movement mechanics of the simulated and real RC cars are consistent, including motor speeds, wheel slippage, and turning radii.

Develop a routine for sensor calibration and stick to it. Consistency is key to ensuring that your data remains reliable over time. Also, consider using calibration tools or software that can automate parts of this process.

Step 5: Enhancing Data Accuracy and Normalization

Use robust techniques to normalize sensor data between simulation and real-world environments. Regularly perform consistency checks to ensure the normalized data stays accurate across different scenarios.

Experiment with different normalization techniques to find what works best for your specific setup. Keep track of any anomalies and adjust your methods accordingly. Consistency checks should be part of your regular testing routine to catch issues early.

Conclusion for Future Research

By following these steps, researchers can systematically improve their simulation-to-reality projects, ensuring more accurate and reliable outcomes. This methodical approach leverages continuous feedback, interdisciplinary collaboration, and iterative testing to bridge the gap between simulations and real-world applications effectively.

Reflecting on my research journey, I realize I could have approached things differently. Starting with a simple simulation, then testing it with the RC car, and making necessary adjustments would have been more efficient. Instead, I began with a complex simulation without considering the practicalities of the real car. This taught me the importance of iterative testing and continuous adaptation in research.

I also learned that the virtual twin setup I initially implemented didn't add much value compared to a simple top-down view camera for real-time feedback. This experience showed me the importance of choosing the right tools and being open to simpler, more practical solutions when they offer better results. Moving forward, I'll consider a more step-by-step approach: planning, building, testing, and refining in stages to avoid last-minute surprises and make sure each step is grounded in reality.

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.

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].

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 Sim2Real transfer. GitHub repositories, such as the "Sim2Real_autonomous_vehicle" project [13], detailed the practical steps and challenges of implementing RL in physical systems.

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 Sim2Real transfer in deep RL for robotics [17]. These articles provided in-depth methodologies and highlighted the challenges of applying RL to autonomous systems.

Synthesis and Research Direction

These diverse sources collectively informed the direction of this research, focusing on the feasibility and complexities of Sim2Real 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.

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 10: 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 4.2: 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 Sim2Real 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 Sim2Real applications, ensuring they can meet the demands of real-world conditions.

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 24: 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 25: 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 26: 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

Figure 27: QR code for video E7. (Video by author.)

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

Figure 28: QR code for video E9. (Video by author.)

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 Sim2Real transfer.

- Click here to go to the video: Video E11

Figure 29: QR code for video E11. (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

Figure 30: QR code for Web App Demo. (Video by author.)

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 DDQN Simulation. (Video by author.)

References

[1] G. Brockman et al., "OpenAI Gym," arXiv preprint arXiv:1606.01540, 2016. [Online]. Available: arXiv. [Accessed: Jan. 29, 2024].

[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 et al., “Proximal Policy Optimization Algorithms,” arXiv preprint arXiv:1707.06347, 2017. [Online]. Available: arXiv. [Accessed: Jan. 29, 2024].

[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.

[28] W. Zhao et al., “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey,” in IEEE SSCI, 2020.

[29] Unity Technologies, “AirSim on Unity: Experiment with autonomous vehicle simulation,” Unity Blog, 2018. [Online]. Available: Unity Blog. [Accessed: Jan. 29, 2024].

[30] Unity Technologies, “Introducing Unity Machine Learning Agents Toolkit,” Unity Blog, 2018. [Online]. Available: Unity Blog. [Accessed: Jan. 29, 2024].

Appendices

Guest Speakers

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.

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.

Installation Guide

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.

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.

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
```

Hardware Setup and Assembly

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.

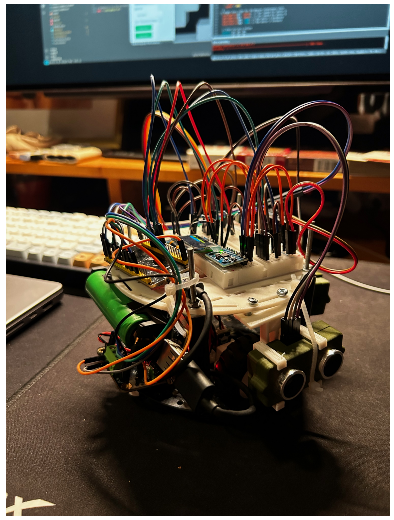

Figure 32: Final RC Car (Image created by author)

Components List

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 Chassis**
  - 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

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

Tools Required

- Screwdriver
- Wire cutter/stripper
- Drill (for mounting the top plate)

Assembly Instructions

Step 1: Base Assembly To assemble the base, you can follow this YouTube video from the makers themselves:

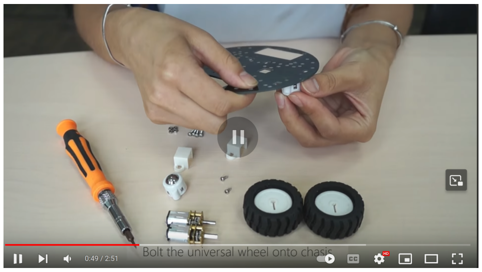

Figure 33: MiniQ 2WD Robot Chassis Quick Assembly Guide

Figure 34: QR code for MiniQ 2WD Robot Chassis Assembly Guide

Step 2: Attach Motor Driver Attach the motor driver to the base using the 2 screws that came with the kit. The motor driver should be positioned on the base such that it fits snugly without obstructing any other components.

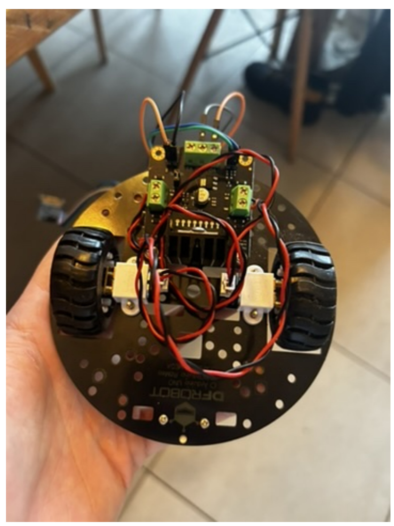

Figure 35: Motor Driver Attached to the Base (Image created by author)

Step 3: Attach ESP32-WROOM-32 Module to the Motor Driver Connect the wires of the motor driver to the ESP32-WROOM-32 as shown in the electrical schematic below:

```
int E1 = 2; //PWM motor 1
int M1 = 17; //GPIO motor 1
```

```
int E2 = 19; //PWM motor 2
int M2 = 4; //GPIO motor 2
```

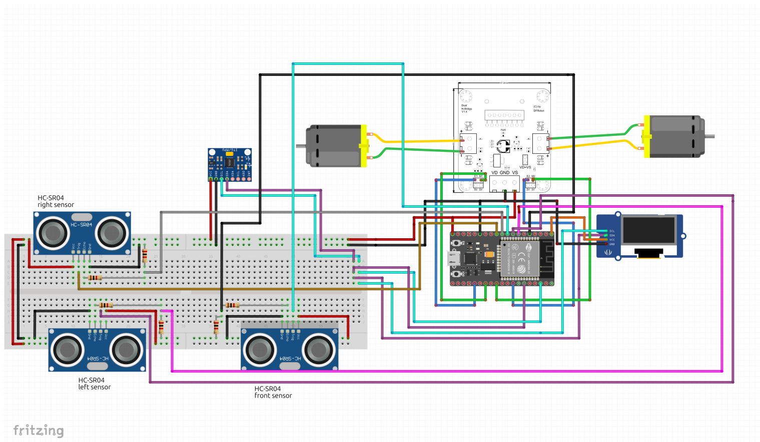

Figure 36: ESP32 Wiring Schematic (Image created by author)

Step 4: Cut the Support Beams Cut the support beams so that we can securely attach the top plate to the base. I cut them to approximately 7cm.

Figure 37: Cut Support Beams (Image created by author)

Step 5: Screw in the Supports on the Bottom of the Bottom Plate Secure the supports on the bottom of the bottom plate with screws.

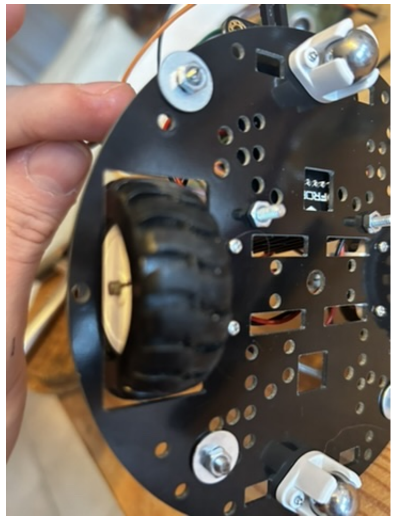

Figure 38: Supports Screwed on the Bottom Plate (Image created by author)

Step 6: Mount All the Supports on the Bottom Plate Mount all the supports on the bottom plate as shown.

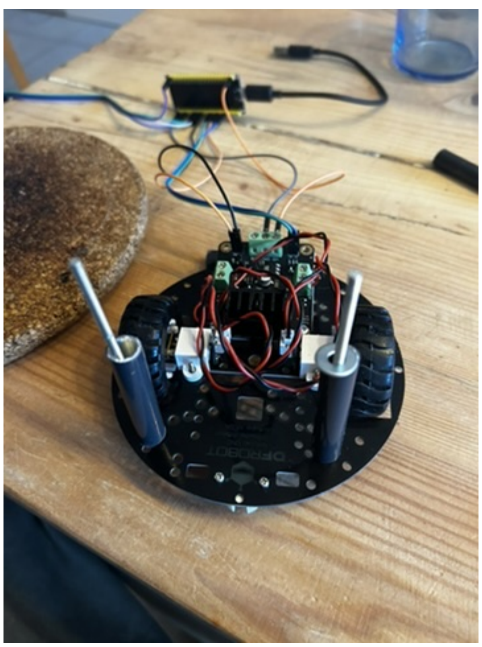

Figure 39: All Supports Mounted (Image created by author)

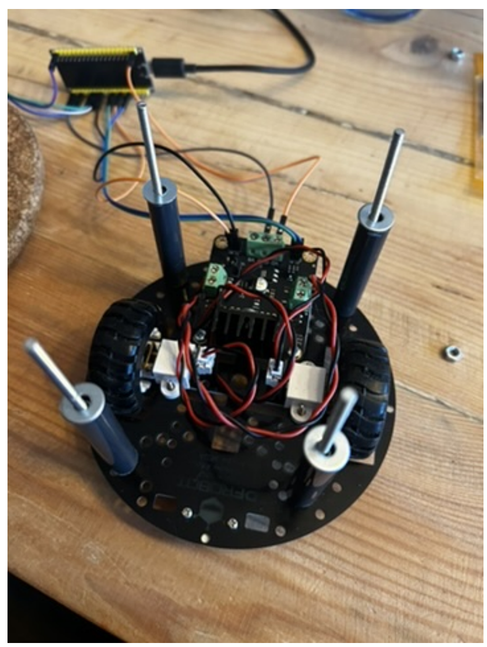

Figure 40: Complete View of Mounted Supports (Image created by author)

Step 7: Attach the Top Plate Drill holes in the top plate to fit the supports and attach it securely.

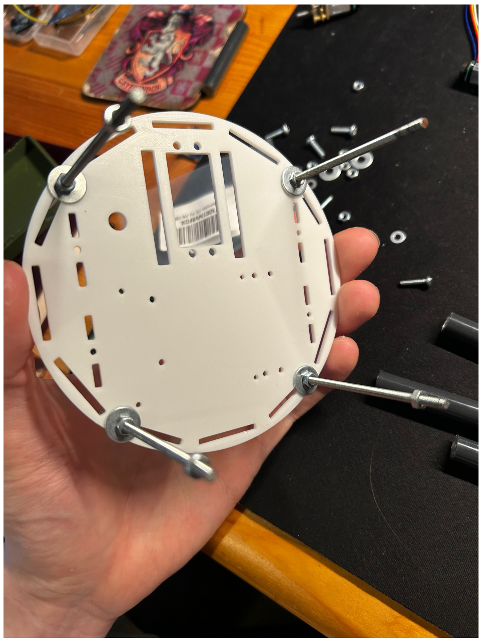

Figure 41: Top Plate Assembly (Image created by author)

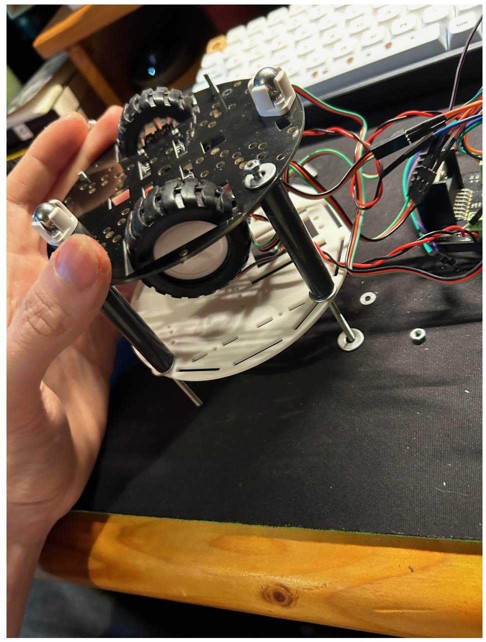

Figure 42: Bottom View with Supports and Top Plate (Image created by author)

Step 8: Attach the Ultrasonic Sensor to the Top Plate Mount the ultrasonic sensor to the top plate.

Figure 43: Ultrasonic Sensor Attached to the Top Plate (Image created by author)

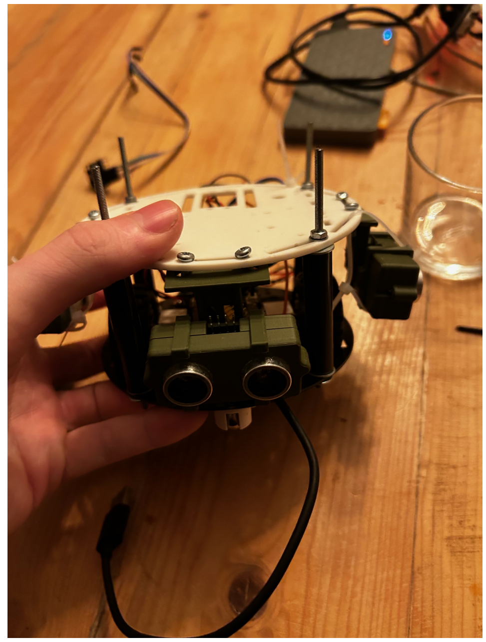

Figure 44: Ultrasonic Sensors Attached (Image created by author)

Step 9: Place the ESP32 on the Top Plate Place the ESP32 on the top plate together with a mini breadboard for the sensor wires. Secure the battery for the ESP32 to the top plate with zip ties.

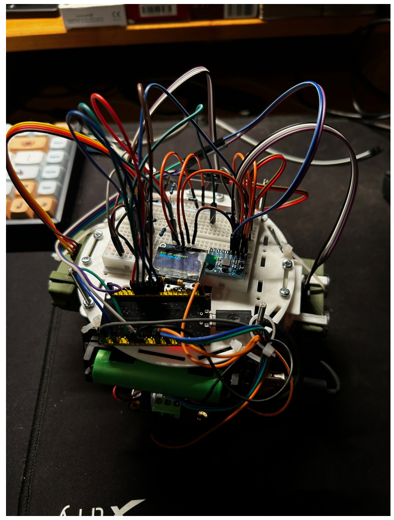

Figure 45: ESP32 Placement on Top Plate (Image created by author)

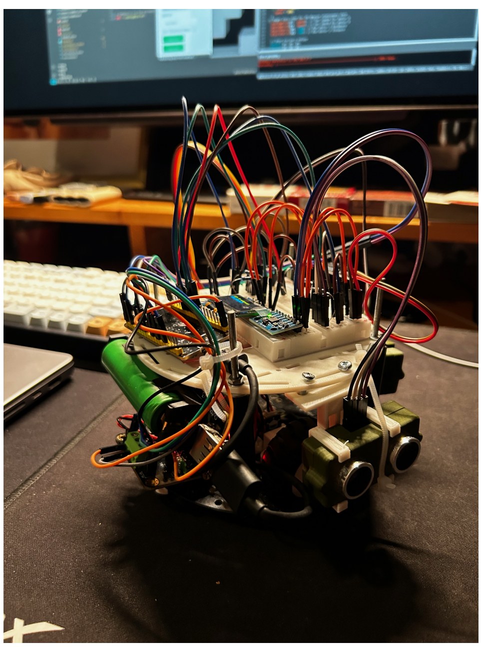

Figure 46: Final RC Car Assembly (Image created by author)

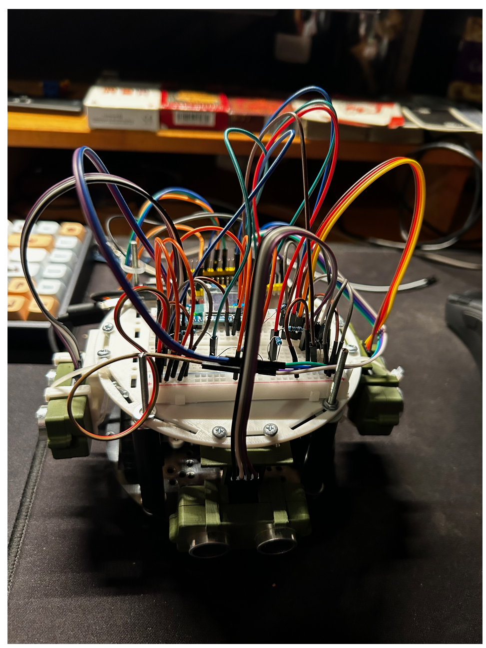

Figure 47: Final Assembly with Sensors and Breadboard (Image created by author)

Wiring Guide

ESP32 Wiring The wiring connections for the ESP32 microcontroller are shown in the diagram below. The pins are connected to the motor driver, sensors, OLED display, and MPU6050 gyroscope.

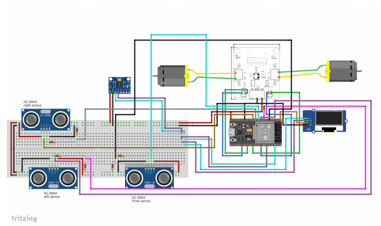

Figure 48: Wiring Diagram for ESP32 (Image created by author)

ESP32 Pins Since the schematic might not be very clear, here is a list of the pins used on the ESP32:

```
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 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
```

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.

Web Application Setup

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

Steps

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

   ```
   cd ./web_app/
   ```

2. Launch the Docker containers with:

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

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).

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.

Building the Maze

Final Result The following images show the final build of the maze used in the project.

Figure 49: Maze Build (Image created by author)

Figure 50: Final Maze Build (Image created by author)

Prerequisites The materials and tools required for building the maze are listed below:

Figure 51: Screws (Image created by author)

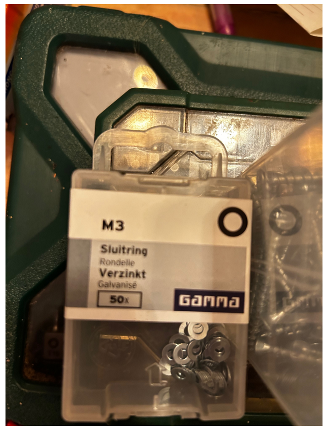

Figure 52: Nuts (Image created by author)

Figure 53: Supports (Image created by author)

- Wood used:
  - Planks cut to 10cm width by 120cm length

Figure 54: Wood Planks (Image created by author)

Step 1: Calculations Where 1 cell is 25cm x 25cm.

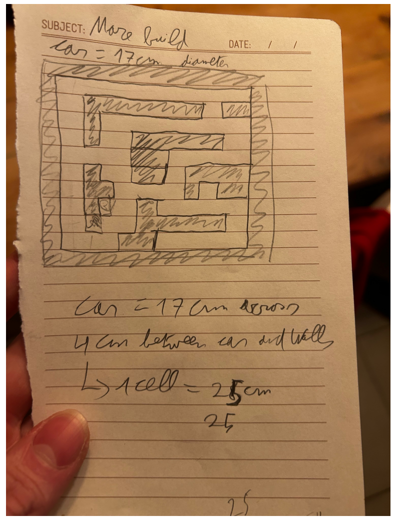

Figure 55: Size Calculations for Maze (Image created by author)

Step 2: Cut the Wood I let the store cut the wooden planks for me to the correct size, as you could see in the prerequisites.

Figure 56: Drilling Wood Frames for Maze (Image created by author)

Step 3: Screw the Wood Together It should turn out like this, repeat this for all the blocks in the maze:

Figure 57: Wooden Frames for Maze (Image created by author)