Exploring the Feasibility of Sim2Real Transfer in Reinforcement Learning

Author Information

Name: Lucas Driessens
Institution: Howest University of Applied Sciences
Course: Research Project
Date: 2024-30-01

Description

This project explores the feasibility of transferring a trained Reinforcement Learning (RL) agent from a virtual environment to the real world, focusing on navigating a maze with a remote-controlled (RC) car.

Abstract

In this research project, I delve into the fascinating realm of artificial intelligence, specifically focusing on reinforcement learning (RL) and its application in real-world scenarios. The crux of my investigation revolves around the challenging question: “Is it possible to transfer a trained RL agent from a simulation to the real world?” This inquiry is particularly examined in the context of maze navigation.

This research is partitioned into sub-questions, which collectively aim to create a comprehensive understanding of the process. Firstly, I explore the various virtual environments available for training a virtual RF-car, seeking the most effective platform for my purposes. Secondly, I delve into identifying the most suitable reinforcement learning techniques for this specific application, considering factors like efficiency, adaptability, and real-world applicability. Lastly, the research seeks to bridge the gap between simulation and reality, investigating the practicality and challenges involved in this transition.

Through this study, I aspire to contribute significantly to the field of AI and robotics, offering insights and methodologies that could potentially advance the implementation of RL in real-world applications. The outcomes of this research could have far-reaching implications, not only in robotics but also in areas where simulation-based training is crucial.

Introduction

The journey of developing autonomous vehicles using reinforcement learning (RL) techniques in virtual environments is marked by continuous learning and adaptation. This paper, originally intended to

showcase successful implementation strategies, has evolved to also highlight the challenges and iterative nature of such projects. The focus remains on the sim2real transfer and the specific challenges encountered in the alignment of an autonomous remote-controlled car.

Background on Reinforcement Learning (RL) Reinforcement Learning (RL) is a paradigm of machine learning where an agent learns to make decisions by interacting with its environment. In RL, the agent seeks to maximize cumulative rewards through a process of trial and error, guided by feedback from its actions. The fundamental elements of RL include the agent, environment, actions, states, and rewards. The RL process can be mathematically described using Markov Decision Processes (MDP) where:

S is a set of states

A is a set of actions

$P(s*t+1|s_t, a_t)$ is the probability that action a_t in state s_t at time t will lead to state $s*t+1$

$R(s*t, a_t)$ is the reward received after transitioning from state s_t to state $s*t+1,$ due to action a_t

Real-World Applications of RL RL has been successfully applied in various fields, including robotics, gaming, healthcare, finance, and autonomous vehicles. One notable example is the use of RL in AlphaGo, developed by DeepMind, which defeated the world champion in the game of Go. In robotics, RL enables robots to learn complex tasks like walking, manipulation, and navigation without explicit programming. In the financial sector, RL algorithms are used for algorithmic trading, optimizing portfolios, and managing risks.

Purpose and Significance of the Study The purpose of this study is to explore the feasibility and challenges of transferring a trained RL agent from a simulated environment to the real world. This transition, known as "sim2real," is particularly examined in the context of maze navigation using a remote-controlled (RC) car. The significance of this research lies in its potential to bridge the gap between theoretical RL models and practical, real-world applications, which is a critical step in advancing the field of AI and robotics.

Overview of the Research Questions The main research question focuses on whether a trained RL agent can be effectively transferred from a simulation to a real-world environment. Sub-questions delve into the selection of virtual environments for training, the identification of suitable RL techniques, the practical aspects of the sim2real transfer, and the evaluation of real-time learning capabilities. These questions aim to comprehensively understand the intricacies involved in applying RL in real-world scenarios.

Main Research Question

Is it possible to transfer a trained RL-agent from a simulation to the real world? (case: maze)

Sub Research Questions

1. Which virtual environments exist to train a virtual RC-car?
2. Which reinforcement learning techniques can I best use in this application?
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.
4. Does the simulation have any useful contributions? In terms of training time or performance?
5. How can the trained model be transfered to the real RC car? (sim2real) How do you need to adjust the agent and the environment for it to translate to the real world?

Methodology

Virtual Environment Design

RCMazeEnv Environment

- **Description**: The RCMazeEnv is a custom class derived from the OpenAI Gym library. It simulates a robotic car navigating through a maze. The environment is designed to replicate real-world physics and constraints within a virtual setting.
- **Maze Structure**:
    - **Starting Position**: Top-left corner of the maze. With the car facing East.
    - **Goal**: Bottom-right corner, representing the maze exit.
    - **Layout**: The maze layout is configurable, allowing for various complexity levels.
- **Robotic Car Specifications**:

- **Movement Actions**: Forward, turn left, turn right.
- **Orientation**: North, East, South, West.
- **Sensors**: Front, left, and right distance sensors to walls.

- **Reward System**:
  - Negative rewards for each step and revisiting positions.
  - Substantial negative reward for hitting a wall.
  - Positive reward for proximity to the goal and reaching the exit.
- **Reset Functionality**: Ability to reset the car to its starting position and reinitialize variables.

Agent Design: Double Deep Q-Network (Double DQN)

- **Algorithm Overview**: The Double DQN algorithm enhances traditional reinforcement learning methods by employing two neural networks, the policy network, and the target network, to reduce overestimation of Q-values.
- **Network Architecture**:
  - **Policy Network**: Selects actions based on the current state.
  - **Target Network**: Provides a stable target for future state evaluation.
  - **Replay Memory**: Stores experiences for learning and optimization.
- **Learning Process**: The agent continually adapts through interaction with the environment, using sensor data to inform movement decisions.

Experimental Setup

- **Environment Setup**:
  - **Overview**: The custom `RCMazeEnv` class, developed based on the OpenAI Gym library, simulates a robotic car navigating through a maze. This environment offers a rich platform for testing and refining reinforcement learning algorithms, focusing on sensor-based navigation and spatial decision-making.
  - **Key Features**:
    1. **Maze Configuration**: A customizable maze layout with a start position at the top-left and a goal at the bottom-right corner.
    2. **Robotic Car Actions**: The car's actions include moving forward, turning left, or right, considering its orientation (North, East, South, West).

- 3. **Sensors**: Equipped with front, left, and right distance sensors for wall detection.
- 4. **Reward System**: Designed to encourage efficiency, penalize wall collisions and revisiting positions, and reward goal proximity and achievement.

$$\mathrm{R} = \begin{cases} -20 & \text{if sensor readings indicate collision} \\ 500 - 200 \times \mathbb{I}(\text{steps} > 1000) & \text{if car\_position equals goal} \\ \frac{50}{\text{distance\_to\_goal}+1} + 50 \times \mathbb{I}(\text{distance\_to\_goal} < \text{previous\_distance}) & \\ \quad -25 \times \mathbb{I}(\text{distance\_to\_goal} > \text{previous\_distance}) & \\ \quad -10 \times \mathbb{I}(\text{car\_position} \in \text{visited\_positions}) - 2 & \text{otherwise} \end{cases}$$

- 5. **Reset Functionality**: Includes a `reset()` method to reinitialize the car's position and variables.
- 6. **Visualization**: A `render()` method for graphical representation of the maze, car, exit, and sensor readings.

- **Rendering Modes**:
  - **training Render Modes:**
    - **Default (Human)**: Utilizes a 2D Pygame rendering for visual output.
    - **Array Mode**: A less resource-intensive mode that displays maze and car positions in the console, facilitating faster training.
  - **Final Render Mode**:
    - **PYopenGL**: A 3D rendering mode that provides a more realistic representation of the maze and car with renders representing the sensor sensors, allowing for a more immersive experience.

- **Agent Setup**
  - **Agent's Role**: The Double DQN agent is central to navigating the RCMazeEnv. It leverages two neural networks (policy and target) to make informed movement decisions based on sensor input, enhancing learning efficiency and effectiveness.
  - **Features**:

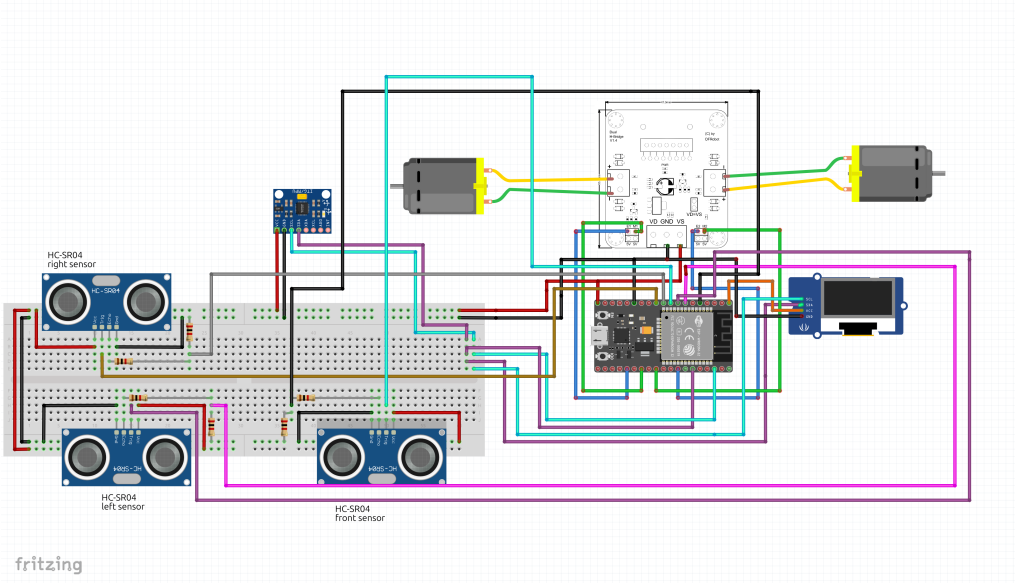
1. **Network Architecture**: Comprises a sequential model with dense layers, tailored for processing sensor data and selecting optimal actions.
2. **Learning Mechanisms**:
   - Utilizes experiences stored in a replay memory for batch learning.
   - Separates action selection (policy network) from Q-value generation (target network) to minimize overestimation bias.
   - Employs periodic updates to the target network to maintain stability.

- **Training Strategy**:
- **Policy Network Training**: Involves fitting the network to batches of experiences, updating Q-values based on both the policy and target network predictions.
- **Action Prediction**: Employs the policy network for action prediction during the maze navigation.
- **Target Network Updating**: Ensures the target network's weights are periodically aligned with the policy network.
- **Operational Flow**:
- The agent iteratively interacts with the environment, making movement decisions based on current sensor data.
- Its performance is continuously monitored and adjusted based on the reward system.

- **Real-World Implementation**
  - **Overview**:
    - The real-world implementation involves assembling an RC robot, designed to navigate through a physical maze using sensor data and reinforcement learning algorithms. This section details the assembly process and hardware setup, crucial for the practical application of the trained reinforcement learning agent.
  - **Components and Assembly**:
    - 1. **Core Components**: The robot is built using an ESP32-WROOM-32 module, motor driver, hc-sro04 ultrasonic sensors, mpu6050 and a mini oled screen.
    - 2. **Assembly Process**:
      - **Base Assembly**: The chassis, provided in the 2WD robot kit, forms the base of the robot.
      - **Motor and Driver Installation**: Motors are attached to the base and connected to the motor driver for movement control.
      - **Sensor Integration**: Ultrasonic sensors (HC-SR04) are mounted on the robot for distance measurement.
      - **Microcontroller and Power Setup**: The ESP32 module is setup with a 1860 li ion battery as a power source to control the robot and process sensor data.
  - **Wiring and Configuration**
    - 2. **ESP32 Module Wiring**: The ESP32 module is wired to the motor driver for directing the movement of the robot based on the agent's decisions. The mini olED screen

is connected to the ESP32 module for displaying the IP address. The MPU6050 accelerometer is connected to the ESP32 module for measuring the car's orientation. The HC-SR04 ultrasonic sensors are connected to the ESP32 module for measuring the distance to the walls.

- **Refer to the wiring diagram below for details**:



- 3. **Programming**: The ESP32 is are programmed to facilitate communication between the agent and the robot. The ESP32 module is programmed to receive commands from the agent and control the robot's movement accordingly. The ESP32 module is also programmed to send sensor data to the agent for decision-making. The web application on the other hand is programmed to send commands from the agent to the ESP32 module and receive sensor data from the ESP32 module, while also rendering the simulation.

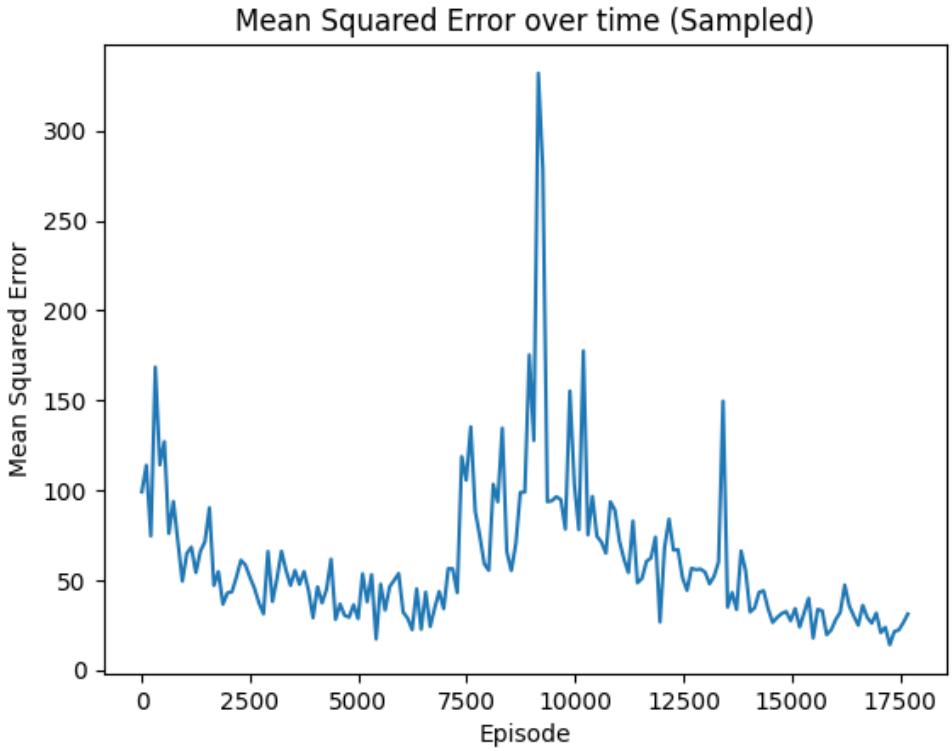
- **Challenges and Adjustments**:
  - **Sensor Calibration**: Fine-tuning sensor readings to match the real-world environment conditions.
  - **Motor Control**: Adjusting motor commands for precise movement in the physical space.
  - **Integration with RL Agent**: Ensuring seamless communication between the robot's hardware and the software agent.

- **Evaluation Metrics**
  - **Simulation Metrics**

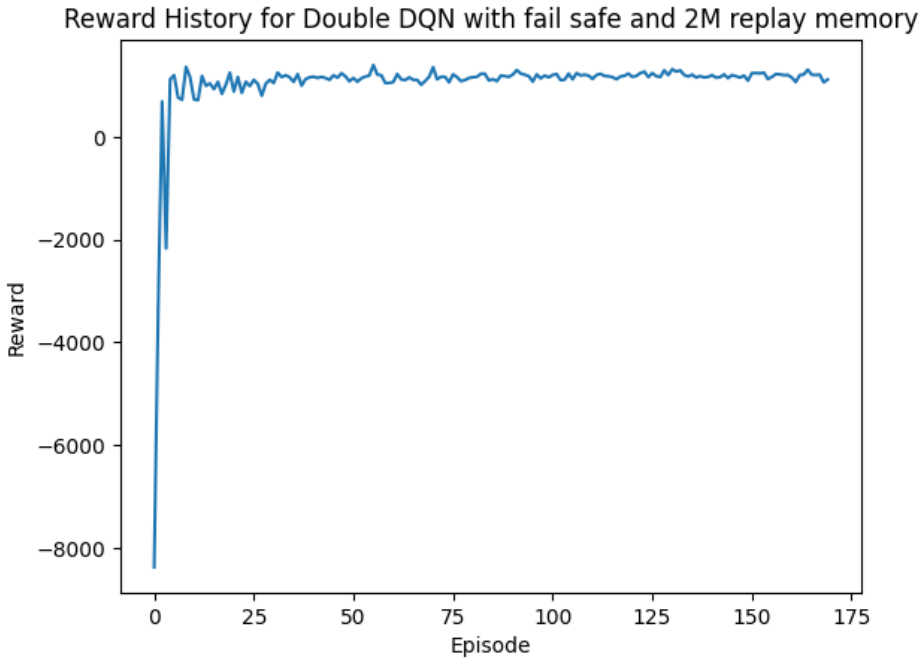
- **Objective**: Assess the agent's ability to solve the maze efficiently in the virtual environment.
- **Episodic Performance**: Analysis of the number of episodes required for consistent maze resolution with optimal rewards.
- **Step Efficiency**: Monitoring the number of steps taken by the agent to complete the maze, indicating efficiency improvements.
- **MSE Loss Measurement**:
  - Formula:

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

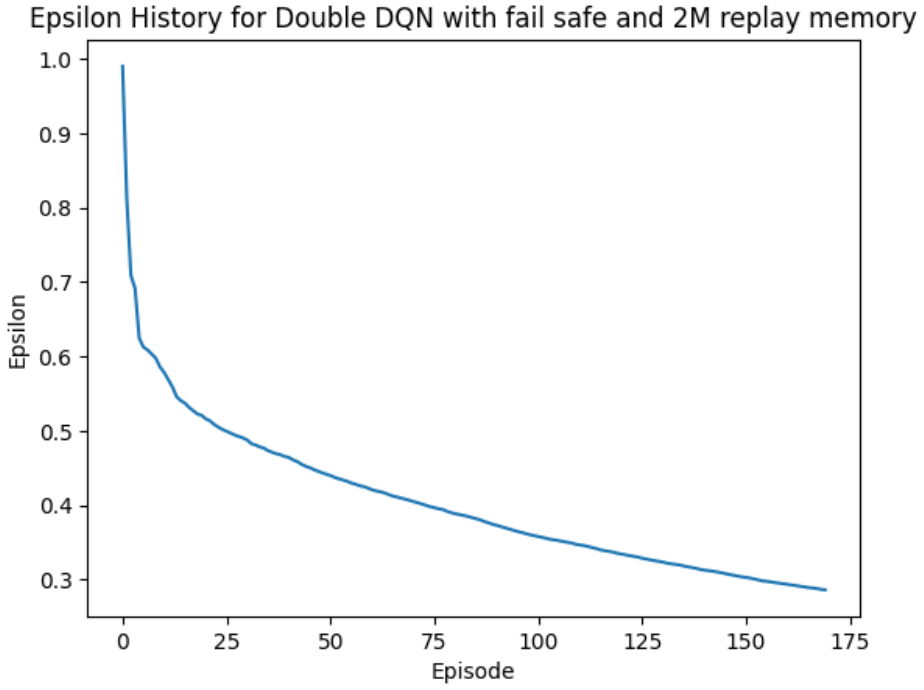
  - Visualization:



- **Reward Trend Analysis**:
  - Chart:



- **Epsilon Decay Tracking**:
  - Chart:



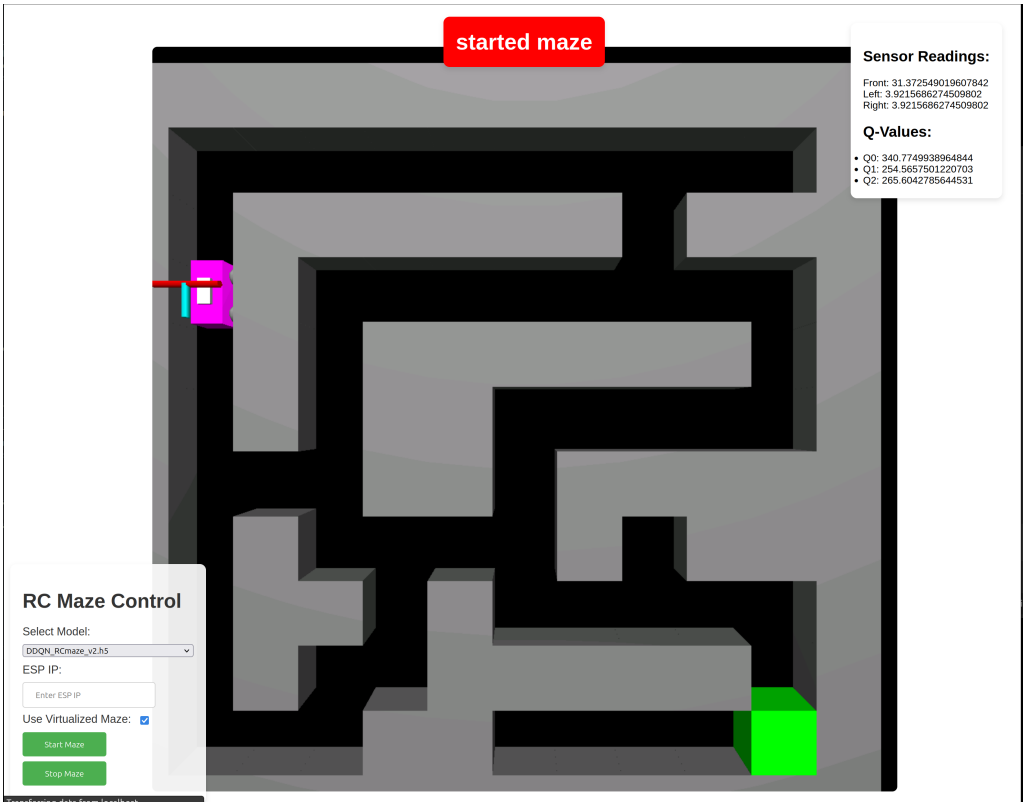
- **Simulation Test Video**:

- Clip: 

  https://github.com/driessenslucas/researchproject/assets/91117911/66539a97-e276-430f-ab93-4a8a5138ee5e

  - **Real-World Metrics**
    - **Maze Navigation**: Visual assessment of the RC car's capability to navigate the real-world maze.
    - **Sensor Data Analysis**: Evaluating real-time sensor data for navigation and collision avoidance.

- **Web Application**
  - **Purpose**: Provides a visualization platform for the simulation and a control interface for real-life testing, acting as a virtual twin of the RC car.
  - **Construction**:
    - **Backend**: Flask for web server and routing.
    - **Simulation Rendering**: PyOpenGL.
    - **Real-Time Communication**: SocketIO for data transfer between the app and the simulation.
  - **Functionality**:
    - **Simulation Streaming**: Capturing and transmitting live simulation snapshots to the web interface.
    - **Data Display**: Real-time visualization of sensor data and Q-values.
    - **User Controls**: Model selection, IP configuration for ESP32, mode selection (simulation or real RC car), and simulation control (start/stop).
  - **Visual and Interactive Interface**:

- Screenshot: 

- Clip:

https://github.com/driessenslucas/researchproject/assets/91117911/b440b295-6430-4401-845a-a94186a9345f

Training Process of the Double DQN Agent

Model Architecture The Double DQN model employed in this research is structured as follows:

```
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)

---
```

This neural network consists of four dense layers, with the output shape and parameter count as detailed above. The final layer outputs three actions corresponding to the movement capabilities of the RC car: moving forward, turning left, and turning right.

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`)**: 0.99993
    - Rate at which exploration probability decreases.
- **Number of Episodes (`EPISODE_AMOUNT`)**: 170
    - Total episodes for training the agent.
- **Replay Memory Capacity (`REPLAY_MEMORY_CAPACITY`)**: 2,000,000
    - Maximum size of the replay buffer.
- **Learning Rate**: 0.001
    - The rate at which the model learns from new observations.

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.

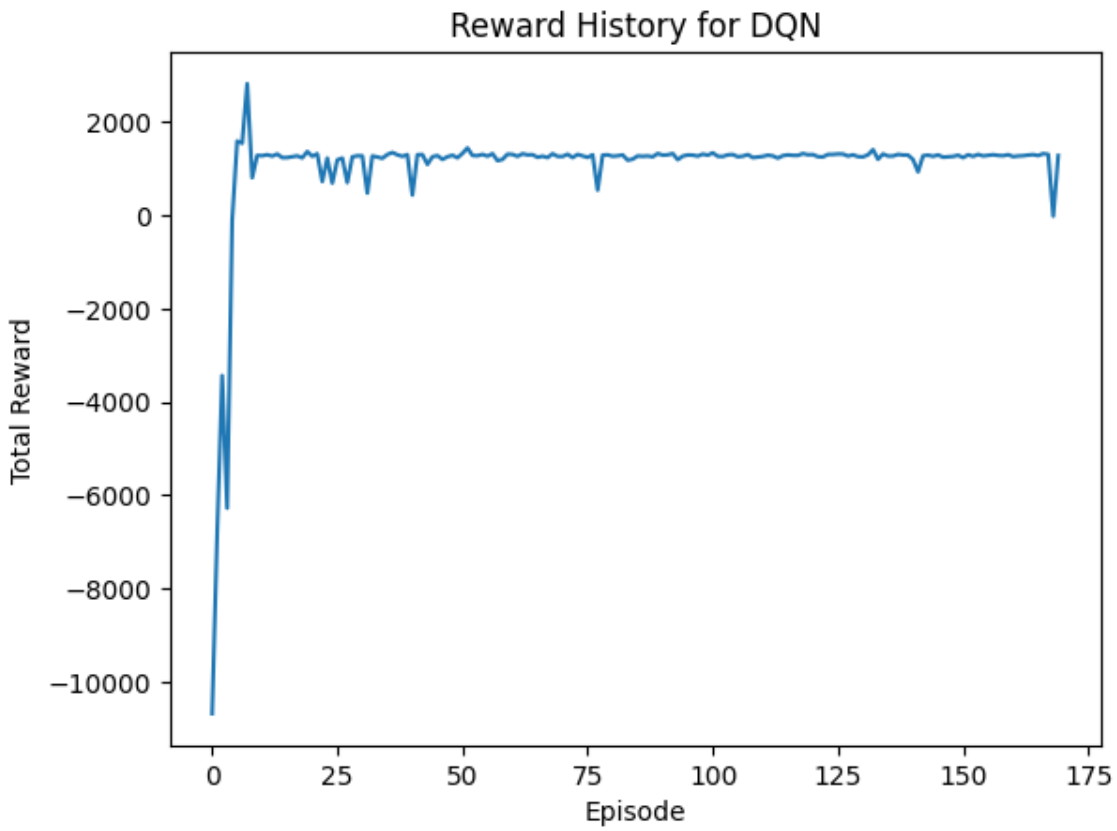
Reinforcement Learning Techniques Overview

1. Deep Q-Network (DQN)

- **Description**: The Deep Q-Network (DQN) combines a deep neural network with a Q-learning framework. It excels in handling high-dimensional sensory inputs, making it ideal for environments demanding detailed interaction.
- **Suitability**: DQN's advanced learning capabilities are tempered by its tendency to overestimate Q-values in complex environments. This limitation could affect its effectiveness in training RC-cars, where environmental dynamics are unpredictable.
- **Integration and Results**:
    - **Visual Representation**:

https://github.com/driessenslucas/researchproject/assets/91117911/a7c5964e-139c-46a1-af79-85280a26c9d2

- **Reward History**:



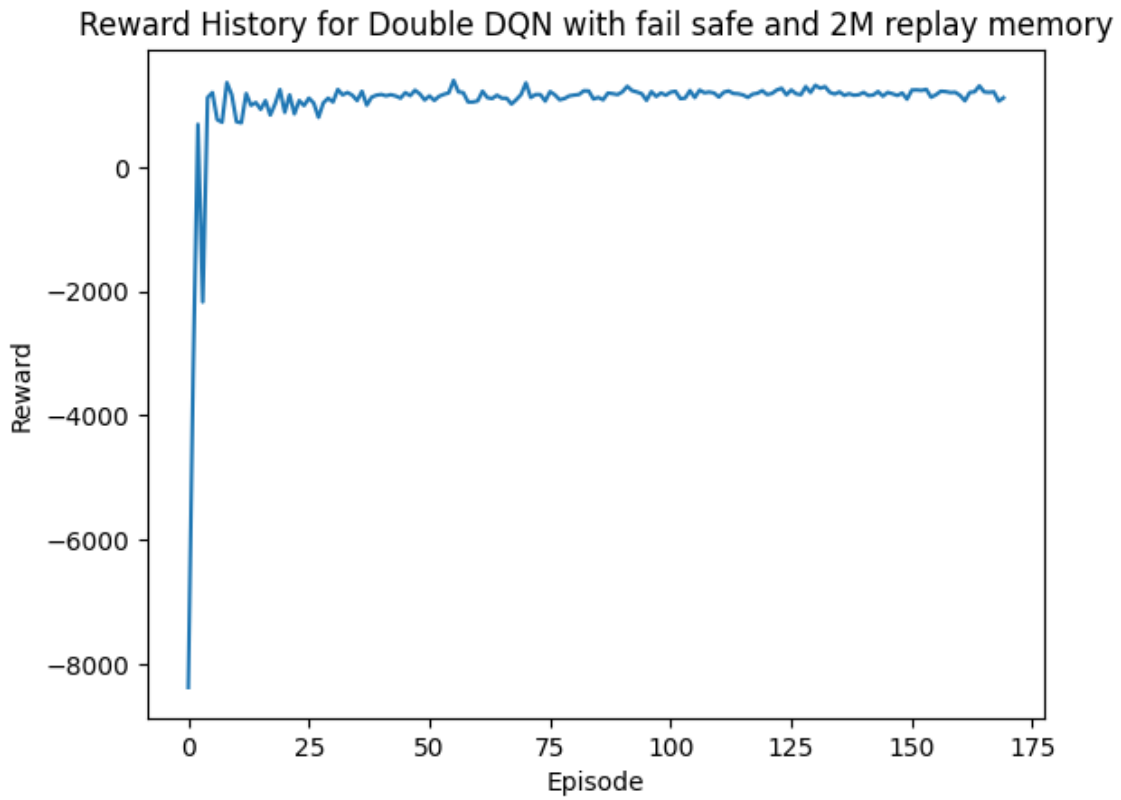
- **Performance**: DQN's performance, while competent, was limited by Q-value overestimation in intricate scenarios.

2. Double Deep Q-Network (DDQN)

- **Description**: The Double Deep Q-Network (DDQN) improves upon DQN by employing two neural networks. This structure effectively reduces overestimation bias by separating action selection from Q-value generation.
- **Reason for Selection**:
  - DDQN's accuracy in Q-value approximation is crucial for navigating complex environments, such as mazes.
  - The RC-car's sensor limitations, which could lead to Q-value overestimations, are better addressed by DDQN.
  - Empirical trials showed DDQN's superior performance in maze navigation tasks.
- **Integration and Results**:
  - **Visual Representation**:

https://github.com/driessenslucas/researchproject/assets/91117911/de50eaf8-49b9-4bf3-8083-8b2bc0963001

- **Reward History**:



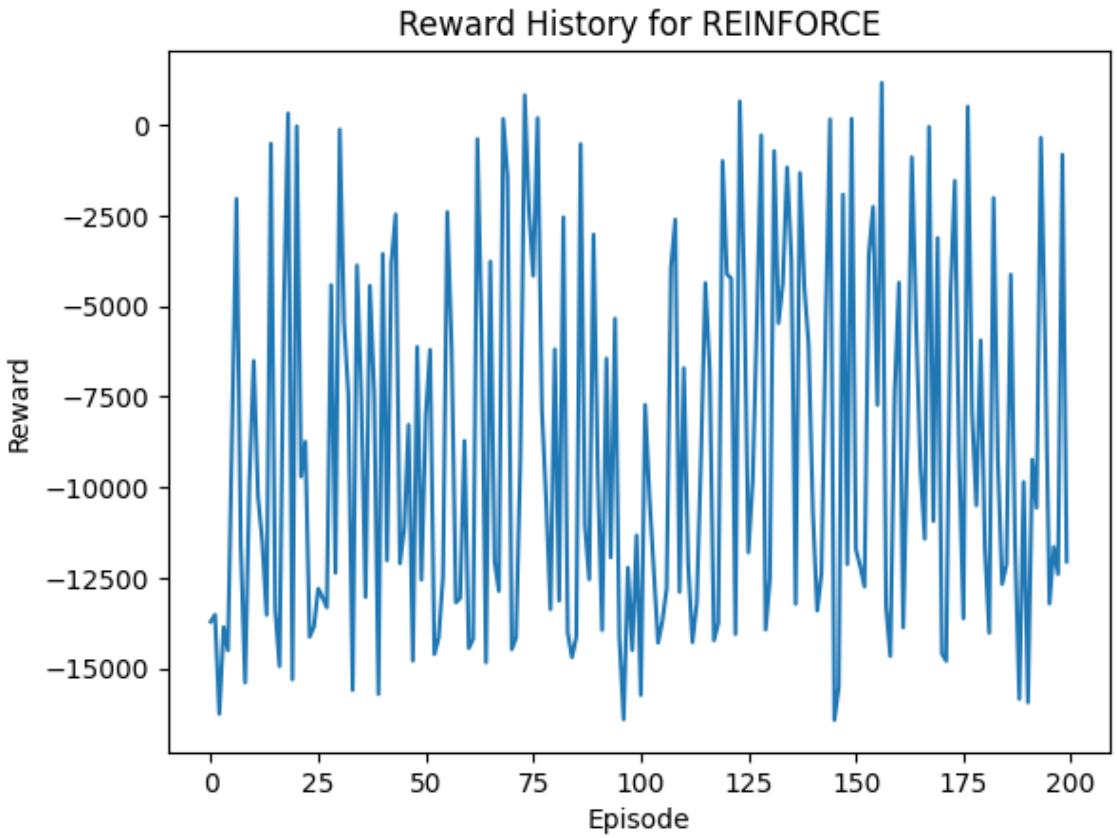
- **Performance**: DDQN solved the environment in an average of 25 steps, compared to DQN's 34 steps, highlighting its efficiency.

3. Proximal Policy Optimization (PPO)

- **Description**: Proximal Policy Optimization (PPO) is a policy gradient method that directly optimizes decision-making policies. It's known for its stability and efficiency in specific RL contexts.
- **Suitability**: PPO's emphasis on policy optimization over value estimation makes it less suitable for RC-car simulations, where accurate Q-value approximation is key.
- **Integration and Results**:
  - **Visual Representation**:

https://github.com/driessenslucas/researchproject/assets/91117911/23a34a9d-7957-4484-a7ce-cfc74c4b9790

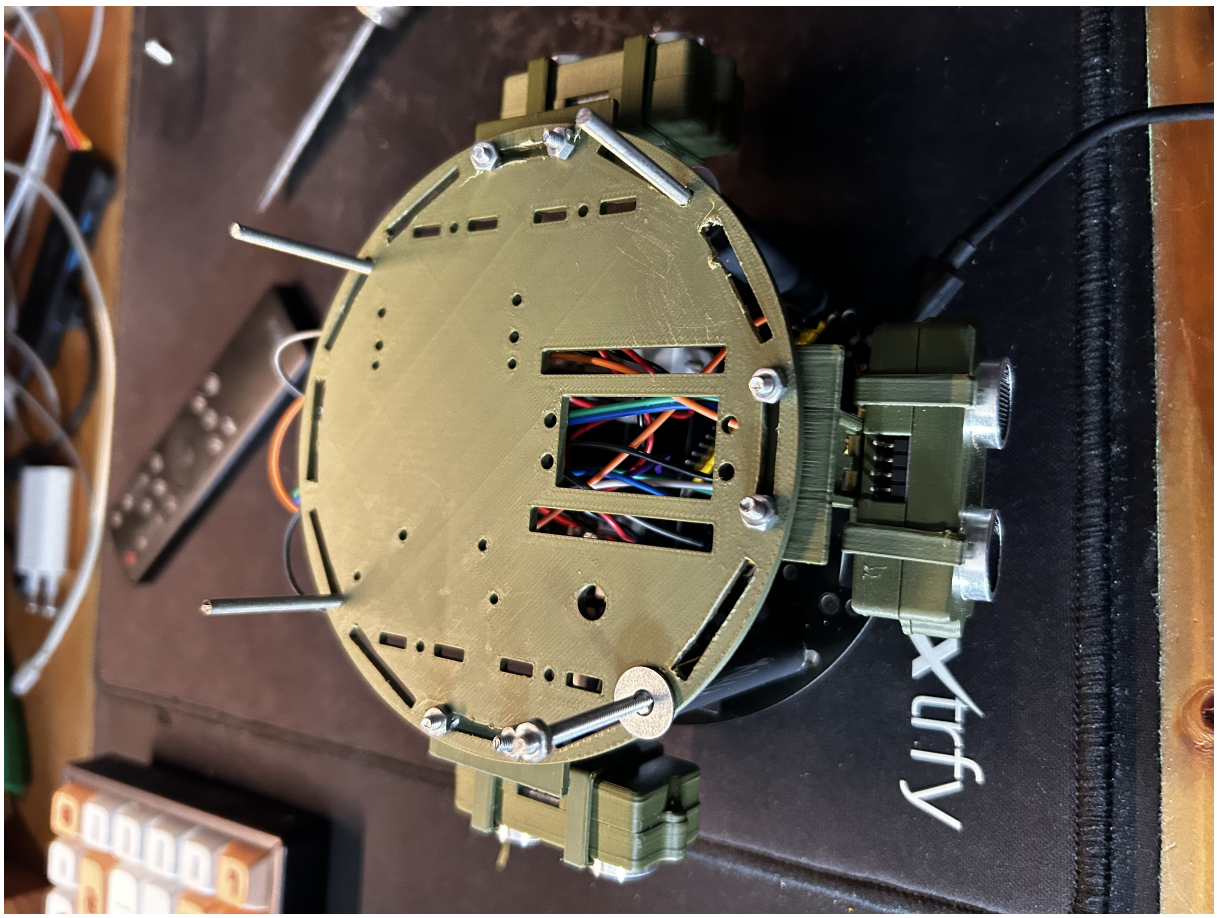
- **Reward History**:



- **Performance**: PPO, while stable, did not align well with the precision requirements for RC-car maze navigation.

Hardware Setup and Assembly

Introduction to Hardware Components This section provides a detailed overview of the hardware components used in the research project, focusing on the assembly and configuration of the RC robot designed for maze navigation.

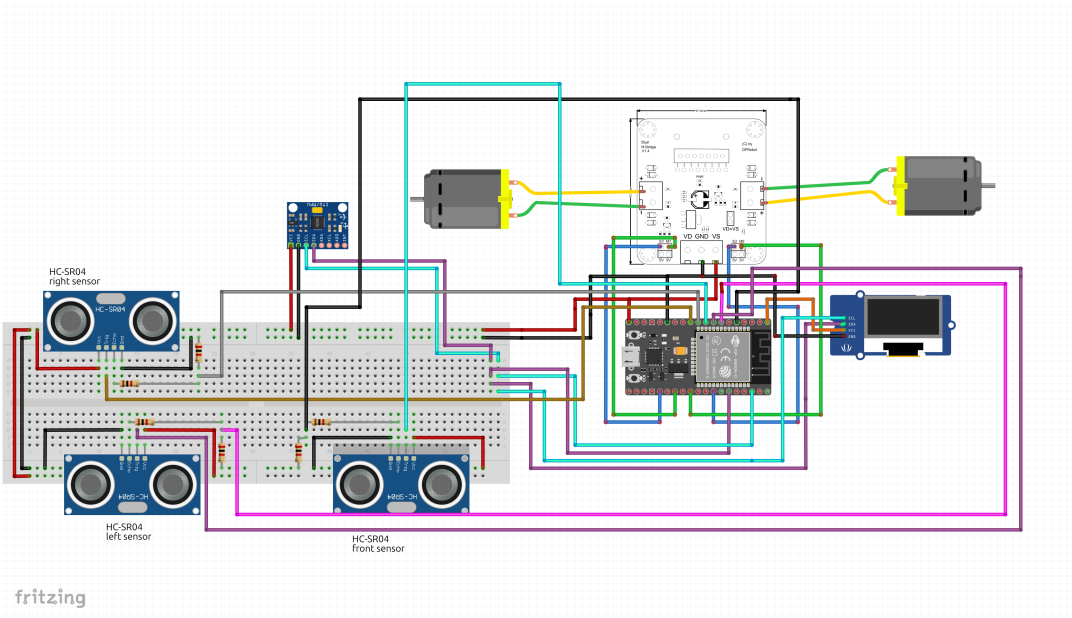


Components List

- **Core Components**:
  - ESP32-WROOM-32 module (Refer to the datasheet at Espressif)
  - 3D printed parts from Thingiverse (hc-sr04, top plate + alternative for the robot kit)
  - Motor Driver - available at DFRobot
  - 2WD robot kit - available at DFRobot
  - Mini OlED screen - available at Amazon
  - Sensors - available at Amazon
  - Battery For ESP 32 - available at Amazon
- **Supplementary Materials**: List of additional materials like screws, wires, and tools required for assembly.
  - 4mm thick screws 5mm long to hold the wood together - available at brico
  - m3 bolt & nuts - available at brico

Wiring Guide

1. **ESP32 Wiring**:



Challenges and Solutions in Implementing RL Techniques and Virtual Environments

Challenge 1: Selection of an Appropriate Virtual Environment

- **Description**: Choosing a virtual environment conducive to effective RC-car training is crucial.
- **Solution**: After evaluating various platforms, **OpenAI Gym** was selected for its simplicity, familiarity from previous coursework, and its focus on reinforcement learning.

Challenge 2: Choosing the Optimal Reinforcement Learning Technique

- **Description**: Selecting the most effective RL technique for training the virtual RC-car.
- **Solution**: Through comparative analysis and empirical testing, the Double Deep Q-Network (DDQN) was identified as the most suitable technique, demonstrating superior performance in navigating complex environments with fewer episodes.

Challenge 3: Sim2Real Transfer - Addressing Movement Discrepancies

- **Description**: Bridging the gap between simulation and real-world in terms of RC-car movement and control.

- **Solution Attempt**: Fine-tuning the frequency of action commands with an async method, waiting for the motor to finish moving or considering a queued action system. Futher more the importance of precise movement in the real world was highlighted, which was not a problem in the simulation.

Challenge 4: alignment Issue and Motor Encoder Implementation

- **Description**: Difficulty in achieving precise straight-line movement in the RC car, with a persistent ~3-degree offset.
- **Solution Attempt 1**: Implementation of motor encoders was pursued to enhance movement accuracy. However, this approach faced the same limitations in achieving the desired precision.
- **Solution Attempt 2**: The motor was replaced with a more powerful one, which initially showed promise in addressing the alignment issue. However, after adding all the other components, the car's weight increased, leading to the same problem. view video
- **Solution Attempt 3**: The use of a MPU6050 accelerometer was explored to measure the car's orientation and adjust the movement accordingly. Even though this approach succeeded to some extent (90 degrees turns were accurate), it was not able to solve the ~3-degree offset issue when moving forward.vdeo of turning 90 degrees video of moving forward
- **Solution**: The final solution was done with removing the RPI5 (previously used for sensor data and running the web app) from the robot all together and using the ESP32 to control both all the sensors and the motors. This allowed for a more lightweight robot, which was able to move more precisely. view video

Challenge 5: Ensuring Consistent and Effective Training

- **Description**: Maximizing training efficiency and performance while maintaining consistency between simulation and real-world scenarios.
- **Solution**: The simulation demonstrated considerable advantages in terms of training efficiency, safety, and computational power, establishing it as an indispensable tool in autonomous vehicle model development.

Challenge 6: Accurate Sensor Data Normalization for Sim2Real Transfer

- **Description**: Aligning sensor data between simulated and real-world environments is critical for model accuracy.

- **Solution**: Implementing specific normalization techniques for both real-world and simulation sensor data ensured consistency and compatibility, enhancing the model's accuracy in real-world applications.

    - 1. **Real-World Sensor Data Normalization:**

      The function `map_distance` normalizes real-world sensor data. It can be represented as follows:

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

      This function keeps distances under 25 cm unchanged and applies a scaling factor of 0.5 to distances beyond 25 cm, adding this scaled value to a base of 25 cm.

    - 2. **Simulation Sensor Data Normalization:**

      The function `normalize_distance` adjusts simulated sensor data to a 0-1 range. Its equation is:

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

      In this function, the distance is first scaled by dividing by `sensor_max_range`. It's then clamped between 0 and 1 before multiplying by 1000 to normalize it within a specific range.

Challenge 7: Integration of Failsafe Mechanisms

- **Description**: Preventing potential collisions and ensuring safe navigation in the real world.
- **Solution**: Development of a failsafe system that prevents forward movement in hazardous situations, retraining the model with this protocol to align real-world behavior with the simulated environment.

Challenge 8: Training Environment and Technique Efficacy

- **Description**: Determining the most effective environment and RL technique for training.
- **Solution**: The DDQN solved the environment more efficiently than DQN, highlighting the importance of technique selection. The simulation provided a safer, more controlled environment for training, reinforcing its selection over real-world training.

Conclusion

This section outlines the practical challenges encountered in the application of reinforcement learning (RL) techniques to autonomous RC cars. The journey began with selecting an appropriate virtual environment, for which OpenAI Gym was chosen 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.

Despite this, transitioning from simulated models to real-world applications unveiled significant discrepancies, especially in movement control and sensor data alignment. Innovative solutions such as the implementation of motor encoders, power adjustments, and accelerometer integration were explored. These methods, however, only partially addressed the main issues. Efforts were also made to normalize sensor data and implement failsafe mechanisms for better real-world alignment.

A notable stride was made by eliminating the Raspberry Pi from the robot's design and instead utilizing the ESP32 module for controlling both the sensors and motors. This adjustment resulted in a more lightweight and precise robot, marking a significant step in overcoming previous challenges.

In essence, this section underscores the iterative and demanding process of applying RL techniques in real-world scenarios. It highlights that while theoretical knowledge forms a crucial base, its practical application necessitates continuous refinement, innovation, and adaptation. The journey through these challenges emphasizes the importance of perseverance and creative problem-solving in the advancing field of autonomous vehicle technology.

Supplementary Materials: Video Demonstrations

Introduction This section provides examples of how I attempted to solve some of the challenges encountered in this research project.

Video 1: mpu6050 90 degree turn

- **Description**: This video demonstrates the use of the MPU6050 accelerometer to measure the car's orientation move until the car has rotated ~90 degrees since the start of the movement. This approach was used in an attempt to address the alignment issues when using a delay to measure the amount of time the car needs to make a 90 degree turn.
- **test 1**:

https://github.com/driessenslucas/researchproject/assets/91117911/32d9e29f-6d5a-4676-b609-2c08923ca1ac

- **test 2**:

https://github.com/driessenslucas/researchproject/assets/91117911/624b40f2-bee8-49f6-961d-1f72ab18fe13

Video 2: mpu6050 to align forward movement

- **Description**: This video demonstrates the use of the MPU6050 accelerometer to measure the car's orientation while driving forward and adjust the movement accordingly. This approach was used in an attempt to address the alignment issues, but it was not able to solve the ~3-degree offset issue when moving forward.
- **test 1**:

https://github.com/driessenslucas/researchproject/assets/91117911/bb9aa643-9620-4979-a70c-ec2826c7dd33

- **test 2**:

https://github.com/driessenslucas/researchproject/assets/91117911/689b590f-3a9a-4f63-ba9c-978ddd08ab53

- **test 3**:

https://github.com/driessenslucas/researchproject/assets/91117911/99da37df-d147-43dc-828f-524f55dc6f70

video 4: New RC-car with encoder and more powerful motor

- **Description**: This video demonstrates the use of a rotary encoder to measure the amount of rotations the wheels have made. This approach was used in an attempt to address the address the ~3 degree offset when moving forward. This approach was looking promising, until adding the other components to the car, which increased the weight of the car, leading to the same problem.
- **test 1**:

https://github.com/driessenslucas/researchproject/assets/91117911/9728e29a-d2fa-48fa-b6e0-e2e1da92228f

- **test 2**:

https://github.com/driessenslucas/researchproject/assets/91117911/b9ce2cc3-85fd-4136-8670-516c123ba442

video 5: Encoder implementation (original RC-car)

- **Description**: This video demonstrates reading out the wheel rotations measured by the rotary encoder. This approach again looked promising, but shortly after this video one of the encoders broke, so no further tests with this specific encoder were done.
- **test 1**:

https://github.com/driessenslucas/researchproject/assets/91117911/ae5129fa-c25f-4f89-92bb-4ee81df9f7a5

video 6: Robot v2

- **Description**: This video demonstrates the final version of the RC-car. This version uses the ESP32 to control both the sensors and the motors. This allowed for a more lightweight robot, which was able to move more precisely.
- **test 1**:

https://github.com/driessenslucas/researchproject/assets/91117911/1773a4f5-8618-4114-ad4c-11781bee4088

Real-World Application and Limitations

Introduction to Sensor and Movement Discrepancies The transition from a simulated environment to real-world application introduces unique challenges, particularly in terms of sensor data interpretation and car movement replication. This section explores these aspects in detail.

Real-World Application

Sensor-Based Navigation In real-world scenarios, the application of sensor-based navigation, as developed in the simulation, can significantly enhance the capabilities of autonomous vehicles. This technology can be pivotal in environments where precision and adaptability are crucial, such as in urban traffic management or automated delivery systems.

Impact on Autonomous Vehicle Movement The insights gained from the simulation regarding vehicle movement can inform the development of more sophisticated autonomous vehicle behavior, particularly in navigating complex, dynamic environments. This knowledge is invaluable for improving the safety and efficiency of autonomous transportation systems.

Limitations

Sensor Data Discrepancies One major challenge lies in the discrepancies between sensor data in the simulation and the real world. These differences can affect the accuracy of the navigational algorithms and, consequently, the vehicle's ability to make real-time decisions.

Movement Replication Challenges Replicating the precise movements of the simulated car in a real-world setting poses significant challenges. Factors such as surface texture, vehicle weight, and mechanical limitations can lead to variations in movement that are not present in the controlled environment of the simulation.

Practical Considerations Practical implementation of these findings necessitates considering variables such as sensor calibration, environmental factors, and hardware capabilities. Overcoming these challenges is essential for the successful application of sim2real transfer in autonomous vehicle navigation.

Conclusion The transition from simulation to real-world application in autonomous vehicle navigation, especially regarding sensor usage and car movement, presents both promising opportunities and significant challenges. Addressing these discrepancies is key to harnessing the full potential of sim2real transfer in practical, real-world scenarios.

Credits

I would like to thank my coach and supervisor, Gevaert Wouter for his guidance and support throughout this research project.

Sources and Inspiration

[1] L. Wan, H. Zhong, J. Xu, and Z. Pan, "Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization," Appl. Sci., vol. 13, no. 11, Art. no. 6400, Jun. 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/11/6400

[2] *Self Driving and Drifting RC Car using Reinforcement Learning*, (Aug. 19, 2019). Accessed: Jan. 08, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=U0-Jswwf0hw

[3] *Reinforcement Learning with Multi-Fidelity Simulators – RC Car*, (Dec. 30, 2014). Accessed: Jan. 08, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=c/_d0Is3bxXA

[4] S. Aggarwal and P. Kumar, “Optimization Maze Robot Using A* and Flood Fill Algorithm,” presented at the 2017 International Conference on Computing, Communication and Automation (ICCCA), Noida, 2017, pp. 1330-1334, [Online]. Available: https://www.researchgate.net/publication/319943074_Optimization_Maze_Robot_Using_A_and_Flood_Fill_Algorithm. [Accessed: Jan. 24, 2024].

[5] “Open Labyrinth mission. python coding challenges - Py.CheckiO,” Py.CheckiO - games for coders. Accessed: Jan. 08, 2024. [Online]. Available: https://py.checkio.org/en/mission/open-labyrinth/share/574bd1ded68c9705c5d6f07c6206be12/

[8] M. A. Dharmasiri, “Micromouse from scratch| Algorithm- Maze traversal|Shortest path|Floodfill,” Medium. Accessed: Jan. 08, 2024. [Online]. Available: https://medium.com/@minikiraniamayadharmasiri/micromouse-from-scratch-algorithm-maze-traversal-shortest-path-floodfill-741242e8510

[7] B. Guta, “Gym2Real: Robust Policies for the Real World,” [Online]. Available: https://bguta.github.io/assets/Gym2Real_Capstone_Project_Report.pdf. [Accessed: Jan. 4, 2024].

[8] FinFET, “FinFetChannel/RayCastingPythonMaze.” Nov. 15, 2023. Accessed: Jan. 08, 2024. [Online]. Available: https://github.com/FinFetChannel/RayCastingPythonMaze

[9] D. Li, “DailyL/Sim2Real_autonomous_vehicle.” Nov. 14, 2023. Accessed: Jan. 08, 2024. [Online]. Available: https://github.com/DailyL/Sim2Real_autonomous_vehicle

[10] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, “D4RL: Datasets for Deep Data-Driven Reinforcement Learning.” arXiv, Feb. 05, 2021. Accessed: Jan. 08, 2024. [Online]. Available: http://arxiv.org/abs/2004.07219

[11] M. H. Miryoosefi, M. Brubaker, J. Kober, and A. G. Schwing, “Reinforcement Learning with Videos: Combining Offline Observations with Interaction,” arXiv:2004.07219 [cs.LG], Apr. 2020. [Online]. Available: https://arxiv.org/pdf/2004.07219.pdf