Research Project

Author Information

Name: Lucas Driessens
Institution: HOWEST Kortrijk
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

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 RF-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?
6. Can Real-time learning be implemented?

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.
  - **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

- **Training Process**:
  - Detailed steps on how the agent was trained, including the number of episodes, learning rates, and other relevant parameters.
  - Use of Jupyter Notebook for documenting the training process (reference to the notebook provided in the README).
- **Real-World Implementation**:
  - Description of the hardware setup, including the RC car and maze specifications.
  - Steps for transferring the trained model to the physical RC car.
- **Evaluation Metrics**:
  - Criteria for measuring the success of the RL agent in navigating the maze.
  - Comparison of virtual and real-world performance.
- **Web Application**:
  - Development of a Flask-based web app for visualizing the RC car and maze environment.
  - Features of the app, including simulation and real-world control options.

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.

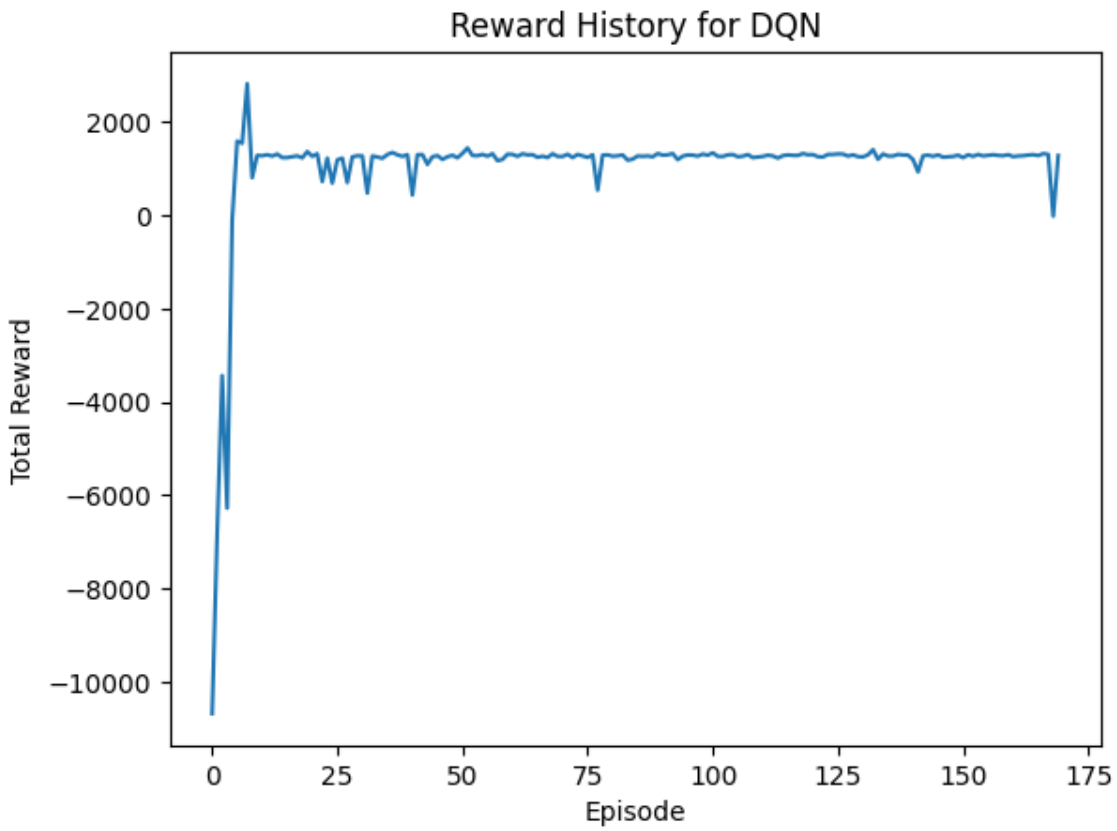
Reinforcement Learning Techniques Overview

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 RF-cars, where environmental dynamics are unpredictable.
- **Integration and Results**:
  - **Visual Representation**:

  - **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 RF-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**:

- **Reward History**:

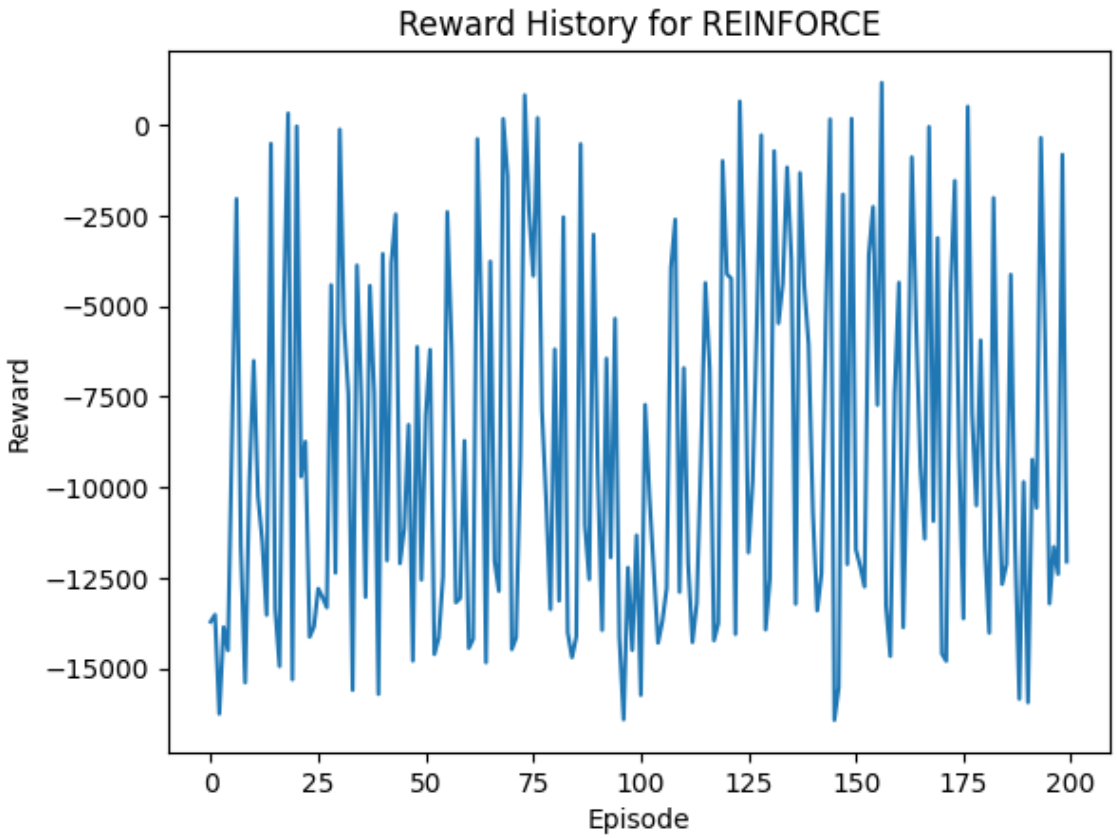
- **Performance**: DDQN solved the environment in an average of 25 steps, compared to DQN's

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 RF-car simulations, where accurate Q-value approximation is key.
- **Integration and Results**:
  - **Visual Representation**:

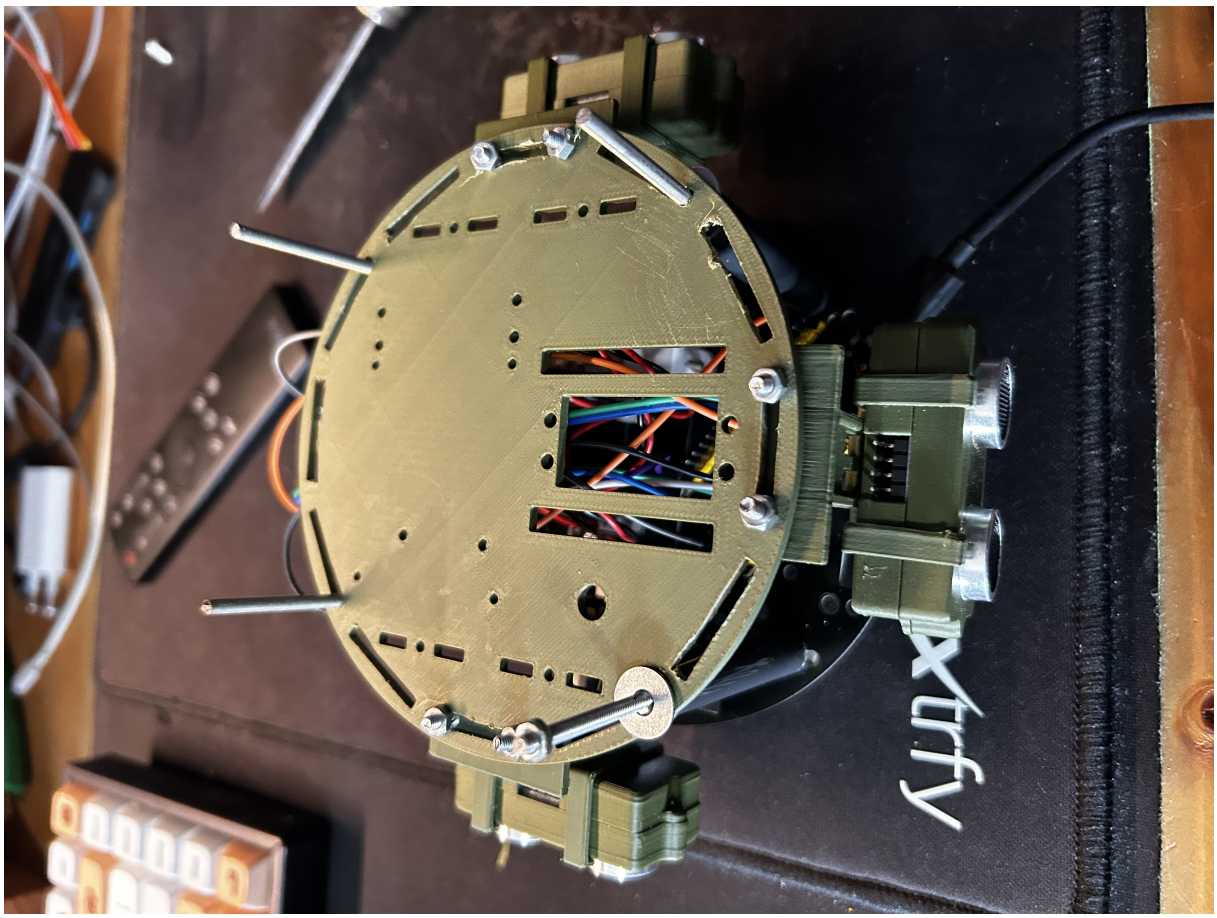
  - **Reward History**:



- **Performance**: PPO, while stable, did not align well with the precision requirements for RF-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.

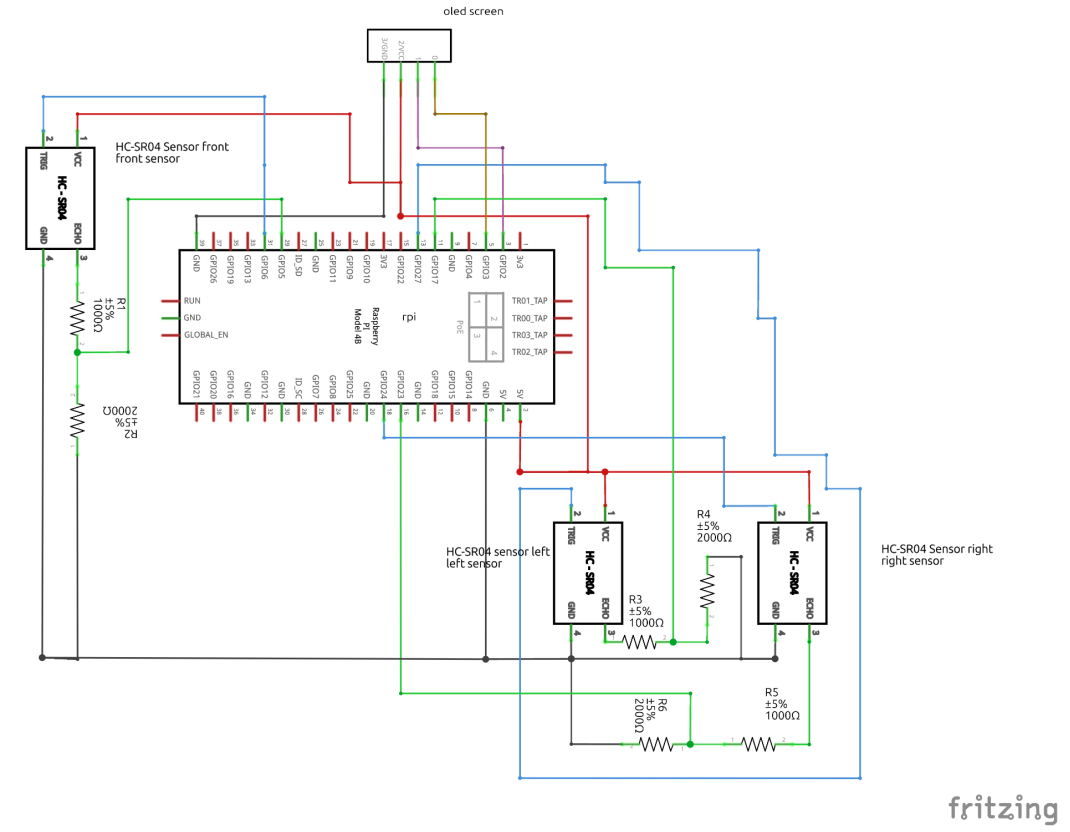


Components List

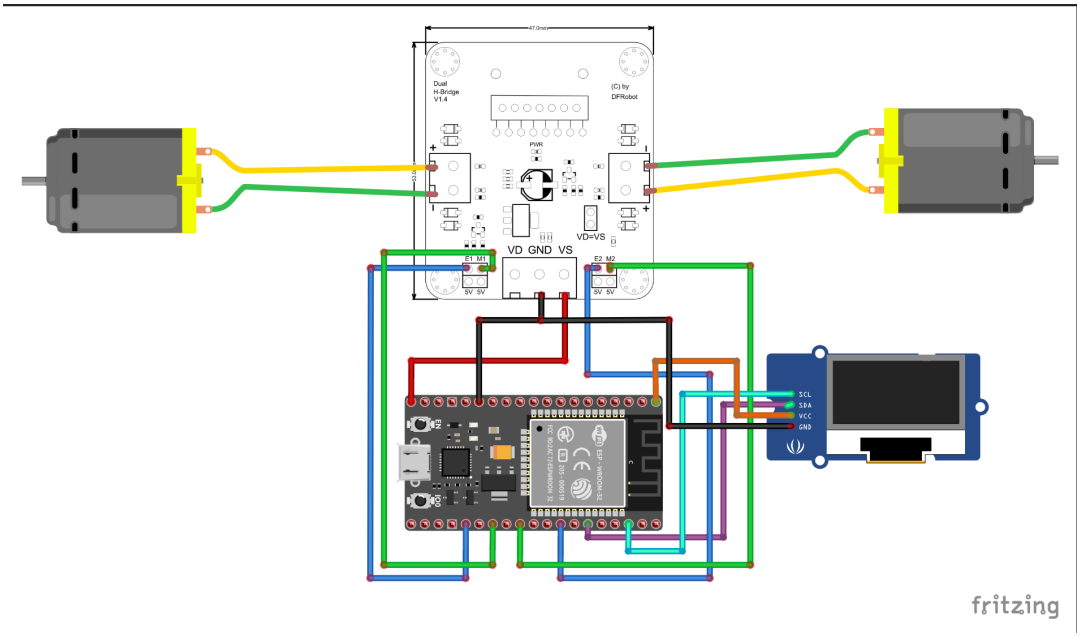
- **Core Components**:
  - Raspberry Pi 5 8gb ram - Available at KiwiElectronics
  - ESP32-WROOM-32 module (Refer to the datasheet at Espressif)
  - 3D printed parts from Thingiverse (hc-sr04, top plate + alternative for the robot kit)
  - Motor Driver - available at DFRobot
  - 2WD robot kit - available at DFRobot
  - Mini OlED screen - available at Amazon
  - Sensors - available at Amazon
  - Battery For RPI 5 - available at Amazon
  - Battery For ESP 32 - available at Amazon
- **Supplementary Materials**: List of additional materials like screws, wires, and tools required for assembly.

Wiring Guide

1. **Raspberry Pi 5 Wiring**:



2. **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 RF-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, particularly for SIM2REAL transfer.

Challenge 2: Choosing the Optimal Reinforcement Learning Technique

- **Description**: Selecting the most effective RL technique for training the virtual RF-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 RF-car movement and control.

- **Solution**: Fine-tuning the frequency of action commands and considering direct motor driver connections or a queued action system. The potential use of a rotary encoder was also identified to enhance real-world movement replication.

Challenge 4: 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 5: 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.

Challenge 6: 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 7: 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

Each challenge encountered necessitated tailored solutions, focusing on the accuracy of RL techniques, the selection of virtual environments, and the adaptability of models for sim-to-real transfer. These solutions underscore the complexity of RL applications in real-world scenarios and highlight the importance of careful consideration in every aspect of model training and implementation.

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

Certainly! Below are the citations for the two sources you provided, formatted in the IEEE style:

[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