# Reinforcement Learning Fundamentals
# Final Project Report

Siddharth Sahu, Tushar Goyal

March 22, 2025

## Contents

# 1 Abstract

The paper discusses VISTA 2.0, a Data Driven Simulator developed by MIT CSAIL. It's designed to train policies in a photo-realistic simulation environment using diverse image augmentation techniques. One key achievement is training agents in this simulator, enabling seamless transition to operating real cars without modifications—a feat unattained in synthesized 3D worlds. The baseline implementation employs a CNN-based network to predict the curvature distribution (mu and sigma) and then giving the curvature to inbuilt step function therefore guiding the agent's actions via API commands to the simulator. Agents adhere to a simple reward function, with loss propagation per episode to manage memory. To overcome limitations, Deep Q Learning is explored, utilizing a replay buffer for continued learning. The paper extensively compares reward structures and learning algorithms on agent performance, assessing adaptability across different environmental conditions like sunny and snowy settings.

# 2 Introduction

## 2.1 Problem being solved

The problem at the heart of this study lies in the realm of reinforcement learning applied to autonomous driving. While synthetic 3D environments have traditionally served as training grounds for autonomous agents, their inability to replicate real-world complexities limits the effectiveness of trained policies when deployed in actual driving scenarios. Addressing this challenge, the research endeavors to bridge the gap between simulation and reality through the development of VISTA 2.0, a Data Driven Simulator. This simulator harnesses real-world data to generate photo-realistic environments, providing a more faithful representation of driving conditions.

One key issue tackled in this study is the enhancement of agent training methodologies within the VISTA 2.0 environment. The baseline approach, employing a convolutional neural network (CNN) to predict curvature parameters from observations, demonstrates promising results but lacks the ability to learn from past experiences due to its episodic memory nature. To overcome this limitation and enable continuous learning, the study proposes the adoption of Deep Q Learning (DQN). By incorporating a replay buffer mechanism, DQN facilitates the retention and utilization of past observations, thereby enhancing the agent's ability to learn and adapt over time.

Furthermore, the research delves into the nuanced exploration of reward structures and learning algorithms. By varying reward functions and assessing their impact on agent performance, the study aims to optimize learning strategies for specific driving tasks. Additionally, comparative analyses between different learning algorithms shed light on their efficacy in training agents for real-world deployment.

A critical aspect of the study involves evaluating the transferability of trained agents across diverse environmental conditions. By simulating transitions between varying weather conditions, such as from sunny to snowy environments, the research assesses the robustness and adaptability of trained policies. This investigation provides insights into the generalizability of learned behaviors, crucial for ensuring the safe and reliable operation of autonomous vehicles in dynamic real-world settings.

## 2.2 Related Literature

Several approaches have been explored in the realm of autonomous driving. Pan et al. propose a Virtual to Real Reinforcement Learning (VR-RL) method, facilitating policy learning through reinforcement learning in virtual environments before transferring to real-world settings. Their novel translation network enables the adaptation of virtual training to real-world scenarios, demonstrating successful adaptation in driving policies. Bojarski et al. introduce an End-to-End Learning system for self-driving cars, employing a convolutional neural network to directly map raw pixel inputs to steering commands. This approach enables the system to autonomously learn internal representations and processing steps, optimizing overall performance without explicit human-defined criteria. Furthermore, O'Kelly et al. address the challenge of rigorous and scalable testing in autonomous vehicle technology. Their simulation framework incorporates deep-learning perception and control algorithms, utilizing adaptive importance-sampling methods to accelerate rare-event probability

evaluation. This approach significantly expedites system evaluation compared to real-world testing, ensuring comprehensive assessment without endangering public safety.

# 3 Formulation & Approach

## 3.1 Formulation

**State Space:**

The State Space is a patch(Region of Interest) of RGB image taken from the camera observation provided by the VISTA simulator. Dimensions: (45x155x3)

**Action Space:**

In the initial implementation the distribution of curvature predicted by the network was used to take action. In our DQN implementation we are using tire angle which is inter convertible with curvature and it can be discretized into 131 discrete angles(action space), ranging from -65°to 65°. This discretization divides the continuous range of possible tire angles into equally spaced intervals, each representing a distinct action that the self-driving car can take. The choice of range (-65°to 65°) for the action space is motivated by the tire angle for the Lexus vehicle spawned in the Vista world.

$$\text{curvature} = \frac{\tan\left(\frac{\pi \cdot \text{tire\_angle}}{180}\right)}{wheelbase}$$

**Reward Function:**

Reward Function for lane following: When the car is on the road (desired task:lane following), the reward is (+1). If the car goes off the road (distance is greater than half the road width), the reward is 0. This represents failure and is also a terminal state.

$$r(x) = \begin{cases} 1 & \text{if } 0 < |x| \leq \frac{\text{width of the road}}{2} \\ 0 & \text{if } |x| > \frac{\text{width of the road}}{2} \end{cases}$$

Reward Function for centerline following: When the car is perfectly on the center line (desired task:centerline following), the reward is the maximum value (+1). For any other state where the car is between the edges but not perfectly centered, the reward is a linear decay based on its distance from the center line. If the car goes off the road, the reward is 0. This represents failure and is also a terminal state.

$$r(x) = \begin{cases} 1 & \text{if } x = 0 \\ 1 - \frac{2 \cdot |x|}{\text{width of the road}} & \text{if } 0 < |x| \leq \frac{\text{width of the road}}{2} \\ 0 & \text{if } |x| > \frac{\text{width of the road}}{2} \end{cases}$$

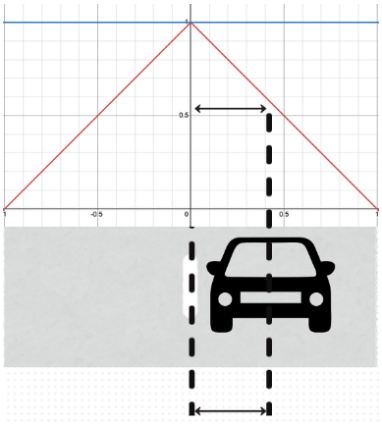


Figure 1: Blue: reward function task 1
Red: reward function task 2

**Terminal Condition:**

The terminal condition for our autonomous driving system is determined by lane departure or excessive rotation. If this distance exceeds half the width of the road, indicating that the vehicle has strayed outside the lane boundaries, the terminal condition is triggered. Additionally, if the vehicle's rotation angle exceeds a predefined threshold leads to terminal.

## 3.2 Approach

**Function Approximation:**

The function approximation used in this code is a Deep Q-Network (DQN). DQN is a type of artificial neural network that approximates the Q-function in reinforcement learning. The Q-function (or Q-value) represents the expected future rewards for taking a specific action in a given state.

**Model Architecture:**

The model architecture consists of convolutional neural networks (CNNs) followed by fully connected layers. Here's a breakdown of the model architecture:

1. **Input Layer**: The input layer receives the observation of the environment, which is a RGB image with dimensions (45, 155, 3).
2. **Convolutional Layers**: Four convolutional layers are used to extract features from the input images. Each convolutional layer applies a set of filters to the input image to detect patterns and features. The filters have varying kernel sizes (5x5 or 3x3) and strides (2), which determine the spatial extent of the convolution operation.
3. **Flatten Layer**: Flattens the output of the last convolutional layer to prepare it for the fully connected layers.
4. **Fully Connected Layers**: Two fully connected (Dense) layers are used for further processing. The first dense layer has 128 units with ReLU activation, and the second dense layer has 131 units with linear activation. The output layer has 131 output units, one for each action from the action space $[-65°, -64°, ..., 64°, 65°]$

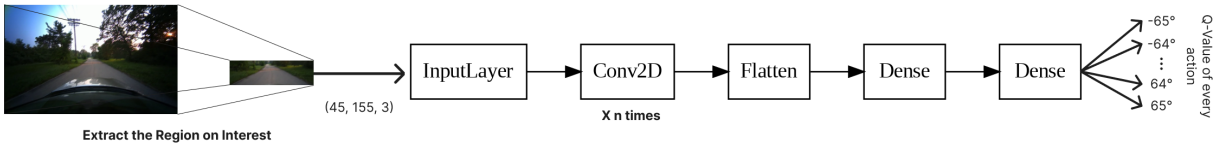



Figure 2: Driving Model Architecture

**Why DQN and CNNs?**

1. **Complex State Space**: The environment state is represented by images, which are complex and high-dimensional. CNNs are well-suited for processing image data and extracting meaningful features.
2. **Non-linearity and Generalization**: CNNs can capture non-linear relationships in the data, allowing the model to generalize across different states of the environment.
3. **Q-Value Approximation**: DQN is used to approximate the Q-values, which represent the expected cumulative rewards for taking actions in a given state. This allows the agent to learn a policy that maximizes long-term rewards over time.
4. **Experience Replay**: The algorithm uses a replay memory to store past experiences (state, action, reward, next state). This helps in breaking the correlation between consecutive samples and improves the stability of training.
5. **Target Network**: A separate target network is used to stabilize training by fixing the target Q-values during updates.

## Additional Notes:

- **Exploration-Exploitation**: The agent balances exploration (trying new actions) and exploitation (selecting actions based on current knowledge) using an epsilon-greedy strategy. Over time, the exploration rate (epsilon) decays, leading to more exploitation of learned policies.

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \times e^{-\text{decay} \times \text{episode}}$$

  where:

  - $\epsilon$ is the exploration rate.
  - $\epsilon_{\text{min}}$ - minimum exploration rate - 0.01
  - $\epsilon_{\text{max}}$ maximum exploration rate - 0.99
  - decay is the decay rate - 0.01
  - episode is the current episode in the learning process.

Overall, the combination of DQN and CNNs is suitable for training an agent to navigate a complex environment based on visual input, such as driving a car in a simulated environment.

# 4 Experimental Setup

## 4.1 Tasks

**Task 1: Lane Following**

This task ensures the agent (car) stays within the designated lane boundaries. It focuses on keeping the car from deviating too far to the left or right and potentially going out of lane. It uses the above mentioned reward function for lane following.

**Task 2: Centerline Following**

This aim to stay centered on the dividing line within the lane. It uses the above described reward function for centerline following.

## 4.2 Dataset

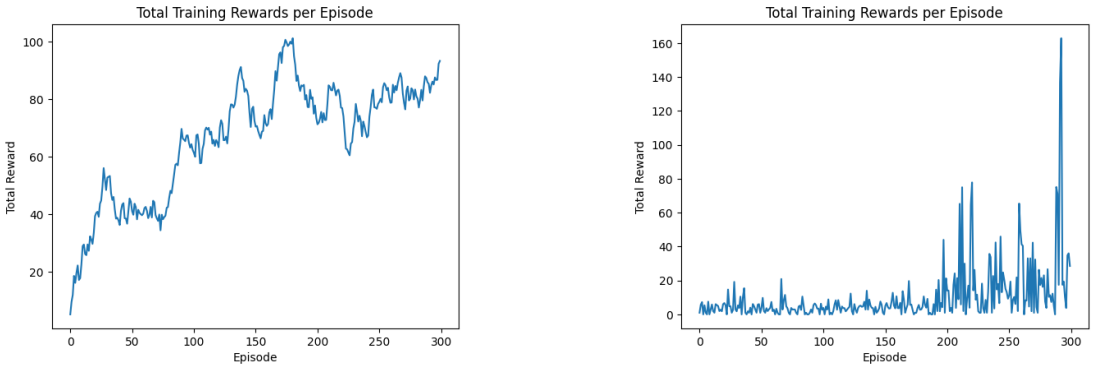
Environments in VISTA are constructed using real-world driving traces collected by humans. A trace refers to the data recorded during a single driving session. Specifically, we'll focus on RGB camera data captured from the driver's perspective as the vehicle navigates the road. This data is gathered while the car is in motion. We have found driving data published by original authors in two different environments: snowy and sunny. The path from which the data is collected is nearby to the MIT Campus. It also provides LiDAR data, which we **don't use** in our training.

Figure 3: (a) Map of the Trace (b) Snowy Trace: Camera Viewpoint (c) Sunny Trace: Camera Viewpoint

## 4.3 Experiments

Train the Deep Q-Network (DQN) using the sunny weather dataset. Evaluate the trained DQN on a separate subset of sunny weather data. Measure performance on average reward per episode. Evaluate the trained DQN on the snowy weather dataset, using the same performance metrics as in the sunny weather test. Compare the performance of the DQN with the baseline and compare performance between sunny and snowy weather conditions.

# 5 Results




(a) Baseline Training Reward Task 2

(b) DQN Training Reward Task 2

Figure 4: Training Reward Comparision

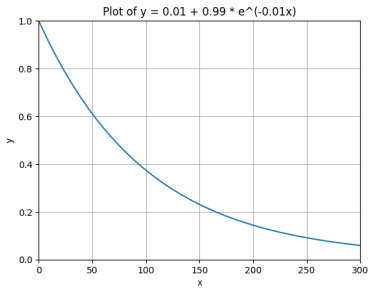



Figure 5: Curve of $\epsilon$ till 300 episodes in DQN

| Model | Training Time (300 Epochs) | Evaluation Reward (Same Environment) |
|---|---|---|
| **Baseline Model** Provided by the Author | 165 mins | 81.2 |
| **(Ours)** Deep Q-Learning | 132 mins | 72.2 |

Table 1: Performance Results on Same Environment as Training (Sunny)

| Model | Evaluation Reward (Different Environment) |
|---|---|
| **Baseline Model** Provided by the Author | 6.4 |
| **(Ours)** Deep Q-Learning | 7.2 |

Table 2: Performance Results on Different Environment (Snowy) from Training (Sunny)

**Youtube Links for DQN Evaluation on Sunny and Snowy Traces**

- **DQN Trained on Sunny Trace — Evaluation on Snowy**
- **DQN Trained on Sunny Trace — Evaluation on Sunny**

# 6 Conclusion

The comparison of training rewards for Task 2, depicted in Figure 4, highlights intriguing insights into the learning dynamics of different models. Notably, the baseline model demonstrates rapid initial learning due to its absence of an exploration component. However, this initial surge in rewards gradually diminishes over time, indicating a plateau in learning effectiveness.

Conversely, our implementation of DQN maintains a high exploration rate, as evidenced by the elevated epsilon values illustrated in Figure 5, persisting until episode 300. This prolonged exploration phase leads to the accumulation of numerous negative experiences, resulting in premature episode terminations, hence a unstable learning graph. With further training with smaller epsilion, the model will be able to further explore the environment and gain better rewards, with also learning from the negative experiences stored in the replay memory, this would make the agent robust to negative conditions. During evaluation the action with max Q value is picked 100% probability.

Remarkably, despite encountering challenges during training, the evaluation phase reveals the DQN's capability to perform comparably to the baseline model. This suggests that the DQN effectively learns appropriate Q-values for actions, ultimately achieving competitive performance levels.

Both the models perform poorly when evaluated in another environment, hinting towards poor generalisation capabilities across scenes.

In this project, we successful setup VISTA Simulator which was out of maintenance since 2 years and made our own fork for the same. We researched through the simulator and find way to discretise the action

space into tire-angle prediction from 131 values, and implemented a CNN based agent using Q-Learning which took in the state (observation) and output the Q Values for each action. This project gave us an insight of how simulators work and this led us to implementation of a famous algorithm with VISTA simulator.

# References


[Boj+16] Mariusz Bojarski et al. *End to End Learning for Self-Driving Cars*. 2016. arXiv: 1604.07316 [cs.CV].

[Pan+17] Xinlei Pan et al. *Virtual to Real Reinforcement Learning for Autonomous Driving*. 2017. arXiv: 1704.03952 [cs.AI].

[OKe+18] Matthew O'Kelly et al. "Scalable End-to-End Autonomous Vehicle Testing via Rare-event Simulation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/653c579e3f9ba5c03f2f2f8cf4512b39-Paper.pdf.