# Assignment 2 - Deep Racer and Stop Sign Recognition

## Task 1: AWS DeepRacer Model

Version 1: **StereoLiDAR**

Training configuration:

**Training configuration**

**Race type**
Object avoidance

**Object avoidance rules**
Collision penalty: 5 seconds
Number of objects: 6
Object positions: --

**Environment simulation**
2022 re:Invent Championship - Counterclockwise

**Reward function**
Show

**Sensor(s)**
Lidar, Stereo camera

**Action space type**
Continuous

**Action space**
Speed: [ 0.5 : 1 ] m/s
Steering angle: [ -30 : 30 ] °

**Framework**
Tensorflow

**Reinforcement learning algorithm**
PPO

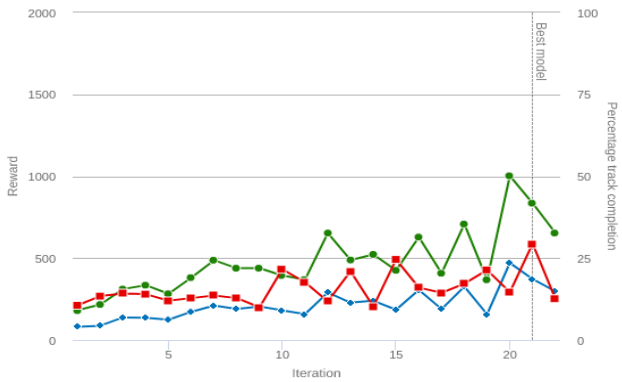
| Hyperparameter | Value |
|---|---|
| Gradient descent batch size | 64 |
| Entropy | 0.01 |
| Discount factor | 0.999 |
| Loss type | Huber |
| Learning rate | 0.0003 |
| Number of experience episodes between each policy-updating iteration | 20 |
| Number of epochs | 10 |

Reward Function: **reward_stereo_LiDAR.py**

Reward Policy:

- ❖ Reward if the agent stays inside the two borders of the track
- ❖ Give a higher reward if the car is closer to the center line and vice versa
- ❖ Penalize reward if the car is steering too much, i.e., zig-zagging
- ❖ Penalize if the agent is too close to the next object
- ❖ Weighted final reward

Reward Graph:




**Stop condition**
Maximum time
03:00:00 / 03:00:00

Version 2: **StereoLidar with reduced min-speed**

Training configuration:

**Training configuration**

**Race type**
Object avoidance

**Object avoidance rules**
Collision penalty: 5 seconds
Number of objects: 3
Object positions: --

**Environment simulation**
Expedition Super Loop - Counterclockwise

**Reward function**
Show

**Sensor(s)**
Camera, Lidar

**Action space type**
Continuous

**Action space**
Speed: [ 0.3 : 1 ] m/s
Steering angle: [ -30 : 30 ] °

**Framework**
Tensorflow

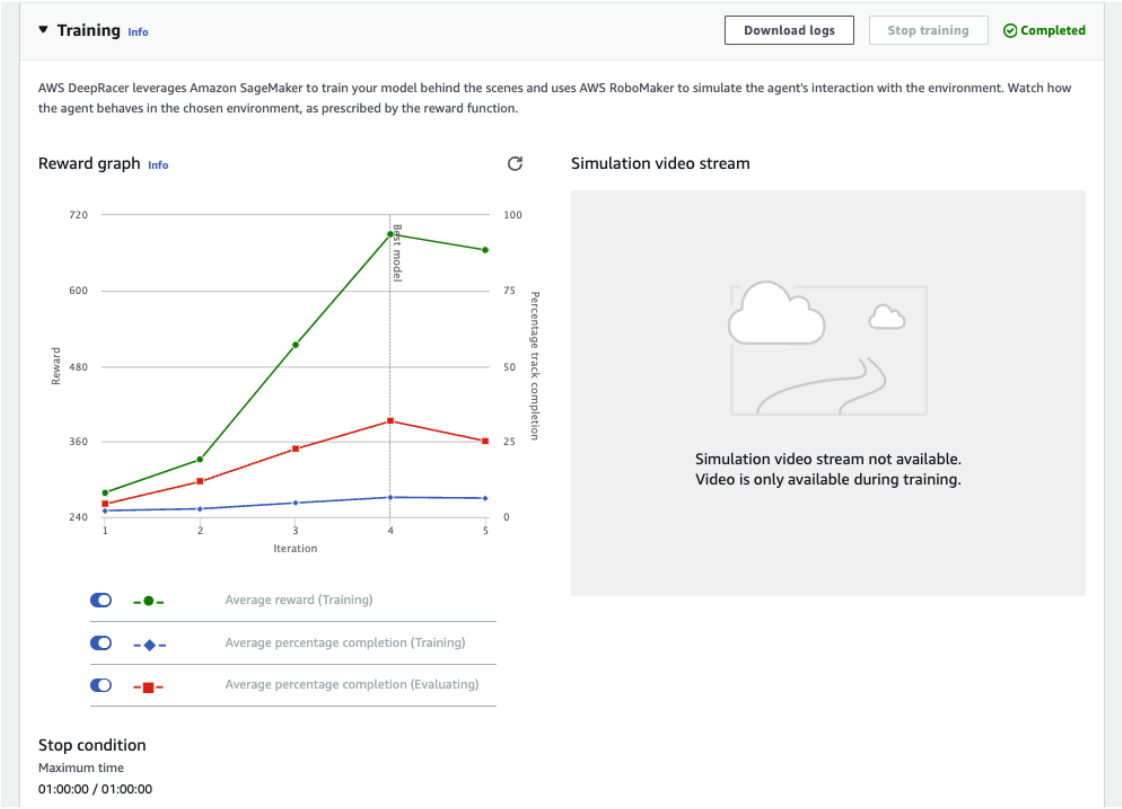
**Reinforcement learning algorithm**
PPO

| Hyperparameter | Value |
|---|---|
| Gradient descent batch size | 64 |
| Entropy | 0.01 |
| Discount factor | 0.999 |
| Loss type | Huber |
| Learning rate | 0.0003 |
| Number of experience episodes between each policy-updating iteration | 20 |
| Number of epochs | 10 |

RewardFunction: model3_reward.py

Reward Policy:

- Reduced minimum speed for obstacle avoidance
- Reduced reward for the distance between obstacles
- Increased reward for staying close to the middle of the lane



Version 3: **Camera Only**

Training configuration:

**Training configuration**

**Race type**
Time trial

**Environment simulation**
A to Z Speedway - Clockwise

**Reward function**
Show

**Sensor(s)**
Camera

**Action space type**
Continuous

**Action space**
Speed: [ 0.5 : 1 ] m/s
Steering angle: [ -30 : 30 ] °

**Framework**
Tensorflow

**Reinforcement learning algorithm**
PPO

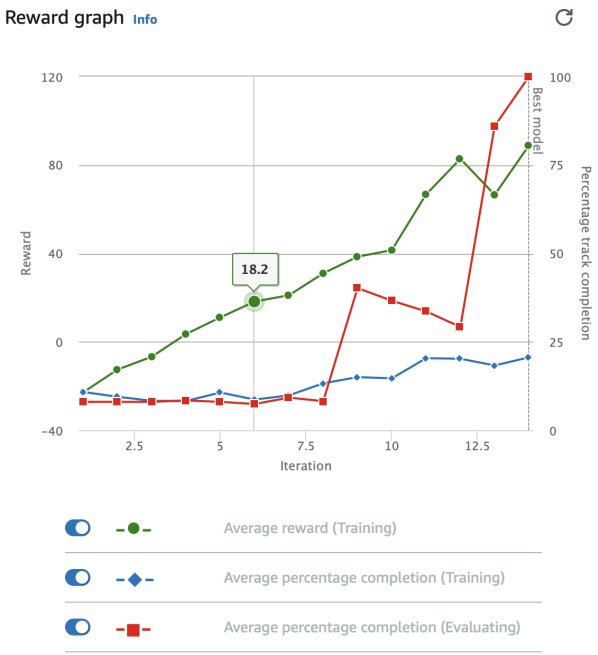
| Hyperparameter | Value |
|---|---|
| Gradient descent batch size | 64 |
| Entropy | 0.01 |
| Discount factor | 0.999 |
| Loss type | Huber |
| Learning rate | 0.0003 |
| Number of experience episodes between each policy-updating iteration | 20 |
| Number of epochs | 10 |

Reward Function: **reward_stereo_LiDAR.py**

Reward Policy:

- ❖ On-track speed rewarded.
- ❖ Center distance penalized.
- ❖ Speed bonuses for acceleration.
- ❖ Waypoint following earns rewards.
- ❖ Off-track incurs penalties.

Reward Graph:




**Stop condition**
Maximum time
01:00:00 / 01:00:00

Version 4: **Camera, LiDAR**

Training configuration:

**Training configuration**

**Race type**
Object avoidance

**Object avoidance rules**
Collision penalty: 5 seconds
Number of objects: 3
Object positions: Placed at 25%, 50%, and 75% of track length.

**Environment simulation**
A to Z Speedway - Clockwise

**Reward function**
Show

**Sensor(s)**
Camera, Lidar

**Action space type**
Continuous

**Action space**
Speed: [ 0.5 : 1 ] m/s
Steering angle: [ -30 : 30 ] °

**Framework**
Tensorflow

**Reinforcement learning algorithm**
PPO

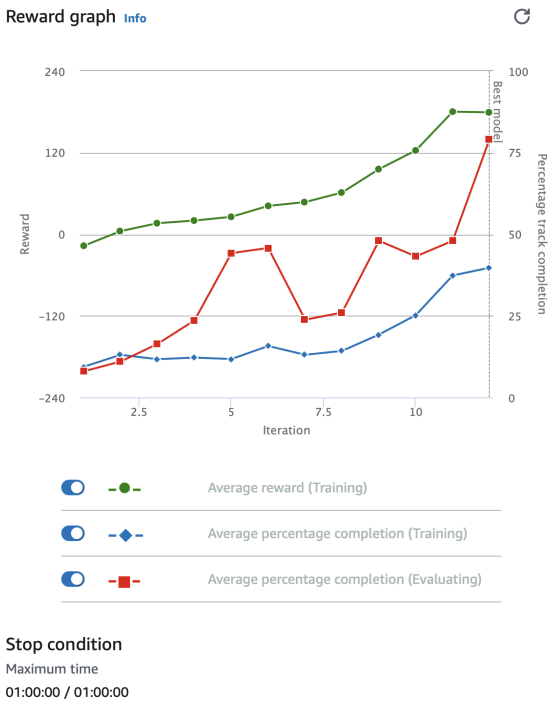
| Hyperparameter | Value |
|---|---|
| Gradient descent batch size | 64 |
| Entropy | 0.01 |
| Discount factor | 0.999 |
| Loss type | Huber |
| Learning rate | 0.0003 |
| Number of experience episodes between each policy-updating iteration | 20 |
| Number of epochs | 10 |

RewardFunction: reward_Camera_LiDAR.py

Reward Policy:

- On-track speed rewarded.
- Center distance penalized.
- Speed bonuses for acceleration.
- Waypoint following earns rewards.
- Off-track incurs penalties.

Reward Graph:

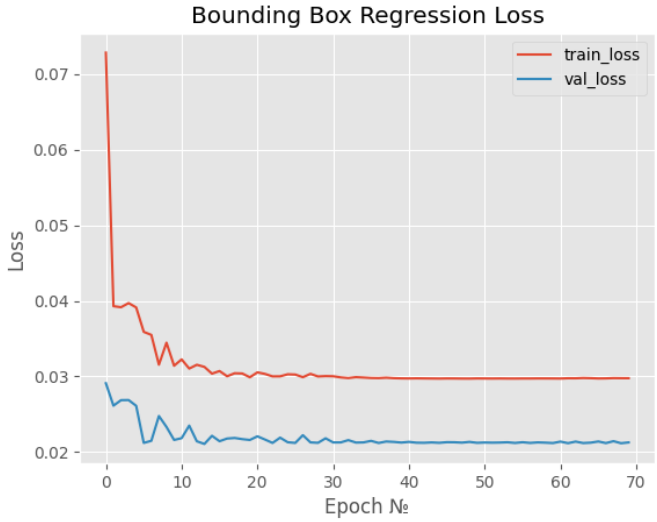


## Task 2: Stop-sign Detection DL Model custom training on a dataset

DataSet: **data_cal.zip**
Source Code: **bounding_box_reg.py**
Model Name: VGG16 from Tensorflow
Trainable Model: False
Number of layers in the model:  24
Algorithm:  Bounding Box Regression
Total Images: 64
Train_images: 90%
Test_images: 10%
Total params: 2264389 (8.64 MB)
Trainable params: 2230277 (8.51 MB)
Non-trainable params: 34112 (133.25 KB)
Optimization: stochastic gradient descent method
Loss Function: Mean Squared Error (MSE)
HyperParameter:

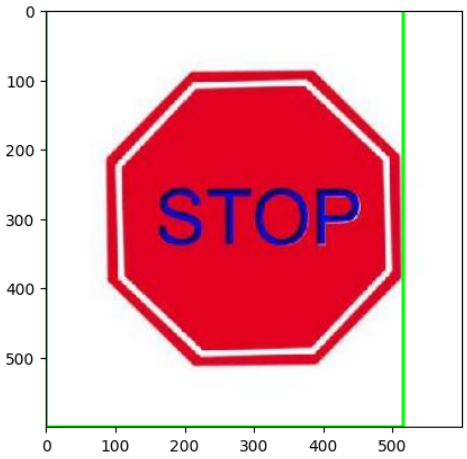
LEARNING_RATE = 1e-4
EPOCHS = 70
BATCH_SIZE = 32

Convergence Curve:



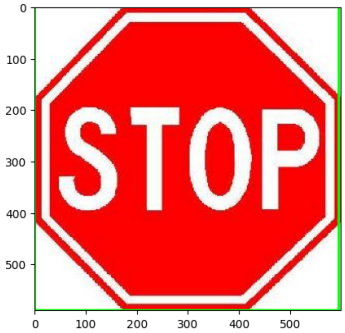
Test input-output image:


Sample_Input_1



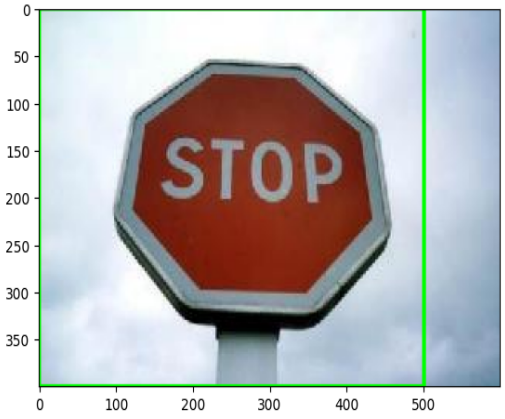

Output_1


Sample_Input_2




Output_2


Sample_Input_3




Output_3


Sample input 4


output_4

## Task 3: Adversarial Examples for Camera-Based Detection

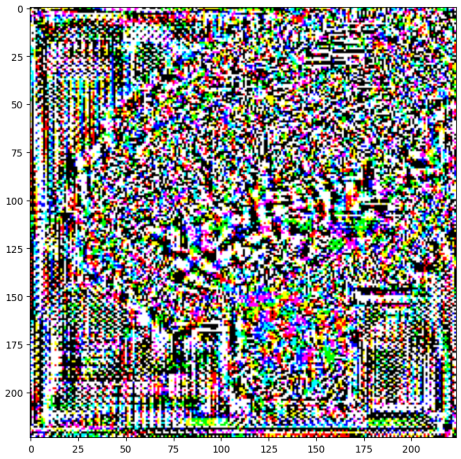
Source code: **adverserial.py**

Model: MobileNetV2 pretrained on ImageNet

Epsilon values = [0, 0.01, 0.1, 0.15]

Test Case 1:


Input Image

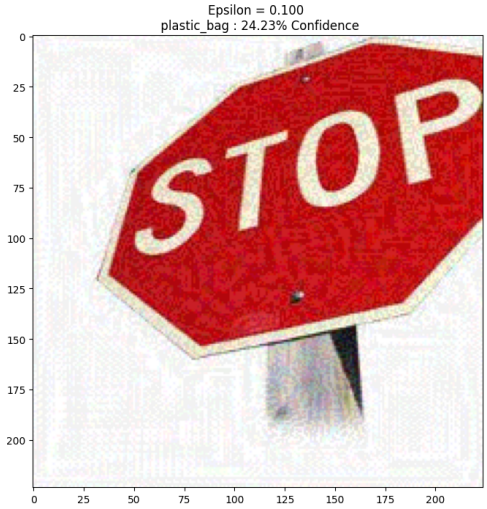



Perturbed Image

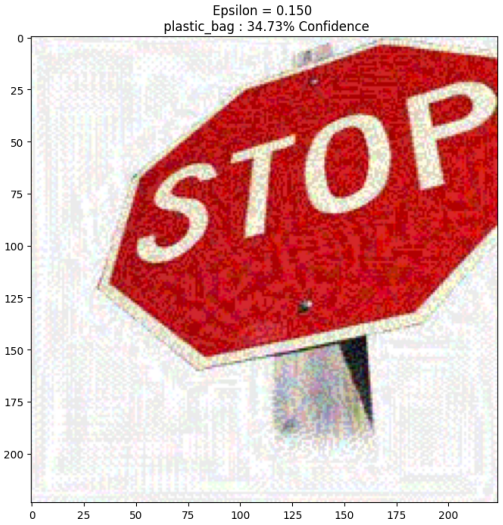
Output Images:


Epsilon=0


Epsilon=0.01




Epsilon = 0.10

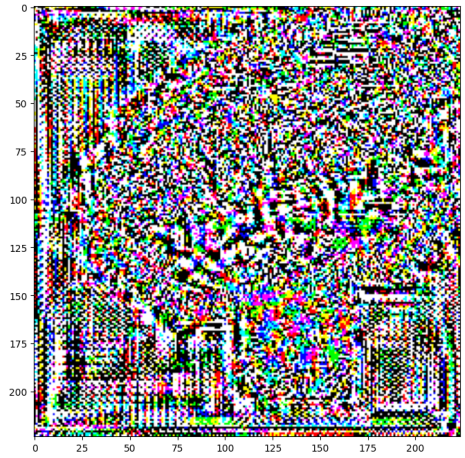



Epsilon = 0.150

Comment: The more the noise, the less the confidence of the model.

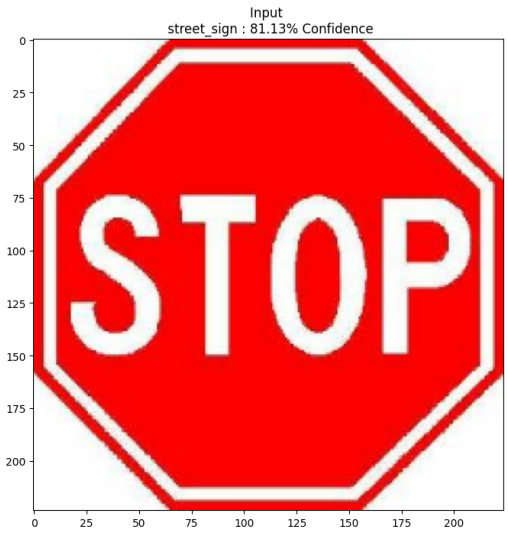
Test Case 2:


Input Image



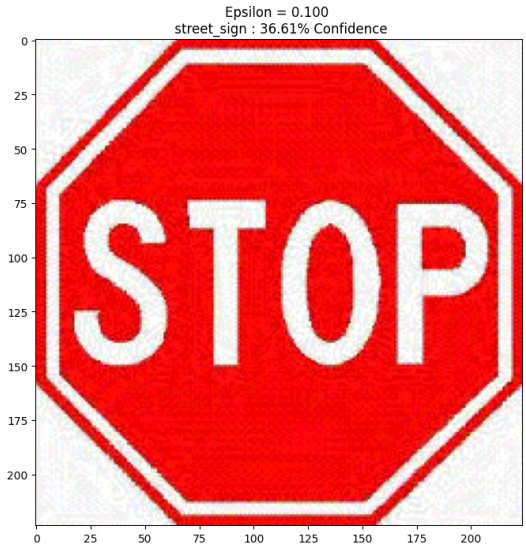
Perturbed Image

Output Images:




Epsilon=0


Epsilon=0.01





Epsilon = 0.10

Epsilon = 0.150

Comment: The more the noise, the less the confidence of the model.

## Task 4: Exploring LIDAR-based Adversarial Examples

So, we hit a bit of a snag with the LIDAR-based adversarial machine learning attacks - turns out the old repos we found aren't much use anymore. The libraries and dependencies are outdated, so we couldn't get any of them to work. But, it wasn't a total loss. We switched gears and dove into the READMEs and research papers to get a handle on how these attacks work. It's fascinating stuff, really. We learned about different strategies for creating adversarial examples, like tweaking 3D objects or messing with LIDAR data to trick the system. It's all about making the machine learning models see things that aren't there or miss what's actually there.

Then, there's the theory side of things, which we got from the research papers. These papers were gold mines for understanding the weak spots in LIDAR systems and how small changes in input can lead to huge mistakes. They also talked about what these attacks mean in the real world, especially for stuff like self-driving cars that rely on LIDAR for navigation. Some papers even compared different attack methods, showing which ones work best and how they can be countered. So, even though we couldn't get our hands dirty with the actual software, we still learned a ton about the sneaky world of LIDAR-based adversarial machine learning.

## Task 5: CTF Scenarios

**Idea 1** for Blue: We'll employ MobileNetV2 to assess the confidence of the captured image featuring a stop sign. If the confidence falls below a specified threshold, DeepRacer will halt its forward movement upon detecting the stop sign image.

**Idea 2** for Red: We will colorize an object with a black marker and put it as an object on the track. The intuition is to check whether DeepRacer can detect an object sharing the same color as the track.

**Idea 3** for Red: We're going to test DeepRacer's perception model by blurring a stop sign image through some drawing.

**Idea 4** for Red: We will put a transparent curtain before an object. So, DeepRacer must detect the object through the curtain. If it fails to detect, then a successful attack happens.

**Idea 5** for Red: We'll experiment with partially obscured objects, placing them behind other items or partially off the track. This will challenge the DeepRacer's ability to recognize incomplete visual information.

**Idea 6** for Red: We can also experiment with cars or humans in front of the stop sign to try and make the DeepRacer focus on the objects in front of them and totally ignore the stop sign.