

Reinforcement Learning Modelling for Autonomous Vehicle Navigation

MSc Research Project
Msc in Data Analytics

Aishwarya Rajguru
Student ID: x22248901

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Aishwarya Rajguru

Student ID: x22248901

Programme: MSc. Data Analytics

Year: 2023-2024

Module: MSc. Research Project

Lecturer: Dr. Catherine Mulwa

Submission

Due Date: 2nd September 2024

Project Title: Reinforcement Learning Modelling for Autonomous Vehicle Navigation

Word Count: 375

Page Count: 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Aishwarya Rajguru

Date: 2nd September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aishwarya Rajguru
Student ID: x22248901

1 Introduction

The main objective of this research is to recognize driver distraction using transfer learning and CNN (Convolutional Neural Network). The system was created using CNN and the pre-trained models ResNet50, VGG16, and VGG19. The "Detection of Driver Detection" study setup, hardware, and software requirements are all included in this configuration file. It also details all of the steps required to complete the study many steps.

2 System Configuration

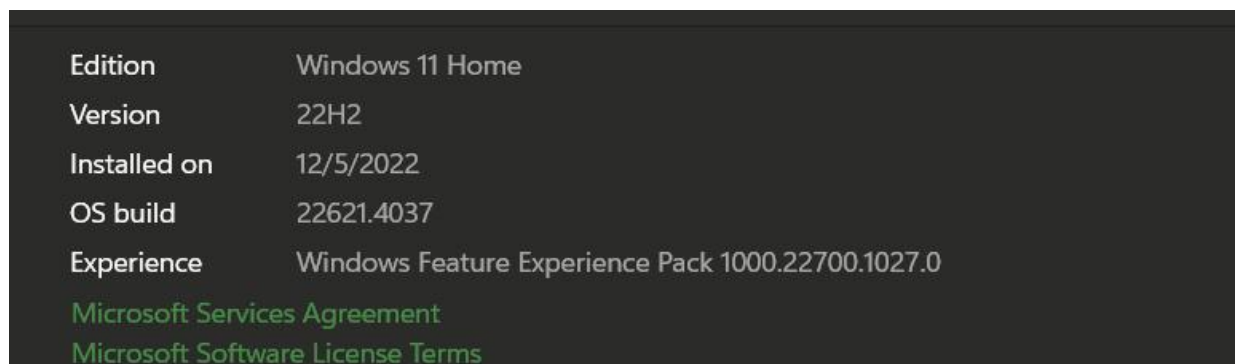
The necessary hardware and software will be covered in this section. The sections that follow are explained below.

2.1 Hardware

Below is the hardware configuration that is required.

Table 1: Hardware Configuration

Hardware	Configuration
Processor	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
Installed RAM	8.00 GB (7.35 GB usable)
System type	64-bit operating system, x64-based processor



Edition	Windows 11 Home
Version	22H2
Installed on	12/5/2022
OS build	22621.4037
Experience	Windows Feature Experience Pack 1000.22700.1027.0
Microsoft Services Agreement	
Microsoft Software License Terms	

Figure 1: Operating system configuration

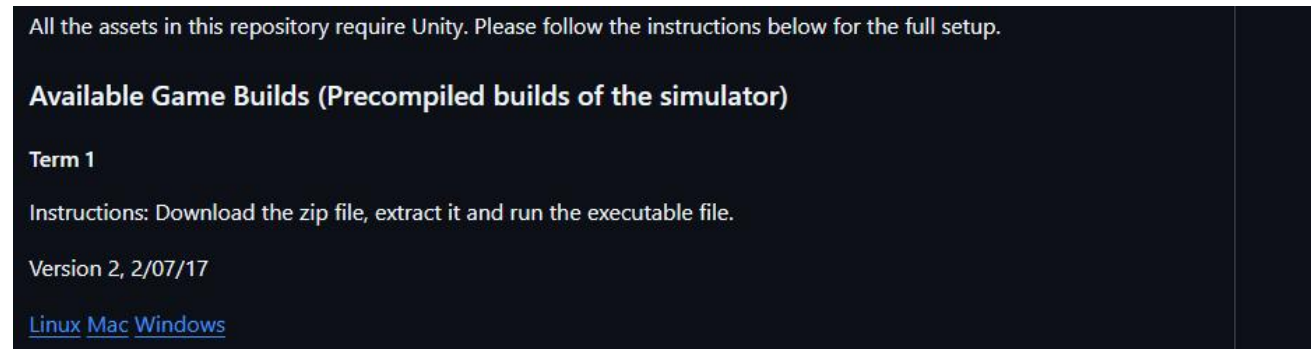
2.2 Software Requirements

Visual studio code editor Udacity self driving Car Simulation

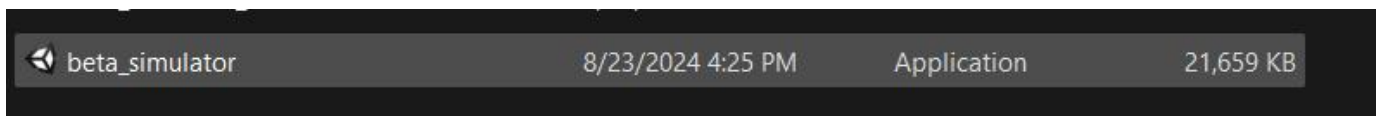
2.2.1 Self driving car simulator

Steps:

1. Go to website :- <https://github.com/udacity/self-driving-car-sim>
2. Download the term 1 for OS in my case Windows



3. After downloading open Beta simulator.exe file



4. Play game in training model to make the images i.e. dataset

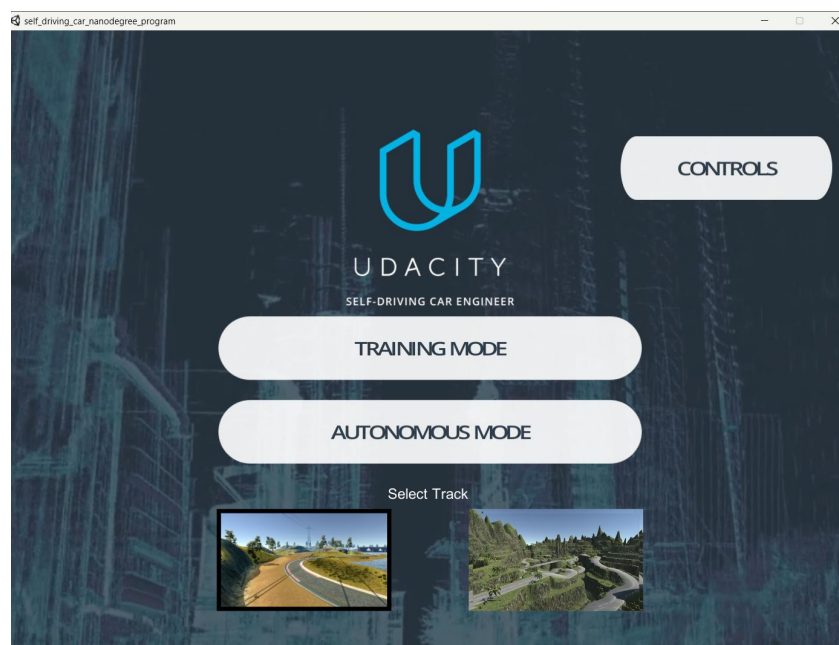


Figure 2: Udacity simulator



Figure 2: Play game with recording ON

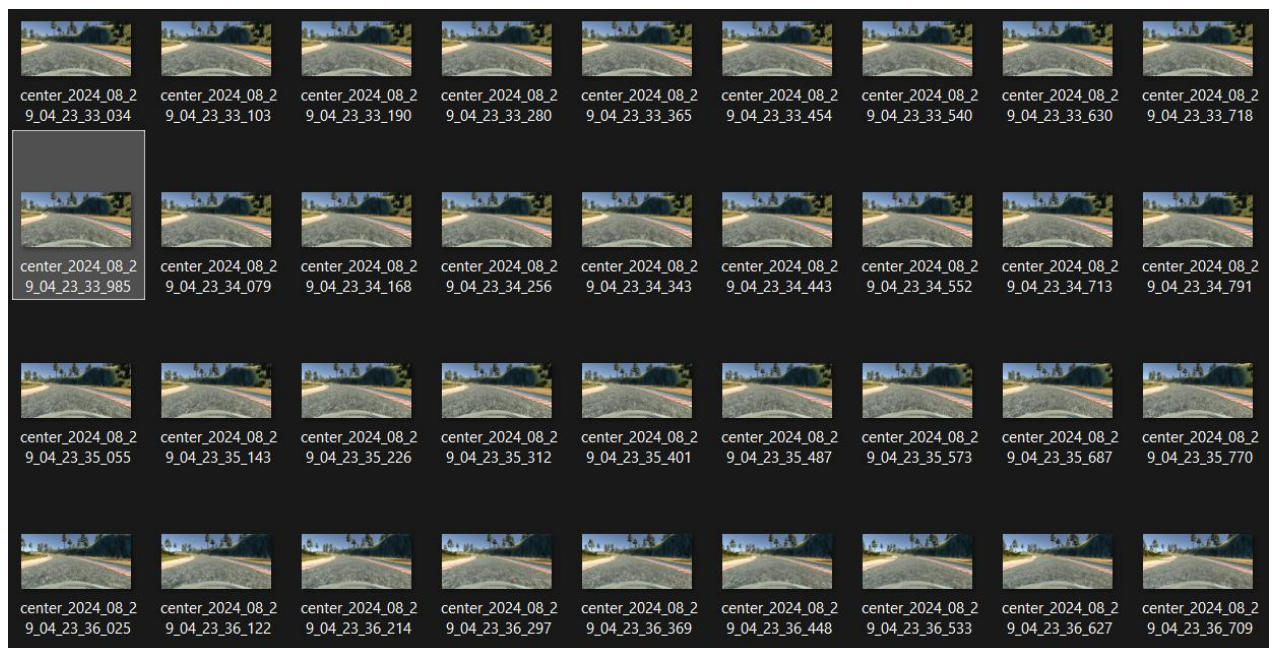


Figure 3 images Dataset will be created in IMG folder

3 Implementation, Evaluation and Results

```
def fetchname(filepath):
    return filepath.split('\\')[-1]

def dataframe(path):
    columns = ['Center', 'Left', 'Right', 'Steering', 'Throttle', 'Brake', 'Speed']
    data = pd.read_csv(os.path.join(path, 'driving_log.csv'), names=columns)
    data['Center'] = data['Center'].apply(fetchname)
    return data

def balance(data):
    nbins = 31
    sample = 500
    hist, bins = np.histogram(data['Steering'], nbins)
    removelist = []
    for i in range(nbins):
        binsdata = []
        for j in range(len(data['Steering'])):
            if data['Steering'][j] >= bins[i] and data['Steering'][j] <= bins[i+1]:
                binsdata.append(j)
        binsdata = shuffle(binsdata)
        binsdata = binsdata[sample:]
        removelist.extend(binsdata)
    data.drop(data.index[removelist], inplace=True)
    return data

def load_images(path, data):
    imagepath = []
    steering = []
    for i in range(len(data)):
        index_data = data.iloc[i]
        imagepath.append(os.path.join(path, 'IMG', index_data[0]))
        steering.append(float(index_data[3]))
    return np.asarray(imagepath), np.asarray(steering)

def augimg(imagepath, steering):
    image = mti.imread(imagepath)
    if np.random.rand() < 0.5:
        pan = ima.Affine(translate_percent={'x': (-0.1, 0.1), 'y': (-0.1, 0.1)})
        image = pan.augment_image(image)
    if np.random.rand() < 0.5:
        zoom = ima.Affine(scale=(1, 1.2))
        image = zoom.augment_image(image)
    if np.random.rand() < 0.5:
        bright = ima.Multiply((0.5, 1.3))
        image = bright.augment_image(image)
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering = -steering
    return image, steering

def preproc(image):
    img = image[65:137, :, :]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img / 255.0
    return img
```

Figure 4 Utils.py

The utils.py file contains all the utility functions like augmenting images, loading images, pre process and batching functions etc.

```
def create_model():
    model = Sequential()
    model.add(Convolution2D(24, (5, 5), (2, 2), input_shape=(66, 200, 3), activation='elu'))
    model.add(Convolution2D(36, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(48, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(64, (3, 3), activation='elu'))
    model.add(Convolution2D(64, (3, 3), activation='elu'))
    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.compile(Adam(learning_rate=0.0001), loss='mse')
    return model

def create_deeper_model():
    model = Sequential()
    model.add(Convolution2D(24, (5, 5), (2, 2), input_shape=(66, 200, 3), activation='elu'))
    model.add(Convolution2D(36, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(48, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(128, (3, 3), activation='elu'))
    model.add(Convolution2D(128, (3, 3), activation='elu'))
    # model.add(Convolution2D(64, (3, 3), activation='elu'))
    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(25, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.compile(Adam(learning_rate=0.0001), loss='mse')
    return model

def create_additional_cnn_model():
    model = Sequential()
    model.add(Convolution2D(32, (3, 3), (1, 1), input_shape=(66, 200, 3), activation='elu'))
    model.add(Convolution2D(64, (3, 3), (2, 2), activation='elu'))
    model.add(Convolution2D(128, (3, 3), (2, 2), activation='elu'))
    model.add(Convolution2D(128, (3, 3), activation='elu'))
    model.add(Flatten())
    model.add(Dense(120, activation='elu'))
    model.add(Dense(60, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.compile(Adam(learning_rate=0.0001), loss='mse')
    return model
```

```

def create_model():
    model = Sequential()
    model.add(Convolution2D(24, (5, 5), (2, 2), input_shape=(66, 200, 3), activation='elu'))
    model.add(Convolution2D(36, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(48, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(64, (3, 3), activation='elu'))
    model.add(Convolution2D(64, (3, 3), activation='elu'))
    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.compile(Adam(learning_rate=0.0001), loss='mse')
    return model

def create_deeper_model():
    model = Sequential()
    model.add(Convolution2D(24, (5, 5), (2, 2), input_shape=(66, 200, 3), activation='elu'))
    model.add(Convolution2D(36, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(48, (5, 5), (2, 2), activation='elu'))
    model.add(Convolution2D(128, (3, 3), activation='elu'))
    model.add(Convolution2D(128, (3, 3), activation='elu'))
    # model.add(Convolution2D(64, (3, 3), activation='elu'))
    model.add(Flatten())
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(25, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.compile(Adam(learning_rate=0.0001), loss='mse')
    return model

def create_additional_cnn_model():
    model = Sequential()
    model.add(Convolution2D(32, (3, 3), (1, 1), input_shape=(66, 200, 3), activation='elu'))
    model.add(Convolution2D(64, (3, 3), (2, 2), activation='elu'))
    model.add(Convolution2D(128, (3, 3), (2, 2), activation='elu'))
    model.add(Convolution2D(128, (3, 3), activation='elu'))
    model.add(Flatten())
    model.add(Dense(120, activation='elu'))
    model.add(Dense(60, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))
    model.compile(Adam(learning_rate=0.0001), loss='mse')
    return model

```

Figure 5 Models

Models are also described in the `utils.py` file.

```

def train_and_optimize(path, model_type='original'):

    datam = dataframe(path)
    ndata = balance(datam)
    imagepath, steering = load_images(path, ndata)

    xtrain, xtest, ytrain, ytest = train_test_split(imagepath, steering, test_size=0.3, shuffle=True, random

    if model_type == 'deeper':
        model = create_deeper_model()
    elif model_type == 'additional_cnn':
        model = create_additional_cnn_model()
    else:
        model = create_model()

    model.summary()

    model.fit(batching(xtrain, ytrain, 15, True), steps_per_epoch=10, epochs=10,
              validation_data=batching(xtest, ytest, 10, False), shuffle=True, validation_steps=200)

    model.save(f'cnn_model_{model_type}.keras')

    cnn_predictions_train = model.predict(batching(xtrain, ytrain, 32, False), steps=len(xtrain)//32)
    cnn_predictions_test = model.predict(batching(xtest, ytest, 32, False), steps=len(xtest)//32)

    min_samples_train = min(len(cnn_predictions_train), len(ytrain))
    min_samples_test = min(len(cnn_predictions_test), len(ytest))

    cnn_predictions_train = cnn_predictions_train[:min_samples_train]
    ytrain = ytrain[:min_samples_train]

    cnn_predictions_test = cnn_predictions_test[:min_samples_test]
    ytest = ytest[:min_samples_test]

    gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
    gbr.fit(cnn_predictions_train, ytrain)

    gbr_predictions = gbr.predict(cnn_predictions_test)
    mse = mean_squared_error(ytest, gbr_predictions)
    r2 = r2_score(ytest, gbr_predictions)

    print(f"Mean Squared Error after optimization ({model_type} model):", mse)
    print(f"R^2 Score after optimization ({model_type} model):", r2)

    return model, gbr

```

Figure 6 Training.py

This is training.py this file will be run after creating dataset from training from simulator explained in the above steps.

Command to run:- python training.py

```
# Define the directory removal function
def remove_directory(path):
    try:
        shutil.rmtree(path)
    except PermissionError as e:
        print(f"PermissionError: {e}")
    except FileNotFoundError as e:
        print(f"FileNotFoundError: {e}")

# Initialize the Socket.IO server
sio = socketio.Server(logger=True, engineio_logger=True)
app = Flask(__name__)

# Load the model from the specified file
print("Loading model...")
#30/08/2024
model_path = sys.argv[1]
print(model_path)
model = load_model(model_path)
print("Model loaded successfully.")
prev_image_array = None
```

```
# Simple PI controller for throttle control
class SimplePIController:
    def __init__(self, Kp, Ki):
        self.Kp = Kp
        self.Ki = Ki
        self.set_point = 0.
        self.error = 0.
        self.integral = 0.

    def set_desired(self, desired):
        self.set_point = desired
        print(f"Desired speed set to: {desired}")

    def update(self, measurement):
        self.error = self.set_point - measurement
        self.integral += self.error
        control_value = self.Kp * self.error + self.Ki * self.integral
        print(f"Control update - Measurement: {measurement}, Error: {self.error}, Integral: {self.integral}, Control Value: {control_value}")
        return control_value

# Initialize the controller
controller = SimplePIController(0.1, 0.002)
set_speed = 25
controller.set_desired(set_speed)

# Function to handle telemetry events
@sio.on('telemetry')
def telemetry(sid, data):
    print("telemetry recccccccccccccccccc")
    if data:
        print("Received telemetry data.")
        steering_angle = data["steering_angle"]
        throttle = data["throttle"]
        speed = data["speed"]
        imgString = data["image"]
```

Figure 7 test.py

This is test.py file here all the configurations for running the vehicle in simulator using the trained models will be done

All the configurations and socket integration is done in this file to connect the simulator through port 4567.

To use the test first the file should be run the followed by the model which we want to use then optionally if user want to record the run user can enter the folder name where user wants to store the run images.

Command:-

With recording:-

```
python test.py "model_name.keras"
```

Without recording:-

```
Python test.py "model_name.keras" "rec"
```

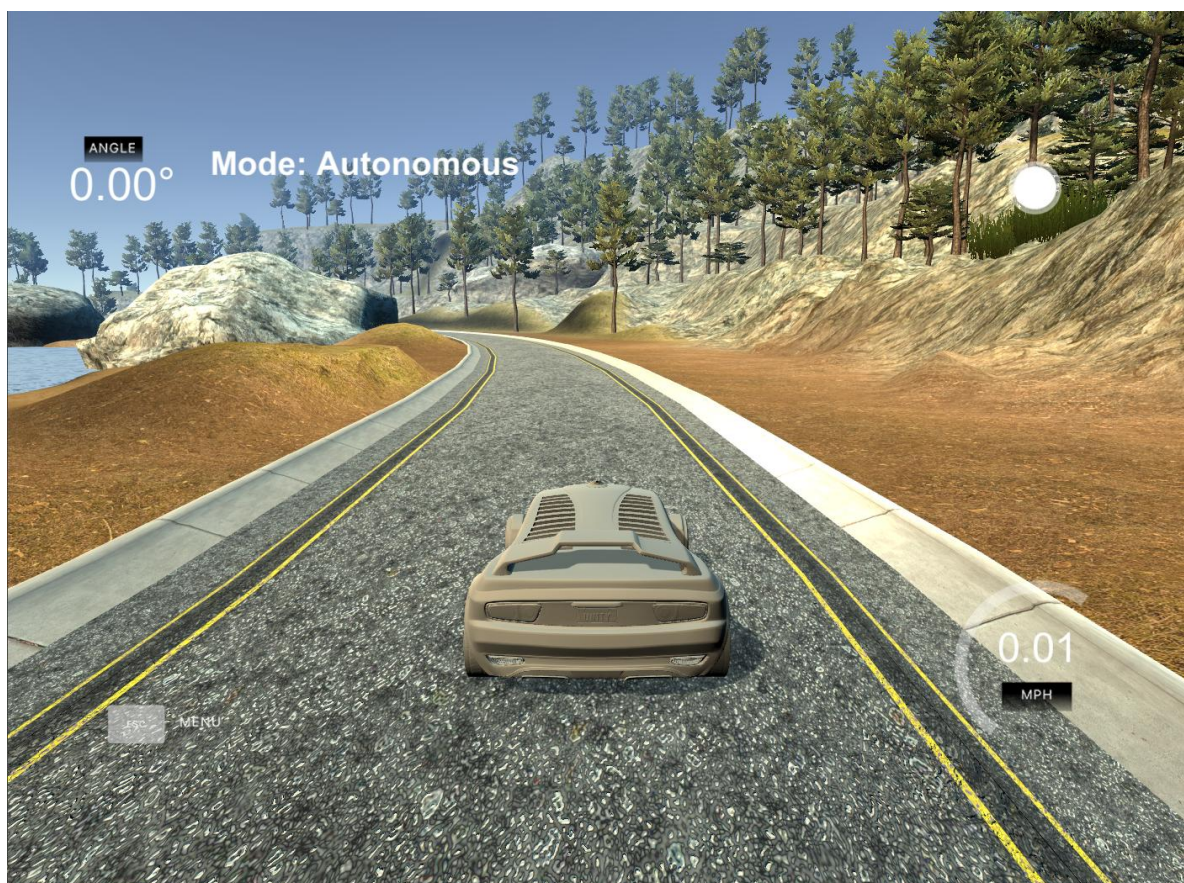


Figure 8 Running in autonomous