# CS 5392 Spring 2023 Assignment-1 Tutorial

**Title: -**

Developing a Reinforcement Learning Agent for Autonomous Driving in the CARLA Simulator using Python

**Objective: -**

- This project aims to develop a reinforcement learning agent based on Q-learning that can effectively drive a car in a CARLA simulator using Python.
- The agent will learn to navigate the environment by acting based on observations and receiving reward signals for its performance.
- By using this approach, the project aims to advance the field of autonomous driving by developing and testing more efficient and safer autonomous driving systems in a safe and controlled environment.

**Level:** Medium

**Background:**

### *i. Reinforcement learning*

Reinforcement learning is a type of machine learning that involves training an agent to make decisions in an environment based on trial and error [4]. In the context of driving a car, reinforcement learning can be used to train an agent to make driving decisions based on its observations of the environment, such as the positions of other cars, road conditions, and traffic signals, weather conditions. We are using Carla as a simulator in our project [4].

### *ii. Carla: -*

Carla (Car Learning to Act) is an open-source autonomous driving simulator the Computer Vision Center developed. This simulator provides a realistic 3D environment for testing and training autonomous driving algorithms.

Carla is designed to be a powerful tool for researchers and developers who are working on autonomous driving technologies. It allows them to test and evaluate their algorithms in a simulated environment, which can be much faster and safer than testing on real-world roads. Additionally, Carla allows for the creation of a wide range of scenarios, including different road layouts, weather conditions, and traffic patterns, which can be used to evaluate the robustness of autonomous driving algorithms [5].

The server-client architecture of Carla allows for a flexible and scalable simulation environment. The server represents the simulation environment, which can be run on a separate machine from the clients. The clients, on the other hand, represent autonomous driving agents, which can be run on multiple machines simultaneously.

In the context of Carla, using reinforcement learning to drive a car involves the following steps:

- **Define the Carla environment**: The first step is to define the Carla environment in which the car will be driving. This involves setting up the road network, traffic flow, weather conditions, and

other environmental factors. We are planning on not taking dynamic weather but only concentrating on steering and collision of the vehicle.

- **Define the car actions**: Next, the actions that the car can take in the Carla environment must be defined. This involves programming the car to be able to accelerate, brake, turn left or right, change lanes, and navigate intersections. Here in our project, we consider 3 actions - pass.

  steer left,
  steer right,
  steer center.

- **Define the rewards**: The rewards system is used to reward the car to take actions that lead to desired outcomes. For example, the rewards system may encourage the car to stay in its lane, avoid collisions with other vehicles, and obey traffic signals. In our project, we plan to consider:

  Reward = 1 if no collisions and at an appropriate speed
  Reward = -1 if not at the proper speed and less reward if there is a collision.

- **Train the agent:** The agent is part of the system that makes decisions based on its observations of Carla's environment. The agent is trained using a reinforcement learning algorithm, such as Q-learning, to make decisions that maximize the rewards it receives.

- **Test the agent**: Once the agent has been trained, it can be tested in the Carla environment to evaluate its performance. The agent's performance can be measured based on metrics such as how well it follows traffic rules, how efficiently it uses fuel, and how safely it drives. In this project is based on collision-free driving with proper speed.

- **Refine the agent**: Based on the results of the testing, the agent can be refined and further trained to improve its performance in the Carla environment.

### *iii. Software Requirements:*

It is recommended to have a more powerful computer with a higher-end GPU and a faster internet connection [1].

- Operating system: Any 64-bit version of Windows or Ubuntu 18.04
- Disk space: At least 165 GB of free space is required, with 32 GB used by the Carla simulator and 133 GB used by related software installations, including the Unreal Engine.
- GPU: An adequate GPU is required to run the simulator, with at least 6 GB of memory, although 8 GB is recommended. A dedicated GPU is highly recommended for machine learning.
- Network ports: The simulator requires two TCP ports, 2000 and 2001, which must not be blocked by firewalls or other applications.
- You need to have a GitHub account linked to Unreal Engine's account.

### *iv. Installation Requirements: -*

- Install Visual Studio 2019 or later with C++ support.
- Install Git, CMake, a file compression software(7Zip), and Make.
- Install Python 3.7 or later.
- Install the Unreal Engine prerequisites (Visual Studio Build Tools, DirectX, Windows 10 SDK)
- Clone the Carla repository from GitHub.

- Configure Carla using CMake.
- Build the Carla client and server executables using Visual Studio
- Download the required assets for Carla.
- Run the Carla server and launch a client to test the installation.

These are the main tasks required to build and install Carla on a Windows machine. The documentation [1] provides detailed instructions for each task, along with links to download the necessary software and dependencies.

### v. Libraries installed: -

- OpenCV-python: -

  OpenCV is an open-source library for computer vision, machine learning, and image processing[6]. In this project, it is useful processing the image captured by the sensor and write it to disk.

- NumPy –

  It is useful for mathematical operations on arrays.

### *vi. Q-Learning: -*

Q – learning is a model-free reinforcement learning which does not require the model of the environment. Given infinite exploration time and partly random policy, Q-learning can identify an optimal action-selection policy for any given FMDP. Here FMDP is a finite Markov decision process [2]. The Q-learning algorithm works by updating the Q-value of a state-action pair using the Bellman equation.

The Bellman equation computes the expected future reward for a given state and action by considering the immediate reward of the current action and the expected future reward of the next state-action pair. The Q-value is updated iteratively until it converges to the optimal value.

Here the Q value is updated per action per possible state. This creates a table. As it is model-free, in order to figure out all the possible states, we can either query the environment or we just simply must engage in the environment for a while to figure it out.

Equation 1: Bellman Equation [2]

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}$$

**Discount Factor**: Here the discount factor is a measure of how important the future reward is. Generally, the discount factor ranges between 0 and 1. We want it high value to get a positive outcome. It must learn the chain of events and the importance of long-term gains than short term.

**Learning rate**: It determines to what extent new information is useful more than old information.

**Estimate of optimal future value**: - It is taken after we have performed our action and later, we update our previous values based partially on the next-step best Q value.

### *vii. Use case: -*

Below is the use case of the Carla architecture we going to use in our project. In this project, we create an environment with the help of Carla's blueprints. We get the vehicle the agent and its environment to have cameras i.e. sensors. Based on the steering actions, the environment updates the q-table using q-learning. The rewards are chosen based on the collision and speed and given to the agent. With training, it will be able to drive a car with no collisions.

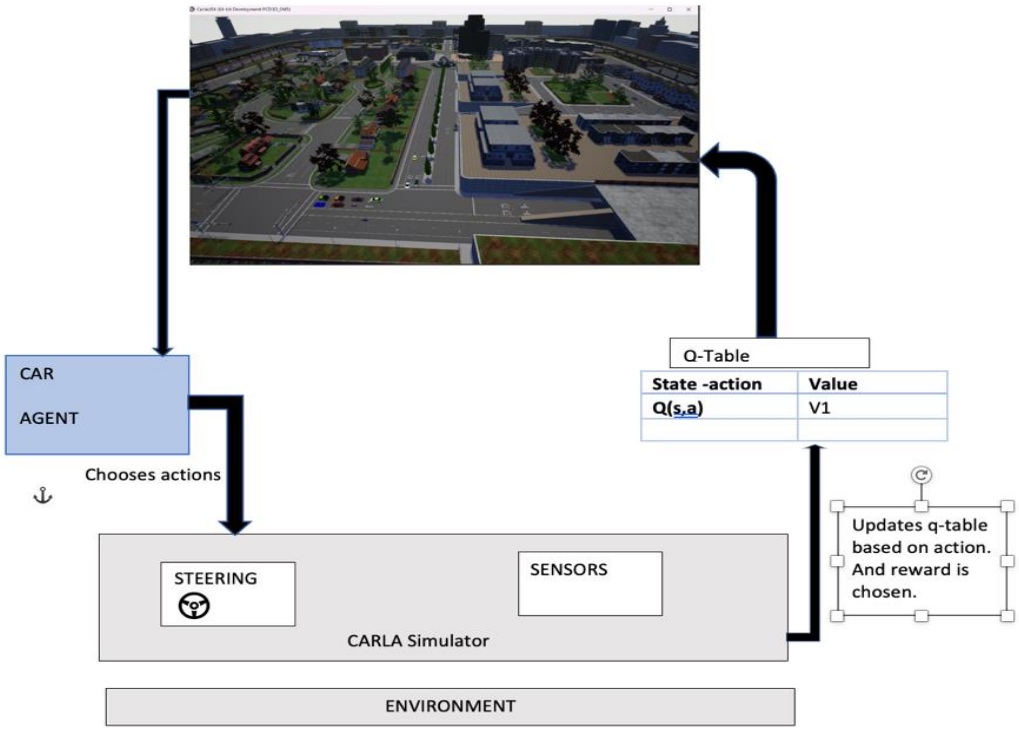


Figure1: Use case- The architecture of the car(agent) in the Carla simulator [3]

**Requirements: -**

- Understanding of Python 3 programming.
- Familiarity with the python package NumPy
- Able to understand Carla's documentation to build the environment.
- Knowledge about client and server architecture.

**Tasks**:

In the project, we referred to Carla's documentation and were able to control the car and work on pedestrians and other actors. We tried to get dynamic weather, traffic, and spawning actors. We have taken a vehicle mustang model and then applied the steering and attached the camera to it. Now we need to create an environment and apply q-learning.

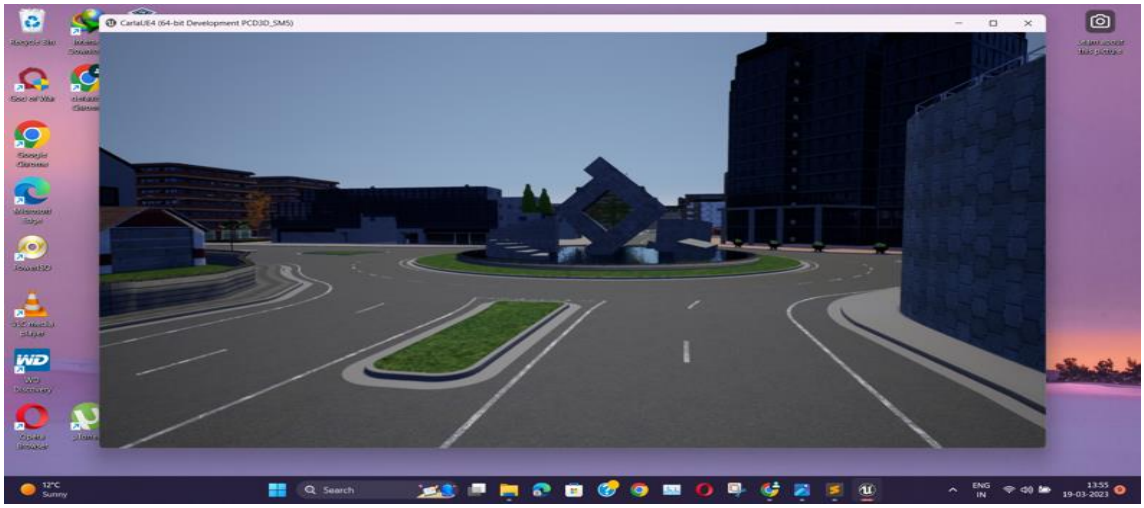

Figure 2 : The environment where we can work. We can use WASD keys to control the car.

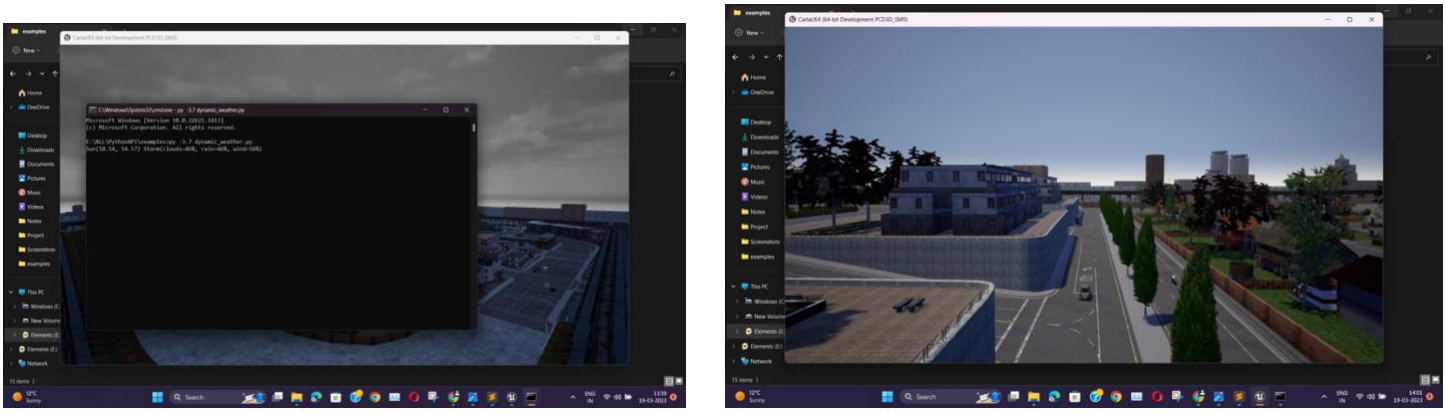

Figure 3 : The dynamic weather with cloudy days and sunlight days.

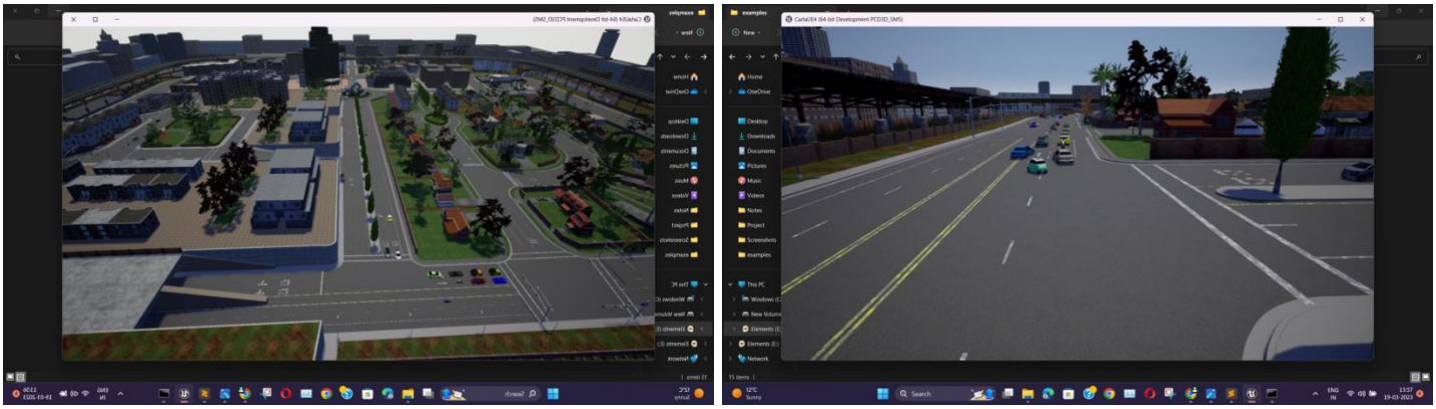

Figure 4; The environment when we are spawning based on the number of vehicles.

Here is a simple snippet of how we add actors to the environment. In our example car as an agent.

```
54
55      client =carla.Client("localhost", 2000)
56      client.set_timeout(20000.0)
57      world= client.get_world()
58      blueprint_library =world.get_blueprint_library()
59      bp= blueprint_library.filter("bmw")[0
60      print(vehicle.bp)
61      spawn_point =random.choice(world.get_map().get_spawn_points())
62      vehicle=world.spawn_actor(bp,spawn_point)
63      vehicle.apply_control(carla.VehicleControl(throttle=1.0 , steer=0.0))
64      actor_list.append(vehicle)
65
```

Figure 6: The code snippet of adding actors to the Carla simulator.

- In line 55, Set up the client using the CARLA client object to the local host 2000.
- In line 57, we retrieve the world object using the client object.
- In line 58, we access the blueprint from the world object.
- In line 61, we spawn actors (vehicles with BMW models) into the world.
- In line 63, we apply control like steering and throttle to the car. Add into actors list.
- Later Spawn the camera and attach it to the vehicle.
- Destroy the actors at the end.
- Similarly other actors like pedestrians walking on the walking path. Any other traffic lights or other features are added.

Figure 7: Our agents in the Carla simulator, a pedestrian walking, and a car that drove and hit to the side.

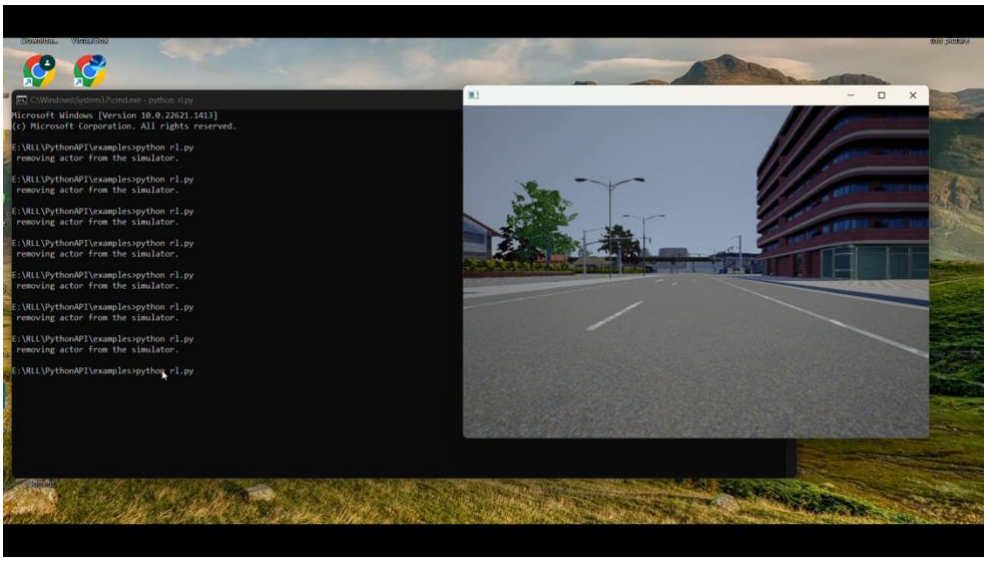


Figure 8: The dialog window for the sensor attached to the vehicle which shows the front view of the car only.

**Learning outcomes:**
After completing the tutorial,

- Learn how to use Python 3
- Learn what is Q-Learning, Bellman Equation, and its role in developing a reinforcement learning agent.
- Learn about the environment and different actions in the Carla simulator.

**Exercises:**

- "Q-Learning Algorithms: A Comprehensive Classification and Applications "details all the variants of Q-learning algorithms and their applications in recent innovations.
- The article "Autonomous Driving in Roundabout Maneuvers Using Reinforcement Learning with Q-Learning" clearly describes Reinforcement learning and q-learning techniques.
- A blog on "An introduction to Q-Learning: Reinforcement Learning" can be used and developed along with the basics of the bellman equation.
- An article on "Implementing Q-learning on Simple Vehicle: Learning Obstacle Avoidance in Different Environments".
- A journal on "Autonomous Vehicle Path Planning using Q-Learning" can be a good exercise to start after this.

**Cited References:**

1. *Carla Documentation for windows build: https://carla.readthedocs.io/en/latest/build_windows/*
2. *Wikipedia.org: https://en.wikipedia.org/wiki/Q-learning*
3. *García Cuenca, L.; Puertas, E.; Fernandez Andrés, J.; Aliane, N. Autonomous Driving in Roundabout Maneuvers Using Reinforcement Learning with Q-Learning. Electronics 2019, 8, 1536. https://doi.org/10.3390/electronics8121536*
4. *Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.*
5. *OpenCV: https://opencv.org/*
6. *Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V., & Ferrari, V. (2017). Carla: An open urban driving simulator. In Proceedings of the 1st Annual Conference on Robot Learning (Vol. 78, pp. 1-16).*