**Constraints:**

**Data availability and quality constraints**: The system will operate within the constraints of the available data, including the quality and quantity of the training data used for the imitation learning process.

Technical constraints: The system will operate within specified technical limitations, including processing power, memory capacity, and sensor accuracy.

Time Constraints: The development of the system must be completed within the specified time.

1. Software interfaces:

* Carla Python API: This is the primary interface used to control Carla simulator and collect data from it. It provides a set of functions and classes that allow for control of the simulation environment, such as spawning vehicles, controlling traffic, and accessing sensor data.
* ROS (Robot Operating System): If the data is being collected for use in a robotics application, the ROS middleware can be used to communicate with Carla simulator and collect sensor data. This allows for integration with other ROS-compatible software packages.
* PyGame: Pygame provides a simple interface for reading input from a variety of input devices, including joysticks, game controllers, and keyboards. To control the vehicle in Carla simulator using a joystick, you can use Pygame to read input from the joystick and map it to the appropriate vehicle control commands. For example, you can use the Pygame joystick functions to read the joystick's position and button states, and use that information to control the vehicle's throttle, brake, and steering.

1. Hardware interfaces:

* Input devices: If the data collection process requires human input, input devices such as joysticks be used to control the carla simulator.

1. Cameras: Cameras are a common sensor used in autonomous vehicles for tasks such as object detection, lane following, and traffic sign recognition. The hardware interfaces for cameras may include the camera sensor itself, as well as any necessary cabling or connectors to connect the camera to the onboard computer or other processing hardware.
2. Lidar: Lidar sensors use laser beams to measure distances and create a 3D point cloud of the surrounding environment. They are commonly used for tasks such as obstacle detection and mapping. The hardware interfaces for lidar may include the lidar sensor itself, as well as any necessary cabling or connectors.
3. Radar: Radar sensors use radio waves to detect objects and measure their distance and velocity. They are commonly used for tasks such as adaptive cruise control and collision avoidance. The hardware interfaces for radar may include the radar sensor itself, as well as any necessary cabling or connectors.
4. GPS: GPS sensors are used to determine the vehicle's location and velocity. They are commonly used for tasks such as navigation and mapping. The hardware interfaces for GPS may include the GPS sensor itself, as well as any necessary cabling or connectors.
5. Control Interfaces: Control interfaces such as steering wheels, pedals, and joysticks are used to control the vehicle's movement. The hardware interfaces for these control interfaces may include any necessary sensors or connectors to detect the position and movement of the controls.
6. Computing Hardware: The computing hardware used for processing and storing data is also relevant to mention in the SRS. This may include the onboard computer or other processing hardware used to collect and process the sensor data.

**Performance metrics:**

1. Mean squared error (MSE): This is a measure of the average squared difference between the predicted and actual values. It is commonly used to evaluate the accuracy of steering angle and speed predictions.
2. Root mean squared error (RMSE): This is the square root of the MSE and provides a measure of the standard deviation of the error.
3. Mean absolute error (MAE): This is the average absolute difference between the predicted and actual values. It is another commonly used metric to evaluate the accuracy of steering angle and speed predictions.

Testing in carla simulator:

1. Launch the Carla simulator and select the map where you want to test the model.
2. Load the trained model into your application code.
3. Initialize the Carla simulator client in your code and connect it to the running Carla simulator instance.
4. Retrieve sensor data from the Carla simulator, such as camera images or lidar point clouds.
5. Preprocess the sensor data to make it suitable for input to the trained model.
6. Pass the preprocessed sensor data as input to the trained model and generate predicted outputs, such as steering angle, speed, and brake.
7. Send the predicted outputs back to the Carla simulator as control commands for the vehicle.
8. Visualize the output of the model in the Carla simulator and evaluate its performance.

During testing, you can vary the input conditions such as weather, lighting, and road conditions to evaluate the robustness of the trained model. You can also compare the predicted outputs with ground truth data to calculate performance metrics such as mean absolute error (MAE) or root mean square error (RMSE) to evaluate the accuracy of the model.