Using the CARLA Simulator to Train A Deep Q Self-Driving Car to Control a Real-World Counterpart on A College Campus

Conference Paper · December 2023

DOI: 10.1109/BigData59044.2023.10386739

CITATIONS
4

READS
548

5 authors, including:

Samir Poudel
Middle Tennessee State University
16 PUBLICATIONS **46** CITATIONS
SEE PROFILE

Sammi Hamdan
Middle Tennessee State University
4 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Khem Poudel
Middle Tennessee State University
51 PUBLICATIONS **264** CITATIONS
SEE PROFILE

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Using the CARLA Simulator to Train A Deep Q Self-Driving Car to Control a Real-World Counterpart on A College Campus

JOSEPH MAY[1], KHEM POUDEL[2], SAMIR POUDEL[3], SAMMI HAMDAN[4], JORGE VARGAS[5]
[1,2,3,4]Computer Science. Middle Tennessee State University, Murfreesboro TN 37132 USA (e-mail: jam2ft@mtmail.mtsu.edu; Khem.Poudel@mtsu.edu; sp2ai@mtmail.mtsu.edu; sah9j@mtmail.mtsu.edu)
[5]Engineering Technology, Middle Tennessee State University, Murfreesboro TN 37132 USA (e-mail: Jorge.Vargas@mtsu.edu)

This work was supported by MTSU.

ABSTRACT Autonomous vehicles (AV) provide many benefits to society. They provide a way for people with the inability to drive a way to travel on their own. While there has been significant progress made in their field, AVs still struggle with handling difficult, urban environments. This study aims to implement a level 4 AV utilizing a deep q neural network within the CARLA simulator. The Carla simulator is used to train and test a self-driving car agent implemented with the deep Q neural network. Sensor fusion is being implemented with three cameras located on the hood, left and right rearview mirrors, radars on the hood and trunk of the car, a GPS, and an IMU. Our simulated vehicle setup mimics a golf cart in the real world, equipped with the same sensor suite and setup. The vehicle is trained on Map 10 of Carla, with dynamic weather effects such as light, dark, clear skies, cloudy, dry, rainy, foggy, and windy. It is also trained with 5 other vehicles on the map as obstacles. Each simulation route is multipath and of random length and complexity. The agent maintains an average speed between 0 and 3.4 kph, on average makes it 7.64 meters away from its starting position, and has 119 meters left to reach the route destination.

INDEX TERMS CARLA Simulator, Deep Q Learning, Machine Learning Self Driving Car, Sensor Fusion.

I. INTRODUCTION

THE introduction of autonomous vehicles (AV) has many potential benefits. Large benefits include assisting in traffic enforcement, optimized traffic flow, increased vehicle usage efficiency, and reduced carbon emissions. More personalized benefits include faster commutes, optimal fuel consumption, reduction in accidents, greater time availability (from less time spent driving), and greater mobility for the disabled, young, old, and poor. [1], [2]

The implementation of AVs is challenging. It requires rote tasks like staying within a lane, turning, and stopping. It also requires more high-level tasks like understanding the right of way, yielding to emergency vehicles, following the appropriate behavior for different road signs, and route planning. [1]–[3] There are six different levels of autonomous driving that exist and are capable of varying degrees of self-actuation and function. [4], [5]

Level 0 is the most basic form of automation where there is 0 automation such that the user performs all management of the vehicle.

Level 1 is described as driver assistance, where the vehicle can monitor specific systems. At this level, the vehicle can perform automated braking, cruise control, and limited lane-keeping.

Level 2 is described as partial automation. The vehicle can do specific tasks on its own such as park, mitigate collisions, and assist steering.

Level 3 is described as conditional automation and is the first level where the vehicle becomes the primary monitor of the driving environment over a human driver. At this level, the car may have full control of the vehicle and perform most driving tasks. However, the human driver is still required to be able to take control in emergencies. Given how fast emergencies can arise on the road, swapping control from AV to human is not guaranteed to be safe. The human may not have enough time or understanding of current road conditions to react as they would if they were driving without the help of the AV. [6] Given the troubles with predicting emergencies, and the unreliable AV-human control switch, it is best to aim for automation at level 4 or above.

Level 4 is described as high automation, wherein the vehicle can perform all driving tasks. Georeferencing is required,

and the driver may still override vehicle control if deemed necessary.

Level 5 is described as full automation. The vehicle controls all driving tasks in all conditions. No human involvement is needed.

We aim to implement a level 4 AV utilizing a deep q neural network within the CARLA simulator.

A. MAIN GOALS

This research has three main purposes. The first is to establish a test bed for Autonomous Vehicle testing and verification on MTSU. We aim to create a functional and extensible setup that supports hardware in the loop and software development which can serve as the foundation for future AV research at MTSU. With that in mind, components and features have been designed to make future work easy to build on. The second goal of this research is to integrate the MTSU campus environment into the CARLA simulator. This supports the first goal and has benefits for AVs. The MTSU campus is a high-density slow-moving environment that contains unique layouts with interesting scenarios for AV testing. There are various types of human obstacles such as pedestrians, skateboarders, and bikers, who frequently and infrequently obey the rules of the road. There are cars of all types including service vehicles like lawn movers, unique road layouts with one-way streets, traffic circles, multi-way stops, and more which provide a varied and rich environment to train an AV in. The final goal of this research is to create a deep Q neural network for an autonomous vehicle that can adhere to all basic traffic rules, and successfully completes routes 80% of the time.

II. BACKGROUND AND PRIOR WORK

A. REINFORCEMENT LEARNING

Reinforcement learning is a subfield of machine learning that focuses on how an agent can learn to take appropriate actions in an environment that maximizes a reward signal. It is inspired by the way humans and animals learn through interactions with their surrounding environment. The AI agent learns by exploring the environment through a set of defined actions and receives feedback in the form of rewards or punishments. The reward or punishment is determined by the result of the agent's actions. For example, an agent may follow the GPS route and receive a reward for correct behavior, or the agent may collide with a building and receive a commensurate punishment for undesirable behavior. The goal is for the agent to learn a policy that maximizes the expected cumulative reward over time. [7]

The environment an agent interacts with is typically modeled as a Markov Decision Process (MDP), [7] which consists of states, actions, transition probabilities, and rewards. At each time step, the agent observes the current state, selects an action based on its policy, and receives feedback in the form of a reward and the next state. Over time as the agent explores the environment, the agent may learn a policy that maximizes the long-term cumulative reward.

Reinforcement learning algorithms use various techniques to optimize the agent's policy. A common method is to use value functions, like the state or action value functions, to estimate the expected return for different states or state-action pairs. [7] The agent learns a better policy by updating the value functions based on the observed rewards and transitions, Other methods, such as policy gradients and Q-learning, also play important roles in reinforcement learning, providing different ways to approximate and update the policy based on the observed experiences.

B. Q AND DEEP Q LEARNING

Q learning is a type of model-free reinforcement learning. [8] Model-free means that the agent is capable of reacting to unseen states and can learn in different environments, which is particularly relevant for driving scenarios where the AV may have to react to unforeseen weather and/or traffic conditions. The q value represents the expected cumulative reward for taking a given action in a particular state which means the q value is a function of state action pairs given by Q(s,a). Q learning combines parts of dynamic programming and Monte Carlo methods. [8]

Q learning begins with a starting Q table that is initialized with random values for all state-action pairs. The agent will interact through the environment by selecting actions based on the current q value and receive rewards based on the q value of the state action pair. [8] Every time the agent interacts with the Q table it will be updated with the following formula:

$$Q(s,a) = Q(s,a) + \alpha * [R + \gamma * max(Q(s',a')) - Q(s,a)] \quad (1)$$

where:

- Q(s, a) represents the current Q-value for state 's' and action 'a'.
- α is the learning rate. The learning rate determines how much the agent updates its Q-values based on new information. Higher rates lead to more dramatic changes in Q values every update.
- R is the immediate reward received after taking action 'a' in state 's'.
- γ is the discount factor, balancing the importance of immediate and future rewards. This is a means to get the agent to work towards long-term goals rather than instant gratification.
- s' is the next state reached after taking action 'a'.
- max(Q(s', a')) represents the maximum Q-value among all possible actions in the next state 's'.

The Q-learning algorithm repeats the process of environment interaction and Q-value updates until the agent converges to optimal Q-values, which correspond to the optimal policy. [8] Through continual exploration of different states and actions, the agent learns which actions yield higher rewards, enabling it to make better decisions over time.

A deep Q network is a reinforcement learning technique that uses Q learning. [9] A neural network of multiple com-

putational layers takes in the environmental state as input and calculates the q values as output. The appropriate Q values are updated by back-propagating weights to each layer of the network, minimizing the difference between predicted Q values and the rewards received for any given action.

Deep Q networks are trained using a replay buffer. As the agent interacts with the environment transitions, collections of state, action, reward, and next state pairs, are stored in the replay buffer. To train, the agent will randomly sample from the replay buffer to reduce the correlation between consecutive experiences. [9] Two agents, a current and a target, are used. The current agent is trained at all time steps. This enables the agent to learn, but also means that the actions the agent takes change frequently making it difficult to predict. The target agent is updated at a set episode number and mimics the weights of the current agent to receive the most current Q values, [9] and thus the best-known actions, but is periodically updated to preserve the stability of the agent. Additionally, this helps to prevent overfitting.

Once the Q-learning process is complete, the agent can follow a greedy policy. This means selecting the action with the highest Q-value in each state to maximize its expected cumulative reward. Learned Q-values can also be used to estimate state values or perform value iteration for planning in larger environments.

A Deep Q Network was chosen for training the self-driving car over other architectures due to its ability to handle complex and high-dimensional environments. Deep Q networks are capable of learning directly from raw sensory inputs, like camera images, and radar without relying on handcrafted features or domain-specific knowledge. This enables the self-driving car to perceive and understand its surroundings in a manner similar to how humans do, which enhances the vehicle's adaptability to real-world scenarios.

Deep Q networks are also capable of handling continuous state and action spaces, which are inherent to self-driving tasks. The continuous and nuanced driving environment requires the car to make precise decisions based on a wide range of possible actions, such as steering, acceleration, and braking, all of which must be done to differing degrees. DQNs can approximate optimal action values in continuous action spaces through techniques like discretization or function approximation, allowing the self-driving car to navigate seamlessly in a continuous decision-making space. Additionally, Deep Q Networks can iteratively refine their decision-making processes through trial and error. Continual learning enables the self-driving car to adapt to various driving conditions, road layouts, and traffic scenarios. Additionally, DQNs can leverage experience replay, a technique that enhances learning stability by reusing past experiences, leading to more efficient and stable training.

C. RELEVANT PROGRESS

In recent years, the field of autonomous vehicles (AV) has made significant progress, laying the foundation for the research and innovation ideas presented in this article. These advances were instrumental in shaping the goals and approaches of this study.

1. Sensor technology: The development of advanced sensors such as LiDAR, radar, and high-resolution cameras has enabled AV to collect detailed, real-time data about its surroundings. These sensors play an important role in providing accurate perception and situational awareness, thereby improving the decision-making and safety of autonomous systems.

2. Deep Neural Networks and Machine Learning: Machine learning techniques, especially deep neural networks, have revolutionized audiovisual ability. Algorithms such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) excel at tasks such as object detection, path recognition, and behavior prediction, allowing AVs to navigate within complex and dynamic environments.

3. Simulation Environment: The emergence of advanced simulators, such as CARLA, has provided a platform for testing and validating AV algorithms in a variety of situations without real risk. The simulator allows researchers to explore a variety of scenarios, weather conditions, and traffic situations, thereby driving the development of audiovisual technology.

4. Automation level: Describing different levels of automation, from basic driver assistance to full autonomy, provides a framework for understanding the development of audiovisual abilities. These levels have guided research efforts and highlighted the challenges associated with the transition of control between humans and autonomous systems.

5. Combined test method: It is becoming increasingly important to integrate simulation testing and validation in the real world. This approach allows algorithms to be fine-tuned in a virtual environment before they are tested in the real world, thereby striking a balance between safety and real-world complexity.

6. Urban Environmental Challenge: Focusing on urban environments, characterized by complex traffic scenarios, pedestrians, cyclists, and varied road layouts, has become important. Research efforts have shifted to addressing the challenges of safe and efficient navigation in these complex contexts.

7. Human and AV interaction: The interaction between AV equipment and road users has received attention, with innovations in communication methods, gesture recognition, and interface design aimed at improving mutual understanding and safety.

8. Legal Framework: Governments and regulators around the world have begun developing guidelines and policies for AV deployment on public roads. This changing landscape shapes research priorities, highlighting the need for robust and verifiable AV systems. The research and innovation ideas presented in this article are relevant to these advances and challenges. The goal of achieving Level 4 autonomous vehicles using Deep Q neural networks reflects advances in AI-based decision-making. Incorporating the campus environment into the simulator emphasizes the importance of

AV testing in controlled, real-world situations. Additionally, the emphasis on following basic traffic rules and completing routes is consistent to ensure safe and predictable AV operation in complex real-world environments.

In summary, the evolution of audiovisual technology, from advances in sensors to regulatory considerations, serves as the foundation upon which the objectives of the document are formulated. By leveraging these developments, the research aims to contribute to the development of audio-visual systems capable of ensuring safe and efficient navigation in difficult urban environments.

D. LITERATURE REVIEW

There have been many studies that have attempted to increase the usability and efficiency of AVs. One way that this has been done is through traffic flow predictions, which can help AVs in planning efficient routes. Yin et al developed a fuzzy neural model that was able to effectively predict traffic flows in urban street networks [13]. Smith and Demetsky compared three different models for traffic flow prediction [14]. These models were Historical Average, ARIMA, and Backpropagation neural network. It was found that the Backpropagation model was the best. Villarroya et al. developed two models based on LSTM to predict traffic flow in Valencia, Spain that were shown to have a small amount of errors [15]. Another way has been through object detection and vehicle classification. Betke et al. developed a real-time vision system that can recognize and track vehicles, road boundaries, and lane marking [**?**]. Chen et al. implemented a hybrid DNN that outperforms traditional DNN on vehicle detection [17]. Petrovskaya & Thrun implemented a moving vehicle detection and tracking module that is able to reliably detect moving vehicles and track them from a fast-moving platform [**?**].

III. METHODS

The code for this project can be found on our Github.

A. CARLA ENVIRONMENT

The **CAR L**earning to **A**ct (Carla) simulator is used to train our agent, generate data, and compare AI driving versus human driving in exact scenario recreations courtesy of simulation. CARLA provides an environment in which we can simulate driving scenarios in various permutations of road congestion, weather conditions, vehicle types, obstacles, and road layouts. Carla uses a client-server architecture where the server will run the simulation and render the scenes in UE4. [10] CARLA has a C++ and a Python API, and the server connection is managed by sockets. We used CARLA Windows version 0.9.14. Download, installation, and configuration instructions can be found on their main homepage Training was done on the default map (map 10), and across various weather conditions such as day, night, rain, and fog which can be seen in Figure 1 below. The training was primarily done in low-density environments, and we attempted to recreate some of the simpler scenarios from the Carla real traffic scenarios. [11]

FIGURE 1. A series of images depicting Map 10 under different weather conditions from above. Top: sunny, clear skies Middle: Foggy afternoon Bottom: rain, night

The Carla simulator provides a realistic and versatile virtual environment for testing autonomous vehicles and offers a wide array of scenarios and conditions that closely mimic real-world driving situations. This allows researchers to expose their deep learning models to a diverse range of challenges, such as varying weather conditions, complex traffic patterns, and unexpected obstacles. By training and evaluating the neural networks within this dynamic simulator, the research findings are more likely to reflect the system's adaptability and robustness in actual on-road scenarios. This generalizability is crucial for real-world implementation, as it ensures the neural network's effectiveness across a broad spectrum of environments and driving conditions.

Furthermore, the utilization of the Carla simulator in the research paper facilitates reproducibility and comparability with other studies. The simulator provides a standardized

platform for testing autonomous vehicle algorithms, enabling other researchers to replicate the experiments and validate the results. This not only ensures the reliability of the findings but also allows for direct comparisons with other neural network architectures or training methodologies. The generalizability of this research is further enhanced by the fact that the Carla simulator is widely recognized and adopted within the autonomous vehicle research community, establishing a common ground for evaluating and benchmarking different approaches.

Our agent is fully self-contained. All automated driving operations are encapsulated in our vehicle, no outside sources are used outside of GPS information. Unlike most methods mentioned in Yurtsever et al, [2] core functions are not separated into modules.

There are three main parts to the code, the **simEnv and DQNAgent classes**, and the main function which contains the training loop.

1) simEnv Class

The simEnv class contains the environment data needed for the DQNAgent. It is responsible for starting and connecting the CARLA client and server, collecting all data necessary to determine the state of the agent, and resetting the environment once a simulation is over.

The code assumes that the Carla server will be up and running, and will attempt to connect the client to the server at localhost port 2000. After the connection between client and server is established, a number of important things happen. The environment will get blueprints, which enable the creation and usage of vehicles and sensors. Following that, vehicle settings are established. The ego vehicle was set to a minimum speed of 10 kph and given a turn speed of 15 kph. NPC traffic is then spawned into the simulator, and the environment is reset to enable testing.

2) Environment Reset

Resetting the environment involves spawning the agent, planning a route, attaching sensors, and activating sensors, and is done before any simulation to ensure consistency.

Every map on Carla has predefined spawn points. A vehicle asset that spawns in a spawn point is guaranteed to be facing the correct direction, and on a drivable road, but otherwise can appear anywhere on the map. For each simulation, a random start point is chosen. Similarly, a random end destination is chosen from the list of available spawn points on the map. This means that in every simulation the distance the vehicle must drive to achieve success is different.

Route planning is done by Carla's global route planner. Every map has waypoints in it, which are just markers in the map that contain a location, and a road direction. Road directions are instructions on how traffic should flow at a particular waypoint such as continue following the lane, change lane left, change lane right, turn left, and turn right. This is the same service Google or Apple Maps would provide to a human driver on the road. The route planner will accept the start location of the simulation and the end location and connect a series of valid waypoints making a drivable route from where the car is to where it should be by the time the simulation ends. Simulations are designed to be point-to-point, when the agent arrives at the destination the simulation is over.

Attaching Sensors is done prior to the agent moving. All attachment transformations are relative to the car model chosen. The agent is driving in the vehicle.micro.microlino car model from Carla's vehicle options. The microlino is a small car model analogous to a smart car, and the most similar option to a golf cart that was found in the simulator's catalog. Sensors require a blueprint, a transform, and a parent to attach to. Blueprints for each sensor have different attributes based on the sensor types. The cameras use a width of 640 and a height of 360, and have a field of view of 110 degrees. Radars record 5500 points per second and have a vertical and horizontal field of view of 30 degrees and a range of 100 meters. All other sensors use default settings. After getting the relevant blueprint and adjusting the attributes, each sensor needs a transform location to spawn in. Transforms are x,y,z locations, and may also be paired with rotations which allow for pitch yaw and roll to be supplied. There are three cameras on the agent, one is attached to the hood of the agent, one is attached to the left side mirror, and the final camera is attached to the right side mirror. This mimics the visual data available to human drivers excluding the rearview mirror. There are two radars on the agent, one located on the hood of the agent, and the other attached to the bumper. There is no usable physical space inside vehicles in Carla, so the GPS and the IMU are just placed in the center of the vehicle. All sensors must be attached to an anchor point which is the agent. In addition to sensors matching the physical golf cart, there is an additional sensor attached to the agent. This is a collision sensor, which records any time the vehicle collides with an object. This sensor is used to unfailingly detect when the agent collides with any objects and apply rewards and penalties.

Activating sensors is done by using the .listen() method. Each sensor returns a type of raw data that must be processed by the network. This data is received by using a lambda function in conjunction with the sensor's listen function. A sample call to a listen function looks like this:

```
self.left_cam.listen(lambda image:
self.rrgb_callback(image, self.sensor_data))
```

The listen will gather the raw data from a sensor which will then be passed into a callback function that processes the data for use. Each sensor has a unique callback method that processes the data and stores it in a dictionary to be used at a later time. For example, camera images are converted to numpy arrays, and scaled to just RGB values, eliminating the alpha channel Carla images also have. Each sensor reports data when the server ticks.

To get the state of the environment there is a get_state() method. Any time the get state is called, the weather will change, and then sensor data is gathered from the sensor

data dictionary, which has been recorded and stored by each sensor's listen function.

The environment currently supports 4 actions which stay straight (0), turn left (1), turn right (2), and brake (3). The turn speed is set to 15 kph, and the agent speed is set to cap at the speed limit of the zone the agent is in.

The step function of the environment accepts an action from the agent, will perform that action, gets a new state, apply a reward/penalty, and send a done flag back. The new state information is received by performing the action and then calling the get_state function described above. The reward/punishment is assigned as an integer value and is additive. Any collision is worth -250 points. At this point in time all collisions are worth the amount of points. There is no distinction made between colliding with a building versus another vehicle versus traffic signage. If the vehicle is below the minimum speed a -2 penalty is added. The speed penalty is to encourage movement and deter driving in circles. Then there is a penalty applied based on the Gaussian distance the vehicle is in at the end of the simulation to where the destination is. If the agent is 0-5 meters away it receives +50, 6-15 meters away +15, 16-30 meters -20, 31-45 meters -30, and anything above 45 meters away receives a -50 penalty. If the agent is at the destination, it receives +200 points. The final item returned by the step function is the done flag. Simulations are done if the agent collides with an object, or the agent reaches the end destination.

Once a simulation is over the sensor suite of the agent must be handled. Each sensor is set to stop listening, which turns off any data communication actions the sensor would engage in. Then each sensor is destroyed and deallocated. This is required as sensors are attached to the agent, and at the end of a simulation, the agent must be despawned so it does not clutter the road for the next simulation. Additionally, the sensor data dictionary's collision value must be reset to false. The collision sensor only records collision events, which means that it is essentially a unidirectional signal that can only set the collision flag to true.

3) DQNAgent class

The DQNAgent class contains all functions relevant to the agent. It requires six parameters to be initialized: the shape of image inputs, the shape of radar inputs, the shape of speed input, gps input shape, future gps input shape, and the number of possible actions. The image inputs are (360,640,3) tensors reflecting the height, width, and RGB values received from the cameras. The radar inputs are of shape (5500, 4) which reflects the number of points per second each radar records, and the 4 values which accompany each point which are the polar coordinates (R, θ) and the distance and velocity relative to the radar. The speed input is of shape (1,) which is just a single value to represent the vehicle's current speed. The GPS and future GPS share the same shape, which is (3,) representing the latitude, longitude, and altitude of the GPS coordinate. GPS input represents the current position, and future GPS input represents the next waypoint the vehicle should move to on its path. The final parameter to initialize a DQN agent object is the number of actions, which is four (straight, left turn, right turn, brake). The agent initializes with an epsilon value of 1, a gamma of .99, and an epsilon decay rate of .995. The agent also has a replay buffer with a size of 1000. The agent uses the Adam optimizer and mean squared error (MSE) as the loss function.

The performance of the Q-network is measured using MSE which indicates how the network's predicted Q-values match the target Q-values. The error value is used to optimize the network's weights to minimize this loss during training. A lower loss value should enable the agent to make better decisions over time, rather than focusing on accuracy in the same way as in traditional supervised learning tasks. The Q-values represent estimates of the expected cumulative rewards for taking different actions in various states, and the network is trained to reduce the difference between its predicted Q-values and the target Q-values.

Like all Deep Q networks, the agent initializes with a q network and a target network that go through an identical create_model() function, and the target network adopts the weights of the q network. The q network will explore and learn, and update the target network as the learning occurs.

The create_model function creates input layers corresponding to the shapes mentioned above, with one input layer per sensor, as well as an input layer for speed and the future GPS location the vehicle should move towards. We use the Xception network with imagenet weights and set include_top to false. Each of the three images received by the network is passed through Xception for recognition and then goes through a flattening layer. The two radar inputs are also flattened. Each input then goes through a concatenate layer, concatenated along the first axis. Then there are three dense layers of sizes 512, 256, and 128 all of which use the relu activation function. The output layer of the model is a dense layer that accepts the number of actions and the final hidden layers as input. Lastly, the model is compiled with mean squared error for loss, and uses the Adam optimizer with a learning rate of .001. In total the model has 798,533,036 parameters, 798,478,508 are trainable, and 54,528 non-trainable parameters.

An epsilon greedy approach was used to select actions. If a random number is less than the epsilon value of the network, A random action is selected from the list of possible actions to stay straight, turn left, turn right, or brake. If the random number is greater than epsilon, we give the current state to the network and select the highest q value, and return that as the action to take.

The agent is updated in the update_q_network, which occurs every 5 steps of the simulation. A random sample of 10 is selected from the replay buffer. Each input is converted into a numpy array and reshaped so that the first dimension is None. The predict method is called using future inputs and the maximum predicted q value is selected, which represents the next state based on the inputs given. Next, the target q value is calculated using the Q equation from above. The rewards of

an episode are scaled by the discount factor of future rewards, the maximum value of the next q, and the completion state of the episode.

```
target_q_values = rewards + (1 - dones)
    *self.gamma * max_next_q_values
```

After this, the q value for the current state is calculated. The current q values are used to select the action taken based on the inputs. This is done using this code:

q_values_actions = tf.reduce_sum(q_values * tf.one_hot(actions, self.num_actions), axis=1)

A one-hot encoding of the actions is created with a depth of num_actions which is 4. The encoding is multiplied element by element across the q values. Then reduce_sum is used to select the q value of the appropriate actions given the state.

The loss of the network is calculated using mean squared error comparing the target q values and the q value actions from above. Finally, the gradients are calculated using gradient tape, and applied to the network using this code:

```
loss = tf.keras.losses.MSE(target_q_values,
q_values_actions)

gradients = tape.gradient(loss,
self.q_network.trainable_variables)
self.optimizer.apply_gradients(zip(gradients,
self.q_network.trainable_variables))
```

4) Simulator Scenarios

Training is done in real-time on map 10 of CARLA. This map is a downtown area featuring multiple intersections, tight turns, many traffic lights, crosswalks, buildings, and foliage. The road network is connected by an outer road that loops around the entire drivable area of the map. Every time the agent gets the current state of the environment the weather in the simulation will change. There are 3 environmental axis for weather: light-dark, cloudy-clear, and dry-rainy. The weather will change by a preset speed factor and the amount of elapsed time in a given scenario. The speed factor and the elapsed time are used to adjust the altitude and azimuth of the sun, as well as the number of clouds in the sky and amount of rain present in the simulation. The severity of weather changes can be adjusted by altering the weather speed factor. In conjunction with changing weather, there are also other traffic present in each simulation. The amount of traffic is customizable, but typically simulations were run with 5 other vehicles present representing a small amount of traffic. Traffic in the simulation is controlled by CARLA's traffic manager system and were set to run on autopilot. This means simulator traffic outside of the agent followed the rules of the road. Traffic was an added obstacle but was not meant to unduly hinder the agent with erratic driving or aggressive behavior.

5) Training Loop

Training is done in two loops. An outer loop runs for a set number of episodes, and an inner loop runs an individual episode. Prior to entering training loops the environment and agent are initialized. Once in the training loop the agent will get the current state, choose an action, get the next state by performing that action, and update the current state, and then update the network. The target network is updated every 5 steps. If the done flag ever returns true, meaning the agent made it to the destination, we break out of the inner loop. Additionally, each episode has a two-minute limit, after two minutes if the agent has not crashed into an object or reached the destination, the episode ends. After an episode ends the epsilon value is decayed, and a new episode begins. Pseudo code for the training loop is below.

```
for episode in range(number of episodes)
    reset environment
    get current state
    for step in range(max steps per episode)
        select action
        perform action
        update state
        update q network
        add to reward tracker
        if done or time_limit:
            break
    decay epsilon
    stop and destroy sensors
```

B. PHYSICAL EQUIPMENT SETUP

We have equipped a golf cart with three cameras, two radars, an inertial measurement unit (IMU), and a GPS to record data for a new college campus driving dataset and to add MTSU as a map within the CARLA simulator. We have selected our sensor suite to fuse sensor capabilities to make the vehicle more resistant to bad weather and to be capable of navigating in denser urban environments. Cameras alone do not suffice due to visibility concerns in the weather and at night. Radars cannot reliably differentiate road signs from surrounding foliage and can have difficulties detecting pedestrians for the same reasons. [2] IMU-GPS systems have impaired reliability in tunnels from high buildings, and mountainous areas and are only reliable for high-level route planning. [2] Combined, our sensor suite gives the AV the ability to plan a global route and respond to local stimuli from start to end. What follows will be a description of the physical devices in place, and their settings as applicable.

1) Cameras

A camera is a device that captures images and will be used here to detect obstacles and lane-keeping. There are three cameras attached to the golf cart. One on the left side is attached to the front left beam of the roof support column. One on the right side is attached to the front right beam of the roof support column. And the final camera on the front of the golf cart. The cameras are identical other than their positioning. They are**Blackfly S GigE** cameras. More information on the cameras may be found here.

2) Radars

A radar is a device that uses radio waves to determine the distance of objects relative to the radar and will be used to monitor the golf carts' surroundings. There are two radars attached to the golf cart. One on the front is attached to the hood, and the other on the back is attached to the bumper. The radars are identical other than their positioning. They are **DesignCore® Antenna on Package mmWave Radar Sensors**. Radars have benefits over cameras because they are hardly affected by illumination or harsh weather conditions. The datasheet for the radar may be found here.

The cameras and the radars work together to provide vision for the car, identifying hazards and viable path options. The cameras and sensors have an effective range and blind spots, which can be seen in the figure below.

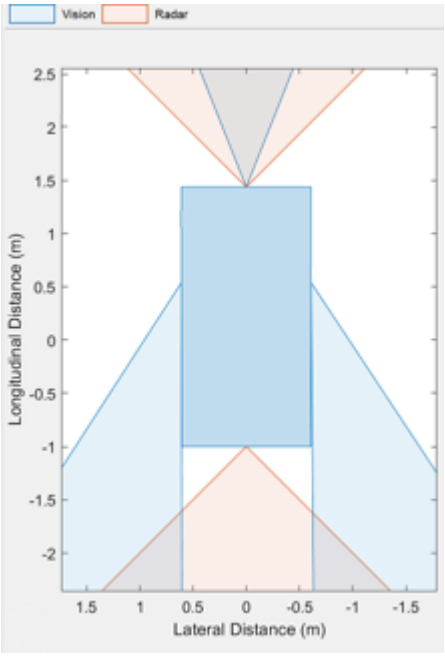

FIGURE 2. A top down view showing the vision cones for cameras and radars on the golf cart demonstrating blind spots

3) Inertial Measurement Unit

An inertial measurement unit measures and reports the force and angular rates of the golf cart via accelerometers and gyroscopes. The IMU will be used to monitor the golf cart's speed. The IMU is a Vectornav **VN-100 IMU** from the Vectornav-100 rugged development kit. More information on this sensor may be found here.

4) GPS

A GPS will identify the position of the golf cart and will be used to help map the MTSU campus as the cart gathers data. It is a **CANmod.gps** from css electronics. More information on this sensor may be found here.

The exact environmental setup is summarized in the table below.

TABLE 1. A table summarizing sensor models attached to the golf cart.

Sensor	Model
Camera	Blackfly S GigE
Radar	DesignCore® Antenna on Package mmWave
IMU	Vectornav-100
GPS	CANmod.gps

IV. RESULTS AND DISCUSSION

The distances measured were the distance traveled from the start point, the distance remaining to the destination at the end of a simulation, the total route distance, speed, and the reward signal the model received, and the reward received versus the weather scenario.

A. AGENT SPEED

In Figure 3 below, it can be observed that the agent traveled most frequently in two speed ranges 6.5-12 kph and 34-39.5 kph. The next most frequent speed ranges were 23-28.5 and 28.5-34 kph. The agent has a 60% chance of driving at 23 kph or above. The agent had a maximum speed of 74 kph, and an average speed of 24.2 kph

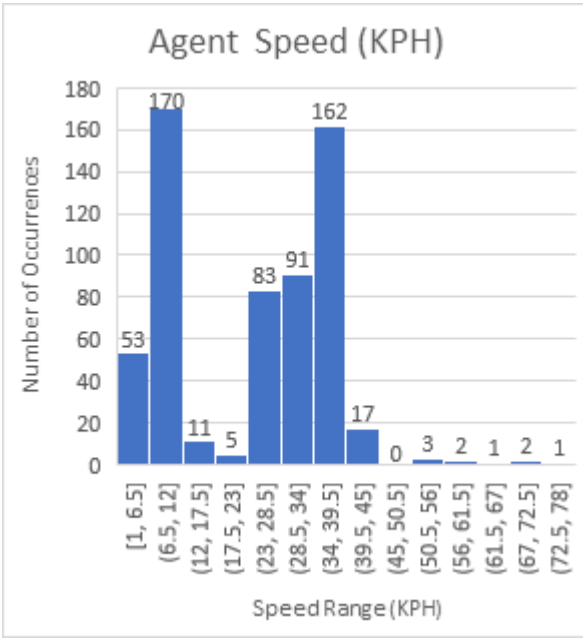

FIGURE 3. A Figure displaying the speed the agent traveled at in simulation episodes in KPH.

. The results make sense given that the agent was set to follow the speed limit and the most frequent speeds align with the speed zone values of map 10 in Carla. The high frequency of low initial values are likely due to the fact that the agent needs time to speed up, and has a minimum speed of 10 kph. Speed values above 50 kph are unlikely, but should not have occurred given that there are no speed zones on map 10 with a speed limit that high.

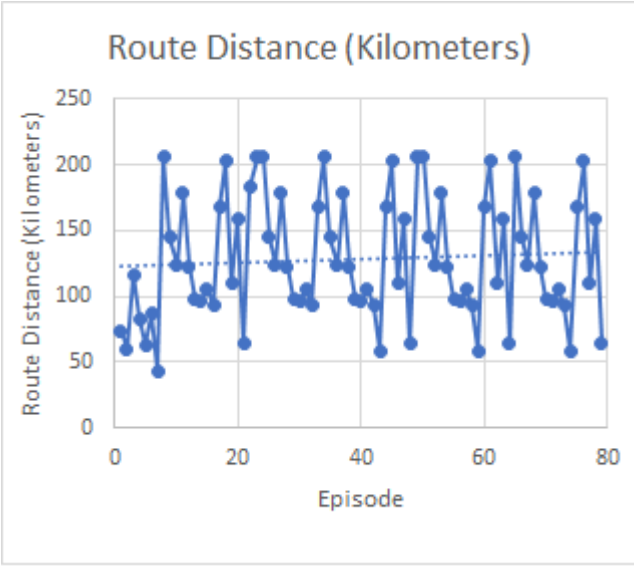

FIGURE 4. **A Figure illustrating the route distances in each episode**

B. ROUTE DISTANCE

In Figure 4 above, the total route distance varies per simulation episode. The maximum route distance was 205 kilometers, the minimum distance was 43.35 kilometers, and the average route distance was 128.04 kilometers. Route length was decided randomly at the beginning of each episode. Some routes were easy featuring only a single turn, others were more complex and required navigating through traffic lights and road signs. While route length was inconsistent, it more accurately reflects real driving scenarios.

C. DISTANCE TRAVELLED AND REMAINING

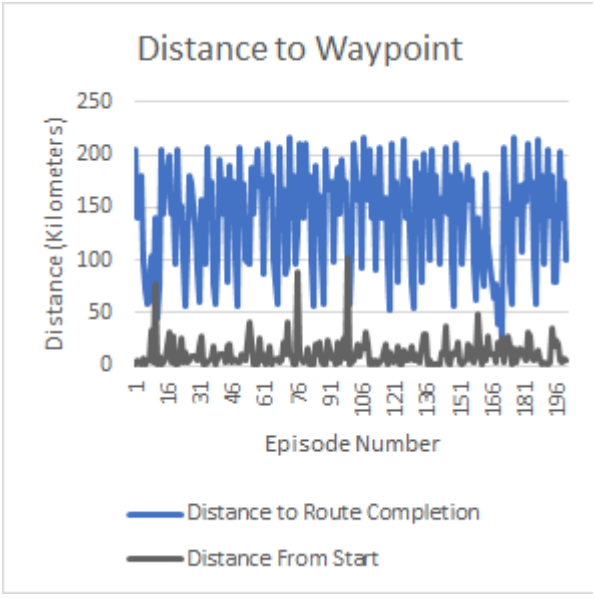

FIGURE 5. **A Figure showing the distances from the start point of an episode (grey) and the distance to the end of the route (blue)**

In Figure 5 the distance the agent had to complete the route is on top in blue, while the distance the agent traveled from the start point is below in grey. The maximum distance left in a route was 216.90 kilometers, the minimum distance left in a route was 22.13 kilometers, and the average distance left to complete in a route was 141.30 kilometers. The agent is not currently equipped with recovery mechanisms like reversing and is set to fail fast. Anytime the agent collides with an obstacle, the episode is over. If the agent navigates too close to any obstacle, it cannot reverse course and can only crash and end the episode. The fail forward nature of the training is meant to encourage flawless driving but also means that any errors are punished heavily, the effects of which can be seen in the distance left to complete a route.

The minimum distance traveled from the episode start point was 0.59 kilometers, the average distance was 12.54 kilometers, and the maximum distance was 130.24 kilometers traveled from the starting point. The reasons for this distance from the starting point are the same as for the distance remaining to complete the route. Any collision regardless of the severity ends the episode. Furthermore, the agent has 10 possible actions. Six of these actions are turns of 25% or greater which means that 60% of all possible actions are turns. This means the agent has an initial bias to select a turn action, which typically resulted in perpetual turning or "doughnutting" from the agent. To combat this a doughnut penalty was applied to any turn action with a turn value of 25% or greater, and if the vehicle ever had a cumulative turn time over 3 seconds the simulation would end to prevent wasting time endlessly turning in circles.

Refinements to navigation can greatly improve the performance of the agent. Further training time and better traffic rule enumeration will enable the agent to complete routes more frequently, up to our goal of 80%, and enable the agent to better avoid road hazards.

D. REWARD BY WEATHER TYPE

In Figure 6, the rewards the agent received for a weather type are displayed. There are 11 different weather types which are: 0 (Sunrise), 1 (rainy sunrise), 2 (cloudy sunrise), 3 (sunny afternoon), 4 (rainy afternoon), 5 (cloudy afternoon), 6 (night), 7 (cloudy night), 8 (rainy night), 9 (heavy foggy), and 10 (rainy, foggy, night). There is a consistent trend downward in the rewards the agent received for a given weather type. The agent performed best under weather condition 0, the sunrise, and fairly similar under conditions 1-4. The agent performed the worst under conditions 9 and 10. Performing poorly in conditions 9 and 10, heavy fog and nighttime fog respectively, makes sense as they are the most difficult road conditions offered in training. As mentioned in Vargas et al. [5], harsh weather effects heavily reduce camera effectiveness and impair the ability of the agent to navigate. The radar on the agent is meant to mitigate this, but given the results, it's clear further safety measures need to be implemented. Extra radar preprocessing can help to identify areas of interest in adverse weather and be used to focus image processing from

the camera. Higher levels of data preprocessing can enable the vehicle to traverse safely in poor weather conditions. Interestingly, the agent performs similarly in conditions 1-6, which indicates consistency of performance even in rainy conditions, and that the main performance issues are associated with night and fog.

Further data preprocessing to focus on safety measures can boost model performance in the future. Passing images through the Xception network is not enough, images need to be recognized and tied to a control module with the appropriate action for any given road signage. For safety and legal reasons, it must be possible to adjust the agent's behavior if road laws are ever changed. The agent is capable of recognizing stop lights and road signage, ut is not equipped with adequate ability to perform the correct action for the signage that has been identified. Radar inputs are sent to the network as 5500x4 matrices times 2 for a front and back radar which makes for 44,000 points of data to examine every time the agent checks the state of the environment. Rather than sending in the entirety of radar data, data should be sent in only with relevant information such as what cluster of points is approaching the vehicle, and how fast they are approaching.

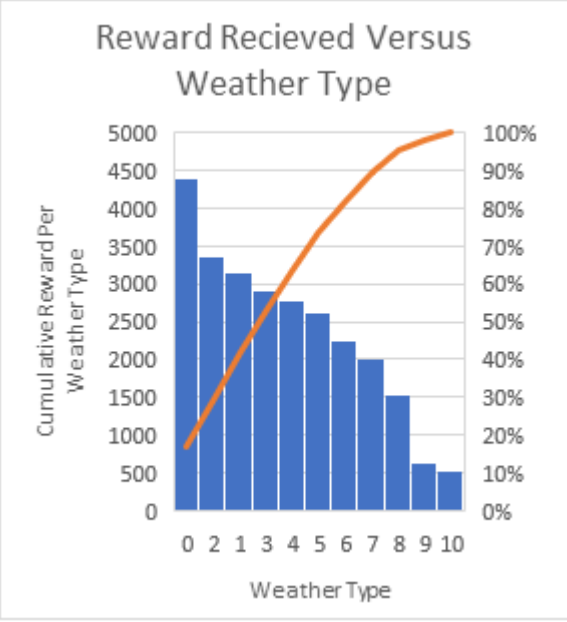

FIGURE 6. A Figure displaying the rewards the agent received for a given weather type.

That data is present for the agent but is also surrounded by radar data that is unimportant such as radar pings off of buildings, vegetation, and other objects that are of no interest to a car. The only radar data the network should see are radar pings that represent objects that may hit the car or objects the car might hit. This means objects within a range determined by the speed of the agent, so that when the agent is moving faster objects are identified with more stopping time, and objects with a certain velocity are recognized by the agent so it may move out of the way and avoid a collision. Once these tweaks are implemented the agent may perform better in poor weather conditions.

E. FUTURE WORK

In the future, a network will be run through more complex scenarios, applying the NHTSA precrash typology. These include scenarios such as yielding to emergency vehicles, yielding in a traffic circle, and weave lane conflicts upon entering/exiting a highway. Additionally, all traffic in training were following CARLA simulator AI to drive on autopilot and according to the rules. We hope to implement scenarios where walkers cross the street at incorrect spaces, cars driving erratically and running lights, and ignoring traffic signs. We hope to achieve a route completion rate of 80% by reorganizing local planning. Furthermore, we hope to compare the agent to human drivers and use that data to fine-tune agent actions. Once our golf cart is finalized we also intend to gather test data in a college campus environment, generate an HD map of the MTSU campus, and add MTSU to CARLA as a drive-able map.

REFERENCES

[1] B. R. Kiran et al., "Deep Reinforcement Learning for Autonomous Driving: A Survey." arXiv, Jan. 23, 2021. doi: 10.48550/arXiv.2002.00444.
[2] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," IEEE Access, vol. 8, pp. 58443–58469, 2020, doi: 10.1109/ACCESS.2020.2983149.
[3] N. Zaghari, M. Fathy, S. M. Jameii, M. Sabokrou, and M. Shahverdy, "Improving the learning of self-driving vehicles based on real driving behavior using deep neural network techniques," J Supercomput, vol. 77, no. 4, pp. 3752–3794, Apr. 2021, doi: 10.1007/s11227-020-03399-4.
[4] M. Vagia, A. A. Transeth, and S. A. Fjerdingen, "A literature review on the levels of automation during the years. What are the different taxonomies that have been proposed?," Applied Ergonomics, vol. 53, pp. 190–202, Mar. 2016, doi: 10.1016/j.apergo.2015.09.013.
[5] J. Vargas, S. Alsweiss, O. Toker, R. Razdan, and J. Santos, "An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions," Sensors, vol. 21, no. 16, p. 5397, Jan. 2021, doi: 10.3390/s21165397.
[6] G. Matthews, C. Neubauer, D. J. Saxby, R. W. Wohleber, and J. Lin, "Dangerous intersections? A review of studies of fatigue and distraction in the automated vehicle," Accident Analysis & Prevention, vol. 126, pp. 85–94, May 2019, doi: 10.1016/j.aap.2018.04.004.
[7] Y. Li, "Deep Reinforcement LEarning: An Overview." arXiv, Nov. 25, 2018. doi: 10.48550/arXiv.1701.07274
[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," Mach Learn, vol. 8, no. 3, pp. 279–292, May 1992, doi: 10.1007/BF00992698.
[9] T. Hester et al., "Deep Q-learning From Demonstrations," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.11757.
[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," in Proceedings of the 1st Annual Conference on Robot Learning, PMLR, Oct. 2017, pp. 1–16. Accessed: Jun. 16, 2023. [Online]. Available: https://proceedings.mlr.press/v78/dosovitskiy17a.html
[11] Osiński et al., "CARLA Real Traffic Scenarios – novel training ground and benchmark for autonomous driving." arXiv, Sep. 19, 2021. doi: 10.48550/arXiv.2012.11329.
[12] K. Chitta, A. Prakash, and A. Geiger, "NEAT: Neural Attention Fields for End-to-End Autonomous Driving." arXiv, Sep. 09, 2021. doi: 10.48550/arXiv.2109.04456.
[13] H. Yin, S. C. Wong, J. Xu, and C. K. Wong, "Urban traffic flow prediction using a fuzzy-neural approach," *Elsevier*, vol. 10, no. 2, pp. 85–98, 2002, doi: https://doi.org/10.1016/S0968-090X(01)00004-3.

[14] B. L. Smith and M.J. Demetsky, "Short-Term Traffic Flow Prediction: Neural Network Approach," *TRANSPORTATION RESEARCH RECORD*, pp. 98–105.
[15] C. Villarroya, C. T. Calafate, E. Onaindia, J.-C. Cano, and F. J. Martinez, "Neural Network-based Model for Traffic Prediction in the City of Valencia," *Procedia Computer Science*, vol. 207, pp. 552–562, 2022, doi: 10.1016/j.procs.2022.09.110.
[16] Poudel, S., Paudyal, R., Cankaya, B. et al. (2023). Cryptocurrency price and volatility predictions with machine learning. J Market Anal, Advance online publication. https://doi.org/10.1057/s41270-023-00239-1
[17] M. Betke, E. Haritaoglu, and L. S. Davis, "Real-time multiple vehicle detection and tracking from a moving vehicle," *Machine Vision and Applications*, vol. 12, no. 2, pp. 69–83, 2000. doi:10.1007/s001380050126
[18] M. N. Hasan, S. Hamdan, S. Poudel, J. Vargas and K. Poudel, "Prediction of Length-of-stay at Intensive Care Unit (ICU) Using Machine Learning based on MIMIC-III Database," 2023 IEEE Conference on Artificial Intelligence (CAI), Santa Clara, CA, USA, 2023, pp. 321-323, doi: 10.1109/CAI54212.2023.00142.
[19] X. Chen, S. Xiang, C.-L. Liu, and C.-H. Pan, "Vehicle detection in satellite images by hybrid deep convolutional neural networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 10, pp. 1797–1801, 2014. doi:10.1109/lgrs.2014.2309695
[20] A. Petrovskaya and S. Thrun, "Model based vehicle detection and tracking for autonomous urban driving," *Autonomous Robots*, vol. 26, no. 2–3, pp. 123–139, 2009. doi:10.1007/s10514-009-9115-1

JOSEPH MAY received his B.S. in computer science from Middle Tennessee State University, Murfreesboro, in 2022 and is currently working towards a master's degree. He served as a research aide to Dr. Joshua L. Phillips working on modeling the COVID-19 spike protein interaction with the ACE2 receptor. His research interests include deep learning, recommender systems, conversational recommender systems, and explainability scores.

KHEM NARAYAN POUDEL (M'15) received the B.E. and MS degree in electronics and communication engineering and information and communication engineering from the Tribhuwan University Kathmandu, Nepal, in 2010 and 2013, respectively, the MS degree in electrical and computer engineering from the University of Utah, Salt Lake City, UT, USA, the MS degree in computer science from the Middle Tennessee State University, Murfreesboro, TN, USA. He finished his Ph.D. degree in Computational Science from the Middle Tennessee State University, Murfreesboro, TN, USA.

He joined the Faculty Member of Middle Tennessee State University, in 2022. His research interests include High-Performance Computing, Big Data, and deep learning in bio-medical Signal processing.

DR. JORGE VARGAS earned his Ph.D. in Electrical Engineering from Florida International University in 2005. Dr. Vargas has over 15 years of experience in academia and has worked with multi-million-dollar funded projects in the fields of RF/Microwave Engineering, Microelectronics, Radar Sensors for Connected and Autonomous Vehicles, Sensor Fusion, and Machine Learning. Prior to joining MTSU in January 2021, he worked as an Associate Professor in Electrical Engineering at Florida Polytechnic University, where he taught several undergraduate and graduate courses on electrical circuits, electronics, logic design, radio frequency, microprocessor design, and sensors. Also, he had the opportunity to design and develop the Electrical and Computer Engineering curricula for the undergraduate and graduate programs, supported collaborative educational grants for external funding opportunities, provided outreach projects to the community, and served as an academic program coordinator in the Electrical Engineering department. His industry background at IBM involves product and process development of RF/Microwave devices.

SAMIR POUDEL completed his Bachelor's degree in Software Engineering from Pokhara University. He is currently pursuing a Master's degree in Computer Science at Middle Tennessee State University, demonstrating a strong commitment to academic and professional growth. Samir's research interests encompass the exciting fields of Artificial Intelligence, Big Data, and Machine Learning. With a focus on cutting-edge technologies, he aspires to make significant contributions to these domains and further his education and research at Middle Tennessee State University.

SAMMI HAMDAN is currently pursuing a Bachelor's degree in Professional Computer Science from Middle Tennessee State University. He plans to pursue a Master's degree in Computer Science in the near future, demonstrating a strong commitment to academic and professional growth. Sammi's research interests encompass the exciting fields of Artificial Intelligence, Biomedical Signal Processing, and Machine Learning. With a focus on cutting-edge technologies, he aspires to make significant contributions to these domains and further his education and research at Middle Tennessee State University.