Privacy-Aware Personalized Advanced Driver-Assistance Systems (ADAS)

Armand Ahadi-Sarkani, faculty advisor Professor Salma Elmalaki

1 Abstract

Many of the current state-of-the-art Advanced Driver Assistance Systems (ADAS) rely solely on sensory and driving data to warn or act on behalf of the driver in dangerous situations. This research project aims to develop an adaptive agent based on reinforcement learning that is able to quantify and understand various high-level human states. These human states will then be used in conjunction with data received from ADAS on vehicles to personalize and improve the relevance of warnings and actions that are taken by these systems. The measurements taken on the human state will be context-aware and thus place significant importance on respecting the privacy of users.

2 Purpose

Many of the current state-of-the-art Advanced Driver Assistance Systems (ADAS) rely on data received from sensors mounted on vehicles, in addition to driving data and telemetry generated with the motion of the vehicle and driver actions. Examples of such driver assistance systems employed on modern vehicles include lane departure warning, forward collision warning, and braking assistance systems. These driver assistance systems only rely on quantifiable, generic data from the vehicle, or the environment in which the vehicle is operating.

The motivation behind this research project is to explore the question of whether or not these systems can be improved by introducing the element of the human physiological state into their decision-making process. This proposed improvement can result in driver assistance systems that are more tailored to the state of mind, health, and predicted behavior of individuals. Additionally, this research project aims to make these systems privacy-aware, ensuring that personal data collected about a human's state does not get shared.

The field of introducing human-in-the-loop interaction to systems is still in its growing stages and is yet to be effectively applied to real-world scenarios that the general public encounters on a daily basis. Promoting safer driving and preventing dangerous road events through driver assistance systems is a top concern for engineers in this field. Introducing greater human interaction in the decision-making process of these systems will help advance this cause.

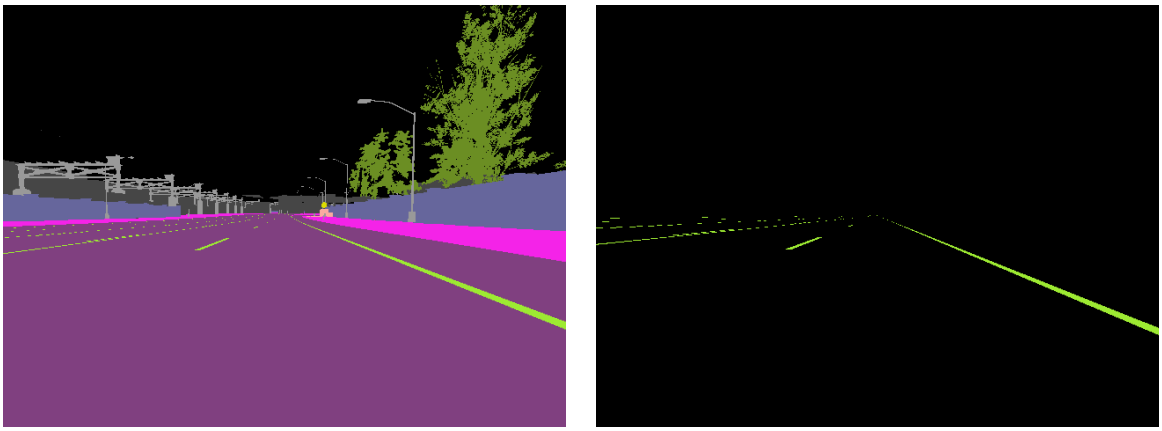
3 Objectives and Current Project Status

This research project aims to first understand how this sensory and telemetric data is collected to create ADAS systems, by using a modern, autonomous driving simulation environment. The simulation environment of choice is CARLA Simulator, an open-source simulator designed for autonomous driving research, based on the OpenDrive standard.

The first stages of this research project involved learning about CARLA Simulator and the various APIs and sensors available. The primary task completed was creating a lane departure warning ADAS sensor using CARLA APIs, and simulating the effects of this sensor using various driving scenarios, both autonomous and involving manual control of the vehicle. To explore how lane departure sensors work, this task was performed using two means and comparing the experimental results of each (sections 3.1 and 3.2).

Currently, the project has advanced to a stage where an intelligent agent is being developed using principles of Q-learning, a branch of reinforcement learning. This intelligent agent takes in driving data from the vehicle and integrates with sensory hardware to continuously learn about when to issue a lane departure warning to a particular driver.

3.1 Visual-based lane departure warning



(a) Unmasked output (b) Masked output

Figure 1: Sample outputs of semantic segmentation cameras

The first method of creating a lane departure warning sensor in CARLA involves visually interfacing with the built-in cameras available to vehicles in the simulation environment. The ideal camera for this is the semantic segmentation camera, which classifies different pixels in an image into multiple classes based on their meaning in the environment in which they reside. In CARLA, such classes of pixels include those that reference buildings, pedestrians, poles, vehicles, traffic signs, or road lines. The road line data is used in this implementation to capture data about the location of lane markings and how they change as the vehicle is in motion. Figure 1(a) shows the standard output of the semantic segmentation camera

taken in a customizable driving scenario. As images like figure 1(a) were received by the semantic segmentation camera, a masking algorithm was applied in order to hide all classes of pixels that did not correspond to road lines. Figure 1(b) shows the output of that masking algorithm applied onto figure 1(a), only showing the CARLA defined "road lines", or lane markings. This masking was applied in order to establish a basis for a lane departure warning system. This system operates by computing the difference between multiple masked frames at once (initially set at $n = 2$ frames) from the continuous sequence of frames received from the camera as the vehicle is in motion. Multiple linear algebra-based algorithms have been employed to compute this difference, as well as interfacing with APIs from SciPy. This visual method for creating a lane-departure warning ADAS system is a work in progress.

3.2 Location-based lane departure warning

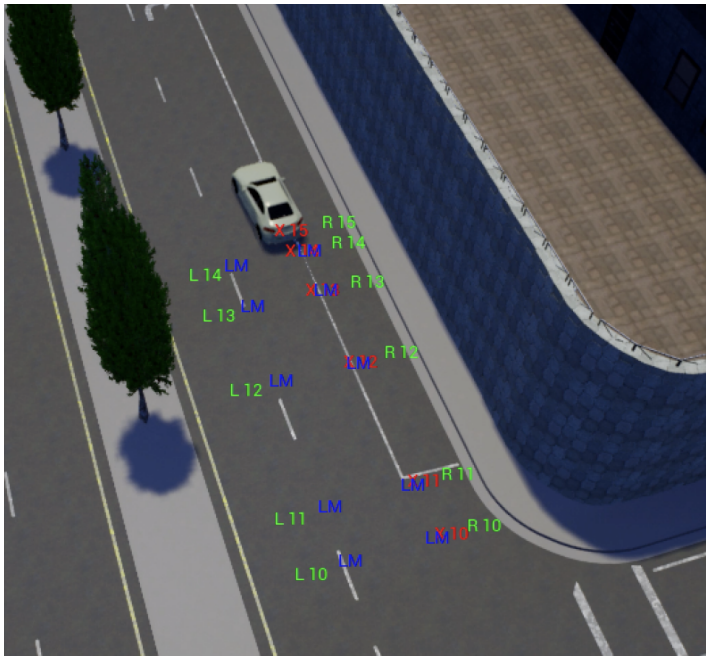


Figure 2: Location of lane markings on CARLA server

The second method of creating a lane departure warning sensor using CARLA involves analyzing vehicle location data in comparison with pre-defined waypoint data provided by the simulator. CARLA waypoints are vector-like objects that describe points in a road and provide information about the location and size of lanes. By continuously polling the location of the vehicle and relaying waypoint data about the nearest left and right lanes to the vehicle, an algorithm was created in order to determine the position of lane markings and how they relate to the vehicle in motion. Figure 2 shows the calculated locations of lane markings (blue), centers of right and left lanes (green), and the vehicle's location (red) on a simulator window, as a vehicle is in motion. If the vehicle is steering towards any of

these lane markings on its right or left side and reaches a preset threshold between these lane markings, a warning is engaged. This method is implemented through a client-server model; the client program is a manually-controlled driving environment, relaying waypoint and location data to the server program, which remotely is able to compute whether or not the vehicle in motion is approaching a lane marking. If the vehicle is approaching a lane from its left or right side, the server program sends a warning to the client. This client program is by default relaying data to the server at around 60 times per second, and the server program is able to filter that data and sample it for computation at a pre-defined sampling rate.

3.3 Q-learning agent and hardware interaction

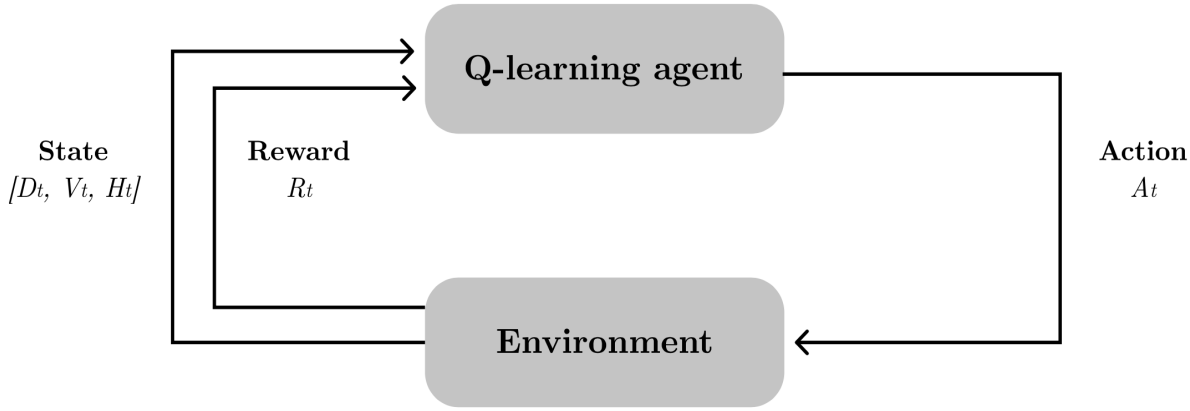


Figure 3: Diagram of a typical reinforcement learning scenario

The current stage of the project is taking all of the lane departure techniques learned in sections 3.1 and 3.2, and integrating them into a new intelligent agent that uses reinforcement learning to continuously learn from the behavior of a particular driver in order to issue warnings at the appropriate time. Reinforcement learning is a subset of machine learning, focused on developing computer agents that learn from their actions in a particular environment. In the case of this project, the agent is the intelligent ADAS sensor, the environment is the vehicle within CARLA Simulator, and an action can either be a warning issued or not issued. An agent starts by taking an action at a certain state, and receiving a reward from the environment based on how useful that action was at that particular state. The agent aims to maximize their rewards as they learn about their environment. The branch of reinforcement learning that will be used in this project is called Q-learning, which aims to identify an optimal policy for selecting an action at a particular state based on a specific function, shown in equation 2. Q-learning is model-free, meaning the agent is always learning, and does not require a set initial training phase. Figure 3 demonstrates the cycle of a typical Q-learning scenario for this application.

In addition, hardware is being introduced in this stage of the project. Currently, the only hardware device being tested with this project is the Oura Ring, a consumer-grade sleep and activity tracker that could provide useful information about a particular human's attentiveness and probability for distraction as they are driving.

3.3.1 Relevant equations

$$S_t = [D_t, V_t, H_t] \tag{1}$$

$$Q'(S_t, A_t) = Q(S_t, A_t) + \alpha * (R_t + \gamma * \max Q(S_{t+1}, A) - Q(S_t, A_t)) \tag{2}$$

3.3.2 Design of the RL Agent

Currently, the Q-learning agent being developed uses the location-based lane departure warning sensor that was created in previous stages of the project (see section 3.2). Future stages may see the addition of more ADAS sensors. The data from the lane departure warning sensor is relayed from a client program to the intelligent agent (which is the successor to the server program described in section 3.2).

The intelligent agent then uses the location-based data from the client as well as hardware data from the Oura Ring in order to formulate a three-element vector to classify a driver's current state, S_t (shown in equation 1). The environment is defined as executing an optimal policy over a Markov Decision Process (shown in section 3.3.7), which includes a state space, action space, and reward function. Based on a state S_t, the intelligent agent takes an action A_t, and waits until the driver responds to that warning to capture a final state, S_{t+1}. The reward function associates a reward R_t with an (S_t, A_t) pair. The Q-learning function, shown in equation 2, helps produce a "quality value", or Q-value, for a particular state transition based on the initial state/action pair, and the final state the driver has reached. α and γ are experimental parameters, referring to the learning rate and discount factor of the experiment, respectively. The Q-values for all of the state and action pairs are stored by the agent in a tabular format for each driver. The Q-learning algorithm is detailed in Algorithm 1.

Algorithm 1: Q-learning Lane Departure Warning Agent

Hyper parameters: α, γ, ϵ;
Require:
States $S_t = [D_t, V_t, H_t]$;
Q table $Q(S_t, A_t)$;
Actions A_t = {0: "no warning", 1: "warning"};
Iterations $iterations = 100$;
State vector size $V = 20$;
Vector sampling rate $T_s = 0.1$;
while *true* **do**
  $S_0 \leftarrow$ State(); // generate state object
  for *i in range(0, iterations)* **do**
    $stateVector = []$;
    $A_i \leftarrow$ chooseAction(S_0);
    sendActionToClient(A_i);
    // vector polls multiple states to model response time
    for *j in range(0, V)* **do**
      $S_j \leftarrow$ State();
      append(stateVector, S_j);
      sleep(T_s);
      if *length(stateVector) >= 1 and isSafe(S_j) and isUnsafe(S_{j-1})* **then**
        $V \leftarrow j$;
        break;
      end
    end
    $R_i \leftarrow$ rewardFunction(S_0, A_i, S_{V-1});
    $Q(S_0, A_i) \leftarrow Q(S_0, A_i) + \alpha * (R_i + \gamma * \max Q(S_{V-1}, A) - Q(S_0, A_i))$;
    $S_0 \leftarrow S_{V-1}$;
  end
end

3.3.3 State space S

The first element of the state vector, D_t, is a discrete value that represents the vehicle's distance from either of the adjacent lane markings. There are currently twelve distance states. The second element, V_t, is a discrete value that represents the vehicle's current velocity relative to the posted speed limit on their particular stretch of road. The third element, H_t, is a discrete value that classifies the driver's human state into several categories based on attentiveness data received from the Oura Ring.

3.3.4 Action space A

The action space A includes two possible actions that the lane departure agent could take at a given state S_t. The agent could either choose to issue a warning (1) or not issue a warning

(0). The agent chooses an action by balancing exploration and exploitation. Based on the value of an experimental parameter ϵ, the agent could choose to explore a certain percentage of the time by choosing a random action. In other instances, the agent will return the action that results in the maximum Q-value for the given state, by $max(Q(S_t, 0), Q(S_t, 1))$.

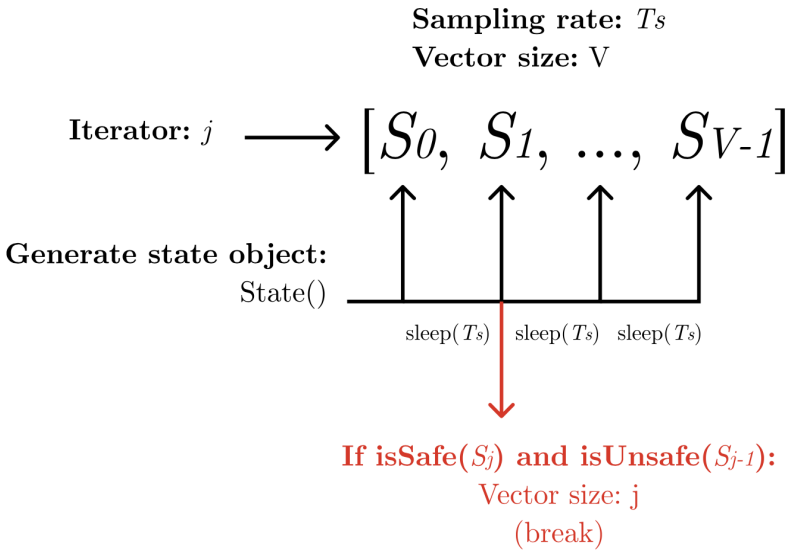


Figure 4: Diagram of polling of states through vector

3.3.5 Reward function $R(S_t, A_t, S_{t+1})$

The reward function associates a reward R_t with a given state, action, and next state tuple (S_t, A_t, S_{t+1}). This reward measures the effect of action A_t in promoting a state transition between S_t and S_{t+1}, and whether the transition between S_t and S_{t+1} was safe. If issuing action A_t resulted in a transition from an unsafe state to a safe state, the reward is generally positive. Similarly, if issuing action A_t resulted in a transition from an unsafe state to a safe state, the reward is generally negative. The Markov Decision Process in section 3.3.7 explicates the possible state transitions and the reward associated with each.

In figure 4, the polling of states in algorithm 1 is visualized. The initial state is always S_0, and the final state is determined by constantly generating new states, sampled at a sampling rate T_s, up to a pre-defined vector size V. If the algorithm encounters a state that is classified as "safe" when the previous polled state is classified as "unsafe", this indicates that the driver has made some type of major corrective action. In this case, the polling of states is halted, and the new vector size becomes the current index of the vector in which this corrective action was encountered (j).

3.3.6 Real-world driving scenarios

Several real-world driving scenarios have been applied to the client program that runs on the CARLA Simulator. Each of these scenarios can optimize for throttle, steering, and response time. There are four main types of response time optimizations that are done to create four distinct driving scenarios. The first is a driver with a fast response time; when this driver is alerted of a lane departure warning, they quickly and aggressively steer towards the center of the lane. The second is a driver with a slow response time; when this driver is alerted of a lane departure warning, they take more time to react to the warning, and steer less aggressively. The third is a cautious driver, which acts the same as the slow response time driver, except, before taking a corrective action towards the center of the lane, they wait and observe their surroundings. The fourth type of driver is a drowsy driver– which consistently swerves around their lane, gradually decreases their speed, and responds to warnings less effectively than the other drivers. There is also a control experiment, which can be applied to any driver by the agent, that only issues a lane departure warning when the driver has actually crossed an adjacent lane.

3.3.7 Markov Decision Process

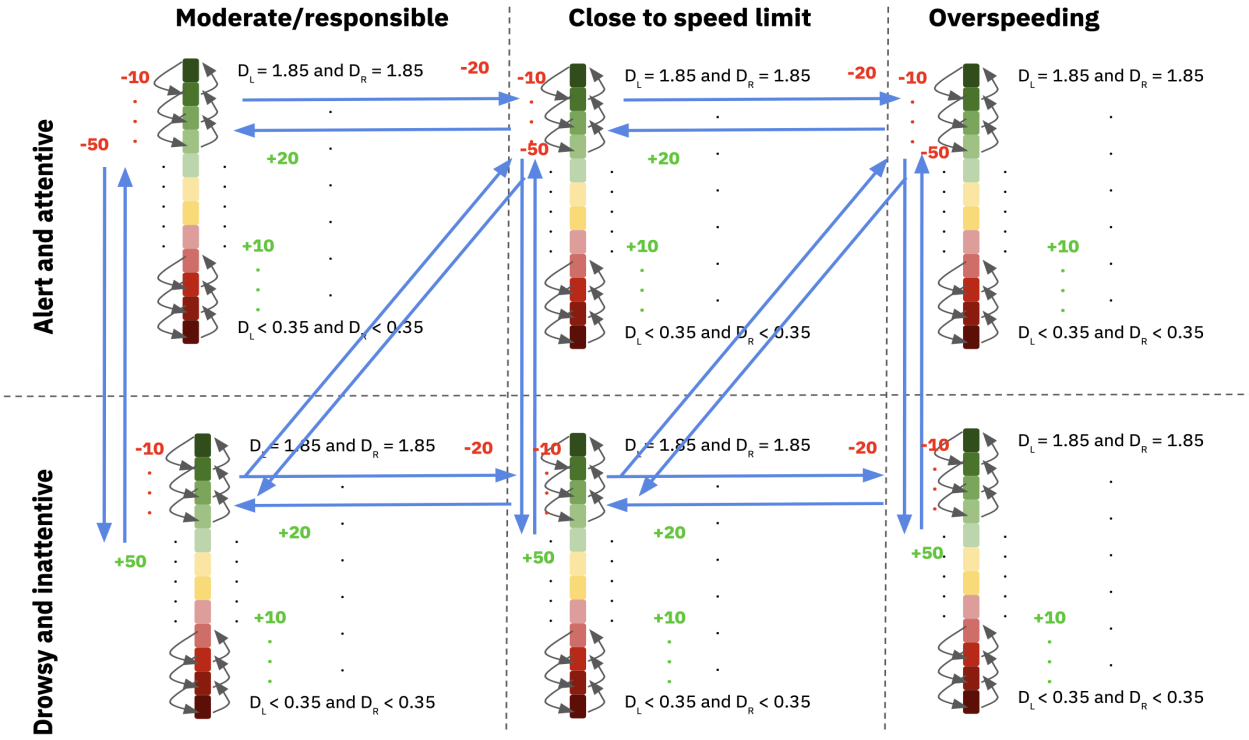


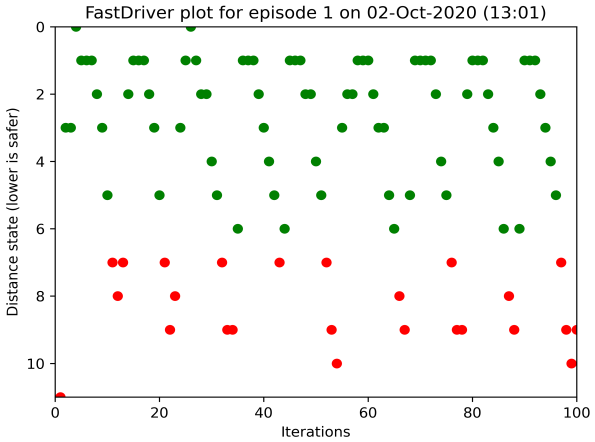
Figure 5: Markov Decision Process

The environment is described using a Markov Decision Process where an optimal policy is executed. In figure 5, the distance state D_t is separated into D_L and D_R, which represent the distance to the left and right lanes, respectively. The blue arrows represent major state

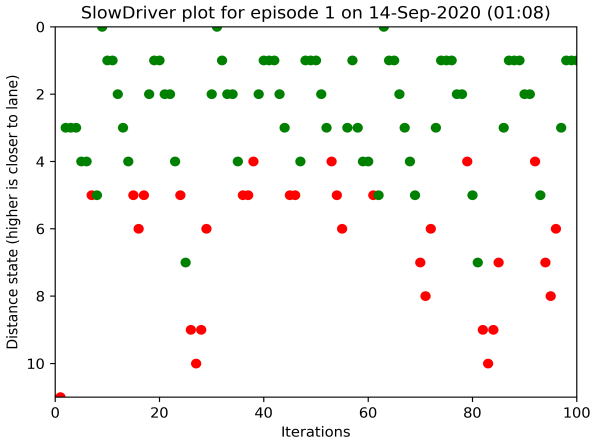
transitions. They are shown only once for clarity, but are possible for all left/right lane distance values in state pairs. Each state transition is associated with a positive (green) or negative (red) reward, independent of the action taken by the agent. The action taken is additionally accounted for in the implementation of the reward function.

The state at the top left corner of the diagram represents the most safe state: the vehicle is centered in between the left and right lanes ($D_L = 1.85$ and $D_R = 1.85$), the driver is alert and attentive, and they are driving at a moderate/responsible speed. Similarly, the state at the bottom right corner of the diagram represents the most unsafe state, reachable by a variety of state transition combinations.

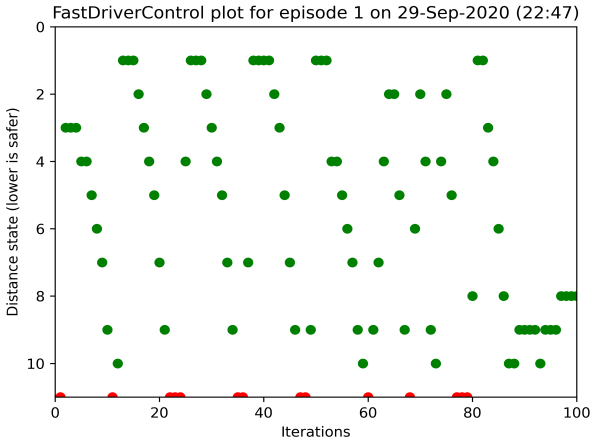
3.3.8 Sample outputs



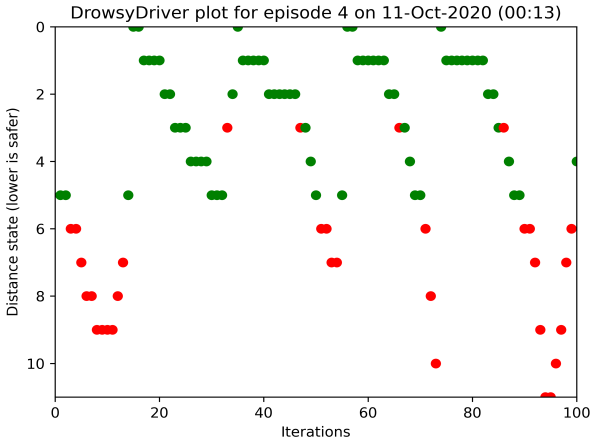
(a) Fast response time driver



(b) Slow response time driver



(c) Control experiment

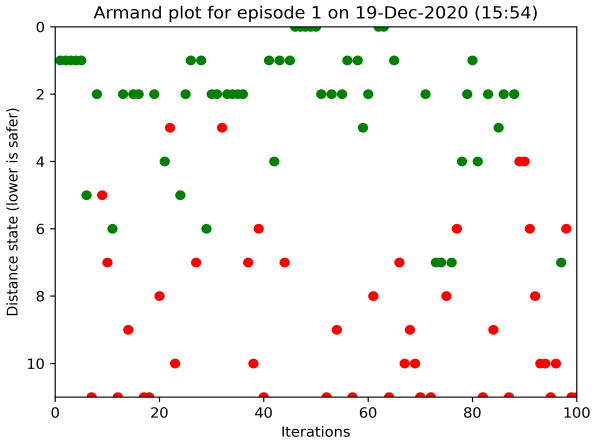


(d) Drowsy driver

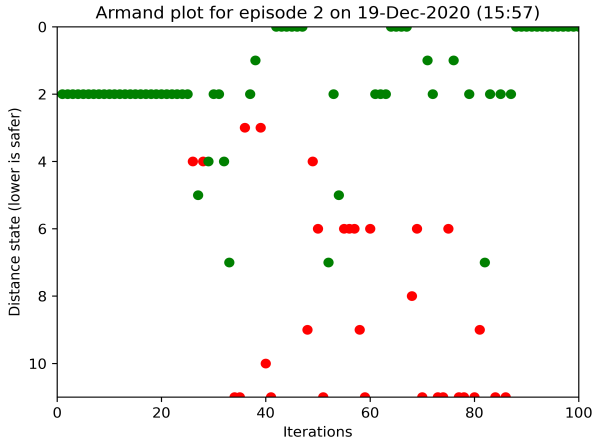
Table 1: Plots of actions taken over time for different drivers at different distance states

Table 1 displays four sample outputs for plots displaying the actions taken over time for different distance states. The x-axis is time, measured in number of iterations. One plot shows one episode of learning (1 episode = 100 iterations in this example). The y-axis is in increasing order from top to bottom, representing the distance state of the vehicle relative to their lane. The lower the distance state number, the closer the vehicle is to the center of the lane. When a warning is issued at a given iteration, distance state pair, a red point is placed on the plot. When a warning is not issued, a green point is placed.

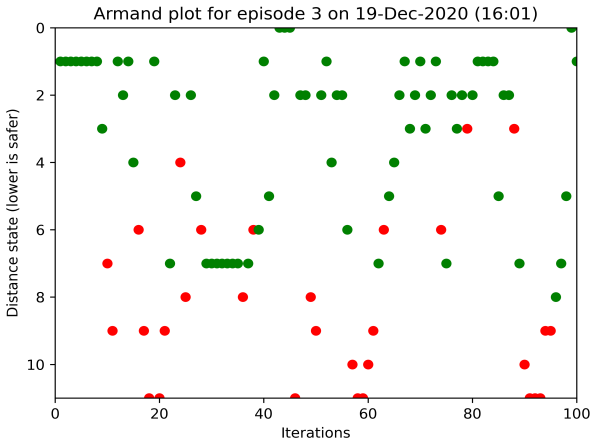
These four sample outputs demonstrate the learning of the agent applied on different types of drivers. These drivers are simulated in a scripted driving environment, where driving behaviors are automated. Table 1(a) shows the warnings that are issued in a given episode for a driver with a fast response time, and table 1(b) shows the same plot for a driver with a slow response time. The agent, by this sample, has learned that it can issue warnings (red points) for the fast response time driver much later than for the slow response time driver, since they take corrective actions in less time. In the slow response time plot, the red points occur much sooner, since they occur at lower/safer distance states. Table 1(c) shows the control experiment, a baseline metric for judging the performance of the agent. The control experiment only issues the warning when the driver has actually crossed an adjacent lane, demonstrated by the red points only placed at the most unsafe state. Table 1(d) shows a sample output for a drowsy driver, which makes unsafe maneuvers indicative of an inattentive human state. The agent learns to issue the warning even at states that may be considered safe for other drivers.



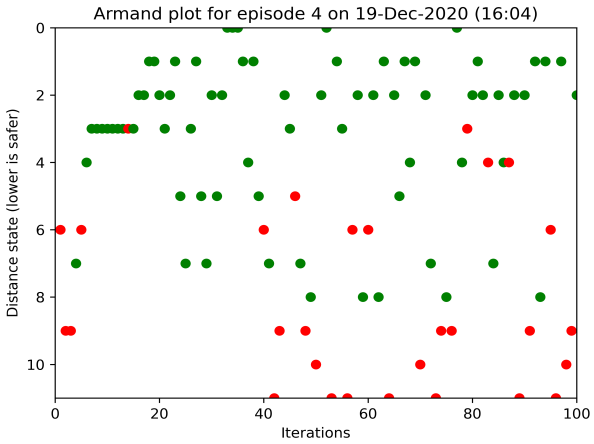
(a) Episode 1



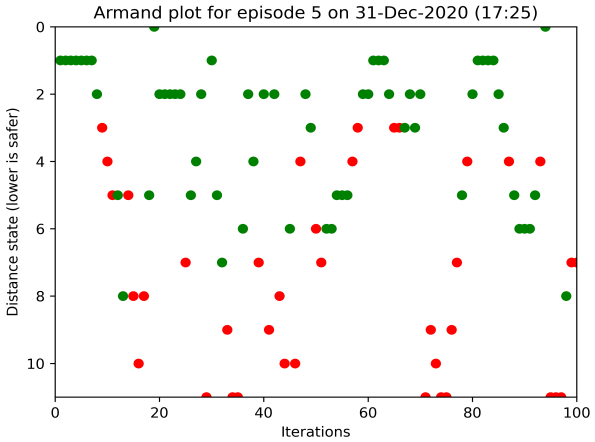
(b) Episode 2



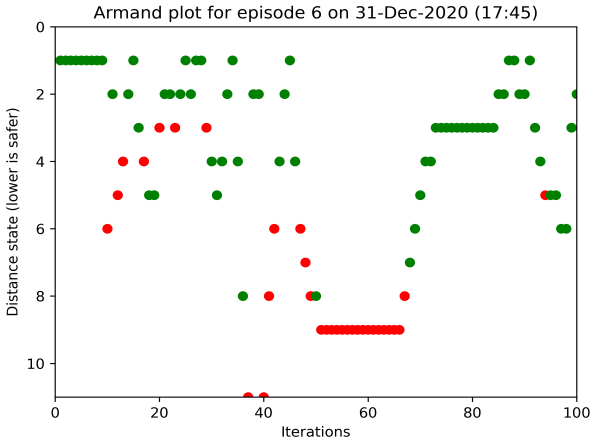
(c) Episode 3



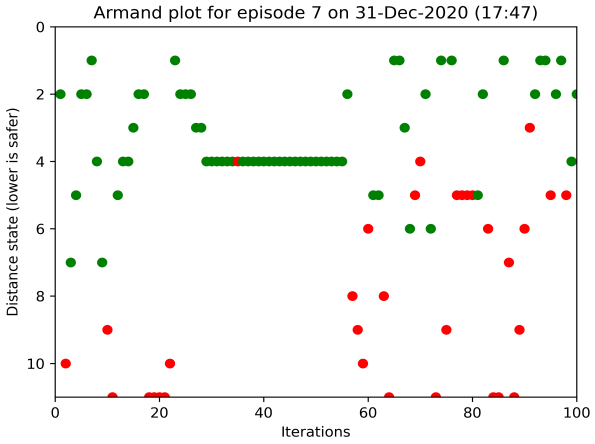
(d) Episode 4



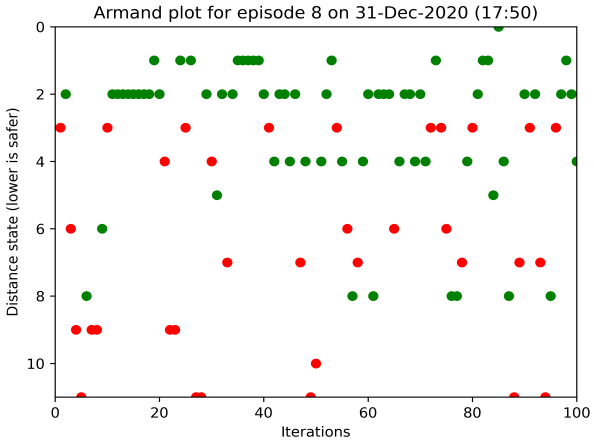
(c) Episode 5



(d) Episode 6



(c) Episode 7



(d) Episode 8

Table 2: Plots of actions taken over time using real human driver (Armand)

Table 2 displays the same plots applied to myself as a human driver using a manual-control driving environment. This experiment was run for eight episodes, demonstrating the progress of how warnings are issued over time. Since this is a manual-control environment (shown in section 3.3.9), as opposed to the scripted driving environment in table 1, the driving behaviors are not predictable, and the vehicle is driving on different stretches of road and is susceptible to different real-world driving errors. Based on these plots, the agent adapts to the behavior of the human driver: initially, it is not discriminating against issuing warnings in the safer states, but as time goes on, warnings tend to be issued in the intermediate and unsafe states, and the safer states tend to have no warning issued. In this experiment, when a driver enables their turn signals, the agent temporarily pauses learning until the driver disables their turn signals.

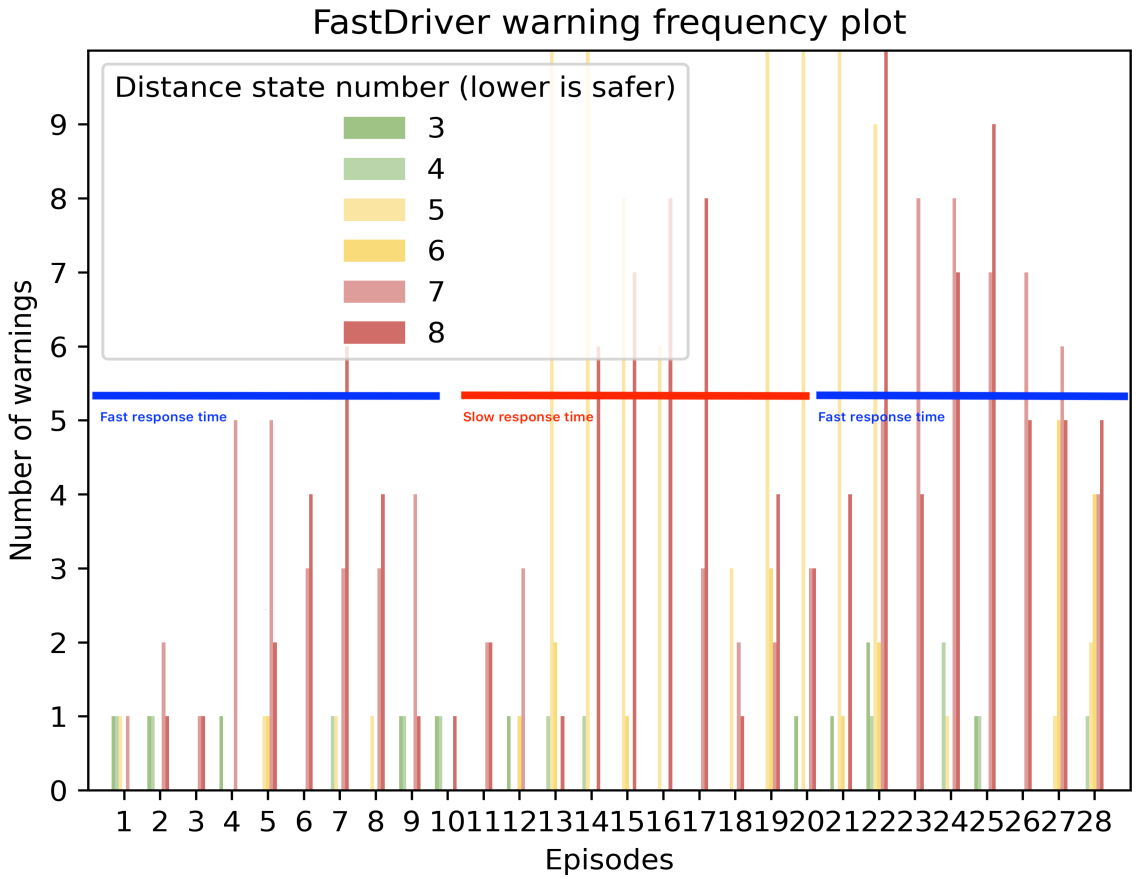


Figure 6: Sample plot of frequencies of warnings issued for fast response time driver

Figure 6 displays a sample warning frequency plot output of the agent (server) program for a driver with a fast response time. The x-axis is time, measured in number of episodes. Each episode generally includes about 100 iterations, taking about 3 s/iteration. Each bar on the x-axis represents a particular distance state (color-coded in the legend). The y-axis represents the number of warnings issued at the given distance state. This output runs 28

episodes starting with an empty Q-table. This experiment was designed to demonstrate the agent's learning progress over time. As time progressed, the agent started to issue warnings more frequently at more unsafe states. Around state 10, the client program dramatically slowed down it's response time for warnings (shown with red horizontal bar), and the agent responded with issuing warnings more frequently at intermediate states (yellow). This concluded around state 21, where the client returned to a faster response time (shown with blue horizontal bar), and the agent began to issue warnings less frequently at intermediate states.

3.3.9 Physical setup

Figure 7: Sample test bed of running environment

Figure 7 displays the test bed that has been used in developing this project. In this test bed, there is an iMac Pro desktop computer running the CARLA simulation environment (in Windows) and a modified CARLA manual control client program (in Windows). The Lane Departure Warning agent is running on the Raspberry Pi (Linux) pictured underneath the iMac Pro. The subject is also wearing the Oura Ring on their ring finger, and an iPhone is in the background running the Oura application that streams data from the Ring via Bluetooth.

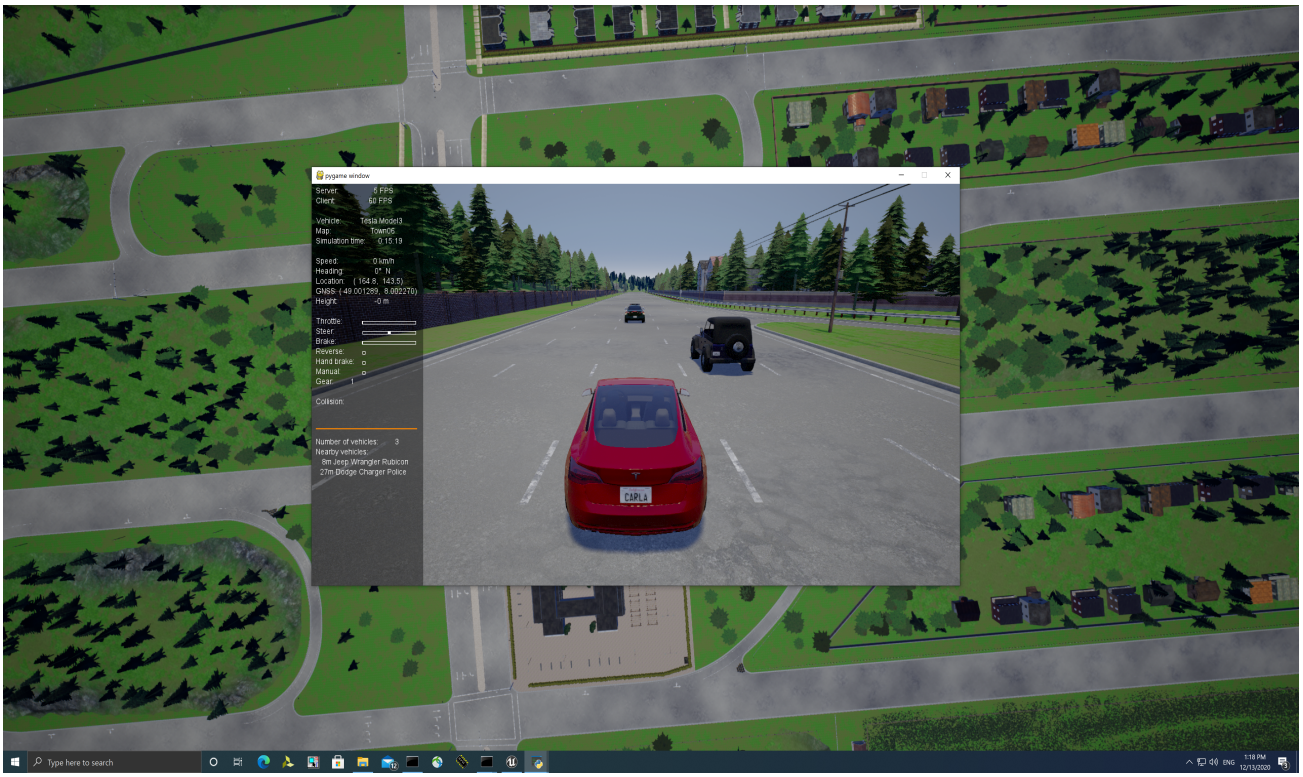


Figure 8: Screenshot of desktop components of test bed

Figure 8 displays the desktop view of the test bed that is mentioned above. In the center, there is the CARLA manual control environment, where a user can manually drive a vehicle throughout the town. This manual control environment has been modified to connect to the Lane Departure Warning server running the Q-learning agent. In the background, the CARLA Simulator view is shown for reference. It shows a top-down view of the town, with all the vehicles driving through it.

3.4 Future objectives

In the future, the Q-learning intelligent agent will be improved in order to accurately learn about the behaviors of various drivers. The algorithm for predicting a driver's human state based on data from the Oura Ring will be fine tuned to ensure accuracy. In addition, other hardware sensors may be introduced if the budget permits. Other ADAS sensors may also be created and integrated with the Q-learning agent, including forward collision warning and blind spot monitoring. Creating the privacy-aware aspect of this project will occur in later stages when the Q-learning agent is in a refined state.

4 Approach

This project will primarily involve developing algorithms and agents with the help of open-source software, such as CARLA Simulator. CARLA will be the primary simulation environ-

ment that will be interfaced with in this project, due to its versatility and variety of available driving environments. As mentioned in section 3.3, a branch of reinforcement learning called Q-learning will be studied and applied to create the intelligent agent that continuously learns about a human's driving behavior and physiological state in order to issue ADAS warnings at the correct time.

The research process will involve interacting with various mediums including hardware and simulation environments. Currently, the Oura Ring is the primary form of hardware being used to relay critical sensory data to the intelligent agent. Due to the ability to create comprehensive models of the human state with the help of this hardware, simulations, and no outside human subjects, this project is feasible given the remote learning situation during the 2020-2021 academic year.

This project will be conducted under the Campuswide Honors Collegium's undergraduate research guidance, and thus a comprehensive thesis will be written at its completion.

5 Responsibilities

This project is individual and not group-based. Armand Ahadi-Sarkani assumes full responsibility for all research work, under the guidance of Prof. Salma Elmalaki, Assistant Professor of Teaching in the Electrical Engineering and Computer Science department. Meetings about research and thesis planning occur approximately every week in a remote format. Credit for this project is being given through EECS 199 and ENGR H199 in the fall.

6 Timeline

Objective	**Estimated Completion**
Improving metrics for lane departure warning system	June 2020
Purchasing and collecting data from hardware sensors	June - August 2020
Interfacing hardware sensors with CARLA Simulator	July - August 2020
Integrating human state data and creating reinforcement learning agent for lane departure warning sensor	August - December 2020
Creating other ADAS sensors, implementing privacy awareness	Winter Quarter 2021
Complete Campuswide Honors Thesis	Winter Quarter 2021

7 Works Cited

Em, Poh Ping, et al. "Vision-Based Lane Departure Warning Framework." *Heliyon*, Elsevier, 6 Aug. 2019, www.ncbi.nlm.nih.gov/pmc/articles/PMC6698973/.

"Lane-Keeping-Assist-on-CARLA." *GitHub*, www.github.com/paulyehtw/Lane-Keeping-Assist-on-CARLA.

"CARLA Simulator." *GitHub, www.github.com/carla-simulator/carla*

Elmalaki, Salma, et al. "Sentio: Driver-in-the-Loop Forward Collision Warning Using Multisample Reinforcement Learning." *Sentio | Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 1 Nov. 2018, www.dl.acm.org/doi/10.1145/3274783.3274843.

"Oura API." *Oura Ring Developer, https://cloud.ouraring.com/docs/*

"SciPy." *SciPy, www.scipy.org*