ENPH353 Engineering Physics Project Lab I: Final Report

Development of an Autonomous Agent for Navigation and License Plate Recognition in a Gazebo Simulation

Rudransh Kumar & Grayson King

Winter Session 2021

THE UNIVERSITY OF BRITISH COLUMBIA

INTRODUCTION

In this course, we were tasked with developing an autonomous agent capable of driving through a simulated environment in Gazebo while obeying traffic laws to collect license plate numbers and plate identifiers.

As shown in Appendix Figure A1, the world consists of an inner driving loop and outer driving loop with 8 cars distributed along the side of the loops. There are also two cross-walks featuring pedestrians that cross periodically and a truck that continually circles the inner loop. The software architecture of the simulation and the agent was built on ROS. The only permissible sensory inputs of the agent is an image feed of its front-facing view, as shown in Figure A2. The chosen path of the agent was a counter-clockwise traversal of the outer loop from the starting position (as shown in Figure A1) followed by a clockwise traversal of the inner loop. In complete traversal of the path, all license plates are detected and alphanumeric information is extracted from the camera feed.

To meet all the performance objectives of the agent, its control algorithms were separated into 4 concerns:

1. *Driving.* The driving module is responsible for the physical navigation of the environment along the intended path in a manner that obeys all traffic laws and keeps the license plates within view of the robot's camera for recognition.
2. *License Plate Recognition.* The license plate recognition module is responsible for detecting the presence of a license plate within view of the robot and extracting the alphanumeric characters that comprise a license plate's identifier and plate number for reporting.
3. *Pedestrian Detection.* The pedestrian detection module is responsible for detecting a pedestrian within the trajectory of the robot and advising when it is safe for the agent to advance without collision.
4. *Truck Detection.* The truck detection module is responsible for detecting the presence of the truck within the trajectory of the robot and advising when it is safe for the agent to advance without collision.

Ultimately, these 4 concerns were implemented as individual ROS nodes. Integration of these nodes was achieved with a combination of ROS subscribers/publishers and ROS services/clients for intercommunication.

In the following sections, we will discuss our development process for each of the stated concerns of the agent and the subsequent process of integration, leading to a cohesive control model for the agent.

DRIVING

In this section, we will discuss the agent's approach to navigation. This includes driving on the inner and outer loops of the course, as well as subsequences to get the robot from the start position to the outer loop and from the outer loop to the inner loop.

IMITATION LEARNING: INNER AND OUTER LOOP TRAVERSAL

Driving on both the inner and outer loops was achieved with imitation learning. Two convolutional neural networks (one for each loop) were trained on manually collected driving data, with the goal of producing a model that could navigate in a similar manner to a human controlling the robot from the teleop keyboard. Further, a discretized approach to driving was opted for, where the command velocities sent to the robot corresponded to one of three states: turning right, turning left, or going straight. The navigation flow of control using the trained CNN is as follows:

1. Read in a picture from the robot's camera,
2. Perform image preprocessing, including downscaling and normalization,
3. Input the processed image to the CNN in array format,
4. Parse the prediction from the CNN and drive the robot at the corresponding command velocities.

To produce a dataset for driving model training, a data collection script (sandra_data.py) was made. This script includes functionality to scale-down images to a certain percentage of their original size, and export these images in a specific directory structure that could be unpacked and parsed. After a number of data collection sessions, these folders of data were zipped up and uploaded to a shared drive folder so they were accessible from the sandra-data-pipeline Colab notebook.

Two main principles led to the successful training of the driving neural networks: Train on lots of varied driving data, and make it easy to adjust the relative ratios of data types in the training and validation sets. With just a large, varied dataset, a fairly well-performing model could be produced, but the performance of these models was not tailored to the competition driving conditions. This resulted in errant turns causing the agent to lose its path. A large increase in model performance was seen when the exact number of each type of data in the training set could be easily adjusted (altering the ratios of right images to straight images to left images). In this way, models with approximately the same validation accuracy as those produced without selective data biasing saw much greater driving performance. An example of a confusion matrix produced from the training of the inner driving model can be seen in Appendix, Figure A3.

The selective data biasing made it so that the models were predicting more reliably on images that were more likely to show up when the agent traversed a particular path. For example, a model for

the outer loop trained on selectively biased data was much better at discerning straight sections from left turns than a model trained on a non-biased dataset. This was achieved by increasing the amount of left and straight driving data to account for the fact that the model would be more frequently predicting on pictures that matched these labels.

SUBSEQUENCES: TRANSITIONING BETWEEN INNER AND OUTER DRIVING SCHEMES

In order to get from the start position of the competition to a location where the outer NN model could be used to navigate, a hard-coded subsequence was used. Initially, rosbag was explored as a way to record manual traversal of this path, with the option to play it within the driving node using a python subprocess. After thorough testing of this approach, it was discovered that rosbag is heavily dependent on the simulation's real-time factor. This fact made it particularly hard to get consistent playback of the rosbag sequences, as the real-time factor would fluctuate throughout competition runs. Instead, a timed approach was opted for, making use of the rospy.get_time( ) method which was synchronized with the simulation's real-time factor. This meant that hard-coded sequences using times collected from calling this method would execute in the same way regardless of fluctuations in simulation speed.

A second hard-coded sequence was required to get the agent to the inner loop after a full traversal of the outer loop. This was done with a similar time-based scheme to the start subsequence, however, more intelligence was required to inform the controller when this subsequence should be executed. The first indicator used to localize the robot throughout its traversal of the outer loop was angular positioning. The total rotation that the robot had undergone over the course of a competition run was tabulated by integrating its angular velocity. Once this angle surpassed 2pi radians the controller could tell that the robot had successfully traversed the outer loop. Next, a marker needed to be identified, after which the robot would immediately begin running the subsequence. This came in the form of a topic published by the license plate detection module (referenced in the next section of this report) with a truth value indicating when plate #1 was no longer in view. This was an arbitrary starting location for the subsequence (the point when plate #1 goes out of view), but with thorough testing, it produced consistent results.

During the execution of this subsequence, the controller was also polling a topic published by the truck detection node, to determine if it was safe to enter the inner loop. Finally, after getting the go-ahead to enter the inner loop the controller switches control back to imitation learning by predicting with the inner driving model.

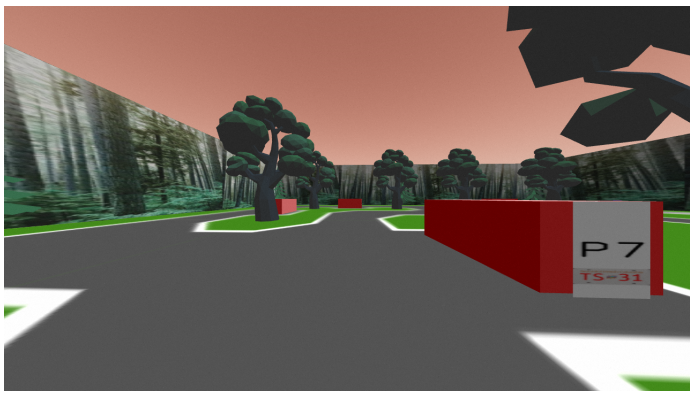
LICENSE PLATE RECOGNITION

In this section, we will discuss the agent's approach to license plate recognition and the development process that led to its effective implementation. During development, this concern was further divided into three sub-concerns: (a) license plate detection, (b) car identifier extraction & recognition, e.g., P7, and (c) alphanumeric character extraction & recognition of plate sequence, e.g., TS-31.

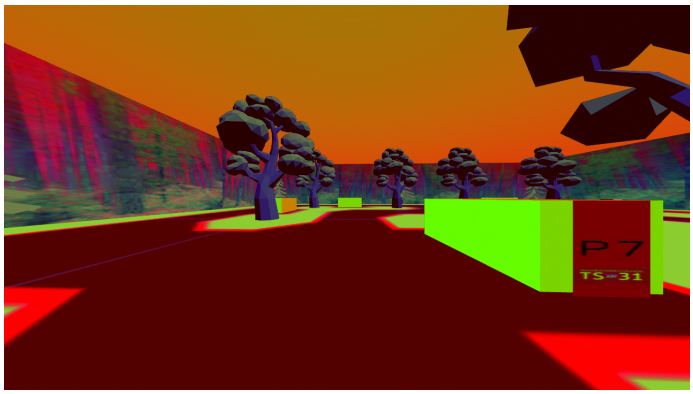
ALGORITHM: LICENSE PLATE DETECTION

To detect license plates, the agent employs the following algorithm:

1. Get the current frame from the published camera feed topic in the BGR colour space.



2. Convert the frame into the HSV colour space.



3. Compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the grey colour of the license plate background.

4. Compute the contours of the mask, and sort the contours by area, as the top and bottom segments of the plate should be the largest two contours of the set.
5. To detect the license plate, the first-largest contour is assumed to be the contour of the top segment of the license plate and is subjected to some empirically-determined criteria. If any criterion is not met, the frame is not analyzed any further. To qualify as the contour of the top segment of the license plate, the largest contour must

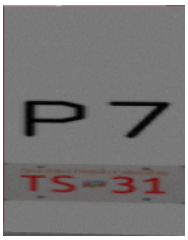
    - exceed an empirically-determined minimum contour area,
    - be of the shape of a quadrilateral, and
    - have an aspect ratio within an empirically-determined range.

6. If the first-largest contour qualifies as the top segment of a license plate, up to 5 of the next smallest contours are analyzed to find some contour that qualifies as the associated bottom segment of the license plate. If none of the next 5 contours qualify against all criteria, the frame is not analyzed any further. To qualify as the bottom segment of the license plate, a contour must

    - be of the shape of a quadrilateral, and
    - have an aspect ratio within an empirically-determined range.

7. If a pair of contours qualify as the upper and lower segments of a license plate, the top-most and left-most pixels of the upper segment and the bottom-most and right-most pixels of the lower segment are found to isolate the license plate from the frame. A negative buffer value is added during cropping to remove any car-coloured pixels from the outer edges of the cropped image.
8. Finally, the cropped image must pass some criteria to reject any ill-sized license plate crops, which may occur if a bottom-segment contour is erroneously selected. To qualify as a license plate, the cropped image must

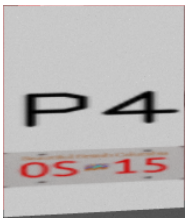
- exceed an empirically-determined minimum area (to reject plates that are too far away for accurate character recognition),
- have an aspect ratio within an empirically-determined range,
- have a height below an empirically-determined maximum height, and
- have a width below an empirically-determined maximum width.



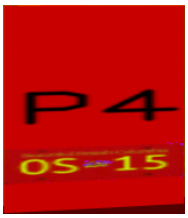
ALGORITHM: CAR IDENTIFIER EXTRACTION & RECOGNITION

If a license plate is successfully detected, the agent employs the following algorithm to extract the car identifier and interpret the identifier as a number from 1-8.

1. The isolated license plate is accepted post-detection in the BGR colour space.



2. Convert the license plate to the HSV colour space.



3. Compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the black colour of the car identifier.

4. Compute the contours of the mask and sort in non-increasing order by contour area. Then, sort the two largest contours by the x-coordinate of their centroids.
5. Compute the bounding rectangle of the contour whose centroid has a larger x-coordinate, and crop the mask with a positive buffer. Finally, resize the characters to the standard input size of the identifier NN for recognition.

ALGORITHM: ALPHANUMERIC CHARACTER EXTRACTION & RECOGNITION OF PLATE SEQUENCE

If a license plate is successfully detected, the agent employs the following algorithm to extract alphanumeric characters that comprise its plate sequence.

1. Using the same HSV-space image for car identifier extraction, compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the red colour of the plate sequence.

2. Compute the contours of the mask, and sort the contours by area. Filter out any contours with an area that is less than some empirically-determined minimum (to reject insignificant contours of other license plate/car features). If there are exactly 4 contours after this filtering, continue. If not, no further analysis is done.
3. Sort each of the 4 contours by x-coordinate of their centroids. The first two are alphabet characters and the last two are numerical characters.

4. Compute the bounding rectangle of each contour, and crop the mask 4 times to isolate each character. Finally, resize the characters to the standard input size of the NNs for recognition.

THREE NEURAL NETWORKS FOR IDENTIFIER AND ALPHANUMERIC CHARACTER RECOGNITION

Three different neural networks were used to determine the value of license plates and identifiers located by the agent. This approach was opted for in order to avoid similar-looking characters showing up in the same dataset (S and 5, for example, would be difficult for the NN to train on). The neural network architecture for the license plates remained exactly the same as the driving models, with the main differences coming in the quality and type of data being fed to the network. First, the images being predicted are binary, which drastically speeds up the training and validation processes and makes it much easier for the trained NN's to generalize. Second, the quality of data used for training was, by nature, much higher than that of the driving NNs. This is a direct result of the effort put in to properly extract clean, isolated binary images of the license plate and identifier characters. Additionally, the number of possible images the NN could be expected to predict was on orders of magnitude smaller than the driving NNs, which led to more immediate positive results. With some selective data biasing, particularly on Rs and Ps, the 3 license plate neural networks were able to predict characters with upwards of 98% accuracy.

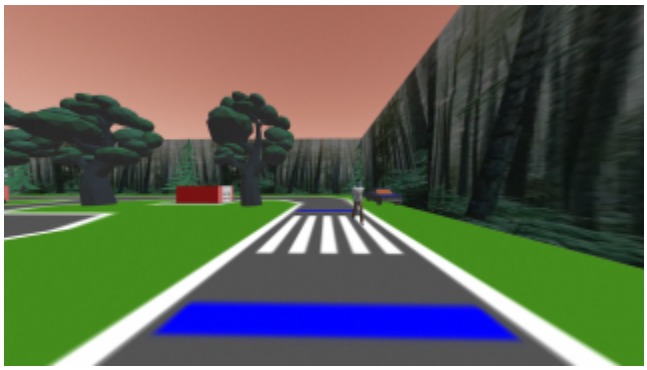
PEDESTRIAN DETECTION

In this section, we will discuss the agent's approach to pedestrian detection.

ALGORITHM: PEDESTRIAN DETECTION

To detect license plates, the agent employs the following algorithm:

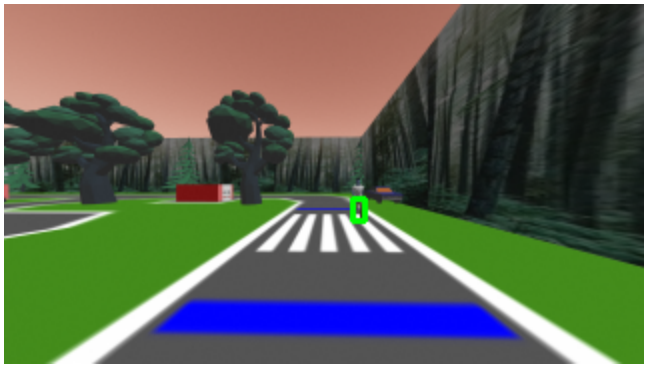
1. Get the current frame from the published camera feed topic in the BGR colour space.



2. To detect when the agent has approached the crosswalk, compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the blue colour of the cross-walk only within the bottommost 50 pixels of the frame.

3. Compute the contour area of the largest contour mask and if it surpasses some empirically-determined minimum, switch the agent into a state of LOOKING_FOR_PEDESTRIAN=1.
4. If the agent is LOOKING_FOR_PEDESTRIAN, continue.
5. Compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the colour of the pants of the pedestrian.

6. Compute the contours of the mask, and sort the contours by area. If there is at least one contour whose area exceeds some empirically-determined minimum area, continue.
7. Depending on the position of the pedestrian, the pant legs can either be detected as one contour or two separate. Since all other contours that are found in the mask are small, if there is more than one contour, compute the ratio of the area of the largest contour to the area of the second largest contour. If the ratio is smaller than some empirically determined maximum, then the second contour is that of another pant leg. Otherwise, it is not and the largest contour is that of both pant legs.



8. Considering either 1 or 2 contours for the pants, compute the left-most, right-most, and bottom-most pixels occupied by the pedestrian.
9. With the bottom-most pixel occupied by the pedestrian, crop the frame to produce a horizontal slice within +/- 5 pixels of the bottom-most pixel.

10. On this horizontal slice, compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the green colour of the grass. Invert the mask to find the regions other than the grass.

11. Compute the contours of the mask, and sort the contours by area. The largest contour will always correspond to the road ahead. Complute the left-most and right-most pixels of the road contour. If either the left-most pixel of the pedestrian is more to the right than the right-most pixel of the road or the right-most pixel of the pedestrian is more to the left than the left-most pixel of the read, the pedestrian is not on the road so set PEDESTRIAN_DETECTED=0. Otherwise set PEDESTRIAN_DETECTED = 1.

12. Since the pedestrian can start crossing at any point and potentially collide with the side of the robot, it is only safe to go when the pedestrian has moved from the road to the grass. Accordingly, set SAFE_TO_GO=1 when there is a falling edge on PEDESTRIAN_DETECTED.

13. If the mask for the blue colour of the cross-walk demarcations over the full size of the camera frame has no contours, set LOOKING_FOR_PEDESTRIAN=0, as the crosswalk has been safely traversed.

TRUCK DETECTION

In this section, we will discuss the agent's approach to truck detection.

ALGORITHM: TRUCK DETECTION

To detect license plates, the agent employs the following algorithm:

1. When '/looking_for_truck' publishes True, switch into a state of LOOKING_FOR_TRUCK=1.
2. If LOOKING_FOR_TRUCK, get the current frame from the published camera feed topic in the BGR colour space.
3. Compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the grey colour of the road.
4. For the contour with the largest area, find its top-most pixel, which corresponds to the interface where the road meets the grass.
5. Consider the bottom portion of the frame that contains the road and compute a mask for all pixels within an empirically-determined HSV colour range corresponding to the black colour of the undercarriage of the truck.
6. If the left-most pixel of the largest contour in the undercarriage mask is in the right 20% of the frame or if the mask has no contours, set TRUCK_DETECTED=0; otherwise, set TRUCK_DETECTED=1.
7. Since the truck can arrive within the frame at any point and potentially collide with the side or back of the robot, it is only safe to turn when the truck has moved from in-frame to out of frame. Accordingly, set SAFE_TO_TURN=1 when there is a falling edge on TRUCK_DETECTED.

INTEGRATION WITH ROS

To tie the various nodes mentioned above to the central driving node, various ROS topics and services were used. A breakdown of the topics and services can be seen below:

TOPICS

- **/is_liscence_plate_detected**: advertises a boolean value that can be used by the central driving node to determine when the second hard-coded subsequence should begin execution.
- **/is_safe_to_go:** advertises a boolean value that informs the driving node when it is safe to proceed past crosswalks.
- **/is_safe_to_turn:** advertises a boolean value that informs the driving node when it is safe to finish the second hardcoded subsequence and enter the inner loop.

- **/looking_for_truck:** advertises a boolean value that informs the truck detection node when it should be surveying the camera feed for the truck. Published by the driving node.

SERVICES

- **/send_all_licenses:** upon request, sends the highest frequency license plate predictions to /license_plate. This service is used by the driving node to flush all unsent license plates after a timeout.
- **/end_run:** upon request, sends the end-trial plate sequence to /license_plate. This service is used by the license plate recognition node to end the trial after all license plates are sent.

APPENDIX

LICENSE PLATE RECOGNITION SOURCE CODE

Colab sandboxes for algorithm development and NN training:

- License-plate-detection-sandbox.ipynb
- License-plate-identifier-extractor-sandbox.ipynb
- License-plate-number-extractor-sandbox.ipynb
- License-plate-recognition-neural-network-training-sandbox.ipynb

References to the repository:

- Sandra-license-plate-recognition.py
- Sandra-license-plate-data-collection.py

PEDESTRIAN DETECTION SOURCE CODE

Colab sandboxes for algorithm development:

- Pedestrian-detection-sandbox.ipynb

References to the repository:

- Sandra-pedestrian-detection.py

TRUCK DETECTION SOURCE CODE

Colab sandboxes for algorithm development:

- Truck-detection-sandbox.ipynb

References to the repository:

- Sandra-truck-detection.py

DRIVING SOURCE CODE

Colab notebook for driving NN training:

- sandra-data-pipeline.ipynb

References to the repository:

- sandra-driving.py
- sandra-data.py

FIGURES

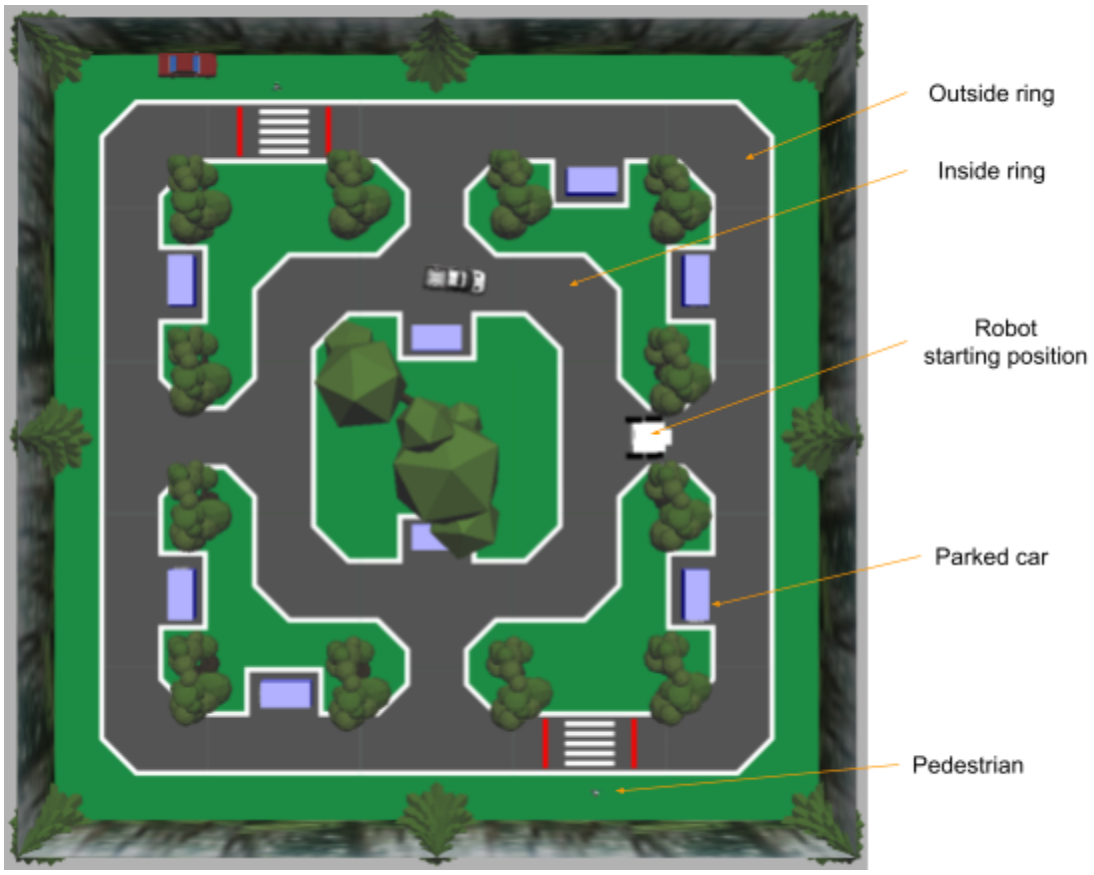


Figure A1: An overview of the simulated driving environment in Gazebo, adapted from the official 2021 ENPH353 Competition Notes.

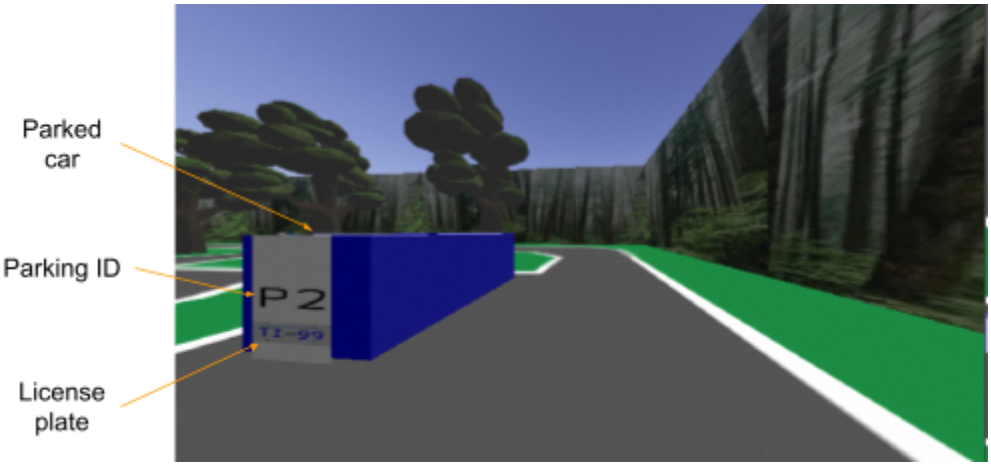


Figure A2: A sample image from the camera feed of the robot used for navigation and license plate recognition, adapted from the official 2021 ENPH353 Competition Notes.

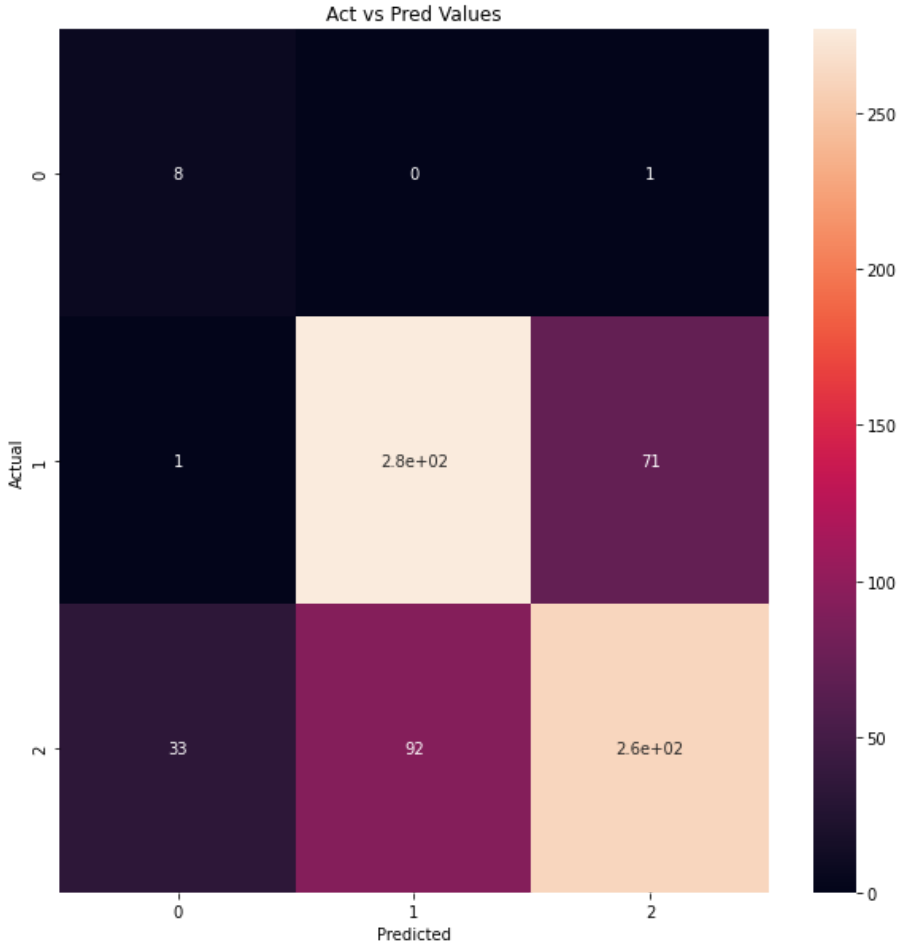


Figure A3: A sample confusion matrix produced during the training of the inner loop driving NN, where labels 0,1,2 correspond to left turn, right turn, and straight respectively.

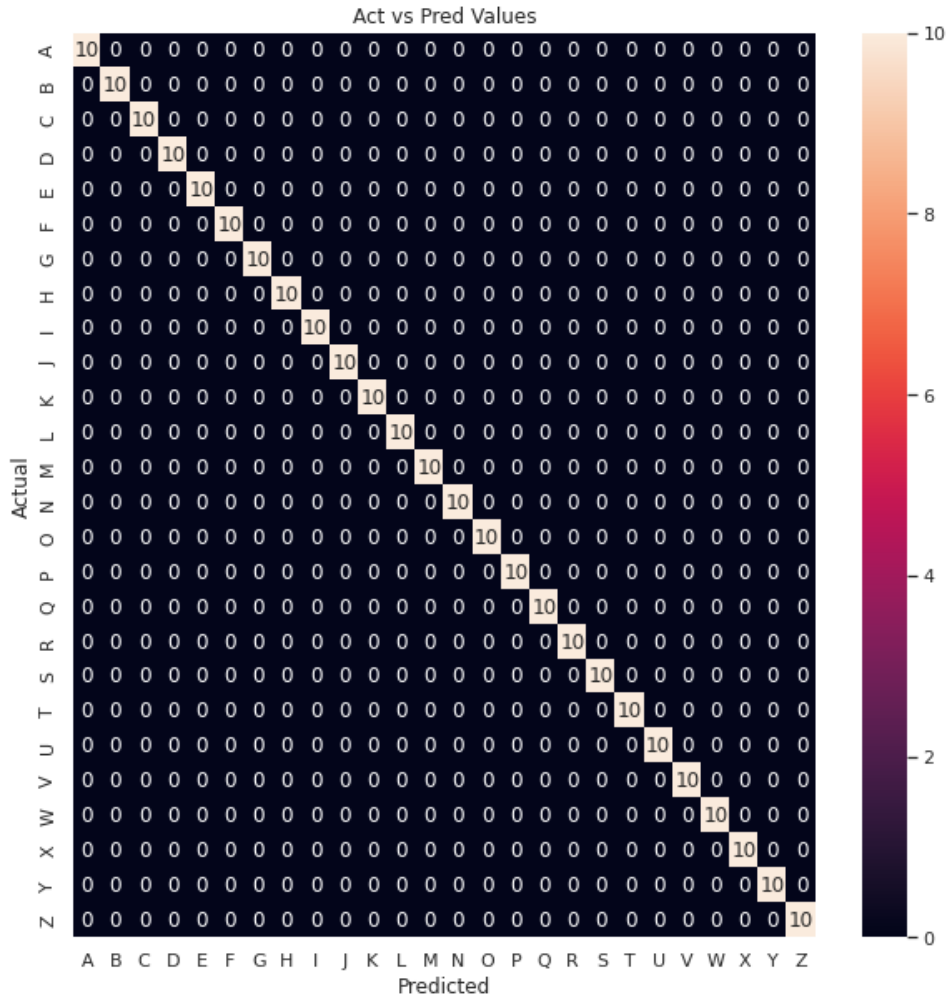


Figure A4: A sample confusion matrix produced during the training of the letter detection NN.

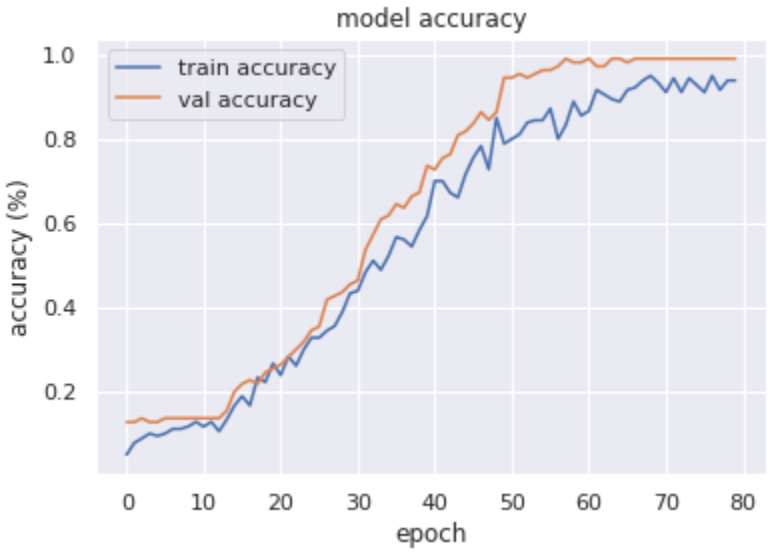


Figure A4: Accuracy plot from training of letters NN.

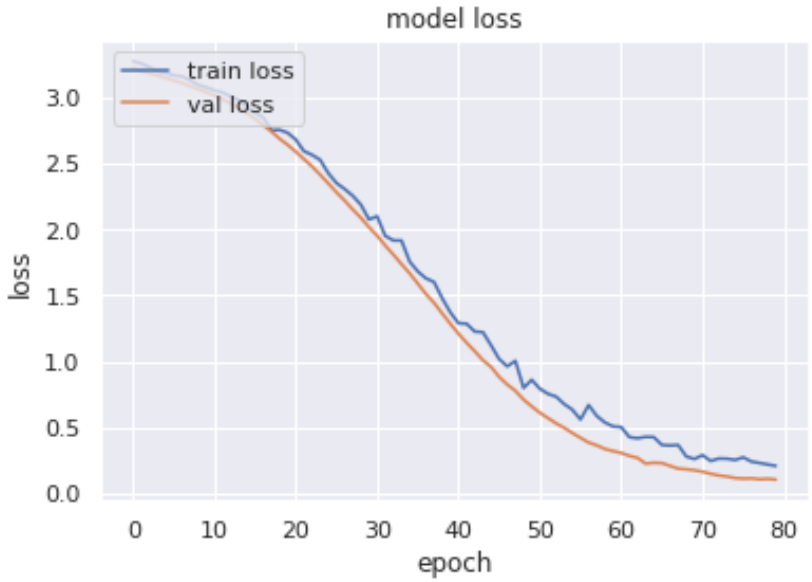


Figure A5: Loss plot from training of letters NN.

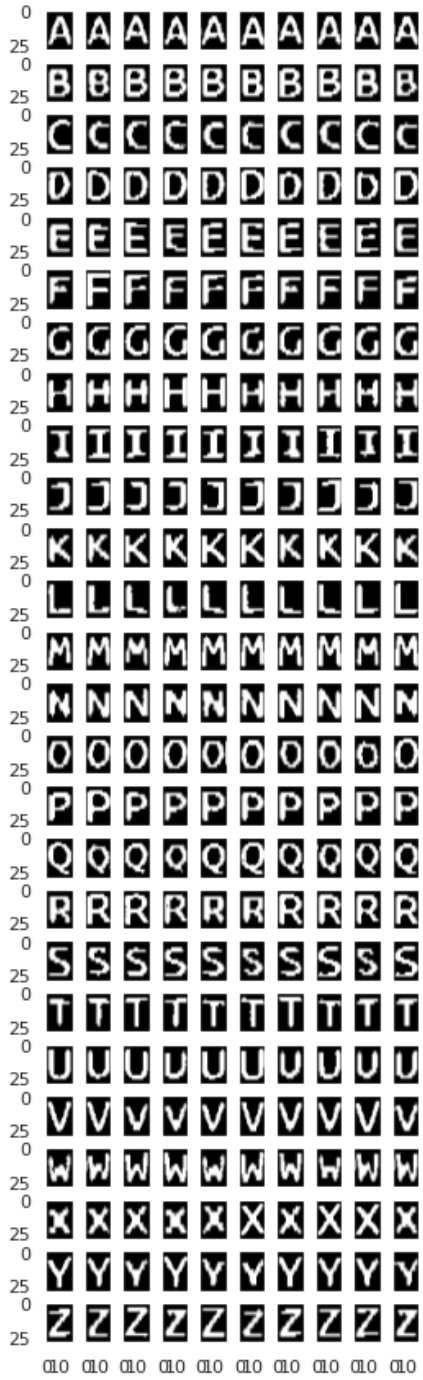


Figure A6: Sample training data visualization for letters dataset.

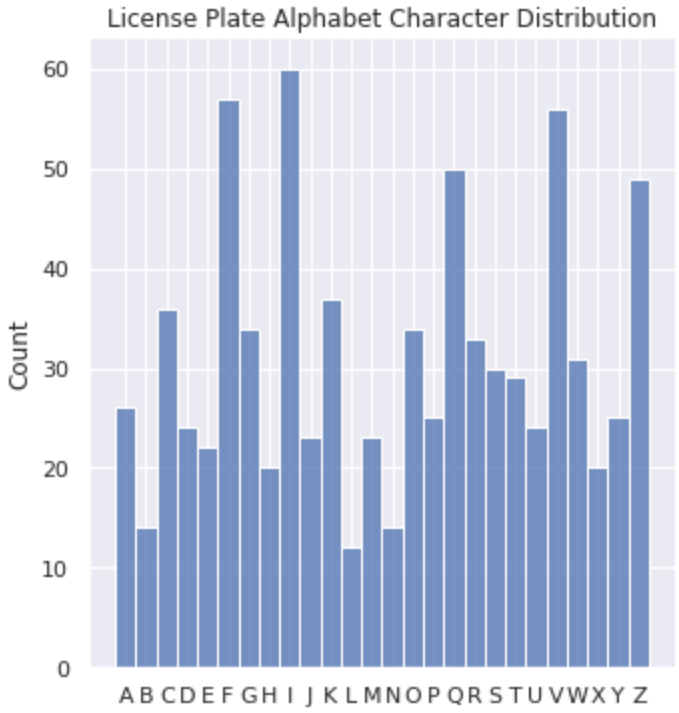


Figure A7: Sample histogram for letters dataset.