# ENPH 353: Control Agent for Driving and License Plate Recognition

Thomas Deckers, James Seto
University of British Columbia

***Abstract*—Classical and deep machine learning techniques were applied to develop an agent to control a robot in a simulated environment. The agent was evaluated in competition on its ability to obey traffic laws and recognize characters on license plates.**

## I. Introduction and Description of Task

Autonomous driving on public roads is becoming a reality in our world. Driverless cars must be able to recognize obstacles such as pedestrians and traffic to avoid potentially fatal collisions. Sensors such as cameras, IR sensors and accelerometers are used to receive information about the controlling agent's surroundings.

In this project, we explore the challenges of developing an autonomous driving agent in a timed competition. The agent is evaluated on its ability to obey traffic laws and recognize license plates of parked cars in a simulated Gazebo environment. Limited to a single front-facing camera, we apply both classical and machine learning techniques to use an image feed to navigate, avoid collisions and recognize license plates while keeping within the boundaries of the road.

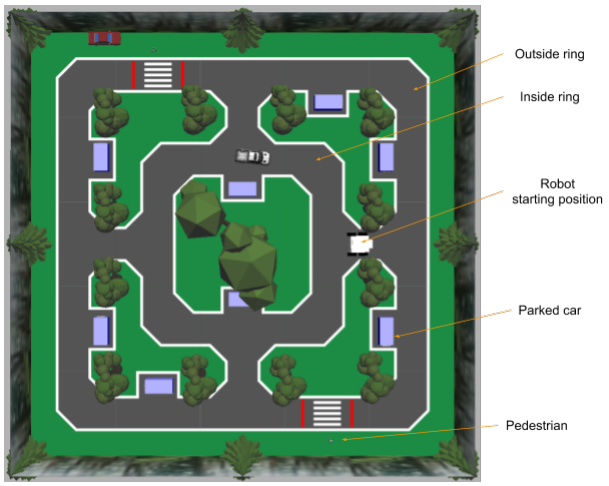


Fig. 1. Competition environment

## II. Architecture and Structure Overview

In our architecture design, we prioritized approaches that would allow us to apply the skills we learned in the course over traditional methods that might have been faster or more reliable. We also wanted a structure that would allow development of components in parallel, to allow for more efficient work in light of the available amount of time. For these reasons among others, our architecture had two systems running simultaneously, one which fully controlled the robot's steering, and one for identifying, reading, and reporting license plates.

This also very naturally divided our communication with the simulation; the driving controller has exclusive control over the robot's velocity, and the license-plate controller has exclusive control over the channel for submitting license plates. This eliminates any risk of problems arising from overlapping commands.

Communication between these two controllers is done via a dedicated channel. This allows the steering controller to know when to turn into the inside ring.

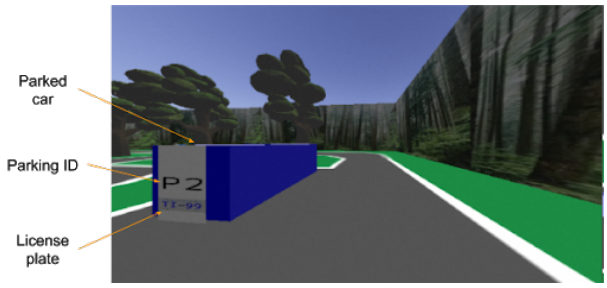


Fig. 2. Robot camera view

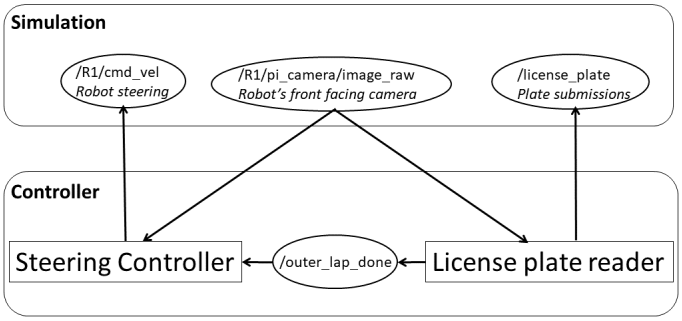


Fig. 3. Simplified illustration of control structure

Steering was controlled was by a convolutional neural network (CNN) trained on data from a human driver for driving around the rings, and was hard-coded to take the first turn onto the outer ring, as well as the turn into the inner

ring. Stopping for the pedestrian and the truck was also done by classical CV.
A classical CV pipeline identified the presence and location of license plates in the frame, and cropped the image into individual characters to be predicted by a CNN. The robot was programmed in the ROS Melodic framework, using rospy.

## III. License Plate Recognition and Submission

### A. Detailed description

The license plate recognition consists of an image pipeline and two CNN models for character recognition. The image pipeline filters the frames from the robot camera output and crops the license plate characters for the CNN to predict. The license plates were fixed in a format of "Letter Letter Number Number", allowing us to split letters and numbers based on the location of the crop. One CNN was trained to predict solely on numbers, and the other on letters. This agent is built in a stand-alone ROS node, and subscribes to the camera node to obtain the frames. The agent publishes to the score tracking node of the competition, as well as the robot controller node to communicate the status of recognized license plates.

This strategy was chosen for simplicity and because of the limitations in training data to work with. The image output from the robot camera is in majority not useful for character recognition; the image pipeline helped reduce the arbitrary data to train a CNN easily. This pipeline also enabled the ability to train the CNN on locally generated data as opposed to obtaining data manually from the competition space.

An attempt was made to apply homography to perform character recognition, but this technique performed poorly in initial tests. The low resolution and noise from the robot camera output caused many false matches of keypoints, and there was a lack of unique keypoints in the competition space to leverage.

### B. Image pipeline

The OpenCV library, a popular computer vision library for Python, was used to manipulate the frames from the robot's camera. First, the frames from the camera were converted into an HSV representation. A mask was used to filter for the blue colour of the parked cars. In order to remove noise and parked cars in the background, a morphological transform was applied.

The characteristic to detect a license plate on the screen was to find two peaks, summing across the columns of the mask. The detected license plate would be situated between the two peaks, with edges corresponding to the peak dropoffs.

An iteration across rows was used to determine the corners of parked car faces, and a normalized view of the license plate was obtained through a perspective transform using the obtained corners. Every character on the license plate were cropped into a uniform size to input into the CNN models.

Additional heuristics were used to filter images that may be too poor quality to be useful. A problem encountered with the CNN models was that predictions on heavily distorted images were falsely confident. To filter image distortion, a heuristic based on the perspective angle of the license plate was implemented. This heuristic was calculated by finding the relative length difference between the two vertical edges of the parked car. This, along with other measures such as confirming the presence of gray in the image, improved the reliability of the license plate recognition by reducing the likelihood of falsely confident predictions.

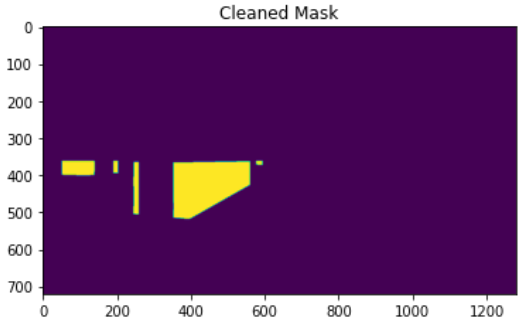


Fig. 4. A binary image of several parked cars after masking and morphological transform. The two tallest columns border the license plate of the car in the foreground.

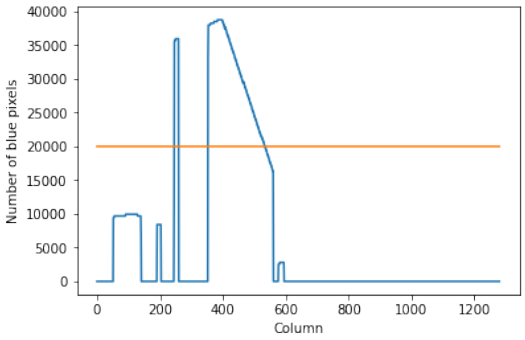


Fig. 5. A plot of the processed image, summed by column, and the threshold used to locate the license plate edges.


Fig. 6. Result of perspective transform

### C. Data generation and training

An image generating script was used to produce a dataset of license plates that imitated the data from the output of the image pipeline. In the competition environment, shadows produced varying brightness of the license plates, and a specific

font and font size was used. Artifacts produced by the robot camera also further reduced the data quality. To compensate for these factors, manual data augmentation was included in the license plate generating script. The augmentation applied a random amount of Gaussian blurring, Gaussian noise and brightness scaling to the generated images. Additional data augmentation, including shears, translations, and size scaling was used to account for the inaccuracies of the image pipeline in sharp perspective angles.

The CNNs used to recognize the license plates were designed to predict a single character. This enabled the training of a robust CNN model with minimal training data. The CNN structure was based off of a Keras tutorial to predict cats and dogs with minimal data, adapted to output a vector using a Softmax activation corresponding to the one-hot representation of a number or letter. The table below outlines the architecture of the CNN.

TABLE I
LICENSE PLATE PREDICTION NEURAL NETWORK ARCHITECTURE

| ***no.*** | ***Layer Type*** | ***Output shape*** |
|---|---|---|
| 1 | Conv 2D | (238,98,32) |
| 2 | Max Pooling 2D | (119,49,32) |
| 3 | Conv 2D | (117,47,32) |
| 4 | Max Pooling 2D | (58,23,32) |
| 5 | Conv 2D | (56,21,64) |
| 6 | Max Pooling 2D | (28,10,64) |
| 7 | Conv 2D | (26,8,64) |
| 8 | Max Pooling 2D | (13,4,64) |
| 9 | Flatten | 3328 |
| 10 | Dense | 64 |
| 11 | Dropout | 64 |
| 12 | Dense | 26 or 10[1] |

[1]Output is 26 for letters, 10 for numbers

The letter CNN was trained for 100 epochs at a learning rate of 5E-5 in order to pick up nuances between similar-appearing letters, while the number CNN was only trained for 50 epochs at a learning rate of 1E-4. The final dataset consisted of 400 license plate images cropped into 1600 characters. Therefore, a final dataset size was 800 per model trained. A training data batch size of 16 was used, and 20% of the data was reserved for validation.

### D. Testing and improvements

Plots of the accuracy and losses were generated for each CNN trained, one of which is displayed below. These plots helped determine overfitting or underfitting of the data. An anomaly is that accuracy of the validation data was consistently higher than the training data. An explanation could be that validation data was not augmented with shear, translation or scaling transforms.

To test the performance of the models, the prediction accuracy was recorded over batches of about 10 test runs of the competition. The specific character as well as the environment and robot conditions were noted during incorrect predictions. This provided insight that led to a range of adjustments, from the architecture chosen to the training dataset and hyperparameters. Early attempts at license plate readings were done by a model trained both on numbers and letters. The poor performance, especially between letters and numbers that appeared similar, led to the realization that the task of predicting numbers and letters could be split. Initial CNN models also performed poorly on license plates in shaded conditions. The font of the training data also initially did not match the font used in the competition environment, leading to the failure to predict certain characters. Randomizing the brightness and replicating the font in the training data significantly improved the performance of the models trained.

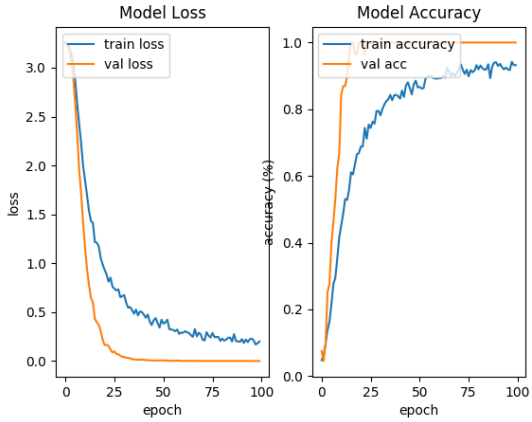


Fig. 7. The loss and accuracy per epoch of the final letter prediction model trained. The CNN was trained with a learning rate of 5E-5, with a dataset size of 800.

### E. Conclusion

Generating and augmenting a dataset locally was a successful strategy in training a model to recognize license plates in the competition space. Brightness variance and font choice in the generated data were the two significant factors that determined the quality of the model. A performance test with the final models yielded an accuracy of 99.7% with 1 missed character over 10 runs and 32 characters per run. Further improvements could be made in predictions at sharp angles. The selected strategy and CNN architecture proved to be adequate for this application.

## IV. DRIVING CONTROL

### A. Detailed description

The majority of steering was controlled by an imitation learning neural-network trained on human driving. The network would take in a downscaled image from the front-facing camera, and determine whether the robot needed to turn left, turn right, or go straight.

Left and right turns had the same linear velocity as when driving straight, with an added angular velocity component. This angular velocity was manually calibrated to allow the robot to take the 90° turns in one motion, while keeping all wheels on the road. Maintaining the same linear velocity

throughout allowed for smooth driving and a steady camera image.

The CNN's architecture was taken from the sample neural network shown in class to distinguish cats and dogs, and adjusted to output the required three commands. This architecture is summarized in the table below.

TABLE II
DRIVING NEURAL NETWORK ARCHITECTURE

| *no.* | *Layer Type* | *Output shape* |
|---|---|---|
| 1 | Conv 2D | (142,254,32) |
| 2 | Max Pooling 2D | (71,127,32) |
| 3 | Conv 2D | (69,125,64) |
| 4 | Max Pooling 2D | (34,62,64) |
| 5 | Conv 2D | (32,60,128) |
| 6 | Max Pooling2D | (16,30,128) |
| 7 | Conv 2D | (14,28,128) |
| 8 | MaxPooling 2D | (7,14,128) |
| 9 | Flatten | 12544 |
| 10 | Dropout | 12544 |
| 11 | Dense | 512 |
| 12 | Dense | 3 |

All layers used ReLU activation, except the last, which used softmax.

The above model was trained on Keras for 6 epochs at a learning rate of 1E-4 with a batch size of 16.

This is a much more complicated model than necessary for this application, given the simplicity of the problem. However, it proved adequate, and its speed was not a limitation on performance.

The robot used separate neural networks for steering in the inner and the outer laps, due to the fact that the car needs to drive in opposite directions in each ring, and due to the presence of the truck in the inner ring. This approach allowed for much faster and easier troubleshooting, due to the ability to make changes to one dataset without worrying about impacts on the other. As well, when the robot is driving counterclockwise, the right turn is only ever needed for course-correction, and vice versa with the left turn when driving clockwise. This simplification ultimately reduces the size of the problem-space.

### *B. Manual control*

The robot was controlled manually when taking the first turn into the outer ring, when switching from the outer ring to the inner, and when stopping for pedestrians.

The turns were done by overriding the neural network's control, and then publishing the highest velocity that didn't cause the car to tip. Code execution would then be paused for the manually calibrated time increment required.

To prevent the car from crashing into pedestrians, we used the red lines delineating the edges of the crosswalk. The controller detects when the sum of red values in the view of the robot's camera is above a preset threshold, then brings the robot to a complete stop. Movement in the frame is then detected by image subtraction. The car remains stopped until no movement is detected.

To clean up the image subtraction data, images from 0.1s apart are cropped and have a Gaussian blur applied before subtraction, and a morphological transform applied afterwards, to reduce noise. Finally, all pixels below a specified threshold are set to 0. With appropriate calibration, the resulting image is entirely dark unless the pedestrian was moving.

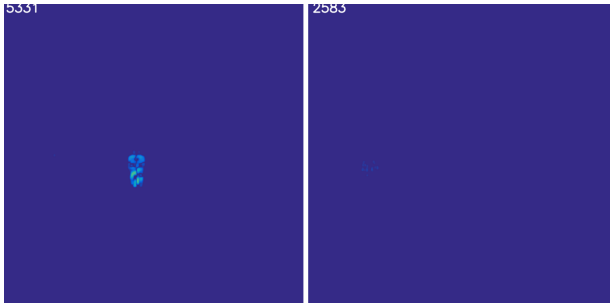


Fig. 8. The results of the cleaned up image subtraction when the pedestrian is in the middle of the road (left), and about to start crossing (right). The sum of pixel values is in the top left of each image.

The pedestrian makes a 90° turn before beginning to cross. Since pedestrian detection stops as soon as the car starts moving, it is important to ensure that the algorithm is tuned to be sensitive enough to detect this turn, otherwise there is a risk of starting to drive just before the pedestrian walks onto the road.

To mitigate the risk of the pedestrian walking into the car while the car is on the crosswalk, the speed of the car is increased while either of the red lines is still in sight. The higher speed causes some slight instabilities at corners, but we saw no issues on the straight crosswalk.

A similar strategy is used to avoid collisions with the truck driving around the inner ring. After all license plates are recognized in the outer loop, the neural network control is overridden and a turning routine lines the robot up at the intersection. Image subtraction is used to determine whether the truck is approaching or passing the robot. If the truck is not approaching, or the truck is no longer detected passing, the controller transitions into the inner loop and initiates the inner loop model for navigation.

### *C. Data generation and training*

In order to collect data from human driving, we used the tele-op twist program to drive the car, and saved a labelled frame from the camera whenever a command was published to the robot's velocity controller. The result of this is that each key-press captures an image. While training the robot, the key for movement was pressed repeatedly, a few times per second. This had the advantage of gaining fine-grain control over what data was captured. In particular, data was captured at a lower frequency on straightaways than in turns, to balance the size of the datasets, and allow for faster training.

Several laps were driven as cleanly as possible with this method. A few tactics were employed to improve the quality of this data:

- A slow real time factor was used in Gazebo to allow for more precise movement.
- The robot model was selected with the movement tool in Gazebo, which makes large arrows appear overlaid on the robot. This makes it much easier to align the car with the road.
- If there was a large issue with a turn, the images were manually deleted, the car was moved back to the start of the turn by clicking and dragging it in Gazebo, and the turn was taken again.
- If a turn wasn't taken at exactly 90°, corrections were made quickly to prevent wiggling on the straightaway in the trained model.

Course-correction data was then collected by modifying the data-collection script to only capture images when turns of the appropriate direction were sent to the robot, then driving intentionally poor laps with frequent course-correction.

One quirk of the above method of capturing data is that an image is never captured at the very end of turns, since the last press of the turn button will always be before pressing the straight button. This was similarly fixed for the outer ring by only collecting left-turn data, and intentionally overturning slightly.

After this point, the neural network was improved incrementally by running the robot and identifying situations where there were problems with the movement. Once a problem was identified, more data was collected of correctly handling the problematic situation.

The final dataset sizes can be seen in table III. 20% of the data was reserved for validation for each network.

TABLE III
DRIVING NEURAL NETWORK DATASET SIZES

| *Network* | *Command* | *Number of images* |
|---|---|---|
| Outer Ring | Straight | 2731 |
| | Left | 1035 |
| | Right | 282 |
| Inner Ring | Straight | 632 |
| | Left | 85 |
| | Right | 458 |

### *D. Testing and improvements*

Settling on the above method of control took several iterations.

We initially planned to collect data by capturing images from the robot being driven by an algorithm, as this would ideally create cleaner data.

However, using the published joint_states topic from Gazebo proved to have too much latency to allow for precise movement. While it certainly would have been possible to slow down the real time factor and optimize the driving code, it would have been a substantial time investment.

The first attempt at a human-trained model only had functionality for going straight and turning left. Left turns had no linear velocity, so the robot would pivot in place. If the robot took perfect 90° left turns every time, it would never need to adjust by turning right. However, due to latency and other imperfections in the system, this quickly proved far too idealistic, and resulted in the car frequently driving off the left side of the road.

Adding right turns to this model came with the side effect that the robot would pivot left and right repeatedly instead of taking corners. This is likely because the robot would overturn, then over-correct indefinitely. The problem is made worse by imperfections and overlaps in the data-set.

This issue was initially fixed by adding a small linear component to the right-turn command. At this point, the robot could drive a full lap while keeping two wheels on the road most of the time. However, keeping the same linear velocity through turns, as in the final result, made for much smoother, natural, and consistent driving.

## V. RESULTS AND CONCLUSION

The license plate recognition agent had a high reliability in test rounds, with only 1 bad character prediction in 10 rounds. Optimizations could be made to improve predictions at sharp viewing angles, but this was not critical to the performance in the competition space.

In the competition, the control agent scored the maximum points in license plate recognition. No collisions were made with pedestrians, traffic or other obstacles, and the agent maintained its position within the boundaries of the road. The round was completed in a time of 109 seconds out of the allotted time of 240 seconds. The quickness of the round enabled us to prevail over other maximum point scoring teams, placing us at third place out of twenty.

One avenue for further improvements to this agent is to generalize its performance to a broader variety of conditions. The current agent is optimized on the configurations given for the competition, but could be adapted to a variety of road layouts or license plate types through additional training and modifications.

One interesting quirk of the driving algorithm that arose throughout the development process was that it could drive many consecutive laps with no issues, but was rather sensitive to small disturbances. This is particularly interesting because the laps it drove were not identical each time, and it would follow a slightly different path after each lap. However, nudging the car slightly could cause it to miss the next turn entirely, or start turning right in the middle of a left turn. This issue was largely addressed in the final model, but could be interesting for further study, especially for generalizing the agent

There is also lots of further room for improvement of the speed of the vehicle. The driving speed was selected to be the fastest possible speed that wouldn't cause the car to tip when stopping. This speed limit could be surpassed by implementing gradual acceleration and deceleration.

The next likely barrier to speed is the prediction speed of the driving neural network. As mentioned, the architecture is more complex than necessary for this application. Camera images could also be further down-scaled to save processing

time. Given the very simple output space, the calculation time may be able to be approach the speed of classical PID implementation, with enough optimization.

Another avenue for speed improvement is in pedestrian detection. Rather than coming to a stop at every crosswalk to check for movement, it may be possible to identify the location of the pedestrian based on a distinctive color. This way, the vehicle can avoid stopping if the pedestrian is not on the road.

Overall, this project serves as a clear example of the potential of CNN's and machine learning when applied to simple control problems. Although classical CV approaches could have been successful in this challenge, with the suite of modern development tools available, the machine learning approach proved to be not only feasible with a short development time, but also yielded reliable and natural results.