ENPH 353 UBC Parking Competition Report

Objectives

Our competition took place in a simulated world running in Gazebo. The world contains a road configuration (Appendix Fig. 1), in which we were tasked with designing an autonomous robot to safely navigate through the roads in the world and accurately report the license plates of the parked cars it drives by. Points are awarded for each correctly-reported license plate, as well as for completing a full lap in the outside ring. The robot also shares the world with two pedestrians and a constantly driving car in the inner ring, and must follow traffic regulations to avoid collisions. Therefore, points are deducted for any collisions with the pedestrians or car, or for failing to stay on the road. During the competition, each robot was given four minutes to report as many license plates as possible.

To achieve these tasks, we were provided access to the front camera feed of the robot (Appendix Fig. 2) and the ability to publish velocity commands. With the video frames from the camera feed, we were challenged to creatively develop control algorithms and implement various image processing and machine learning techniques to achieve the required functionality.

Brainstorming

Timeline:

1. Week 1 (Nov 1 - 7)
    a. Read paper on imitation learning
    b. Complete path following
        i. Test different methods
    c. Start testing plate detection and perspective transform
2. Week 2 (Nov 8 - Time Trials)
    a. Complete plate recognition
    b. Start pedestrian detection
        i. Stop at the red line
        ii. Test computer vision methods (e.g. background subtraction / thresholding)
3. Week 3 (Time Trials - Competition Day)
    a. Complete pedestrian recognition and avoiding
    b. Vehicle avoiding
    c. Fine tuning

Main Components:

- Road Following
    - Remain strictly on road at all times
    - Possible methods:
        - Imitation learning (Read paper)
        - Hard-coded PID (Lab 3)
        - Reinforcement Learning (Lab 6)
- Pedestrian Detection and Tracking
    - Do not touch pedestrian
    - Possible methods:
        - Background Subtraction
        - Hue Thresholding
        - CNN with 2 labels
            - Bounding box (4 int values)
            - Pedestrian direction (Binary)
    - Consider direction that pedestrian is facing
    - Stop before red line and align properly
- License Plate Recognition
    - Use SIFT to form bounding box of license plate (Use blue side borders)
    - Use perspective transform to convert to nice rectangular image
    - Feed transformed image into trained CNN for parsing of position number and plate number
    - NOTE: Use blank license plate template for SIFT reference image
- Vehicle Detection
    - Use similar methods to pedestrian detection

Modules

Navigation / Path Following (Sam)

After reading “End to End Learning for Self-Driving Cars”, we were quite interested in implementing an imitation learning based approach to navigation. After some thought, however, we chose to stick to a PID approach because we had more experience with PID through our labs. This would also allow us to have more time for designing our neural network, as well as other tasks.

Trial 1: Road following (Road Following Video)

For all of our PID controls, we implemented only P (proportional) control, since we found that in our labs, it was sufficient for the robot to be able to path-follow smoothly. Our first approach was to binarize the video feed through HSV thresholding (using hsv_thresholding.py), filtering out everything besides the road itself. Our robot would follow the road by checking the x-coordinate of the centroid of the road each frame and align it to the middle of the screen by publishing

command velocities. We successfully found a set of parameters which produced a binarization as shown in Appendix Fig. 3.

There were several key issues with this implementation. Firstly, when approaching parked cars, the centroid of the road was affected by the pieces of road under the parked cars, causing the robot to veer towards the parked cars. Secondly, when approaching intersections, the robot was confused by which path it should take, and often drove in between the two paths (see video).

Trial 2: Line following (Line Following Video)

To solve the issues present in Trial 1, we decided to use the white lines on the side of the road for path-following. We also decided to first tackle the outside ring and worry about the inside ring later on, if we had time. This allowed us to use the outer white line for line-following, since it is straight throughout all the intersections and wouldn't be affected by the parked cars. However, since we wanted the car to stay on the centre of the road, we had to design our PID such that the outer white line always stayed at the right of the screen (driving CCW). We successfully obtained the HSV parameters, which produced a binarization similar to that shown in Appendix Fig. 4.

White Line Filter			
	H	S	V
Upper	50	150	255
Lower	0	0	245

Another issue we encountered was that the lines were not vertically straight (due to the perspective of the car, the lines slant towards the centre), so there was not a specific x-value to align the centroid of the outer white line to (if the centroid was higher up, it should be aligned to an x-value closer to the centre of the frame, but if the centroid was lower on the white line, it should align more to the right of the screen). In order to adjust for this perspective skew, we decided to use a "perfect" frame, in which the robot's view is perfectly centered on the road. Then, by finding the equation of the outer white line, we could map any y-value of the white line's centroid to its desired x-value on the line. Appendix Fig. 5 shows an example of a "perfect" frame.

Doing some simple measurements and calculations on the "perfect" frame with a ruler, pen, and paper, and knowing the dimensions of the video feed, we were able to determine the equation of the line.

$$x = int(my + b)$$
$$m = 8.7/6$$
$$b = 2.4 * (frame\ width) / 24.6$$

As the robot drives, for each video frame from the camera feed, the image would be masked to only contain the lower right portion (so the outer white line is the only white line showing and the inner white line excluded). Then, by finding the coordinates of the line's centroid, we could determine the desired x-value using our equation. Command velocities were then sent based on how the centroid's x-coordinate should be adjusted compared to the desired x-coordinate.

After this, the only issue remaining was that since the crosswalk is also white, it would interfere with the centroid of the outer white line. Since each crosswalk is marked by red lines, we decided that whenever the red line was detected (discussed later), the robot would simply drive straight. This turned out to be a simple and effective solution because by the time the robot arrived at the crosswalk, it was already properly aligned. (see video for finalized PID)

Pedestrian Detection and Avoidance (Ralph)

Problem to Address

In addition to the large penalty that we would have accrued for colliding with pedestrians (-10), we wanted to avoid any possibility of our run being ended due to the physics of the simulation. Avoiding these risks required a thorough assessment of the status of the pedestrian crossing the road.

PID Optimization for Consistent Approach (Sam)

To properly identify when it was appropriate for us to continue past the crosswalk, we needed to be able to get a clear view of the entire crosswalk and the pedestrian. Due to the limited FOV of the robot, we needed to be able to approach the crosswalk at a perpendicular angle to get a full view of the pedestrian at all times. To do so, we optimized the PID control, diminishing oscillations, allowing us to consistently drive to the crosswalk with minimal deviation while maintaining a complete view of the road and the pedestrian. Appendix Fig. 5 shows the robot's perspective as it approaches the crosswalk; it is completely perpendicular to the red line and has a full view of the stop line, zebra lines, and pedestrian. This consistency made it easier for us to assess the status of the pedestrian.

HSV Thresholding to Find Red Stop Line (red_line_detector.py)

Using the hsv_threshold.py file provided in the competition notes and some sample images of the stop line in the simulation world, such as Appendix Fig. 5, we identified potential ranges of HSV values to mask the images received from the robot's camera so that we would only see the red stop lines. Using our own script, threshold_test.py, we were able to concurrently output the thresholded images, as well as the raw images, seen by the robot camera to verify our HSV range. We finally arrived at the following set of upper and lower HSV values:

Red Line Filter

	H	S	V
Upper	10	255	255
Lower	0	200	0

Having filtered our camera images, we determined that there was a red line right in front of the robot if the number of non-zero pixels in the masked image was larger than our chosen threshold of 35000 pixels. This threshold allowed us to stop right before the line since the number of non-zero pixels increased as the red line got closer. We had initially used a threshold of 50000, but lowered it to account for the lag in communication between the nodes and the non-instant deceleration of the robot. See Appendix Fig. 10 for the result of masking Appendix Fig. 5.

Background Subtraction to Detect Pedestrian (pedestrian_detector.py)

Lastly, we needed to identify the pedestrian's status to determine when it would be ideal to continue past the crosswalk. We solved this by implementing background subtraction on frames from the robot camera. Every 100 ms, the previous image is subtracted from the latest image provided by the robot camera and converted into grayscale for simplicity. If the number of non-zero pixels in the grayscale image fell below our threshold value (100 pixels), we determined that the pedestrian was not moving and, thus, must be waiting on the side of the road, signalling the ideal time to continue forward. By using this information only when the car was stationary, we were able to filter out any pixels that were unchanged, resulting in a blur if the pedestrian had changed positions. See Appendix Fig. 11 for the result of background subtraction

License Plate Processing (Ralph) (license_plate_detector.py)

Problem to Address

As our primary objective and main source of points, accurate reporting of the license plates scattered throughout the arena required thorough identification and processing. By processing the license plates and position IDs of the cars, we are able to provide ideal data to the neural network for a confident prediction.

HSV Thresholding to Isolate License Plate

After attempting to use HSV thresholding to directly locate the license plates, we realized that we would need to filter out the white road lines that shared similar HSV values with the plates. We tackled this by using a blue HSV filter on the camera feed to focus on the blue cars holding the license plates. Using a bounding rectangle on all non-zero pixels (blue), we cropped the original image and applied a white HSV filter to the smaller image to find the area containing the license plate. Again, a bounding rectangle was created to enclose the white area.

To further improve these filters, we used gaussian blurring on the original image to remove noise and closed holes and filtered out small white islands using dilation and erosion, as defined by the *morph()* method in the file. The final HSV values we used for blue and white masking, as chosen similar to the red line masking, are as follows:

Blue Car Filter				License Plate (White) Filter			
	H	S	V		H	S	V
Upper	125	255	210	Upper	0	0	205
Lower	115	125	100	Lower	0	0	90

See Appendix Fig. 13 for HSV filtering of a blue car and license plate.

Sample Selection Based on Aspect Ratio

After enclosing the white pixels in the cropped image with a bounding rectangle, we then screen the image depending on the bounding rectangle's aspect ratio. After testing with various samples, we decided that the acceptable aspect ratio range (in decimal form) should be 0.67 - 0.82. Anything less (narrow) usually had a column of the license plate chopped off, while anything greater (wide) usually had the position ID or the license plate missing. Similar to before, the vertices of the bounding rectangle are used to crop the original image. The cropped out license plate and position ID are then saved in a global variable. After the first acceptable plate is collected, license plate images are collected, after passing aspect ratio screening, for 2 more seconds. Out of these screened samples, the sample with the largest aspect ratio is deemed the best and is pushed for further processing. See Appendix Fig. 12 for various license plates throughout the selection process

Segmenting and Perspective Warping of License Plate and Position ID

The best sample of each license plate in the arena is then HSV thresholded, inverted in binary, and morphed (dilation and erosion) to highlight the position ID and license plate.

License Plate Filter			
	H	S	V
Upper	0	0	255
Lower	0	0	0

Contours are detected on the masked image and a bounding rectangle with minimum area (cv.minAreaRect) is used to enclose the contours that contain the greatest area (the license plate). The vertices of the angled bounding rectangle are then used to crop and execute a

perspective transform on the best sample (not masked), resulting in a rotation of the license plate so that it is easier to segment for prediction. See Appendix Fig. 12 (Right) for a rotated plate. The processed license plate can be seen in Appendix Fig. 6. The position ID is cropped using the top vertices of the license plate bounding rectangle and saved.

Preparing Data for NN Prediction

The license plate and position ID are then processed as follows:

- Cropped based on predetermined values, chosen after thorough testing, to center the letters / numbers in 5 separate images
- Resized the letters / numbers so that they are 40 pixels high and 42 pixels wide
- Converted into grayscale
- Thresholded using adaptive gaussian thresholding to account for differing brightnesses
- Normalized so that image arrays contain values between 0 - 1 inclusive

After processing, the letters and characters are passed through the neural network for prediction and the most reasonable prediction is saved (eg. a letter cannot be predicted as a number). See Appendix Fig. 6 and 7 for an example of a license plate before and after processing.

Neural Network Training and Design (Sam) (CompetitionCNN.ipynb)

We used Google Colab. to design, implement, and test our neural network (NN), as was done in Lab 5.

Generating License Plates (final_project_license_generator.ipynb)

Since one of the biggest factors associated with our NN's ability to accurately predict license plates is the amount of training data provided, we decided to start with generated license plates as it is a quick and easy way to generate a large amount of data (as opposed to driving around the arena collecting "real-world" images). The license plates generated using the license_generator.ipynb script from Lab 5 produced license plates which were very different from what the plates in the arena looked like (Appendix Fig. 6), and we had to heavily modify and process the generated plates to mimic those in the simulated world.

A summary of the modifications performed are listed below:

- Blurring the blank plate template to make the "Beautiful British Columbia" less defined
- Changing the font and font size of the license plate to that used in the simulated world
- With the characters written onto the license plate, it is blurred again to create the effect of the lower quality images in the world
- Compressed vertically, since the plates in the world are much flatter than the generated plate

- Shrink the image. Since the segmented plates from the arena usually had dimensions of around 40 by 170 pixels, we had to resize the generated plates (higher resolution) so the data is compatible for training the NN
- Segment the license plates to subimages of size 40 by 42 pixels each containing a character

To ensure that we had sufficient training data for each letter, we generated at least 40 license plates for each letter.

Binarizing Training Data

To further improve the performance of our NN, we decided to only feed binarized images into our NN, as recommended by our instructor. We chose to use the same adaptive gaussian thresholding method (cv2.adaptiveThreshold) as was used for the real-world data in order to keep the image processing method consistent. By adjusting the thresholding parameters through trial and error, we were able to binarize the generated images so that they looked very similar to the real world images (see Appendix Fig. 7). When binarizing the generated images, we added some random uncertainties to the binarizing parameters so that the images would show up with some variation in order to mimic real-world uncertainties.

One of the issues we ran into was that after we had binarized an image, we would convert the image array as a jpg file and save it. When we loaded the image from our folder, however, the image was no longer binary (we could tell since the image array contained values beside 255 and 0). We discovered that by saving the image as a png file, the binarization was preserved, so this was an easy fix.

Similar to the real-world data, after binarizing the generated images, we normalized them before feeding them into our NN for training.

Training Data Image Augmentation

Before feeding our generated images into the NN, we used TensorFlow's tf.keras.preprocessing.image.ImageDataGenerator to further augment the images to mimic the various uncertainties we expected to be present in the license plates obtained from the world. Specifically, we added shear and rotation to mimic perspective skews, height and width shifts to account for any inconsistencies in segmenting the license plates, and varying zooms to account for the uncertainty in character sizes since the dimensions of each segmented license plate is slightly different.

Training

We attempted to use the same NN structure from Lab 5, but when we ran the Python notebook, we ran into an error stating the size of our data did not fit the layers of the network. Through some testing, we discovered that it was because our image sizes were too small (40 by 42 pixels), and so by removing two layers from the network structure (one Conv2D layer and one

MaxPooling layer), the NN was able to train successfully. In our testing, we did not discover any adverse effects from removing the two layers, as we were able to obtain accurate predictions.

At this point, we briefly considered whether we should redesign the NN and enlarge the real-world images instead of shrinking the generated images (so that we could use the NN structure from Lab 5). But we soon decided against this since enlarging the image would not produce any more useful information that would help the NN make more accurate predictions.

For training, we decided to use a validation split of 20%. We trained the model through 100 epochs, where the batch size of the training data set was 16 (as in our lab). After training, we obtained a confusion matrix (Appendix Fig. 9) which indicated that all the images in the training set could be correctly predicted.

Testing and Retraining

After training, we input around 400 binarized real-world subimages into our NN, and it made about eight incorrect predictions. In particular, it tended to confuse the letter Z with the number 2, and sometimes Y and V. To account for these incorrect predictions, we added the real-world subimages to our training set and retrained the model. After this, we integrated the model into our robot. After testing about ~100 runs, we observed that in about every 20 runs, it would make one incorrect prediction; thus, it had an accuracy of about 95% for a perfect run. Specifically, it would sometimes mistake a P for an F, but these were very rare mistakes, and we decided the model was reliable enough for competition.

Integration (Sam and Ralph)

The robot controller is made up of 4 ROS nodes, with interactions seen in Appendix Fig. 8:

1. Red Line Detector
2. Pedestrian Detector
3. PID Controller
4. License Plate Detector

In summary, the PID controller (PID) follows the road by default, as detailed in Navigation above, and is updated by the Red Line Detector (RLD) if a red line is right in front of it and by the Pedestrian Detector (PD) if there is significant difference between consecutive frames (100 ms apart). If the PID sees that there is a red line ahead, it stops completely. Only when the robot is stopped do the messages published by the PD matter, as it constantly reports movement due to the robot's movement otherwise. When the PD reports that there is no movement, the robot dashes forward 'blindly' (not following the road). After crossing the next red line, the robot continues following the road. Refer to table below to see how the status of the PID is determined.

Red Line Detected	Movement Detected	PID Status
True	True	Stop
True	False	Go
False	True	Follow road
False	False	Follow road

The License Plate Detector (LPD) acts mostly separated from the other nodes, with the exception of the "/lpd_status" topic to report the status of the competition run. When the LPD first reports that the competition run has started, the PID executes a short command to drive the robot into position to follow the outer ring. When the LPD reports that the run is over, the PID stops completely. The LPD uses the trained NN saved from Google Colab. to make its predictions for publishing and, after publishing a prediction for each of the outer license plates, stops the competition run.

All 4 nodes use the callback triggered by the robot camera as the main control loop, with other callbacks acting as a means to update the state of the robot.

Lastly, the real time factor of the simulation was capped at 0.6 in order to improve the frequency of the frames captured by the robot camera. See here for a sped-up run with the robot controller completed.

Conclusion and Future Improvements

Looking back at our work in this project, there are still things we could do better that we will consider for the future such as:

- Utilize machine learning for tasks like navigation and license plate detection and segmentation
- Expand the versatility of our robot controller to handle the inner ring and scan license plates at sharper angles
- Increase speed and efficiency of our run
- Use object oriented programming to design our ROS nodes to improve modularity and neatness

However, with what we've accomplished, we are incredibly proud of our work and have gained valuable experience with computer vision, neural networks, ROS, and Linux. These skills will be extremely beneficial for us and we look forward to practicing them in future projects. The competition was a very challenging and enjoyable experience and we are very inspired by the work completed by other student teams.

Appendix

Figures

Figure 1: Simulated world layout (Source: ENPH 353 Competition Notes)

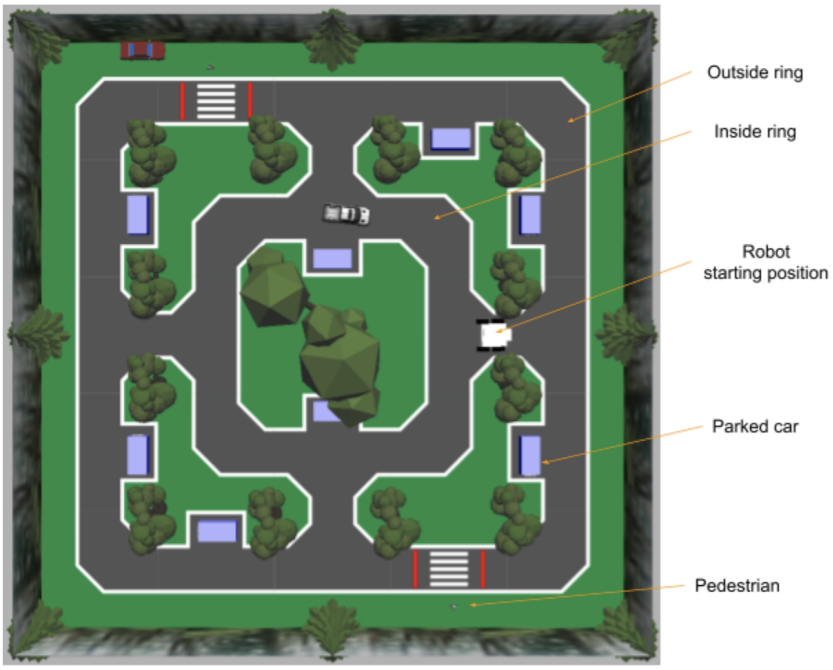

Fig. 1 Simulation playing arena overhead view

Figure 2: Robot camera feed (Source: ENPH 353 Competition Notes)

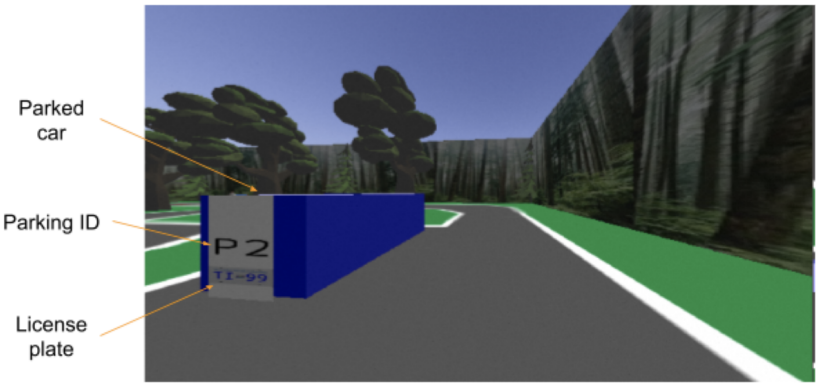

Fig. 2 In simulation camera view from robot
(used as input for team controller)

Figure 3: Road following binarization

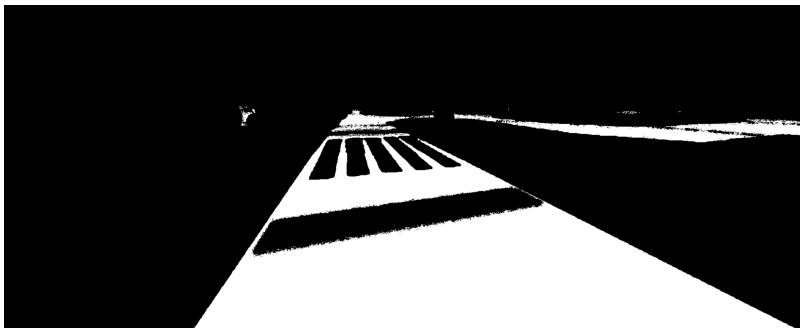

Figure 4: Line following binarization

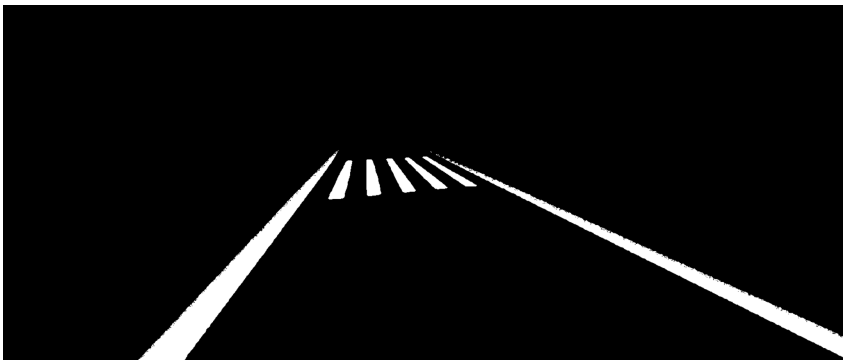

Figure 5: Example of “perfect” frame used for determining equation of outside line as well as view of robot as it approaches crosswalk

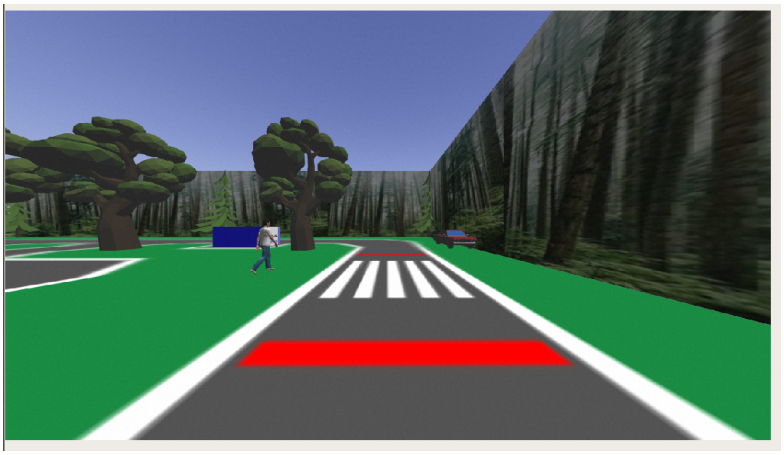

Figure 6: Comparison of Lab 5 generated license plate (left) and real-world license plate before modification and processing (right)

Figure 7: Comparison of generated license plate subimage (left) and real-world license plate subimage (right) after modification and processing

Figure 8: ROS Node Diagram

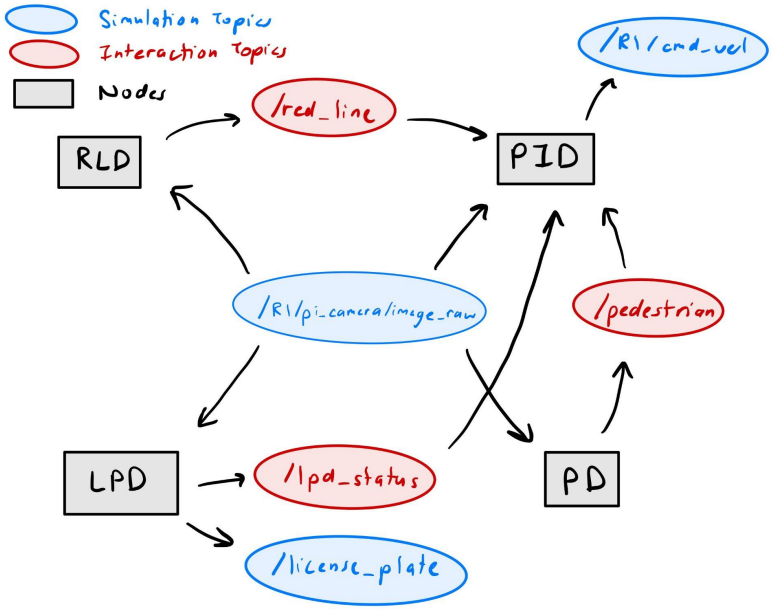

Figure 9: Confusion matrix (note this isn't the exact confusion matrix from the model we used in competition, as we re-ran the Colab. notebook afterwards. However, it shows the

same information - no incorrect predictions from the training set)

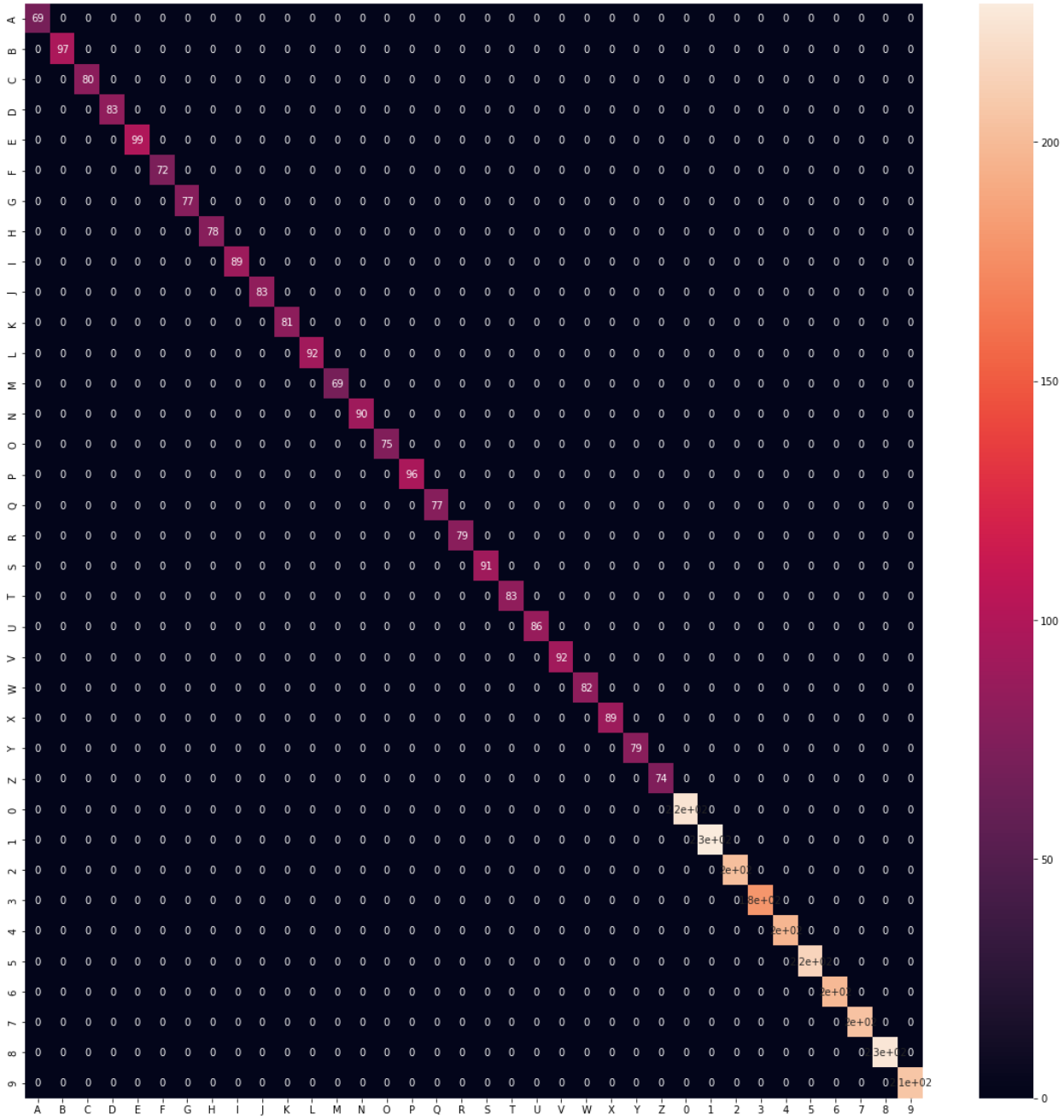

Figure 10: HSV Thresholded Red Line

Figure 11: Background Subtraction at Crosswalk. Left: Stationary Pedestrian. Right: Moving Pedestrian.

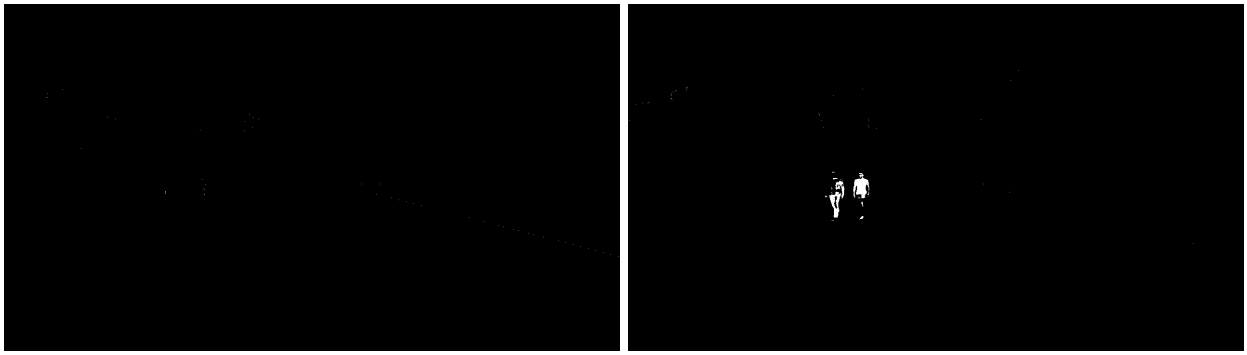

Figure 12: Sample License Plates
Left: Large aspect ratio means portion is cropped out
Middle: License plate is slightly angled
Right: License plate is rotated

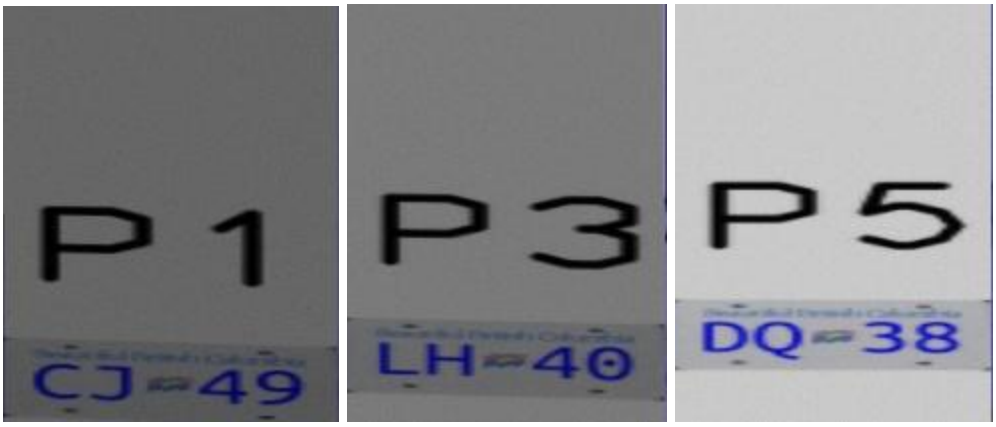

Figure 13: HSV Filtering. Left: Blue car masking. Right: License plate (white) masking