Enph 353
Andrew Rimanic
37824142

Introduction

This was originally intended to be the ENPH 353 final report for Andrew and Rohan. Unfortunately, Rohan was forced to defer this course so this report will only cover the license plate reading aspect of the project.

1 Plate Detection

1.1 Color Segmentation

The first step in the process of finding a license plate is to remove as much of the irrelevant information as possible. In this case that was very easy because all of the cars are painted colors that do not appear anywhere else in the world. The cars come in three colors, blue, green, and yellow. I created a separate mask for each color by thresholding hue and saturation ranges. Note that it is important to do this step in the HSV color space because it allows me to control for changes in brightness by setting the value parameter of each pixel to a constant. The three color range masks were then combined using a logical OR operation to acquire an image that featured only cars.

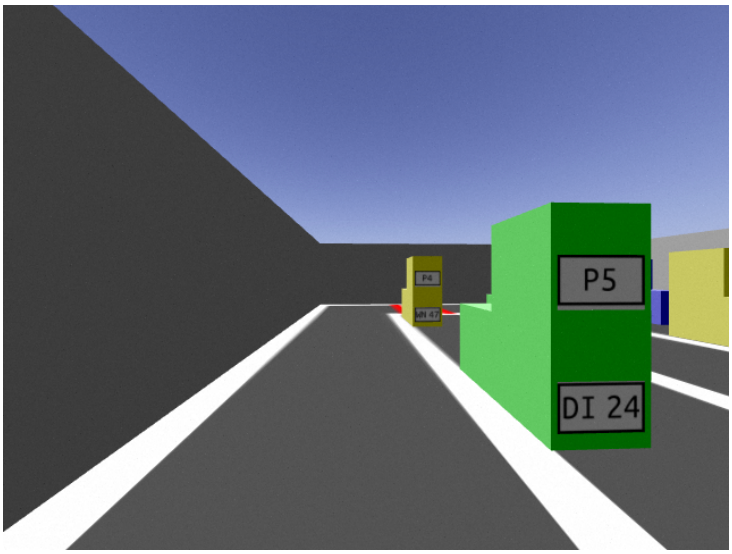

Figure 1: Original Image

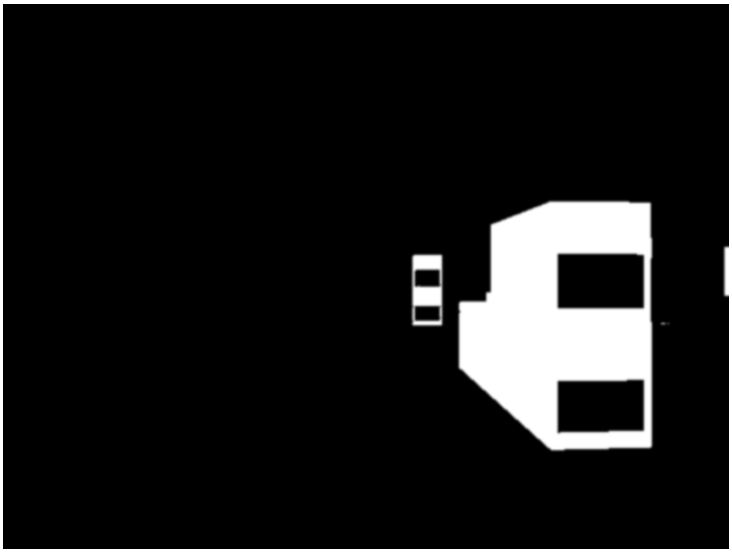

Figure 2: Mask as a result of color segmentation

1.2 Contour Detection

It turns out that the mask I found in the Color Segmentation section is actually more useful than the result of applying said mask to the original image. It contains all of the information needed to find the contours that will show the locations of the license plates.

Contour detection returns a hierarchy of contours. The top level of the hierarchy contains all of the outer contours of the cars. The children of these top level contours are the inner contours that represent the license plate and spot number. I crop a bounding box of each child contour and group them together as a pair.

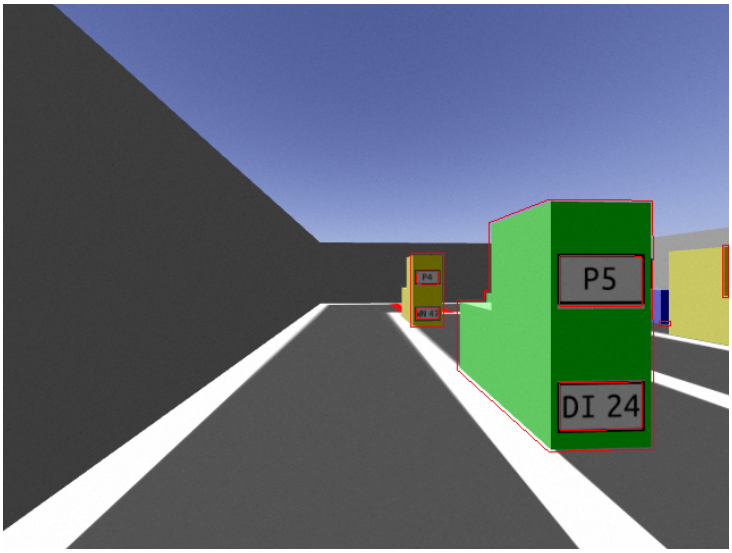

Figure 3: Contours found

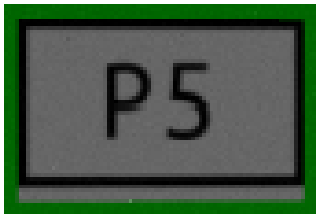

Figure 4: Result of cropping along contours

Figure 5: Result of cropping along contours

1.3 Perspective Transformation

The next step is to correct the perspective of the license plate. This is done by using a four point corner transformation[1] to acquire a rectangular image of the license plate.

The transformation I used requires the coordinates of each of the 4 corners to be specified. I was able to find these corners by splitting the untransformed image into quadrants and applying Harris Corner Detection to each quadrant. The maximum response in each quadrant was likely a license plate corner. The 4 corners were then passed to the persepective transform along with the original image. The result is usually a well cropped license plate. There were instances where the original image was too skewed to properly detect the corners. This usually led to the corner of the car being detected as the corner of the license plate which resulted in a poor perspective transformation. Luckily, the license plate has a known aspect ratio and the incorrectly transformed images always had an aspect ratio less than that of a license plate, so any results with low aspect ratios were simply removed.

Figure 6: Perspective correction and better plate isolation

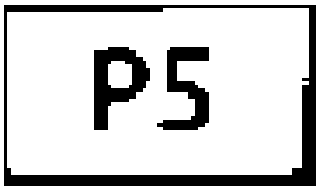

Figure 7: Perspective correction and better plate isolation

1.4 Letter Isolation

Letter isolation is a fairly simple task once the perspective is corrected for. The license plates are thresholded to get a grayscale image with all pixels being either white (255) or black (0). Gaps between letters are found by taking the average pixel value along each column. White space will return values near 255, while letters will have a significantly reduced average value due to the black pixels.

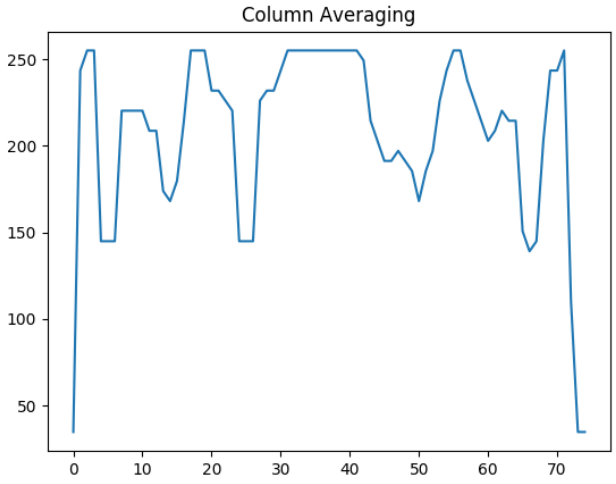

Figure 8: Average of "DI24" along columns

Applying a peak finder with a high threshold value (around 250) to the vector of average column values gives the locations of the gaps between letters. By using the left and right edges of each peak I was able to accurately crop the edge of each character.

At this point it becomes very easy to determine which of the images in a pair is the spot number and which is the license plate. The spot number will only have 3 characters compared to the license plate which will have 4.

An example of the original image and the resultant individual character crops are shown below.

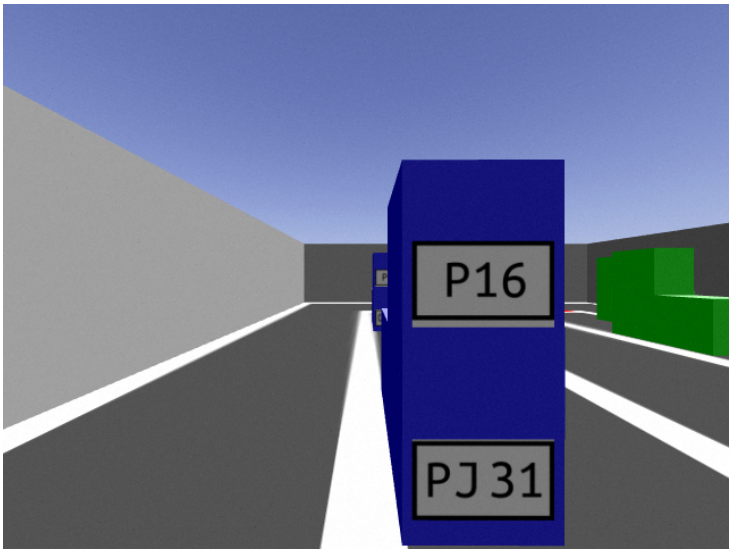

Figure 9: Original Image

Figure 10: Cropped spot number

Figure 11: Cropped spot number

Figure 12: Cropped spot number

Figure 13: Cropped plate

Figure 14: Cropped plate

Figure 15: Cropped plate

Figure 16: Cropped plate

1.5 Classification

The image data is now in the format required to be classified by a Convolutional Neural Network. There are 3 CNNs used in this classification, one for letter classification, one for number classification from 0-9, and another for number classification between only 0 and 1. All networks share very similar architecture.

1.5.1 Architecture

I ended up using the following architecture for my number and letter detection networks

1. 30x60x1 pixel input image

2. 16 channel 3x3 filter convolutional layer with same size padding
3. 2x2 max pooling layer
4. 32 channel 3x3 filter convolutional layer with same size padding
5. 32 channel 5x5 filter convolutional layer with same size padding
6. 2x2 max pooling layer
7. 32 channel 7x7 filter convolutional layer with same size padding
8. 50% dropout layer
9. $< numlabels > \times 5$ fully connected layer
10. $< numlabels > \times 5$ fully connected layer
11. $< numlabels >$ fully connected layer

All layers use ReLu activation. The model was compiled using categorical cross-entropy loss and Adam optimization.

1.5.2 Training

To train the networks I built a data generator that creates images of a single black letter or number on a white background. The letter is given slight variations in its placement and size. A random amount of image pixellation is also added to the image. I experimented with adding some blur to the images after pixellation but found that it only made the CNNs perform worse.

Data was generated in this fashion such that each training set had roughly 1000 images for each unique character in the set. These images were then split into training and validation sets before being put into batches for training. I tried various batch sizes ranging from 32 to 5000 and found that a batch size of 1000 was effective for training accurately while still maintaining a reasonable training time.

Each CNN was trained for 500 epochs, but with the early stopping callback enabled, few required more than 50 epochs to reach 100% validation accuracy.

1.5.3 Performance

Each CNN reached a point where it was able to classify any letter produced by the data generator with 100% accuracy. Unfortunately this didn't translate as well as expected when tested in the ROS simulation. It is hard to say why this is because visually, the images produced by the data generator look extremely similar to the ones extracted from the ROS simulation. In almost every case the incorrect prediction was made with more than 99% confidence so it is likely that the training set does not encompass all of the variables present in the simulation data. It's possible that the perspective transform does not perfectly align each letter vertically, or that the perspective is not perfectly unskewed. Some images also feature random black pixels from the plate borders. To address these sources of error I would suggest adding a few variables to the data generator. These are rotation, stretching in X and Y, and random pixel noise.

Nonetheless, I was able to detect the majority of license plates correctly. A few tricks that helped performance were:

1. Setting a very large lower limit for image size. This meant that only the highest resolution images were processed. Higher resolution images had a better chance of being correctly identified, but it also mean that the robot had to be very close to a plate before it would read it

2. Keeping a running count of the number of times each plate was found, whether correct or incorrect, and then choosing the plate which appeared the most times. This had the effect of weeding out any errant misclassifications. This count was done independently for each parking spot to control for screen time. This way plates were only compared to other plates which had the same amount of screen time. Of course, this only works if the classification of the parking spot is consistent so it was very important to make the corresponding CNNs very stable.

Overall, performance was good, but not as good as I would have hoped for. I was able to achieve some perfect runs but the most common result by far was getting all parking spots correct and 5/6 plates correct. The mislabeled plate was usually in the same spot each time and upon closer inspection I found that this image looked different than the others. There was often a horizontal line of missing pixels through the letters in this plate that I wasn't

able to adapt to. I can only conclude that there must be a bug in my image preparation that occasionally removes these pixels.

Resources Used

[1] I was able to find a useful snippet of code that performed a four point transform to get a "straight on" perspective of the license plates. The code can be found at

https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/