ENPH 353 Self-Driving Car

Final Report

April 17, 2022

Sabrina Ashik

Olivia Kim

Background

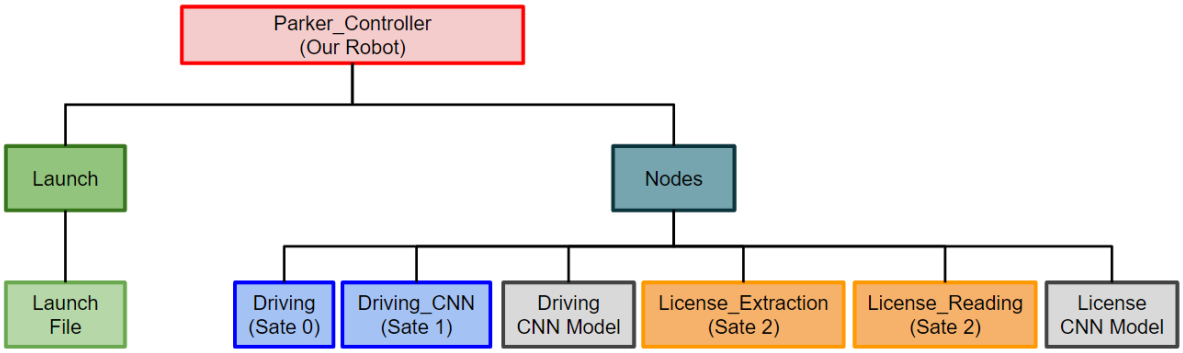
The goal of this project is to develop an autonomous car that drives through a simulated environment in ROS. In addition to driving around, the car must be able to detect the cars it is passing by and return its license plates and associated parking IDs. We must implement both the driving and the license reading algorithm and finally the overall integration.

Software Architecture

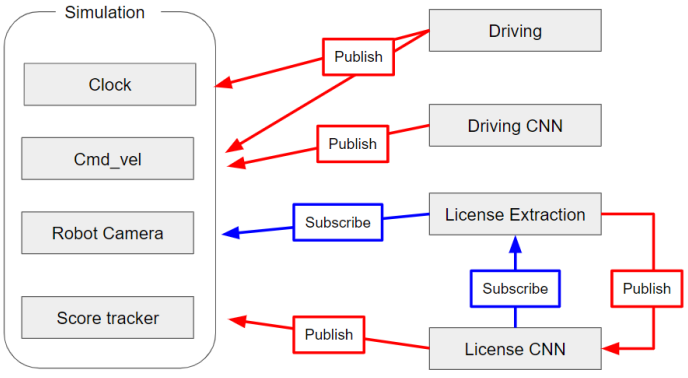
Our robot follows a state machine with the following states:

Sate 0	Drive the car to the outer loop, then orient the car
Sate 1	Drive on the outer Loop
Sate 2	Detect and Read License Plate
Sate 3	Detect Pedestrian
Sate 4	Drive into the inner Loop
Sate 5	Drive on the inner loop

We implemented each of the states as a node in our file structure. This structure allowed for efficiency and ease when debugging. This way, we are able to focus on an individual aspect of the car at a time and we are able to break our tasks into smaller modules. Furthermore, it makes it easier for us to test robots throughout our design process. However, due to the time constraints of this project, we were only able to implement from State 0 to State 2. Following is our file structure:



As mentioned above, each state is a “node” inside of our “Nodes” file. We divided our Sate 2 into two nodes, License_Extraction and License Reading, because these were very different tasks the robot had to complete. These nodes communicate with each other using the Publisher-Subscriber method. This is also how the car communicated with the clock, command velocity, and the score tracker.



Driving Control Mechanism

As outlined in State 0, State 1, and State 4 of our state machine structure, there are 3 modes of driving control that the robot will follow.

Hard-coded Driving

To initiate the driving State 0 (start driving forward and turn to the left to enter the outer loop), the time and speed measurements were manually taken via trial and error on the Teleop-Twist keyboard.

These commands will be hard-coded directly to the *cmd_vel* subscriber, which will listen to the commands outlined and drive out onto the outer loop. We were able to get away with hard-coding this section as it was a short track, however, ran into certain issues when attempting to hard-code the entire

CNN Controlled Driving

After State 0 is completed and the robot has driven onto the outer loop, we will switch to State 1, which is a CNN-controlled method of driving. This CNN model was built to drive in State 1 specifically, which was the outer loop of the competition space. We will need a similar but differently trained CNN model to implement driving in the inner loop at State 4.

I. Data Engine (Extraction & Wrangling)

A. Extraction

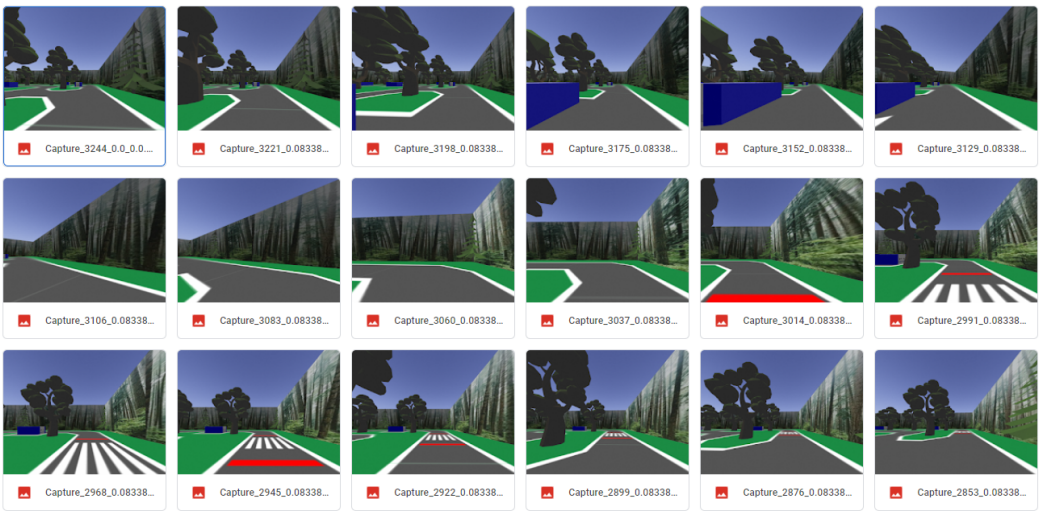
To build and train our Neural Network, a dataset of camera feed images and the corresponding driving commands that match each frame was collected. To quickly extract these 2 data, we manually drove around the outer loop using the Teleop-Twist Keyboard while a *pi_camera* subscriber ("`self.camera_sub`") and *cmd_vel* subscriber ("`self.car_maual_driver_sub`") ran in parallel to access the camera feed and driving command data.

To reduce the size of data collected (later useful to prevent RAM storage errors), *rospy.time* was used to record data at given time intervals instead of at every camera frame. The final time interval used to collect 140 total raw image samples was 0.5 seconds (2 fps recorded).

B. Wrangling

In order to link each camera feed image to its corresponding driving command data, each collected frame was labeled as the string-form of the *cmd_vel* command (*linear_x* and *angular_z* commands) associated with it, using an underscore "_" to separate these commands for extraction later. To further reduce the size of data samples collected, the images were resized to 25% of their original dimensions using CV2 before writing to the folder.

At the end of this data wrangling stage, we were able to compile an organized folder of camera feed images labeled as their matching driving commands:



II. CNN Model

A. One-hot Encoding

After collecting and wrangling the image and driving data in the steps outlined above, this data was uploaded to Google Drive where a set of one-hot encodings was then generated using this given data.

5 modes of driving were established, each being:

- “Straight” (driving straight with only positive *linear_x* value given)
- “Straight_Left” (driving straight while turning to the left, positive *linear_x* and *angular_z* values given)
- “Left” (turning to the left, only positive *angular_z* value given)
- “Straight_Left” (driving straight while turning to the right, positive *linear_x* and negative *angular_z* values given)
- “Right” (turning to the right, only negative*angular_z* value given)

Depending on the name of each image, the frames were categorized into one of these five driving modes for the neural network to train on.

B. Building Model

Two datasets, X dataset for the images and Y dataset for the driving modes (5 outlined above), were extracted from the one-hot encoded data and were used to train and test our samples.

The training parameters used to build this CNN model was as follows:

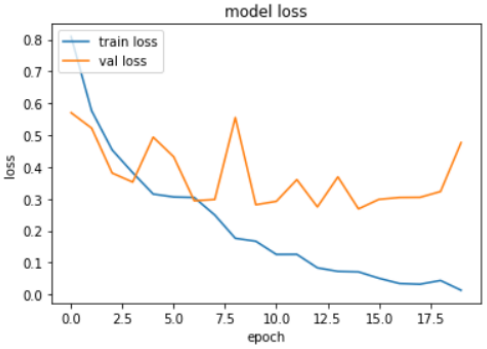
- 0.2 validation split (112 training, 28 test of 140 total examples)
- 1e-4 learning rate
- Batch size = 5, trained over 20 epochs

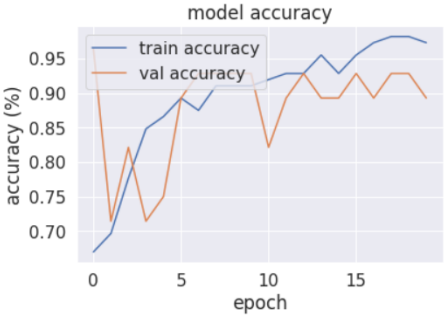
The layers were adjusted over several iterations in order to achieve accurate predictions while keeping a workable model size. Note that the “dense (Dense)” layer was adjusted to be lower after seeing that the model accuracy has plateaued after ~13 epochs.

Summary of Neural Network:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 178, 318, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 159, 32)	0
conv2d_1 (Conv2D)	(None, 87, 157, 16)	4624
max_pooling2d_1 (MaxPooling2	(None, 43, 78, 16)	0
flatten (Flatten)	(None, 53664)	0
dropout (Dropout)	(None, 53664)	0
dense (Dense)	(None, 64)	3434560
dense_1 (Dense)	(None, 5)	325

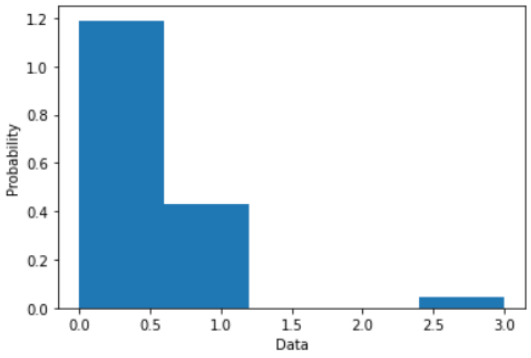
Total params: 3,440,405
Trainable params: 3,440,405
Non-trainable params: 0





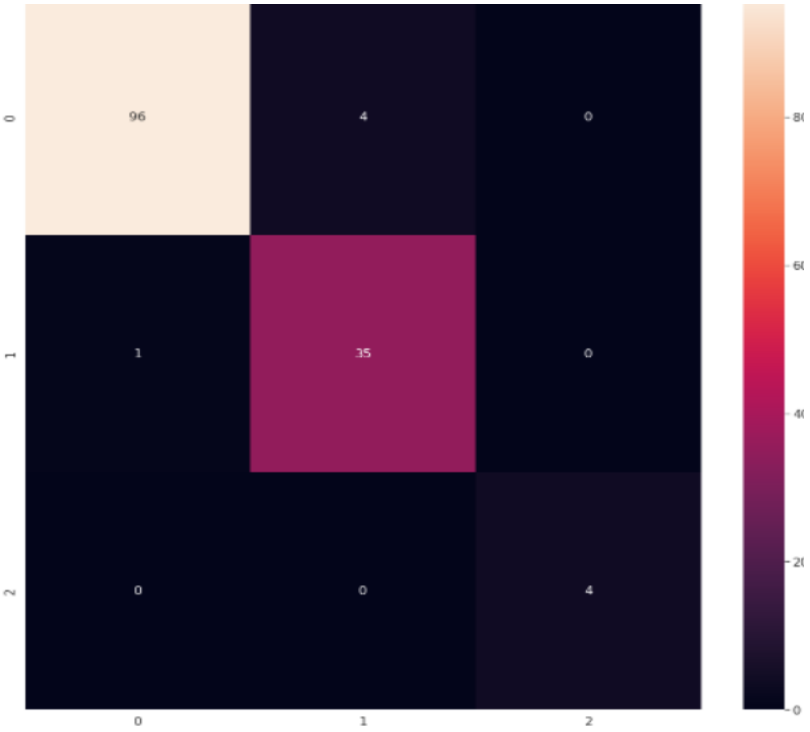
I. Testing and Comparing Models

After training the model and Google Colaboratory, it was exported and configured into the ROS folder to start predicting when the robot is launched.



The following probability histogram shows the prediction probability for each entry of one-hot encoding. From this histogram and the confusion matrix, we can observe that our data was mostly skewed towards the first entry which was “Straight”. This is reasonable because the competition space mostly consisted of straight road paths and few turns. This resulted in our dataset consisting mostly of situations where the required driving command was “Straight”.

Confusion matrix



To improve our model, we would collect more data on specific turns, and incorporate a smoother driving algorithm for manual driving so that the model can learn to drive closer to our expected behavior.

Pedestrian and Truck Interrupt

Although we were not able to complete the pedestrian and truck detection/interrupts due to time constraints of this project, our proposed implementation is as follows:

For pedestrian detection, we would have used CV2 to detect a red line at the bottom of the frame in order to detect that we are approaching a crosswalk. Once this line was detected, the robot would stop. By looking at the top part of the frame, we can isolate the second red line of the crosswalk. Next, we will detect the pedestrian using the findContour function. When the pedestrian is not at the crosswalk, it should only detect one big contour which is the red line. However, if the pedestrian is at the crosswalk, it would block the red line and there would be two contours detected. This would have allowed us to figure out when the pedestrian is not at the crosswalk and the robot should be driving.

Similarly, for truck detection, we would wait at the intersection into the inner loop until detecting a grey figure (truck) using the same findContour function. After the truck passes, we would configure it back to the inner loop driving command and drive until it detects the truck again.

License Detection Mechanism

There are two parts to License Detection as mentioned previously. The first part is License plate detection. This module is responsible for detecting the license plate as the robot drives around the track and sends a picture of the license plate to the second part, which is the License CNN node. The second module, License CNN, is responsible for detecting individual letters and numbers on the license plate image, then sending it to a neural network to predict the reading. After it gets the reading for all 4 alphanumeric digits, it will then publish it to the score tracker alongside its parking ID.

I. License Plate Detection

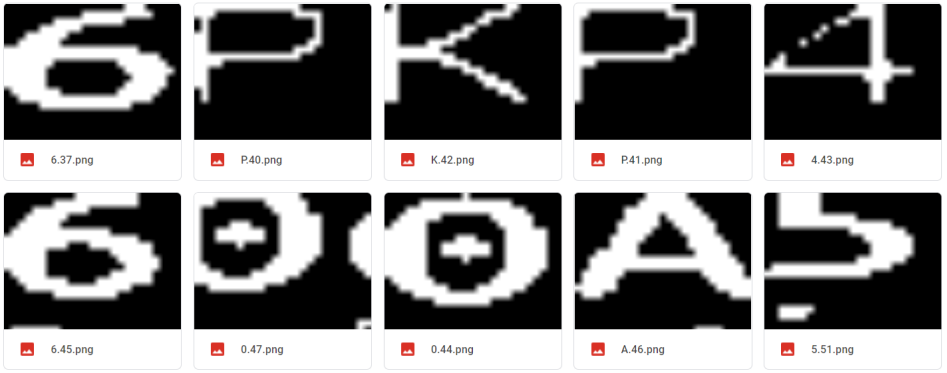
This node subscribes to the robot's camera view and uses the incoming frame as the input data. Then, it will process each incoming frame to find the license plates. As the cars are always parked on the left side of the road and the license plates are located near the bottom, only the bottom left corner of the frame was considered when processing. This was done by cropping out the unwanted area before beginning to process the image.

A. Image Processing

The method used for detecting the license plate was the findContour function in OpenCV. This requires a binary version of the image. Therefore, the first step was to convert the incoming images to an HSV image. This was done to account for the change in brightness of the images. Due to the difference in sunlight in the simulation, some license plates were lighter gray than the others. This made it very difficult to find a binary threshold that worked for all different lighting of images with the regular binary conversion function in OpenCV. However, HSV images are able to filter for Hues, Saturation, and Values within the set range. This allowed for the better binary conversion of images.

The threshold was set for the binary image so that everything except for the car would be black. Because the middle portion of the car was gray (As it had the parking ID and the license plate), only the left and the right-hand side of the car turned white. Therefore, we were able to detect a car by looking for the two largest contours that existed within the frame and checking if they are tall rectangular with a large enough area (the exact value was area > 2800). This was done by counting the number of edges the two largest contours have, and if they each have 4 edges then they are most likely rectangles.

The license plates are in between these rectangles at the bottom. Therefore, using the bottom right edge location of the right rectangle and the bottom-left edge location of the right rectangle, we were able to find the exact location of the license plate. Finally, the image was cropped to just show the license plate, and then published to the license reading node. Here are the processed cropped images:



II. License Reading

After receiving the license plate images from the License Extraction node, it must further process the images in order to first detect the alphanumerics and then feed them to the CNN.

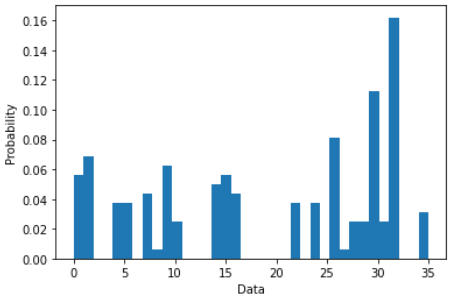
A. Image Processing

First, the image was converted to the binary image where most of the plate is black except for the alphanumerics. Then each alphanumeric and its location were found by applying rectangular contours around them using the boundingRect function in OpenCV and finding the location of each contour. After the locations were found, each alphanumeric was cropped and resized from a (25, 25) size image to a (25, 25, 3) size image in order to input it to the CNN.

B. CNN Model

The processed images were saved on Google Drive for CNN. The name of the images was changed to the alphanumeric it represented. This made the one-hot-encoding process easier as the Y dataset pair of the images can easily be determined from its filename. We had one CNN for both letters and numbers, resulting in 36 possible pairs ([A, B, …, Y, Z, 1, 2, …, 8, 9]).

To train the model, validation split of 0.2, learning rate of $1x10^{-4,}$ and epoch size of 80 were used. The composition of the data is shown in the following histogram.

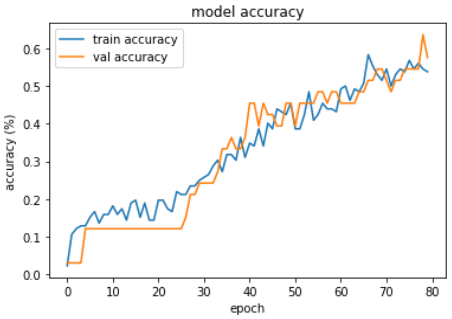


The total dataset size was 165 images. However, as the composition shows, there are some letters and numbers missing from the dataset. This led to the poor accuracy of our model on the actual simulation.
The lack of data is due to our data collection method. In order to collect the images, we had to restart the simulation and drive by every single car. This was very time-consuming and difficult as restarting the simulation did not guarantee that the new license plates had a completely different set of alphanumerics. Collecting more data would have drastically improved our performance

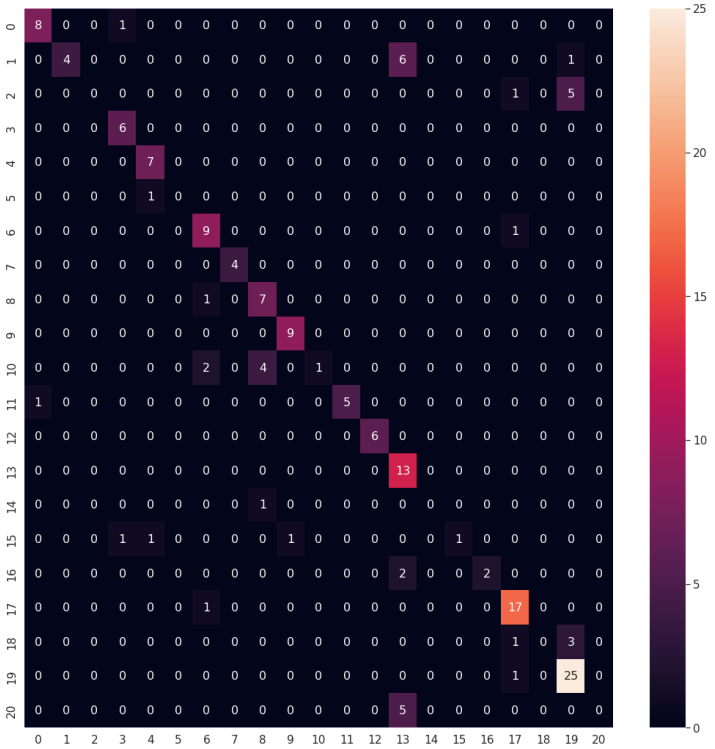
Summary of Neural Network:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 23, 23, 32)	896
max_pooling2d (MaxPooling2D)	(None, 11, 11, 32)	0
conv2d_1 (Conv2D)	(None, 9, 9, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 4, 4, 64)	0
conv2d_2 (Conv2D)	(None, 2, 2, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 512)	33280
dense_1 (Dense)	(None, 36)	18468

Total params: 108,068
Trainable params: 108,068
Non-trainable params: 0



Confusion Matrix:



Conclusion

During the competition, our robot was able to start and stop the clock and the car was successfully able to drive on the road. Although the clock stopped after 25 seconds, which was our strategy, the robot continued to drive afterward and was able to make a full circle around the track.

However, we failed to integrate the integration of the License Reading node and the Score Tracker, and hence the robot was unable to return the license reading even though it was detecting the license plates. Some failure methods we identified during the building process were as follows:

Failed Methods

I. **Hard Code Driving**

After hard-coding Stage 0 of driving, we asked the question: “Why can’t we simply hard-code the entire driving algorithm instead of implementing a complicated CNN?”. After attempting to hard-code the first half of the outer loop, this method proved to not only be highly inefficient but also unreliable, as the driving behavior in hard-coded driving was highly sensitive to changes in the real-time factor. This resulted in the robot driving slightly differently at every roslaunch.

II. **Binary Detection of license plate**

In order to detect the license plate, our initial approach was to simply convert the frames to binary images and then detect the plate location. However, this approach didn’t work as well because there would be white dots on unwanted areas due to shadows or shades. Furthermore, finding the edge of the license plate in order to crop out just the license plate was extremely difficult with this method. This is because there was no guarantee that the white part found was the right edge of the plate.

III. **Image Wrangling with python**

In order to train the license plate CNN model, we tried to wrangle the sample license plate images by adding noises. However, we found that the wrangle images were not very accurate to what the camera sees. This is because sometimes the letters would be skinner or squeezed in a corner, etc. Hence we decided to collect the images that the robot sees by driving around and saving the detected license plate images instead. This helped to increase the accuracy of our model.

IV. **RAM Storage**

The biggest issue when training and testing different versions of the driving CNN was the size of the data sample.

We often ran into RAM storage issues where we were unable to compile or restructure the dataset due to RAM failure. This was troubleshot by reducing the dataset collected (by resizing the camera feed images and collecting data at a lower frequency) and adjusting the number of layers in our neural network. Studying the

accuracy trend in each iteration of the epoch allowed us to balance between model accuracy and the number of layers we needed.

Future Improvements

With more time, we would develop both the Driving and the License CNN model further. For both of the CNN models, it can be seen that some data are either missing, or there is not enough data for the model to be accurate. This was largely due to a lack of time. Therefore, given more time, we would have been able to collect more data (specifically on left/right turns and some alphanumeric digits) and have the same amount of data for each hot encoding, which would have resulted in a better performance.

Additionally, the integration of each node could have been implemented more effectively. For example, the license plate reading node was not integrated property with the score tracker, which resulted in the robot not being able to send in its reading. Additionally, the state 0 node was not integrated with state 1 at all. Ideally, as state 0 ends, it would send a message to state 1 to begin its tasks. However, for the competition, state 1 was programmed to simply wait 5 seconds, which accounted for state 0. This led the transition between state 0 and state 1 to be very rough and there was unnecessary wait time in between.