ENPH 353 Machine Learning & Computer Vision Project

PayByRobot: An Autonomous Parking Agent

TEAM FAMAXSEVEN

Author:

Ruixin Qiu (74011354)

Background

Currently, UBC Parking has some difficulty in recruiting people to record vehicle license plates on campus. In order to solve the problem, UBC Parking has initiated a project to collaborate with UBC Engineering Physics students. According to the project, the students need to design an autonomous parking agent which can move around the campus, upload the license plates and identify parking IDs without breaking the traffic rules. For the sake of security, only one camera is allowed to be placed at the front of the robot.

In order to evaluate the performance of the robot before putting it into use, UBC Parking has provided the students with a Gazebo playground and a scoring criteria to simulate their robots. In the virtual Gazebo world shown in Figure 1, there are two main tracks (inner loop and outer loop), eight parked vehicles, two pedestrians and one moving vehicle. UBC Parking will also hold a competition for students to show their design virtually. The students who score the highest points will win the competition. As a team (Team Famaxseven) which has been invited to the competition, our goal is to design a robot that can score full points using the least amount of time.

Table 1. Scoring criteria for the competition

Criteria	Points
Hit pedestrian	-10 each
Hit vehicle	-5 each
Two wheels outside track	-2 each
Complete one lap	+5
Upload correct license plate and parking ID (outer loop)	+6 each (+36 in total)
Upload correct license plate and parking ID (inner loop)	+8 each (+16 in total)

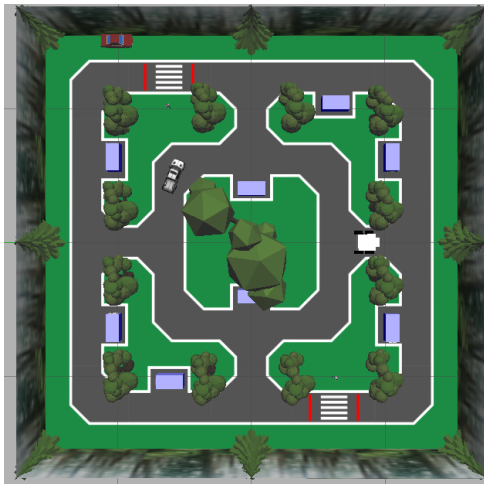

Figure 1. Virtual Gazebo playground

Overview

In order to score full points in the competition, our robot PayByRobot must be able to avoid violating traffic rules, recognize the environment and identify the license plates accurately. To achieve the goal, our robot PayByRobot utilizes three main systems: plate recognition system, object detection system and PID controller system. All the three systems work collaboratively to manage the motion and functions of PayByRobot. The plate recognition system can locate and identify the letters and numbers on the license plates and the parking IDs. The object detection system can detect the locations of pedestrians and moving vehicles. The driving system can cooperate with the other two systems to move the robot around the virtual world safely. The details of each system will be further discussed later in this report.

Software Architecture

Github, Google Colab and Google Drive are used in this project to help our team share and backup our codes, data and model files. We put our four controller scripts (ObjectDetector.py, PlateRec.py, PID_controller.py and competition.launch) on Github in the repository of qrx10/ENPH353-Competition under my_controller\src. The object detector script is responsible for the operation part of the object detection system. It detects pedestrians and moving vehicles and publishes their locations through a ROS publisher node. The plate recognition file is responsible for the operation part of the plate recognition system. It locates the license plates and identifies their letters and numbers. It uploads these messages through two ROS publisher nodes. The PID controller script is the brain of our system which collects and processes data from the plate recognition system, and the object detection system is to control the motion of the robot.

Google Colab is used to construct pipelines to train the models. PlateNumberRec.ipynb and LocNumRec.ipynb are the pipelines we set to train CNN networks for license plates and parking IDs identification. Meanwhile, ObjectDetectionYOLOv3Tiny.ipynb is used to train the custom YOLOv3-Tiny model for the object detection system.

Besides, Google Drive is used to store large files that cannot fit in Github. There is also a link in the README.md directed to the folders and instructions on where to put these files when necessary. The folder Competition Model contains all the h5 model files needed for the project. Competition Data contains all the required data for training.

Plate Recognition System

Plate recognition system is the first system that our team constructed. It acts as one eye in our whole system. It not only recognizes the license plate, but also helps our robot to identify its location. There are two steps involved to identify the license plates and parking IDs.

1. Locating License Plates & Parking IDs

For the first step, our team first collected some random images containing the parked vehicles and analyzed their features. We noticed that the license plate and the parking ID for a parked vehicle is on a white board at the rear end of the vehicle shown in Figure 2. From the scripts that generated the license plate, we found that the white boards have a width of 600 pixels and a height of 1800 pixels. The license plate is located at 1250 pixels below the top of the white board. Because the license plates are small and hard to locate, we decided to locate the large white board first and then calculate the location of the license plate in relation to the location of the large white board.

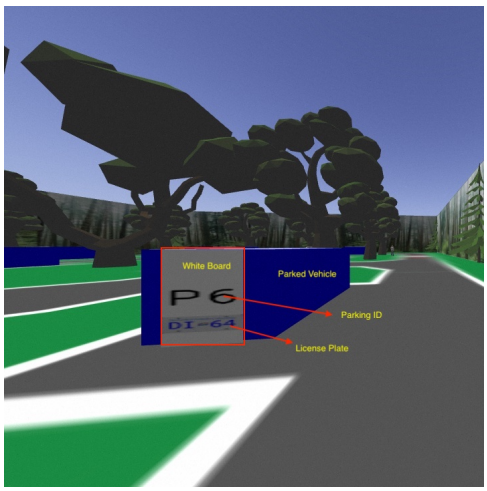

Figure 2. Example of a parked vehicle with labels

In order to locate the large white board, we examined the color code of the white board. We first converted the image into a grayscale image shown in Figure 3. Then we used a color picker to get the color of the white board, which was between 100 and 105. Therefore, in order to extract the white board from the background, we tested it by using two binary masks of threshold value 100 and 105. The result is shown in Figure 4.

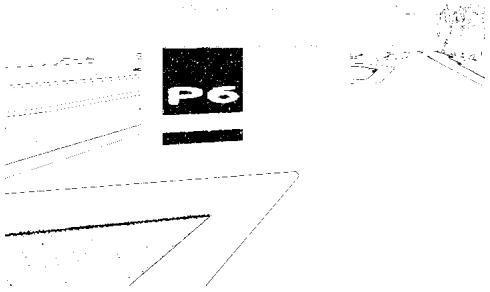

Figure 4. Image after binary masks of 100 and 105

From the results, we can see that only the white board is black and most of the other parts are white. Then we can reverse the colors and draw contours around the upper part of the white board. However, we noticed some small white dots inside the contours which would reduce the accuracy of our contour drawing before we did so. Therefore, we blurred the image before binary masking. Besides,we cropped out the top third part of the image, because, as is known to all, the parked cars would not appear in the sky. In addition, we increased the number of binary masks because brightness varies with locations.

After preprocessing the images, we calculated contours on the binary image using OpenCV and sorted the contours by their areas. The largest contour which has an area larger than 12000 should be the contour around the upper part of the white board. Then we could approximate the contour to a polygon. As we knew that the white board was a rectangle, we expected the approximated polygon to have four vertices. The four vertices were then sorted in the order of top left, top right, bottom left and bottom right. These four vertices indicate the location of the upper part of the white board as is shown in Figure 5. With the location and the shape of the upper part of the white board, we can use proportionality to calculate the location and the shape of the whole white board. However, as is shown in Figure 5, the white board is not a rectangle anymore. Instead, it has been distorted because the angle is not 90 degrees from the camera to the plate. Therefore, we projected the distorted white board onto a 600×1200 blank sheet using perspective transformation as is shown in Figure 6.

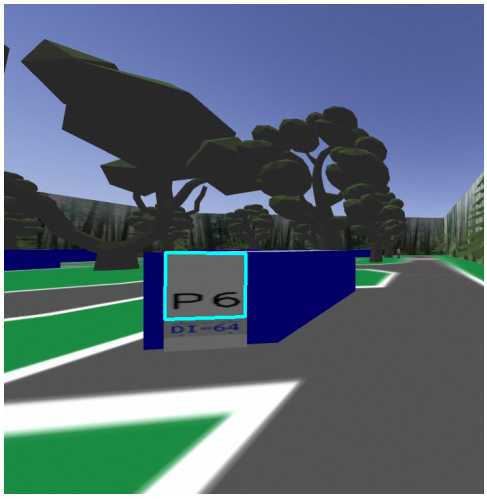

Figure 5. Upper part of the white board being bounded by blue contours

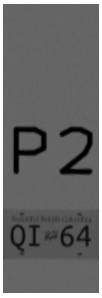

Figure 6. White board being perspective transformation

2. Recognizing License Plate & Parking IDs

After locating the license plate and parking IDs, we now have an undistorted grayscale white board. Our team can crop out the letters and numbers shown in Figure 7. The numbers and letters from the plate licenses have a size of 100×240, and the parking IDs have a size of 60×80. We labeled the images by putting them into different folders. We also created a folder named "%" to store the flag of British Columbia, which indicates that the letters are not well cropped and that the detection has inaccurate results. The main task now is to develop Convolutional Neural Network (CNN) models to identify each character accurately.

Figure 7. Cropped characters from the white board

In order to enlarge our database, we added noises to the 1500 images collected. We also randomly blurred, sheared, rotated, brightened, zoomed and cropped the images. As a result, we have a database which is nearly five times as large as the original one. The benefit of this method to generate new data is that all data is based on what we would get from the robot in the competition. The prediction would be more reliable and accurate. The histogram of the datasets are shown in Figure 8.

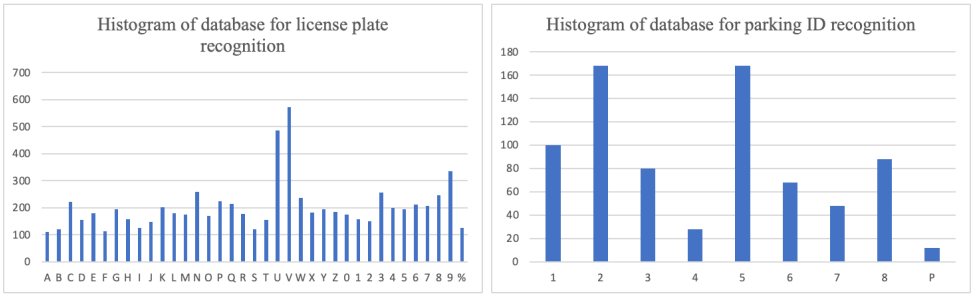

Figure 8. Histograms of datasets for license plate and parking ID recognition

The next step is to build a pipeline to construct and train the models. We built two pipelines to construct CNN models to recognize the license plates and parking IDs respectively. We randomly shuffled the data and split it into two sets: 80% in the training set and 20% in the validation set. The structures of the models are shown in Figure 9. We used a learning rate of 0.0001 for both models. The epochs of the model to recognize plate licenses is 150. The number of the model to recognize parking IDs is 80. Both models show high validation accuracy after being trained. The plate recognition model has an validation accuracy of 95% while the parking number recognition model has an validation accuracy of 99% as is shown in Figure 10.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 238, 98, 32)	320
max_pooling2d (MaxPooling2D)	(None, 119, 49, 32)	0
conv2d_1 (Conv2D)	(None, 117, 47, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 58, 23, 64)	0
conv2d_2 (Conv2D)	(None, 56, 21, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 28, 10, 128)	0
conv2d_3 (Conv2D)	(None, 26, 8, 128)	147584
max_pooling2d_3 (MaxPooling2	(None, 13, 4, 128)	0
flatten (Flatten)	(None, 6656)	0
dropout (Dropout)	(None, 6656)	0
dense (Dense)	(None, 512)	3408384
dense_1 (Dense)	(None, 37)	18981

Total params: 3,667,621
Trainable params: 3,667,621
Non-trainable params: 0

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 78, 58, 32)	320
max_pooling2d_12 (MaxPooling	(None, 39, 29, 32)	0
conv2d_13 (Conv2D)	(None, 37, 27, 64)	18496
max_pooling2d_13 (MaxPooling	(None, 18, 13, 64)	0
conv2d_14 (Conv2D)	(None, 16, 11, 128)	73856
max_pooling2d_14 (MaxPooling	(None, 8, 5, 128)	0
conv2d_15 (Conv2D)	(None, 6, 3, 128)	147584
max_pooling2d_15 (MaxPooling	(None, 3, 1, 128)	0
flatten_3 (Flatten)	(None, 384)	0
dropout_3 (Dropout)	(None, 384)	0
dense_6 (Dense)	(None, 512)	197120
dense_7 (Dense)	(None, 9)	4617

Total params: 441,993
Trainable params: 441,993
Non-trainable params: 0

Figure 9. Model structure for the plate recognition model (right) & parking ID recognition model (left)

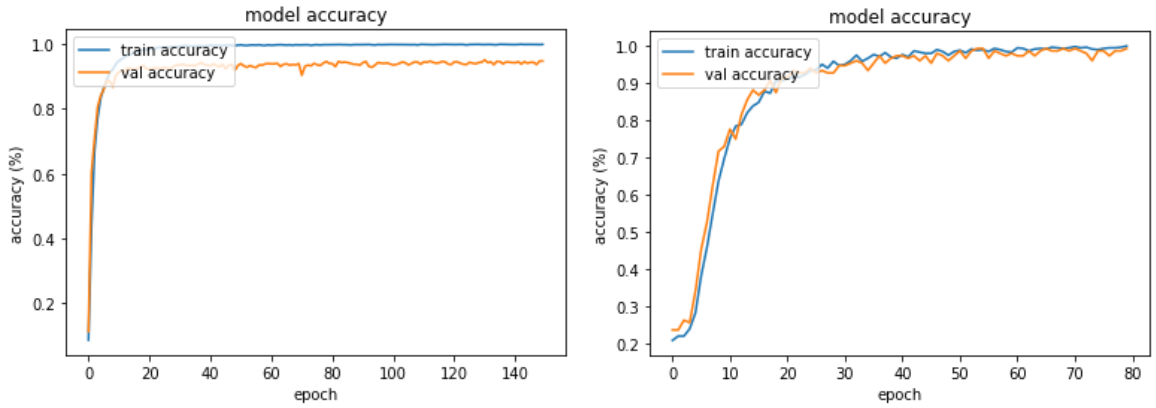

Figure 10. Model accuracy for the plate recognition model (right) & parking ID recognition model (left)

With our CNN models, we can predict the parking IDs and license plates, and publish them. But before publishing them, we checked the location of the top left and right vertice of the white board. If they were too close to the edge (i.e. has a distance less than 37 to the edge), we ignored these predictions because we found that the accuracy of the prediction was a function of distance to the edge shown in Figure 11.

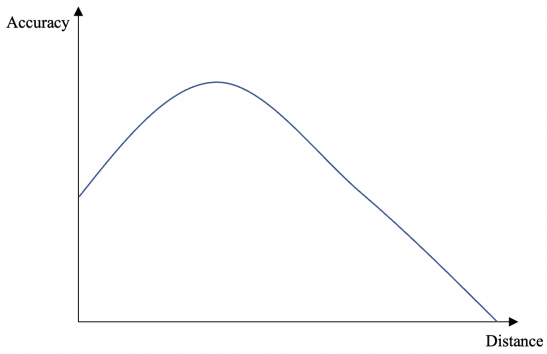

Figure 11. Accuracy of the prediction versus distance of the white board to the edge of the image

Object Detection System

Plate recognition system is the first system that our team constructed. It also acts as the other eye in our whole system to detect pedestrians and moving vehicles, and helps prevent our robot from hitting them. We built our own object detector using YOLOv3-Tiny, which is a “state-of-the-art, real-time object detection system”. YOLOv3-Tiny applies a single neural network to the full

image, divides the image into regions, and then predicts the bounding boxes and probabilities for each region[1].

The first step is to gather and label the data. We collected 1900 images from the robot, of which nearly 700 contain pedestrians or moving vehicles. We then used LabelImg[2], a graphical image annotation tool, to label the images in YOLO format, which are shown in Figure 12. In order to improve the robustness of our models, we generated new images by cropping, shearing, zooming and adding noises to the images. As a result, our database is three times larger than the original one, and the images are resized to 416×416.

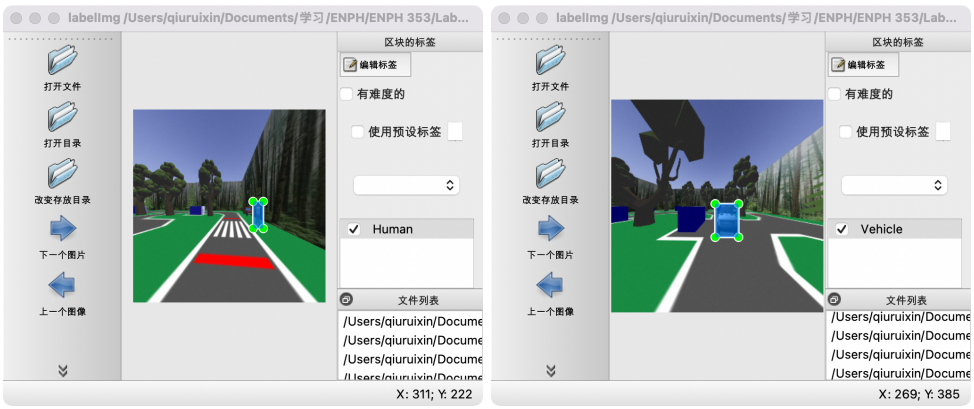

Figure 12. Labeling of a pedestrian (left) and moving vehicle (right) using LabelImg

The second step is to construct a pipeline to train the model. By following a tutorial online made by DepthAI[3], our team built up a pipeline to train the model. We used 85% of the images for training and 15% for validation. We then initialized our model by specifying the class number as 2: "Human" and "Vehicle". The number of filters can be calculated by $(N+5)\times 3$ where N is the number of classes. We used the default model structure from the YOLOv3-Tiny framework. The learning rate is set to be 0.001 and the maximum batch size is set to be 4000. After two hours of training for 3100 iterations, the model achieved a mean average precision (mAP@0.5) of 99.47%.

We tested the object detection model manually in different conditions in the Gazebo world to detect pedestrians and moving vehicles shown in Figure 13. The result showed that it is reliable in prediction but slow in speed. The detection is not synchronized and has a lagging of 2 seconds. However, we can solve this problem by making it detect the objects every four frames. Then it can publish the location and the name of the objects.

[1] YOLO: Real-Time Object Detection, https://pjreddie.com/darknet/yolo/
[2] LabelImg, https://github.com/tzutalin/labelImg
[3] DepthAI Tutorial: Training a Tiny YOLOv3 Object Detector with Your Own Data, https://colab.research.google.com/github/luxonis/depthai-ml-training/blob/master/colab-notebooks/Easy_TinyYolov3_Object_Detector_Training_on_Custom_Data.ipynb

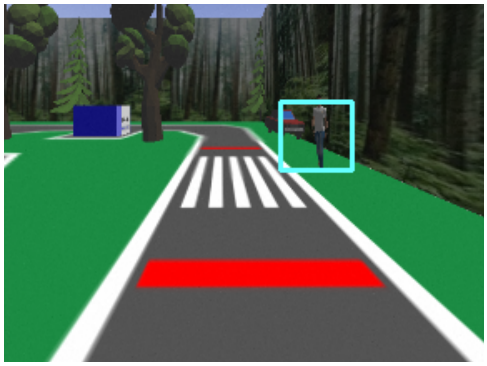

Figure 13. Detected pedestrian is bounded by blue box

Driving System

The driving system is the brain of the robot. It processes the images and the data to determine the driving direction and speed. It has a PID controller implemented inside that works with the plate recognition system to publish velocity commands to the robot. The driving system also works with the object detection system to avoid collision with pedestrians or moving vehicles.

1. PID Controller

The strategy for moving the robot is rather straightforward. The robot first extracts the road from the image stream. By determining the distance between the center of mass of the road and the center of the image, it adjusts its speed and direction accordingly. However, the major problem we met is that the robot is not stable when there is a crossing. It does not know whether to turn left or right when there is a crossing.

In order to improve the performance of the robot, we divided the Gazebo world into 10 sections as is shown in Figure 14. Each section contains no more than one corner. In addition, each parking ID or the redline in front of the zebra line indicates the start of the respective sections. Therefore, the robot can know which section it is in by scanning the parking IDs or the redlines from the plate recognition system. Then our team designed a route that can cover most areas using the shortest distance, as is shown in Figure 14. Knowing the section it is in, the robot can focus on the direction that it should follow. For example, when the robot is in Section 3, it can mask 10% of the left part of the image from the camera, as a result of which, the center of road is not affected by the crossing. Therefore, the robot can still move forward when the crossing appears. This approach also enables the robot to get to the inner loop after completing the outer loop. When the robot notices that it is in Section 1, it masks 10% of the right part of the image so that it would have a higher weight turning left than moving forward. After it reaches Section 7, it masks 15% of the left part of the image to keep the robot from leaving the inner loop.

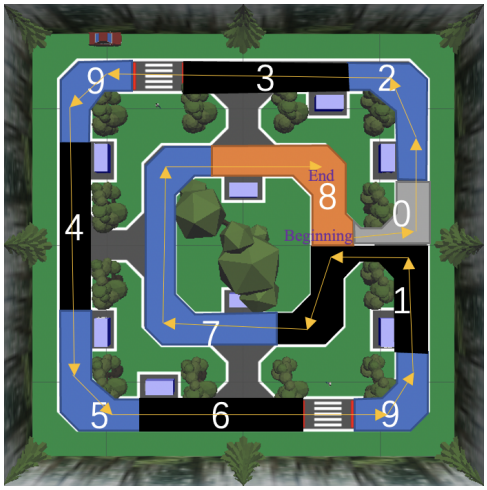

Figure 14. The route of

After determining the strategy, we set the parameters of the PID controller through many experiments. We also let the robot run autonomously to collect license plate numbers for further training of the CNN model. However, the speed of the robot should be further determined when building the collision avoidance system.

2. Avoiding collisions

Since safety is the top priority, collisions with pedestrians and moving vehicles are strictly prohibited. Therefore, the driving system also works with the object detection system to prevent the robot from hitting them or being hit by them.

Because pedestrians cross the road at the zebra line, the robot stops at the red line in front of the zebra line and detects the location of the pedestrians. The robot waits until the pedestrians' location changes to another side. For example, when the pedestrian is at the left side of the road, the location of the pedestrian is less than 150. The robot will wait for the pedestrian until the location of the pedestrian is larger than 150. In order to prevent the robot from being hit by the pedestrians when it is passing the zebra line, the robot must spend less time crossing the zebra line than the waiting time of the pedestrians. After repeated experiments, we finally determined that the minimum velocity should be 0.19 in the outer loop.

After the robot enters the inner loop, avoiding collision with the moving vehicle is the major concern. The robot starts detecting the location of the moving vehicle when it enters Section 1. When the robot detects the moving vehicle, it immediately stops for 6 simulation seconds until the moving vehicle is not in its vision. However, there is also a great chance that the moving vehicle will hit our robot from the back. Although there is no score deduction in this matter, we will lose control of our robot as a result. Therefore, we need to make the robot move faster in the inner loop. After checking the code for the moving vehicle, we found that the speed of the

vehicle has a random uniform distribution from 0.3 to 0.6. And after several tests, we found that the speed to ensure the safety of our robot is 0.45, which also enables the robot to scan the plate accurately.

Final Result

Before the competition, we tested the robot for several nights and made small changes to the code to ensure the best performance of the robot in the competition. We conducted more than 30 simulations before the competition, and the robot got the full score for 27 simulations. Therefore, the probability for the full score in the simulations is about 90%.

In the final competition, our robot was the only one that received a full score (57 points) within 74 simulation seconds out of six teams. Therefore, our team won first place in the competition unsurprisingly.

Conclusion

In this project, we successfully built a robot that can move around the campus, upload the license plates and identify parking IDs without breaking the traffic rules. We used computer vision to locate the license plates. Then we used convolution neural networks to predict the license plates. In addition, we employed YOLOv3-Tiny in the object detection system to detect pedestrians and moving vehicles, and adopted PID to control the motion of the robot. All the three systems work well with each other to control the robot. The final result showed that all the three systems perform perfectly.

However, we also encountered some problems and difficulties in the project. For example, we first tried to use SIFT to detect the letter “P” on the vehicle in order to locate the license plate, but the detection was inaccurate and slow, so we had to give up this plan. Then, we attempted to use reinforcement learning to control the movement of the robot, but due to time constraints, we had to use a PID controller instead.

During the process of our testing, the main problem we met is the incorrect recognition of the license plate numbers. After further investigation, we found that the main problem lies in the inaccuracy of extracting the license plate numbers, because the number extraction from the license plate numbers is highly dependent on the white board extracted. Therefore, if we had more time, we can solve the problem by using connected region extraction labeling and analysis to extract the characters accordingly. In addition, we can also try to record the probabilities of the predictions given to a single plate and choose the highest probability for the recognition.

To conclude, despite that there are still some problems that need to be fixed further, our robot can fully satisfy the requirement when in standard conditions.