上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：Parking Spot Number Detection Using Fisheye Camera

学生姓名：朱元立，尹郭馨，夏伟豪，张圣安，王译萱

学生学号：517370910017,517370910043,517370910061,517370910153,517370910214

专　　业：电子与计算机工程

指导教师：吴继刚

学院(系)：密西根学院

上海交通大学
毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：朱元立 夏伟豪 王泽宣
张圣安 尹郭馨

日期：2021年8月10日

上海交通大学
毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密□，在___年解密后适用本授权书。

本论文属于

不保密☑。

（请在以上方框内打“√”）

作者签名：朱元立 夏伟豪 王诗萱 张圣安 尹郭耀　　指导教师签名：吴纯洁

日期：2021年 8 月 10 日　　日期：2021年 8 月 10 日

使用鱼眼摄像头识别停车位号码

摘要

汽车司机往往容易忘记他们的停车位，特别是当停车场很大，或者停车完很长时间后。解决这一问题的关键挑战在于，在检测和识别停车位号码时，很难同时保证准确率和效率。在本项目中，我们提出并实现一个基于鱼眼摄影机的停车位号码识别检测系统。它遵循鱼眼畸变校准-停车位检测-投影变换-光学字符识别-应用程序演示的流程。首先图像校准可以消除鱼眼摄像头拍摄照片时产生的畸变。与此同时，我们结合了车位检测、形状变换等多个步骤，保证了停车位识别的准确性。停车位检测可以避免后续识别步骤被图中车牌号所干扰，而形状变换则可以返回一个俯视角下的车位图。最后，光学字符识别可以检测出车位号码，之后号码会被传输到用户端的应用上。在这个问题上，我们非常重视算法处理速度和识别准确率，因为我们希望提供一个可靠而且高效的识别。实时计算不仅保证了用户的快速响应，而且为语义地图等未来应用提供了可能。

关键词：鱼眼摄像头，鱼眼畸变校准，光学字符识别，停车位检测，视频处理。

PARKING SPOT NUMBER DETECTION USING FISHEYE CAMERA

ABSTRACT

It is sometimes hard for drivers to remember the parking slot they take. This is especially the case when the parking lot is large, or when the drivers leave for long. The key challenges to solve this problem lie in the difficulty of achieving both effectiveness and efficiency when detecting and recognizing parking spot numbers. In this project, we proposed and implemented a real-time parking spot number detection system based on the fisheye camera. It follows the pipeline of fisheye calibration −parking slot detection −projective transformation −optical character recognition (OCR) −APP demo. First, calibration will be applied to eliminate the distortion caused by fisheye camera. Meanwhile, our algorithm combines multiple steps, such as locating slots and transforming shapes, to ensure its accuracy on parking id recognition. Parking lot detection will be used to prevent the detection process from being interfered by the vehicle plate number, while transformation can return a frame in vertical view. Finally, OCR is used to extract spot number, which would be sent to the user-side APP. When addressing this problem, processing speed and detection accuracy are highly valued so that we can provide a reliable and efficient recognition. Real-time computing not only ensures quick responses for the users, but also brings about the possibility for future applications like the semantic map.

Key words: Fisheye camera, Fisheye calibration, Optical character recognition, Parking lot detection, Video processing.

Contents

Chapter 1 Introduction

When in a huge parking lot, it is hard for drivers to remember the final parking location. It would be a great help if we can use the assistant of the camera on the vehicle to help users detect and remember the parking spot number. In light of the excellent field of view (FOV) of fisheye cameras among all other types of on-vehicle cameras, the fisheye camera becomes the best candidate to manipulate this task. In this way, when driving the car in the parking lot, we could collect real-time videos through the fisheye cameras. We then process the videos and get the spot ID in real-time, and then send the results to the phone APP, as shown in the Illustration 1–1. Through the real-time spot ID recognition, it is also meaningful for car companies to better build the auto parking assistant system. When users are in a completely unfamiliar parking lot, through the on-vehicle camera they can easily build the cartographic semantics, from which if users would like to park their vehicle inside a specific parking slot, giving the spot ID would trigger the auto parking assistant to help park in that place.

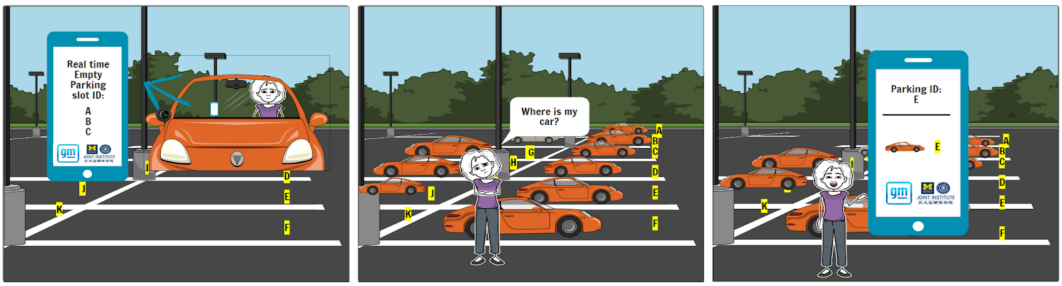


Illustration 1–1 Program background

Our project design is novel based on the following several points. First, there is little dataset about fisheye images on parking spot ID. More specifically, although there have been a lot of fisheye datasets, digits datasets, letter datasets, real-time object detection datasets and parking lot detection datasets, none of them really integrate all the aspects together so a real-time alphanumeric dataset shot by the fisheye camera is unavailable, which makes it hard to train accurate machine learning models to solve this problem. Second, although the submodules of this project are relatively mature, none of the modules have been fully combined together for any similar projects. For example, there are

available OpenCV API and MATLAB API to deal with the calibration of the fisheye images. There are already some papers discussing OCR[1] and parking lot detection[2]. However, none of them did the job of recognizing the parking spot ID in real-time.

However, if we think of this problem separately, we can adapt the technologies from those different aspects and integrate them with appropriate adjustment and improvement to work for our project. Therefore, we will not be limited by the fact that we do not have enough appropriate datasets that contain fisheye images on parking slots. Firstly, we could calibrate the fisheye images into those that have normal digits and letters with the help of MATLAB. Next, we could adopt OCR methods to extract information from the normal images. One thing to notice is that when applying normal OCR, it is really easy to extract the license plate number of the car out. Therefore, we could get help from the parking lot detection to specifically picked out the characters on the parking spot to identify. With proper transformation after the parking slot detection, the characters shot by cameras would be sketched into normal characters to better apply OCR on. In this way, the parking spot ID is recognized and recorded in real-time and we will pass all the results to the user's mobile APP.

Chapter 2 Design Specification

2.1 User Requirements

Our target customers are drivers of cars with standard fisheye camera set and troubled by parking. The customers have some main problems in terms of parking. Looking for their car in a large parking lot is a big challenge, especially when they do not remember your parking slot ID and are in some busy places, such as a shopping mall in the city center.

To address the above problem, we list several most important requirements for our product. The first one are functional requirements and the last two are non-functional requirements.

1. The product should be able to send the accurate parking slot ID to the customer's phone in real-time.
2. The product should rely on already equipped hardware instead of installing new hardware. Otherwise, it would have a higher cost.
3. Our product should be able to function well independent of the parking slot styles and surroundings. The surroundings and styles of different parking lots can be quite different from each other and our product should be able to work in as many parking lots as possible.

2.2 Engineering Specifications

For the three customer requirements, we need to translate them into engineering specification to set measurement of the success of our project. The overall engineering specification is listed in Table 2–1. The following subsections are going to explain how these specifications come from and why we choose these values.

Table 2–1 Engineering specifications and their corresponding values

Engineering Specification	Value
Processing Speed	0.1 - 1 second per frame
Parking Slot ID Recognition Accuracy	>90%
Parking Slot Missing Rate	<10%
Field of View	170°
ID Number Length	1-10
Recognized Characters	a-z,0-9,A-Z

2.2.1 Performance

For the first one functional requirements, the performance is important both in terms of processing speed and accuracy. Since we have to process the video input from the fisheye camera on the car in real-time, we have to be able to detect the parking slot ID fast enough so that we would not miss any single parking slot.

We assume the speed of a car in the parking lot will not exceed 20km/h, which means 2.78m/s. The width of most parking slots is around 2.7m, which means our processing speed should be strictly faster than 1 second per frame. Otherwise, we may miss some parking slots. However, there will be typically more than one slot each frame and in order to increase accuracy and decrease missing rate, it would be better to process each parking slot multiple times from different angles. Therefore, ideally, we would like to achieve 100ms per frame. Those values remain the same even we shrink our car and parking slot proportionally.

One of the most important requirements is accuracy. We have to recognize parking slot ID accurately. In addition, since this is based on the result that we have already detected a parking slot in a frame successfully, the missing rate of parking slot detection should be low. After discussion with our sponsor, the criteria are set that the accuracy should be higher than 90% in order to give a good customer experience. Since one frame could contain multiple parking slots, which means during one move, the car could capture the same parking slot multiple times. If there exists one time that we could successfully

recognize the correct spot ID, we will consider it a success for this specific parking slot.

2.2.2 Hardware Restriction

To fulfill the second user requirement, our product should be able to process the video input from fixed hardware. In our case, the most important hardware is the camera. Our sponsor provided the parameter of the camera as shown in Table 2–2 and Illustration 2–1. The restriction of installation height and angle specifies how our algorithm is going to observe the parking slot. Therefore, the existed algorithm may not be applicable in this problem or we may do some prepossessing. In addition, from Table 2–1, we can see that the field of view of the fisheye camera is very broad, and our product should be able to process the video input with such a large range. Therefore, our engineering specification includes the field of view because we want to fully use the hardware and be able to detect any parking slot and its ID number in the whole view instead of only part of the view.

Table 2–2 The parameter of installed camera on the car

Parameter	Value
Horizontal Field of View (HFOV)	170-200°
Vertical Field of View (VFOV)	80-150°
Pixel	1MP-2.5MP
Installation Height	1.00-1.20m
Installation Angle Downwards	45 ± 10 °
Installation Angle Backwards	0 - 5 °

Illustration 2–1 Camera installed position and angle towards the ground

2.2.3 Reliability

For the third customer requirement, we need to make sure our product is reliable, which means our product should be able to work in different parking lots. To simulate different environments and test our product, we need to quantify the criteria.

1. Length and position of the slot ID. Different parking lots may have different length of ID number. Our product should be able to detect ID number with length 1-10 characters, which covers most parking lot styles. We assume the position of the ID is in the area of the parking slot.
2. Characters included in the parking slot ID. For most parking lot ID, characters include 0-9, a-z, A-Z. Therefore, our product should be able to recognize those characters.

2.3 Quality Function Deployment

	Weight (1-10)	Processing rate per image	ID Recognition Accuracy	Spot ID Character Type	Camera Field of View	Max Slot ID length	Max Missing Rate	*Parking Slot Detection*	*Optical Character Recognition*
								Benchmark	
Recording accurate spot ID in real time	8	9	9				7	3	5
No extra hardware	4				9			3	3
Widely Applicable	5	9	9	8		8		4	3
Measurement Unit		ms	%		°		%		
Target Value		10	90	a-z, 0-9, A-Z	170	10	10		
Total		18	18	8	9	8	7		
Normalized		27%	27%	12%	13%	12%	9%		
Parking Slot Detection		5	1	1	5	1	5		
Optical Character Recognition		5	3	4	3	4	3		

Competitive benchmarks
5 = satisfies perfectly
4 = satisfies "mostly"
3 = satisfies "somewhat"
2 = satisfies "slightly"
1 = doesn't satisfy at all

Illustration 2–2 Quality Function Deployment of our project

Illustration 2–2 shows our final QFD. It is worth noting that one of the most important engineering specification is ID recognition accuracy. It makes sense because we need to provide customer the accurate ID number for them to find their car. A wrong number will lead the customer to a wrong place, which is definitely a serious problem.

For two related technologies mentioned in the previous section, parking slot detection and optical character recognition, neither of them can fulfill all customer requirements. For parking slot detection, it cannot recognize ID number at all. For optical character recognition, it is also not widely applicable in different surroundings. For example, it cannot distinguish parking slot ID from licence plate and it cannot recognize ID number that is not horizontal in the picture.

We can also see that most engineering specifications are unrelated or negatively related. Processing speed is negatively related with accuracy, With more time processing the image, we can get more accurate result. In addition, larger field of view will potentially

increase missing rate and processing speed. For those contradiction, we need to consider balance between them.

Chapter 3 Concept Generation and Selection

There are three main tasks we need to accomplish in this project. Firstly, we need to build a testing environment including car with fisheye camera and simulated parking lot. Secondly, we need to design an algorithm to extract spot number from the image. Finally, we should develop a phone application so that users can see the result. The reason we test our project in the simulated environment instead of real environment is that we do not have enough budget to get a real car with a high-quality fisheye camera and rent a server with high performance to run the algorithm.

3.1 Hardware

Although it is best to drive a real car with camera installed in a real parking lot, it is not suitable for our experiment for following reasons:

1. A real car costs much, either for buying or renting one. It is unnecessary as we just need to move the position of the camera.
2. We cannot change the character pattern in a real parking lot. We need to test different patterns for validation.
3. The parking spots in our university doesn't have IDs. We need to go to the shopping mall to test our camera.

Moving a small car in a simulated parking lot with proper scale is more suitbale for us. Therefore, Raspberry Pi, as a complete and widely-used embedded platform, becomes the best choice for us. Various kinds of fisheye camera modules are available for this platform, too.

A common size of a simulated car with Raspberry Pi board on it is about an A4 paper, which makes the build of simulated parking lot easier. A parking spot unit can be designed and printed on an A4 paper, while combining several A4 papers will give us a simulated parking lot. We can easily adjust the size as well as the character pattern for the parking spot. In addition, A4 paper is very portable.

3.2 Parking Spot Number Detection

The main obstacle in this project is the algorithm part, which is how can we extract parking slot ID from a photo shot by the fisheye camera. In order to recognize spot ID on the ground, one technique that we obviously need to use is optical character recognition (OCR), which can convert image of typed or written texts into machine-encoded text. This maturing technique is widely used in the area such as document scanning. However, we have to do some preprocess for our photo before applying optical character recognition. There are many problems we need to solve before existed OCR model can directly work without retraining the model. We abandoned the retraining idea because there are no existed database for out project and it requires quite a lot data to train an OCR model. Following are two main obstacles we need to solve before applying existed OCR model.

1. Since it is took by the fisheye camera with oblique view, the photo has serious distortion.
2. There may be multiple texts in the image besides parking spot number. We have to only keep the characters in the parking spot region.

3.2.1 Fisheye Calibration

As the data set of fisheye image (video) with characters is hard to find, we need to do calibration first so that we can use normal data set with undistort images. Many works have been done for fisheye model:[3] suggests four projection models: Stereographic $r = 2f\tan(\theta/2)$, Equidistance $r = f\theta$, Equisolid angle $r = 2f\sin(\theta/2)$ and Sine law $r = f\sin(\theta)$;[4] estimates and optimizes the intrinsic parameters with equidistance model, without using any existing frameworks;[5] uses Para-Catadioptric-Like Camera and applies simulation and real-data verification;[6] works with the calibration toolBox for MATLAB, which could applies to any central omnidirectional (includibng fisheye) cameras with basic calibration algorithms; OpenCV library provides a calibration and undistortion feature. Most of the done work are followed by two steps: (1) estimate

intrinsic parameters of the camera, (2) apply parameters to calibrate images and do some correction.

Among those calibration methods, both OpenCV and MATLAB fit our project.

The advantage for OpenCV is listed below:

1. Fisheye camera model on OpenCV has well-designed interface as well as carefully-written documents, as shown in Illustration 3–1.

Illustration 3–1 Interface

2. OpenCV provides Python based algorithm which can be easily run on small platform, such as Raspberry Pi on the car.
3. OpenCV is widely used for good calibration quality on edge even on fisheye camera with large view angle (> 170). Function undistortImage() can automatically fix the fuzzy edge and return a clear image as shown in Illustration 3–2.

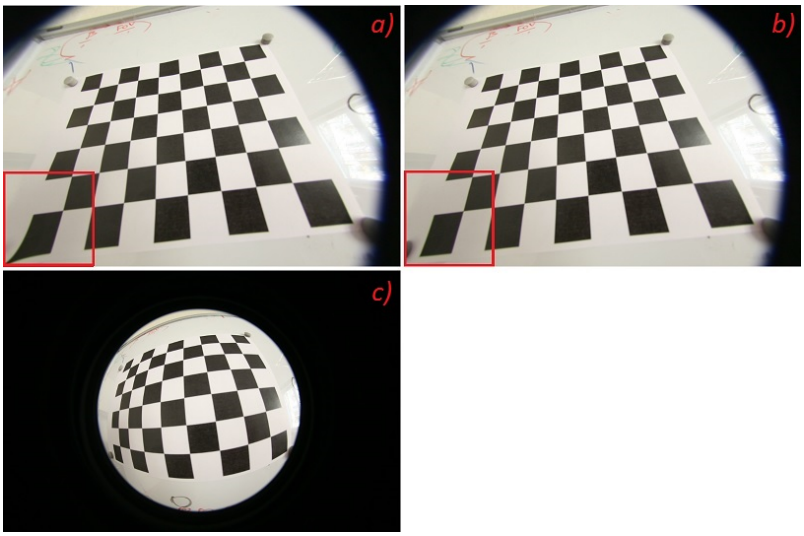


Illustration 3–2 Calibration result

The model used by OpenCV is described as following:

Let P be a point in 3D of coordinates X in the world reference frame (stored in the matrix X) The coordinate vector of P in the camera reference frame is:

$$Xc = RX + T, \tag{3–1}$$

where R is the rotation matrix corresponding to the rotation vector om: R = rodrigues(om); call x, y and z the 3 coordinates of Xc:

$$x = Xc_1, \tag{3–2}$$

$$y = Xc_2, \tag{3–3}$$

$$z = Xc_3. \tag{3–4}$$

The pinhole projection coordinates of P is $[a; b]$ where

$$a = x/z, b = y/z, \tag{3–5}$$

$$r^2 = a^2 + b^2, \tag{3–6}$$

$$\theta = a\tan(r). \tag{3–7}$$

Fisheye distortion:

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8). \tag{3–8}$$

The distorted point coordinates are $[x'; y']$ where

$$x' = (\theta_d/r)a, \tag{3–9}$$

$$y' = (\theta_d/r)b. \tag{3–10}$$

Finally, conversion into pixel coordinates: The final pixel coordinates vector $[u; v]$ where:

$$u = f_x(x' + \alpha y') + c_x, \tag{3–11}$$

$$v = f_y y' + c_y. \tag{3–12}$$

where the parameters are prepared as intrinsic matrix

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3–13}$$

and fisheye distort matrix

$$\begin{bmatrix} k_1 & k_2 & k_3 & k_4 \end{bmatrix}. \tag{3–14}$$

However, OpenCV doesn't match MATLAB well. The advantage of using MATLAB is listed below:

1. MATLAB provides an APP for easy calibration estimation. The UI for APP Camera Calibrator is shown in Illustration 3–3.

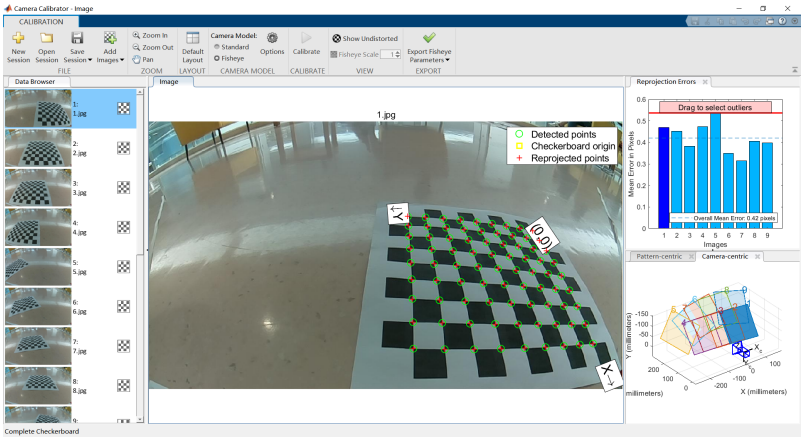


Illustration 3–3 Camera Calibrator UI

   Simply adding photos of checkerboard, we can easily get fisheye parameters. This APP also provides functions like Reprojection Errors and Camera-centric view, which could help us improve the quality of calibration.
2. It is easier for MATLAB to connect the fisheye camera on the car and take photos with the help of MATLAB Support Package for Raspberry Pi Hardware. Compared to taking photos on the car and transfer via SSH, directly taking photos from MATLAB saves a lot of time.
3. As OpenCV doesn't match MATLAB well, if we choose MATLAB as working platform, OpenCV becomes less attractive.

However, MATLAB also has shortage. The view angle will be changed by adjusting the size of photo, and we cannot control it. We may lose some resolution to get a larger view angle.

(1) Parking Slot Detection

Given the calibrated fisheye video frames, we would like to firstly extract parking spot position from them. This is because we want to prevent the optical character recognition from being disturbed by other objects in the video such as the car plates.

There are some existed methods to detect the parking slot. Currently, the most popular and accurate method is deep learning method. For example, DMPR-PS is a deep learning model of parking-slot detection that performs extremely well on their dataset ps2.0[7], which achieves 99.42% precision and 99.37% recall. It first divides image into S*S grid and extracts a S*S*N feature map from I using CNN. The architecture of the neural network with S=16, N=6 is shown in Table 3–1.

However, we tried to use different models including DMPR in our case. Because of the different angle of views, their model cannot predict any result. It requires retraining for the weight of the model. As mentioned above, we do not have dataset to utilize deep learning model and collecting thousands of data and labeling all corner points of the slots is too time consuming. Therefore, we decide to use traditional computer vision method for this step.

Table 3–1 Architecture of the neural network used in DMPR[7] with S=16, N=6

Layer Type	Filters	Size / Stride	Output Size (C×H×W)	
conv+norm+relu	32	3×3 / 1	32×512×512	
conv+norm+relu	64	4×4 / 2	64×256×256	
conv+norm+relu	32	1×1 / 1	32×256×256	×1
conv+norm+relu	64	3×3 / 1	64×256×256	
conv+norm+relu	128	4×4 / 2	128×128×128	
conv+norm+relu	64	1×1 / 1	64×128×128	×1
conv+norm+relu	128	3×3 / 1	128×128×128	
conv+norm+relu	256	4×4 / 2	256×64×64	
conv+norm+relu	128	1×1 / 1	128×64×64	×2
conv+norm+relu	256	3×3 / 1	256×64×64	
conv+norm+relu	512	4×4 / 2	512×32×32	
conv+norm+relu	256	1×1 / 1	256×32×32	×2
conv+norm+relu	512	3×3 / 1	512×32×32	
conv+norm+relu	1024	4×4 / 2	1024×16×16	
conv+norm+relu	512	1×1 / 1	512×16×16	×3
conv+norm+relu	1024	3×3 / 1	1024×16×16	
conv+activation	6	1×1 / 1	6×16×16	

Sebastia[2] provides a non-deep-learning method to extract parking lot with traditional parking slot. It first using horizontal gradient to perform a coarse detection of the parking lot markings. Then, it estimates the lateral distance from the vehicle to the parking lot row. Finally, it has enough information to detect the parking slot as shown in 3–4. With these methods, it can reach an average precision of 0.84.

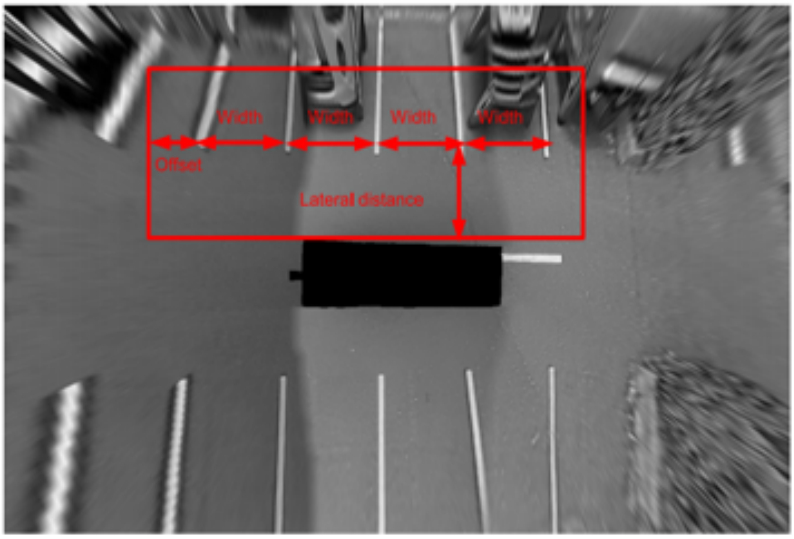


Illustration 3–4 Lateral detection[2]

Another non-deep-learning method is provided by[8]. The algorithm flow chart is shown in Illustration 3–5 It utilizes Hough transformation[9] to detect lines in the image. Lines detected can be very discrete in the image, it is important to connect discrete lines and delete trivial lines and group lines to get the result of parking lot detection. This paper also presents the idea of parking lot tracking. Since we are not dealing with static image but video, we can use context information to track the parking lot to increase the detection speed and accuracy. Currently, dealing with video is not in our plan and we are going to make it as our future work.

Inspired by those methods, we designed and implemented our own non-deep-learning parking slot detection algorithm and we would show the detail in the later section.

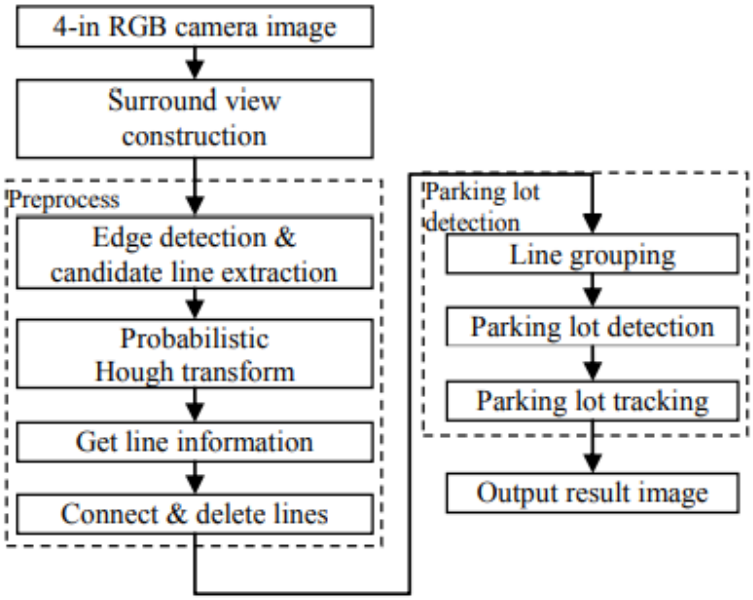


Illustration 3–5 The algorithm flow chart of parking lot detection described in[8]

3.2.2 Projective Transformation

Most of the concept shown below comes from the youtube video[10].

The step —projective transformation is that after calibration and parking spot detection, we would transform a perspective view of the parking slot into a orthographic view object. Illustration 3–6 shows an example in our project for the projective transformation.

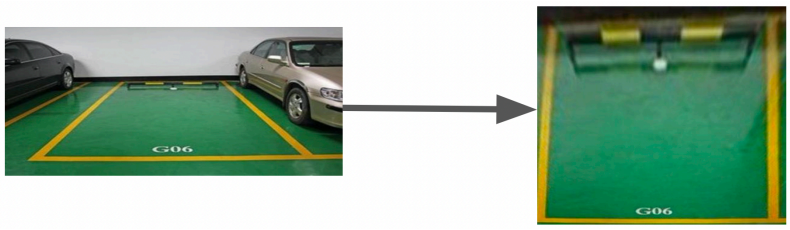


Illustration 3–6 Image projective transformation[10]

For an image $I(x, y)$, we can use a 2D linear transformation matrix T,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = T \begin{pmatrix} x \\ y \end{pmatrix}. \tag{3–15}$$

where matrix T is a combination of scale, rotation, shear and mirror, to ralize the projective transformation.

There are different kinds of 2D linear transformation methods available shown in the Illustration 3–7, as we need to rotate and project, we would choose the first group - perspective or homography.

Group	Matrix	Distortion	Properties
Perspective or Homography 8 DOF	$\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$		Maps lines to lines but parallelism may not be preserved
Affine 6 DOF	$\begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$		Collinearity and parallelism preserved
Similarity 4 DOF	$\begin{pmatrix} s_{11} & s_{12} & t_x \\ s_{21} & s_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$		Angles and ratio of lengths are preserved
Euclidean/ Isometries 3 DOF	$\begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$		Length and area preserved

Illustration 3–7 Different kinds of 2D linear transformation methods[10]

With homography, for the point x in image plane in the Illustration 3–8, we want to project it onto the planar surface, corresponding to X, with

$$\mathrm{x} = HX. \tag{3–16}$$

where H is a 3×3 matrix.

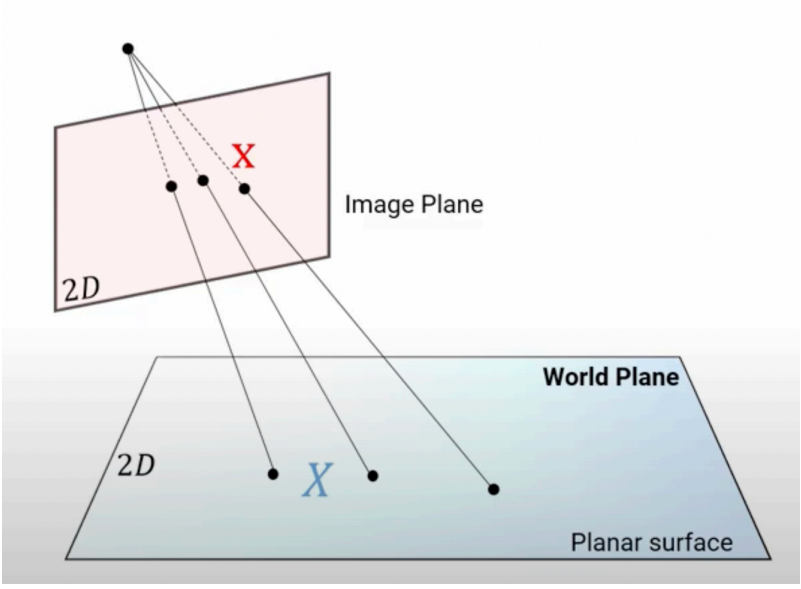


Illustration 3–8 Illustration for homography[10]

To implement homography, we would need at least 4 points to solve for H with matrix shown as Illustration 3–9:

Four points:

$(p_1, p_1'), (p_2, p_2'), (p_3, p_3'), (p_4, p_4')$

$$a_i = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x_i' & y_i x_i' & x_i' \\ 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y_i' & y_i y_i' & y_i' \end{bmatrix}$$

$$\mathrm{AH} = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 x_1' & y_1 x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 y_1' & y_1 y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2 x_2' & y_2 x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2 y_2' & y_2 y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3 x_3' & y_3 x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3 y_3' & y_3 y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4 x_4' & y_4 x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4 y_4' & y_4 y_4' & y_4' \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix} = 0$$

Illustration 3–9 Projective transformation formulas[10]

To sum up, we would write the equation as follows and solve the variable in it

$$\begin{pmatrix} x'/\lambda \\ y'/\lambda \\ \lambda \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3–17}$$

Actually for projective transformation, we hesitate on whether to use this step.If we use the projective transformation, then it would increase the processing time, and maybe let our project become more uncertainty, somehow, as we maybe change the outlook of the parking slot ID.If we do not use the projective transformation, it means our next step —Optical Character Recognition (OCR) may need to directly process the distorted ID, which would decrease the final accuracy. As the OCR we choose are quite sensitive to the overall structure of the character.

We compare these two methods according to the criterions, including accuracy, computing speed, interface, platform cost and computing cost. And we decide to use projective transformation, mainly due to accuracy.

3.2.3 Optical Character Recognition

Next, to extract parking spot number information from cropped fisheye video frames, we turn to Optical Character Recognition method. It can convert the character information in images to ASCII code. There are two ways to do this. The first ones are traditional methods. They require fewer computational resources, but they are less adaptive to text of different fonts and locations. One of the possible solutions are:

1. Firstly, we detect edges in video frames. To do this, we can apply a filter to perform derivative on each pixel to strengthen the edges. In this way, character information can be highlighted as shown in Illustration 3–10.

Illustration 3–10 Edge detection[1]

2. Secondly, we will adopt segmentation. For example in Illustration 3–11, we can generate the vertical projection profile of a video frame, where the value change represents the boundaries between text and background. Using this profile, we can indicate the position of characters and thus do segmentation in the original frames.

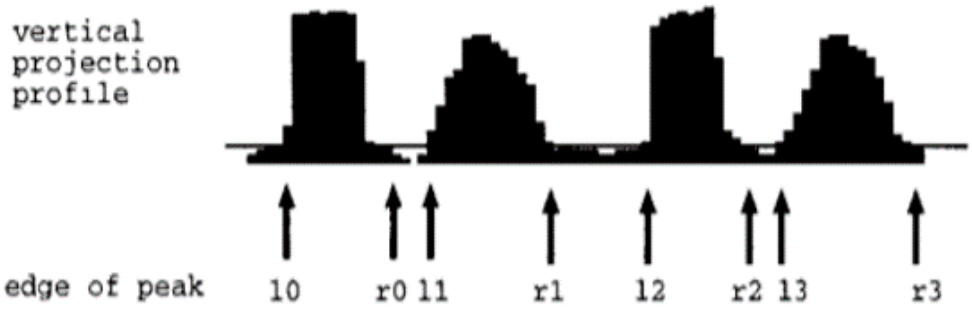


Illustration 3–11 Segmentation[1]

3. Finally, we can make recognition. Till this stage, it reduces to a classification problem because we can assume that character is either a letter or number. Therefore, we can use Matrix Matching, Structural Analysis, or even a neural network here to do the prediction for each character.

The second ones are machine learning methods with neural network. After retrained with new dataset, they can work with characters of various perspectives and styles. Since the parking spot frames do not have a uniform mode, we would turn to this method in our project. Its general steps are described below:

Firstly, a region proposal network is adopted to find position of the text. That is done by using a small sliding window in the convolutional feature maps, and then the network structure in Illustration 3–12 can judge which proposal contain texts and decide on which fine-scale text proposals to output[11].

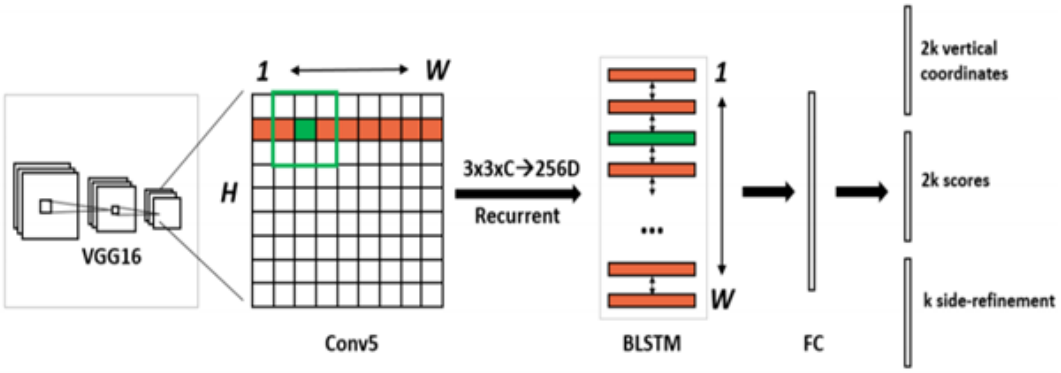


Illustration 3–12 Region proposal network[11]

Secondly, OCR uses a recognition network to give machine-encoded text. In[12], this is done with a convolutional recurrent neural network as shown in Illustration 3–13, where convolutional layers output feature sequences; recurrent layers make per-frame predictions based on those sequences; and transcription layers translate prediction results into labels[13].

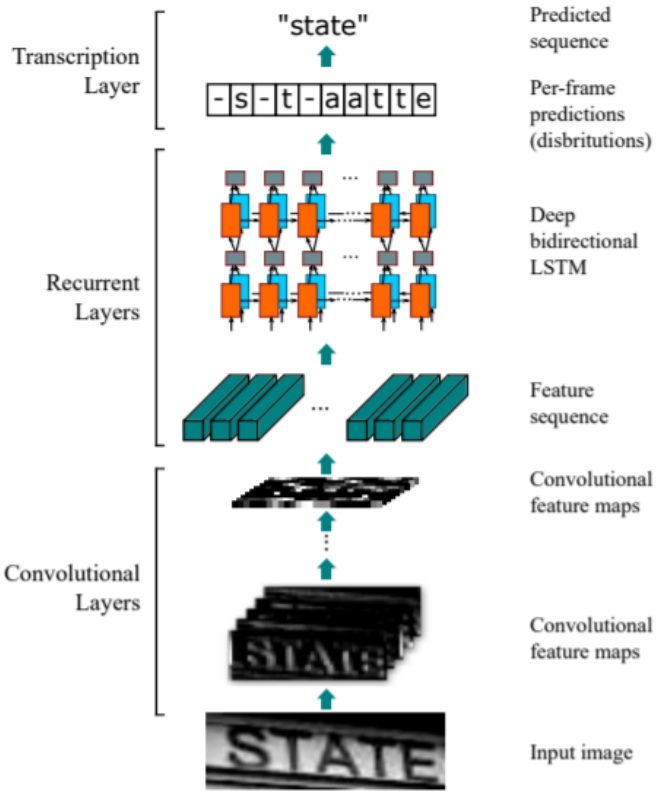


Illustration 3–13 Convolutional recurrent neural network[13]

3.2.4 Overall Algorithm Flow Chart

From what we discussed above, we designed the algorithm flow chart as shown in Illustration 3–14. The figure shows each step of how we processed the photo and the ideal output of each step. The photo shot by fisheye camera is first processed by calibration to eliminate distortion. Then we detect the four corners of the parking spot and we have enough information to do projective transformation and convert the spot into a rectangle with its orignial ratio, Finally, we can simply using existed OCR model to extract parking number from it.

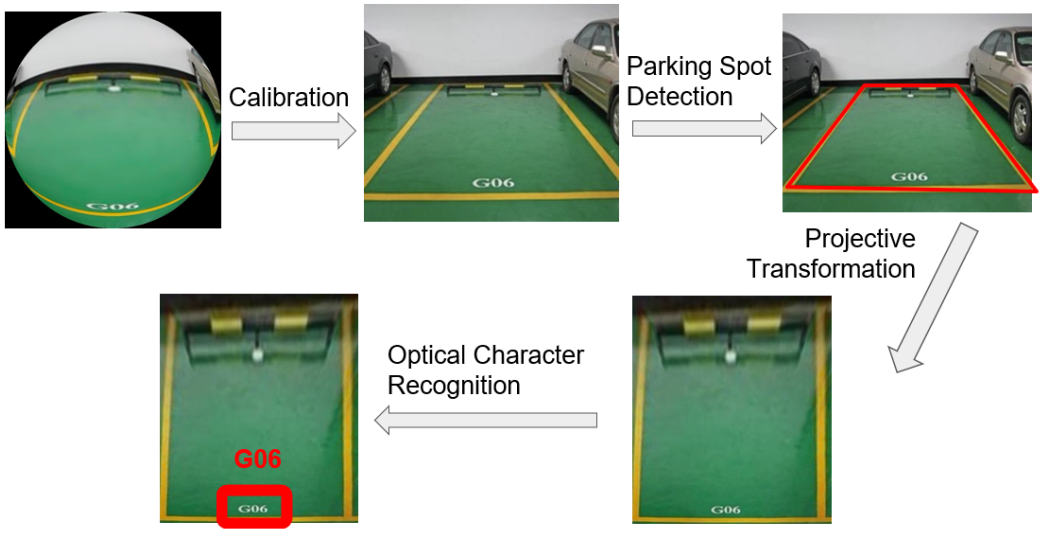


Illustration 3–14 The algorithm flow chart to detect spot number from image shot by a fisheye camera

3.3 Mobile App

Finally, we would like to show the resulted parking spot ID and the corresponding pictures on the mobile app. Since the development of Android mobile app, which is on Android Studio, can be done on all three major platforms, i.e. Windows, MacOS, Linux, while the development of IOS mobile app, which is on XCode, can only be done on the MacOS system, and the emulator on Android Studio is more powerful than that on XCode, we choose to implement an Android mobile app to show our results.

Chapter 4 Implementation Description

To implement our design, we follow the pipeline as shown in Illustration 4–1. First, MATLAB take photo in our simulate parking lot from the fisheye camera module on Raspberry Pi car. Next, calibration will be applied to the photo to remove distortion. Then, computer will run the algorithm to detect parking spot and spot number in the image. Finally, the result will be sent to phone and user could see the result in real-time.

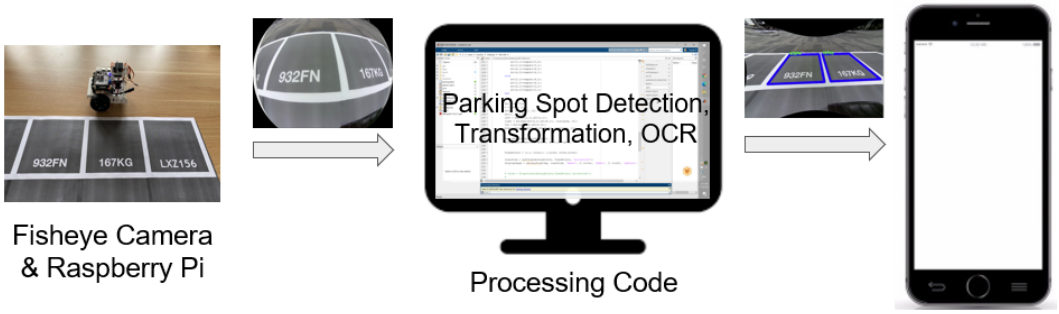


Illustration 4–1 The pipeline of detection system from taking photo to output results

4.1 Hardware Preparation

4.1.1 Raspberry Pi Car

We buy the car and Raspberry Pi platform from Hiwonder as shown in Illustration 4–2, from Taobao. It has following features which are good for our project.

Illustration 4–2 Car from Hiwonder

1. It has motors and wheels so that the car can move freely in the parking lot.
2. It provides a battery so that it can get rid of power lines. We can easily move it in the parking lot.
3. It provides a mobile app, so we can control the car remotely and get the camera video in real-time for testing.

4.1.2 Speed Reduction

The car still needs to be adjusted before it is used. The speed of the car is too fast.In order to simulate the real world case, the speed of the car should proportionally shrink as the parking spot width. We focus on width here because in most cases, parking spots are lined up as shown in Illustration 3–4. The width of the real parking spot is 2.7m and the width of our parking spot is 17cm as shown in 4–8. Therefore the size decreases $2.7m/17cm = 16$ times and the speed should also decrease 16 times. The speed of a car in the parking lot is usually 5-15km/h, which is 140-420cm/s so the speed of our car should be in a range 8.75-26.25cm/s. However, the rotate speed of our motor is 120-240RPM, which is 2-4 rounds/s, and the radius of the wheel is 3.5cm. It means the speed of our car is 44-88cm/s and is far above the real case. Therefore, we need to

reduce the speed of the car.

For the reason that we want to test our product in different speed, we used Lego module to build speed reduction gear so that we can easily change the combination of the gears to achieve different speed of the car. In the Illustration 4–3, it shows a combination of gears that transfer speed from a 8 tooth gear to a 48 tooth gear, which reduces speed by 6 times. In our above example, the speed of our car falls into 7.33-14.67 cm/s, which fits our requirement.

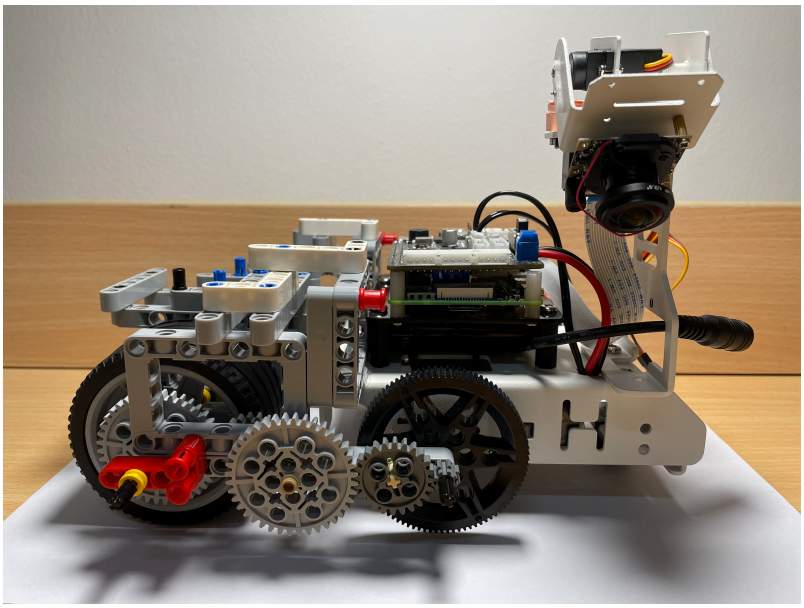


Illustration 4–3 Using Lego gear combinations to reduce the speed of the car

4.1.3 Fisheye Camera Module

We buy the fisheye camera module according to the engineering specifications, as shown in Illustration 4–4, from 1688.

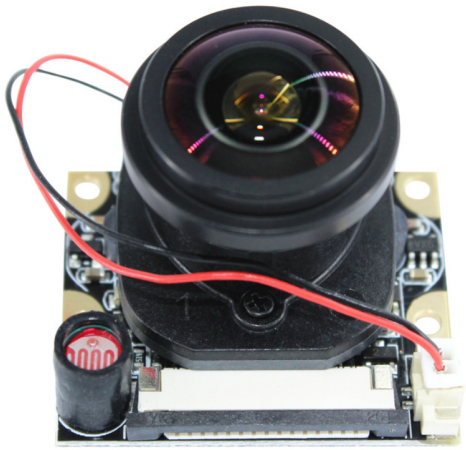


Illustration 4–4 Fisheye camera module

The parameters for this module is listed below in Table 4–1.

Table 4–1 Parameters for camera module

Parameter	Value
View angle	175°
Max resolution	2592 × 1944
Max frame rate	30 fps
Interface	CSI

4.1.4 Setup

(1) Combination of car and camera

The combination of car and camera module is shown in Illustration 4–5. The CSI line of fisheye camera module should be inserted into the CSI input of Raspberry Pi board. The scale of our simulated system is approximately 1:10.

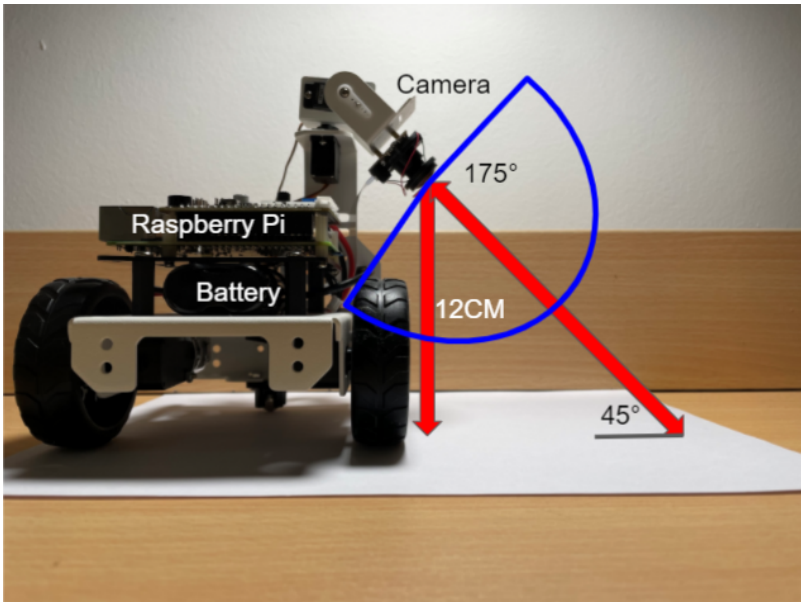


Illustration 4–5 Car parameters

(2) Initialization of Camera

We need to enable camera on Raspberry Pi OS. Execute:

```
sudo raspi-config
```

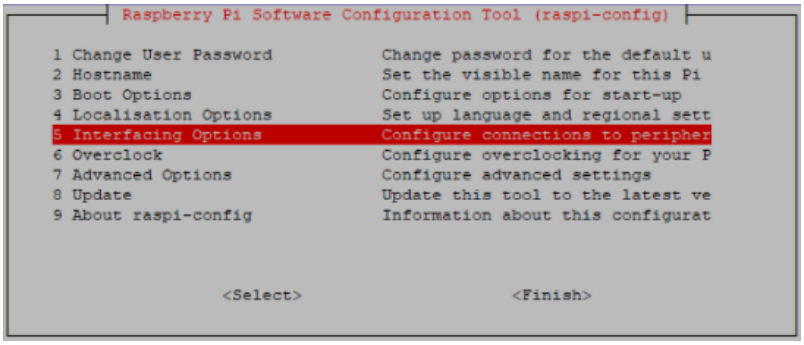


Illustration 4–6 Window after sudo raspi-config

Select Interfacing Options

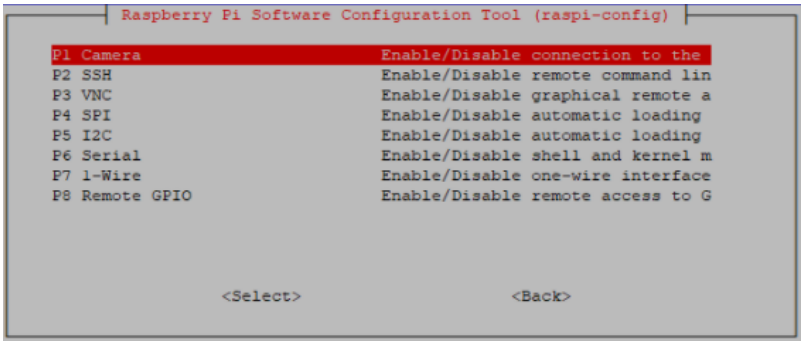


Illustration 4–7 Window after select interfacing options

Select Camera, then select Yes. To test the validation of camera, execute:

```
vcgencmd get_camera
```

The correct result should be:

```
supported = 1 detected = 1
```

(3) Initialization of MATLAB Connection

We then initialize MATLAB connection on the car. First search and install MATLAB Support Package for Raspberry Pi Hardware. Then execute:

```
mypi = raspi('ip address','username','password');
```

on MATLAB. MATLAB will install necessary components on Raspbian system. After successful installation, a raspi object will be created.

Then we can create a cameraboard object, execute:

```
mycam = cameraboard(
  mypi,'Resolution','1280x720','Brightness',50,'Quality',100);
```

To test the camera, execute:

```
imshow(snapshot(mycam));
```

(4) Simulated parking spot

We print simulated parking spot on A4 papers. A sample size is shown in Illustration 4–8.

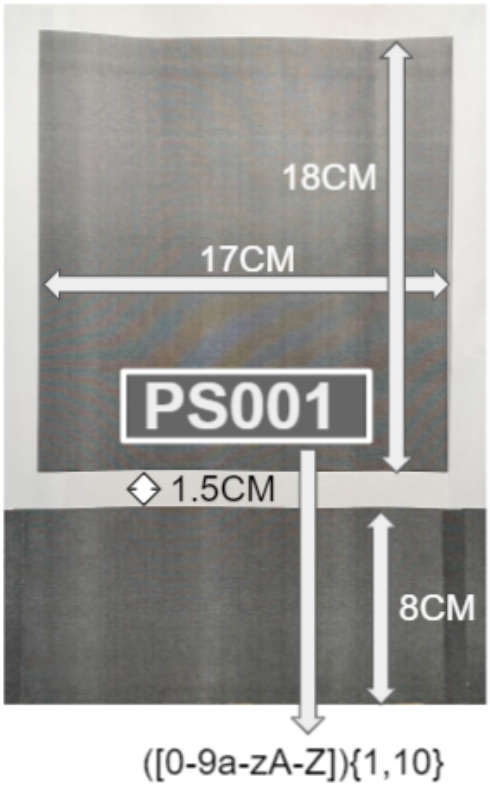


Illustration 4–8 Basic parking spot unit

And we place spot units with consecutive IDs together to construct a parking lot as shown in Illustration 4–9. The printed parking lot is shown in Illustration 4–10.

Illustration 4–9 Parking lot

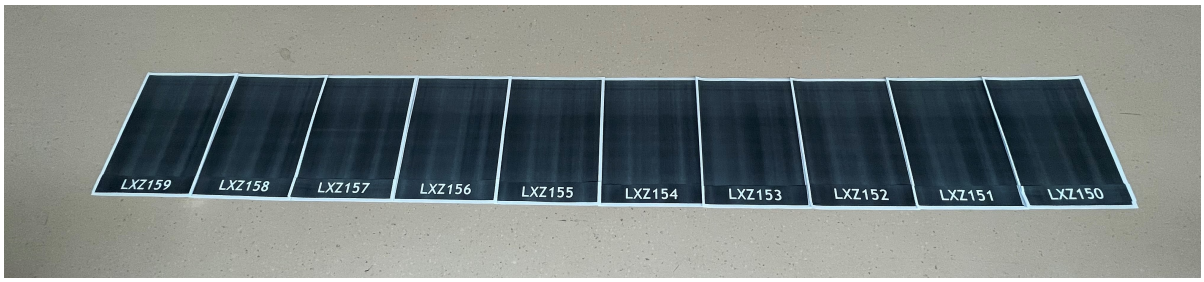


Illustration 4–10 Printed parking lot

(5) Remote Control

We write a program with pygame to remotely control the movement of car. It uses Up, Down, Left and Right key on keyboard of the computer to drive the car at any direction and speed.

4.2 Algorithm

With the image taken taken Raspberry Pi car, we implement an algorithm to calibrate it and extract parking spot number from the image as shown in Illustration 3–14. The detailed implementation of each step is shown as following.

4.2.1 Fisheye Calibration

(1) Parameters Preparation

We use Single Camera Calibrator App of MATLAB to prepare the calibration parameters. We first take photos of a printed checkerboard as shown in Illustration A–1, with each box $18mm \times 18mm$. The environment is shown in Illustration 4–12. Some sample photos are shown in Illustration 4–13.

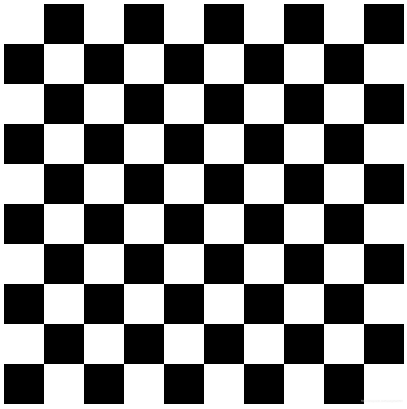


Illustration 4–11 Sample checkerboard

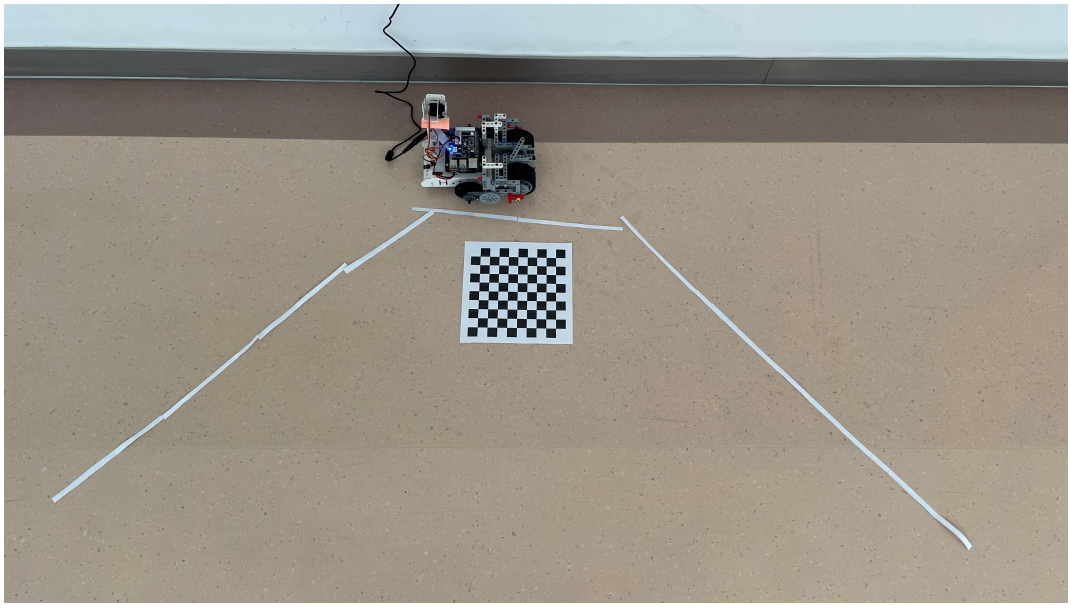
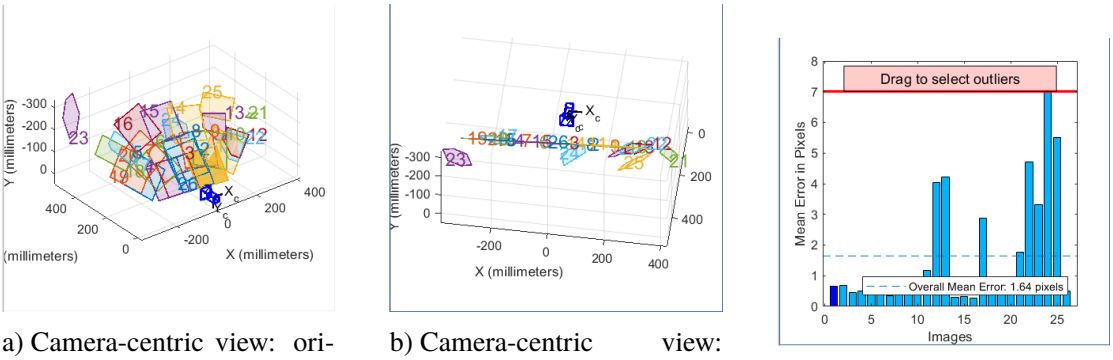


Illustration 4–12 Taking photos

Illustration 4–13 Sample photos

Add all the photos to Single Camera Calibrator App, select ”Camera Model: Fisheye” and press ”Calibrate”. After calculation, we get the first set of parameters. As shown in Illustration 4–14, the projection error is not very good.



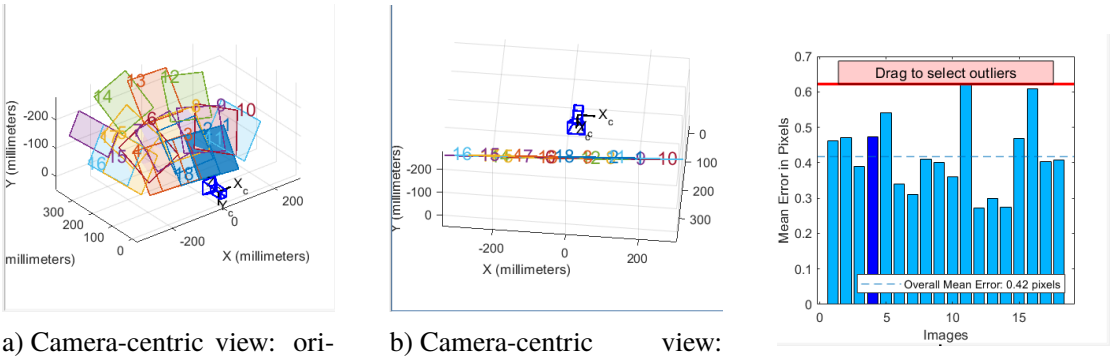
a) Camera-centric view: origin

b) Camera-centric view: plane

c) Projection errors

Illustration 4–14 Examine panel before optimization

We remove some bad photos from calibration set and re-calculate parameters. The optimized result is shown in Illustration 4–15.



a) Camera-centric view: origin

b) Camera-centric view: plane

c) Projection errors

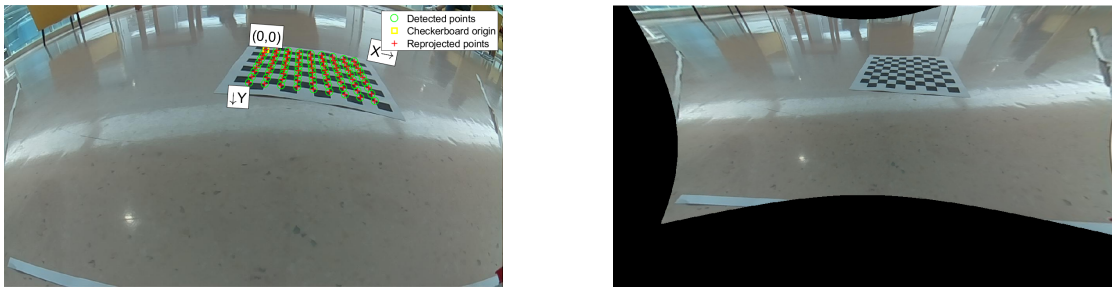
Illustration 4–15 Examine panel after optimization

Here are some sample result for calibration, shown in Illustration 4–16 and 4–17.

a) Before calibration

b) After calibration

Illustration 4–16 Sample result 1



a) Before calibration

b) After calibration

Illustration 4–17 Sample result 2

Then we press ”Export Fisheye Parameters” and save the parameters for later use.

(2) Applying Calibration

After we get the calibration parameters, we can apply calibration to the photos now. The function is:

```
undistortFisheyeImage(
   origin image, fisheyeParams.Intrinsics,
   'OutputView','same','ScaleFactor', 0.8);
```

4.2.2 Parking Slot Detection

After calibration, we can do parking lot detection now. As discussed in the previous section, in the real using case, existed deep learning model provides good accuracy and speed to do the job. However, we do not have enough data to retrain the model to fit in our simulated case. Therefore, for our demo purpose, we applied traditional computer vision technique to develop an algorithm that can achieve the purpose.

We implemented 2 methods. Method one has high accuracy and works well under different environment. However, it tries to find the whole parking spot which means it is based on the assumption that the parking spot line is continuous. However, it is not always the perfect case. It happens a lot in the real life that some part of the parking spot is covered or worn that the line is not continuous. To solve this problem, we also implement method 2, which can handle the case that line is not continuous but has lower resistance to the distractions in the environment.

(1) Method 1

An example of the image input in our real using case is shown in Illustration 4–18. The first step is to convert it into binary image. For an grey image in computer, it is essentially a matrix where each element is an 8-bit integer and black is presented as 0 while white is presented as 255. We first find the maximum and minimum value in the graph and

then set all pixels larger than $(max - min) * 0.7 + min$ to white and remaining pixels are black. The value 0.7 is chosen by testing our system under different environments and light conditions. The above can be done by `im2bw` function in MATLAB. The result after conversion from Illustration 4–18 is shown in Illustration 4–19.

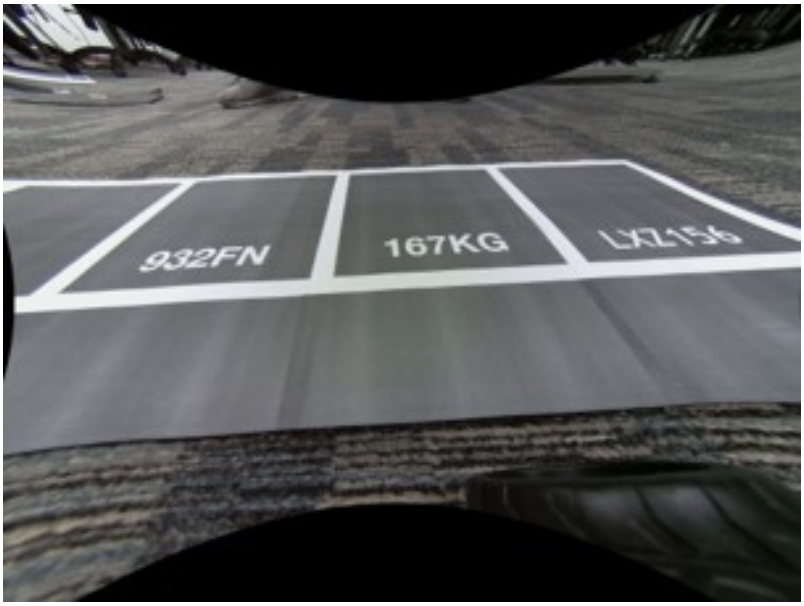


Illustration 4–18 An example of the image input in real using case

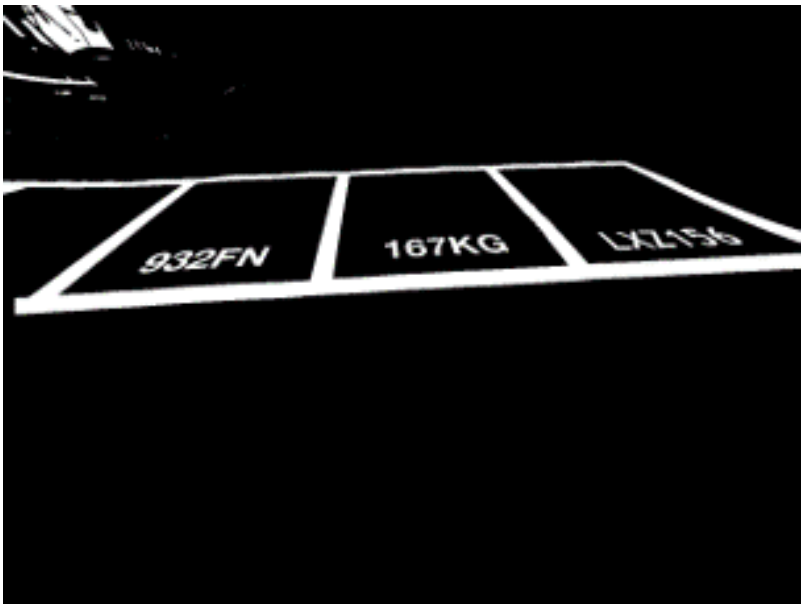


Illustration 4–19 The binary image converted from Illustration 4–18

Next, we extract parking slot line from the binary image. We simply count the white region with largest area as the parking slot line. This method is very fast and accurate

when there are not many distractions. There are two limitations for this method. First, when there are larger white areas than parking slot lines, the return would be wrong. In addition, the line has to be continuous, otherwise, the result would only return part of the parking slot lines. The output of this step is shown in Illustration 4–20.

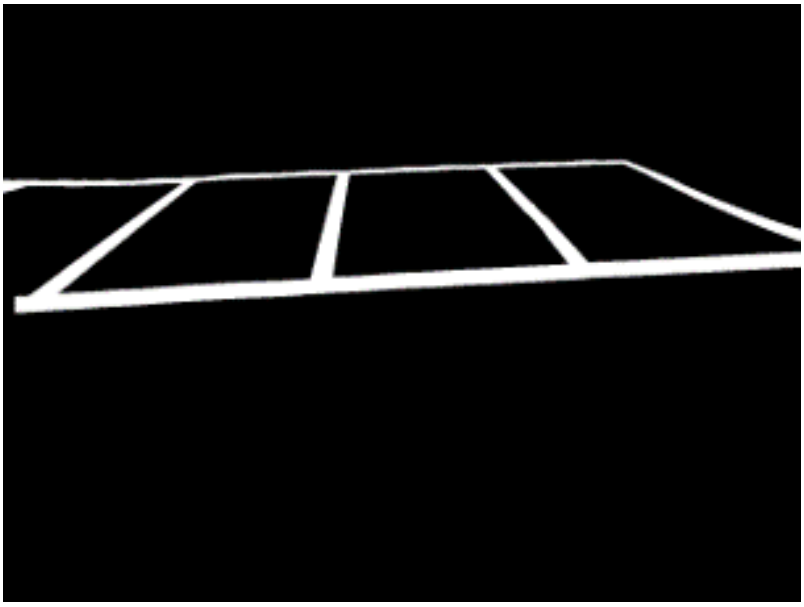


Illustration 4–20 The output of eliminating small white region in Illustration 4–19

Finally, we separate each parking slot and for each parking slot we do corner detection. The separation is straightforward, especially with the assumption that white spot line is continuous. We find each black region that is enclosed by the white region. Corner detection is done by MATLAB `corner` function. It is an implementation of Harris corner detector[14]. The main idea is to find the point where gradient of both horizontal and vertical line is large. The result is shown in Illustration 4–21. With final 4 fixed points, we have enough information for protective transformation.

Illustration 4–21 The result after applying parking slot separation and corner detection to Illustration 4–20

(2) Method 2

We can see in method 1, because we extract largest white region, it cannot handle the circumstances that some parts of the parking spot line are covered such as Illustration 4–22. No single spot has whole parking spot lines so none of them will be detected.

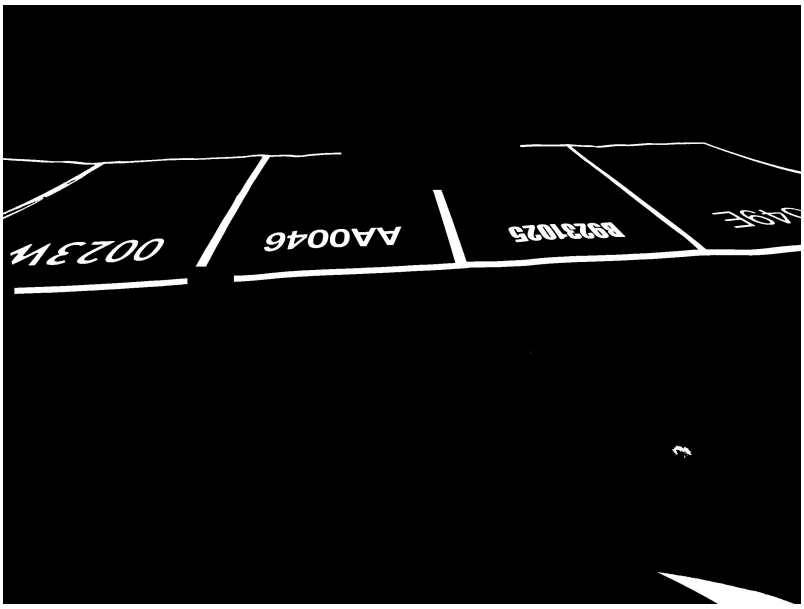


Illustration 4–22 A sample binary image that some parts in the graph are covered

To solve this problem, we implemented a simplified version of Illustration 3–5. The basic idea is to detect line segments and then group them to form whole lines. In this way, even some parts are covered and lines are not continuous, we can still find the line by grouping. For the first step, we use edge detection and the result is shown in Illustration 4–23. Then we use Hough transformation to detect line segments and the result is shown in Illustration 4–24.

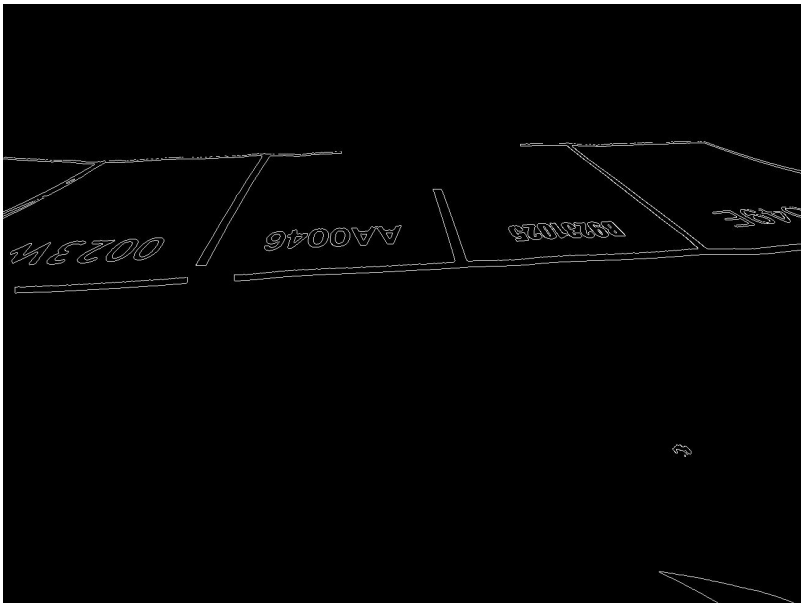


Illustration 4–23 The result of applying edge detection to Illustration 4–22

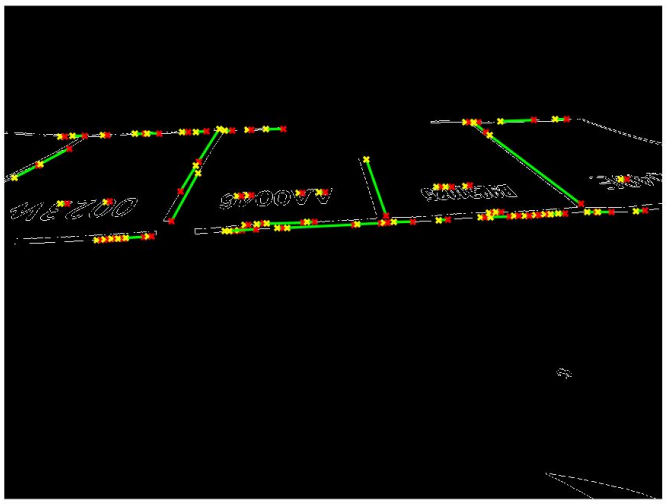


Illustration 4–24 The result of applying Hough transformation to Illustration 4–23

Then we applied line grouping according to angles and distance. After some experiments, we group two line segments A_1A_2, B_1B_2 together if the angle between A_1A_2 and B_1B_2 is less than 5 degree and the distance between A_1 to B_1B_2 and A_2 to B_1B_2 is less than 20 pixels. After grouping, we need to choose which lines are the ones we want. Overall, it is obviously that we need to detect two horizontal lines and multiple vertical lines to represent parking spots. Therefore, for horizontal lines, we choose two groups

whose sum of the length of all the line segments is largest and the mean angles between the lines segments and horizontal is less than 5 degree. We can see in Illustration 4–24 that there are three groups of the horizontal line segments and third line is caused by the characters. However, the total length of the line segments are not as long as the other two, so the result will be correct. For vertical lines, since we have already confirmed the horizontal lines, we first eliminate distractions outside two horizontal lines. Then we set a threshold T that if the total length of the line segments in a group is greater than T, then it is a valid vertical line that is part of the parking spot. For threshold T, we choose it as one third of the distance between two horizontal lines. In this example, we find four groups that fit our requirement as vertical line.

Finally, for each line segment group we confirmed that are parts of the parking spots, we apply linear fitting to them to find the final line. For each line segment, we sample a point every 20 pixels. In this way, a longer line segment will have heavier weight. Then, we apply linear fitting to all the sample points in a group and get the final result as shown in Illustration 4–25. This step can be done by using MATLAB **polyfit** function. Then, we simply find the cross point between horizontal lines and vertical lines to get the exact four points location of each parking spot.

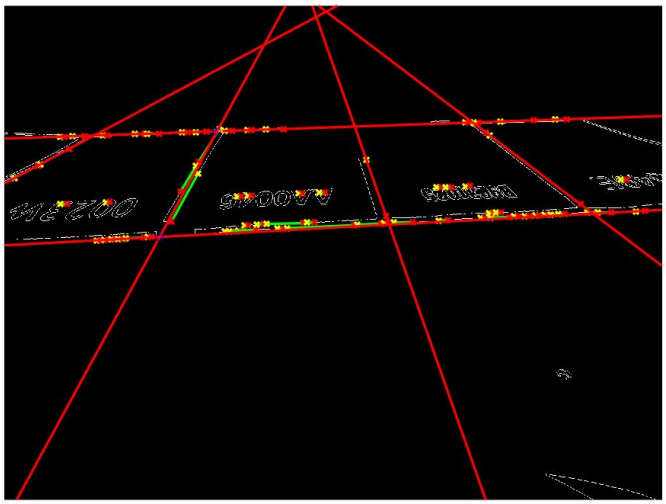


Illustration 4–25 Applying linear fitting to 4–24 and the final parking spot detection result by using method 2

4.2.3 Projective Transformation

To do projective transformation, we first need to get the corresponding points in the new image. We first select the type or parameters in advance, which is actually setting the coordinates of the corresponding points (the transformed quadrilateral size). For example, at this time we want to recognize the parking slot, so we set the size ratio to be 10:11, and the corrected shape of the parking slot can basically meet our requirements.

After getting original four points in the image and set their corresponding points after projective transformation in the new image, we can get the tranformation matrix shown in Illustration 3–7 with MATLAB function **cp2tform**.

Finally, we use the function **imtransformation** in MATLAB to implement the projective transformation and crop the image to only keep the parking spot region as shown in Illustration 4–27.

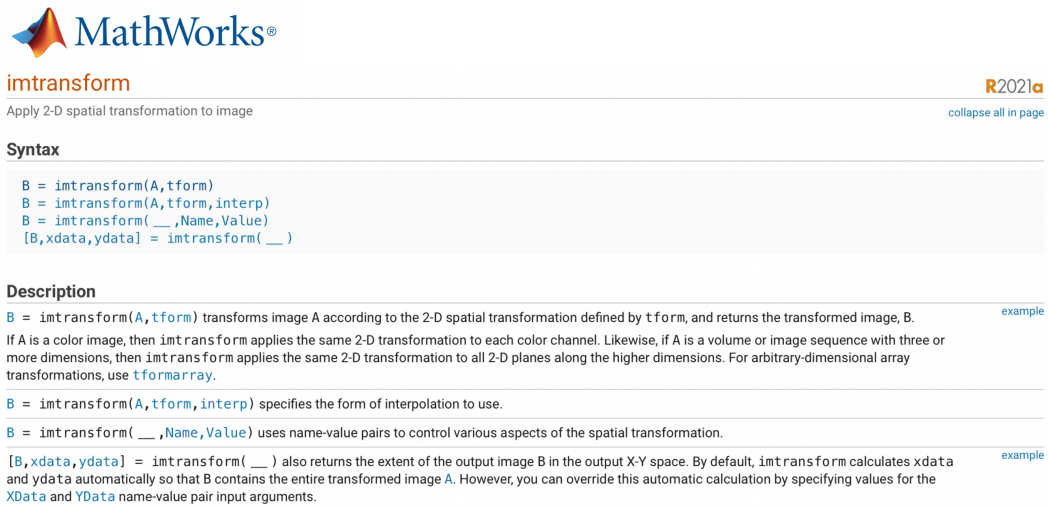


Illustration 4–26 Image transformation in MATLAB

4.2.4 Optical Character Recognition

Given the transformed image as shown in Illustration 4–27, we can then apply OCR on it. Since we do not have a large enough set of data to develop and train a machine learning model, we choose to deploy existing models. This is possible because of our

previous steps in the pipeline. Since we have already done parking slot detection and transformation, the image should now be able to fit the pattern of the massive data that are used to train those existing models.

Illustration 4–27 Transformed image

We research on two existing OCR models. The first one is[12] implemented with Pytorch. The text detection part of this model is based on a Connectionist Text Proposal Network, which detects text proposals directly in the convolutional feature maps of images[11]. Its recognition part is based on a Convolutional Recurrent Neural Network, which works well in lexicon-free text recognition tasks using integration of feature extraction, sequence modeling and transcription[13]. Combining these two networks, this method can provide prediction result together with a confidence score as shown in Illustration 4–28. It features a high accuracy when it comes to long and complex texts. Also, it can address Chinese characters, which makes the method more applicable in real life. However, we find that it takes about 2 second to process one frame on average (Intel Core i7-9750H CPU @ 2.60GHz). Since this project emphasizes real-time feedback, and we set 0.1s – 1s as target processing time, we do not take this method.

Illustration 4–28 Pytorch-implemented OCR result

The second is MATLAB built-in OCR function. It contains pretrained data files from Tesseract Open-Source OCR. According to MATLAB official document, its OCR function works best when there is a uniform background, and the whole format is like a document. Although this is not the case when it comes to the id on parking slot, we simulate such a suitable environment with previous steps of parking slot detection and transformation. The most significant point is the processing time. In our experiments, MATLAB OCR takes about 0.3 second to process one frame, which is much faster than previous pytorch-based OCR. Since our engineering specifications require both high accuracy and low latency, we must also take computing time into consideration. And therefore, we finally choose this method. A sample result is shown in Illustration 4–29.

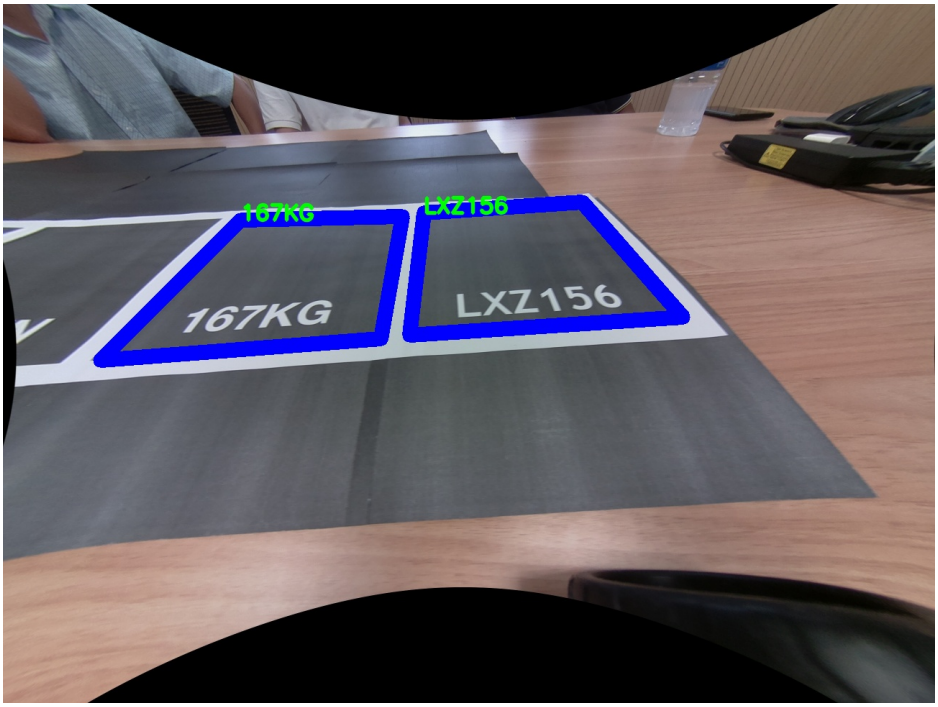


Illustration 4–29 MATLAB OCR result

4.3 Mobile App

After we extract the information using the optical character recognition, we would like to show the spot ID and the corresponding pictures on the mobile app. For this communication, we set up a back-end server to store the information in the database and establish a web framework to post and get the information from the IP address.

4.3.1 Database Management System

Firstly, we set up a database. The high reliability, security and scalability makes the postgresql one of the best candidates for our database management system. We create a database to store the parking spot ID, the posted timestamp, the link to the original image after calibration, and the link to the new image with parking slot and recognized ID highlighted. Illustration 4–30 below shows the items and their types in the postgresql database. The time section will automatically be filled in the current timestamp when posted all other information. And when images are posted alongside with the spot ID,

the images will not be stored in the postgresql database but in the back-end file system. The filename of the image file is turned into a URL, and the URL is then stored in the postgresql database alongside the spot ID. Table 4–2 shows a sample record in our database.

```
 Column   |           Type           | Collation | Nullable |      Default
----------+--------------------------+-----------+----------+--------------------
 id       | character varying(255)   |           |          |
 time     | timestamp with time zone |           |          | CURRENT_TIMESTAMP
 imageurl | text                     |           |          |
 newimage | text                     |           |          |
```

Illustration 4–30 Database structure

Table 4–2 Sample table contents

id	time	imageurl	newimage
E849 932FN	2021-07-12 20:18:51.467126+08	https://127.0.0.1/media/E849%20932FN%20 1626092331.4580514.jpeg	https://127.0.0.1/media/E849%20932FN%20 1626092331.4603899new.jpeg

4.3.2 Web Framework

We then set up a web framework to connect the back-end database and front-end application using Django with Nginx and Gunicorn following the official toturial on Digital Ocean[15]. The following steps are heavily based on this tutorial[15].

The first thing we need to do is to modify the Django project setting file to include our current IP address as one of the allowed hosts, so that this IP address could be used to connect to current Django server instance we configured. Next, we modify the Databases configuration to connect the Django server with the postgresql database we created previously.

Next, to start and stop the application server in a more robust way, we will use the Gunicorn *systemd* services and socket files, where the socket will be created when starting the servers and will be listening for any connections. Whenever there is a connection, the *systemd* will automatically begin the Gunicorn process to deal with the connection.

After we have configured the Gunicorn service, we need to then set up the Nginx to pass traffic to the Gunicorn. We created a new web site configuration file in Nginx $site-available$ directory to let the server listen and respond to our IP address.

Since Apple apps are requiring HTTPS traffic and Android apps's default setting is also the HTTPS traffic, we need to support HTTPS in our host server. To do this, we need to have a public key be signed by a certification authority. Because it requires us to have a qualified domain name where ours only have the IP address, we are going to be our own certification authority to generate and use our self-signed certificate in our project. After we have modified the $/etc/ssl/openssl.cnf$ file to provide information about our own IP address, we use openssl to create a self-signed key and certificate pair. Next, we modify the Nginx configuration file again to provide certificate and key information there. If new HTTP traffic comes in, we will automatically redirect to HTTPS using $return$ 301.

4.3.3 Backend API

To interact with the database, we need to implement ways to both insert records into the database and retrieve records from the database, which are corresponding to the HTTP POST request and the HTTP GET request respectively. This is done by implementing functions in the Django $app/views.py$ file.

For the HTTP GET function, we need to create a connection with the database and execute a $select$ SQL command to retrieve all the information from the database. Then we need to pass the information into a json object to return to the front end.

For the HTTP POST function, since we are handling image file from the request, we need to first store the image in our backend file system and then store the url to the file into the database. We do not really need to provide any response to the front end so returning an empty json object would be enough.

After we have implemented the above two functions, we need to modify the

routing/urls.py file to tell Django how to map each API to each function. Basically, we are adding the function names in the *views.py* file to the path in the *routing/urls.py* file.

4.3.4 Mobile App Development

Basically, we want to see a list of spot information on the mobile app. So the User Interface of the mobile app would be a list, where each item of the list will contain the spot ID, the timestamp, and the recognized image with parking slot and spot ID highlighted, like the one shown in Illustration 4–31.

Illustration 4–31 Information on the mobile app

Since the fish-eye camera on the car will collect images in real-time, the images will also be processed in real-time and continuously sent the images to the back-end database. Therefore, we also implemented an auto refresh feature so that the app will send HTTP GET request to the backend server every 0.5 second. In this way, it will fetch the latest results in real-time and show the most recent information on the top of the screen. The current 0.5 second refresh frequency is determined by the current frame speed and it can be changed faster when we increased the frame speed in the future improvement. If the user is too eager about the results and would like to see the most recent one whenever they want, we also implement a pull-down to refresh feature in our app. Since the whole user interface is a list ordered by the timestamp with the most recent one on the top, user could scroll down the page to see previous spot information. In this way, users could track their parking trajectory and know where they have parked their car.

4.4 Component Connection and Parallel Computing

Given the previous hardware and detection algorithms, we further implement programs that connect these components and enable the flow of the whole pipeline. In our first prototype, the Raspberry Pi car take photos automatically by its inner program and send those to the computer via SSH. To handle this, we adopted python watchdog library, which monitor Raspberry Pi' s target directory. Once new images appear there, watchdog fetches its path and call the detection program. After extracting text information, watchdog will draw a corresponding bounding box and send the results to the APP server. This workflow can be found in 4–32.

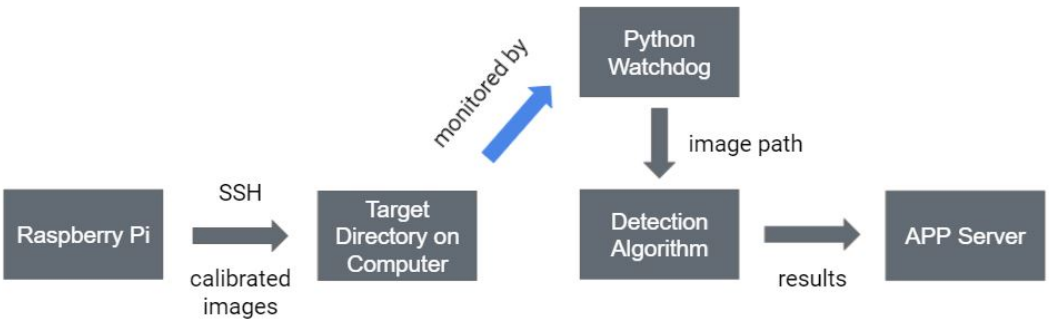


Illustration 4–32 Project workflow, initial version

In our optimized pipeline, the Raspberry Pi car takes photos under the command from MATLAB, instead of automatically with certain frequency. And as a result, we modified the previous connection programs. Now two monitoring programs are adopted. One of them is still based on python watchdog library, but it monitors the directory where MATLAB stores calibrated images. The other is in MATLAB based on directory content difference, which monitors the directory where python stores detection results. Once previous detection is completed, MATLAB will get notified and start taking next images. And we set up an image buffer on MATLAB side to ensure python and MATLAB run their programs concurrently. In this way, a parallel structure is set up, and parallel computing can further shorten the processing time. The optimized workflow can be found in 4–33.

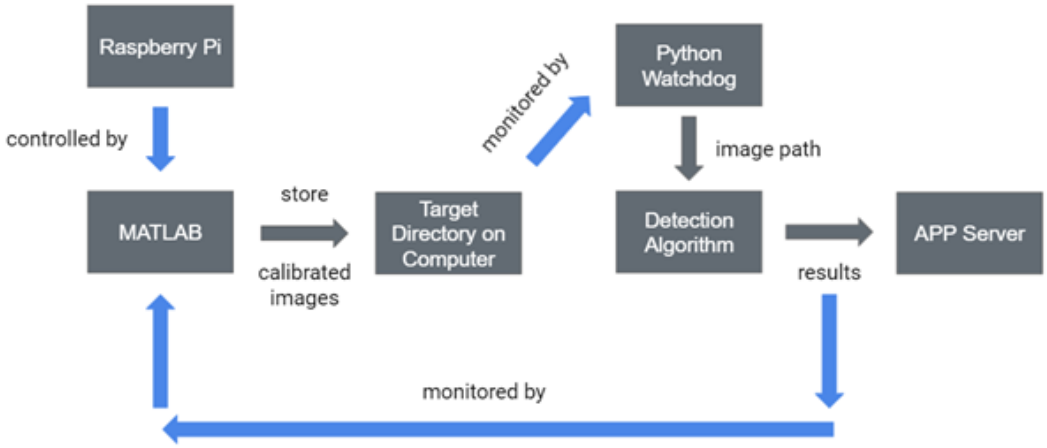


Illustration 4–33 Project workflow, optimized version

Chapter 5 Manufacturing Plan

Now we have implemented a prototype for our project. To make it a marketable product, companies can make the following improvements:

- Currently we are detecting parking slot with traditional computer vision methods. We locate the parking slot based on border and corner information in video frames, since we do not have a large dataset to do machine learning. When it comes to company scenario, massive dataset with labels can be used to train a neural network that can capture parking slot in frames. Also, instead of finding the whole slot, the company can search only for two corners closer to the car, and then do some vertical extension to construct a parking region. In this way, parking slot detection accuracy can be improved.
- In this project, we adopt OCR function provided by MATLAB to recognize parking slot ID. This function is based on Tesseract open-source trained model. However, due to the features of Tesseract training data, the function is more comfortable with document-formatted image. Therefore, the company can retrain the OCR function with parking slot number image, so that the model can be more adaptive to the corresponding size and style.
- For now, we use lots of wireless transmission in our prototype. In particular, the Raspberry Pi car uses SSH to transfer image data to the computer; and the computer also needs to send POST request to the APP server. That' s due to the limitations in computing resources and web techniques. However, in real usage, the company can set up the whole system inside the car. In this way, the fisheye camera can take wired connection to on-vehicle computers, which wrap up the whole processing steps and fetch data for the APP. Also, the processing can be accelerated depending on the computing power of the on-vehicle computers.
- In our demo, we only use one camera on the right side of the car as a demo of our algorithm and system. For a real car, there are four fisheye cameras on each side of the car. In that case, the algorithm can be easily adapted to four camera situation. In addition, the camera on the rear of the car can be used to record the final parking spot

driver actually parks his car.

- Finally, we currently calibrate the distortion caused by fisheye cameras before making any detection. That's because the open-source trained models can only handle text without distortion. And the company can instead train a new model with non-calibrated data from fisheye videos. In this way, the pipeline can be shortened, and the latency can get decreased.

Chapter 6 Validation Results

6.1 Validation Plan

Recall our engineering specification in Table 2–1, it is necessary to validate each one of the specification. Field of view has been fixed by our hardware and other parts have to be tested one by one. We simulated several parking lots, each with 5 spots using different parking spot number. Then, we drove our car along the three parking lots to collect data and process the result.

For processing speed, though it is important to measure the time of whole process, which is from the time when we take the photo to the time when the result is shown on the phone, it is not the most important factor to determine the fluency of our output. Since our goal is 0.1-1 second per frame, the most time-consuming step in the pipeline determines the speed of the output. Through several sets of tests, we record the average time for shooting pictures, fisheye calibration, recognizing spot number, and the whole pipeline time for each frame.

The result is going to be validated by both recognized ID accuracy and ID missing rate because when our car passes through one parking slot, it will take multiple photos, which means one parking spot will be shown in multiple frames. Missing rate is going to be measured as $1-R/T$, where R is the number of parking spot that is recognized in at least one frame during the driving of the car and T is the total parking spot we used to measure. It means if our algorithm fail to detect the parking spot in all frames, the spot is missed. Missing rate should be less than 10%. Another criterion is the number recognition accuracy rate, which is the number of character that we have successfully recognized over the total number of characters occurred in all of the frames. The accuracy rate should be higher than 90%.

In order to validate our product can deal with parking spot number with different ID length and characters, we printed quite a lot parking slots with different spot number styles as our simulation environment.

6.2 Experiment Results and Analysis

For the processing speed, we choose a sample of 100 frames. As we previous said, the ID recognition part is the most time-consuming step and it also relies on high-quality parking lot detection to ensure the occurrence of OCR. So the 100 frames we choose is made sure to have parking lot detected so that it cannot omit the OCR step. Notice that to increase the speed, we have implemented a multi-thread approach. Illustration 6–1 and 6–2 show the screenshots of the timing results from the two threads. Table 6–1 shows the average time for the tests and the last column is not a simple addition of all the former columns since this is a multi-thread algorithm. We can see that the average time for the whole pipeline is 0.3297, which meets our engineering specifications that we should have the process speed within 0.1 to 1 second per frame interval.

```
begin shot
0.23
end shot, begin calibrate
0.05
end calibrate and imwrite
```

Illustration 6–1 Thread for shooting pictures and calibration

```
Got event for file images/740.jpg
completed  0.12277507781982422
LXZ152
one frame time 0.38478517532348633
send to server time 0.06935644149780273
```

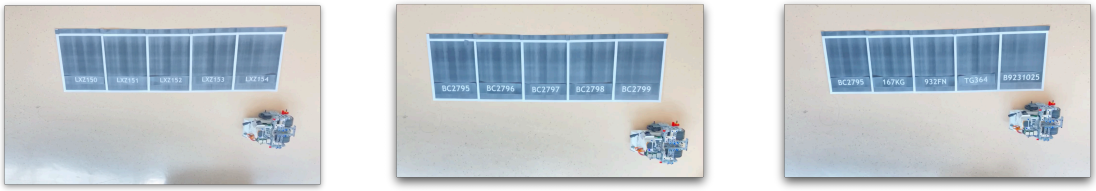
Illustration 6–2 Thread for recognizing spot ID and send them to servers

Table 6–1 Process speed results

	Thread 1		Thread 2		Whole pipeline
	Shoot	Calibration	ID recognition	Send to server	using multithreading
Average Time (s)	0.1887	0.0565	0.1536	0.0917	0.3297
Max Time(s)	0.50	0.28	0.3377	0.3485	0.6128

For the Accuracy, we choose 3 parking slots to run our Raspberry Pi car on, and the parking slots we choose are shown below in Illustration 6–3. We monitor our recognition

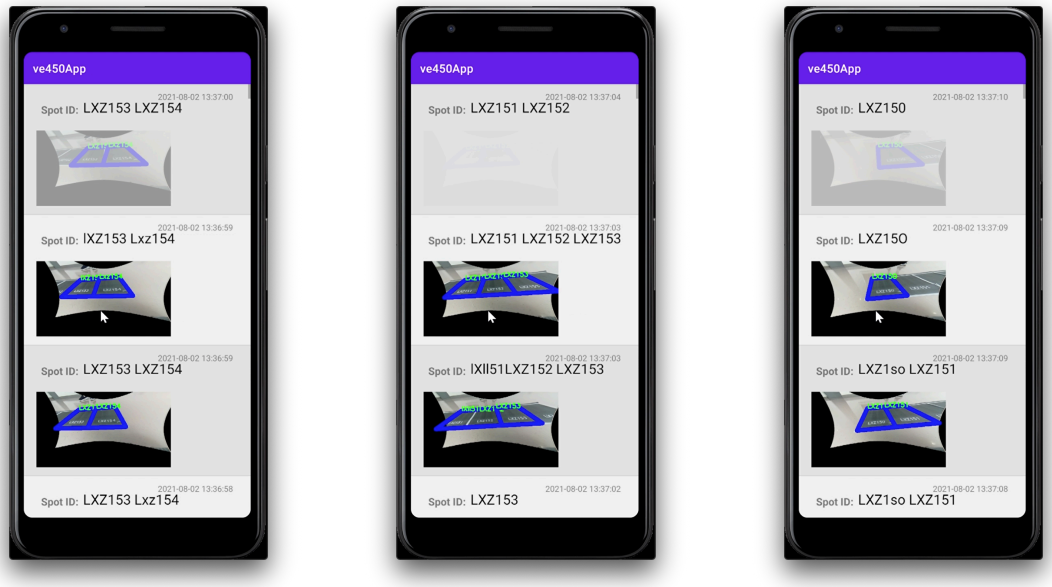
results through the phone apps, which will record the parking spot IDs so that we can easily see the successful recognition rate and missing rate.



a) Parking lot 1 b) Parking lot 2 c) Parking lot 3

Illustration 6–3 Three parking lots used for analysing accuracy

From the 3 runs on the simulated parking slots, we captured 51 frames in total and we counted the total number of characters occurred in all of the frames as well as how many of them are successfully recognized. The fraction we got is 0.9398, which meets our engineering specifications. And for the meeting rate, during all of the tests, all of the parking slots have been recognized successfully in at least one of the frames so our missing rate is 0%. Sample screenshots for a run are shown in Illustration 6–4.



a) Sample screenshot 1 b) Sample screenshot 2 c) Sample screenshot 3

Illustration 6–4 Three sample screenshots in rum 1

Chapter 7 Project Schedule

7.1 Schedule

Our project schedule is summarized as shown in the Gantt Chart Illustration 7–1. A more detailed version is Illustration in the appendix.

WBS NUMBER	TASK TITLE	TASK OWNER	DURATION	PHASE ONE		PHASE TWO		PHASE THREE				PHASE FOUR		
				WEEK 3	WEEK 4	WEEK 5	WEEK 6	WEEK 7	WEEK 8	WEEK 9	WEEK 10	WEEK 11	WEEK 12	WEEK 13
				5.24-5.30	5.31-6.6	6.7-6.13	6.14-6.20	6.21-6.27	6.28-7.4	7.5-7.11	7.12-7.18	7.19-7.25	7.26-8.1	8.2-8.6
1	**Project Conception and Initiation**													
1.1	Project Definition and Initiation	All	4d											
1.2	Literature Review	All	5d											
1.3	Project Design	All	5d											
1.4	Schedule Planning	All	4d											
2	**DR #1 + Project Preparation**													
2.1	Budget + Purchase Material	xwh, zyl	4d											
2.2	Literature Search and Benchmarking	xwh, zsa	3d											
2.3	Quantification of Design Specifications	zyl, wyx	3d											
2.4	Make Demo Parking Plot	wyx	5d											
2.5	Build Car + Camera System	ygx, xwh	5d											
2.6	Collect Camera Data	TBD	5d											
2.7	Existed Algorithm Testing	TBD	5d											
	DR#1 Report	All	1d											
3	**Algorithm Implementation**													
3.1	Own Algorithm Design	TBD	4d											
3.2	Algorithm Implementation	TBD	3d											
3.3	Algorithm Improvement	TBD	10d											
3.4	Test Set Fetching	TBD	10d											
	DR#3 Presentation	All	1d											
4	**Improvement and Finishing**													
4.1	Integrating System	TBD	5d											
4.2	Finishing Prototype	TBD	5d											
4.3	Final Improvement	All	10d											
	Final, Oral Defense, Expo	All	3d											

Illustration 7–1 Gantt Chart for the project schedule

We set the design review and the final expo as the milestones shown as the red block in the Illustration 7–1.

- Before the report due time of design review 1, we finished all the project preparations, including purchasing all materials, building the car and camera system, making demo parking slots, etc. Also we started to collect fisheye camera data.
- Before the presentation of design review 3, we managed to implement the draft version of our algorithms and gave the output of the recognized parking slot ID.
- Until the final expo, we succeeded in displaying the recognized parking slot ID with high accuracy and within the limited process time and tried to make some improvements, including increasing our recognition rate, better performance under special

cases, increasing our processing speed, etc.

7.2 Budget

The budget of our project is shown in Table 7–1, mainly including a fisheye camera, Raspberry Pi, a remote control car, and route and server.

Table 7–1 Budget for the project

Item	Quantity	Purchased From	Number	Cost (Total)	Contact
Fisheye Camera	1	深圳市龙岗区乐幻智能经营部*		¥126.00	深圳市龙岗区龙岗街道南联社区爱南路212-214号A401 13751329627
Raspberry Pi	1	深圳市龙岗区乐幻智能经营部*		¥309.90	深圳市龙岗区龙岗街道南联社区爱南路212-214号A401 13751329628
Raspberry Extension Board	1	深圳市龙岗区乐幻智能经营部*		¥217.82	深圳市龙岗区龙岗街道南联社区爱南路212-214号A401 13751329629
Car Bracket	1	深圳市龙岗区乐幻智能经营部*		¥79.21	深圳市龙岗区龙岗街道南联社区爱南路212-214号A401 13751329630
Battery	1	深圳市龙岗区乐幻智能经营部*		¥178.22	深圳市龙岗区龙岗街道南联社区爱南路212-214号A401 13751329631
Route and Server	1	上海圆迈贸易有限公司 **	TL-WDR5620 千兆版	¥126.00	上海市嘉定工业区叶城路1118号19层1901室 021-39915587
Deceleration Gear	1	乐高玩具（上海）有限公司		¥27.00	上海市徐汇区淮海中路999号IAPM商场办公楼(ICC2)31楼
Wooden Plant	4	徐州中豪家具有限公司***	120mm*40mm *1.7mm	¥226.80	睢宁县沙集镇兴国村
Print	134	印萌自助打印		¥26.80	上海市闵行区文俊路上海交通大学(闵行校区)
Macrowave Model	1	深圳市龙岗区乐幻智能经营部*		¥29.70	深圳市龙岗区龙岗街道南联社区爱南路212-214号A401 13751329631

* https://m.tb.cn/h.fax7JOP?sm=f4e746
** https://item.jd.com/4772588.html
*** https://m.tb.cn/h.4z4nO6m?sm=fdb44f

Total = ¥1340.65

Chapter 8 Discussion and Conclusion

8.1 Discussion

Now the project is completed, thinking back, there are several alternative choices we could make.

We would try to collect more data in a real parking slot with a real car with a camera installed, and then use deep learning to train our model. Currently, there is no available dataset that exactly fits our setting. So in order to apply deep learning to train the model, we would need to collect the data in a real parking slot by ourselves, which is a bit costly and time-consuming. Due to the limited 2-month duration, we chose to use the traditional preprocess before applying the OCR, instead of retraining the OCR, which is economic and less time-consuming. Also, we chose to use a small car in a simulated parking lot with proper scale instead of driving a real car with a camera installed in a real parking lot, as it is more economical, flexible, and easier to demo in the expo. Yet our model would eventually be installed on the real vehicle and collect the video and data from the fisheye camera installed on the vehicle. There is still some work and process needed to realize this step. And collecting data in a real parking lot with a real car with a camera installed would be closer to this purpose.

Besides, for the timeline, actually, we start early and plan ahead. At the beginning of the project, we have already realized the limited 2-month duration. Also, we noticed that the video from the fisheye camera would be the essential key and foundation of our whole project. So as long as we made a general decision of our overall structures, namely the decision to use a small car in a simulated parking lot with proper scale instead of driving a real car with a camera installed in a real parking lot, we started to purchase the necessary materials, including the fisheye camera, the Raspberry Pi, and a remote control car, build our simulated parking slot and start to collect the fisheye video. Then we gradually work on the algorithms part to process the video. Thus, in general, our process is as scheduled.

8.2 Conclusion

When driving in a huge parking lot, it is quite difficult for people to track where they have parked their car or even remember the final parking location. Therefore, using the camera on the car to detect and remember the spot ID really helps the user find the vehicle easily.

With the help of the fisheye camera on the car to record the spot ID in real-time, it is also meaningful in terms of the industry purpose. Big data companies could use it to collect the parking lot information and use it to analyze the driving habit of the driver. A parking lot management system could also use the information to trace every car in the parking lot. And specifically, a car company could use the data for the cartographic semantics to better construct the driverless car.

Since the main purpose of our project is to record the spot ID in real-time, high accuracy and process speed are highly demanded among the engineering specifications. Thus, our target is to track the parking slots in real-time, recognize and record the parking slot ID with the real-time videos from the on-vehicle fisheye camera.

In this project, we proposed and implemented a real-time parking spot number detection system based on the fisheye camera. First, we collect photos from the Raspberry Pi car with fisheye camera installed. Then, our algorithm follows the pipeline of fisheye calibration −parking slot detection −projective transformation −optical character recognition −APP demo to detect and display parking slot IDs in the images. Parking slot detection reduces the interference from other obstacles. Transformation returns a frame in vertical view and helps OCR to extract spot number. Finally, the results are sent to the phone and users could see the result on their mobile APPs in real-time.

When addressing this problem, the processing speed and detection accuracy are essential. Now we have implemented a prototype for our project, succeeded in giving the recognized parking slot ID with a high accuracy 94% (> 90% in the Engineering Specifications) and high processing rate performance (0.33 second per frame < 1s in Engineering Specifications) according to our validation implementation. Yet to make it

a marketable product, companies could make some improvements listed in the Section 8.3 and also extend its applications including the cartographic semantic map.

8.3 Future Works

Currently, for every parking slot, our accuracy of detecting the parking slot ID is around 94%. To further improve the accuracy, we recommend that for the same parking slot, as we would take photo shots for several times, we can collect all the results of the same parking slot, and take the most popular recognized ID results as the final recognized results. Also we recommend to use Matching Pursuit, namely, tracking the moving of the parking slot object in the videos, and then choose the output of the recognized ID results to optimize this problem and increase the accuracy, or to extend the use of Matching Pursuit to other applications.

Besides, currently we are able to handle the incontinuous parking spot line by applying Hough transformation to detect line segments and grouping them to form a whole line (more details can refer to methods 2 in subsection 4.2.2). An alternative method that we recommend to have a try in the future is to focus more on the two points that are closest to the vehicle, and from these two points, generate a rectangle and apply optical character recognition inside it, to get rid of the dependence on the continuous parking slot line to have a more accurate recognition.

Moreover, we recommend to collect more data in a real parking slot with a real car with a fisheye camera installed, and then use deep learning to train the model. As the model would eventually be installed on the real vehicle and collect the video and data from the on-vehicle fisheye camera, collecting data in a real parking slot with a real car with a camera installed would be closer to this purpose.

References

[1] CHEN D, ODOBEZ J M, BOURLARD H. Text detection and recognition in images and video frames[J]. Pattern Recognition, 2004, 37: 595-608.

[2] HOUBEN S, KOMAR M, HOHM A, et al. On-vehicle video-based parking lot recognition with fisheye optics[C]. 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), 2013: 7-12.

[3] FLECK M M. Perspective projection: the wrong imaging model[J]. Department of Computer Science, University of Iowa, 1995: 1-27.

[4] ZHU H, YANG J, LIU Z. Fisheye Camera Calibration with Two Pairs of Vanishing Points[C]. 2009 International Conference on Information Technology and Computer Science: 2009, 1: 321-324.

[5] WU Y, LI Y, HU Z. Easy Calibration for Para-catadioptric-like Camera[C]. 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006: 5719-5724.

[6] HENSEL S, MARINOV M B, SCHWARZ R. Fisheye Camera Calibration and Distortion Correction for Ground Based Sky Imagery[C]. 2018 IEEE XXVII International Scientific Conference Electronics - ET. 2018: 1-4.

[7] HUNAG J, ZHANG L, SHEN Y, et al. DMPR-PS: A novel approach for parking-slot detection using directional marking-point regression[C]. IEEE International Conference on Multimedia and Expo (ICME), 2019: 212-217.

[8] K. HAMADA M F, Z. Hu, CHEN H. Surround view based parking lot detection and tracking[J]. IEEE Intelligent Vehicles Symposium, 2015: 1106-1111.

[9] DUDA R O, HART P E. Use of the Hough Transformation to Detect Lines and Curves in Pictures[C]. Artificial Intelligence Center (SRI International). 1971: 11-15.

[10] Youtube. TEACHING P U. Class 12 - Projective Transformations[EB/OL]. 2019[2021-08-11]. https://www.youtube.com/watch?v=54Qtu3S9HJU%5C&t=287s.

[11] TIAN Z, HUANG W, HE T, et al. Detecting Text in Natural Image with Connectionist Text Proposal Network[C]. European conference on computer vision, 2016: 56-72.

[12] GitHub. OCR Based on Connectionist Text Proposal Network and Convolutional Recurrent Neural Network[EB/OL]. 2020[2021-08-11]. https://github.com/courao/ocr.pytorch.

[13] SHI B, BAI X, YAO C. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 2016, 39(11): 2298-2304.

[14] HARRIS C, STEPHENS M. A combined corner and edge detector[C]. Proceedings of the 4th Alvey Vision Conference. 1988: 147-151.

[15] DigitalOcean. GLASS E. How To Set Up Django with Postgres, Nginx, and Gunicorn on Ubuntu 20.04[EB/OL]. 2020[2021-08-11]. https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-20-04.

Acknowledgements

We would like to extend our sincere gratitude to our mentor Jerry Wang, Jony Xu, and Tiger Zhang from GM, for their instructive advice and useful suggestions, and the attendance of the weekly meetings on our project, timely reply, and materials provided. We are deeply grateful for their help throughout this project.

Our thanks would go to our instructor Jigang Wu for his useful suggestions on our project, schedule, presentation, and reports.

Finally, thanks to our sponsor GM for their attention to our project and for providing funding for our project.

Appendix A Engineering Changes Notice (ECN)

A.1 Fisheye Calibration

This part is deleted as we turn to MATLAB for calibration part.

To test the camera in Raspberry Pi, we execute:

```
raspistill -o image.jpg
```

This will generate a file named "image.jpg" in current path. For more usage of **raspistill**, please refer to

```
man raspistill
```

Then we need to establish ssh connection. To create the ssh key pair, execute:

```
ssh-keygen
```

and choose all default options. Then execute:

```
ssh-copy-id <username>@<ip_address>
```

To test the ssh connection, execute:

```
ssh <username>@<ip_address>
```

We use fisheye camera model of OpenCV in our calibration part. The calibration parameters should be prepared before we apply calibration to the photos. The main function used here is:

- **cv2.fisheye.calibrate**

  This function is used to calculate calibration parameters from given checkerboard photos. A sample checkerboard picture is shown in Illustration A–1.

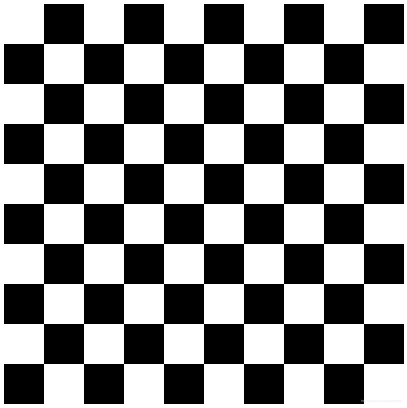


Illustration A–1 Sample checkerboard

We take photos of a checkerboard at different places as shown in Illustration A–2.

Illustration A–2 Checkerboard at difference angles

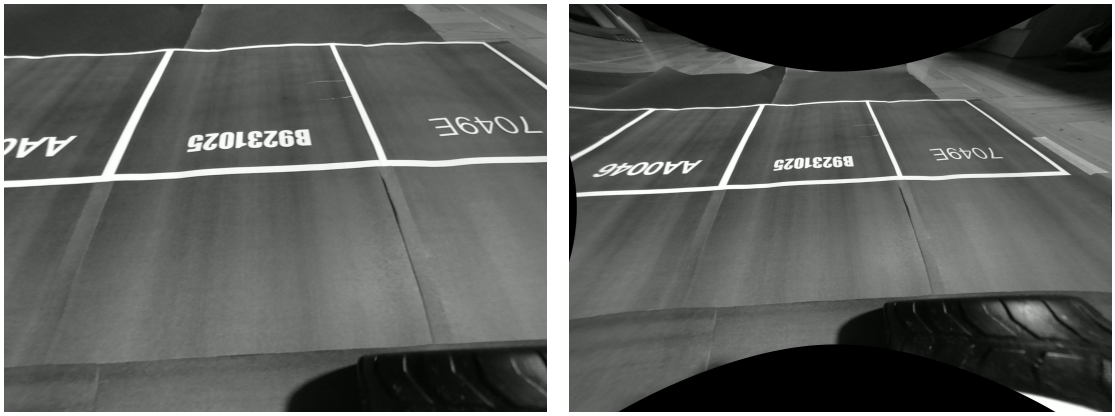
Applying these photos to **cv2.fisheye.calibrate**, we will get the intrinsic and distort parameters.

After we get the calibration parameters, we can apply calibration to the photos now. The main functions used here are:

1. **cv2.fisheye.estimateNewCameraMatrixForUndistortRectify**

   This function is used to adjust the calibrate parameters. If we directly apply the undistort function, we will get the picture shown in Illustration A–3a. The scale is kept while some information is missing. After we

use **cv2.fisheye.estimateNewCameraMatrixForUndistortRectify**, full information will be kept as shown in Illustration A–3b.



a) Directly apply the function b) Adjust the parameters

Illustration A–3 Comparison between the original and the new undistort pictures

2. **cv2.fisheye.initUndistortRectifyMap**

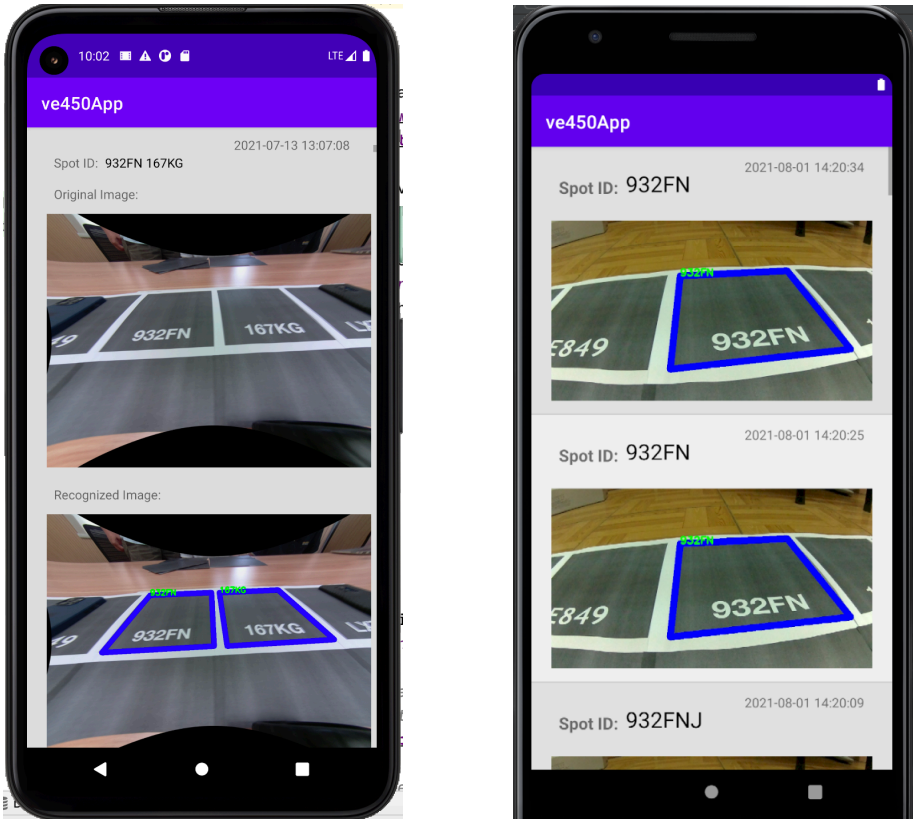
   This function will init the mapping matrix for **cv2.remap** use.

3. **cv2.remap**

   This function will do the calibration. An undistort picture will be the output.

A.2 Mobile App Development

Previously, the user interface of our mobile app includes the original picture after calibration, as shown in the Illustration **??**, which we thought would give users a clear comparison between the original look of the parking slot and the recognized one processed by our algorithm. However, since the information in the original image is already involved in the processed one, we decided to remove the original image and only show the one that has been processed, to reduce the time for sending images to the server. Also, we modify the font size of the spot ID to make it more clear and informative.



a) Original user interface of our mobile app

b) New user interface of our mobile app

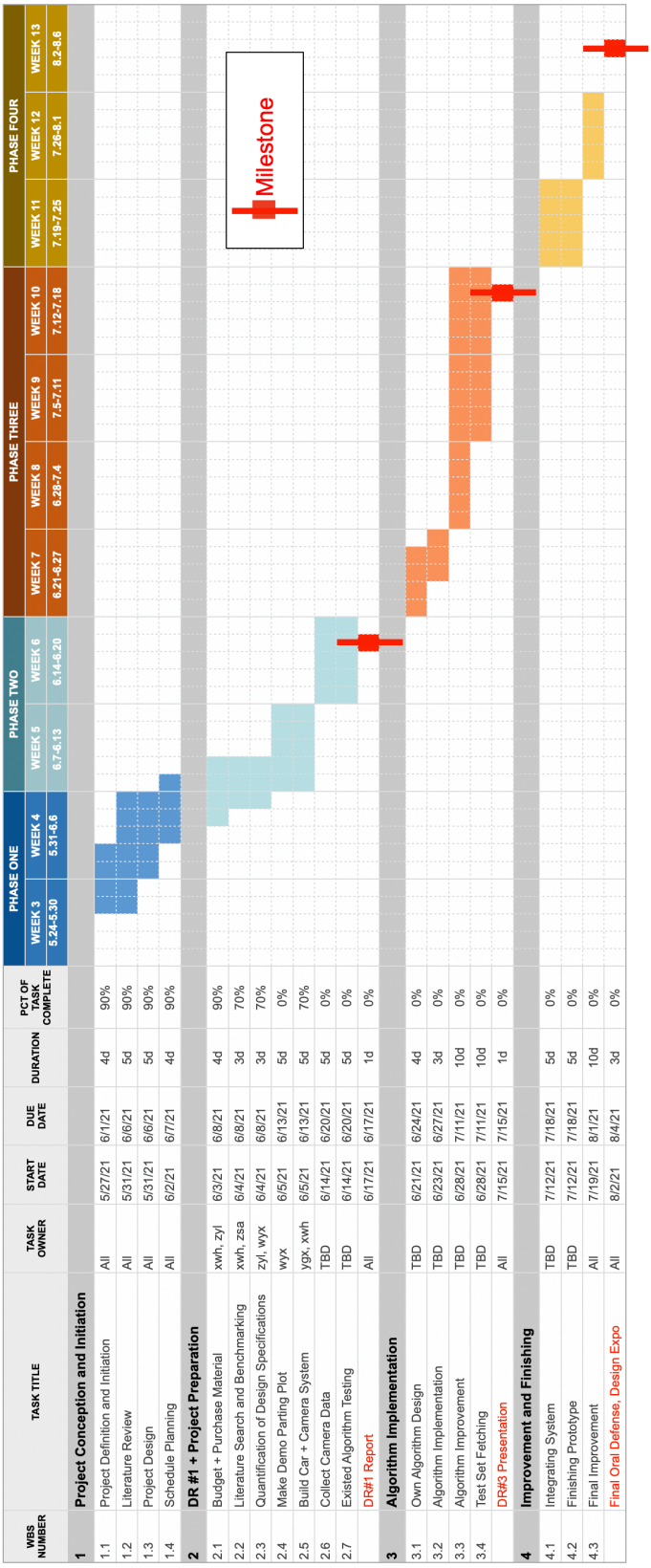
Illustration A–4 Comparison between the original and the new user interface of our mobile app

A.3 Component Connection and Parallel Computing

Before Design Review 3, our workflow follows 4–32, where the Raspberry Pi car take photos automatically with a certain frequency, and send those images to the computer via SSH. The directory it sends images to is monitored by python watchdog library, which will then call the detection program on the new image path.

After Design Review 3, we optimize our workflow as 4–33. Now two monitoring programs are adopted. Once MATLAB finishes taking one image, python will start process it. Once previous detection is completed, MATLAB will get notified and start taking next images. The two programs run concurrently to improve both memory and time.

Appendix B Detailed Gantt Chart



WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Project Conception and Initiation**					
1.1	Project Definition and Initiation	All	5/27/21	6/1/21	4d	90%
1.2	Literature Review	All	5/31/21	6/6/21	5d	90%
1.3	Project Design	All	5/31/21	6/6/21	5d	90%
1.4	Schedule Planning	All	6/2/21	6/7/21	4d	90%
2	**DR #1 + Project Preparation**					
2.1	Budget + Purchase Material	xwh, zyl	6/3/21	6/8/21	4d	90%
2.2	Literature Search and Benchmarking	xwh, zsa	6/4/21	6/8/21	3d	70%
2.3	Quantification of Design Specifications	zyl, wyx	6/4/21	6/8/21	3d	70%
2.4	Make Demo Parting Plot	wyx	6/5/21	6/13/21	5d	0%
2.5	Build Car + Camera System	ygx, xwh	6/5/21	6/13/21	5d	70%
2.6	Collect Camera Data	TBD	6/14/21	6/20/21	5d	0%
2.7	Existed Algorithm Testing	TBD	6/14/21	6/20/21	5d	0%
	DR#1 Report	All	6/17/21	6/17/21	1d	0%
3	**Algorithm Implementation**					
3.1	Own Algorithm Design	TBD	6/21/21	6/24/21	4d	0%
3.2	Algorithm Implementation	TBD	6/23/21	6/27/21	3d	0%
3.3	Algorithm Improvement	TBD	6/28/21	7/11/21	10d	0%
3.4	Test Set Fetching	TBD	6/28/21	7/11/21	10d	0%
	DR#3 Presentation	All	7/15/21	7/15/21	1d	0%
4	**Improvement and Finishing**					
4.1	Integrating System	TBD	7/12/21	7/18/21	5d	0%
4.2	Finishing Prototype	TBD	7/12/21	7/18/21	5d	0%
4.3	Final Improvement	All	7/19/21	8/1/21	10d	0%
	Final Oral Defense, Design Expo	All	8/2/21	8/4/21	3d	0%

Illustration B–1 Gantt Chart (detailed version) for the project schedule

Appendix C Team Member

Yixuan Wang is currently a senior Electrical and Computer Engineering student in the University of Michigan-Shanghai Jiao Tong University Joint Institute (UM-SJTU JI). She also attended the dual-degree program and have earned a bachelor of science in engineering degree in computer science at the University of Michigan. She is working towards the Master of Science in Information Technology, Privacy Engineering (MSIT-PE) Program of the School of Computer Science at Carnegie Mellon University (CMU). Her main research interests include AI, machine learning, privacy, and human computer interaction. In the University of Michigan, she had some research experience in the area of artificial intelligence and machine learning application on privacy. She is working on a project studying human's privacy preference and human's willingness to trade personal data. After graduation from Carnegie Mellon University, she plans to work as a privacy engineer or software engineer in a high-tech company.

Weihao Xia was born in Nanjing, China, in 1999. He is currently a senior Electrical and Computer Engineering student in UM-SJTU JI, and will receive the B.S.E degree in Electrical and Computer Engineering. During his Bachelor's degree, he has some IPP experience with small-sized infrared camera development. He has also taken several courses in the field of computer science, including algorithm and system design. After graduation from SJTU, he will work towards the Master's degree in Electrical Engineering at University of Pennsylvania. His main research interests include information processing, decision theory and distribution system. After obtaining a Master's degree at University of Pennsylvania, he plans to do further research in the field of information theory and distribution system for a PhD degree.

Guoxin Yin has been studying at Shanghai Jiao Tong University, China to pursue a Bachelor of Engineering (B.E) degree in Electrical and Computer Engineering since 2017. She has also attended the dual-degree program and has received a Bachelor of Science in Engineering (B.S.E) degree, in field of Computer Science, from the University of Michigan, USA, in 2021. She will pursue the Master's degree in Computer Science at Columbia University in the City of New York in the following two years. After graduation from Columbia University, she plans to work in a tech company to apply what she learned at school to the industry. During her four years college life, she gradually developed her interests in software development and computer system. Meanwhile, she is actively involved in extracurricular activities and attended several school-wide performances.

Yuanli Zhu majors in Electrical and Computer Engineering in Shanghai Jiao Tong University and has received the degree in Computer Science from the University of Michigan. After graduation, he is going to pursue master's degree in Computer Science at the University of Michigan. He had some research experience in the area of programming languages and artificial intelligence. He had an internship in Cisco working on the prediction of signal intensity through satellite map. In the University of Michigan, he spent one year doing research in program synthesis in different domains including regular expression, superoptimization and SQL queries. In the future, he plans to try something new and dig more into compilers and operating systems. After graduation, he plans to find a job in a high-tech company to explore cutting-edge technology in those area.

Shengan Zhang is a senior student majoring in Electrical and Computer Engineering at Shanghai Jiao Tong University. He has also received an B.S.E. degree in Computer Science at the University of Michigan. After graduation, he is going to pursue a master's degree in Computer Science at the University of Illinois Urbana-Champaign. He had some research experience in machine learning and artificial intelligence. At the University of Michigan, he did some research in the application of computer vision algorithm on transportation data. When pursuing the master's degree in the future, he plans to try more on parallel computing and distributed systems.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY
CAPSTONE DESIGN (BACHELOR'S THESIS)
INDIVIDUAL CONTRIBUTION REPORT

Student Name：朱元立

Student ID Number：517370910017

Major: Electrical and Computer Engineering

This project really fits my interest and ability. In last four years, I had multiple project experience in the area of computer vision and autonomous vehicles and I am very interested in it. Therefore, to further explore this area, I chose this project, parking spot number detection using fisheye camera.

As we mentioned in the report, this project is mainly divided into 3 parts, hardware part including the remote-control car and fisheye camera, algorithm part to detect spot number and phone application part to show the final result. The most challenging part is the algorithm part because other parts are mainly engineering problems while parking spot number detection requires more thinking and innovation. Since I am the team leader and I am also interested in computer vision, the main work I am responsible for in this project is the algorithm part.

The first step was literature review. For the first week (week4), all of our team members collected and read papers in this area by searching "parking spot number detection using fisheye camera" and we found three areas of literature turned out to be very useful, parking spot detection, character recognition and fisheye calibration. Then, in week5, we divided our work and focused on different areas. I was responsible for digging more into parking spot detection and I read about 5 papers in this area carefully. I found that years ago people used different technique including line detection, corner detection and other techniques to find parking spot but recent research preferred different architecture of neural network. In week 6, I tried to install and run three different parking spot detection APIs found in GitHub and in the meantime, my teammate also found other components of API and partly done the hardware part. By checking the input and output of different component from some sample photos taken by the fisheye camera, I proposed the final design pipeline, fisheye view calibration −parking spot detection − projective transformation −optical character recognition (OCR).

In week 7 and week 8, I used MATLAB to implement our design from parking spot detection to OCR. In fact, projective transformation and OCR are directly provided by MATLAB computer vision tool boxes. The most difficult part is parking spot detection. The previous 3 APIs I found are all neural network based, which means they have to

be retrained before they can be used. Otherwise, from our experiment, they can detect nothing because of the different environment. However, we do not have so much data to retrain our model. Though I could not find the code of the paper which does not use neural network, I tried to reimplement their work. However, those papers have some assumptions that we do not satisfied. For example, one paper describes a method that can be applied in bird view, which means parking spot is a rectangle and is not true in our case. However, because our parking lot is simulated, which means we have less distractions than in the real case, I came up the algorithm that directly detect largest white region as parking spot line, which simplifies our problem quite a lot and performs surprisingly well. In week 9, I did some testing on about one hundred pictures collected by my teammate and adjust the parameter to make it perform better.

In week 10, we held an offline meeting with our GM sponsors and mentors. It was a very busy week to make everything work fine and presented the prototype. My main work was to make algorithm part successfully fit into the rest of the pipeline of the system, including getting photo from our Raspberry Pi car and sending result to the server. I also managed to compiled MATLAB into Python package so that it can be called by the main Python process. Finally, our prototype works and the demo was quite successful. They gave our team high compliment and also raised some potential improvement for us to do for the last few weeks.

One problem they thought should be improved was that our car ran too fast that the processing speed of our algorithm could not follow the speed of the car and it may miss some parking spot. In order to solve this problem, I tried to add gear reduction unit on our car. However, we planned to use different speed to validate our result and it is inconvenient to change gear reduction unit. Therefore, I used Lego to build up the module on our car that can easily change the speed of the car by simply changing the combination of the gears. The substitution of Lego gears is very convenient, which makes our testing easier.

Another problem was that our algorithm could only deal with perfect case. If there were some distractions, such as the coverage of part of the spot, our algorithm would

fail completely. The main reason was that we used a simple algorithm in our parking spot detection. To solve this problem, I redone literature reviews about parking spot detection and focused on non-neural-network method. By comparing different methods 3 papers, I implemented an algorithm that used Hough transformation to detect line segment and then doing line segment to group the line segment to form a complete line. In this way, even if some parts of the parking spot are covered, as long as some line segments are detected, the parking spot could still be detected.

Besides technique contribution, I still spent lots of time in holding meetings, presenting in three design reviews and writing reports. In this thesis, I was mainly responsible for writing user requirement and engineering specifications, parking spot detection algorithm, assembling of Lego module and validation plan part. Those technique communications made me think harder and clearer on our project such as what users truly want and what part of the project could be improved.

In this Capstone Design, I really learnt a lot from the project, from the GM sponsors and instructors, and also from my excellent teammates. I felt very lucky to be assigned to this project and be the team leader of this wonderful team.

上海交通大学 毕业设计（学士学位论文）

单独工作报告

SHANGHAI JIAO TONG UNIVERSITY
CAPSTONE DESIGN (BACHELOR'S THESIS)
INDIVIDUAL CONTRIBUTION REPORT

Student Name：王译萱

Student ID Number：517370910214

Major: Electrical and Computer Engineering

My main work for this project lies in the literature search on projective transformation, parking spot detection, and fisheye video processing, working on projective transformation, and parking spot detection, designing and building our simulated parking spot, assisting coding for Android APP, as well as working on the technical communication including the demo animation video, the expo video, design review presentations and reports with my teammates.

At first, after we had a meeting with our sponsors and had a general idea about our project, I together with my teammates had a brainstorming about our project including project implementation methods, customer requirements, and engineering specifications, schedule, etc. And we decided to first literature search for the related topics to collect more information and then make further decisions. After dividing the project into several steps, I worked on the literature review on fisheye video processing and different methods of parking spot detection.

Actually when I literature searched with keywords "fisheye camera" and "parking spot," there were several related output articles. Yet there is no article which fits our situation perfectly. For example, one article detects the parking spot with the fisheye cameras of the indoor parking lot. Yet in our case, the fisheye cameras are on-vehicle. However, when we implemented our algorithm, we figured out that actually we could apply the methods to detect the parking spot white lines in this paper on our project. Thus we should not only literature search the articles just suitable for our project, but also learn and extract different ideas and methods through the literature review.

After a brainstorming discussion on the materials collected, we decided to split our concept flows into several steps, including collecting videos from the Raspberry Pi car, calibration, parking spot detection, projective transformation, optical character recognition, and mobile APP. Our teams decided to divide these steps roughly equally. And I was responsible for the search of the available implementations or APIs for projective transformation and worked on the weighted decision matrix of parking spot detection and transformation part, and presented my findings in design review 2. After the group meeting, we made our decision based on the criterion including accuracy, speed, cost,

etc.

At first, I implemented the projective transformation with Python. Yet soon I realized that we were working on a project and we need to corporate with each other. As the other steps in our algorithm, including calibration, parking spot detection, can be easily implemented with Matlab, to save the time of code processing, which is one of the key engineering specifications, we finally decided to implement the projective transformation with Matlab.

Besides, I designed and built our simulated parking spot. First, I collected the length and width information of our simulated toy car from Weihao Xia. According to these parameters and the real parking spot parameters, I set the width of the white lines of our parking spots, designed the outlook of the parking spots, drew the draft version of the parking spots with the software Pages on Mac, and printed them out. Then I stuck the separated parking spots together with glue and tested this draft parking spot with our simulated toy car. Gradually, I adjusted the parking spots ID with different sizes, fonts, lengths, as well as the parking spots white lines with different widths, according to our real implementation experience and results. And we tested and validated the hypothesis that our detection would work better with thick white parking spot lines and with less obstruct in the background. Actually, our Algorithm would be implemented with the real parking spot dataset in the future. And at that time, our algorithm has to fit the real cases. And this is why we tried different kinds of parking spots. Also, this is one of the directions on how to improve our algorithm —increase the accuracy under some special cases including dealing with obstructs in the background of the parking spots. Besides, previously our parking spots are printed paper, which is easily be wrinkled or damaged. So we decided to handle it. I purchased and built the parking spots with planks to make our car drive on the planks easier and have a better demo.

Moreover, I corporated with Yuanli Zhu and implemented the Matlab code to output the intersection points of several certain input lines, which are useful in displaying the final detected parking spot blocks. In addition, I corporated with Guoxin Yin and Shengan Zhang, helped design and build the Android APP for our project, tested the implemen-

tation of our APP.

For the demo animation video, we scheduled a meeting and collected the materials needed for the video, including the real-time controller car driving, real-time code processing, and real-time APP results output display. Then I wrote the subtitles for this demo video, recorded the aside in the play, added the subtitles for this video, and edited the video into the limited 60 seconds. Corporated with Guoxin Yin, I designed and edited the expo video, including collaborated with my teammates to collect more materials needed for the video, e.g., the algorithm flow chart, the screenshot of our codes, etc., writing the subtitles for our concept diagram and algorithms and our demo animation video, adding subtitles including Chinese and English version, editing the video into roughly 3 minutes, etc.

In terms of technical communication, after discussing the general idea of the project implementation with all other teammates, I made our group schedule according to the timeline of our design course, and summarized it with a Gantt chart. Corporated with Yuanli Zhu, I drew the basic structure of the quality function deployment chart for design review 1 presentation. I worked on the conclusion part of the design review 1 report and finalized the report. For design review 3 presentation, I worked on the introduction part, talking about our problems, ideas, and customer requirements, and engineering specifications. For the design review 3 report, I worked on the transformation part in the concept selection and implementation, plan, budget, and conclusion. For the final thesis draft, with the collaboration of my teammates, I worked on the latex format including the abstract, content, tables, etc. For the final version, I worked on the discussion, conclusion, future work, and acknowledgement section.

From this Capstone Design, I really learnt a lot from the project. I gained the experience to design, implement and produce a product from an initial idea with my teammates. Thank the sponsors and instructors Jerry Wang, Jony Xu, and Tiger Zhang from GM for their weekly useful advice and close attention on our project, thank our instructor Jigang for the weekly instructions and also thank my excellent teammates for this wonderful teamwork experience.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：夏伟豪

Student ID Number：517370910061

Major: Electrical and Computer Engineering

In the project Parking Spot Detection Using Fisheye Camera, I am responsible for the design and maintenance of the raspberry pi car and Raspbian platform, and the calibration part of the algorithm. The job includes assembling the model car; looking for available fisheye camera module and installing it on the model car; config the working environment and testing the camera function on the Raspbian system; designing and implementing the fisheye calibration algorithm and preparing the corresponding API. The workload or me is approximately 15 hours per week. The final contribution of me to the project is: providing a remotely controlled car with fisheye camera modules capturing photos; calibrating the photo and providing the API for further processing.

Although it is best to drive a real car with camera installed in a real parking lot, it is not suitable for our experiment for following reasons: a real car costs much, either for buying or renting one. It is unnecessary as I just need to move the position of the camera; I cannot change the character pattern in a real parking lot. I need to test different patterns for the validation; the parking spots in our university doesn't have IDs. I need to go to the shopping mall to test our camera. Moving a small car in a simulated parking lot with proper scale is more suitable for us. Therefore, Raspberry Pi, as a complete and widely used embedded platform, becomes the best choice for us. Various kinds of fisheye camera modules are available for this platform, too.

A common size of a simulated car with Raspberry Pi board on it is about an A4 paper, which makes the build of simulated parking lot easier. A parking spot unit can be designed and printed on an A4 paper, while combining several A4 papers will give us a simulated parking lot. I can easily adjust the size as well as the character pattern for the parking spot. In addition, A4 paper is very portable. As the data set of fisheye images (video) with characters is hard to find, I need to do calibration first so that I can use normal data set with undistort images. Many works have been done for fisheye model. Most of the done work are followed by two steps: (1) estimate intrinsic parameters of the camera, (2) apply parameters to calibrate images and do some correction.

Among those calibration methods, OpenCV fits our project best: fisheye camera model on OpenCV has well-designed interface as well as carefully written documents;

OpenCV provides Python based algorithm which can be easily run on small platform. As I am using Raspberry Pi on the car as our working platform, MATLAB is not suitable; OpenCV is widely used for good calibration quality on edge even on fisheye camera with large view angle (>170). Function undistortImage() can automatically fix the fuzzy edge and return a clear image as shown in Figure

I buy the car and Raspberry Pi platform from Hiwonder from Taobao. It has following features which are good for our project: it has motors and wheels so that the car can move freely in the parking lot; it provides a battery so that it can get rid of power lines. I can easily move it in the parking lot; it provides a mobile APP, so I can control the car remotely and get the camera video in real-time for testing.

I buy the fisheye camera module according to the engineering specifications from 1688. The parameters for this module are listed below: View angle 175, Max resolution 2592x1944, Max frame rate 30 fps, Interface CSI.

The combination of car and camera module is done then. The CSI line of fisheye camera module should be inserted into the CSI input of Raspberry Pi board. The scale of our simulated system is approximately 1:10.

I need to enable camera on Raspberry Pi OS. Execute: sudo raspi-config. Select Interfacing Options in the window appeared, then select Camera, then select Yes. To test the validation of camera, execute: vcgencmd get_camera. The correct result should be: supported = 1 detected = 1. I can now test the camera. Execute: raspistill -o image.jpg. This will generate a file named ”image.jpg” in current path. For more usage of raspistill, please refer to man raspistill. Then I need to establish ssh connection. To create the ssh key pair, execute: ssh-keygen, and choose all default options. Then execute: ssh-copy-id username@ip_address. To test the ssh connection, execute: ssh username@ip_address.

I use fisheye camera model of OpenCV in our calibration part. The calibration parameters should be prepared before I apply calibration to the photos. The main function used here is: cv2.fisheye.calibrate. This function is used to calculate calibration parameters from given checkerboard photos. A sample checkerboard picture is printed,

and I take photos of a checkerboard at different places. Applying these photos to cv2.fisheye.calibrate, I will get the intrinsic and distort parameters. After I get the calibration parameters, I can apply calibration to the photos now. The main functions used here are: cv2.fisheye.estimateNewCameraMatrixForUndistortRectify. This function is used to adjust the calibrate parameters. If I directly apply the undistort function, I will not get the complete picture. The scale is kept while some information is missing. After I use cv2.fisheye.estimateNewCameraMatrixForUndistortRectify, full information will be kept. Then, function cv2.fisheye.initUndistortRectifyMap will initial the mapping matrix for cv2.remap use. Function cv2.remap will do the calibration. An undistort picture will be the output.

In week 4, I did literature review on fisheye calibration, searched, and read corresponding works and designed our way to perform calibration. In week 5, I did benchmark on fisheye camera and video processing, tested our model car and camera. In week 6, I wrote benchmark of fisheye calibration and video processing, developed camera system on Raspberry Pi. In week 7, I worked on cluster processing and conclusion for DR2, tested camera functions on Raspberry Pi platform. In week 8, I tested photo taken and calibration in Linux MATLAB, adjusted fisheye camera's setting to make model parking lot looked real. In week 9, I setup the working process on the car: taking photo per 0.5 second, calibrate, send photo to another computer via SSH, and collected the first data set. In week 10, I upgraded the pipeline on the car, took extra photos for better calibration quality, adjusted the event trigger for the work on MATLAB, worked for the hardware on car part of DR3. In week 11, I worked on animation part of the expo video, hardware, and calibration part of the DR3 report, redesigned the working process on car, now using MATLAB to directly connect car and take photos.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY

CAPSTONE DESIGN (BACHELOR'S THESIS)

INDIVIDUAL CONTRIBUTION REPORT

Student Name：尹郭馨

Student ID Number：517370910043

Major: Electrical and Computer Engineering

During the whole capstone project period, my main job is to work with my teammates on the literature review, then focus on the back end server and mobile APP development to demonstrate the results and work on the technical communication such as design review presentations and reports with all my other teammates.

In this beginning of our capstone project, we were thinking of utilizing machine learning methods to train a good model to detect and recognize the spot ID. As we were going to use the fisheye camera to recognize the parking spot ID in real time, I searched a bunch of fisheye images dataset, such as those are in the CVPR2020 workshop. As the parking spot ID will be alphanumeric, I searched a bunch of number dataset such as MNIST handwritten digit databse and letter dataset that are provided by University of California, Irvine. And as we are going to achieve our recognition goal in real time, I also searched some real time object detection dataset such as YOLO. In addition, since our project is going to apply optical character recognition to extract the parking spot ID out, in order to avoid the inteference from the license plate number, we would like to specfically detection the parking spot lines out. In this way, we could focus on the parking spot ID inside the parking spot lines. Therefore, I also searched a bunch of parking spot datasets used for parking spot detection.

As the project was pushing forward, I was mainly working on the technical communication part of our project, with some literature review on the fisheye calibration and also helped assembled our Raspberry Pi car that was going to be used to simulate real-time driving. To begin with, when working with our first design review to talk about the benchmarking, requirements and specifications of our project, I together with my teammates to have a brainstorming about our customer requirements and engineering specifications. Then for the design review 1 presentation, I work on the project background and overview part. For the design review 1 report, I work on the title page, abstract and introduction parts. As the design review 2 approaches and we need to have our concepts generated and select the best one for our project development, I helped work on the literature review on the fisheye image calibration and run sample calibration codes using OpenCV library on Google Colab. In addition, I worked on the algorithm overview and the calibration part in the design review 2 presentation, to give an overview of the flow

chart of our algorithm design and present in detail whether to choose calibration using design matrix in our future design. Later, I prepared and presented a 4-slide presentation aiming at "Improve your presentation" in front of the whole class.

As my teammates began to wrap up the whole pipeline of the algorithm part of our prototype, I began to work on how to show the parking spot ID and the corresponding pictures on user's mobile APP. For this communication, I think it is better to store all the information in the cloud back end server so that whenever users would like to know the results, they can fetch from the server. Therefore, after my teammate rent a cloud server from Tencent, I began implementing the back end part of the server. I implemented a postgresql database management system to store the parking spot ID, the recognition timestamp, the original image after calibrated and the recognized image with parking spot lines highlighted and parking spot ID recognized. The timestamp is automatically set up whenever our software codes successfully post new information into the database. And when they do that, we will store the images they posted into our back end file system and only store the corresponding url of the images, which is a string object, into our database. After the database management system has been set up, I used Django as the web framework to allow users to interact with the IP address we rent. I implemented a post function for the HTTP POST request to let us post information to the database and store the posted images to our back end file system. To ensure that every file we save is unique, I renamed all the name of the files to let it compose of the image id and the timestamp. Moreover, I also implemented a get function for the HTTP GET request to fetch all the information from the databases, ordered by the timestamp with the latest one on the first. This would be used for the front end to get all the records from our database. After all the backend APIs were set up, I configured the Nginx and Gunicorn servers to start and stop the application in a more robust way. After I have set up all the backend services, I began working on implementing the mobile APP to interact with the backend server. I chose to implement the Android mobile APP instead of the IOS mobile APP in the following reasons. First, the Android Studio, which is the tool to develop Android mobile APP project, can be used in wider platform including Windows, MacOS, Linux than XCode is, where XCode is the only application to develop IOS mobile APP and it could only be used on MacOS system. Secondly, the emulator on Android Studio is

more powerful than those of XCode, which makes it more convenient to test and debug because we do not need an actual phone to do that. I implemented a list of parking spot information on the User Interface of the Android APP, where each item contains the spot ID, the timestamp, the original image after calibration and the recognized images with parking spot lines and recognized ID highlighted. I also implemented an auto refresh feature to automatically fetch the newest results from the back end database and a pull-down to refresh feature to manually refresh the latest results. After all the work for the mobile APP has been done, I worked on the backend server and mobile APP parts in the design review 3 presentation and reports and also helped design the final expo video.

During the whole capstone process period, I learned about how to work with my teammates to design and manufacture a product from an initial idea. And since I have previous knowledge and experience on the mobile APP development, I was also mainly worked on that part to better help demonstrate our final deliverables. Thank all my teammates for this wonderful group work experience and thank my instructor, Jigang for the weekly instruction and especially, thanks to the sponsors and mentors in General Motors for the wonderful technical help during the whole capstone design period.

上海交通大学 毕业设计（学士学位论文）
单独工作报告

SHANGHAI JIAO TONG UNIVERSITY
CAPSTONE DESIGN (BACHELOR'S THESIS)
INDIVIDUAL CONTRIBUTION REPORT

Student Name：张圣安

Student ID Number：517370910153

Major: Electrical and Computer Engineering

In the project Parking spot Number Detection Using Fisheye Camera, I'm mainly responsible for carrying out literature review to find proper methods; design and construct our working pipeline; cooperate with my teammates to do technical communication stuff, i.e., design review presentations and reports.

At the early stage of this project, we tried to figure out an optimal solution to the problem. Since there are no existing products in market, we researched on papers to find sub-solutions and combine them. I read about papers on optical character recognition (OCR) to extract parking spot number from video frames. I found that although there were current methods regarding parking spot number recognition, they were not adaptive to our project. That was mostly because the camera they used was placed on the wall of the lot instead of on the car. Since detection based on fisheye camera placed on vehicle hasn't been not proposed, we decided to apply primary OCR methods on parking video frames. To ensure great detection performance, we proposed applying parking spot detection and projective transformation before doing OCR. Then I started to research on the difference between traditional and machine-learning-based OCR methods. I reached the conclusion that although traditional methods require fewer computational resources, they are less adaptive to text of different fonts and locations. On the other hand, machine learning can work with characters of various perspectives and styles, which would be a better choice since the parking spot frames don't have a uniform mode.

Then I started to test OCR models with different implementation. They mainly came from python libraries and GitHub repositories. The first model I found is tesseract, and it was already implemented in pytesseract library. However, I found that this version of tesseract is not adaptive to our data format at all. That's because the training data it took are more document-like image, and as a result, it couldn't recognize parking spot number, which was of different font with document texts and located in various positions of the image. Then I tried with a GitHub project implemented with python keras. From its intro doc, I learned that it could work on various kinds of input styles, such as words in a billboard or road sign. But when I tested it on parking spot number image, I found that it would output an English word anyway. I came to realize that this was a lexicon-based model, which took an English word dictionary as its data label. Since

parking spot numbers are always some arbitrary strings, we couldn't use this method even though it was adaptive to different fonts and text positions. Finally, I found another GitHub project implemented with pytorch. The text detection part of this model is based on a Connectionist Text Proposal Network, which detects text proposals directly in the convolutional feature maps of images. Its recognition part is based on a Convolutional Recurrent Neural Network, which works well in lexicon-free text recognition tasks using integration of feature extraction, sequence modeling and transcription. It features a high accuracy, and it can also address Chinese characters. However, I find that it takes about 2 second to process one frame on average. Since we set 0.1s −1s as target processing time in engineering specification, we took it as our back-up plan, and used the MATLAB OCR function.

After that, I wrote codes to connect different components of the project. Before that, Weihao set up a calibration algorithm on the Rasberry Pi car, and calibrated video frames would be sent to a laptop via ssh. Based on that, I wrote a program monitoring that specific directory, and it would call a detection method implemented by Yuanli once new frames were sent to the directory. I wrote the program in python by deploying a library called watchdog. As long as a new image file appeared in the monitored directory, its path would be fetched by the watchdog event-handler, and I called Yuanli's function on that file. To avoid busy waiting, I made the program sleep every 0.1 second. After the parking spot number was recognized, my program would draw a bounding box on the corresponding position because python plot is much faster than MATLAB plot. And then the program would send the processed data to the APP server set up by Guoxin. After DR3, we made some further modifications to the project: now the photographing command was issued by MATLAB, instead of taking photos automatically, and the calibration was also done by MATLAB. To run the pipeline, I wrote two monitoring programs which run parallelly. One of them was still in python monitoring the calibrated image directory. The other was in MATLAB monitoring whether python had finished processing previous frames. This was done by comparing directory difference. Both photo-taking and detection took about 0.3 seconds, but processing each photo took about 0.4 seconds. Therefore, this parallel structure cut the whole running time by 1/3. What's more, I added multi-threading to the code that sent our results to the APP server, which

further made our project more efficient.

As regard to technical communication, in week 4, I presented my literature review results on OCR with my teammates. In week 5, our team worked on the DR1 presentation while my job was the benchmarking and related work section, especially on OCR and parking spot detection. In week 6, I wrote the benchmarking, info source of parking spot detection and optical character recognition in DR1 report. In week 7, my team prepared for DR2 presentation regarding conception generation and selection, and my task was to I worked on OCR methods comparison. In week 10, I was mainly responsible for writing the conclusion and future work part of DR3 presentation. In week 11, I worked on the expo poster. For DR3 report, I wrote sections on OCR in Concept Generation; OCR in Design Description; Abstract part; and Manufacturing Plan section. Finally in week 12, we drafted our final thesis according to DR3 report, and my task was to move and format DR3 page 24 - 32 to the thesis.