

学号： 2016220102015

姓名： 李达梽

专业方向： 信息工程

企业名称： 罗格斯大学

毕设课题名称： 基于深度学习的停

车空位找寻及车牌追踪系统

企业指导教师： Bo Yuan

院内代管教师： 王瑞锦

**信息与软件工程学院**

**毕业设计（顶岗实习）**

**中期报告**

目 录

[1. 毕业设计的进展情况 1](#_Toc38274992)

[1.1. 课题工作完成情况 1](#_Toc38274993)

[1.2. 知识技能学习情况 1](#_Toc38274994)

[1.3. 职业素养学习培养 1](#_Toc38274995)

[2. 存在问题与解决方案 3](#_Toc38274996)

[2.1. 存在的主要问题 3](#_Toc38274997)

[2.2. 解决方案与可行性研究 3](#_Toc38274998)

[3. 前期任务完成度与后续实施计划 13](#_Toc38274999)

[参考文献 14](#_Toc38275000)

## 

## 1. 毕业设计的进展情况

### 1.1. 课题工作完成情况

本项目由四位罗格斯大学在读学生共同完成，由于个人技能不同进行不同模块的分工。课题完成情况应从四个角度分析：首先，操作树莓派及连接摄像头的硬件支持同学，目前已经完成了树莓派实时录像，抽样采取图片提供真实的测试集。其次，负责深度学习模块的同学目前已经大致完成了所有的预定目标。目前可以实现的功能有，根据停车场划分不同的车位并且进行编号。对单一车位进行车牌识别，目前准确率可以达到90%。但是目前尚未能完成绑定车牌号与车位。然后，由我负责的服务器传输数据模块基本上已经建立起了基础的构架。由于没有真实的数据进行操作，目前数据库还存在人工添加数据测试调试的情况。服务已经搭载在云服务器上，可随时访问。最后，安卓应用端的开发进程已经基本完善，目前可以做到的有排列停车信息，完成地图组件的加载和部署。尚未完成从服务器接收数据并完成预期功能显示。

### 1.2. 知识技能学习情况

学习了树莓派摄像头捕捉影像及连接Python OpenCV进行数据采样采集，输出到Python中。学习了OpenCV如何进行图片切割，根据停车位四条白线进行停车位图片剥离。深度学习方面曾经尝试霍夫变换检测。个人学习技能包括flask框架学习，MongoDB操作学习，pymongo连接数据库及操作数据库学习。Python后台程序学习，HTTP协议的复习，Linux环境下shell脚本学习。安卓方面了解了使用Flutter框架开发，安卓地图组件等。

### 1.3. 职业素养学习培养

作为一名学生，我们应该时刻了解最新的行业情况，紧随行业趋势。我们所在的环境有优秀的同学及专业的导师进行方方面面的指导，我们应该珍惜如此宝贵的资源，向更深层次，更先进的科技迈出步伐。我们的这个项目，出于观察到机器视觉的社会性火热，机器视觉的未来发展前途而敲定下来。再结合我们本科期间所学习的client/server服务框架的知识，以服务性为主的方式提供一个停车助手的APP。解决社会上停车高峰期无处可停，大型停车场自家车难找等问题。

## 2. 存在问题与解决方案

### 2.1. 存在的主要问题

（1）树莓派无法进行远程拍摄，离开电源线无法启动

由于树莓派是一块高功能性的主板制成，需要直连电源才可以开机进行录像和抽取关键帧。使用可移动电源有电压不稳定的问题，且市面上无直接购买的安全可移动电源。所以我们现在的思路是网络上寻找训练集，手机拍摄测试集。

（2）OpenALPR输出非字符串

车牌识别模块我们调用的OpenALPR，由于在Windows平台上这是一款exe软件，并不可以直接输出可供编程操作的字符串。所以还无法实现车位和车牌号进行绑定。

（3）数据库无真实数据填充，服务器任处在模拟阶段

由于程序还没有全部连接起来，服务器目前的数据库都是人为添加，模拟我们需求的字段并传送给安卓端。后端监听程序也只有一个模型，并没有真实的数据进行监听然后随时修改数据库，还处于stall状态。

（4）安卓端还没有真实的数据传输，处在模拟阶段

安卓的程序开发目前已经完善，但是只有后台服务器传输的认为添加数据进行模拟测试。

### 2.2. 解决方案与可行性研究

（1）树莓派问题

目前我们决定抛弃树莓派的实际作用，树莓派只是我们最先提出来可以固定在某停车场上方墙壁进行录像捕捉的工具，实际上我们的项目可以有许多的替代品。比如说远程传输摄像头等。考虑经费问题及问题复杂度简化，我们选择现场直接手机拍摄停车场状况然后当做我们的测试集。硬件部分暂时已经舍去，只考虑软件模拟部分。

（2）车位识别

车位识别主要考虑了两种办法，第一种是直接采用Python的OpenCV库，对图像上车位的四个顶点进行人为捕捉，然后将这些顶点信息记录在另一个文件当中。图2-2-1展示了停车车位的顶点信息，以及如何绑定这些顶点信息信息成为一个车位进行车位编号。

图片包含 游戏机

描述已自动生成

图 2-2-1

这种车位识别方法存在局限性，对于过大的停车场需要手动输入的信息太多，不利于操作人员操作，需要高额的时间成本。其二，该方法的实现只是针对于采集信息后的指定停车场，并不能通用于所有的停车场。如果还需要对其他的停车场部署我们的停车软件APP，需要工作人员重新定义一个.yml文件记录顶点位置。甚至摄像头的位置都要固定在某一个地方，在定义完顶点之后都不允许移动，否则会产生严重的识别错误或者车位偏移现象。

第二种车位识别方法是使用霍夫变换检测直线，并经过挑选可用的直线勾勒出一个停车位。霍夫变换(Hough Transform)是图像处理中的一种特征提取技术，该过程在一个参数空间中通过计算累计结果的局部最大值得到一个符合该特定形状的集合作为霍夫变换结果。霍夫变换于1962年由Paul Hough首次提出，最初的Hough变换是设计用来检测直线和曲线，起初的方法要求知道物体边界线的解析方程，但不需要有关区域位置的先验知识。这种方法的一个突出优点是分割结果的Robustness。即：对数据的不完全或噪声不是非常敏感。然而，要获得描述边界的解析表达常常是不可能的。后于1972年由Richard Duda & Peter Hart推广使用，经典霍夫变换用来检测图像中的直线，后来霍夫变换扩展到任意形状物体的识别，多为圆和椭圆。霍夫变换运用两个坐标空间之间的变换将在一个空间中具有相同形状的曲线或直线映射到另一个坐标空间的一个点上形成峰值，从而把检测任意形状的问题转化为统计峰值问题。图像中所有可能的直线像素的检测，可以通过在图像中进行边缘检测子得到，所有边缘幅值超过某个阈值的像素都可以看作是可能的直线像素。在最一般的情况下，当我们没有任何有关图像中的直线信息，因此，所有方向的直线可能通过任何边缘像素。而在现在实现中，这些直线的数目是无限的，然而，为了实际目标，只能有限数目的直线方向。直线的可能方向定义了参数K的一个离散化，因此参数q也被采样为有限数目的值。所以参数空间不是连续的，而是被表示为矩形单元，称之为累计数组(accumulator array) A,它的元素是累计单元(Accumulator cells)　A(k,q).　对于每个边缘元素，确定其参数k和q。这些参数表示了通过此像素的允许方向的直线。对于每条这样的直线，直线参数k和q的值用来增加累计单元A(k,q)的值。如果公式y=ax+b所表示的直线出现在图像中，A(a,b)的值会被增加很多次，而次数等于直线y=ax+b作为可能通过某个边缘像素的直线被检测到的数目。该方法可以较好的适应于不同的停车场，不需要人为地去定义每一个停车位的顶点在哪里，但是不代表可以实现完全的自动识别。由于摄像头角度问题，训练集摄像头及停车场都不太一样，遇到不一样的停车场任然需要人为地将图片中非停车场区域的部分进行剔除操作。然后在Python中存在霍夫变换检测，学习参数之后可以直接调用，也非常的方便。

针对以上两种解决思路，我们小组采用的策略是双管齐下。由于时间上的紧迫以及需要立刻完成前后端的连接，进行数据传输实验方便另外两个模块调整。我们决定目前采用第一个方案进行一个快速解决，然后马上进入到下一阶段。若毕业设计时间充裕，我们在完成自身学业的保障下也会对车位识别进行更多方面的考虑，选择使用更加复杂但实用性更高的模型。

（3）车牌识别及车位绑定

车牌识别我们目前采用的是OpenALPR，由于这是一个exe可执行文件，目前还无法实现字符串格式的输出然后与车位进行绑定。首先我们的车位分割及标号操作是成功的，其构想是使用cv.write将之前已经实现的车位识别根据其图片的位置，一个一个剪裁下来保存在一个文件夹里。然后根据文件名对停车位进行标号，具体实现如图2-2-2所示。

图片包含 游戏机, 截图

描述已自动生成在得到每个停车位的停车情况之后，深度学习模块的同学将每张图片放入到OpenALPR里面，得到车牌号然后与车位进行绑定再传输到服务器当中。车牌识别方面根据单个图片进行输入，能达到90%的识别准确度。存在的技术问题是还无法将OpenALPR程序里的输出调用出来成为字符串再供Python进行操作。目前已经有了大致的解决思路。第一种方案是，在OpenALPR的指导手册里面我发现可以使直接在Linux上面运行。可以由shell脚本实现将图片自动导入到OpenALPR里面进行识别然后进行输出重定向到预定文件。Python再通过读取文件的方式将指定字符串传输到服务器。第二种解决方案是根据OpenALPR的源码，直接调用其C++所写的源码完成输出到Python的过程，其工程量较为巨大而且存在修改失败的风险，但是我们也在分工进行尝试。第三种方法是直接读取OpenALPR程序运行的内存。此方法较为简单但是存在技术复杂度，我们知道程序的运行数据都是存储在内存当中，如果可以直接知道OpenALPR输出的内存编码的情况下，Python可以直接读取内存信息然后存储在变量当中。

图2-2-2 剪裁图片

我们目前准备采用的策略是方法一，我比较了解shell脚本可以在短时间之内解决问题而且不存在任何修改程序的风险。但是我们小组的其他成员也在尝试其他两种方法实现数据的读取。在下次里程碑我们将根据不同方法的表现情况决定具体采用的方法。

（4）数据库及服务器框架

数据库我们考虑过采用关系型数据库如MySQL还是非关系型数据库如MongoDB。基于问题复杂度考虑，我们选择使用适合我们目前此项目的非关系型数据库MongoDB。如果后期会考虑增加用户信息收集的话，我们决定还是采用MongoDB一样可以实现并且可以简化问题。服务器我们采用的Python语言支持flask和Django等框架。由于小组成员之前曾完成过一个flask框架实现的网页，我们决定采用flask降低问题的复杂程度并且可以提供更好的服务器支持。如何使用Python操作MongoDB是该模块设计的重要问题之一，其核心代码如下：

import pymongo

class park\_db:

def \_\_init\_\_(self,host,port):

self.host=host

self.port=port

self.client=pymongo.MongoClient(host=self.host,port=self.port)

self.db=self.client.park

self.collection=self.db.parklots

self.collection\_client=self.db.backup

def all\_init(self,park\_dic):

self.collection.delete\_many({})

self.collection.insert\_many(park\_dic)

# def back\_write(self,park\_dic):

# self.collection\_client.delete\_many({})

# self.collection\_client.insert\_many(park\_dic)

#

# def back\_read(self):

# all\_list = []

# results = self.collection\_client.find().sort('park\_no', pymongo.ASCENDING)

# for result in results:

# del result['\_id']

# all\_list.append(result)

# return all\_list

def update\_one(self,updated\_one):

condition={'park\_no':updated\_one['park\_no']}

self.collection.update\_one(condition,{'$set':updated\_one})

def update\_many(self,updated\_many):

for i in updated\_many:

self.update\_one(i)

def find\_all(self):

all\_list=[]

results=self.collection.find().sort('park\_no',pymongo.ASCENDING)

for result in results:

del result['\_id']

all\_list.append(result)

return all\_list

def find\_empty(self):

empty\_list=[]

results=self.collection.find({'status':False})

for result in results:

del result['\_id']

empty\_list.append(result)

return empty\_list

def find\_yours(self,plate):

result=self.collection.find\_one({'plate':plate})

if result==None:

return "Wrong plate number"

del result['\_id']

return result

#testing example

park=park\_db('127.0.0.1',27017)

我选择采用一个class封装起来，将整个停车场视为一个对象，方便后期服务器写入数据，返回数据调用。然后就是flask服务器模块，其代码如下：

from flask import Flask,request

import db

from flask import jsonify

last\_update=None

app = Flask(\_\_name\_\_)

#list\_a is the old one, list\_b is the new one

def get\_diff(list\_a,list\_b):

diff\_list=[]

for i in range(len(list\_a)):

if list\_a[i]['status'] != list\_b[i]['status'] or list\_a[i]['plate'] != list\_b[i]['plate']:

diff\_list.append(list\_b[i])

else:

continue

if diff\_list==[]:

return "No update information"

else:

return diff\_list

# @app.route('/post', methods=['POST'])

# def post():

# uid = request.form['uid']

# name = request.form['name']

# print('uid: %s, name: %s' % (uid, name))

# return 'OK.'

@app.route('/',methods=['GET'])

def all\_send():

global last\_update

last\_update=db.park.find\_all()

return jsonify(last\_update)

@app.route('/empty',methods=['GET'])

def empty\_send():

results=db.park.find\_empty()

return jsonify(results)

#getting URL eg. http://127.0.0.1:5000/yours?plate=D33LOP

@app.route('/yours',methods=['GET'])

def yours\_send():

plate=request.args.get('plate','Flask')

results=db.park.find\_yours(plate)

return jsonify(results)

@app.route('/update',methods=['GET'])

def update\_send():

global last\_update

new\_update=db.park.find\_all()

print(last\_update)

print(new\_update)

results=get\_diff(last\_update,new\_update)

last\_update=new\_update

return jsonify(results)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

目前服务器已经全部搬运到阿里云服务器上，方便安卓端模块设计的同学进行访问和测试。

（5）后台程序管理，后台监听程序

之前已经提到深度学模块暂时无法返回一个完整的数据，所以监听程序只是一个模型，还无法实现对接。但是MongoDB和flask的运行都会占用一个完整的terminal，添加合适的脚本会更加利于服务器的管理。服务器采用的系统是CentOS 7, 由于yum不支持MongoDB server的直接下载浪费了很多时间。其初始化进程脚本如下：

#!/bin/bash

pkill mongod

pkill python3

rm -rf /root/logs/db\_log.txt

rm -rf /root/logs/server\_log.txt

初始化进程和日志记录之后，服务器会启动MongoDB服务，flask服务还有监听数据库修改服务，其启动脚本如下：

#!/bin/bash

log\_path=/root/logs

pycode\_path=/root/Capstone\_server

nohup mongod > ${log\_path}/db\_log.txt 2>&1 &

com\_1=$(ps | grep "mongod")

if [ -n "${com\_1}" ]

then

echo "mongodb service start!"

else

echo "mongodb start failed, please try again"

exit

fi

nohup python3 ${pycode\_path}/run.py > ${log\_path}/server\_log.txt 2>&1 &

com\_2=$(ps | grep "python3")

if [ -n "${com\_2}" ]

then

echo "Falsk start successfully"

echo "Access the web from ip address 39.107.76.62"

else

echo "Flask start failed, please check codes and restart"

exit

fi

#python3 ${pycode\_path}/listener.py > /dev/null 2>&1 &

目前服务器设计情况基本如上所示，若能保障完成自身学业的情况下，后期会对服务器采用压力测试，访问限制等安全方面强化措施。

（6）安卓应用端程序网络连接模块

安卓应用程序基于Google的flutter框架进行开发，和服务器的通信机制基本上分为三种。第一种是使用flutter自带的dart原生网络请求HttpClient，第三方网络请求Http以及flutter当中的Dio。Dart语言出现的比较晚，吸取了前辈语言的经验，Dart语言简简单单的几行代码即可搞定GET请求，也属于非常现代化了。但是Dart原生请求存在一定的缺点，就是它对post请求的支持不是非常完美。第二种网络模式是库http，该方法相对于dart来说又是一种进步。相对于dart来说也更加简洁明了，但是在post模块上他会把Map型的body自动转化成表单提交。Dio相对于前两种方式而言在使用时更为简洁明了，并且支持Restful API以及拦截器和Cookie的管理等功能所以在网络模块的搭建上使用Dio。Dio其实还相当于现在安卓非常流行的OKhttp+Retrofit，能够节省开发人员大量时间更高效的专注于其他方面的工作。根据我们的项目需求，安卓端会按照两分钟一次的频率向服务器发送一次停车场情况的请求，然后更新成为实时数据。更新频率可根据实际运行情况及用户使用情况进行更改。

（7）安卓地图组件

我们的项目还需要获取用户的地理位置及停车位置方便用户找到自己的爱车，所以我们的移动应用端还需要加入地图模块的组件。可考虑的地图模块由Google支持的Googlemap，但是Google提供的API在国内市场并不开放，而且是付费制度，只能作为一个备选方案。第二个可考虑地图组件是高德地图，由于现在的flutter推出的插件版本是0.0.1，考虑到性能稳定性因素，我们也不考虑使用高德地图。最后对于地图模块的开发使用的是基于Leaflet的地图插件，使用的为瓦片地图，在前端用户使用时进行展示渲染时，兼容性相对来说也较好。地图组件的代码如下：

class SptsMap extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: new AppBar(title: new Text('Leaflet Maps')),

body: FlutterMap(

options: MapOptions(

center: LatLng(40.5185, -74.4551),

minZoom: 15.0,

),

layers: [

new TileLayerOptions(

urlTemplate: "https://api.tiles.mapbox.com/v4/"

"{id}/{z}/{x}/{y}@2x.png?access\_token={accessToken}",

//瓦片地图的URL

additionalOptions: {

'accessToken': 'pk.eyJ1IjoibWluZ21pbmdwIiwiYSI6ImNrOHdoYnp3dDBuNTgzZm82OTE3NTU2bWwifQ.Jr9pY9uxuij-6J78hXnnmw',

'id': 'mapbox.streets',

},

//urlTemplate: "https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",

//subdomains: ['a', 'b', 'c']

),

new MarkerLayerOptions(markers: [

new Marker(

width: 45.0,

height: 45.0,

point: new LatLng(40.5185, -74.4551),

builder: (context) => Container(

child: IconButton(

icon: Icon(Icons.location\_on),

color: Colors.red,

iconSize: 45.0,

))

]) }

安卓端现在完成了基本程序的开发，可以在程序主页面显示停车场的停车状况，然后加载地图信息。根据用户所输入的车牌号码在系统中搜寻自己车辆的位置，将返回位置信息以及车辆所在停车位编号信息。

## 3. 前期任务完成度与后续实施计划

前期任务的完成情况目前来看还是效率不错，在树莓派上面我们已经实现最开始指定好的计划，但是需要在后期更改实施计划，因为之前的树莓派拍摄并不支持远程供电。在深度学习模块的开发上面，我们前期完成了最基本的任务指标，还差将车牌号与车位号进行绑定，预计来说最近两天就会完成深度学习模块的基本设计。但是我们也在考虑后期使用更加复杂算法和模型。OpenALPR只是一个提供的车牌识别软件，我们希望在后期实现一个由Python语言编写，借鉴其他CNN模型自己编写的车牌识别程序。大量的训练集我们已经在目前的学习阶段从网上找到合适的。还需要硬件支持的同学去搜集我们最后要实现的现实测试集。深度学习目前完成的功能还不具备很强的各方面环境适应性。比如说移动摄像头或者更换停车场都需要工作人员重新进行编程处理。当然在时间如此紧迫的环境之下，还有自身知识储备和实践能力不足的当下，能完成基本的任务指标已经达到了最初的目的。五月份希望能对、整个项目再进行一次全方面的优化，比如说停车场识别，希望能够达到通过霍夫变换检测或者别的识别算法，通过不同位置的摄像头对不同的停车场进行一个自动车位识别功能。然后服务器模块我们目前的完成度还是很低的。实现基本的功能点，能做到和安卓端进行数据交互并根据深度学习模型监听停车位变化情况修改数据库提交更新请求。目前可以考虑的是给服务器增加服务运行状况的监听，之前编写的程序只包括了服务启动监听，检测是否打开MongoDB或者flask的服务。虽然我们的项目不一定能做到非常大的用户规模，但是基于一个开发人员的严谨工作态度，我人为增加压力测试，安全问题解决，提高服务器安全程度还是有必要的。而且在未来的一周之内我们要完成初步对接。实现从停车场监控视频流里面提出关键帧，到深度学习网络里面经过识别提供有效数据到服务器。然后服务器及时修改数据库，再到安卓端访问服务器，服务器读取数据库数据返回给安卓端的整个流程实现。之后再对各个组件进行优化。

## 参考文献

[1] Flask教程，w3cschool [<https://www.w3cschool.cn/flask/>]

[2] Flutter Documentation，Google [<https://flutter.dev/docs>]

[3] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick

，Mask R-CNN. The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2961-2969

[4] 学习Web 开发的最佳实践，HTTP |

MDN – Mozilla[<https://developer.mozilla.org/zh-CN/docs/Web/HTTP>]

[5] 计算机网络技术在项目信息管理中的运用.  
《图书馆理论与实践》.被北京大学《中文核心期刊要目总览》收录PKU.被南京大学《核心期刊目录》收录CSSCI.2012年3期.何琪.

[6] OpenALPR [<https://github.com/openalpr/openalpr>]

[7] MongoDB Manual [<https://docs.mongodb.com/manual/core/document/>]

[8] P.V.C. Hough,Machine Analysis of Bubble Chamber Pictures, Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959.