SMART PARKING USING MACHINE LEARNING

Bachelor of Engineering

in

ELECTRONICS AND TELECOMMUNICATION ENGINEERING

by

Rajiv Iyer	115A2032
Sree Ganesha Chellapa	116A2102
Navin Subbu	116A2105
Muthu Sumathy Thevar	116A2112

Under the Guidance of:

Prof. Hema Raut

Department of Information Technology

SIES graduate School of Technology

2019-2020

" SMART PARKING USING MACHINE LEARNING "

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

OF THE DEGREE OF

BACHELOR OF ENGINEERING

In

ELECTRONICS AND TELECOMMUNICATION ENGINEERING

By

Rajiv Iyer	115A2032
Sree Ganesha Chellappa	116A2102
Navin Subbu	116A2105
Muthu Sumathy Thevar	116A2112

UNDER THE GUIDANCE OF

Prof. Hema Raut

Department Of Electronics and Telecommunications

SIES GRADUATE SCHOOL OF TECHNOLOGY
NERUL, NAVI MUMBAI – 400706

ACADEMIC YEAR

2019 – 2020

CERTIFICATE

This is to certify that the project entitled **" SMART PARKING USING MACHINE LEARNING "** is a bonafide work of the following students, submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering** in **Electronics and Telecommunication Engineering.**

Rajiv Iyer	**115A2032**
Sree Ganesha Chellappa	**116A2102**
Navin Subbu	**116A2102**
Muthu Sumathy Thevar	**116A2112**

Guide (Internal)	Head of Department (Electronics and Telecommunication)	Principal (S.I.E.S GST)

PROJECT REPORT APPROVAL

This project report entitled **" SMART PARKING USING MACHINE LEARNING "** by following students is approved for the degree of **Bachelor of Engineering** in **Electronics and Telecommunication Engineering**.

Rajiv Iyer	**115A2032**
Sree Ganesha Chellappa	**116A2102**
Navin Subbu	**116A2105**
Muthu Sumathy Thevar	**116A2112**

Name of External Examiner : _____

Signature : _____

Name of Internal Examiner : _____

Signature : _____

Date :

Place :

DECLARATION

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Rajiv Iyer	**115A2032**	_____
Sree Ganesha Chellappa	**116A2102**	_____
Navin Subbu	**116A2105**	_____
Muthu Sumathy Thevar	**116A2112**	_____

Signature

Date:

ACKNOWLEDGEMENT

It gives us immense pleasure to thank Dr.Vikram Patil, our Principal for extending his support to carry out this project. We also thank to our Hod Prof.Preeti Hemnani for her support in completing the project. We wish to express our deep sense of gratitude and thank to our Internal Guide Prof.Hema Raut for her guidance, help and useful suggestions,which helped in completing our project work in time.

Also we would like to thank the entire faculty of Electronics and Telecommunication department for their valuable ideas and timely assistance in this project, last but not the least, we would like to thank our teaching and non teaching staff members of our college for their support, in facilitating timely completion of this project.

Project Team

Rajiv Iyer

Sree Ganesha Chellappa

Navin Subbu

Muthu Sumathy Thevar

ABSTRACT

In a country where it took 60 years to acquire 100 million vehicles but added another 100 million in just the next ten years, free and illegal parking has become both a serious urban planning and public health issue.

We are building a system in which we will be making use of live feed from CCTV cameras or pre-recorded video and detect the entry/exit of the vehicle using machine learning and Image Processing algorithms. Detection of number plate of the car is done using Optical character recognition (OCR) or Automatic License Plate Recognition (ALPR) and also store the image of vehicle in the database for ensuring security. If feasible the system will try to distinguish the entry/exit of vehicle it is determined whether motion of vehicle is towards camera or away from camera i.e. zooming in or zooming out the license plate. After retrieving the license number on the plate and an image of the car, the system will interface with an application module called 'Smart-Park' application. This module will perform several functions such as looking up the license plate in a database, determining if a car belongs to a resident (aka member), tracking the entry and exit of all the cars and finally using the historic activity for smart spot allocation. The application uses Google Firebase cloud database and storage for persisting all the data. Using the firebase allows the application to be truly distributed and concurrent in real-time. The activity data captured by the application can be used in future to develop a real-time security notification system depending on the entry-exit timestamp of a vehicle. The smart park application allows for pre-designation of parking spots to members and allocate them depending on the vacancy. It also allows an efficient and optimized parking system to be implemented using the intelligence available based on the parking behavior (early/late arrivals) of it's members. The vacancy of a parking lot is checked using IoT, in which the presence of a car in a parking lot is detected using a proximity sensor/ distancing meter. The vacant spaces data is fed into the Real-time database in Google Firebase.

Contents

List of Figures

Chapter 1

INTRODUCTION

License plate numbers are being used to uniquely identify a vehicle. License Plate Recognition (LPR) system plays an important role in many applications like electronic payment system (toll payment and parking fee payment), to find stolen cars, traffic surveillance. For instance, in malls the time difference between the time, the car entered and left the parking lot is used for calculating the parking fee. LPR is convenient and cost efficient as it is automated. In this project we will be recognizing the license plate of the vehicles, time stamping them and carry out the prediction of their entry and exit time. When a vehicle enters the boom-barrier, license plate and make of the vehicle is automatically recognized and a database is created. On exiting, the same are validated. Now in housing societies or commercial complexes, a pre-existing database of residents/employees containing the above info along with name and parking spot number will be used to validate. Parking spots will be allotted to every individual at start and remaining spots will be vacant for visitors. During setup of the system, the total available spots can be configured as Members and Outsiders/Visitors only spots. When a particular employee doesn't reach the office campus at the intended time, his/her parking space will be then be allotted to someone else for that day provided a parking spot is available. The parking spaces will consist of UDM sensors linked to NodeMcu/ESP32 Wi-Fi board that detects the presence of car parked. This will give a confirmation that the car has been parked successfully in that parking lot. When the car is moved back from the space, the slot is retained back in the system.

If a visitor approaches the boom barrier, system will perform the usual process of detection plus it will allot the parking space number displayed on the boom barrier. The feature of prediction of entry and exit time is critical for residential societies. So, if an unregistered vehicle parks anonymously or parks for out of bounds limit, then the security guards will be alerted. Certain residential societies have parking spots available for unregistered vehicles of visitors/outsiders.

1.1 Motivation

With growing popularity of Smart Cities, there is always a demand for smart solutions for every domain. There are multiple domains in a smart city and Smart Parking is one of the popular domains in the Smart City Parking industry has seen a number of innovations such Smart Parking Management System, Smart Ge Control, Smart Cameras which can detect types of vehicle, Smart Payment System, Smart Entry System and many more. With increase in development of affordable cars in the modern world, everyone has access to private vehicles and thus increases the requirement of free space for parking them. In a metropolitan city like Mumbai, there's a lack of free space. To overcome this issue, multi-storey parking areas were created. But in coming years even this system will become inefficient. Thus, a new system is required to overcome this issue.

1.2 Objectives

The main objective here is to provide a system which provides a real-time detection of cars and intelligent processing of their parking activity which includes spot allocation, tracking of their entries and exits. This project automates the location of vacant spots and intelligent allocation of an available spot in an efficient manner. It helps to resolve the growing problem of traffic congestion as search time for parking is reduced thereby potentially reducing traffic. User experience is enhanced as finding a parking spot which is vacant through real-time detection of license via ML technology and smart trend analysis of the parker's past behavior can reduce human error and decrease management costs. It also helps Improve safety for parking lot employees and security guards as the system will have automated identification of the vehicle owners through a real-time database.

1.3 Scope

It is found that in housing complexes and office buildings, parking lots used are not occupied for the whole day. So, if a system is developed to find the duration for which the car stays in the plot, the same lot can be allocated to a visitor when it is unoccupied. Also, a physical handbook has to be maintained to record the arrival and departure of the vehicles. This increases the error probability of wrong data entries and also missing out some entries. We have proposed the smart parking system using the Internet of Things (IoT) along with a smart parking spot allocation system backed by member registration and member activity database that can be part of a solution for the parking problem. This system helps in organizing the parking lot and helps the driver to reach their parking spots easily as they known which space is vacant. The parking space can be detected using an Infrared sensor that connects to the ESP12-E (NodeMCU) module that was programmed through Arduino IDE. This system will help security to know the availability of parking space in real time.

1.4 Organization of the report

Chapter 1 briefs about the problem statement describing the parking problems and how it affects the serious urban planning and how certain technical solutions and their application could help in reducing the search traffic. The scope of the proposed system is also discussed to understand the practicality of the solution and certain applications.

Chapter 2 is about understanding various solutions that preceded to address this problem statement and understanding its efficiency. More or less a dozen research paper references were studied to understand, highlight and compare the uniqueness of each solution presented by the respective authors. Our perspective on each paper is presented without obscuring important details.

Chapter 3 is all about our perception of the initial solution thought out to the problem statement. We create an efficient flow-chart diagram and methodology i.e. steps to approach a practical solution and a prototype to back it up. The nature of this application is network based - wireless network and our take on the new perspective of the solutions from the literature survey and the user end input.

Chapter 4 deals with all the technical aspects that lie beneath the theory of the solution proposed with both hardware and software and why we chose the same, keeping in mind the basic requirements.

Chapter 5 explains the details of the implementations and workings of the system while explaining the concepts involved in all the electronic components and how they act cohesively in the proposed solution; understanding the harmony between the hardware functionality with the software backing the operation. The primary component i.e. the NodeMCU ESP 8266, its working, specifies its necessity for our project along with other components like a microcontroller and other passive components used. This chapter also covers the implementation and business logic of the lot occupancy management software and the backend database architecture with it's implementation details.

Chapter 6 summarizes the working and the results of the tests performed and discussions regarding the implementation details and the analysis of the performance measures.

Chapter 7 discusses more on the practicality and feasibility of the solution. The prototype of our project used a lot of sensors and microcontrollers-without an efficient power supply - taking into consideration that we focused more on the "feedback control" loop that made our solution unique and if focused more on the research and development, the project hardware and its intricate details and efficiency would be improved.

Chapter 2

Review of Literature

Giuseppe Amato from ISTI-CNR [1] and his colleagues developed a smart parking system using hardware and software based on IoT, and mobile application which is used by the driver can easily check parking information and use mobile transactions to pay the parking fee. Here they have used real time car occupancy detection which uses convolutional neural Network (CNN) which is running on-board of a smart camera with limited resources. The results show that it is robust to light condition changes and even from shadows. Here detection of car parking occupancy is done by deep learning. they exhibit very high accuracy, even in presence of noise. This method is also robust to non-standard parking behaviors, such as cars occupying multiple parking lots. In that case, our solution tends to classify both slots as busy. The goal of our study is to improve the parking process by reducing the time that is required to park a car.

Rachapol Lookmuang's [2] experiments show that Convolutional Neural Networks (CNN) are effective enough to be applied for finding the vacant parking lots using computer vision. The accuracy of the network increases with no of images used for training the model. They have tested their approach using a test set produced with a camera placed in a different place (perspective and viewpoint) and their approach reduces errors caused by natural factors such as light etc. This real time parking system can not only tell if a parking space is available or not but it can all locate the free parking space for the car to be parked. In this system we can also pay the parking fee through mobile which reduces the time for standing in queue and paying the fee for parking. The system will detect the vehicle plate number and use it to inform the driver where his/her car is parked and also for the purpose of security monitoring. The system works accurately even where the parking area have low light. They concluded that CNNs have good generalization capabilities in predicting parking status when tested on a dataset completely different from the training dataset.

Leonardo Dominguez [3] used an open source platform is taken as base and by improving it to evaluate only objects that are moving. The algorithms run on a distributed platform that operate multiple cameras and sensors, to support security management. The initial outputs gave partially good performance, reaching near a real-time LPR detection, even for high resolution images. On the other side, the true positives are less than expected. management. The first results are reasonable in performance, reaching near a real-time LPR detection, even for high resolution images. The first results are reasonable in performance, reaching near a real-time LPR detection, even for high resolution images. The

algorithms required to have the plates more focused even if the text is readable, so they propose using smarter algorithms for better output.

Thanh Nam Pham [4] introduces a novel algorithm that increases the efficiency of the current cloud-based smart-parking system and develops a network architecture based on the Internet-of-Things technology. This paper proposed a system that helps users automatically find a free parking space at the least cost based on new performance metrics to calculate the user parking cost by considering the distance and the total number of free places in each car park. Cost can be reduced by saving time which will reduce emission of fuel and it will save the environment. This system realizes that if we use the percentage of free spaces in each car park as a parameter for planning with regard to forwarding the users, the waiting time of the user for the service will be greatly reduced compared with that in an ordinary network. Thus this system is an efficient system to reduce time and money at the same time.

Evan [5] has explained how to use tensorflow models for training a custom object detection model. In his githubrepository he has given a step by step guide for training a model. Along with this , he has noted down the flaws in the process and also how those flaws can be tackled. The method is applicable for any particular object that is needed to be detected using Machine Learning. we have incorporated his method for training our model for license plate detection using the photos that we've captured.

Chapter 3

Flow of System

3.1 Flow Diagram

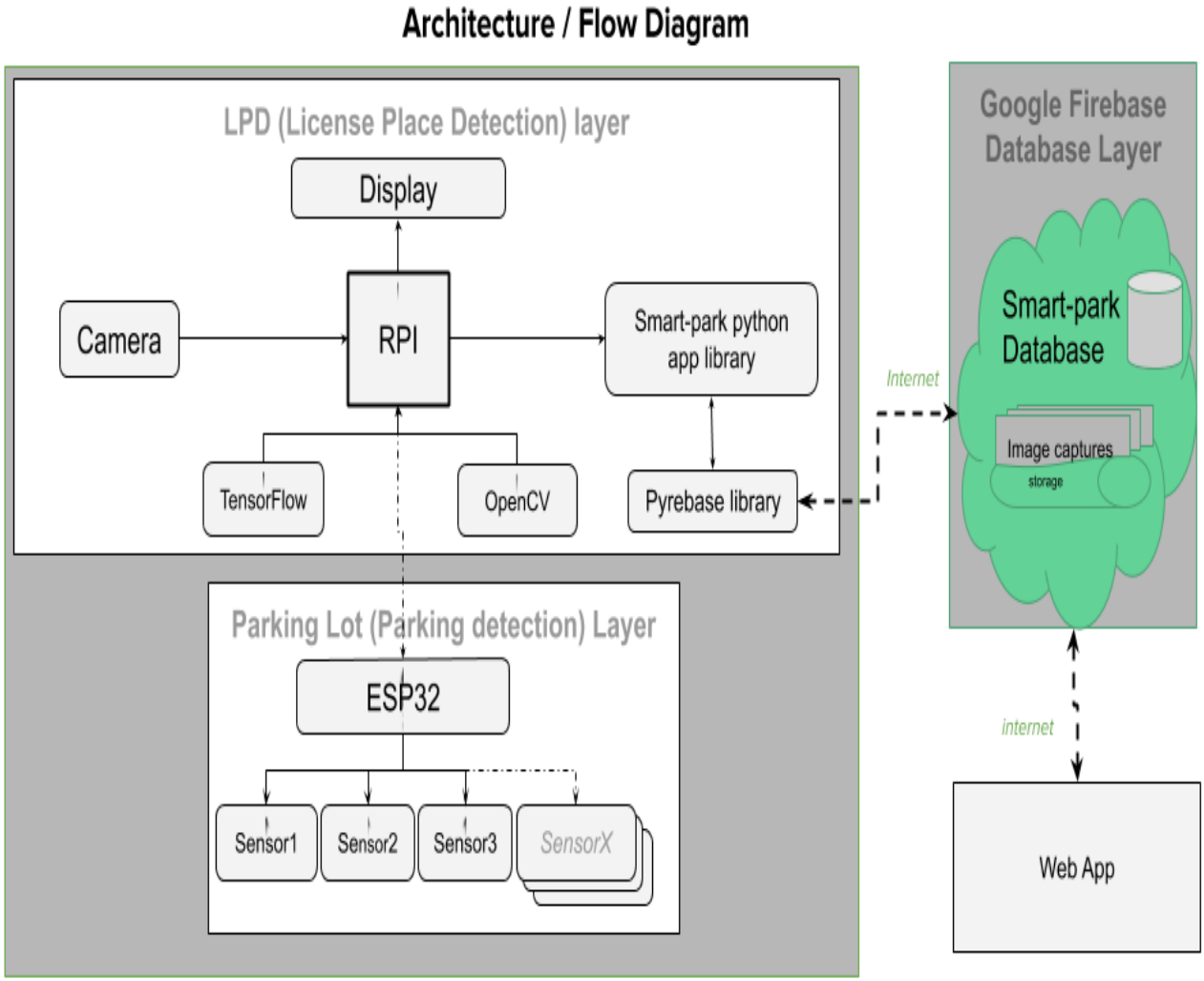


Figure 3.1: Flow Diagram

3.2 Methodology

This project aims to take all the efficient aspects of the projects proposed in the literature survey and integrate it with an efficient feedback to be part of a solution for the parking problem.

3.2.1 License Plate Detection

When the car enters the gate of the parking space the camera installed in front of the gate detects the car. We have trained the system to identify the license plate by using Tensorflow. Tensorflow is a free and open source software library for differentiable and dataflow programming across various range of task. It is basically used for machine learning application. Thus by using this we train our model to detect license plate from a live video.

In order to detect the license plate, first we will resize the original image to the required size making sure that the number plate remains in the frame after resizing. Then convert the image to gray scale which usually speeds up the following process and also no need to deal with the colour details as it is not required when processing an image. It is necessary to remove the unwanted details of the image as our main focus is to find license plate. Hence, useless information (noise) can be removed by using bilateral filter (blurring). Next step is to perform Edge detection which displays only the edges that have the intensity gradient between the minimum and maximum threshold values. This can be done using the canny edge method. Once the edge detection is done, we need to find the closed surface area in the image in order to extract the license plate for which contours need to be differentiated. While finding the contours we may obtain several results, from which only 10 contours will be filtered. These results may consist of closed surface contours which involves license plate number since it is a closed surface. So we filter those contour images by setting it up in a loop and simultaneously check for a rectangle shape (four sided closed figure). Once detected the contour is saved in a variable and a rectangle box is drawn around the license plate. Now after detection of the license plate, the remaining unnecessary information can be removed by masking the entire image except the area of the number plate. Basically Character Segmentation is done where we will be cropping out the license plate from the image. After identifying the license plate and extracting the license plate from the image, extraction of the characters from the license plate will be processed using OCR (Optical character recognition). OCR initially removes unnecessary parts of the image by converting the colour image into black and white image. Thus the image will be left with black and white text. In the next step, OCR does character recognition by scanning pixel by pixel and comparing it with a set of database stored. This gives us the characters of the number plate.

3.2.2 Google Firebase

Firebase provides a real-time database and backend as a service. Firebase in this project is used to store the data of prototype. Firebase here acts as a cloud. Firebase cloud is

a NoSQL document database that lets to easily store, sync, and query data for our web apps. Firebase here is also used as a web application development platform.

Firebase allows the application to exchange data as Json docs which also is natively supported by python collection data types like Dictionary and List.

The characters obtained from OCR will be stored in the Real time database via RPI microcontroller which is connected to Wi-Fi system. Along with the license plate number, details such as date and time of the car arrival and exit will also be recorded.
We will send data to webserver for looking up the availability of space for vehicle parking. Here we are using firebase as lot database to get the parking availability data. For this we need to find the Firebase host address and the secret key for authorization.

3.2.3 Parking Space Allotment (Smart Parking)

Using 'Proprietary Smart Parking Algorithm', entry time of resident/employee car in the parking space will be recorded. Initially the residents/employees will be pre-assigned a particular lot in the parking area. If the resident/employee doesn't come reporting time and the outsider spots are filled, his/her parking lot will be given to an outsider/visitor. This process is called time stamping. A timestamp is the current time of an event that is recorded by a computer. Thus the computer maintains accurate current time, calibrated to minute fractions of a second. By studying the arrival and exit time of the car we can check whether the parking lot is vacant or not. As soon as the car enters parking space, the ultrasonic sensor which is used to detect if the parking slot is available or occupied, sends the data to ESP8266 accordingly. If the parking space is occupied it sends '1' and if the space is vacant it sends '0'. Simultaneously when the car enters the parking space, arrival time is recorded. This information along with license plate number is stored in the database. Similarly, the exit time of the car is also noted and updated in the database.

All the functionality of the above described 'smart parking' logic is implemented as a python application module (smartpark app) that can be invoked with three basic python functions:

- **IsCarParked(main_license) -**

  Call to check if a car is IN or OUT.
  String Main_license – Parameter containing license number.
  Return True if Main_license is already parked.

- **RegisterCarEntry(main_license, entrytimestamp,local_carimage) –**

  Call to register entry of vehicle
  String Main_license – Parameter containing license number.
  datetime.timestamp entrytimestamp. – Parameter containing time car seeking entry.
  Returns the parking spot the car is allotted to park or None if no spots are available or error occurred.

- **FindNextAvailableSpot(lic, entry_timestamp) –**

  Call to find out next available spot.

  String lic – Parameter containing license number.

  datetime.timestamp entrytimestamp. – Parameter containing time car seeking entry.

  Return Spot Number or None if no spot is available or error occurred.

- **RegisterCarExit(main_license, exittimestamp) –**

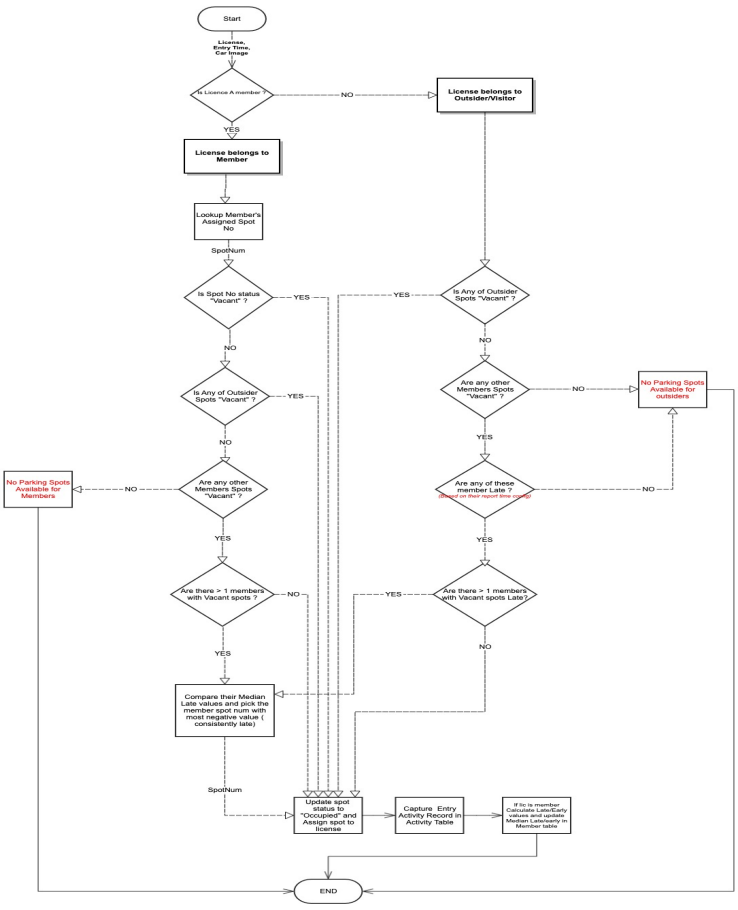
  Call to register exit of vehicle.

  String Main_license – Parameter containing license number.

  Datetime.timestamp exittimestamp – Parameter containing time car exited the lot.

  Returns the parking spot number that was vacated or None if no spots were vacated or error occurred.

Most of the smart allocation logic is implemented in RegisterCarEntry() function and FindNextAvailableSpot() here is a flowchart of the smart logic implemented by this method:



Chapter 4

System Requirements and Description

4.1 Hardware and Software Requirements

To implement the proposed system, it was required that we need electronic components and a suitable bridge to connect us to the medium processing the data i.e. the software and the hardware. Selecting any random microprocessor would affect the budget of the project – but at the same time considering the efficiency and power handling capacity required us to shortlist nodemcu esp8266 after considering its specifications from the data sheet For the software requirements we needed a suitable Ide to process all the data.

4.2 Hardware Description

Raspberry Pi Camera: This 8mp camera module is capable of 1080p video and still images that connect directly to your Raspberry Pi. This is the plug-and-play-compatible latest version of the Raspbian operating system, making it perfect for time-lapse photography, recording video, motion detection and security applications. Connect the included ribbon cable to the CSI (Camera Serial Interface) port on your Raspberry Pi.

Raspberry Pi: Raspberry Pi is an open source platform for IoT and Credit Card size computer. It is used for projects where it needs low power computing and compact system design. It has its own 32bit microprocessor and all other components required for a computer in a compact design.

Ultrasonic distance measure: An Ultrasonic Sensor is a device that measures distance to an object using Sound Waves. It works by sending out a sound wave at ultrasonic frequency and waits for it to bounce back from the object. Then, the time delay between transmission of sound and receiving of the sound is used to calculate the distance.

NodeMCU ESP 8266: The central unit used for collecting the sensor output pulses and processing the values, a nodemcu esp8266 is used. The particular hardware was chosen due to cost efficiency and the ability to transmit data over one of its four Wi-Fi modes and connect over LAN. The faster Wi-Fi allowed us to use the data to transmit over a web-interface. Its digital pins are used to take the input from the sensor since the output from the sensor is in the form of pulses. Connected via USB, providing the power supply to the board allowed us to test the computation and analysis of the data. The Access point (AP) mode of the board allowed us to use one of the boards as the central node.

4.3 Software Tools and Libraries Description

Anaconda Navigator: Anaconda Navigator is an Open Source graphical user interface for conda package and environment manager. It is simple Software interface to work with different platforms in Python.

Spyder: The Scientific Python Development Environment in short Spyder is a powerful scientific environment included in the Anaconda Navigator. This is an environment to code and perform them.

VS Code:- Visual Studio Code is a powerful IDE environment provided by Microsoft for Windows, Linux and macOS platforms. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

VS Code:- Visual Studio Code is a powerful IDE environment provided by Microsoft for Windows, Linux and macOS platforms. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

Jupyter Notebook:- The Jupyter Notebookis an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

Firebase Python Lib:- A simple python wrapper for theGoogle Firebase API.

Tensorflow: Tensorflow is an open source Machine Learning and Neural Network platform. With flexible tools and ecosystems and a huge community, it is currently leading in the field of Machine Learning. It provides algorithms and models for ML. there are many pretrained models and also gives facilities for custom model training.

Arduino Ide: The Ide used for the microcontroller nodemcu esp8266 is an Arduino Ide where the basic code for the amount of water flowing through the tap using the flow rate is written. For various nodemcu(s) it was required that the code of same logic be copy-pasted to ensure the basic value that it can measure using different sensors at a given time frame without producing any unnecessary delays in the processing of the system and compiling and uploading the program occupying an efficient amount of memory space.

Web App: The web-interface was designed using various languages and integrating it on notepad ++. The languages used are JavaScript, CSS and Google firebase. The web-interface was modified from the template and the program and logic for the values to be processed and displayed was added in the sections of the web-interface's source program where we can handle all the back-end operations and provide the user with the data he needs.

Chapter 5

IMPLEMENTATION

5.1 Existing System

The existing smart parking is built using an ultrasonic sensor to detect vehicle presence in the allocated lot for the car. When a car reaches the boom-barrier, a sensor senses the presence of a car and opens the gate automatically. The ESP8266 NodeMCU will be used here as the main controller to control all the peripherals attached to it. As the car enters it increments the number of cars in the parking space, and when the car exits it decrements the number of car present. There is no time stamping used in this system.

5.2 Proposed System

Our main purpose is to build a parking system to save time and to increase security. Our parking system has an organized way of allotting residents/employees a permanent parking space. People waste time in finding parking spots to park their car so we can create a system efficient enough to overcome this time barrier. When the car enters the parking area the camera detects the number plate noting the time of entry. There will be a database consisting of details of the car owner if he is a resident/employee. If an outsider wants to park the car, the system will check for free outsider designated parking spaces. By using Time Stamping we will be able to assign a resident's/employee's spot to a visitor/outsider if he/she arrives on out of bounds time. We can increase security by using Time Stamping. If time permits we will implement a alerting system wherein if the outsider or an employee has parked the car for more time than designated time, an alert message will be sent thus increasing security and reducing time throughout the process. If time permits we will implement a real-time web portal that can provide a dashboard and administration features by interfacing with google firebase.

5.3 Software Implementation

5.3.1 License Plate Detection Using Tensorflow and OpenCv

When a live video is captured in a video camera, usually the whole video is being recorded. This increase the need for more hardware storage. To overcome this, we have used Machine Learning. Using Tensorflow we trained a custom object detection model for detecting the object, in this case the license plate of a car and capture an image. On comparing R-CNN, Fast R-CNN, Faster R-CNN, SSD Mobile Lite and YOLO, we concluded that for our purpose Faster R-CNN was optimal. It was considered as it has better accuracy and speed. The hardware requirement for this is not much, most of the modern computers with a decent GPU will perform well. Faster R-CNN has 3 levels of neural network, Feature Network, Region Proposal Network (RPN), Detection Network. Feature Network is a pretrained classifier which generate good features from the image. RPN is 3-layer convolutional network, it generates bounding boxes around places with high probability of object being present. Detection Network also known as RCNN network takes input from both the above layers and generates the final class and bounding box. Both RPN and Detection Network needs to be trained.

OpenCV is used to stream a live video from the video camera installed on the boom barrier at the entrance. Usually the whole video is being recorded for security purposes but recording vehicle entry and exit is not much of a use as it will need more hardware storage which can be used for other purpose. To overcome this, we have used Machine Learning. Faster R-CNN is used for our purpose. The video captured is processed through this model, detection algorithms find the most probable locations of objects and draw bounding boxes. Each box has scores of probabilities, the overlapping boxes are eliminated based on the scores. The box with highest score is retained and others are destroyed. If the object is found, the bounding box that shows the objects passes the coordinates and the score which are then displayed in green on the live video. When the score of the detected object is greater than the desired value, an image is captured with the timestamp of when it was captured and stored in a local storage. Refer figures (ML codes) for the full code.

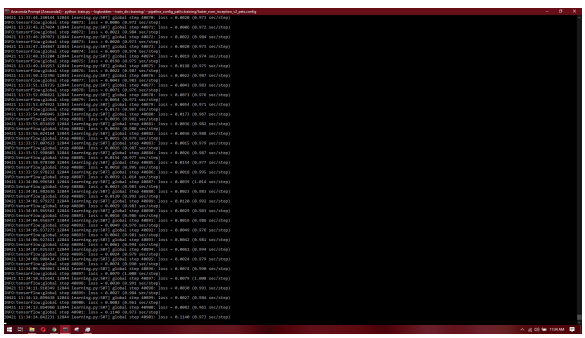


Figure 5.1: Model Training

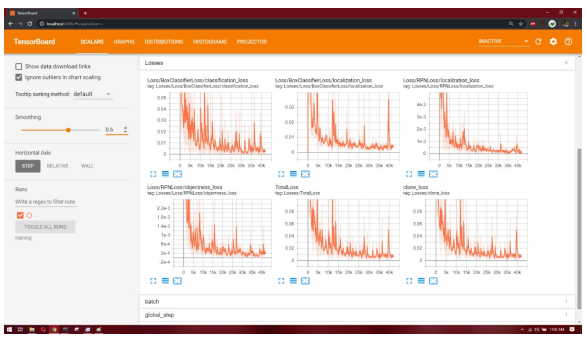


Figure 5.2: Losses Graph

```python
# Import packages
import os
import cv2
import numpy as np
import tensorflow as tf
import sys
import datetime

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")

# Import utilites and modules
from utils import label_map_util
from utils import custom_visualization_utils as vis_util
import ocr_extraction as ocr
import firebase_storage as fb_store

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'inference_graph'

# Grab path to current working directory
CWD_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used
# for object detection.
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')

#Saving Images when Object is detected
IMAGE_SAVE_PATH = os.path.join(CWD_PATH,'detected')
truth = False

# Number of classes the object detector can identify
NUM_CLASSES = 1

## Load the label map.
# Label maps map indices to category names, so that when our convolution
# network predicts `5`, we know that this corresponds to `king`.
# Here we use internal utility functions, but anything that returns a
# dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
                                                            max_num_classes=NUM_CLASSES,
                                                            use_display_name=True)

category_index = label_map_util.create_category_index(categories)

# Load the Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)

# Define input and output tensors (i.e. data) for the object detection classifier
# Input tensor is the image
image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

# Output tensors are the detection boxes, scores, and classes
# Each box represents a part of the image where a particular object was detected
detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

# Each score represents level of confidence for each of the objects.
# The score is shown on the result image, together with the class label.
detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')

# Number of objects detected
num_detections = detection_graph.get_tensor_by_name('num_detections:0')

# Initialize webcam feed
video = cv2.VideoCapture(0)
ret = video.set(4,1280)
ret = video.set(3,720)

```

Figure 5.3: Tensorflow Initialization

```
# Initialize webcam feed
video = cv2.VideoCapture(0)
ret = video.set(4,1280)
ret = video.set(3,720)

while(True):

    # Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]
    # i.e. a single-column array, where each item in the column has the pixel RGB value
    ret, frame = video.read()
    # cv2.imshow("Test_Video", frame)
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_expanded = np.expand_dims(frame_rgb, axis=0)

    # Perform the actual detection by running the model with the image as input
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes, num_detections],
        feed_dict={image_tensor: frame_expanded})

    # Draw the results of the detection (aka 'visulaize the results')
    capture,frame,confidence = vis_util.visualize_boxes_and_labels_on_image_array(
        frame,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,
        line_thickness=8,
        min_score_thresh=0.60)

    # Here we check if the reuired Confidence value and if it is satisfied
    # Image is Captured with Time and OCR is processed, followed by data storage in
    # Google Firebase
    if confidence > 90 :
        print(confidence)
        count = count + 1
        print (count)
        if count == 5 :

            # Time of Detection
            time = datetime.datetime.now()
            print ("Current date and time : ")
            current_time = time.strftime("%d-%m-%y_%I:%M %p")
            time_image = time.strftime("%d%b_%I-%M%p")
            print(current_time)
            local_image_path = time.strftime("%B")
            local_image_name = "CAP_{}.png".format(time_image)
            img_name = "{0}/{1}/CAP_{2}.png".format(IMAGE_SAVE_PATH,local_image_path,time_image)
            cv2.imwrite(img_name, capture)
            print("{} written!".format(img_name))

            # Storing in Firebase
            text = ocr.static_image_ocr(capture,truth)
            print(text)
            fb_store.firebase_realtime_db(text)
            fb_store.firebase_store(local_image_path,local_image_name)
    else:
        print('No')
        count = 0
        print(count)

    cv2.imshow('Object detector', frame)

    if cv2.waitKey(1) == ord('q'):
        break

# Clean up
video.release()
cv2.destroyAllWindows()
```

Figure 5.4: Initialize Webcam and Object detection

```
    # Create a display string (and color) for every box location, group any boxes
    # that correspond to the same location.
    box_to_display_str_map = collections.defaultdict(list)
    box_to_color_map = collections.defaultdict(str)
    box_to_instance_masks_map = {}
    box_to_instance_boundaries_map = {}
    box_to_keypoints_map = collections.defaultdict(list)
    confidence = 0
    if not max_boxes_to_draw:
      max_boxes_to_draw = boxes.shape[0]

    for i in range(min(max_boxes_to_draw, boxes.shape[0])):

      if scores is None or scores[i] > min_score_thresh:

        global capture
        capture = image.copy()
        confidence = int(100*scores[i])

        box = tuple(boxes[i].tolist())
        if instance_masks is not None:
          box_to_instance_masks_map[box] = instance_masks[i]
        if instance_boundaries is not None:
          box_to_instance_boundaries_map[box] = instance_boundaries[i]
        if keypoints is not None:
          box_to_keypoints_map[box].extend(keypoints[i])
        if scores is None:
          box_to_color_map[box] = groundtruth_box_visualization_color
        else:
          display_str = ''
          if not skip_labels:
            if not agnostic_mode:
              if classes[i] in category_index.keys():
                class_name = category_index[classes[i]]['name']
              else:
                class_name = 'N/A'
              display_str = str(class_name)
          if not skip_scores:
            if not display_str:
              display_str = '{}%'.format(int(100*scores[i]))
            else:
              display_str = '{}: {}%'.format(display_str, int(100*scores[i]))
          box_to_display_str_map[box].append(display_str)
          if agnostic_mode:
            box_to_color_map[box] = 'DarkOrange'
          else:
            box_to_color_map[box] = STANDARD_COLORS[
                classes[i] % len(STANDARD_COLORS)]


    # Draw all boxes onto image.
    for box, color in box_to_color_map.items():
      ymin, xmin, ymax, xmax = box
      if instance_masks is not None:
        draw_mask_on_image_array(
            image,
            box_to_instance_masks_map[box],
            color=color
        )
      if instance_boundaries is not None:
        draw_mask_on_image_array(
            image,
            box_to_instance_boundaries_map[box],
            color='red',
            alpha=1.0
        )
      draw_bounding_box_on_image_array(
          image,
          ymin,
          xmin,
          ymax,
          xmax,
          color=color,
          thickness=line_thickness,
          display_str_list=box_to_display_str_map[box],
          use_normalized_coordinates=use_normalized_coordinates)
      if keypoints is not None:
        draw_keypoints_on_image_array(
            image,
            box_to_keypoints_map[box],
            color=color,
            radius=line_thickness / 2,
            use_normalized_coordinates=use_normalized_coordinates)
    # cv.imshow('Utils Captured', capture)
    return capture,image,confidence
```

Figure 5.5: Confidence calculations and capture Image

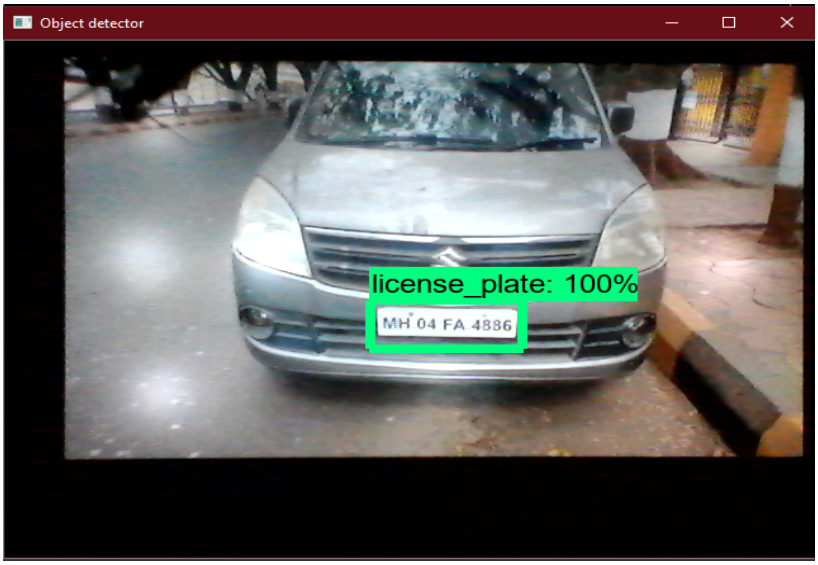


Figure 5.6: Output of Object Detection

License plate detection is done using OpenCV (Open source computer vision) to analyse and process the number plate in the database to know whether the number plate is an registered number in the database or an outsider. After capturing the license plate by OCR (Optical Character Recognition) number is obtained. OCR is an electronic method of conversion of images of typed, handwritten or printed text thus giving the number which is to be processed.

5.3.2 Image segmentation and Optical Character Recognition (OCR)

Now that the license plate was detected from the live feed and the frame of the video has been captured, the image is the processed for increasing the accuracy of OCR. First the image is read from the directory and saved into a variable. Then the image is resized and converted to grey scale image to convert it into a black and white image. This is then processed through bilateral filtering to reduce noise in the image.

Edge detection is performed to detect the edge of the license plate. There are edge detection methods such as Sobel's Operator method, Canny Edge, Prewitt's method etc., but for our purpose we have used canny edge detection as it gives the best result for our needs. The edge detection process displays only the edges of the different objects in the frame. Contouring is done for drawing out similar areas and forming areas of common features. Among all the contours we need to extract the contour with the license plate. To do that we find contours with maximum white pixels as the license plate is white color for private vehicle. We can change it to black or yellow if needed. All the contours that satisfy the requirements are then counted. Then arranged in descending order based on the area. The contour with largest area is believed to be the one with license plate.

Then all the other contours are deleted and this contour is retained. Then masking is performed to isolate the contour and the cropped. The cropped frame is then processed for OCR. Optical character recognition in Python is done with the help of Google's Tesseract Library. Tesseract is an open source platform for OCR. It's pretrained library

can recognized letters and numbers from an image. Here we use it on the cropped frame. This returns a string which has the characters. For us the string gives us the number plate details. The text string is passed to the main code. Refer Fig (OCR code) for the code of this process.

```python
# Import Libraries
import cv2 as cv
import imutils
import numpy as np
import pytesseract
import string

whitelist = string.digits + 'abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ'
text_string = ''

# This Function Extracts Characters from a static imagem
def static_image_ocr(frame,show_process):
    """
    Parameters
    ----------
    frame : Numpy Array
        Image as a numpy py array passed from the main code
    show_process : Boolean
        To display the whole process, if false display only original image and Output image with
        Detected License Plate number

    Returns
    -------
    text_string : String
        A string which contains the characters recognised from the OCR, in this case the license plate number

    """
    image = frame.copy()
    cv.imshow('Original Image',image)

    image = cv.resize(image, (620,480),interpolation = cv.INTER_AREA )

    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY) #convert to gray scale
    gray = cv.bilateralFilter(gray, 11, 17, 17) #Blur to reduce noise
    edge = cv.Canny(gray, 30, 200) #Perform Edge detection

    # Retaining only the contour with number plate
    contours = cv.findContours(edge.copy(), cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)
    contours = sorted(contours, key = cv.contourArea, reverse = True)[:10]
    count = None

    # loop over our contours
    for c in contours:
     # approximate the contour
     peri = cv.arcLength(c, True)
     approx = cv.approxPolyDP(c, 0.018 * peri, True)

     # if our approximated contour has four points, then
     # we can assume that we have found our screen
     if len(approx) == 4:
      count = approx
      break

    if count is None:
     detected = 0
     print("No contour detected")
    else:
     detected = 1

    if detected == 1:
     cv.drawContours(image, [count], -1, (0, 255, 0), 3)

    # Masking the part other than the number plate
    mask = np.zeros(gray.shape,np.uint8)
    new_image = cv.drawContours(mask,[count],0,255,-1,)
    new_image = cv.bitwise_and(image,image,mask=mask)

    # Now crop
    (x, y) = np.where(mask == 255)
    (topx, topy) = (np.min(x), np.min(y))
    (bottomx, bottomy) = (np.max(x), np.max(y))
    Cropped = gray[topx:bottomx+1, topy:bottomy+1]
    # cv.imshow('image',image)

    # resize image
    scale_percent = 300 # percent of original size
    width = int(Cropped.shape[1] * scale_percent / 100)
    height = int(Cropped.shape[0] * scale_percent / 100)
    dim = (width, height)
    invert = cv.resize(Cropped, dim, interpolation = cv.INTER_NEAREST)

    # To display the whole process
    if show_process == True :
        cv.imshow('Grayscale Image',gray)
        cv.imshow('Bilateral Filter',gray)
        # cv.imshow('Contours', Countured)
        cv.imshow('Canny Edge Detection',edge)
        cv.imshow('Mask',new_image)
        cv.imshow('Cropped',Cropped)
        cv.imshow('Resize', invert)

    # Perform OCR on the cropped frame
    text = pytesseract.image_to_string(invert, config='--psm 11')
    print("Detected Number is:",text)

    whitelist = string.digits + 'abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    text_string = ''
    for char in text:
        if char in whitelist:
            text_string += char
        else:
            text_string += ''
    print(text_string)

    out = cv.putText(image, text_string, (420,450), cv.FONT_HERSHEY_SIMPLEX,0.75, (255,255,255), 1, cv.LINE_AA)
    cv.imshow('Output', out)
    cv.waitKey(0)
    cv.destroyAllWindows()
    return text_string
```

Figure 5.7: Image Segmentation

Image Processing and Optical character recognition :

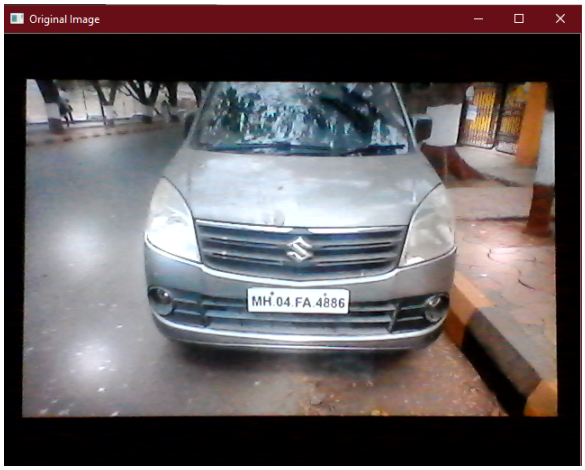


Figure 5.8: Original Image

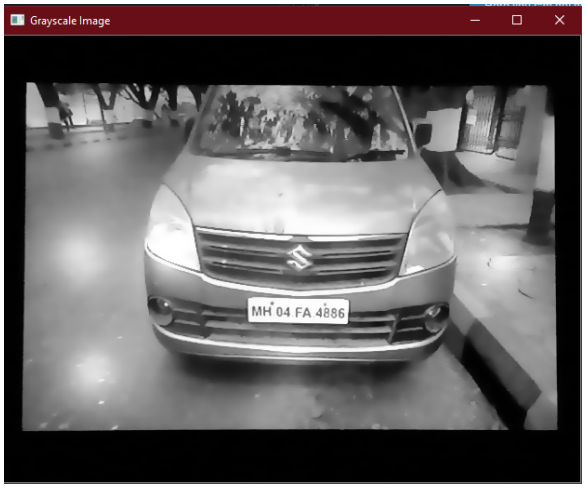


Figure 5.9: Grayscale Image

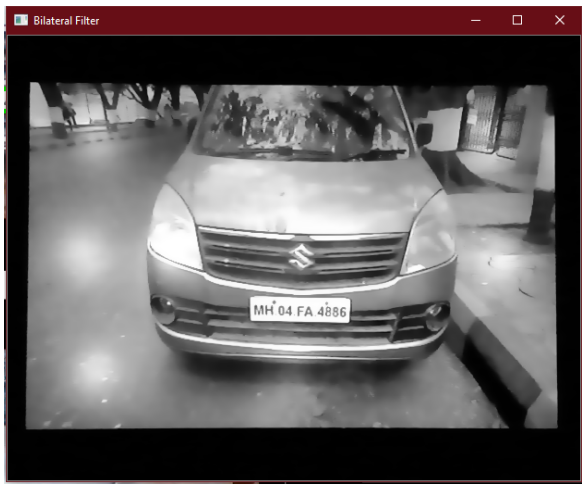


Figure 5.10: Filtered Image

Figure 5.11: Canny Edge Detection

Figure 5.12: Detected Contours

Figure 5.13: License Plate Contour

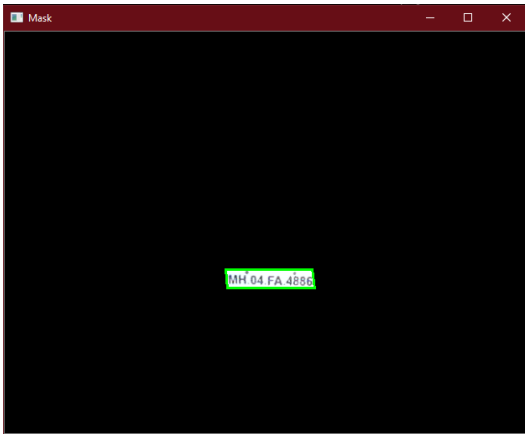


Figure 5.14: Masked Image

Figure 5.15: Extracted License Plate

Figure 5.16: License Plate Recognised and Displayed

5.3.3 Firebase Realtime Database and Storage

Google firebase is an online real-time database management system. We are using this feature to record the save the license plate number and the snapshot of the car from the front when it was detected. When the license plate is detected and processed through OCR, the text string that is returned is pushed to firebase. Simultaneously, the image captured is saved in the firebase storage for future references.

The dataset for smartpark app is organized and stored in three main tables within Google Firebase Realtime database as follows:

- **Members-**

  This table contains the car license and other details of members of the society or parking lot establishment using this smartpark system. All members are pre-assigned a parking spot.

- **Spots –**

  Each record in this spots table represents a parking spot. Parking spot are referenced via number starting from 1. The smartpark application automatically scales to the number of spot records provided by the spots table. Each record has a 'Type' attribute to indicate if that spot is designated as a member spot or an outsider spot. There is also a 'Status' attribute that contains the value "Vacant" or "Occupied" to indicate its occupancy.

- **Activity –**

  This table acts as an audit table that records all the parking activity. This table is organized as a dictionary indexed by license. Entry and exits of members and outsider are recorded. A dynamic median of the late or early arrival of each member is calculated and updated into the member table record of that member.

Here is a schema diagram that shows the relationship between the key fields (columns) of the three tables:

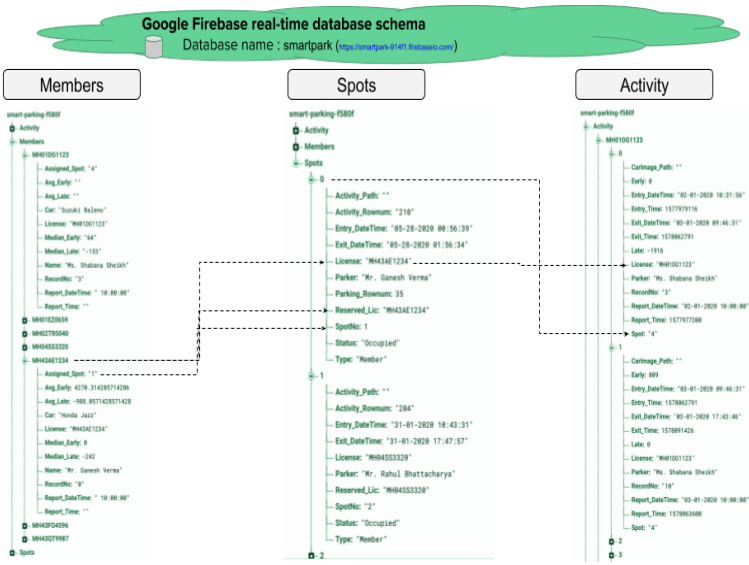


Figure 5.17: Figure: Google Firebase SmartPark Database Schema

Members table:


```
smart-parking-f580f
  Activity
  Members
    MH01DG1123
      Assigned_Spot: "4"
      Avg_Early: ""
      Avg_Late: ""
      Car: "Suzuki Baleno"
      License: "MH01DG1123"
      Median_Early: "64"
      Median_Late: "-153"
      Name: "Ms. Shabana Sheikh"
      RecordNo: "3"
      Report_DateTime: " 10:00:00"
      Report_Time: ""
    MH01EZ0659
    MH02TR5040
```


Spots table:


```
smart-parking-f580f
  Activity
  Members
  Spots
    0
      Activity_Path: ""
      Activity_Rownum: "206"
      Entry_DateTime: "09-19-2020 21:39:10"
      Exit_DateTime: "09-19-2020 22:40:11"
      License: "MH04FA488"
      Parker: "Mr. Ganesh Verma"
      Parking_Rownum: 30
      Reserved_Lic: "MH04FA488"
      SpotNo: "1"
      Status: "Vacant"
      Type: "Member"
    1
```


Activity table:


```
smart-parking-f580f
  Activity
    MH01DG1123
      0
        CarImage_Path: ""
        Early: 0
        Entry_DateTime: "02-01-2020 10:31:56"
        Entry_Time: 1577979116
        Exit_DateTime: "03-01-2020 09:46:31"
        Exit_Time: 1578062791
        Late: -1916
        License: "MH01DG1123"
        Parker: "Ms. Shabana Sheikh"
        RecordNo: "3"
        Report_DateTime: "02-01-2020 10:00:00"
        Report_Time: 1577977200
        Spot: "4"
      1
        CarImage_Path: ""
        Early: 809
        Entry_DateTime: "03-01-2020 09:46:31"
        Entry_Time: 1578062791
```

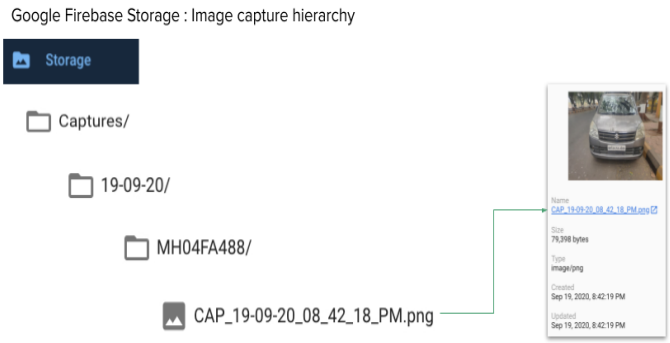



Figure 5.18: Figure: Images Storage Hierarchy

5.3.4 Parking Lot Occupancy Management (Smart Parking)

Parking Lot occupancy and management is implemented via a python module named SmartParkApp. This module provides several key functions that is called from the LPR system. It uses the google firebase database to persist all the datasets used by this application. This module is responsible for implementing the 'smartness' in the smart park app. The key implementation aspects are as follows:

The application allows for all members to be registered as members via a membership table and assign a parking spot to them. For the purposes of this project we have 7 members but this can be dynamically changed by adding more records to the member table.

The application does not hard code the number of available spots that it manages instead it allows the user of a system to dynamically configure the spot count by adding them to the spot table. The number of records in the spot tables automatically determines the lot size. For the purposes of this project, we have kept this lot size to 10 spots.

The application allows parking spots to be designated as members only or outsiders only spots. For the purpose of this project we have designated 7 spots as member spots and 3 as outsiders/visitors spots. All these can be dynamically configured as well.

Every member has a reporting time attribute which indicates when the member is expected to enter a parking lot and request an allotment for his/her spot.

All parking activities are tracked by the application by recording them in an activity table.

All date and time values are stored in both human readable form in the long datetime format viz. "23-09-2020 9:45:00" and in their machine-readable format as a float timestamp format viz. 1600834500 for efficient datetime processing and manipulations.

Smart Requirements Solutions

If a member's spot is vacant then the arrival of that member requires the application to allot that spot to that member.

If an outsider spot is vacant then the arrival of an outsider requires an application to allot that spot to that outsider.

If all the outsider spots are occupied and a new outsider arrives then this outsider is allocated a member's spot provided that spot is vacant. If several member spots are vacant then the application needs to smartly choose one of those member's spot based on a data based "smart" criteria. The "smart" criteria implemented by the application involves calculating a deviation of a member's entry time from their pre-configured reporting time as mentioned above. This allows for the most consistent late arriving member to lose their spot to an outsider.

If the entry time of an outsider's vehicle is less than a member's reporting time then that member's spot should not be allotted to the outsider as the member can arrive before the reporting time and is entitled to his/her spot.

In scenarios where one member is allotted another member's spot and the application has to choose from several member's vacant spots then it chooses a spot based on the member's past parking activity. The application aggregates all the entry exit times of the member and determines an early or late vector series. This vector series is then used to calculate a unique median score for that member. A low median score indicates impunctuality. By comparing multiple member median scores, we can now determine which member has been most unpunctual. This scientific criterion can now be used to pick the member whose spot needs to be allocated to outsider.

Below are code snippets of following important functions/methods within smartparkapp python application module:

- IsCarParked()
- RegisterCarEntry()
- FindNextAvailableSpot()
- RegisterCarExit()
- CalcAllMemberMedianAndAverage()
- UploadCarImage()

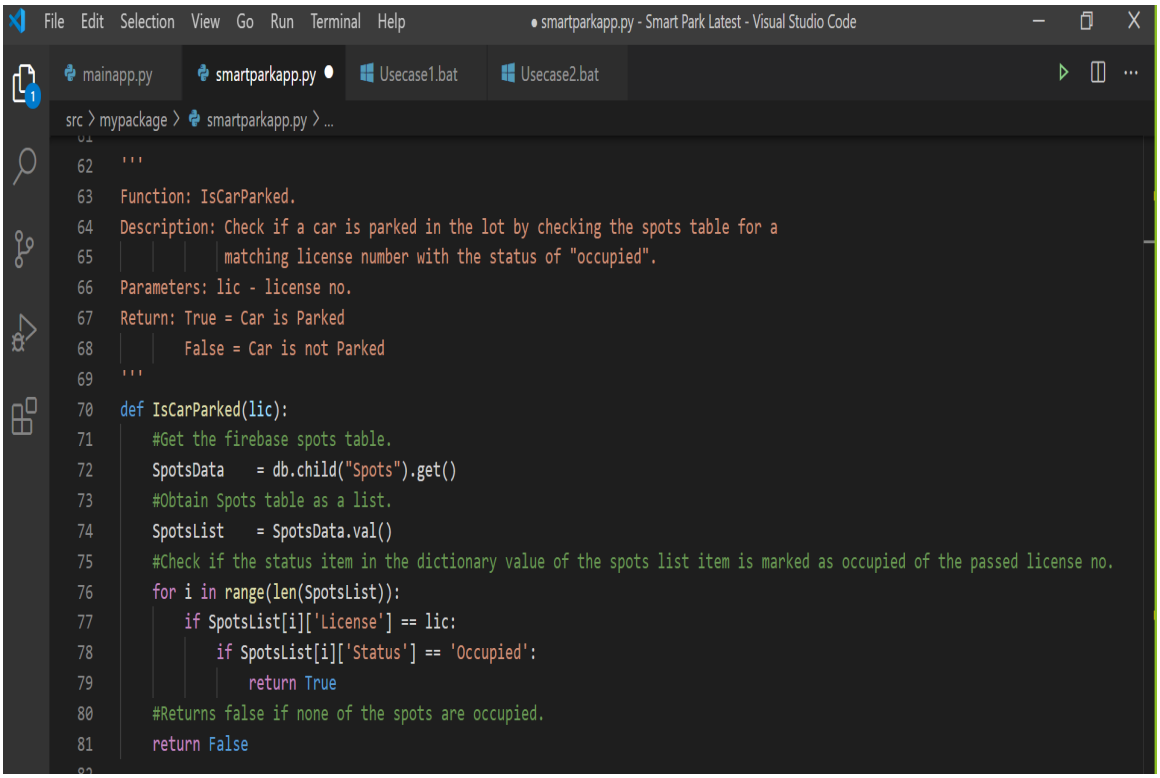


Figure 5.19: IsCarParked Method

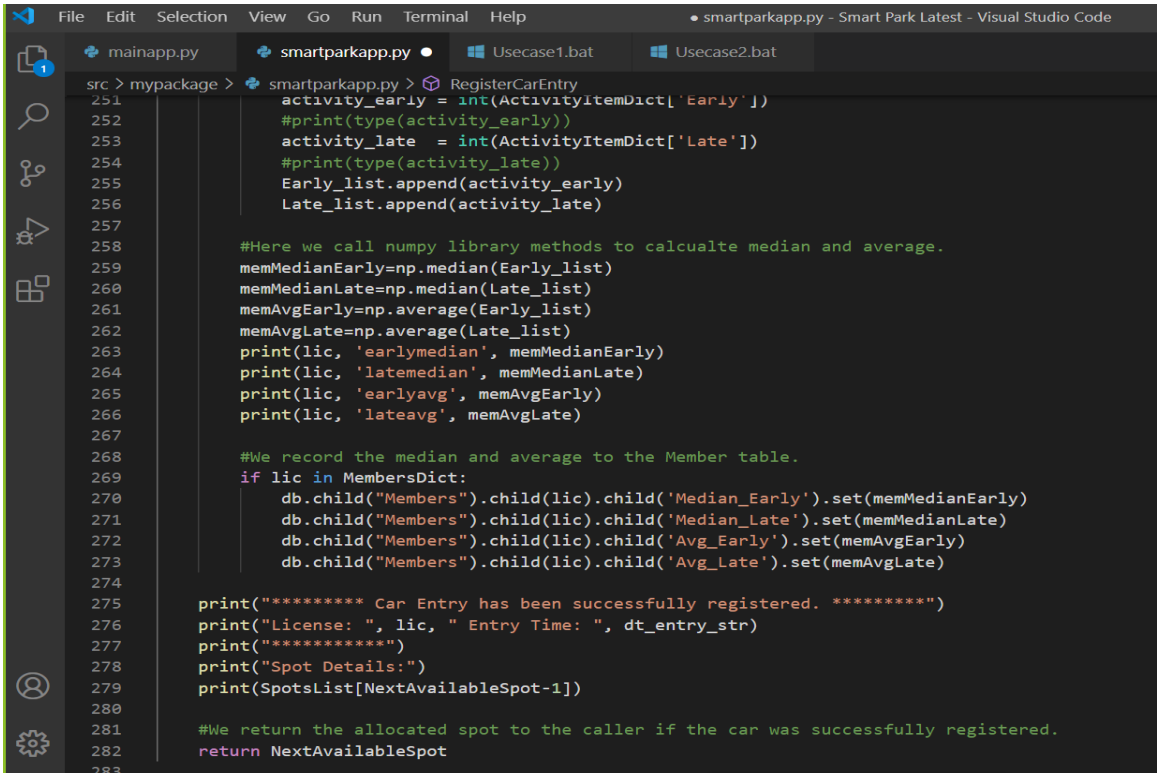


Figure 5.20: RegisterCarEntry Method

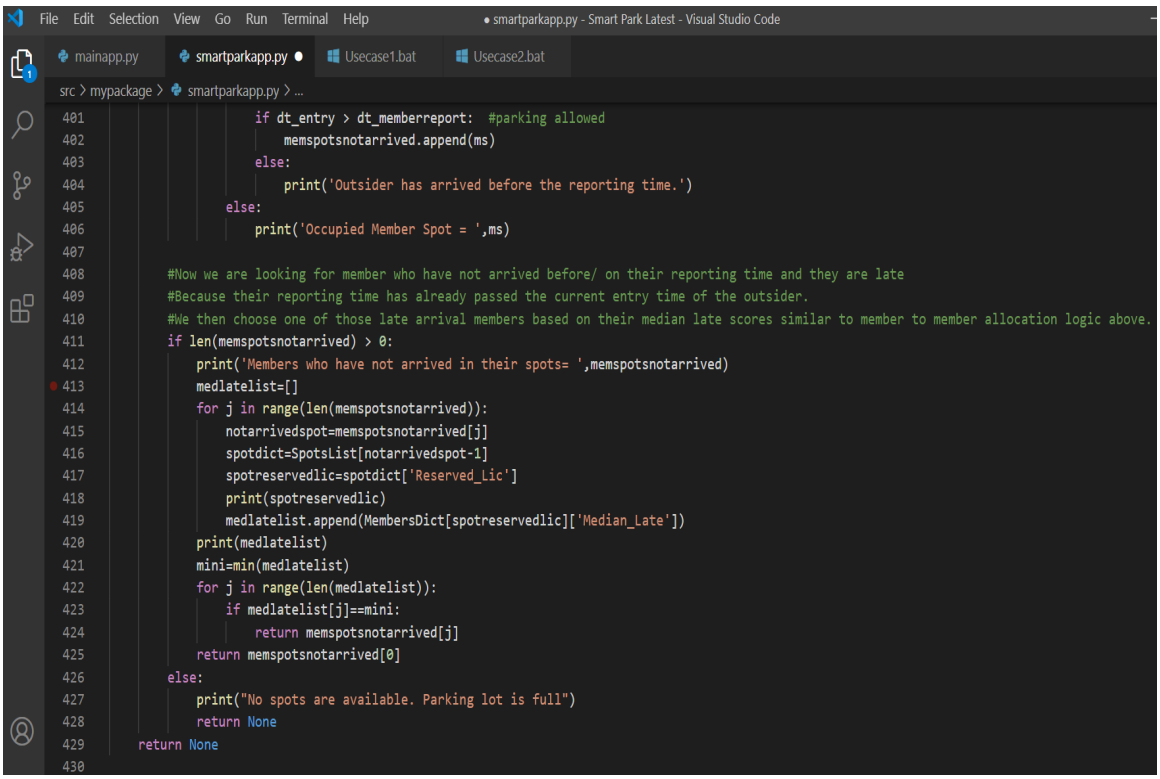


Figure 5.21: FindNextAvailableSpot Method

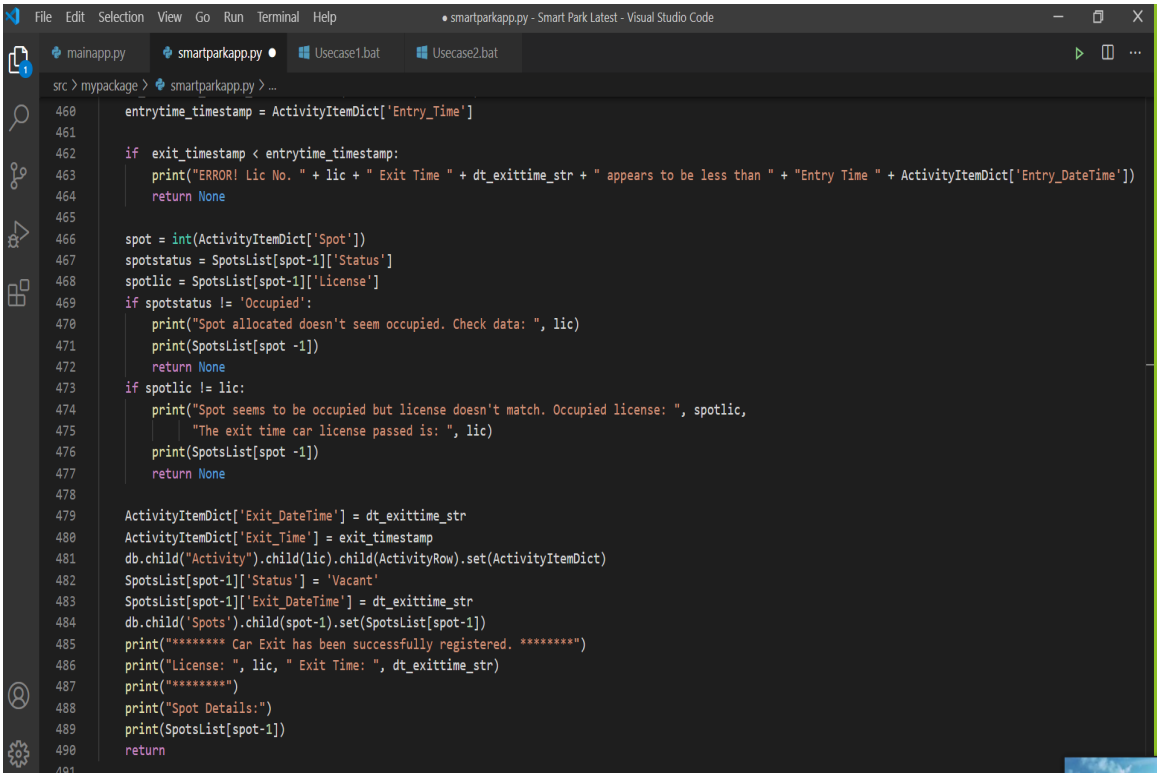


Figure 5.22: RegisterCarExit Method

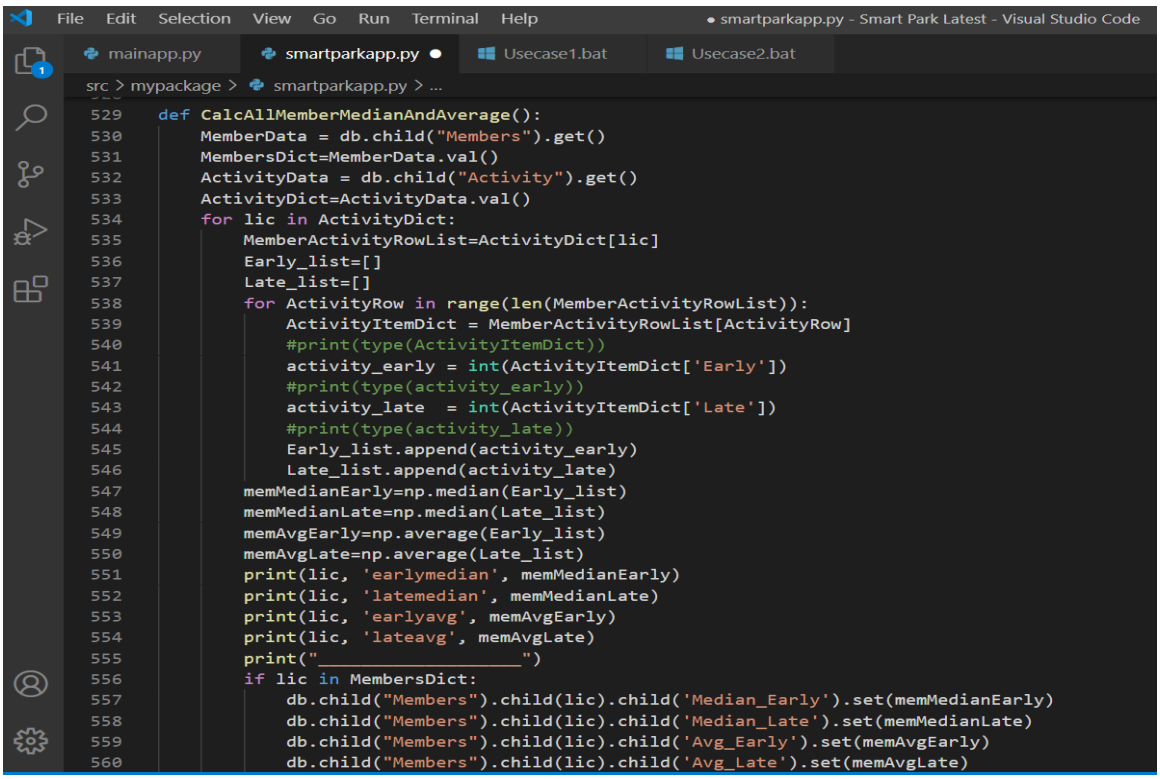


Figure 5.23: CalcAllMemberMedianAndAverage Method

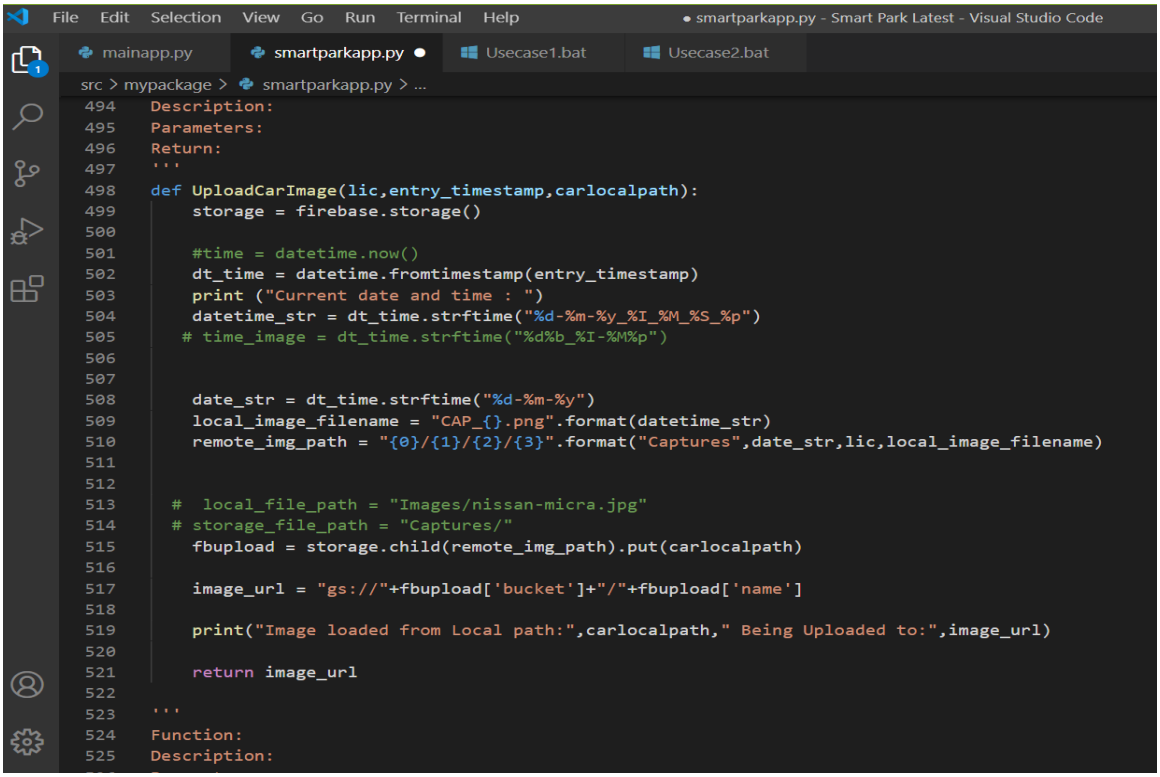
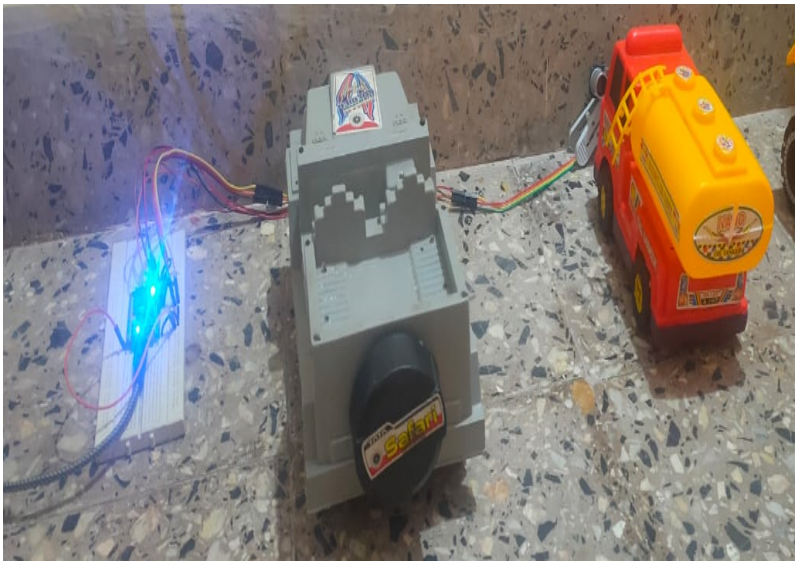


Figure 5.24: UploadCarImage Method

5.3.5 Parking Spot Occupancy Detection using IoT

The second part of this system is to keep track of the status of the parking spots in the area such as keeping a track of empty lot and no of occupied lots. For achieving this task we have implemented Internet of Things in our project using the simple modules such as NodeMCU development board, a microcontroller board with ESP8366 WiFi chip embedded in it and the Ultrasonic Distancing Meter (UDM) for measuring distance. UDM is a sonar based distance measurement device. Both of the components are connected with wires and here NodeMCU is the driving element of the system and UDM is the driven element. NodeMCU is connected to a Local Wifi Network, by achieving this, the system is connected to the internet and can be controlled from any part of the world.

The UDM is fitted in front of the parking space perpendicular to the length if the parking space. When it is unoccupied the distance measured by UDM is very high as there is no surface for the sound waves to rebound from. When a car is parked in the UDM senses decrease in distance as the bonnet of the car acts as a surface to reflect the sound waves back. The NodeMCU is programmed such that when the distance measured falls below a certain threshold, it declares the parking lot as occupied. The updated status of the parking lot is pushed to the Firebase RTDB. One NodeMCU can keep track of 6 parking lot, so multiple such systems can be installed for a large number of parking lots. When the parking space is occupied the system uploads '1' to firebase, this indicates that the space is occupied and if the car moves out it is reset to '0' indicating it to be vacant and allow other cars to park in that area.



BE_final

```
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

// Initialise Firebase database ID, Token Secrets
// SSID, password of the local WIFI Network
#define FIREBASE_HOST "nodemcu-led-b3637.firebaseio.com"
#define FIREBASE_AUTH "bk1ScvsVBNfLitAt4pZO3EqEtM6SfC2cLn7MJAcb"
#define WIFI_SSID "Anonymous"
#define WIFI_PASSWORD "batman123"

//Firebase RTDB Path
const String path = "/Parking";
const String Floor = "/Floor 1"; //initalise the number according to the floor
```

Figure 5.25: Firebase and WiFi credentials

```
// defining pins for UDM connection
const int trigPin1 = D0;
const int echoPin1 = D1;
const int trigPin2 = D2;
const int echoPin2 = D3;
const int trigPin3 = D4;
const int echoPin3 = D5;

// defines variables to store the values received from UDMs
long duration;
int distance;
long Sensor1,Sensor2,Sensor3;

// Function for calculation of Distance based on UDM readings
void SonarSensor(int trigPin,int echoPin)
{
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
distance = (duration/2) / 29.1;
}
```

Figure 5.26: UDM pins and Distance Calculating Function

```
void setup() {
  // This Code runs only once and hence used to connect to wifi
  // and initials pins
  // Start Serial Monitor for Reading Outputs
  Serial.begin(115200);

  // connect to wifi.
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("connected: ");
  Serial.println(WiFi.localIP());

  // Firebase Initialised
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  // Initialising the pins as Input or Output
  pinMode(trigPin1, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin1, INPUT); // Sets the echoPin as an Input
  pinMode(trigPin2, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin2, INPUT); // Sets the echoPin as an Input
  pinMode(trigPin3, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin3, INPUT); // Sets the echoPin as an Input
}
```

Figure 5.27: One time run Loop for Setting up the system and connecting to Wifi

```
void loop() {
  // this code runs repeatedly till the system is shut off

  SonarSensor(trigPin1, echoPin1); //Function Call for Parking 1
  Sensor1 = distance;
  if(Sensor1<10) // Loop to check distance
  {
    Firebase.setInt(Floor + "/Parking 1", 1); // Parking space Occupied
    }
    else
    {
      Firebase.setInt(Floor + "/Parking 1", 0); // Parking space Vacant
      }

  SonarSensor(trigPin2, echoPin2); //Function Call for Parking 2
  Sensor2 = distance;
  if(Sensor2<10) // Loop to check distance
  {
    Firebase.setInt(Floor + "/Parking 2", 0); // Parking space Occupied
    }
    else
    {
      Firebase.setInt(Floor + "/Parking 2", 1); // Parking space Vacant
      }

  SonarSensor(trigPin3, echoPin3); //Function Call for Parking 3
  Sensor3 = distance;
  if(Sensor3<10) // Loop to check distance
  {
    Firebase.setInt(Floor + "/Parking 3", 1); // Parking space Occupied
    }
    else
    {
      Firebase.setInt(Floor + "/Parking 3", 0); // Parking space Vacant
      }
  // Print the reading in Serial Monitor
  Serial.print(Sensor1);
  Serial.print(" \t ");
  Serial.print(Sensor2);
  Serial.print(" \t ");
  Serial.print(Sensor3);
  Serial.println();
  delay(10);
}
```

Figure 5.28: Main code

Chapter 6

Results

6.1 Object Detection Output

Figure 6.1: Detected License plate and Confidence score

Figure 6.2: Original image Captured

Figure 6.3: License Plate Recognised and Displayed

6.2 RealTime Database(RTDB) & Storage

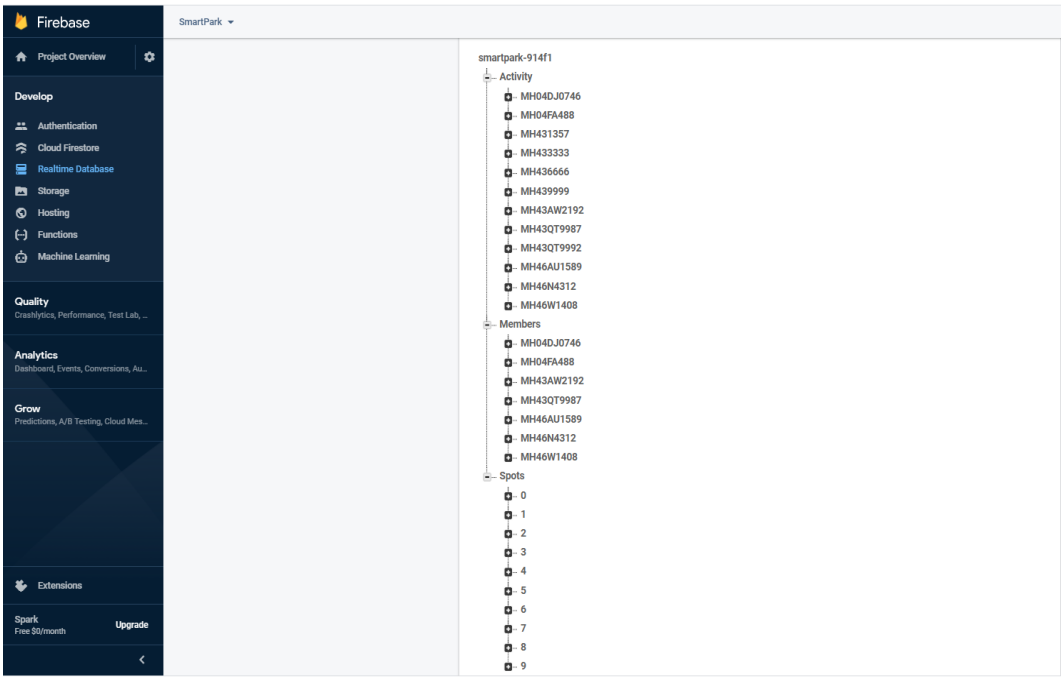


Figure 6.4: Firebase Real-Time Database

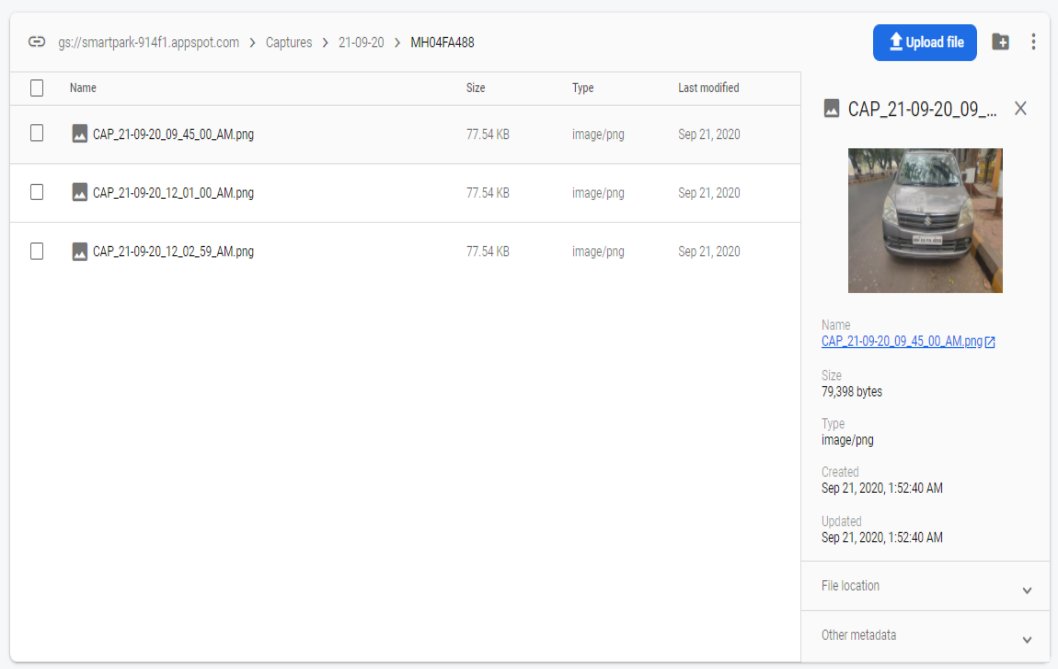


Figure 6.5: Firebase Storage

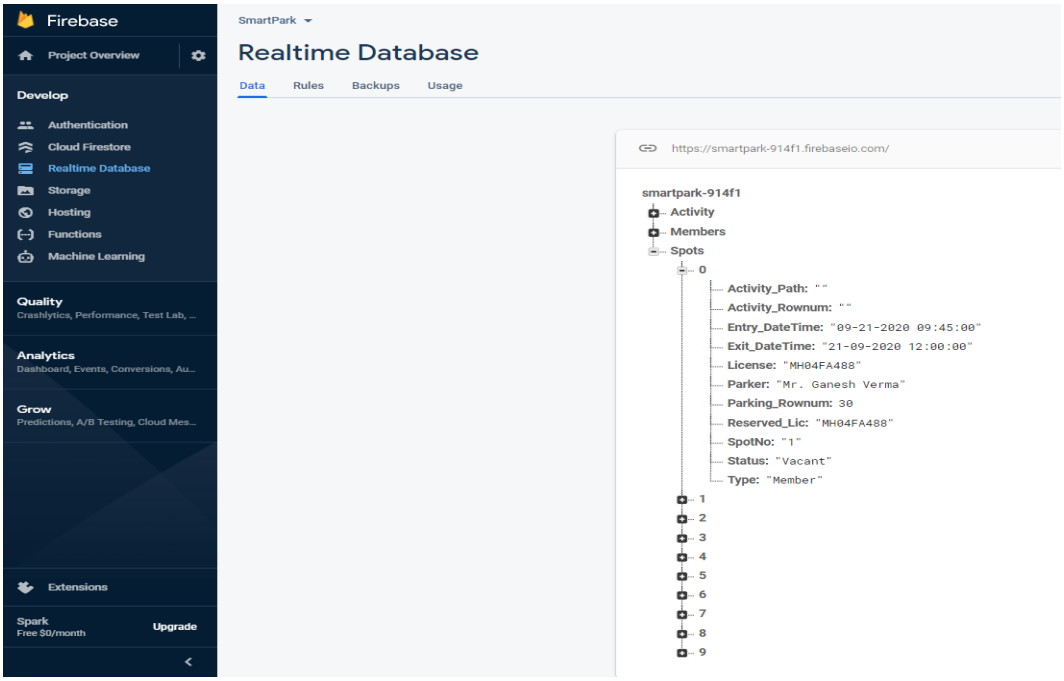


Figure 6.6: When Parking Spot 1 is Vacant

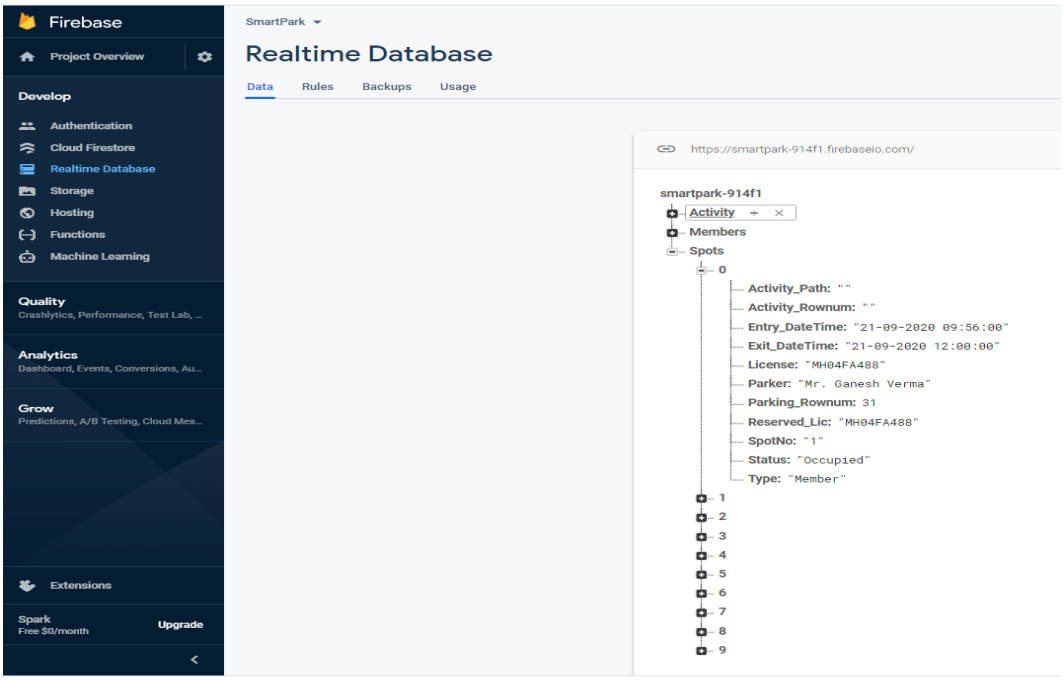


Figure 6.7: When Parking Spot 1 is Occupied

6.3 Webpage Output

The parking lot availability is also fed to the firebase RTDB which is used for WebApp to display the status of the area.

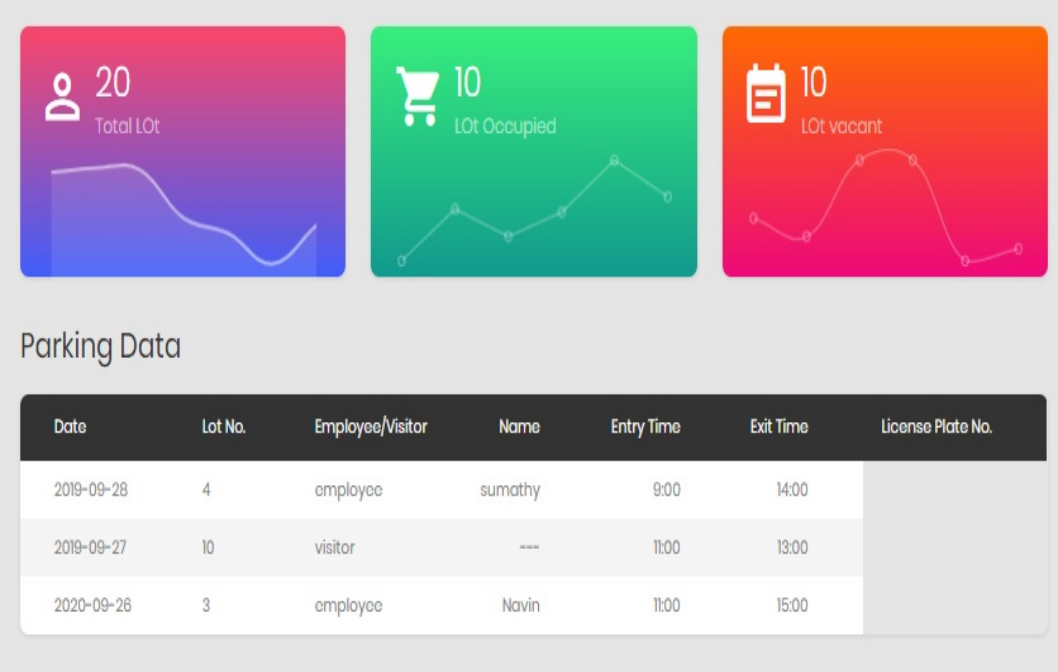


Figure 6.8: web page front end

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

In this work, a first approach of using existing surveillance cameras for detecting license plates is presented. An open source platform is used and improved to evaluate only objects that are moving. The algorithm run on a distributed platform that operate multiple cameras and sensors, to support comprehensive security management. The first results are reasonable in performance, reaching a near real-time LPR detection, even for high resolution images. The final output of the license plates was found to be very close to the actual value on the plates. However, some errors creep in depending upon the quality of the original image, the lighting, the shakiness and general condition of the license plate.

7.2 Future Scope

In future the system can be extended which is not only specific to a private parking like malls, company parking, etc. but can also be implemented on public parking which has extending features such as reduction of traffic as search time for parking is reduced ,enhanced user experience as user can easily find the vacant parking lot, real-time data and trend insight which is achieved by studying the arrival and leaving time of a particular car using ML, decreased management costs as this reduces the labour cost and can be more efficient. Since data stored is real-time safety can be ensured which helps prevent parking violations and suspicious activity.

Chapter 8

References

[1] Amato, Giuseppe, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, and Claudio Vairo. "Car parking occupancy detection using smart camera networks and deep learning." *In 2016 IEEE Symposium on Computers and Communication (ISCC), pp. 1212-1217. IEEE, 2016.*

[2] Lookmuang, Rachapol, Krit Nambut, and Sasiporn Usanavasin. "Smart parking using IoT technology." *In 2018 5th International Conference on Business and Industrial research (ICBIR), pp. 1-6. IEEE, 2018.*

[3] Dominguez, Leonardo, Juan Pablo D'Amato, Alejandro Perez, Aldo Rubiales, and Rosana Barbuzza. "*Running License Plate Recognition (LPR) algorithms on smart survillance cameras. A feasibility analysis."* In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1-5. IEEE, 2018.

[4]Pham, Thanh Nam, Ming-Fong Tsai, Duc Binh Nguyen, Chyi-Ren Dow, and Der-Jiunn Deng. "*A cloud-based smart-parking system based on Internet-of-Things technologies." IEEE Access 3 (2015): 1581-1591.*

[5] https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10
Evan(EdjeElectronics),[]"How to train a TensorFlow Object Detection Classifier for multiple object detection on Windows",

[6] https://en.wikipedia.org/wiki/Opticalcharacterrecognition
Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo.

[7] https://www.raspberrypi.org/about/ *The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.*

[8] https://en.wikipedia.org/wiki/Tesseract(software)
Tesseract is an optical character recognition engine for various operating systems. It is free software, released under the Apache License, Version 2.0 and development has been spon- sored by Google since 2006.

[9] https://github.com/tesseract- ocr/tesseract
The library for OCR t ool in Tesseract posted in github.

[10]https://towardsdatascience.com/fasterrcnn-explained-part-1-with-code-599c16568cff *Mike Liao, March 1,2108 Detecting Objects in (almost) Real-time: FasterRCNN Explained with Code.*

[11] Goswami, Subrata. "A deeper look at how Faster-RCNN works." (2018).

[12]https://opencv.org/about/ OpenCV, open source image processing library for python

URKUND

Document Information

Analyzed document	smart_parking (1) (1).pdf (D78843781)
Submitted	9/9/2020 5:49:00 PM
Submitted by	Hema
Submitter email	hema.raut@siesgst.ac.in
Similarity	2%
Analysis address	hema.raut.sies@analysis.urkund.com

Sources included in the report

W	URL: https://www.ijeat.org/wp-content/uploads/papers/v9i1/A1963109119.pdf Fetched: 9/9/2020 5:51:00 PM	2
W	URL: https://docplayer.net/38893112-Smart-parking-system-based-on-embedded-system-and-s ... Fetched: 9/9/2020 5:51:00 PM	1
W	URL: https://github.com/tpmabdulkareem/Smart-Parko Fetched: 12/28/2019 8:06:50 AM	1