**AI ENABLED CAR PARKING USING OPENCV**

## A PROJECT REPORT

***Submitted by***

**PRIYADHARSHINI V (311420104052)**

**REVANYA S**  **(311420104057)**

**SAI KASTHURI S**   **(311420104060)**

**NADHIYA S**   **(311420104309)**

***In partial fulfillment for the award of the degree   
of***

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**MEENAKSHI COLLEGE OF ENGINEERING, WEST K.K NAGAR**

**TABLE OF CONTENTS**

**CHAPTER TITLE**

## INTRODUCTION

|  |  |
| --- | --- |
| 1.1 | PROJECT OVERVIEW |
| 1.2 | PURPOSE |

1. **IDEATION & PROPOSED SOLUTION** 
   1. [PROBLEM STATEMENT DEFINITION](file:///C:\Users\ELCOT\Downloads\NM%20FINAL%20PROJECT%20REPORT.docx#_TOC_250028)
   2. EMAPTHY MAP CANVAS
   3. IDEATION & BRAINSTORMING
   4. PROPOSED SOLUTION
2. REQUIREMENT ANALYSIS
   1. [FUNCTIONAL REQUIREMENTS](file:///C:\Users\ELCOT\Downloads\NM%20FINAL%20PROJECT%20REPORT.docx#_TOC_250025)
   2. [NON-FUNCTIONAL REQUIREMENTS](file:///C:\Users\ELCOT\Downloads\NM%20FINAL%20PROJECT%20REPORT.docx#_TOC_250024)
3. PROJECT DESIGN
   1. [DATA FLOW DIAGRAMS](file:///C:\Users\ELCOT\Downloads\NM%20FINAL%20PROJECT%20REPORT.docx#_TOC_250017)
   2. [SOLUTION & TECHNICAL ARCHITECTURE](file:///C:\Users\ELCOT\Downloads\NM%20FINAL%20PROJECT%20REPORT.docx#_TOC_250016)
   3. USER STORIES
4. CODING & SOLUTIONING
   1. [FEATURE 1](file:///C:\Users\ELCOT\Downloads\NM%20FINAL%20PROJECT%20REPORT.docx#_TOC_250007)

5.2 FEATURE 2

1. RESULTS
   1. PERFORMANCE METRICS

1. ADVANTAGES & DISADVANTAGES
2. CONCLUSION
3. FUTURE SCOPE
4. APPENDIX

SOURCE CODE

**CHAPTER 1**

**INTRODUCTION**

* 1. Project Overview

Car parking is a common problem faced by drivers in busy urban areas. For example, imagine you are driving to a shopping mall during peak hours. As you approach the mall, you notice that the parking lot is full, and several other cars are circling around looking for available spots.

You join the queue of cars, hoping to find an available spot soon. However, as time passes, you realize that the parking lot is overcrowded, and it's becoming increasingly difficult to find a spot. You start to feel frustrated and anxious, knowing that you might be late for your appointment or miss out on a great shopping opportunity.

AI-enabled car parking using OpenCV is a computer vision-based project that aims to automate the parking process. The project involves developing an intelligent system that can identify empty parking spaces and it gives the count of available parking spots.

* 1. Purpose

AI-enabled car parking is a system that uses computer vision to monitor parking spots and detect vacant spaces.AI-enabled car parking is a system that uses computer vision to monitor parking spots and detect vacant spaces. They offer convenience for vehicle users and efficient usage of space for urban-based companies. Automated car park systems save time, money, space and simplify the often tedious task of parking.

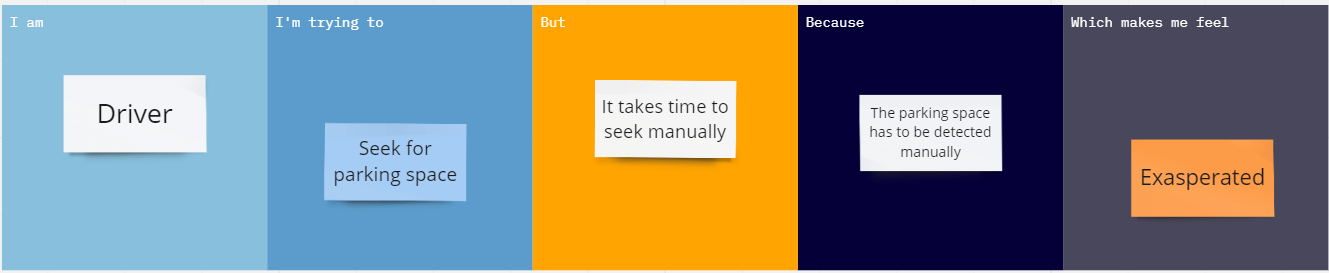
## CHAPTER 2

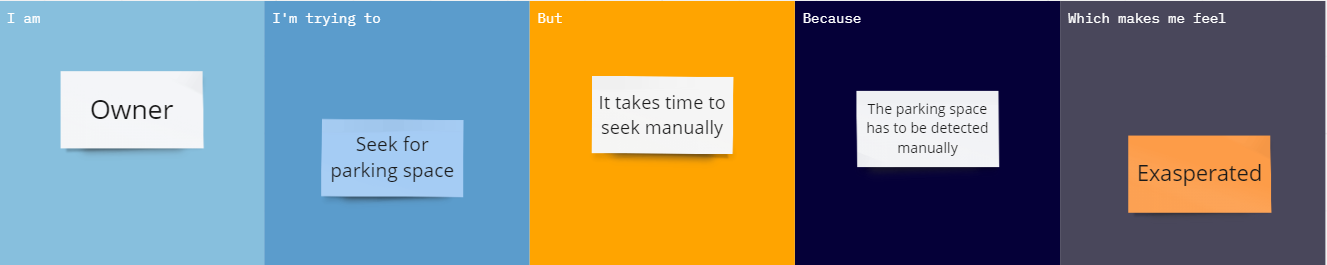
## IDEATION & PROPOSED SOLUTION

## Problem Statement Definition

## Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love.

## A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you’ll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Problem Statement (PS)** | **I am (Customer)** | **I’m trying to** | **But** | **Because** | **Which makes me feel** |
| PS-1 | Driver | Seek for parking space | It takes time to seek manually | The parking space has to be detected manually | Exasperated |
| PS-2 | Owner | Seek for parking space | It takes time to seek manually | The parking space has to be detected manually | Exasperated |

## Empathy Map Canvas

## An empathy map is a simple, easy-to-digest visual that captures knowledge about a user’s behaviours and attitudes.

## It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user’s perspective along with his or her goals and challenges.

## 

## Ideation & Brainstorming

## Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

## Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

## Step-1: Team Gathering, Collaboration and Select the Problem Statement

## 

## Step-2: Brainstorm, Idea Listing and Grouping

## 

## Step-3: Idea Prioritization

## 

## Proposed Solution

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Parameter** | **Description** |
|  | Problem Statement (Problem to be solved) | How might we seek for parking availability? |
|  | Idea / Solution description | Idea in this project is to create a Parking Space Counter. We will find how many total cars are present and how many spaces are vacant to park. |
|  | Novelty / Uniqueness | Uniqueness of this project is, it also finds out the count of occupied parking spaces in a car-parking-lot |
|  | Social Impact / Customer Satisfaction | 1. Reduces wait-time. 2. The car park areas are represented in rectangle which helps the user to find the empty slot easily. |
|  | Business Model (Revenue Model) | Commission Per Booking: For every a parking space booking , the parking owner charges a certain amount from the user. A part of this amount is returned to the parking app owner for using the app's platform to take bookings. |
|  | Scalability of the Solution | Above 90% efficient. |

**CHAPTER 3**

**REQUIREMENT ANALYSIS**

## Functional requirements

## Following are the functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Functional Requirement (Epic)** | **Sub Requirement (Story / Sub-Task)** |
| FR-1 | Admin Confirmation | Define new parking areas specify a range of parking lots  the parking cost per minute/hour |
| FR-2 | Parking Operator Confirmation | Send the vehicle plate number and reservation password for verification |
| FR-3 | Parking Operator | Issue bills to users on checkout. |
| FR-4 | User Registration | Registration through Form Registration through Gmail |
| FR-5 | User Confirmation | Confirmation via Email  Confirmation via OTP |

## Non-Functional requirements

## Following are the non-functional requirements of the proposed solution.

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | **Usability** | The system is designed for user friendly environment and ease of use |
| NFR-2 | **Security** | Unauthorized person cannot access the panel and  database, do not read and write the information |
| NFR-3 | **Reliability** | It should be reliable to both Operator and Admin environment |
| NFR-4 | **Performance** | Saves the persons and operators precious time |
| NFR-5 | **Availability** | The availability of the parking slots are detected by cameras using the method OpenCV |
| NFR-6 | **Scalability** | To manage the demand for availability, high backend  team is assigned. |

## CHAPTER 4

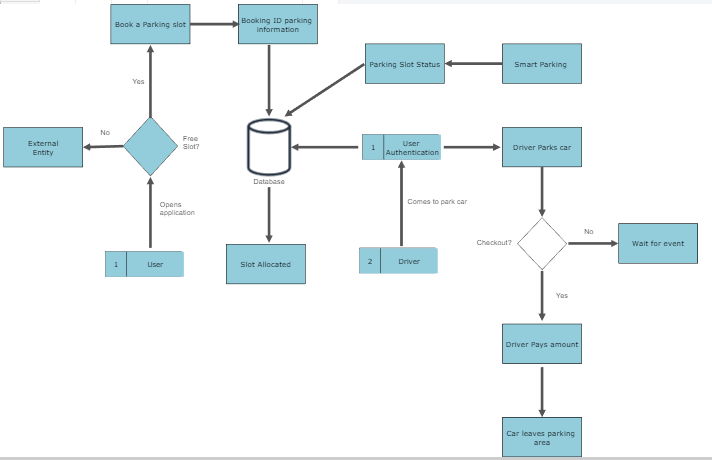
## PROJECT DESIGN

## Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

**Data Flow Diagram:**





## Solution & Technical Architecture

## Solution Architecture:

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

* Find the best tech solution to solve existing business problems.
* Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
* Define features, development phases, and solution requirements.
* Provide specifications according to which the solution is defined, managed, and delivered.

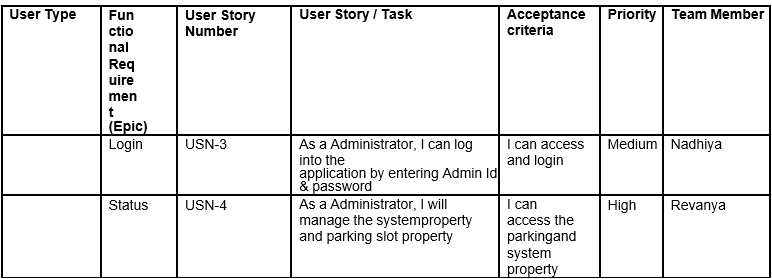
## 

## Technical Architecture:

## 

## User Stories

****

****

## CHAPTER 5

## CODING AND SOLUTIONING

## Feature 1

TRAINING A MODEL ON PARKING SLOT AVAILABILITY

## We have trained a deep learning model on a dataset "parking space detection" and have got an accuracy of 01.0000 and the testing set (validation testing) gave an accuracy of 0.2317.

## CNN Model.ipynb

## import numpy

## import os

## import tensorflow as tf

## from tensorflow.keras import applications

## from tensorflow.keras.preprocessing.image import ImageDataGenerator

## from tensorflow.keras import optimizers

## from tensorflow.keras.models import Sequential, Model

## from tensorflow.keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D

## from tensorflow.keras import backend as k

## from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping

## # 1. Load Test and Train Files

## files\_train = 0

## files\_validation = 0

## cwd = os.getcwd()

## folder = '/content/drive/MyDrive/train\_data'

## for sub\_folder in os.listdir(folder):

## break

## path, dirs, files = next(os.walk(os.path.join(folder,sub\_folder)))

## files\_train += len(files)

## folder = '/content/drive/MyDrive/train\_data/test'

## for sub\_folder in os.listdir(folder):

## path, dirs, files = next(os.walk(os.path.join(folder,sub\_folder)))

## files\_validation += len(files)

## print(files\_train,files\_validation)

## # 2. Set key parameters

## img\_width, img\_height = 48, 48

## train\_data\_dir = "/content/drive/MyDrive/train\_data"

## validation\_data\_dir = "/content/drive/MyDrive/train\_data/test"

## nb\_train\_samples = files\_train

## nb\_validation\_samples = files\_validation

## batch\_size = 32

## epochs = 15

## num\_classes = 2

## # 3. Build model on top of a trained VGG

## model = applications.VGG16(weights = "imagenet", include\_top=False, input\_shape = (img\_width, img\_height, 3))

## # Freeze the layers which you don't want to train. Here I am freezing the first 5 layers.

## for layer in model.layers[:10]:

## layer.trainable = False

## x = model.output

## x = Flatten()(x)

## # x = Dense(512, activation="relu")(x)

## # x = Dropout(0.5)(x)

## # x = Dense(256, activation="relu")(x)

## # x = Dropout(0.5)(x)

## predictions = Dense(num\_classes, activation="softmax")(x)

## # creating the final model

## model\_final = Model(inputs = model.input, outputs = predictions)

## # compile the model

## model\_final.compile(loss = "categorical\_crossentropy",

## optimizer = optimizers.SGD(lr=0.0001, momentum=0.9),

## metrics=["accuracy"]) # See learning rate is very low

## # Initiate the train and test generators with data Augumentation

## train\_datagen = ImageDataGenerator(

## rescale = 1./255,

## horizontal\_flip = True,

## fill\_mode = "nearest",

## zoom\_range = 0.1,

## width\_shift\_range = 0.1,

## height\_shift\_range=0.1,

## rotation\_range=5)

## test\_datagen = ImageDataGenerator(

## rescale = 1./255,

## horizontal\_flip = True,

## fill\_mode = "nearest",

## zoom\_range = 0.1,

## width\_shift\_range = 0.1,

## height\_shift\_range=0.1,

## rotation\_range=5)

## train\_generator = train\_datagen.flow\_from\_directory(

## train\_data\_dir,

## target\_size = (img\_height, img\_width),

## batch\_size = batch\_size,

## class\_mode = "categorical")

## validation\_generator = test\_datagen.flow\_from\_directory(

## validation\_data\_dir,

## target\_size = (img\_height, img\_width),

## class\_mode = "categorical")

## # Save the model according to the conditions

## checkpoint = ModelCheckpoint("car1.h5", monitor='val\_acc', verbose=1, save\_best\_only=True, save\_weights\_only=False, mode='auto', period=1)

## early = EarlyStopping(monitor='val\_acc', min\_delta=0, patience=10, verbose=1, mode='auto')

## #Start training!

## history\_object = model\_final.fit\_generator(

## train\_generator,

## steps\_per\_epoch = len(train\_generator),

## epochs = epochs,

## validation\_data = validation\_generator,

## validation\_steps = len(validation\_generator),

## callbacks = [checkpoint, early])

## import matplotlib.pyplot as plt

## print(history\_object.history.keys())

## plt.plot(history\_object.history['accuracy'])

## plt.plot(history\_object.history['val\_accuracy'])

## plt.title('model accuracy')

## plt.ylabel('accuracy')

## plt.xlabel('epoch')

## plt.legend(['train', 'test'], loc='upper left')

## plt.show()

## plt.plot(history\_object.history['loss'])

## plt.plot(history\_object.history['val\_loss'])

## plt.title('model loss')

## plt.ylabel('loss')

## plt.xlabel('epoch')

## plt.legend(['train', 'test'], loc='upper left')

## plt.show()

## model\_final.save("myModel.h5")

## model\_final.summary()

## Feature 2

## CREATION OF WEB APP

## We have created a web application to find the parking space availability with a good User Interface for the sake of registering and logging into the user account . This Application gets the user details and checks for the car parking availability and displays the result to the user.

## Identify the parking slot.ipynb

## from \_\_future\_\_ import division

## import matplotlib.pyplot as plt

## import cv2

## import os, glob

## import numpy as np

## from moviepy.editor import VideoFileClip

## cwd = os.getcwd()

## # %matplotlib inline

## # %config InlineBackend.figure\_format = 'retina'

## def show\_images(images, cmap=None):

## cols = 2

## rows = (len(images)+1)//cols

## 

## plt.figure(figsize=(15, 12))

## for i, image in enumerate(images):

## plt.subplot(rows, cols, i+1)

## # use gray scale color map if there is only one channel

## cmap = 'gray' if len(image.shape)==2 else cmap

## plt.imshow(image, cmap=cmap)

## plt.xticks([])

## plt.yticks([])

## plt.tight\_layout(pad=0, h\_pad=0, w\_pad=0)

## plt.show()

## from google.colab import drive

## drive.mount('/content/drive')

## test\_images = [plt.imread(path) for path in glob.glob('/content/drive/MyDrive/test\_images/\*.jpg')]

## show\_images(test\_images)

## #Color Selection and Edge Detection

## # image is expected be in RGB color space# image

## def select\_rgb\_white\_yellow(image):

## # white color mask

## lower = np.uint8([120, 120, 120])

## upper = np.uint8([255, 255, 255])

## white\_mask = cv2.inRange(image, lower, upper)

## # yellow color mask

## lower = np.uint8([190, 190, 0])

## upper = np.uint8([255, 255, 255])

## yellow\_mask = cv2.inRange(image, lower, upper)

## # combine the mask

## mask = cv2.bitwise\_or(white\_mask, yellow\_mask)

## masked = cv2.bitwise\_and(image, image, mask = mask)

## return masked

## white\_yellow\_images = list(map(select\_rgb\_white\_yellow, test\_images))

## show\_images(white\_yellow\_images)

## def convert\_gray\_scale(image):

## return cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

## gray\_images = list(map(convert\_gray\_scale, white\_yellow\_images))

## show\_images(gray\_images)

## def detect\_edges(image, low\_threshold=50, high\_threshold=200):

## return cv2.Canny(image, low\_threshold, high\_threshold)

## edge\_images = list(map(lambda image: detect\_edges(image), gray\_images))

## show\_images(edge\_images)

## #Identify area of interest

## def filter\_region(image, vertices):

## """

## Create the mask using the vertices and apply it to the input image

## """

## mask = np.zeros\_like(image)

## if len(mask.shape)==2:

## cv2.fillPoly(mask, vertices, 255)

## else:

## cv2.fillPoly(mask, vertices, (255,)\*mask.shape[2]) # in case, the input image has a channel dimension

## return cv2.bitwise\_and(image, mask)

## def select\_region(image):

## """

## It keeps the region surrounded by the `vertices` (i.e. polygon). Other area is set to 0 (black).

## """

## # first, define the polygon by vertices

## rows, cols = image.shape[:2]

## pt\_1 = [cols\*0.05, rows\*0.90]

## pt\_2 = [cols\*0.05, rows\*0.70]

## pt\_3 = [cols\*0.30, rows\*0.55]

## pt\_4 = [cols\*0.6, rows\*0.15]

## pt\_5 = [cols\*0.90, rows\*0.15]

## pt\_6 = [cols\*0.90, rows\*0.90]

## # the vertices are an array of polygons (i.e array of arrays) and the data type must be integer

## vertices = np.array([[pt\_1, pt\_2, pt\_3, pt\_4, pt\_5, pt\_6]], dtype=np.int32)

## return filter\_region(image, vertices)

## # images showing the region of interest only

## roi\_images = list(map(select\_region, edge\_images))

## show\_images(roi\_images)

## #Hough line transform

## def hough\_lines(image):

## """

## `image` should be the output of a Canny transform.

## 

## Returns hough lines (not the image with lines)

## """

## return cv2.HoughLinesP(image, rho=0.1, theta=np.pi/10, threshold=15, minLineLength=9, maxLineGap=4)

## list\_of\_lines = list(map(hough\_lines, roi\_images))

## def draw\_lines(image, lines, color=[255, 0, 0], thickness=2, make\_copy=True):

## # the lines returned by cv2.HoughLinesP has the shape (-1, 1, 4)

## if make\_copy:

## image = np.copy(image) # don't want to modify the original

## cleaned = []

## for line in lines:

## for x1,y1,x2,y2 in line:

## if abs(y2-y1) <=1 and abs(x2-x1) >=25 and abs(x2-x1) <= 55:

## cleaned.append((x1,y1,x2,y2))

## cv2.line(image, (x1, y1), (x2, y2), color, thickness)

## print(" No lines detected: ", len(cleaned))

## return image

## 

## line\_images = []

## for image, lines in zip(test\_images, list\_of\_lines):

## line\_images.append(draw\_lines(image, lines))

## 

## show\_images(line\_images)

## #Identify rectangular blocks of parking

## def identify\_blocks(image, lines, make\_copy=True):

## if make\_copy:

## new\_image = np.copy(image)

## #Step 1: Create a clean list of lines

## cleaned = []

## for line in lines:

## for x1,y1,x2,y2 in line:

## if abs(y2-y1) <=1 and abs(x2-x1) >=25 and abs(x2-x1) <= 55:

## cleaned.append((x1,y1,x2,y2))

## 

## #Step 2: Sort cleaned by x1 position

## import operator

## list1 = sorted(cleaned, key=operator.itemgetter(0, 1))

## 

## #Step 3: Find clusters of x1 close together - clust\_dist apart

## clusters = {}

## dIndex = 0

## clus\_dist = 10

## for i in range(len(list1) - 1):

## distance = abs(list1[i+1][0] - list1[i][0])

## # print(distance)

## if distance <= clus\_dist:

## if not dIndex in clusters.keys(): clusters[dIndex] = []

## clusters[dIndex].append(list1[i])

## clusters[dIndex].append(list1[i + 1])

## else:

## dIndex += 1

## 

## #Step 4: Identify coordinates of rectangle around this cluster

## rects = {}

## i = 0

## for key in clusters:

## all\_list = clusters[key]

## cleaned = list(set(all\_list))

## if len(cleaned) > 5:

## cleaned = sorted(cleaned, key=lambda tup: tup[1])

## avg\_y1 = cleaned[0][1]

## avg\_y2 = cleaned[-1][1]

## # print(avg\_y1, avg\_y2)

## avg\_x1 = 0

## avg\_x2 = 0

## for tup in cleaned:

## avg\_x1 += tup[0]

## avg\_x2 += tup[2]

## avg\_x1 = avg\_x1/len(cleaned)

## avg\_x2 = avg\_x2/len(cleaned)

## rects[i] = (avg\_x1, avg\_y1, avg\_x2, avg\_y2)

## i += 1

## 

## print("Num Parking Lanes: ", len(rects))

## #Step 5: Draw the rectangles on the image

## buff = 7

## for key in rects:

## tup\_topLeft = (int(rects[key][0] - buff), int(rects[key][1]))

## tup\_botRight = (int(rects[key][2] + buff), int(rects[key][3]))

## #print(tup\_topLeft, tup\_botRight)

## cv2.rectangle(new\_image, tup\_topLeft,tup\_botRight,(0,255,0),3)

## return new\_image, rects

## # images showing the region of interest only

## rect\_images = []

## rect\_coords = []

## for image, lines in zip(test\_images, list\_of\_lines):

## new\_image, rects = identify\_blocks(image, lines)

## rect\_images.append(new\_image)

## rect\_coords.append(rects)

## 

## show\_images(rect\_images)

## ('Num Parking Lanes: ', 12)

## ('Num Parking Lanes: ', 12)

## #Identify each spot and count num of parking spaces

## def draw\_parking(image, rects, make\_copy = True, color=[255, 0, 0], thickness=2, save = True):

## if make\_copy:

## new\_image = np.copy(image)

## gap = 15.5

## spot\_dict = {} # maps each parking ID to its coords

## tot\_spots = 0

## adj\_y1 = {0: 20, 1:-10, 2:0, 3:-11, 4:28, 5:5, 6:-15, 7:-15, 8:-10, 9:-30, 10:9, 11:-32}

## adj\_y2 = {0: 30, 1: 50, 2:15, 3:10, 4:-15, 5:15, 6:15, 7:-20, 8:15, 9:15, 10:0, 11:30}

## 

## adj\_x1 = {0: -8, 1:-15, 2:-15, 3:-15, 4:-15, 5:-15, 6:-15, 7:-15, 8:-10, 9:-10, 10:-10, 11:0}

## adj\_x2 = {0: 0, 1: 15, 2:15, 3:15, 4:15, 5:15, 6:15, 7:15, 8:10, 9:10, 10:10, 11:0}

## for key in rects:

## # Horizontal lines

## tup = rects[key]

## x1 = int(tup[0]+ adj\_x1[key])

## x2 = int(tup[2]+ adj\_x2[key])

## y1 = int(tup[1] + adj\_y1[key])

## y2 = int(tup[3] + adj\_y2[key])

## cv2.rectangle(new\_image, (x1, y1),(x2,y2),(0,255,0),2)

## num\_splits = int(abs(y2-y1)//gap)

## for i in range(0, num\_splits+1):

## y = int(y1 + i\*gap)

## cv2.line(new\_image, (x1, y), (x2, y), color, thickness)

## if key > 0 and key < len(rects) -1 :

## #draw vertical lines

## x = int((x1 + x2)/2)

## cv2.line(new\_image, (x, y1), (x, y2), color, thickness)

## # Add up spots in this lane

## if key == 0 or key == (len(rects) -1):

## tot\_spots += num\_splits +1

## else:

## tot\_spots += 2\*(num\_splits +1)

## 

## # Dictionary of spot positions

## if key == 0 or key == (len(rects) -1):

## for i in range(0, num\_splits+1):

## cur\_len = len(spot\_dict)

## y = int(y1 + i\*gap)

## spot\_dict[(x1, y, x2, y+gap)] = cur\_len +1

## else:

## for i in range(0, num\_splits+1):

## cur\_len = len(spot\_dict)

## y = int(y1 + i\*gap)

## x = int((x1 + x2)/2)

## spot\_dict[(x1, y, x, y+gap)] = cur\_len +1

## spot\_dict[(x, y, x2, y+gap)] = cur\_len +2

## 

## print("total parking spaces: ", tot\_spots, cur\_len)

## if save:

## filename = 'with\_parking.jpg'

## cv2.imwrite(filename, new\_image)

## return new\_image, spot\_dict

## delineated = []

## spot\_pos = []

## for image, rects in zip(test\_images, rect\_coords):

## new\_image, spot\_dict = draw\_parking(image, rects)

## delineated.append(new\_image)

## spot\_pos.append(spot\_dict)

## show\_images(delineated)

## final\_spot\_dict = spot\_pos[1]

## print(len(final\_spot\_dict))

## def assign\_spots\_map(image, spot\_dict=final\_spot\_dict, make\_copy = True, color=[255, 0, 0], thickness=2):

## if make\_copy:

## new\_image = np.copy(image)

## for spot in spot\_dict.keys():

## (x1, y1, x2, y2) = spot

## cv2.rectangle(new\_image, (int(x1),int(y1)), (int(x2),int(y2)), color, thickness)

## return new\_image

## marked\_spot\_images = list(map(assign\_spots\_map, test\_images))

## show\_images(marked\_spot\_images)

## # Save spot dictionary as pickle file

## import pickle

## with open('spot\_dict.pickle', 'wb') as handle:

## pickle.dump(final\_spot\_dict, handle, protocol=pickle.HIGHEST\_PROTOCOL)

## #Save image for CNN model

## def save\_images\_for\_cnn(image, spot\_dict = final\_spot\_dict, folder\_name ='for\_cnn'):

## for spot in spot\_dict.keys():

## (x1, y1, x2, y2) = spot

## (x1, y1, x2, y2) = (int(x1), int(y1), int(x2), int(y2))

## #crop this image

## # print(image.shape)

## spot\_img = image[y1:y2, x1:x2]

## spot\_img = cv2.resize(spot\_img, (0,0), fx=2.0, fy=2.0)

## spot\_id = spot\_dict[spot]

## 

## filename = 'spot' + str(spot\_id) +'.jpg'

## print(spot\_img.shape, filename, (x1,x2,y1,y2))

## 

## cv2.imwrite(os.path.join(folder\_name, filename), spot\_img)

## 

## # save\_images\_for\_cnn(test\_images[0])

## #Use trained CNN model to make predictions

## ## Imports for making predictions

## from PIL import Image

## from keras.applications.imagenet\_utils import preprocess\_input

## from keras.models import load\_model

## from keras.preprocessing import image

## cwd = os.getcwd()

## top\_model\_weights\_path = 'car1.h5'

## class\_dictionary = {}

## class\_dictionary[0] = 'empty'

## class\_dictionary[1] = 'occupied'

## from PIL import Image

## model = load\_model('/content/drive/MyDrive/train\_data/myModel.h5')

## def make\_prediction(image):

## #Rescale image

## img = image/255.

## #Convert to a 4D tensor

## image = np.expand\_dims(img, axis=0)

## #print(image.shape)

## # make predictions on the preloaded model

## class\_predicted = model.predict(image)

## inID = np.argmax(class\_predicted[0])

## label = class\_dictionary[inID]

## return label

## def predict\_on\_image(image, spot\_dict = final\_spot\_dict, make\_copy=True, color = [0, 255, 0], alpha=0.5):

## if make\_copy:

## new\_image = np.copy(image)

## overlay = np.copy(image)

## cnt\_empty = 0

## all\_spots = 0

## for spot in spot\_dict.keys():

## all\_spots += 1

## (x1, y1, x2, y2) = spot

## (x1, y1, x2, y2) = (int(x1), int(y1), int(x2), int(y2))

## #crop this image

## spot\_img = image[y1:y2, x1:x2]

## spot\_img = cv2.resize(spot\_img, (48, 48))

## 

## label = make\_prediction(spot\_img)

## #print(label)

## if label == 'empty':

## cv2.rectangle(overlay, (int(x1),int(y1)), (int(x2),int(y2)), color, -1)

## cnt\_empty += 1

## 

## cv2.addWeighted(overlay, alpha, new\_image, 1 - alpha, 0, new\_image)

## 

## cv2.putText(new\_image, "Available: %d spots" %cnt\_empty, (30, 95),

## cv2.FONT\_HERSHEY\_SIMPLEX,

## 0.7, (255, 255, 255), 2)

## 

## cv2.putText(new\_image, "Total: %d spots" %all\_spots, (30, 125),

## cv2.FONT\_HERSHEY\_SIMPLEX,

## 0.7, (255, 255, 255), 2)

## save = False

## 

## if save:

## filename = 'with\_marking.jpg'

## cv2.imwrite(filename, new\_image)

## 

## return new\_image

## predicted\_images = list(map(predict\_on\_image, test\_images))

## show\_images(predicted\_images)

## #Run code on video

## video\_name = 'cars8\_stat3.mp4'

## cap = cv2.VideoCapture(video\_name)

## ret = True

## count = 0

## while ret:

## ret, image = cap.read()

## count += 1

## if count == 5:

## count = 0

## 

## new\_image = np.copy(image)

## overlay = np.copy(image)

## cnt\_empty = 0

## all\_spots = 0

## color = [0, 255, 0]

## alpha=0.5

## for spot in final\_spot\_dict.keys():

## all\_spots += 1

## (x1, y1, x2, y2) = spot

## (x1, y1, x2, y2) = (int(x1), int(y1), int(x2), int(y2))

## #crop this image

## spot\_img = image[y1:y2, x1:x2]

## spot\_img = cv2.resize(spot\_img, (48, 48))

## label = make\_prediction(spot\_img)

## #print(label)

## if label == 'empty':

## cv2.rectangle(overlay, (int(x1),int(y1)), (int(x2),int(y2)), color, -1)

## cnt\_empty += 1

## cv2.addWeighted(overlay, alpha, new\_image, 1 - alpha, 0, new\_image)

## cv2.putText(new\_image, "Available: %d spots" %cnt\_empty, (30, 95),

## cv2.FONT\_HERSHEY\_SIMPLEX,

## 0.7, (255, 255, 255), 2)

## cv2.putText(new\_image, "Total: %d spots" %all\_spots, (30, 125),

## cv2.FONT\_HERSHEY\_SIMPLEX,

## 0.7, (255, 255, 255), 2)

## cv2.imshow('frame', new\_image)

## if cv2.waitKey(10) & 0xFF == ord('q'):

## break

## #out.write(image)

## cv2.destroyAllWindows()

## cap.release()

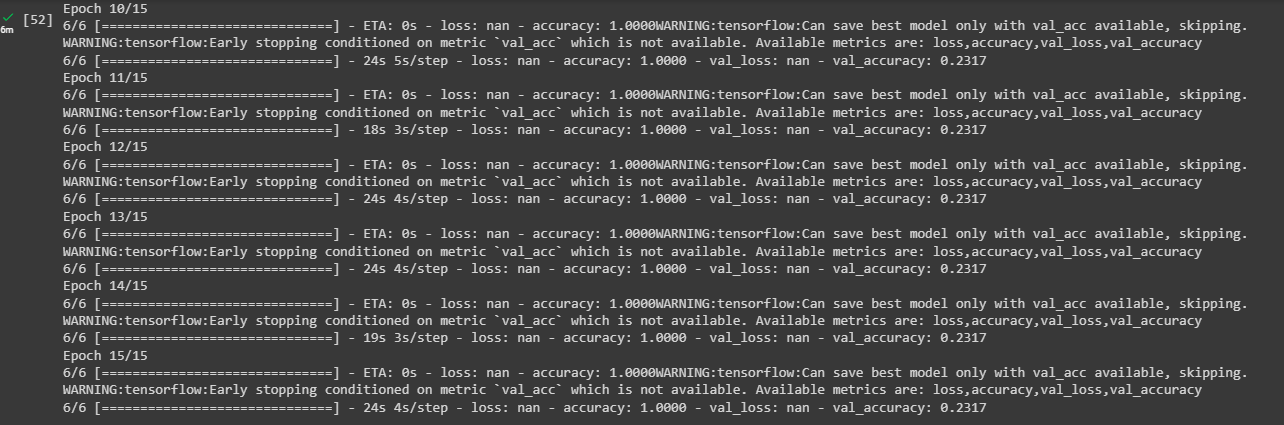
## CHAPTER 6

**RESULTS**

## 6.1 Performance Metrics

Performance Metrics:

1. Accuracy: The percentage of correctly detected car parking space availability in parking space. It is calculated by dividing the number ofempty spaces in the slot and the number the number occupied space in the slot from the total availability lot.



1. Parking Occupancy: As parking occupancy levels continue to fluctuate, it is essential to determine when and how long your visitors are parking. Data visualization reports such as heatmaps will allow you to analyze your occupancy levels based on date and time.

3. Execution Time: The time taken to execute the parking space availability and producing an efficiency to the model in real-time environment.

## CHAPTER 7

## ADVANTAGES & DISADVANTAGES

## ADVANTAGES:

## Optimized parking – Users find the best spot available, saving time, resources and effort. The parking lot fills up efficiently and space can be utilized properly by commercial and corporate entities.

## Reduced traffic congestion – Traffic is reduced because fewer cars are driving around looking for an available parking space.

## Enhanced User Experience – A smart parking solution enhances the overall user experience by providing real-time information about available parking spaces.

## Increased Revenue – Smart parking solutions can increase revenue for businesses by providing more efficient use of their parking spaces.

## Reduced Carbon Footprint – A smart parking solution reduces carbon emissions by reducing the amount of time cars spend driving around looking for an available parking space

## DISADVANTAGES:

## Expensive Construction & Installation - A parking management system can cost a lot of money. For example, the statistical feature, ticketing technology, and reporting tools are just some things that increase the price.

## Requires Regular Maintenance - The parking systems are usually automated, but they require regular maintenance to ensure everything is working smoothly.

## System Breakdown - The parking management system can break down at any time, which can cause inconvenience to the users

## CHAPTER 8

## CONCLUSION

## The system benefits of smart parking go well beyond avoiding the needless circling of city blocks. It also enables cities to develop fully integrated multimodal intelligent transportation systems that don’t rely on cars in the first place. Developing smart parking solutions within a city requires data standardization and management; mobile phone integration; hardware and software innovation; and coordination among various stakeholders . These technical solutions and stakeholders are the same data structures and development groups integral to making a smart phone -enabled, multimodal, fully integrated transportation solution a reality. In effect, the technical enablers and multi-stakeholder coordination effort behind development of a local smart parking solution creates a launch pad toward full transportation system integration.

## CHAPTER 9

## FUTURE SCOPE

## In the future, car parking systems are likely to incorporate advanced technologies and innovative solutions to enhance efficiency, convenience, and sustainability. Here are some potential features and components of a future schema in a car parking system:

## Smart Sensors: Parking spaces could be equipped with advanced sensors capable of detecting the presence or absence of vehicles. These sensors can transmit real-time data to a centralized system, enabling accurate monitoring and management of available parking spaces.

## Automated Guidance Systems: To streamline the parking process, future car parking systems may employ automated guidance systems. These systems could utilize various technologies like artificial intelligence, computer vision, and machine learning to guide drivers to available parking spaces through digital signage or in-car navigation systems.

## Integrated Payment Systems: Future car parking schemas may incorporate seamless payment systems. Drivers could make payments electronically through mobile apps, digital wallets, or integrated vehicle payment systems. This eliminates the need for physical tickets or payment machines, reducing the time spent in the parking process.

## Vehicle Recognition Technology: Advanced vehicle recognition technology, such as license plate recognition or RFID (Radio Frequency Identification), can be integrated into parking systems. This allows for quick identification and authentication of vehicles, ensuring efficient access control and security.

## Real-Time Data and Analytics: Future parking systems could collect and analyze real-time data on parking space utilization, occupancy rates, and traffic flow. This information can be used to optimize parking operations, predict peak demand periods, and provide valuable insights for urban planning and traffic management.

## Intelligent Parking Reservations: A future schema may include intelligent parking reservation systems that enable drivers to pre-book parking spaces through online platforms or mobile apps. This feature can help minimize the time spent searching for parking, reduce congestion, and improve overall user experience.

## Sustainability and Green Initiatives: Future parking systems are likely to prioritize sustainability by integrating eco-friendly features. This may include incorporating electric vehicle charging stations, promoting shared mobility services, and implementing energy-efficient lighting systems to reduce environmental impact.

## Multi-modal Integration: To support the growing trend of multi-modal transportation, future car parking schemas could integrate with other transportation modes. This could involve linking parking systems with public transportation networks, bike-sharing services, or ride-hailing platforms, allowing users to seamlessly switch between different modes of transport.

## Remote Monitoring and Control: With the advent of Internet of Things (IoT) technology, future parking systems may enable remote monitoring and control. Operators can remotely manage parking facilities, monitor occupancy, adjust pricing, and detect and address any issues or maintenance requirements.

## Data Privacy and Security: As parking systems become more connected, data privacy and security will be of paramount importance. Future schemas will need to implement robust security measures to protect user information, prevent unauthorized access, and ensure the integrity of the parking system.

## It's important to note that the actual implementation of these features may vary depending on technological advancements, infrastructure development, and specific needs and requirements of different regions or organizations.

## CHAPTER 10

## APPENDIX

**Index.html**

<!DOCTYPE html>

<html>

<head>

<title>An Interacative Parking Map of Innsbruck</title>

</head>

<body id="page-top" class="index backgroundImage">

<!-- Navigation -->

<nav class="navbar navbar-default navbar-static-top">

<div class="container">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header page-scroll">

<button type="button" class="navbar-toggle" data-toggle="collapse"

data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#page-top">Start</a>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar- collapse-1">

<ul class="nav navbar-nav navbar-right">

<li class="hidden">

<a href="#page-top"></a>

</li>

<li class="page-scroll">

<a href="#parkraum">The Parking Space</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container-fluid -->

</nav>

<!-- Section -->

<section id="parkraum">

<div class="container">

<div class="row contentDiv">

<div class="page-header">

<h1>The Parking Space of Innsbruck</h1>

</div>

<p class="lead">This is an interactive map.

<hr>

<div>

<div class="form-group dropdown theme-dropdown clearfix">

<form id="mapForm">

<button id="getPositionButton" type="button" class="btn btn-sm btn-info" onclick="currentPosition();">Aktuelle Position anzeigen</button>

<label for="selectParkzone">Parkzone auswählen: </label>

<select id="selectParkzone" class="btn btn-sm">

<select>

<button id="resetButton" type="button" class="btn btn-sm btn-info" onclick="resetMap();">Zurücksetzen</button>

</form>

</div>

<div id="map" class="iframe-container"></div>

<hr>

</div>

</div>

<!-- /.row -->

</div>

</section>

<!-- End section -->

<!-- Bootstrap core JavaScript

<!-- Latest compiled and minified CSS -->

<linkrel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

<!-- Optional theme -->

<linkrel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstraptheme.min.cs" integrity="sha384rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

<scriptsrc="https://code.jquery.com/jquery-3.1.0.min.js" type="text/javascript"></script>

<link rel="stylesheet" href="./css/custom.css">

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.0-rc.3/dist/leaflet.css"/>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.3/css/font-awesome.min.css" />

<script src="https://unpkg.com/leaflet@1.0.0-rc.3/dist/leaflet.js"></script>

<script src="http://leaflet.github.io/Leaflet.label/leaflet.label.js"></script>

<script src="./js/parkraum.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<script>

$(document).ready(function() {

initMap();

placeZonesOnMap();

loadScrolling();

populateParkzoneDropdown();

});

</script>

< /body>

< /html>

**Parking.css**

html {

position: relative;

min-height: 100%;

}

body {

/\* Margin bottom by footer height \*/

margin-bottom: 60px;

}

.footer {

position: absolute;

bottom: 0;

width: 100%;

/\* Set the fixed height of the footer here \*/

height: 60px;

background-color: #f5f5f5;

}

/\* Custom page CSS

-------------------------------------------------- \*/

/\* Not required for template or sticky footer method. \*/

body > .container {

padding: 60px 15px 0;

}

.container .text-muted {

margin: 20px 0;

}

.footer > .container {

padding-right: 15px;

padding-left: 0px;

}

code {

font-size: 80%;

}

.iframe-container{

position: relative;

width: 100%;

height: 80%;

padding-bottom: 56.25%; /\* Ratio 16:9 ( 100%/16\*9 = 56.25% ) \*/

}

.iframe-container > \*{

display: block;

position: absolute;

top: 0;

right: 0;

bottom: 0;

left: 0;

margin: 0;

padding: 0;

height: 100%;

width: 100%;

}

.backgroundImage {

background-image:url('./../images/nordkette.png');

-webkit-background-size: cover;

-moz-background-size: cover;

-o-background-size: cover;

background-size: cover;

background-repeat: no-repeat;

}

.contentDiv {

# background-color: #ECECEA;

background-color: rgba(255,255,255, 0.8);

# background-color: white;

padding-left: 0.5cm;

padding-right: 0.5cm;

position: relative;

}

p {

font-family: ‘Lucida Sans Unicode’, ‘Lucida Grande’, sans-serif;

}

hr {

height: 30px;

border-style: solid;

border-color: #8c8b8b;

border-width: 1px 0 0 0;

border-radius: 20px;

}

.legend {

line-height: 18px;

color: #555;

}

.legend i {

width: 18px;

height: 18px;

float: left;

margin-right: 8px;

opacity: 0.7;

}

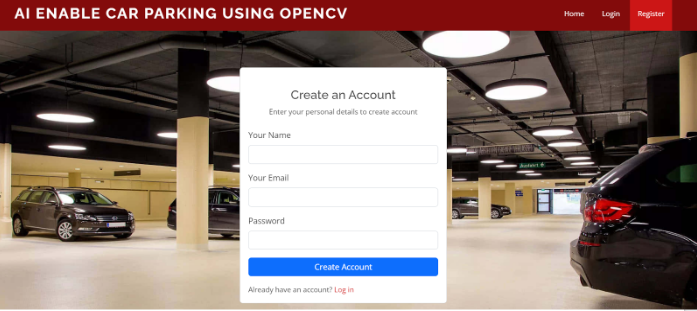
## APPENDIX 1

## SCREENSHOTS

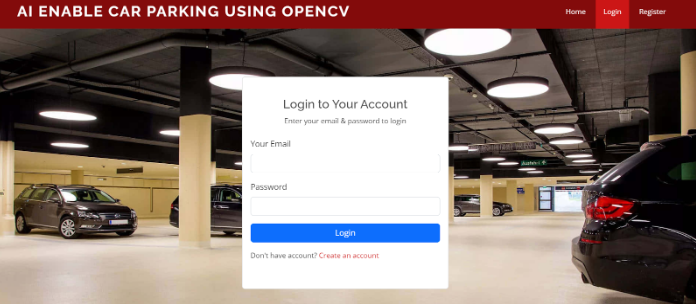
**FIGURE A1.1 HOME PAGE**



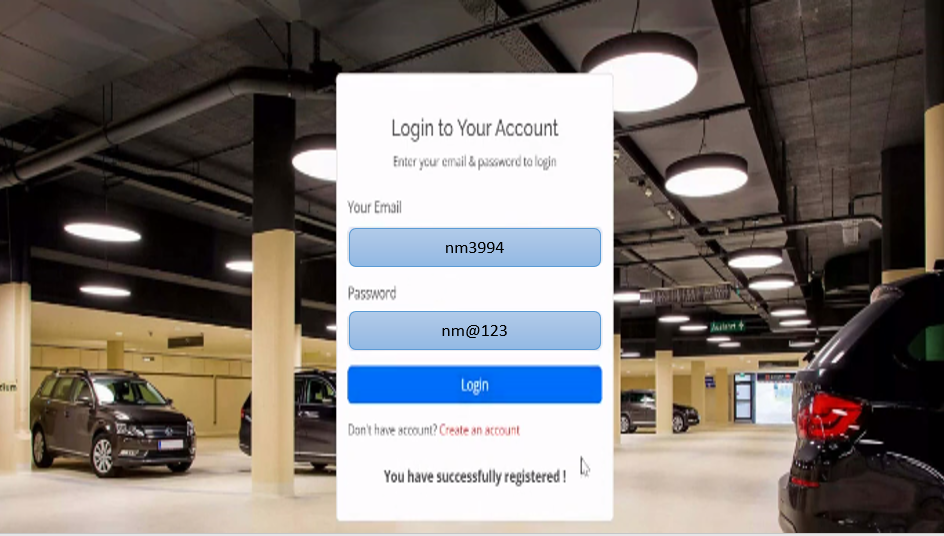
**FIGURE A1.2 USER LOGIN**



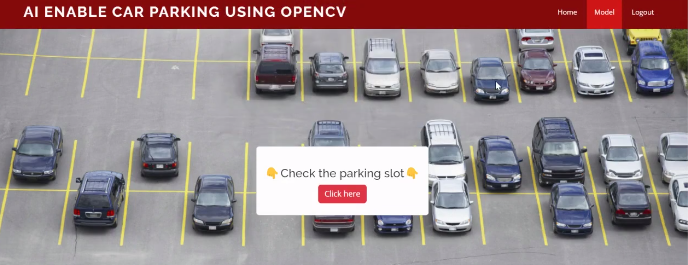
**FIGURE A1.4 REGISTERATION OF THE USER**

****

**FIGURE A1.6 ACCOUNT REGISTERATION**

****

**FIGURE A1.7 AVAILABILITY**

****

## APPENDIX 2

## SAMPLE CODING

import cv2

import os

from flask import Flask, request, render\_template

from datetime import date

from datetime import datetime

import numpy as np

from sklearn.neighbors import KNeighborsClassifier #scikit-learn

import pandas as pd

import joblib

#### Defining Flask App

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

#### Saving Date today in 2 different formats

datetoday = date.today().strftime("%m\_%d\_%y")

datetoday2 = date.today().strftime("%d-%B-%Y") #date fullMonthName year

#### Initializing VideoCapture object to access WebCam

face\_detector = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

#try:

# cap = cv2.VideoCapture(1)

#except:

cap = cv2.VideoCapture(0)

for user in userlist:

for imgname in os.listdir(f'static/faces/{user}'): #returns length from the list of static -> faces

img = cv2.imread(f'static/faces/{user}/{imgname}')

#imread() Loads an image from a file.the function returns an empty matrix

resized\_face = cv2.resize(img, (50, 50)) #resizes width and height to (50,50)

faces.append(resized\_face.ravel()) #ravel() returns 1D array of resized\_face

labels.append(user)

faces = np.array(faces)

knn = KNeighborsClassifier(n\_neighbors=5) # KNeighborsClassifier looks for the 5 nearest neighbors.

knn.fit(faces, labels) #

joblib.dump(knn, 'static/face\_recognition\_model.pkl')

#joblib is faster in saving/loading large NumPy arrays, whereas pickle is faster with large collections of Python objects

#dump - used to write any object to the binary file

#load - used to read object from the binary file.

#### Our main function which runs the Flask App

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

files = dir('template-database/\*.mat');

numfile = length(files);

if numfile == 0

id\_acc = -1;

return

end

%% Calculate the Hamming distance bw extracted feature with the database

hm\_dist = zeros(1, numfile);

for i = 1 : numfile

% Load each account

clear template mask name exinfo

load(['template-database/', num2str(i), '.mat'])

% Calculate the Hamming distance

template\_extr = reshape(template\_extr, 20, 480);

mask\_extr = reshape(mask\_extr, 20, 480);

template = reshape(template, 20, 480);

mask = reshape(mask, 20, 480);

hm\_dist(1,i) = gethammingdistance(template\_extr, mask\_extr, template, mask, 1);

end

%% Threshold and give the result ID

id\_acc = find(hm\_dist <= threshold);

if length(id\_acc) < 1

id\_acc = 0;

else if length(id\_acc) > 1

id\_acc = 1;

end

end