BILKENT UNIVERSITY

Department of Computer Engineering

SENIOR DESIGN PROJECT

Parkhound: Parking Spot Detection and Navigation System

Low Level Design Report

Group Members
Arda Türkoğlu
Ege Turan
Berkin Inan
Görkem Yılmaz
Ata Coşkun

Supervisor
Varol Akman

Jury Members
Uğur Doğrusöz
Ercüment Çiçek

Contents

1 Introduction

With the constant increase in population and owned cars in cities, finding a parking spot becomes a more prominent problem. Drivers spend hours around a parking lot to find an available spot to find their cars [4]. A 35% travel during rush hours looking for free parking spots that are hard to find. This makes parking very important to regulate vehicle flow and reduce atmospheric pollution [5].

This search causes drivers to waste time and gas. Therefore, parking systems are very important for regulating the traffic flow and sustaining transportation of the city. Because of the increasing number of personal vehicles, many foundations cannot provide enough parking slots or their parking areas face several parking problems. Since foundation can have many different and large-scale parking areas.

In addition to parking availability, there are two other aspects of outdoor vehicle parks. One of them is the security of the parking area which includes car security and protecting the area of the property. It is not always easy to control a large area separated for cars and this uncontrolled area can be a place for illegal activities. Secondly, some outdoor vehicle parks demand some regulations about parking, such as placing the car between the white lines. Normally, this kind of problem is handled by the security personnel in the foundation. They are walking around and making controls but again, this is not easy and has a high percentage of error-prone in the large outdoor vehicle parks.For almost a decade, many foundations have used sensor-based control parking slot maintenance systems in closed car parking spaces.

However, the management of the outdoor car parking spaces is not easy for sensory systems due to the safety of the devices and the placement difficulties faced by the producers. Also, these sensory systems cannot handle many security issues and regulations about parking. Therefore, we came up with an idea of creating a computer vision system named ParkHound to maintain the parking spots in outdoor vehicle cars. Our system is based on the image processing unit and machine learning system that uses predetermined data set for understanding the status of the parking slot in the open air and maintaining the specified regulations and the security concerns. Our system will be implemented on the camera system and can be used for the large-scale open vehicle parks.

1.1 Object Design Trade-offs

1.1.1 Functionality vs. Security

In the initial plans, we intended to show the live camera footage to the users for them to better understand the location of the parking spaces. However, this would create issues about security because live recordings of these cameras may not be allowed to be public. We decided to remove showing the live recording to users. However, in practice our system will be used as a guide to the parking area security. In general, security office is responsible from the parking area and they have credential to use camera record. In commercial level, our system will become automatic and it's data only be controlled by the security.

1.2 Compatibility vs Extensibility

ParkHound is designed as cross-platform software. It has mobile and web support. This bring a lot of development work. But we are using React JS for our web and cross-platform. Using same language and development tool for mobile and web user-interface will provide flexibility for our system. Updates and the change for new features will be reflected easily. In addition, our system design based on packages and those packages are divided into controllers and working modules. This modular design enable us to extend our system easily. With that design approach, we handle

compatibility target by serving desired product and providing good architecture for extensibility of our system.

1.3 Space vs Time

Our server will make predictions on the video frames. Thus, we need to have some video data in the server. We can minimize this data size by increasing the run-time of our model. Thus running pour model on the CPU is not a good idea for our run-time. Because we want to give real-time update to the parking area. To handle this we run our model on GPU. Using CUDA cores for matrix operations in deep learning provide better run-time. Also, adapting our system for several GPUs will also increase our run-time. By using parallel computing principles we will both decrease data amount in the server and the run-time of our system.

1.4 Robustness vs Cost

Main working principle of the ParkHound depends on the deep learning model for car detection. As parking areas get larger processing time needed for predictions will increase. Instead of deploying our model to small server, we need to deploy our model to cloud environment with high processing feature. This increases the cost for our system. To ensure robustness we will test performance of our system on different deployment environments. That way we will provide robust and affordable service for our customers.

1.5 Interface Documentation Guidelines

Class	Name of the class
Description	Description of the class.
Attributes	
Name of an attribute	Description of attribute
Methods	
Signature of method	Description of method

1.6 Engineering Standards

Throughout this report, as well as in our previous ones, we have used the Unified Modeling Language Standard (UML) [2] for the diagrams depicting the details of the system. Also, IEEE (Institute of Electrical and Electronics Engineers) [1] citation style has been used in our references.

1.7 Definitions, Acronyms, and Abbreviations

- **Parking Lot**: Collection of designated areas for vehicles to park. A parking lot can be composed of Parking Zones.
- **Parking Zone**: Areas inside the parking lot that divides the lot into manageable sections. A parking zone contains many parking spaces.
- **Parking Space**: A single spot where a vehicle can be parked on.
- **CUDA**: CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs[3].

2 Packages

2.1 Packages' Diagram

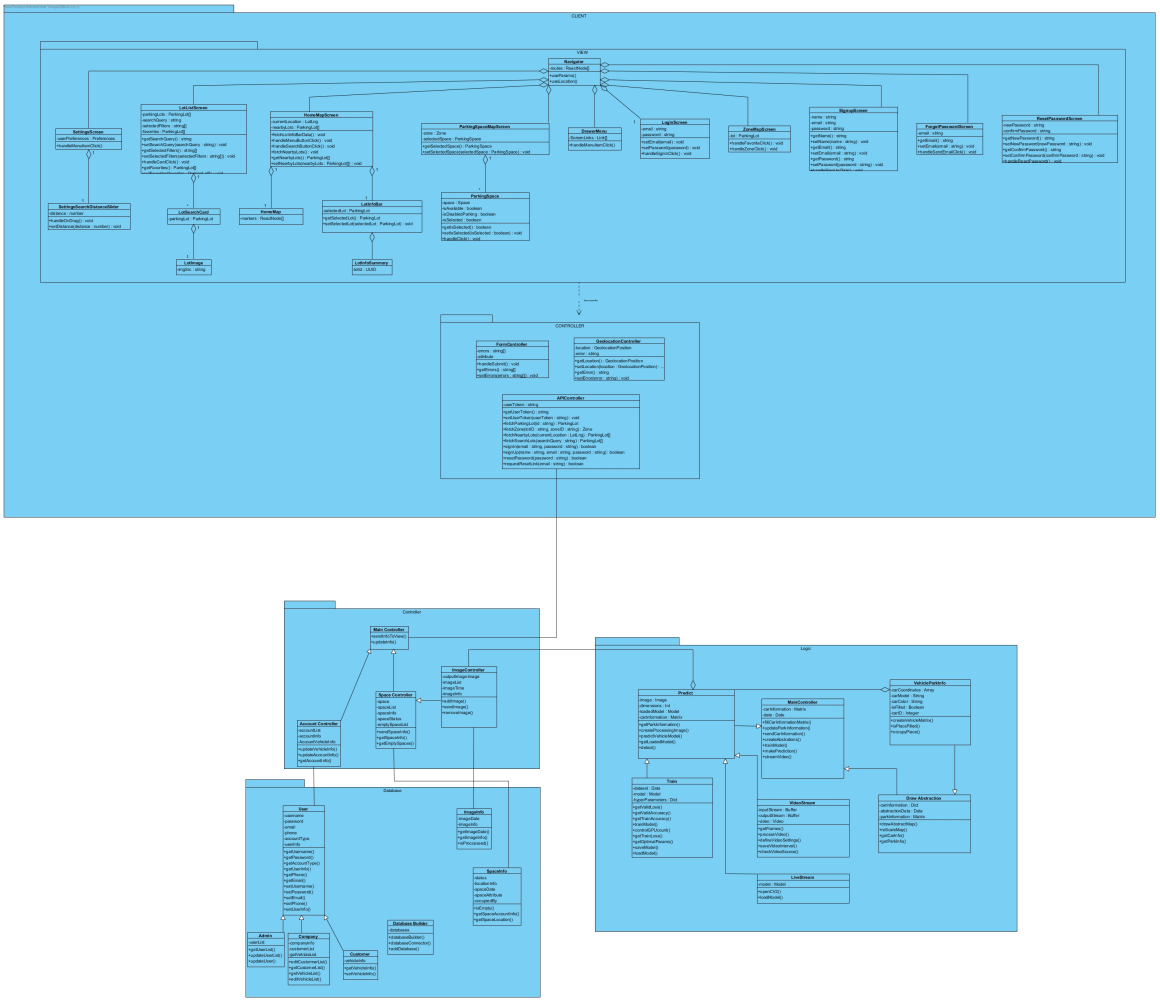

Figure 1: Full Diagram of the System of Parkhound

2.2 Client

The client consists of 4 main structures. These structures are screens, components, navigation and global state. While screens are mostly self-contained, other structures are shared throughout the application. Screens are connected to the global state and responsible for rendering components on a page. Components are the small parts that are either stateful or stateless to construct a screen. Navigation handles the transition between the pages and global state is the data that is shared by the all structures, such as user account information.

2.2.1 Screens

- **HomeMapScreen**: This screen will be the default screen when the user launches application. This screen will show a map centered around the current location of the user and display the nearby parking lots. Number of available spaces will be shown on the icons of the parking lots. If there are no available space in a lot, the lot icon will be colored differently. This screen contains the LotInfoBar component for a quick access to information about a selected lot.
- **ZoneMapScreen**: This screen will display the parking zones inside a lot. Zones are groups of parking spaces to manage the spaces more easily. The parking zones will be shown in a top-down manner, similar to a map of the parking lot. Zones with available space will shown with a different color than the zones at full capacity. The parking lot's name will displayed at the top and can be pressed to view detailed information about the lot's capabilities. The user can add the lot the favorites list and can also navigate back to the HomeMapScreen.
- **ParkingSpaceMapScreen**: This screen displays the available parking spaces inside a zone. Spaces are spots inside a zone that a car can be placed. Available spaces are shown with a different color than the spaces that are full. Entry and exit are displayed on the map for the user the orient themselves. A selected space is highlighted and the user can navigate to that spot by giving them directions.
- **LotListScreen**: This screen lists the nearby parking lots as cards. On the cards, name of the parking lot, distance from the user's current location and the number of available spaces are displayed. If the user has marked any parking lot as a favorite, they are displayed at the top of the list, separated from the other parking lots. The user can search and filter the parking lots on this screen.
- **SettingsScreen**: This screen lists the options that user can modify. These options are profile information, theme, notification settings and language. User can also change the search distance radius for searching parking lots.
- **ForgotPasswordScreen**: The user can request a password reset link to be sent to their registered email address from this screen. The user can also return to the sign-in screen.
- **ResetPasswordScreen**: After clicking the password reset link, the user is directed to this screen. They can enter a new password to change the password of their account.
- **LoginScreen**: On this screen, the user can enter their email and password to login to their account. From this screen, the user can navigate to forgot password or sign-up screen.
- **SignupScreen**: On this screen, the user can create an account by entering their name, email and password. From this screen, the user can navigate to login screen.

2.2.2 Components

- **HomeMap**: This is the map component on the home screen. This component will show the parking lots and the user's current location on the map.
- **LotInfoBar**: This component will show information about a selected lot such as number of available spots, address, distance and time to reach the destination. It is rendered inside the HomeMapScreen.

- **LotInfoSummary**: Given the lot's id, this component will contain the information about the lot as a summary.
- **ParkingSpace**: This component shows the state of a single space in a zone. Possible states are full or available. It can also show special icons such as disabled parking, to show which spaces are suitable for disabled parking.
- **LotSearchCard**: This component shows the general information about a lot. It is used to list the lots when searching and filtering.
- **LotImage**: This component renders the image of the lot. It is used in LotSearchCard.
- **DrawerMenu**: A sidebar menu to access the main screens of the app quickly. These screens include HomeMap, Parking Lots and Settings.
- **SettingsSearchDistanceSlider**: A slider component for user to adjust the search range of the parking lots nearby.

2.2.3 Navigation

Navigation is a controller that handles the transition between the two pages. When transitioning from one screen to another, the previous screen is unmounted and freed from the memory, while the new one is mounted. Navigation controller handles which screen matches the current path and mounts that screen. It also enables sharing data between two screen, without the use of the Global State.

2.2.4 Global State

Global State is a state accessible by all the components. It is required because of the tree hierarchy in React. If data needs to be accessed from a component several levels deep inside the application, either the data needs to passed down from all the parent components or a shared state is required. Former method creates a codebase that is difficult to read and maintain. Global State contains data such as user authentication token, user info and preferences.

2.3 Controller

The Controller Unit provides communication between views, database and logic unit. Depending on the user's requests, it reads and writes to the database, reads from the logic unit and responds to requests from the view unit.

- **Main Controller**: This controller organizes the account and space data which come from the database and logic unit.
- **Account Controller**: This controller collects appropriate account information from the database and writes new updates to the database.
- **Space Controller**: This controller collects appropriate space information from the database or logic unit through the image controller(depends on user's request) and writes new updates to the database.

- **Image Controller**: This controller is in the bridge role database and logic unit. Some requests require data retrieval directly from the logic unit and for this situation the image controller sends the output image of the logic to the space controller. Some requests require data retrieval from the database and the image controller reads data from the database and sends it to the main controller through the space controller.

2.4 Database

The Database Unit provides the databases and all data of the system such as account information and processed images from the logic unit. Depending on the other packages requests, it stores and writes data to databases. Also, sends requested data to the controller unit.

- **User**: This class is the parent class that is responsible for general user tasks such as storing user information.
- **Admin**: This class has the admin role. Admins are able to manage user accounts and edit their information.
- **Company**: This class represents the park owners in the system.
- **Customer**: This class represents the accounts that use the system to search and collect data from parking areas.
- **Database Builder**: This class is responsible for initiating the database and its appropriate conditions.
- **Image Info**: This class is responsible for organizing raw results of the image processing.
- **Space Info**: This class is responsible for organizing image information to determine space information of the parks.

2.5 Logic

The Logic Unit manages the model and parking area classification. It uses model with its dataset and processes the video stream. It converts the video stream into frames and uses predict method with the model to get information of the parking area. Logic can use live stream and given video stream to make its prediction. After that it creates the VehicleParkInfo object to update park information. In addition, it creates abstract representation of the parking area with using predefined model. Thus, system can both update and create parking information using the Logic package.

- **Train**: This class trains a deep learning model to be used later in the prediction. Parked car images are the training data and a video will be used to test the model's accuracy and loss.
- **Predict**: This class makes the prediction to determine how many places in a car park are empty or occupied. It uses an input video stream in order make prediction on that data.
- **Video Stream**: This class loads the video and processes it to get a frame by frame input for prediction. It also saves specific video intervals and defines the video's settings.
- **Draw Abstraction**: This class gets the car park information from the Main Controller and draws a visual sketch of the parking lot in application for users to see. It gets the car and park information from the input data object and re-scales the sketch accordingly.

- **Vehicle Park Info**: This class represents the information of the parking area. After model predictions created with the video stream, Vehicle park info is updated to represents the current situation of the parking area. This information used too update the database.
- **Live Stream**: This class will be used in live stream video inputs for our live prediction. An OpenCV module will be used to process input (live stream) and determine the empty and occupied parking spaces moment to moment.
- **Main Controller**: This class can be considered as the controller of other classes. It fills the car information matrix that is used in some other classes and sends the matrix to those classes. It controls the training, prediction and live video processing procedures.

3 Class Interfaces

3.1 Client

3.1.1 View

Class	Navigator
Description	This class handles the rendering of the active screen
Attributes	
routes	Array of the routes that can be rendered in the application
Methods	
useParams()	Exposed from this class, handles transferring parameters between two screens
useLocation()	Exposed from this class, handles programmatic navigation between screens

Class	HomeMapScreen
Description	This class is the initial screen the user sees. It contains the map.
Attributes	
currentLocation	Current location of the user, contains the latitude and longitude of the location
nearbyLots	Data of the parking lots near the user's current location
Methods	
fetchLotInfoBarData()	Requests the data of the selected parking lot to be shown in the LotInfoBar
handleMenuButtonClick()	Method to perform when the Menu button is pressed. Shows the drawer menu.
handleSearchButtonClick()	Method to perform when the Search button is pressed. Directs the user to the LotListScreen
fetchNearbyLots()	Requests the data of the nearby parking lots to be shown in the map.
getNearbyLots()	Accessor of the nearbyLots attribute
setNearbyLots()	Mutator of the nearbyLots attribute

Class	ZoneMapScreen
Description	This class is the screen that shows the zones in a parking lot.
Attributes	
lot	Data of the parking lot currently being displayed
Methods	
handleFavoriteClick()	Method to perform when the Favorite icon is pressed. Adds the lot to the favorites.
handleZoneClick()	Method to perform when a zone on the map is pressed.

Class	ParkingSpaceMapScreen
Description	This class is the screen that shows the parking spaces in a zone.
Attributes	
zone	Data of the zone currently being displayed. Received as prop from the ZoneMapScreen.
selectedSpace	Data of the parking space currently selected. Shows the space the user intends to park.
Methods	
getSelectedSpace()	Accessor of the selected parking space data
setSelectedSpace()	Mutator of the selected parking space data

Class	LotListScreen
Description	This class is the screen that shows the list of parking lots
Attributes	
parkingLots	List of the parking lot to be displayed on the screen
searchQuery	User's query for searching for a parking lot
selectedFilters	Filters selected by the user to filter the list
favorites	Parking lots marked as favorite by the user
Methods	
getSearchQuery()	Accessor of the search query in the search input
setSearchQuery()	Mutator of the search query in the search input, called when input value changes
getSelectedFilters()	Accessor of the filters to be used in the filtering the list
setSelectedFilters()	Mutator of the filters, called when user changes the filters
handleCardClick()	Method to be called when a parking lot card is pressed
getFavorites()	Accessor of the favorites to be shown in the list separately
setFavorites()	Mutator of the favorites, called when the favorites are fetched from the server

Class	SettingsScreen
Description	This class is the screen that shows menu of the user preferences
Attributes	
userPreferences	User's preference data of the account and the application
Methods	
handleMenuItemClick()	Method called when a menu item is pressed

Class	ForgotPasswordScreen
Description	This class is the screen that user can request a password reset
Attributes	
userPreferences	User's preference data of the account and the application
Methods	
handleMenuItemClick()	Method called when a menu item is pressed

Class	ForgotPasswordScreen
Description	This class is the screen that user can reset their password. This screen is accessed by a link sent to the user's email
Attributes	
newPassword	New password entered by the user
confirmPassword	Retyping of the new password entered by the user
Methods	
getNewPassword()	Accessor of the new password from the new password input field
setNewPassword()	Mutator of the new password. Called when the new password input field is changed
getConfirmPassword()	Accessor of the retyping of the new password from the confirm password input field
setConfirmPassword()	Mutator of the retyping of the new password. Called when the confirm password input field is changed
handleResetPassword()	Method called when the Reset Password button is pressed. Sends a request to the server to replace the user's password with a new one entered by the user.

Class	LoginScreen
Description	This class handles the user login
Attributes	
email	Email address entered by the user
password	Password entered by the user
Methods	
setEmail()	Mutator of the email input value
setPassword()	Mutator of the password input value
handleSignInClick()	Initiates the login process with the current email and password values

Class	SignupScreen
Description	This class handles account creation
Attributes	
name	Name entered by the user
email	Email address entered by the user
password	Password entered by the user
Methods	
getName()	Accessor of the name input value, from the name input field
getEmail()	Accessor of the email input value, from the email input field
getPassword()	Accessor of the password input value, from the password input field
setName()	Mutator of the name input value
setEmail()	Mutator of the email input value
setPassword()	Mutator of the password input value
handleSignUpClick()	Initiates the account creation process with the current name, email and password values

Class	HomeMap
Description	This component displays the map and markers on it
Attributes	
markers	Locations of the elements that will be displayed on the map. It is received from the HomeMapScreen

Class	LotInfoBar
Description	This component displays information about the selected lot
Attributes	
selectedLot	Data of the parking lot which is currently selected
Methods	
getSelectedLot()	Accessor of the selected lot data
setSelectedLot()	Mutator of the selected lot, called by parent of the component

Class	LotInfoSummary
Description	This component displays summary information about the lot it received
Attributes	
lotId	Unique ID of the parking lot

Class	ParkingSpace
Description	This component shows the state of a single space in a zone.
Attributes	
space	Data of the parking space, such as which parking lot and zone it belongs in
isAvailable	Boolean showing if the space is available or not
isDisabledParking	Boolean showing if the space is reserved for disabled parking or not
isSelected	Boolean showing if the space is selected by the user or not
Methods	
getIsSelected()	Accessor of the selected state of the space
setIsSelected()	Mutator of the selected state of the space
handleClick()	Method called when the space is pressed on

Class	LotSearchCard
Description	This component shows the general information about a lot on a card for the listings
Attributes	
parkingLot	Data of the parking lot

Class	DrawerMenu
Description	This component shows a menu for quick access to other screens
Attributes	
screenLinks	Links to the other screens
Methods	
handleMenuItemClick()	Method called when a menu item is pressed on

Class	SettingsSearchDistanceSlider
Description	This is a component for adjusting the search range between a limited range
Attributes	
distance	Radius of the search circle in kilometers
Methods	
handleOnDrag()	Method called when the slider is changed by dragging
setDistance()	Mutator of the distance value

3.2 Controller

Class	Main Controller
Attributes	
accountController	Account information which are send to appropriate views.
spaceController	Space informations which are send to appropriate views
Methods	
sendInfoToView()	The method that send space and account information to the appropriate views.
updateInfo()	Mutator method for account and space information.

Class	Account Controller
Attributes	
accountList	Array of current accounts in the system.
accountInfo	Account information of the active user.
accountVehicleInfo	Vehicle information of the active user.
Methods	
updateVehicleInfo()	Mutator method for accountVehicleInfo.
updateAccountInfo()	Mutator method for accountInfo.
getAccountInfo()	Accessor method for account information.

Class	Space Controller
Attributes	
space	Space object.
spaceList	List of spaces.
spaceInfo	Information about the spaces.
spaceStatus	Status of the spaces.
emptySpaceList	List of empty spaces.
Methods	
sendSpaceInfo()	The method that sends space informations to appropriate views.
getSpaceInfo()	Accessor method for spaces.
getEmptySpaces()	Accessor method for emtpySpaceList.

Class	Image Controller
Attributes	
outputImage	Space object.
imageList	List of images.
imageTime	The time of the image information.
imageInfo	Image data that is send to database to record it.
Methods	
addImage()	Add captured image to the system.
sendImage()	The method that sends the image to the database.
removeImage()	The method that removes the image from the database.

3.3 Database

Class	User
Attributes	
username	Name of the user's account.
password	Password of the user's account.
email	Email of the user's account.
phone	Phone number of user's account.
Methods	
getUserName()	Accessor method for username.
getPassword()	Accessor method for password.
getAccountType()	Accessor method for account type.
getUserInfo()	Accessor method for user information.
getPhone()	Accessor method for phone.
getEmail()	Accessor method for email.
setUserName()	Mutator method for username.
setPassword()	Mutator method for password.
setEmail()	Mutator method for email.
setPhone()	Mutator method for phone.
setUserInfo()	Mutator method for user information.

Class	Admin
Attributes	
userList	Array of users in the system.
Methods	
getUserList()	Accessor method for userList.
updateUserList()	Mutator method for userList.
updateUser()	Mutator method for specific user whose information updated by admin.

Class	Company
Attributes	
companyInfo	Space object.
customerList	List of customers who are members of the company.
vehicleList	List of vehicles who are members of the company.
Methods	
setCustomerList()	The method that sends space informations to appropriate views.
setVehicleList()	Accessor method for spaces.
getCustomerList()	Accessor method for emtpySpaceList.
getVehicleList()	Accessor method for emtpySpaceList.

Class	Customer
Attributes	
vehicleInfo	Information about the customer's vehicle.
Methods	
getVehicleInfo()	Accessor method for vehicleInfo.
setVehicleInfo()	Mutator method for vehicleInfo.

Class	Database Builder
Attributes	
databases	Array of current databases in the system.
Methods	
databaseBuilder()	Initiator method for database.
databaseConnector()	The method that start or terminate the connection with database.
addDatabase()	Add new database to the system.

Class	Image Info
Attributes	
ImageInfo	Image configuration(matrix) which are taken from logic module.
ImageDate	The time informatiıon of the image configuration.
Methods	
getImageInfo()	Accessor method for imageInfo.
getImageDate()	Accessor method for imageDate.
isProcessed()	The method that checks whether the image process is done in logic unit.

Class	Space Info
Attributes	
status	Status of the space.
locationInfo	Location of the space such as lot, zone etc.
spaceDate	The time information about that space info.
occupiedBy	Account information of someone who occupies the space.
Methods	
isEmpty()	The method that checks whether the space is empty or not.
getSpaceAccountInfo()	Accessor method for space account information.
getSpaceLocation()	Accessor method for space location.

3.4 Logic

Class	Predict
Attributes	
image	Input image will be used in prediction.
dimensions	Dimensions of the model prediction constraint
loadedModel	Pre-trained TensorFlow or Keras model.
carInformation	Acquired information about car from the prediction.
Methods	
getParkInformation()	Information about the empty and occupied spaces that is acquired from the prediction.
createProcessingImage()	Prepare image for model processing.
predictVehiceModel()	Use model to predict information about some of the features of parked cars such color and shape.
getLoadedModel()	Get the pre-trained model for prediction.
detect()	Make predictions about car park information.

Class	Train
Attributes	
dataset	Dataset that used to train the model.
model	A model that is created with certain hyperparameters that will be trained with the dataset
hyperParameters	Parameters whose values will used to control the learning process.
Methods	
getValidLoss()	Accessor method to get validation loss of the training.
getValidAccuracy()	Accessor method to get validation accuracy of the training.
getTrainAccuracy()	Accessor method to get training accuracy of the training.
trainModel()	Use dataset to train the model.
controlGPUcount()	Control GPU count for the training.
getTrainLoss()	Accessor method to get the training loss.
getOptimalParams()	This function determines the optimal parameters for the training procedure.
saveModel()	This function saves the model on disk to load and use in prediction.
loadedModel()	A method to load a pre-trained model.

Class	VideoStream
Attributes	
inputStream	Input video that will be used in prediction.
outputStream	Output video streaming that will be processed & shows labels
video	Resulting video.
Methods	
getFrames()	It processes the video to divide it into frames.
processVideo()	It processes the frames by making prediction on it.
defineVideoSettings()	It defines the settings for video such as light and contrast to get a better prediction results.
saveVideoInterval()	It saves specified video interval on the disk
checkVideoSource()	Check if the video is valid.

Class	LiveStream
Attributes	
model	Deep Learning model that will used in live prediction.
Methods	
openCV2()	A module that will be used in live video prediction.
loadModel()	It loads the pre-trained model to make prediction.

Class	Draw Abstraction
Attributes	
carInformation	A matrix that contains the data acquired about the cars.
abstractionData	A matrix that contains locations (coordinates) of detected cars.
parkInformation	Contains information about the empty and occupied car spaces.
Methods	
drawAbstractMap()	It draws a visual sketch of the parking lot by using the abstractionData matrix.
reScaleMap()	It rescales the map according to the screen sizes.
getCarInfo()	Accessor function for getting the car information.
getParkInfo()	Accessor function for getting the park information.

Class	VehicleParkInfo
Attributes	
carCoordinates	It gives car location in the parking area.
carModel	Car model that acquired by prediction.
carColor	Car color that acquired by prediction.
isFilled	Boolean value to control occupation of parking area.
carID	This is unique id given for each car.
Methods	
createVehicleMatrix()	Create new vehicle information matrix for the parking area.
isPlaceFilled()	Control that is selected location is occupied.
occupyPlace()	In case of occupied fill the location.

Class	MainController
Attributes	
carInformation	Acquired information about car from the prediction.
date	Date of the predictions.
Methods	
fillCarInformationMatrix()	It updates car information taken with using makePredictions.
updateParkInformation()	Updates the vehicle park info matrix.
sendCarInformation()	Send acquired car information.
createAbstractions()	Create park abstraction.
trainModel()	Train model using the dataset.
makePrediction()	Make predictions with using the trained model.
streamVideo()	Stream the video for the predictions.

4 References

[1] Ieee reference guide. https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf. Accessed on 4-10-2020.

[2] Unified modeling language. https://en.wikipedia.org/wiki/UnifiedModelingLanguage, May 2011. Accessed on 4-10-2020.

[3] Cuda zone. https://developer.nvidia.com/cuda-zone, Sep 2020. Accessed on 5-10-2020.

[4] BritishParkingAssociation. Motorists spend nearly four days a year-looking for a parking space. www.britishparking.co.uk/News/motorists-spend-nearly-four-days-a-year-looking-for-aparking-space, Oct 2015. Accessed on 4-10-2020.

[5] CIRCONTROL. The parking of the future: problems, challenges and solutions. https://circontrol.com/the-parking-of-the-future-problems-challenges-and-solutions/, May 2020. Accessed on 4-10-2020.