Android Interface for Car Parking Systems

by

Kaung Htet Thar

A capstone project report submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering
Information and Communication Technology

Examination Committee: Dr. Matthew N. Dailey (Chairperson)
Dr. Mongkol Ekpanyapong (Co-chairperson)

Nationality: Myanmar

Asian Institute of Technology
School of Engineering and Technology
Thailand
May 2018

Acknowledgements

I would like to express my gratitude towards Dr. Matthew N. Dailey for his expert advice and encouragement throughout this project, as well as Dr. Mongkol Ekpanyapong for his extremely useful suggestions. I also would like to appreciate to the AIT Vision members for their kind and support. Moreover, I am deeply grateful to Mr. Ramesh Marikhu who help me to finish the integrating part with AIT VISION Lab's backend and Mr. Visutr Boonnateephisit for his ideas for my project, and for their times. Finally, I never forget to thank to all my friends who help and encourage me to complete my capstone project.

Abstract

Drivers of cars must always find parking spaces. Drivers often say “there’s not enough parking” in a neighborhood. Oftentimes, specific areas have limited parking at specific times. Parking problems often exist for institutions, downtown neighborhoods, business activities, and specific events. Asia, in many places, needs more parking spaces than exist, especially in Bangkok. There are many existing parking finder applications, and a great deal of research on traffic jam reduction has taken place. Parking lot applications usually include features such as pointing to parking lot locations, choosing parking spaces, storing details about a car’s location, notifying drivers about how long they have been parking, and choosing between map and satellite views. An application that can help drivers find parking spaces in real time will save driver time, decrease usage of fuel, and reduce traffic jams caused by drivers hunting for available parking spaces. The main point of this project is to provide information about available parking spaces via an Android device, using the AIT parking management system on the back end.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Drivers of cars must always find parking spaces. Drivers often say "there's not enough parking" in a neighborhood. Oftentimes, specific areas have limited parking at specific times. Parking problems often exist for institutions, downtown neighborhoods, business activities, and specific events. Asia, in many places, needs more parking spaces than exist, especially in Bangkok, where there are around 7 million cars and only 400,000 publicly accessible parking spaces (Thaitech, 2015). IBM Research (2011) found that 30% of traffic jams are caused by people hunting for parking spaces (Forbes, 2014).

There are many existing parking finder applications, and a great deal of research on traffic jam reduction has taken place. Parking lot applications usually include features such as pointing to parking lot locations, choosing parking spaces, storing details about a car's location, notifying drivers about how long they have been parking, and choosing between map and satellite views. An application that can help drivers find parking spaces in real time will save driver time, decrease usage of fuel, and reduce traffic jams caused by drivers hunting for available parking spaces.

1.2 Problem Statement

Worldwide, finding available parking spaces is a serious problem. There are applications available, but they only work with specific lots. Currently, there is much interest in automated parking management. In this project, I will build an Android application that integrates with a system for classification of parking space availabilities from cameras. Beyond basic photos and videos, the application will provide information about available parking spaces in real time. I will focus on developing an application that is efficient and useful for its users.

1.3 Objectives

The main objective of this project is to provide information about available parking spaces via an Android device, using the AIT parking management system on the back end. To achieve this main objective, I will pursue the following specific objectives.

1. Develop an Android application for parking lot information.

2. Integrate the front end with server-side analytic modules able to classify available car parking spaces based on cameras.
3. Display results of the analysis using video streaming to the Android application.
4. Evaluate the application and back-end system on a real-world parking lot at AIT.

1.4 Limitations and Scope

1. Only outdoor parking lots are currently supported by the AIT back end.
2. The video stream that can be viewed by the Android application will depend on the stream provided by the back end.
3. If the back end is extended to work with indoor parking lots, the Android application should be able to support parking information for indoor lots without much further modification.
4. The Android application will only use video analytic result data from the database and will not be involved in image processing.
5. The application will be developed on Android only.

1.5 Proposal Outline

Chapter 2 presents a literature review on Android application architecture, Android devices, REST for rich clients, video streaming to mobile applications, and parking lot management systems with real-time applications.

Chapter 3 presents the methodology I will use for the project with a system overview and system design.

Chapter 4 describes the results of my experiments to validate the methods.

Chapter 5 concludes the project and describes possible future work.

Chapter 2

Literature Review

2.1 Android Devices

As stated in Wikipedia (2016), Android is a mobile operating system developed by Google, designed for touchscreen mobile devices such as smartphones and tablets, and written in Java (UI), C (core), and C++. Mobile devices run on batteries and have low-power processors compared to desktop or laptop PCs. Android's user interface is mainly based on direct control using swiping, tapping and pinching with touch gestures, and manipulated on-screen objects with a virtual keyboard for text input.

2.2 Android Mobile Applications

According to Wikipedia (2016), a mobile application is a software application designed to run on mobile devices such as smartphones and tablet computers. Developing applications requires understanding the constraints and features of these devices.

Shedge (2013) introduces the main building blocks of mobile applications. These building blocks are components that application developers can use to build Android applications. It is important to understand Android components to put an application together. The main components are Activities, Intents, Services, Content Providers, and Broadcast Receivers.

1. ***Activities*** are the views a user sees on the device at one time as a single screen. Activities are the most visible part of an application.
2. ***Intents*** are messages that are sent to major building blocks or components.
3. ***Services*** are components running in the background that do not have any user interface components. Services run without any interface but can perform the same actions as activities.
4. ***Content Providers*** are interfaces for sharing data between applications.
5. ***BroadcastReceivers*** allow Android applications to broadcast events without knowing the identities of the receivers.

Moreover, as stated on the Android Developer website (2016), ***AsyncTask*** is used to enable synchronization for the User Interface (UI) thread. Asynctasks perform background operations and distribute results to the UI thread without employing threads or handlers.

2.3 REST for Rich Client

Fielding (2000) introduced Representational State Transfer (REST), an architectural style used for Web service development. There are six constraints and five HTTP methods that comprise a REST system. The Six goals or constraints are client-server, cacheable, layered system, stateless, code on demand, and uniform interface. The five main HTTP methods are GET, PUT, DELETE, OPTION, and POST.

HTTP was originally developed with the Web services in mind, and it has a rich palette of verbs, URIs, Internet media types, and request and response headers. JSON (JavaScript Object Notation) is commonly used for REST services. It is an easy-to-use format for building HTML clients running Javascript. It can be decoded easily by Web browsers with embedded JavaScript logic that collects REST resources for extremely responsive user interfaces.

We normally use a Web server to deliver HTML pages and JavaScript. JavaScript can connect RESTful services to get and parse JSON and update the HTML document.

2.4 Video Streaming to Mobile Applications

The VideoLAN organization (2016) provides the VLC media player, which can decode many video and image formats. libVLC provides the multimedia framework to the VLC media player. It allows developers to create multimedia applications using VLC features such as image transfer and video streaming.

As stated in the VideoLAN documentation:

> VLC media player which can be used as a server and as a client to stream and receive network streams. VLC is able to stream all that it can read. VLS (VideoLAN Server), which can stream MPEG-1, MPEG-2 and MPEG-4 files, DVDs, digital satellite channels, digital terrestrial television channels and live videos on the network in unicast or multicast. Most of the VLS functionality can now be found VLC. Usage of VLC instead of VLS is advised (VideoLAN organization, 2016).

2.5 Parking Lot Management Systems

Joshi, Khan, and Motiwalla (2012) present SFpark, which was developed to improve the parking systems in San Francisco. See Figure 2.1 for a screenshot. The system focuses on usefulness. It uses supply and demand levels to adjust meter prices.

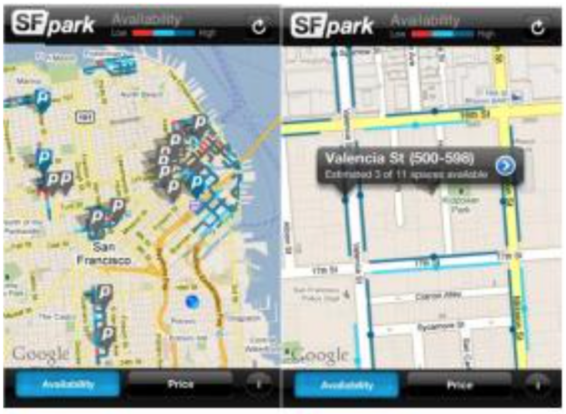


Figure 2.1: SFpark parking system. Reprinted from Joshi, Khan, and Motiwalla (2012).

Netca is a Web application for parking management in China. It was produced by Balizhuang company and developed to address the lack of local parking spaces. A Web application allows drivers to view a large number of parking spaces. People who drive into a town can use available spaces. This application connects drivers to the parking lot owner and provides digital maps that show available parking lot locations and allows search for a parking space. A sample screenshot is shown in Figure 2.2.

Figure 2.2: Netca Web application. Reprinted from Joshi, Khan, and Motiwalla (2012).

Al-Kharusi and Al-Bahadly (2009) present an outdoor parking system capable of transmitting video images from cameras. One or more cameras are used to view the lot. The system processes the images taken by the cameras. To transfer images to the computer, a wireless network is used. The video camera captures images, a transmitter sends the images to the receiver in the

control room, and the receiver sends signals to a computer. The computer processes the signals and sends them to the indoor transmitter. Video images and camera data are transmitted to a computer. A compact wireless video interface is used to transfer video from one location to another without the use of expensive and chaotic cabling.

To test different scenarios for car parking, the authors prepared car park positions with empty and filled cars to test the user interface with different numbers of cars parked in different positions, as shown in Figure 2.3.

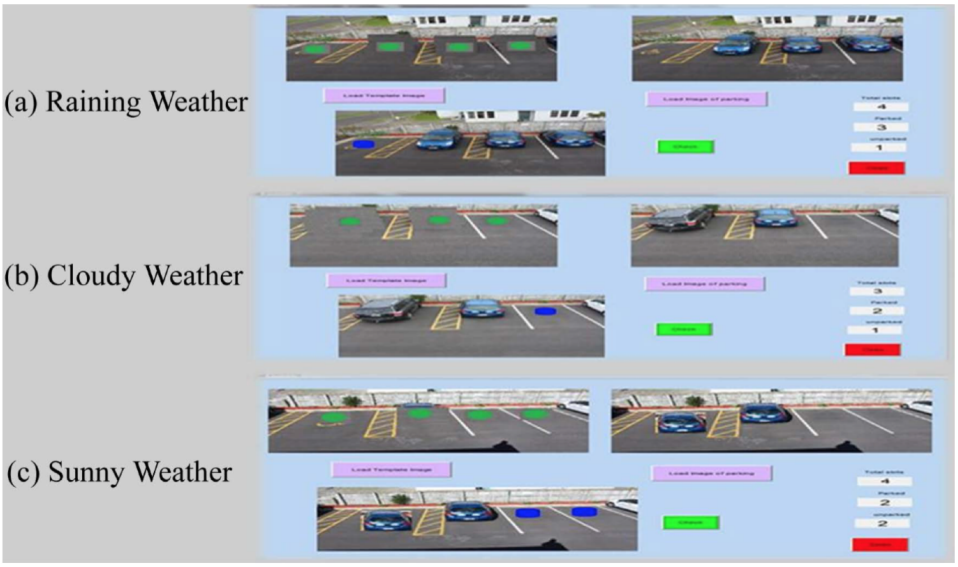


Figure 2.3: Car park display for different conditions. Reprinted from Al-Kharusi and Al-Bahadly (2009).

Noor and Kin (2009) introduce a parking system for buildings consisting of five main modules. They include shortest path calculation, image processing, updating status, I/O, and time keeping. An algorithm is used to support choosing parking destinations by assigning patrons to parking spaces. Additionally, image processing is used with CCTV images to get information from the captured images.The updating status module uses the shortest path module. The database update module is used to obtain new parking space availability status. The I/O module is used to get data from the driver about a selected parking location as a user interface command. For a reserved space, the time can be reloaded and extended.

The parking space system consists of three major subsystems. They are car parking user interface, the sensor node system, and the administrator management system. The major subsystem flow starts with the administrator management system. The administrator is required to log in using a specified username and password for security. After that, the system will provide many activities, and the administrator can choose to create a new map or edit an existing map, as shown in Figure 2.4.

Figure 2.4: Module to draw car parking layout. Reprinted from Noor and Kin (2009).

A sensor node system is used to determine the status of each space. A ZigBee module, a camera module, and microcontroller boards are involved in the sensor node. The system compares the pixels between the parking lot with the subsequent image and an existing background parking lot image. When the pixels of a new image are close to the background, the system shows an empty space. When the difference is high, the system shows an occupied space, as shown in Figure 2.5.

210	243	243	243	243	243	251	251
222	243	235	235	243	243	251	251
222	241	235	235	222	251	241	251
222	241	235	235	222	210	251	241
222	235	235	235	222	251	251	251
222	251	241	241	241	241	241	230
222	251	210	210	210	251	251	251
222	251	210	210	210	251	251	251

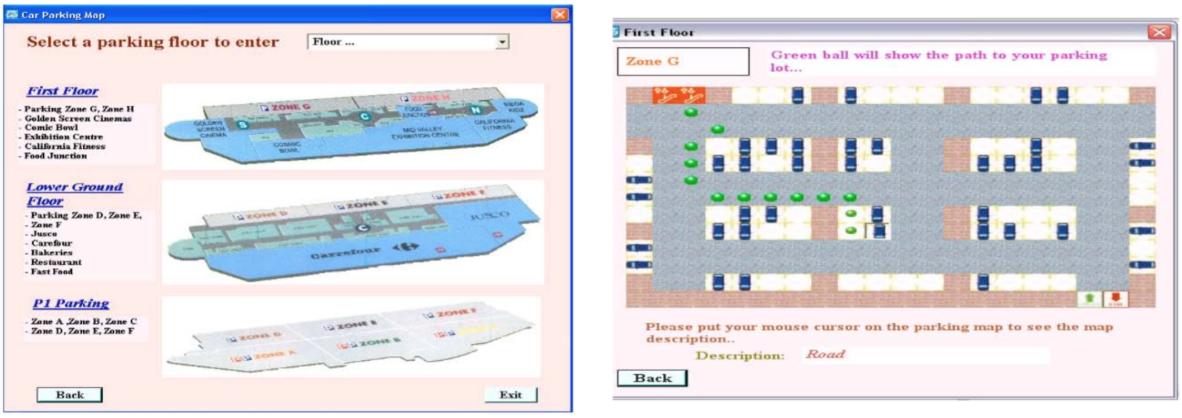
(a)

210	243	243	243	243	243	251	251
222	**58**	**58**	**58**	**58**	**58**	**63**	251
222	**58**	**58**	**58**	**58**	**58**	**63**	251
222	**63**	**56**	**58**	**60**	**56**	**63**	241
222	**63**	**56**	**63**	**60**	**56**	**63**	251
222	**65**	**56**	**60**	**60**	**60**	**55**	230
222	**65**	**56**	**65**	**65**	**65**	**55**	251
222	**65**	**56**	**65**	**65**	**65**	**55**	251

(b)

Figure 2.5: Image pixel occupancy rule. (a) Empty space. (b) Occupied space. Reprinted from Noor and Kin (2009).

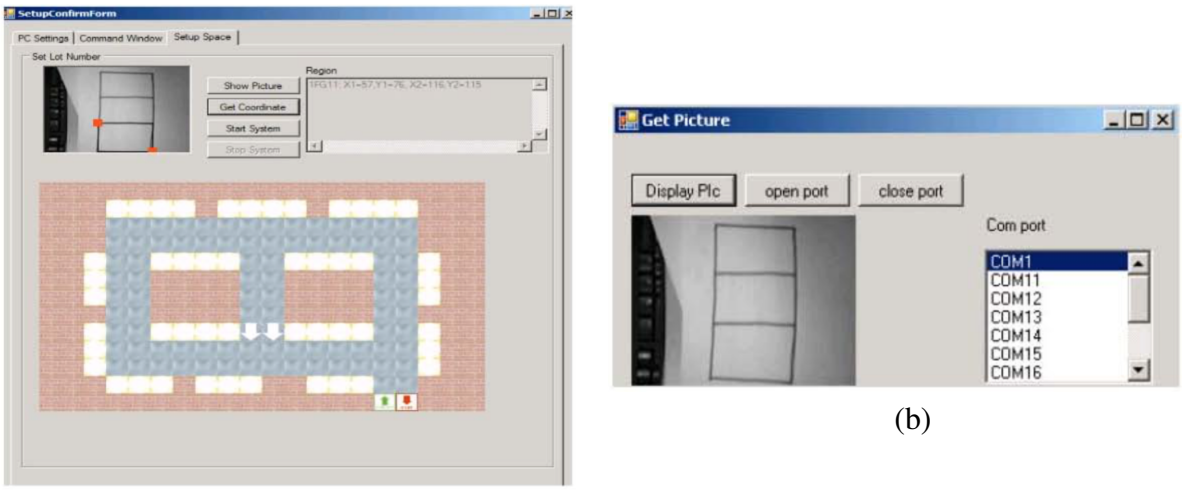
The user interface, as shown in Figure 2.6 shows a main menu, when patrons arrive at the entrance and check the ticket machine, allowing them to select a parking floor to enter. Then the screen will show the menu to search for spaces and patrons select an available parking space on the display monitor before taking a parking ticket.



(a) (b)

Figure 2.6: Parking system user UI menus. (a) Main menu. (b) Floor Selection. Reprinted from Noor and Kin (2009).

In the image processing module, each parking space is tracked using administration tools to obtain pixel coordinates, as shown in Figure 2.7 (a). The red dot is drawn to detect the pixel coordinates of the parking space and to give a space ID. Coordinates and IDs are stored in a RabbitCore microcontroller. When the camera module detects a vehicle entering each slot, the microcontroller sends the busy status to a central computer via ZigBee wireless communication. The information is collected by a serial reader that reads the serial data transmitted via the ZigBee communication port (see Figure 2.7 (b)).



(a) (b)

Figure 2.7: Image processing module. (a) Parking space coordinate region. (b) Serial reader program. Reprinted from Noor and Kin (2009).

Only three parking spaces with ID are available in the test scenario. When a user arrives at the parking entrance, the driver can press the View button and a layout is displayed on the monitor in Figure 2.8 (a). The system guides the user to the available parking space and reserves the

parking space for 10 minutes. The system does not update the database during this time. The reservation is not changed, but the next guest who arrives at the parking entrance receives a different parking space, as shown in Figure 2.8 (b).

If after 10 minutes the parking space is still empty, the sensor node sends a message to inform the management node of the current status and the database is updated with the state of the space parking shown in Figure 2.8 (c). Then, when a third driver arrives at the entrance and presses the show button, the driver is assigned to a different parking space, as shown in Figure 2.8 (d).

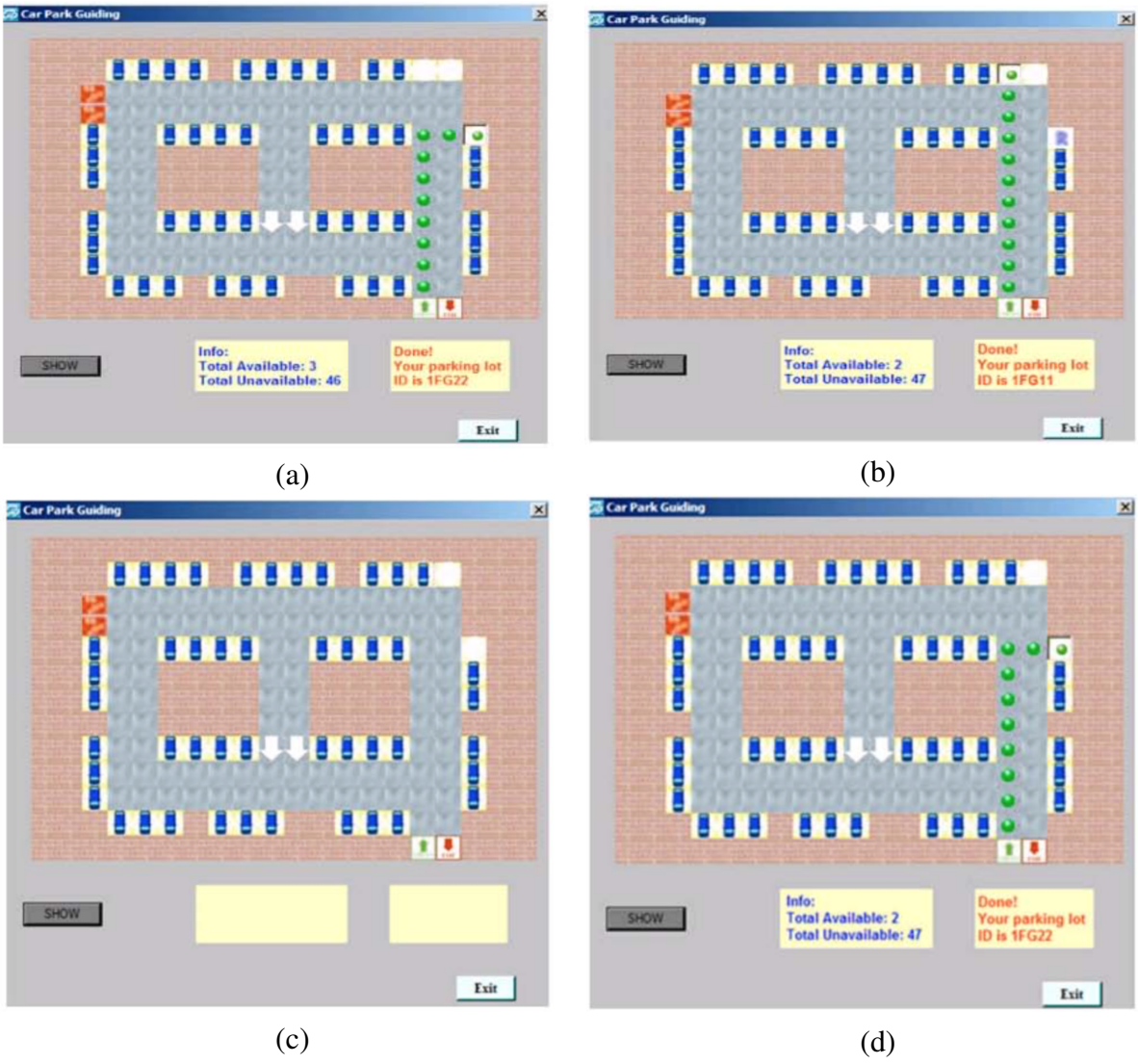


Figure 2.8: Three drivers subsequently arrive at the lot entrance. (a) First driver arrives at entrance. (b) Second driver arrives at entrance. (c) Reservation period expired. (d) Third driver arrives at entrance. Reprinted from Noor and Kin (2009).

Chapter 3

Methodology

3.1 System Overview

The flowchart in Figure 3.1 shows the overall architecture of my system. The camera analysis modules running on the compute server detect and classify available parking spaces. After detection and classification, the application streams video via a Web application. These steps have already been developed by the AIT VISION team. My Android application displays images to the client and provide information to the user with a REST architecture design for the Web application.

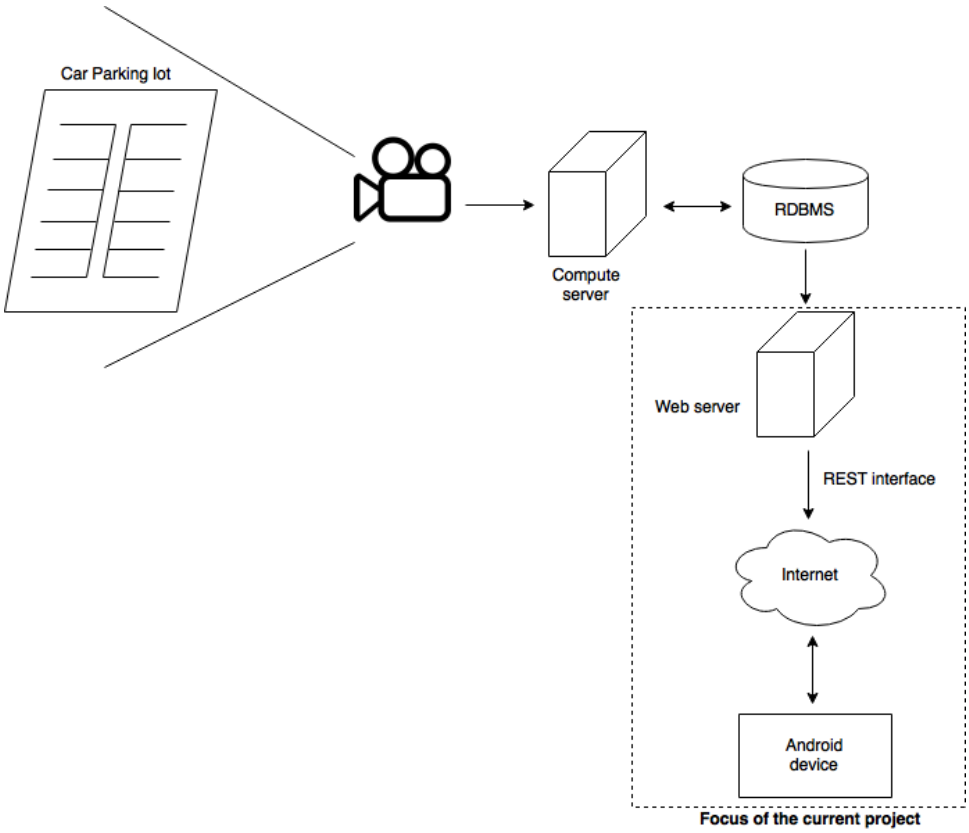


Figure 3.1: Overall system architecture.

My application requires the following features:

1. Display parking information for nearby parking lot based on location of the driver or location specified by a user.
2. Highlight nearest available parking lot given locations and occupancy of parking lots.
3. Display the above information on a map.
4. Display images of the selected lot with space availability image when the user wants to see them.
5. Send users to a different lot based on a Google map if the destination parking lot has become fully occupied.

3.2 Android Application

Android applications are written in the Java programming language. Here I describe some important facts about Android development.

3.2.1 Android Studio setup

Android Studio is an integrated development environment (IDE) for developing Android applications. It can view the interface we working on and the code for the components. The Android Manifest contains version codes and API specifications as provided in the **build.gradle** file. The Manifest declares feature and permission requirements. Moreover, it can be updated automatically when I create a new module in the application with the **setting.gradle** files, and it can be developed rapidly.

3.2.2 Android Phone

Media Codecs are not supported for all Android platform versions. They are only supported on Android version 3.0 and above. When I build aganist older Android versions, it can run on any Android device. Although application will run on any Android device, I test primarily on the Samsung Galaxy Tab 3. See Table 3.1 for the specifications of the device.

Devices	Models
Network	2G and 3G: GSM 850 / 900 / 1800 / 1900 - SM-T211, SM-T215
CPU	Dual Core Application Processor, 1,2 GHz
OS	Android OS, v4.1.2 (Jelly Bean)
Display	TFT, 1.024 x 600
Resolution	3-megapixal
Camera	Primary: 3.15-megapixal. Secondary: 1.3-megapixal
Video	Codec: MPEG 4, H.264, H.263, WMV, DivX Playback: 1080p@30fps. Recording: 720p@30fps
Connectivity	Wi-Fi 802.11 a/b/g/n, dual-band, Wi-Fi Direct, DLNA, hotspot
Memory	Internal: 8/16 GB, 1 GB RAM
Audio	MP3, WAV
Battery	Non-removable Li-Ion 4000 mAh battery

Table 3.1: Technology specifications of the Samsung Galaxy Tab 3. Reprinted from GSMARENA (2016).

3.2.3 Design

Six components are used to build the application in Android Studio.

3.2.3.1 Activity Lifecycle

An application typically has multiple activities, and the user moves back and forth among them. The activity lifecycle flow is the main part of the Android API and is important for managing activities. Hardware functions within Android devices connect with this framework.

3.2.3.2 Intents

Intents perform runtime binding in different applications. They are used to launch activities and contain a description of an action that holds a passive data structure. Intents are used with ***startActivity*** to launch an Activity, ***broadcastIntent*** to send a message to components of BroadcastReceiver, and ***startService(Intent) or bindService(Intent, ServiceConnection, int)*** to communicate with a Service running in the background.

3.2.3.3 BroadcastReceivers

Broadcast Receivers receive messages broadcast by other applications or by the system itself. They allow applications to receive intentions sent by the system or other applications. Creating a receiver in code and registering it requires the ***Context.registerReceiver()*** method.

3.2.3.4 Services

A service is a component that runs long non-UI operations in the background. It can perform interprocess communication (IPC) like network transactions, play music, perform file I/O or interact with providers in the background. It requires ***startService()*** and ***bindService()*** methods in the application to perform long term operations in the background. One example is Started. Started is a service started when an activity starts it by called startService(). It can run in the background after the component that started is destroyed.

3.2.3.5 Content Providers

Content providers manage sharing of a structured set of data. Applications use content providers to share data. I build content providers in my application to mange data such as audio, video, images, and personal contact information via the ***com.android.provider*** package.

Moreover, a Content Resolver provides access to a content provider. It is used as the name implies; it accepts requests from clients and resolves these requests by linking them to the content provider with authority. I will develop a content provider to store a mapping from authorities to Content Providers using the ***com.android.content*** package.

In Figure 3.2, a service receives the intent data from the starting Android component. The service updates a content provider, and the activity is notified by the content provider. This works for services running in their own process. To enable communication, an activity can register a broadcast receiver for an event and the service sends events.

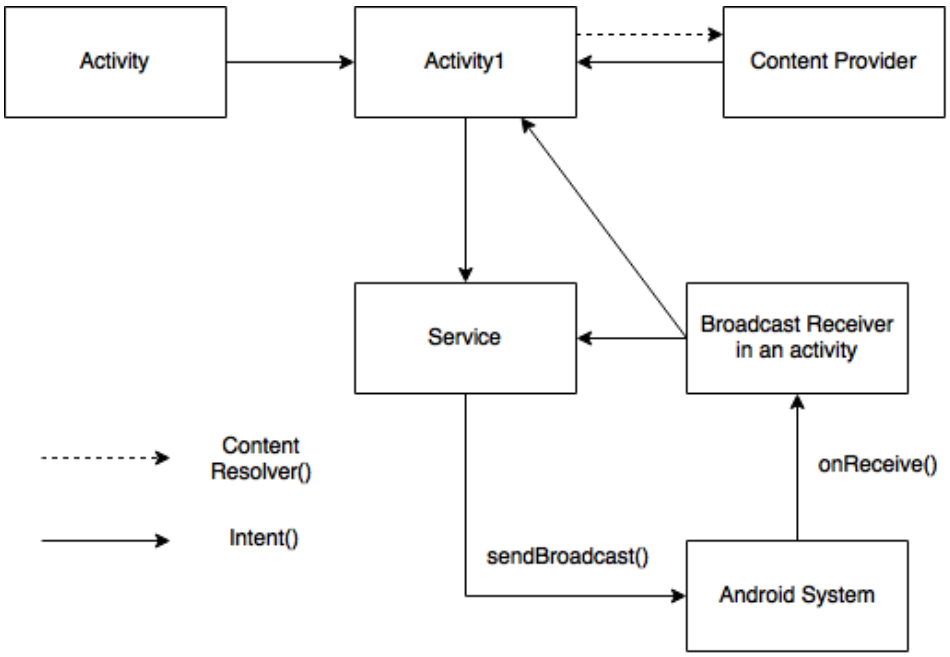


Figure 3.2: Main data flow among application components.

3.2.3.6 AsyncTask

Network operations on a separate thread from the user interface prevent the network delays. This class provides to fire off a new task from the User Interface thread and it allows application to perform background operations. It will be used to build application to keep threads running for long time using the **java.util.concurrent** package.

3.2.3.7 Google Maps API

The Google Maps API allows developers to integrate Google Maps into applications on the Wab, Java-based phones, and other mobile devices. Applications using this API allow users to control the map with touch gestures. I can build this API into my application with the key class ***MapView*** to get data from Google Map Services. I can build locations and map services into my application using the classes of the **com.google.android.gms.maps** package.

Before I run an application with a Google Maps Activity, I need to create a Google Maps API key. According to Figures 3.3 and 3.4, the key can be created to use in an application at the website **console.developers.google.com** in two steps:

1. Creating Google Maps API with my Google Account

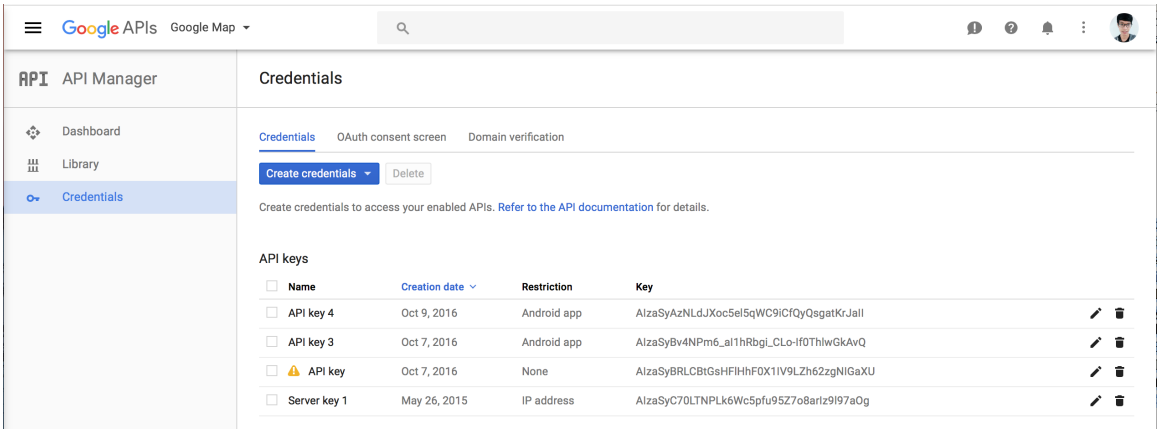


Figure 3.3: Getting Google Maps API from Google.

2. Adding a Google Maps API key in ***app/res/values/google_maps_api.xml***.

Figure 3.4: Adding Google Maps API in the application.

After that the Google Maps Activity can be included in the application using Google map services as shown in Figure 3.5.

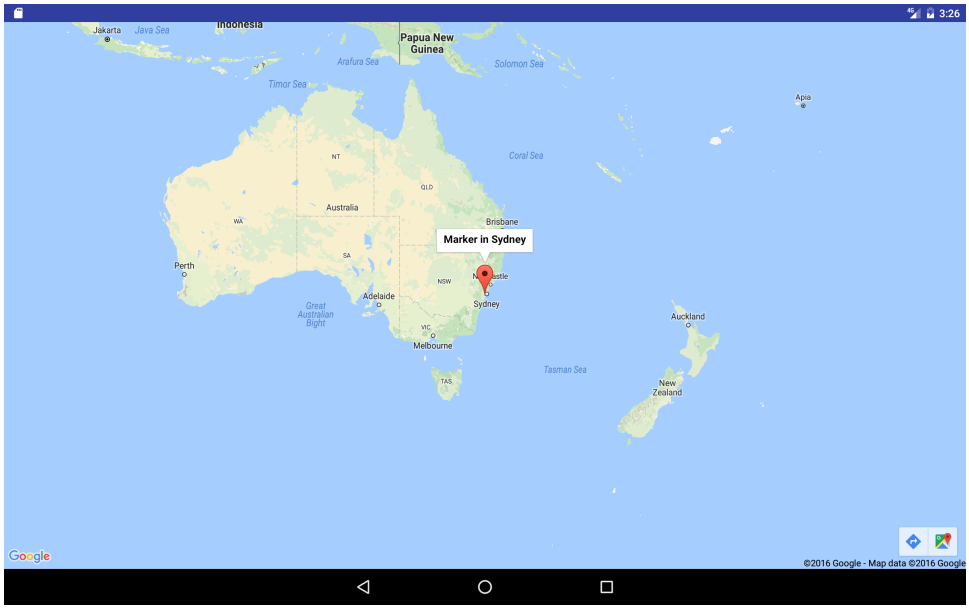


Figure 3.5: Google Map Application run on emulator.

3.2.3.8 Mockup of User Interface

The application provides two options, namely finding and displaying every nearby parking lot on a Google map and going directly to the nearest parking lot with empty spaces, as shown in Figure 3.7.

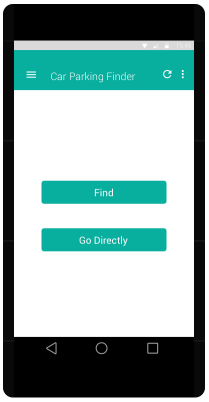


Figure 3.6: Main Menu.

After clicking the Find button, the application shows parking lots, and the user can select the specific parking lots on the Google map, as shown in Figure 3.8.

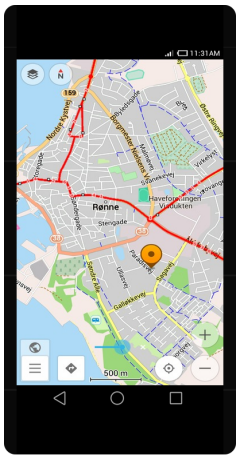


Figure 3.7: Google Map.

After selecting the parking lots, as shown Figure 3.9, the application provides parking information and two options. The two options are to display image of the parking lot and to navigate to the specific parking lot.

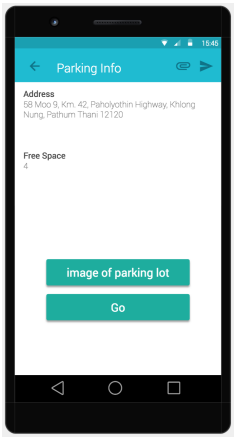


Figure 3.8: Car parking lot Information.

On clicking the show an image of parking lot in Figure 3.10, the parking lot's image was displayed.

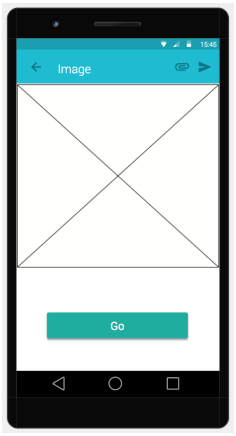


Figure 3.9: Show an image.

On clicking the Go button from the parking lot displays view or the Go button in the main menu view, the application provides navigation for the driver, and a real-time display of parking availability is displayed, as shown in Figure 3.11.

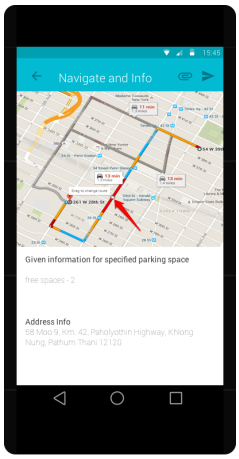


Figure 3.10: Navigation.

3.3 Integration of Front End with Server-side Analytic Modules

I create a sample database for integration and testing of the front end with server-side analytic modules. After creating the database, I pull the RESTful API to overlay information on the Google Map about availability of parking space in the lots.

3.3.1 Creating Local Database for REST API

The connectivity of the VGL database and the application is the main part of my project. But the application needs to be tested first, so I created a local mysql database. In the database, I created tables to store information about parking lots.

The org.apache.http.client package is used to define parking information in the application in JSON format. In PHP, JSON data is created as output data. PHP allows generation of various forms of parking lot information from mySQL database. The data for parking lots are generated in PHP in JSON format using json_encode method on an array.

For the markers of parking lots in the google map, the application needs to know the latitude and longitude of the parking lots. Therefore, the JSON data for the whole table is pulled from the the database. After parsing the data, the GoogleMap.addmarker method is used to add the markers to the map.

To display the list of parking lots in the RecyclerView, JSON data is parsed again. This time, it need to parse the data of the specific parking lot for the specific number of available parking spaces. The JSON API assumes the parking id and JSON parsed the information of the parking, as shown in Table 3.2.

	Parkings
primary	id
	name
	address
	zipcode
	totalslots
	freespace
	lat
	long
	url

Table 3.2: Table in sample database.

3.4 Displaying Image and Information Activity at Client

My application needs to integrate information from the camera and video analytics as an image (jpg file) and database provided by the AIT VISION team. For data on parking spaces, my application needs to know the parking lot's id, name, latitude, longitude, image URL, and number of free spaces and total number of spaces. I use a database created by the AIT VISION team. Databases can be defined as the Client database and VGL database. Client database can provide id, name, latitude, longitude and url of image. VGL database can provide information about free spaces and total number of parking spaces.

For the data of empty spaces and total number spaces, JSON data is parsed the EmptyParkingTotal.totalspace and EmptyParkingSlot.emptyspaces from the VGL database. These data is used update to the client database with PHP, as shown in Figure 3.12.

The parking lot images from the camera are posted to a jpg file available from the Web server. See Figure 3.13 for a sample image. The URL is available from the client database. When the

application gets a source from web server, it puts the result in the output of the Web service for the client through the client database. This prevents the user can not interfering with the VGL database.

When users want to view available parking space image and information, the image and data from the JSON array that are relevant to the parking lot's location is chosen to set into the relevant view.

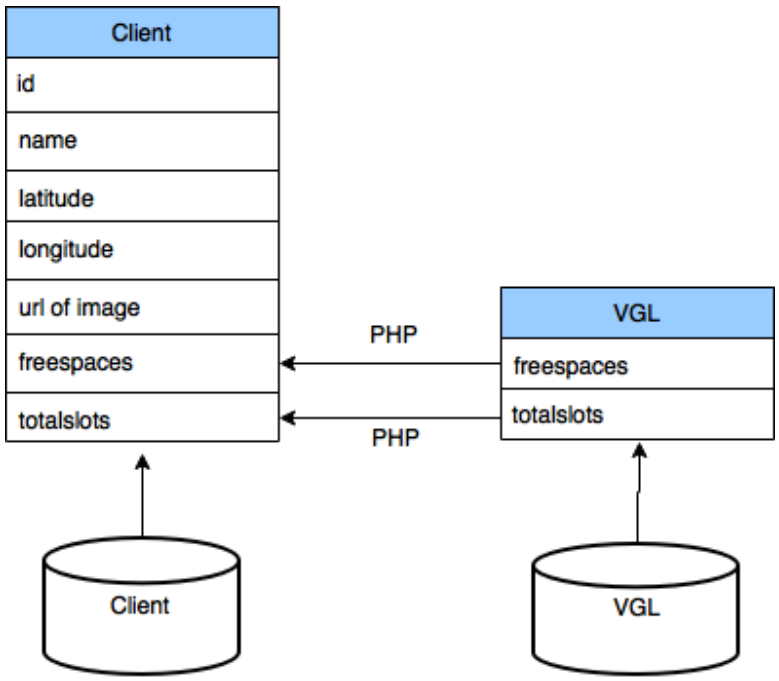


Figure 3.11: Integration of the Client and VGL database.

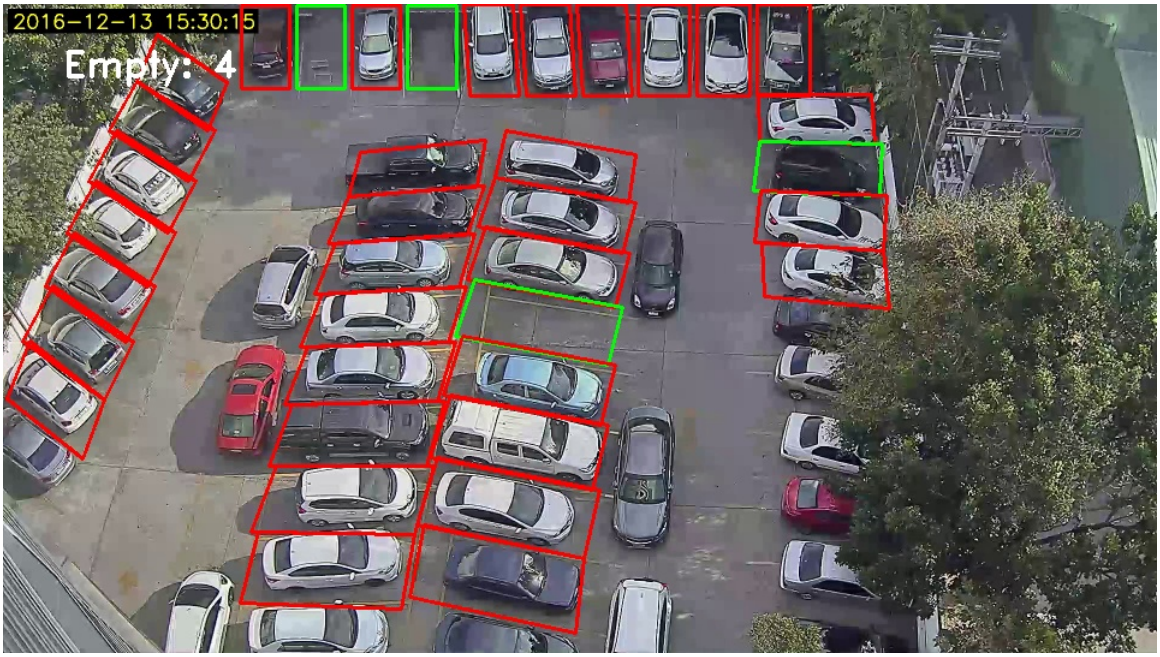


Figure 3.12: Sample image is provided by AIT VISION team.

Iteration plan

1. Get Map from server and display in Android device. - DONE
2. Get user's current location with Longitude and Latitude - DONE
3. Get duration and distance to navigate to the car parking using Google Map service polyline - DONE
4. Create mysql database to get the parking information - DONE
5. Put the data to the application based on Google Map API - DONE
6. Run the video streaming of url from each one of parking spaces - DONE
7. Draw the polylines to parking lots from current location - DONE
8. Calculate the distance and duration to the parking lots - DONE
9. Display the video stream for each parking lots - DONE
10. Draw the polyline for each parking locations - DONE
11. Show the parking informations with Recycler.View - DONE
12. Send drivers to nearest space when specific parking space is full - DONE
13. Put the localhost database and front end system to the internet (Hosting) - DONE
14. Integrate with exists video analytic system. (Video setup and Server setup) - DONE

15. Make the buttons more readable - DONE
16. Reload image and video in streaming video activity every 10 seconds - DONE
17. Put Webservices from application to CSIM Webserver - DONE
18. Reload every 10 seconds for the image processing from camera view of CSIM Webserver - DONE
19. Show the updating information of parking lot from database - DONE

3.5 Evaluation plan

To evaluate the application and back-end system from AIT VISION, I evaluate the method against a video from the parking lot of Chayakarn Company Limited in Bangkok. For image transfer, I test with Android devices that have Android version 3.0 or above. To test image quality and performance, I evaluate the methods according to performance and error handling.

Chapter 4

Experimental Results

In this chapter, I describe an Android interface to a car parking system using screenshots collected from a the Samsung Galaxy Tab 3. I describe the results of my experiments to evaluate the proposed system design. I used Android Studio to develop the application. The results can be seen in the following sections.

4.1 Application Features and Activities

The application provides two options, as shown in Figure 4.1, which are finding and displaying nearby parking lots on a Google map and checking a specific parking lot.

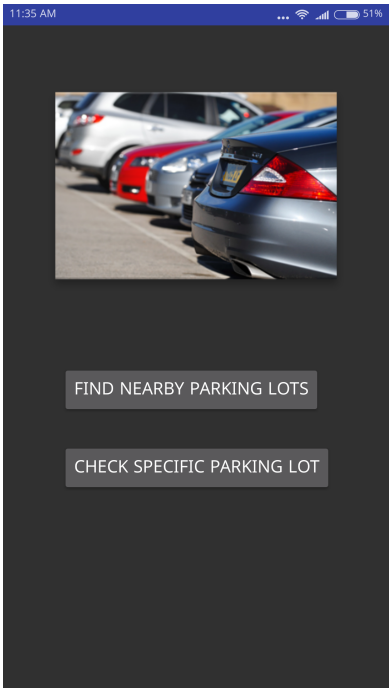


Figure 4.1: Main page.

After clicking the Find Nearby Parking Lots button, the application shows parking lots nearby, and the user can the specific parking lots by tapping the marker on the google map (Figure 4.2).

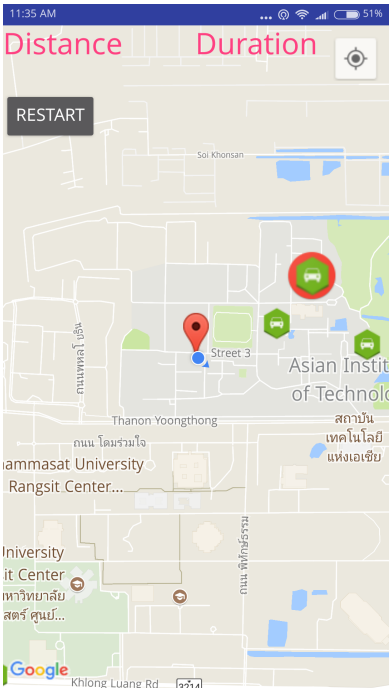


Figure 4.2: Displaying parking lots and current location.

After selecting a parking space, the application provides parking information and two options Camera View and Go. Camera View means displaying image and Information of free parking space ,and Go means to start navigation to the specific parking lot (Figure 4.3).

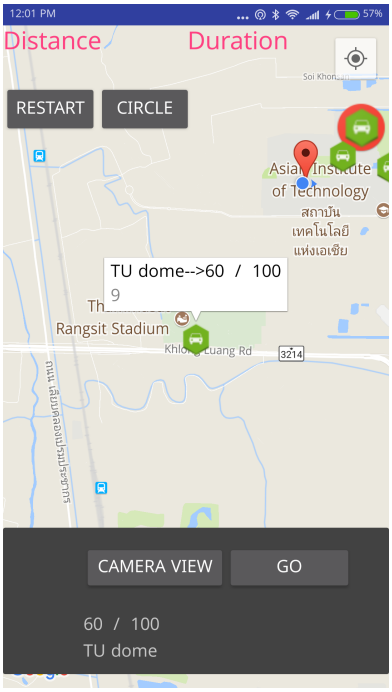


Figure 4.3: Providing parking information and options to show a camera view of the parking lot and to go to the selected lot.

On clicking the Camera View button, the real-time parking space's information and camera view is displayed. The layout has two images from the VLC streaming player in video view and loading directly from an image URL in the Image view.

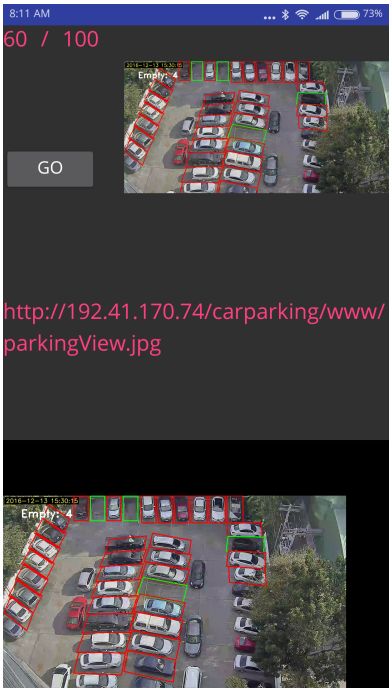


Figure 4.4: Information of real-time parking space.

On clicking the Go button, the application provides a navigation for the driver, and the information of real-time free parking spaces was displayed.

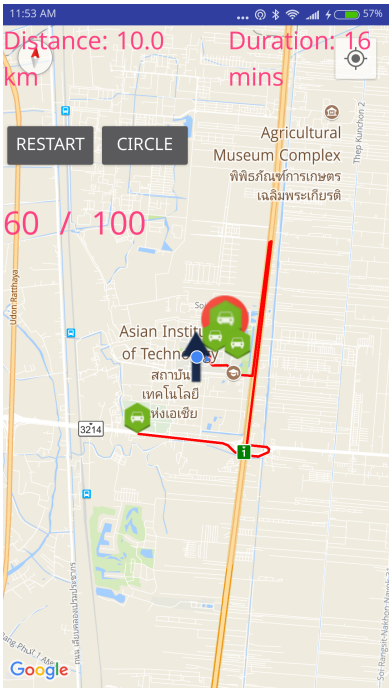


Figure 4.5: Displaying polylines, duration and distance to the destination lot.

Moreover, If a specific parking lots becomes full, a dialog box pops up to go to a nearby parking space and cancel the dialog box.

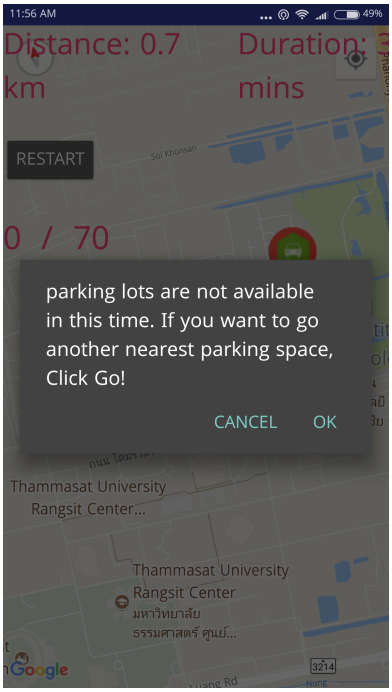


Figure 4.6: Displaying notification of no empty spaces and reset to nearest parking lot.

On clicking the Check Specific Parking Lot button, the application displays a search text box.

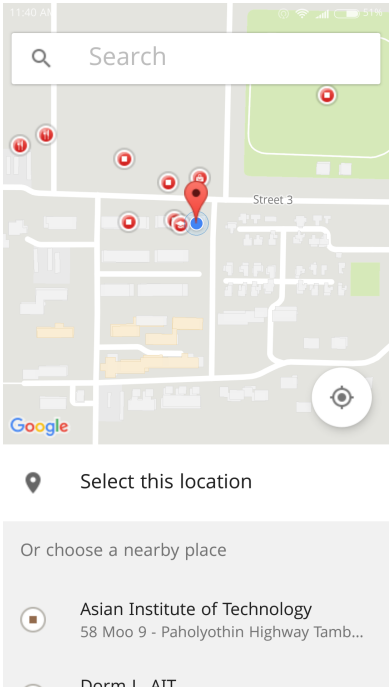


Figure 4.7: Typing the specific location area.

After selecting the specific location, the application shows the specific location area and its status.

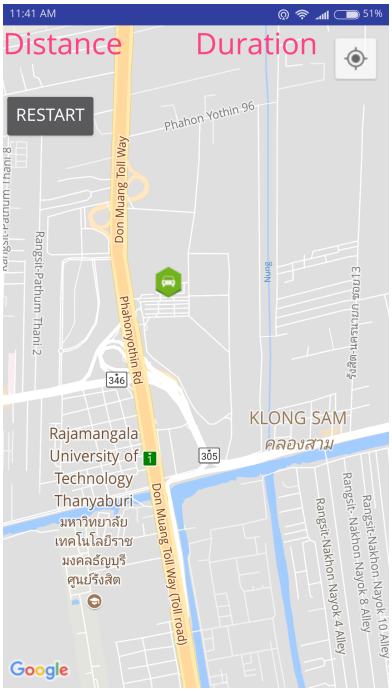


Figure 4.8: Specific location area.

After selecting the specific location, the application shows two circles. They are red circle (6km) and blue circle (1km).

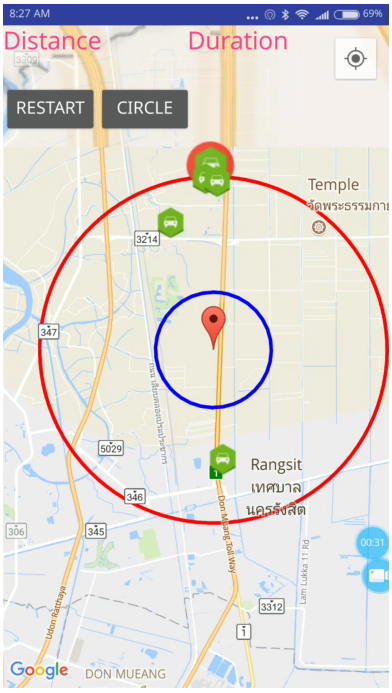


Figure 4.9: Two circles with 6km and 1km.

4.2 Testing Variety of Transferring Image Method

Application used to display the image with two methods that are using input stream method and VLC media player method.

Firstly, the url of the available image address from the web page is used. So, the url from *http://192.41.170.74/carparking/www/parkingView.jpg* is used for the image address. The image is already created by the AIT VISION team. When Activity run, the application is started to use in media player using java.net.URL() method as input stream. Decoding the image from the stream use the BitmapFactory.decodeStream() method. After decoding the image, the image is set in the image view and display on the screen every 10 seconds. The method is directly used the image from Web services so the image is displayed on the screen every 10 seconds is perfectly working. When the address is for the mp4 format, the image will be disappeared from the FrameLayout.

To test the image view, I set an image URL address in the database for the VLC media player method. If the surface view is used, the video is set to play the video using the vlcVout.setMedia(). When the surface view is not used, and The data are in JPG format, the module sets the Texture view to display using the same method. So, the image and video can be displayed on one FrameLayout. For replaying every 10 seconds, the thread needs to use and the setMedia method can not be destroyed with Backpress or the application crashes.

4.2.1 Performance Testing

I test the average run time of Samsung Galaxy Tab 3 and Redmi Note 3, as shown in Table 4.1. I mainly focus on testing and running my application on the Samsung Galaxy Tab 3. But the application completely supports both devices. But Samsung Tab 3 is slightly slower than the Redmi Note 3 for getting information from background UI Thread. For example, information from the RESTful API and image transfer occurs every 10 seconds. My application could definitely support Android version 4 and above, and it is more faster with 3 GB RAM.

Mobile Devices	Android version	CPU	Memory	Storage
Samsung Tab 3	4.4.2	2 GHz	1 GB - RAM	16 GB
Redmi Note 3	6.0.1	1.8 GHz	3 GB - RAM	32 GB

Table 4.1: Average run time resources used on two different devices.

4.2.2 Error Handling

The Wifi and cellular data connections are the main components needed to run the application. To handle problems for the driver, the application displays no cellular data in the dialog box, as shown in Figure 4.9.

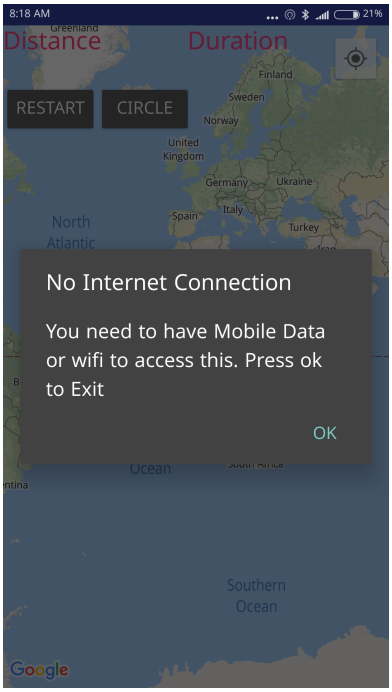


Figure 4.10: Dialog box with no cellular data status.

Chapter 5

Conclusion and Future Work

To summarize, my integration of a front end with server-side analytic modules able to classify available car parking spaces based on cameras with Android devices is working. The Samsung Galaxy Tab 3 is relatively slow to compared to the Redmi Note 3. The camera for the real-time parking space monitoring is not currently setup as the AIT VISION project is still ongoing and will be deployed in 2018. But I have successfully tested the display of the images based on video captured by the AIT VISION team using videos of Chayakarn Co.'s parking lot in Bangkok. The application provides information every 5 seconds, and the image transfer reloads every 10 seconds. The application also provides distance and duration information based on driving mode from the Google Maps API.

The marker of a parking lot needs to be removed from the Google map before updating the status to display current information. If WiFi is disconnected or the JSON date can not be parsed on return from the Web service, the marker disappears from the map. In future work, the application would include backup of parking information to the device's memory and performing a more comprehension evaluation with multiple Android devices.

References

Al-Kharusi, H., & Al-Bahadly, I. (2014). Intelligent parking management system based on image processing. *World Journal of Engineering and Technology*, *2014*.

Android-Developer. (2016). Asynctask. (System available at `https://developer.android.com/reference/android/os/AsyncTask.html`)

Fecheyr-Lippens, A. (2010). A review of http live streaming. *Internet Citation*, 1–37.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. Unpublished doctoral dissertation, University of California, Irvine.

Joshi, P., Khan, M. R., & Motiwalla, L. (2015). Global review of parking management systems & strategies. *Department of Operations and Information Systems, University of Massachusetts Lowell-Robert J. Manning School of Business*. (System available at `http://www.transport-research.info/sites/default/files/project/documents/20150807_141918_57605_p111115001.pdf`)

Noor, N., & Kin, L. (2009). Smart parking system using image processing techniques in wireless sensor network environment. *Inform. Technol. J*, *8*, 114–127.

Organization, V. (2016). Streaming. (System available at `http://www.videolan.org/vlc/streaming.html`)

Shedge, K., Pathak, M. S., & Rokade, S. (2013). Android-broadcast receiver. (`Systemavailableathttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.9805&rep=rep1&type=pdf`)

Team, G. (2016). Technology specifications. (`Systemavailableathttp://www.gsmarena.com/samsung_galaxy_tab_3_7_0-5422.php`)

Vishal K. Kumkar, G. N. M. (2015). Mobile web server on android platform with live video streaming. *International Journal of Computer Science and Mobile Computing*.

Wikipedia. (2016). Android (operating system). (System available at `https://en.wikipedia.org/wiki/Android_(operating_system)`)

Writer, S. (2015). Thaitech. (System available at `http://tech.thaivisa.com/parking-duck-parking-space-bangkok/2730/`)