GALATASARAY UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

SCREEN POSITIONING ALGORITHM FOR AUGMENTED REALITY APPLICATION ON ANDROID DEVICES

Doğa ERIŞIK

Jan 2016

SCREEN POSITIONING ALGORITHM FOR AUGMENTED REALITY APPLICATION ON ANDROID DEVICES

(ANDROID ARTTIRILMIŞ GERÇEKLİK UYGULAMALARI İÇİN EKRAN KONUMLANDIRMA ALGORİTMASI)

by

Doğa ERIŞIK, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

in the

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

Feb 2016

This is to certify that the thesis entitled

SCREEN POSITIONING ALGORITHM FOR AUGMENTED REALITY APPLICATION ON ANDROID DEVICES

prepared by **Doğa Erışık** in partial fulfillment of the requirements for the degree of **Master in Computer Engineering** at the **Galatasaray University** is approved by the

Examining Committee:

Yrd. Doç. Dr. Gülfem Işıklar ALPTEKİN (Supervisor)
Department of Computer Engineering
Galatasaray University ----------

Yrd. Doç. Dr. Teoman NASKALİ
Department of Computer Engineering
Galatasaray University ----------

Yrd. Doç. Dr. Ayşe TOSUN
Department of Computer Engineering
İstanbul Technical University ----------

Date: ----------

ACKNOWLEDGEMENT

I would like to thank to Ahmet Karaman for his contributions. I would also like to thank my supervisor Gülfem Işıklar Alptekin, and Özlem Durmaz İncel for giving me the opportunity to work such an interesting subject and learning interesting approaches that help me a lot. I am also grateful for their understanding of my time limitations and encourage they have provided. I also would like to thank to Okan Burçak and Mustafa İşbilen who are working for Yapı Kredi Bank and all my colleagues for understanding my dedication to the thesis. This work is supported within the scope of SAN-TEZ program of the Turkish Ministry of Science, Industry and Technology under the grant number 0307.STZ.2013-2.

Jan 2016
Doğa ERIŞIK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The ‘Augmented Reality’ technology is a combination of four different peripherals. These peripherals consist of camera, computer infrastructure, markers and the real world. Augmented reality can be summarized as these four units being positioned in the real world with three dimensions. The ultimate goal of augmented reality is to create a convenient and natural immersion. Development of mobile technologies has contributed significantly to the improvement of this domain. Using it, humans can interact with objects which surround them via their mobile devices.

In this thesis, we've developed an augmented reality application which runs on mobile devices with Android operating system. This application is developed with the corporation of Yapı Kredi Bank with the financial support of Ministry of Industry and Sciences. The purpose of this application is to correctly place bank merchants (will be referred as point of interests (POI)) and offices on device’s screen by calculating their positions relatively to user’s current position.

A new screen positioning algorithm that optimizes application’s performance and minimizes its energy consumption is proposed for this application. Furthermore, additional features and services (user choice filters, map support, QR Code screen) that use different sensors have been developed in the most energy efficient way. To accomplish energy efficiency we added different GPS fetching modes and libraries. These modes are particular to our application. Also we compare our algorithm to another algorithm and we saw that our application has a better accuracy on screen positioning and direction finding.

RESUME

La technologie « Réalité Augmentée » est une combinaison de quatre périphériques différents. Ces périphériques sont composé de; premièrement caméra, deuxièmement l'infrastructure informatique, troisièmement les marqueurs et enfin du monde réel. Donc réalité augmentée peut être résumée comme quatre unités étant positionnés dans le monde réel à trois dimensions.

Le but ultime de la réalité augmentée est de créer une pratique et naturelle immersion. Développement des technologies mobiles a contribué de manière significative à l'amélioration de ce domaine. Par conséquent les humains peuvent interagir avec les objets qui les entourent utilisant leurs appareils mobiles.

Dans notre projet, nous avons créé une application de réalité augmentée qui marchera sur les appareils dont le système d'exploitation est Android. Cette application est développée pour Yapi Kredi Bankası avec le soutien du Ministère Turc de Science, Industrie et Technologie. Le but de cette application est de placer correctement les marchands et les bureaux de Yapi Kredi sur l'écran de l'appareil, en calculant leurs positions relatives à la position de l'utilisateur.

Pour cette application, nous avons étudié, principalement, l'algorithme de positionnement à l'écran de l'application, l'optimisation de la performance et la minimisation de la consommation d'énergie. Des fonctions supplémentaires sont été ajoutées telles que les filtres de choix de l'utilisateur, l'écran de carte du monde où on peut voir tous les points d'intérêts, l'écran pour lire des code QR et l'écran de connexion. Pour diminuer la consommation d'énergie, on a implémenté trois différentes modes, et deux différentes bibliothèques pour acquisition de GPS. On a

ainsi comparé notre application avec un autre et on a prouvé que le nôtre a des résultats mieux pour positionner les marchandises et choisir leurs directions.

ÖZET

Artırılmış gerçeklik teknolojisi dört farklı çevre biriminin birleşimidir. Bu çevre birimleri kamera, bilgisayar alt yapısı, işaretleyici ve gerçek dünyadan oluşmaktadır. Artırılmış gerçeklik bu farklı dört birimin üç boyutlu olarak gerçek dünyada konumlandırılması olarak tanımlanabilir.

Artırılmış gerçekliğin hedefi, gerçek bir yere ya da nesneye bilgi ve anlam katmaktır. Mobil cihaz kullanımı günümüzde çok artmış ve bu artış ile birlikte artırılmış gerçeklik uygulamaları daha da önemli bir hale gelmiştir. İnsanların üzerlerinde taşıdıkları bu cihazlar ile etraflarında gördükleri cisimler ile etkileşime geçebilir hale gelmişlerdir.

Tez için geliştirilen uygulama ve algoritma Bilim ve Sanayi Bakanlığı'nın finansal desteği ve Yapı Kredi Bankası'nın teknik ve veri desteği ile birlikte Android işletim sistemli mobil cihazlar üzerinde çalışacak şekilde geliştirilmiştir. Tasarlanan uygulamanın amacı, Yapı Kredi Bankası'na ait üye iş yerleri ve banka şubelerini mobil cihazın ekranında, kullanıcıya göre konumlarını hesaplayarak doğru bir şekilde yerleştirmektir.

Bu tezde, uygulamanın temelini oluşturan ekrana yerleştirme algoritması üzerine odaklanılmıştır. Algoritmanın akademik yazında önerilen diğer algoritmalara kıyasla daha basit, daha az pil harcayan ve daha hızlı çalışan bir algoritma olması hedeflenmiştir. Bunun dışında filtre seçenekleri, harita desteği, QR Kod okuma ekranı gibi diğer özellik ve servislerin de, mobil cihazın pil ömrü ve performansını en arttıracak şekilde çalışması sağlanmıştır. Bu enerji kazancını üç farklı GPS edinme yolu kullanarak ve iki farklı kütüphane kullanılarak sağlanmıştır. Aynı zamanda önerilen algoritma akademik yazında bulunan farklı bir algoritma ile karşılaştırılmış ve

sonuçların çoğu durumda daha iyi olduğu gösterilmiştir. Önerilen algoritma ara yönleri tayin etme ve ekrana konumlandırmada doğruya daha yakın sonuçlar vermiştir.

1. INTRODUCTION

In this thesis, an augmented reality application for Android-based mobile devices is developed in cooperation with Yapi Kredi Bank. The application lists real merchants of the bank. Using the GPS coordinates, they are positioned on device's screen. We create a perspective projection to figure out where the objects should appear on the device's screen, so we can augment them. In particular, the focus of this thesis is to develop an efficient screen positioning algorithm. In this chapter, an introduction to developing augmented reality applications on Android will be made. Then our mobile application and the challenges we met and the contributions we made in this thesis will be explained.

1.1.AUGMENTED REALITY ON ANDROID

Augmented reality refers to a wide spectrum of technologies that project computer generated materials, such as text, images and video, onto users' perceptions of the real world (Yuen et al., 2011). It is related to a more general concept called mediated reality, in which a view of reality is modified (possibly even diminished rather than augmented), by a computer (Wikipedia, 2015). As a result, the technology functions by enhancing one's current perception of reality. The development of mobile technologies has contributed significantly to the improvement of this domain. Objects can be seen from different angles or a similar product can be used to make observations on the subject.

We can benefit from the following features of a mobile device:

- Location-based mobile data services
- GPS
- Compass
- Gyroscope
- Thermometer
- Accelerometer
- GeoMegnetic Field
- Camera

1.2. SENSOR-BASED AUGMENTED REALITY APPLICATION SOFTWARE (SARAS)

In our research, given points of interest (POIs) are aimed to be positioned on device's screen. These POIs refer to bank merchants and they are listed as a screen projection of actual objects. Figure 1.1 illustrates a representation of this screen projection.

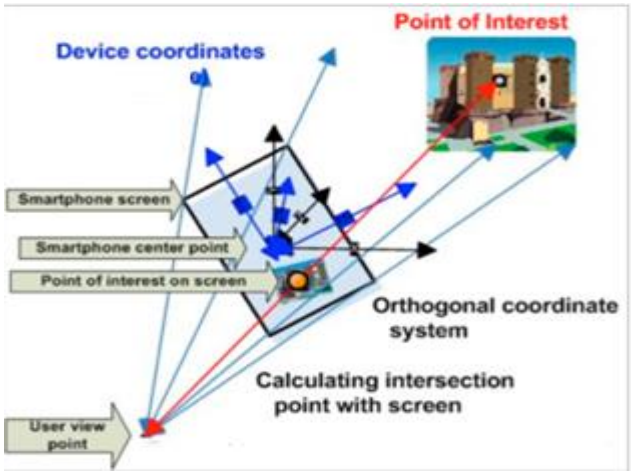

Figure 1.1 Screen Projection

Doing so, the following steps are followed:

1. To find the direction of the mobile device.
2. To calculate the POIs that should be visible on screen.
3. To calculate their coordinates on screen.
4. To create their projection on screen as buttons.

Therefore, as the user changes his/her phone's direction, these listed POIs will be recalculated and relocated on screen, in a most quick way.

There are additional features such as:

- Distance and POI's type filters
- Google map (POIs are listed on this map)
- GPS modes
- Choice for GPS fetching libraries
- Customer login
- QR Code reading

In this document, all these features and their implementations will be explained, together with the AR implementation on Android devices will be told.

1.3.CHALLENGES IN AUGMENTED REALITY APPLICATIONS

The most frequently encountered problems when developing an AR application on mobile devices can be listed as follows:

- Challenges of real-time calculation of the reference frame,
- Difficulty to estimate the noise coming from the sensor,
- Difficulty of place the POI to the exact location on the screen,
- Difficulty to predict and satisfy users' demands,

- Difficulty to create real-time graphics, and
- Difficulty to overcome the problems of the poor quality camera.

In this thesis, our objectives are to overcome these problems and additionally to optimize the application in terms of energy consumption. In this document, the emphasize is on screen positioning algorithm. For more detailed explanations on energy optimizations, you can refer to (Karaman, 2015).

1.4.THESIS OBJECTIVES

In this thesis, a new approach to screen positioning algorithm in AR applications is introduced. The main aim is to improve our knowledge in following areas:

1. Developing an AR application.
2. Creating models and designing architectural infrastructure of a large-scale application.
3. Introducing a new approach to screen positioning on AR applications.
4. Comparison to other applications for evaluating the performance of the proposed tagging/positioning algorithm.
5. Optimizing energy consumption (Karaman, 2015).
6. Comparing different libraries (Karaman, 2015).
7. Comparing different modes of usage (Karaman, 2015).
8. Comparison of our algorithm with other algorithms in terms of computation time (Karaman, 2015).

The tests and comparison of the proposed algorithm to other similar applications show that the introduced application with the proposed algorithm meets our expectation. As a result, an application which produces reliable outputs in the most energy efficient way possible has been created

.

2. FUNCTIONAL ANALYSIS OF SARAS

In this chapter, the software model of our thesis is told. Representation of actors of our application, explanation of architecture and usage of our application using UML diagrams such as activity and use case diagrams can be found in this section.

2.1. ACTORS

Actors of the system are as follows:

- **SARAS**

It is the main user of our system. It is an augmented reality application for mobile devices which aims to show information on screen.

- **YKB Customer**

Yapı Kredi Bank's customers who uses SARAS application. They identify themselves to system using their Turkish Citizenship number. They can change application's filters, and they can access to POI's campaign information.

- **Potential Customer**

This actor represents the users who are not Yapı Kredi customers yet, but who uses SARAS. They can access to general campaign information.

- **Database**

This database is located in Galatasaray University and it keeps customers T.C. numbers, campaign and POI's location information. The data is received from bank servers.

2.2.USE CASE DIAGRAMS: SARAS USAGE

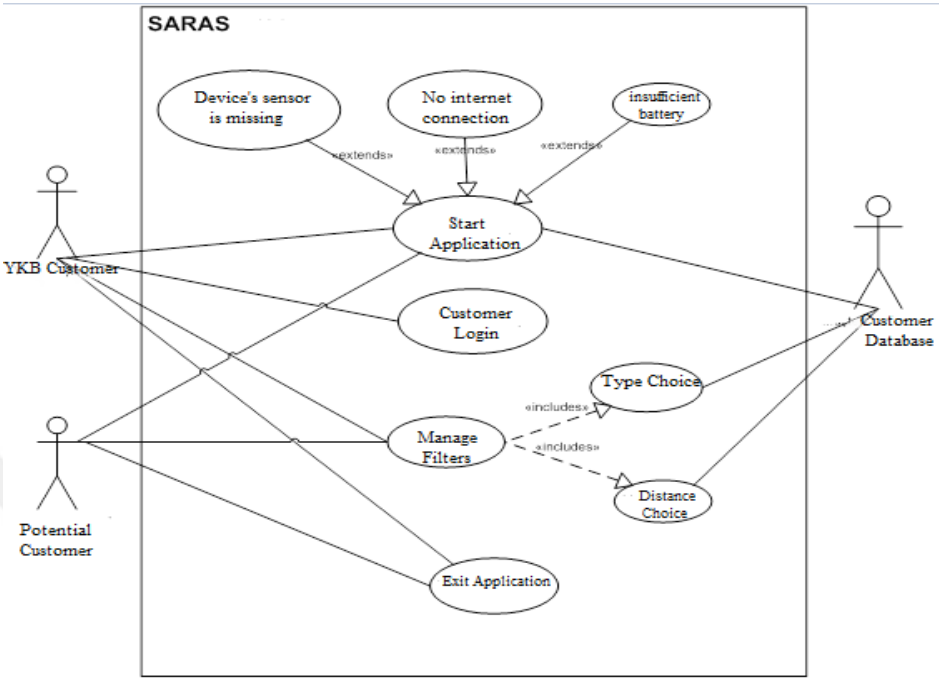

Figure 2.1 Use Case Diagram of SARAS

In Figure 2.1, following scenario occurs:

1. User wants to launch the application.
2. If the camera is working, if the device has internet connection, and if the battery is enough for the application, the application initialized.
3. T.C. ID number information screen is shown on screen. User can skip this or identify himself/herself to system. So she/he can access to the program as a bank customer or a general user.
4. User can filter POIs by their types or their distance.
5. POIs are listed on screen.
6. If the GPS connection does not exist or it is limited, user can choose to use the QR Code of the POI for accessing campaign information.
7. User exits from the program.

2.3. ACTIVITY DIAGRAMS

2.3.1. Activity 1: System Classifies the User

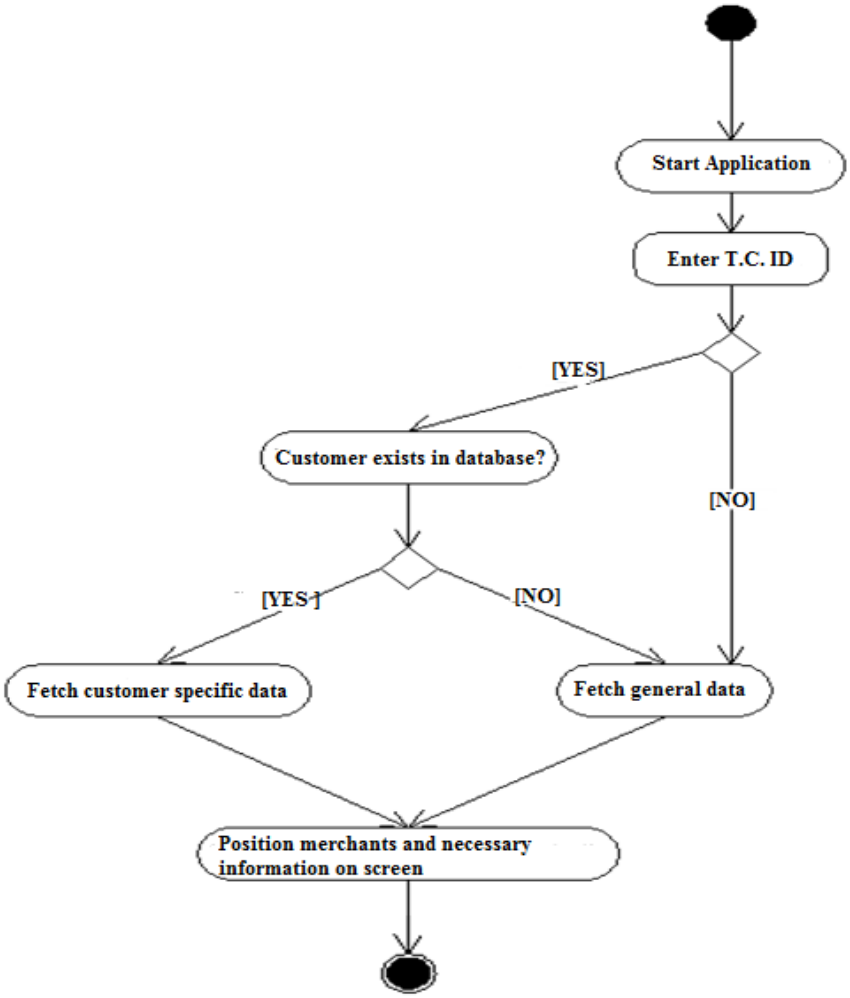

Figure 2.2 User Classification Activity Diagram

In Figure 2.2 user launches the application. At the first time, the application asks for T.C. ID number of the user. If user prefers to give his/her T.C. ID number, the system controls if it belongs to the bank's customer database. If it is the case, bank's customer specific campaigns will be fetched from the remote database. Otherwise, general campaign information are fetched and shown to the user.

2.3.2. Activity 2: Showing POI's Details

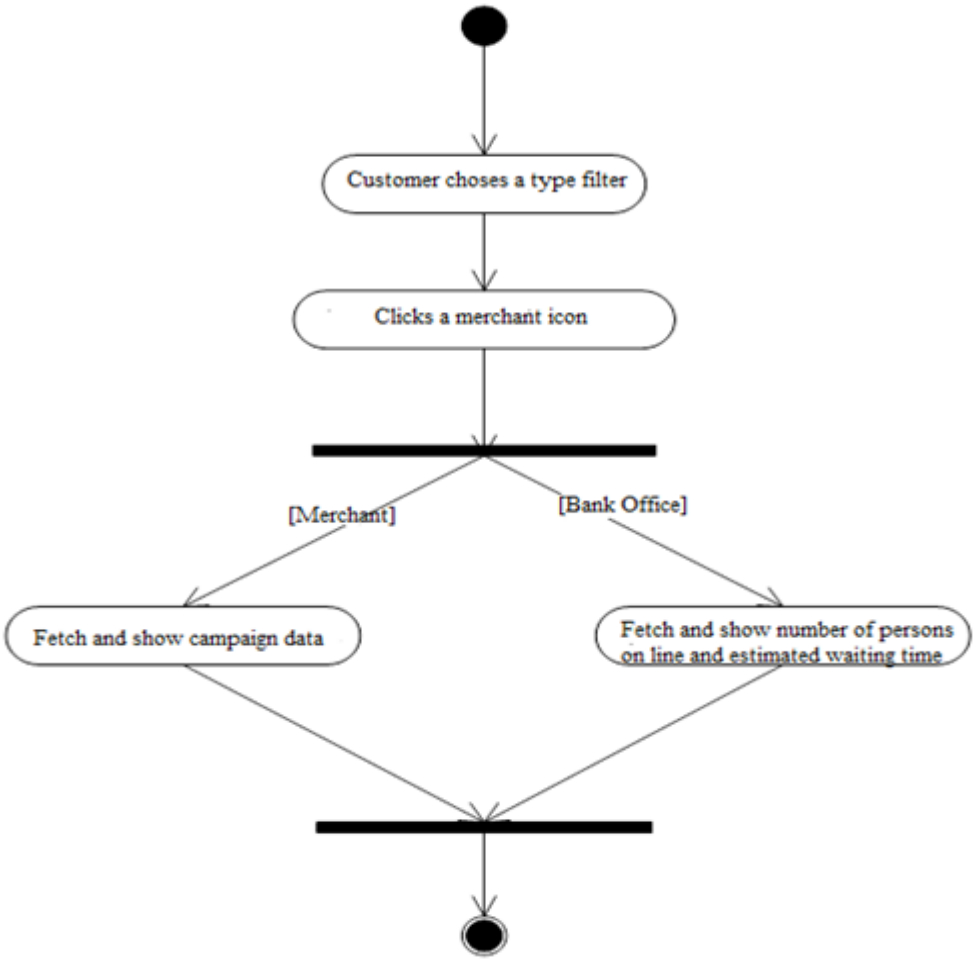

Figure 2.3 POI Details Activity Diagram

In Figure 2.3, fistful the application is initiated, user defines his/her preferred POI type via filters. Only the selected types of POIs are placed on the screen. Customer can press on a specific POI icon on screen in order to see additional details. If the chosen POI is the bank branch, SARAS shows how many people waits on the queue. Otherwise, if it is a common bank merchant, campaign information is shown to the user, if exists.

2.3.3. Activity 3: Case: No GPS Connection

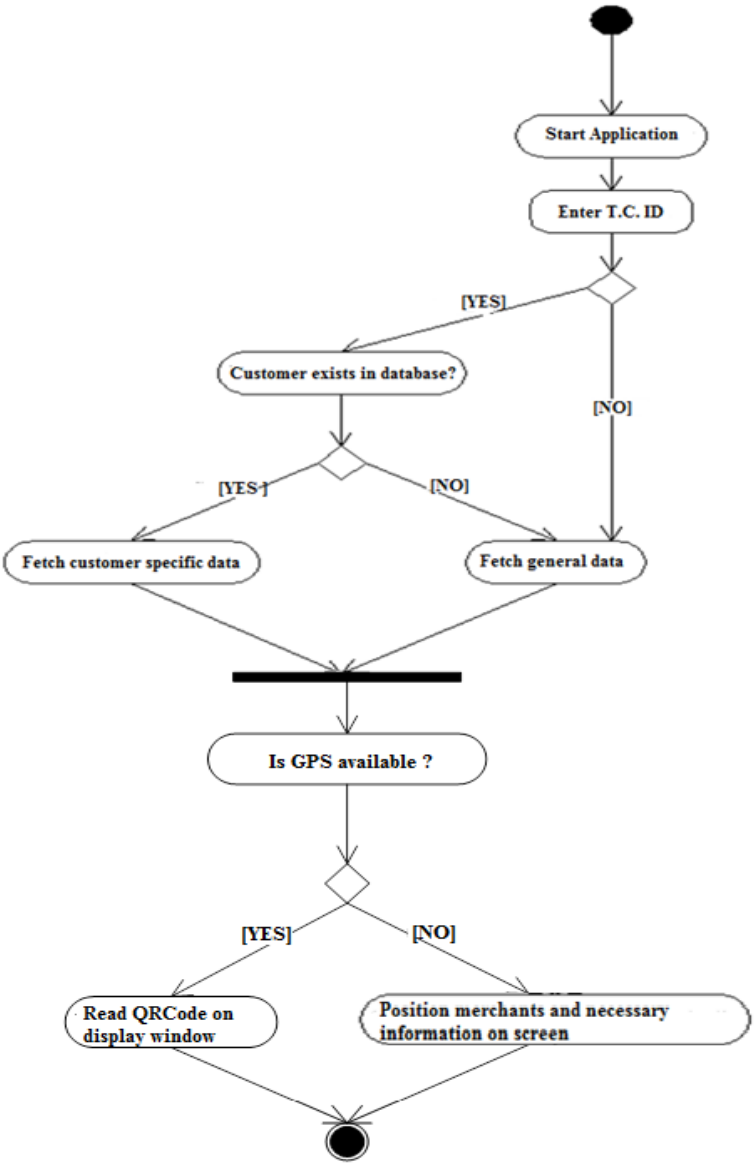

Figure 2.4 No GPS Activity Diagram

Figure 2.4 shows what will happen if the application is used in an environment without GPS connection. User initiates the application. The same controls as in the previous activity are executed. Then, the GPS connection is controlled. If there is not any

connection or the connection is limited, user should use QR Code screen for accessing campaign information.

User can read QR Code stickers placed on bank merchants' stores and offices. Related campaign information is fetched from the remote database using merchant id hidden on this sticker.

3. SIMILAR APPLICATIONS

There are already a few mobile commercial applications developed with the same objective. In this section, we give short information of such applications.

3.1.LAYAR

Layar (Layar, 2009) uses a mobile phone's built-in camera, compass, and GPS to display content about the outside world as the user navigates through it. Someone who has downloaded the application on their phone can walk through a city and see which buildings have apartments that are for rent, learn interesting facts about specific neighborhoods, and read reviews of the restaurants they pass.

3.2.GOOGLE GOGGLES

Google Goggles (Google Goggles, 2014) is an image recognition mobile application developed by Google. It is used for searches based on pictures taken by handheld devices. For example, taking a picture of a famous landmark searches for information about it, or taking a picture of a product's barcode searches for product's information.

3.1.WIKITUDE

Wikitude (Wikitude, 2008) is mobile augmented reality software, which is developed by the Austrian company Wikitude GmbH. It displays information about the users' surroundings in a mobile camera view, including image recognition and 3D modeling.

For location-based AR, the position of objects on the screen of the mobile device is calculated using the user's position (by GPS or Wifi), the direction in which the user is facing (by using the compass) and accelerometer. Figure 3.1 illustrates Wikitude application's screenplay.

Figure 3.1 Wikitude (Wikitude Image,2008)

3.2.TUSCANY+ (APPLE)

Tuscany+ (Tuscany+, 2010) is the first AR application created specifically for the tourism sector. It is based on innovative technology that, when pointed at any real space through the phone's camera, overlays a range of information, be it multimedia, virtual elements, geo-localized data, or information from websites. It is created with the idea of offering travelers in Tuscany an interactive, real-time guide in order to enhance the trip, already rich in culture and traditions, with virtual and multimedia technology. Figure 3.2 shows main screen of Tuscany+.

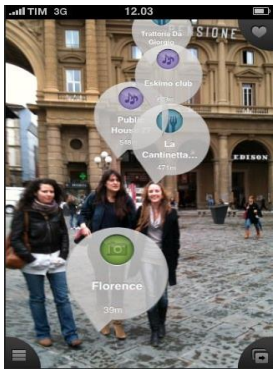

Figure 3.2 Tuscany + (Tuscany+ Image, 2010)

4. SCREENS AND MENUS

In this section, screens, menus of SARAS and their utilizations are explained.

4.1. MAIN SCREEN

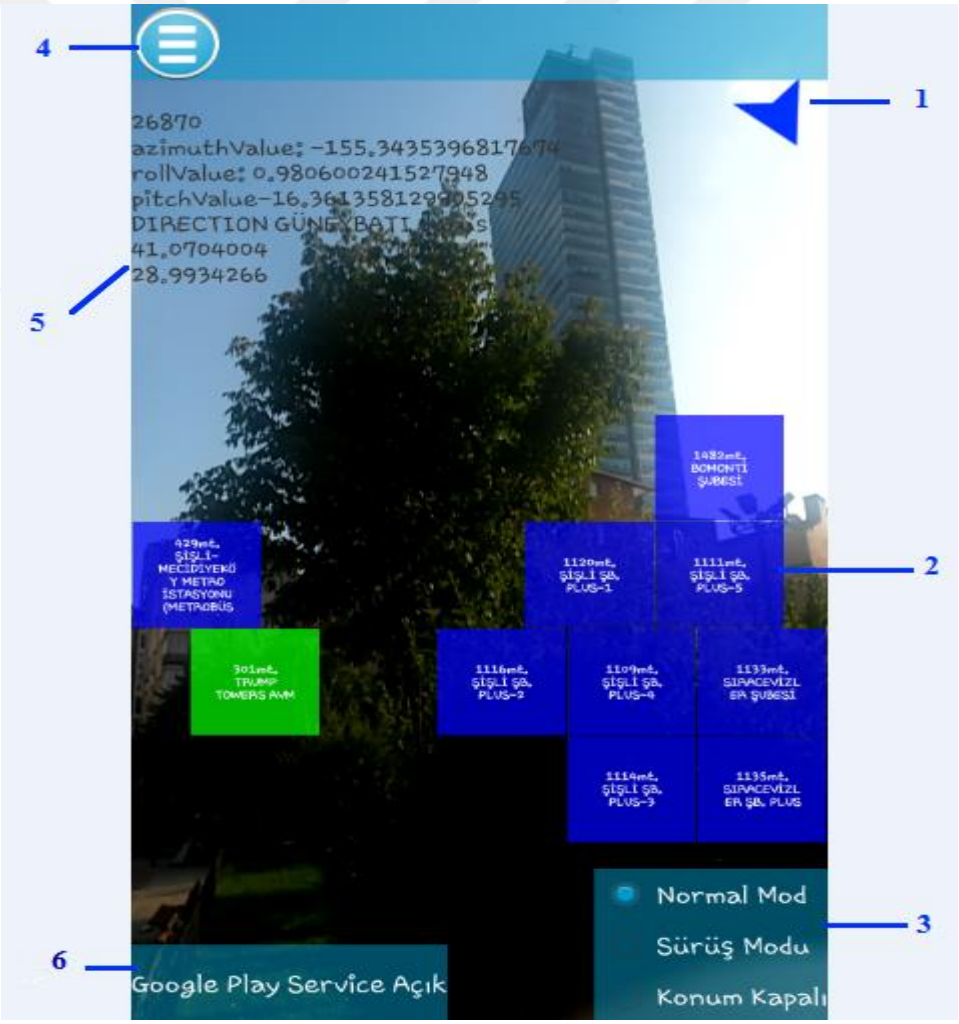

Figure 4.1 SARAS Overall Screen

1. This is the compass. Its pointy edge points the North. When it is clicked, the calibration information page is accessed. This page shows how to calibrate the mobile device. The device should be calibrated if this compass is not correctly pointing North.

2. This is how the POI is shown on the screen. There are three categories and depending on these categories, button's background color is changed. Red is for associate work places, green is for ATM and blue is for bank office. In Figure 4.2 you can see these different colors of buttons.

Figure 4.2 Button Colors for POI Types

3. These are GPS modes. User can change the frequency of GPS updates using this radio group. This is explained in section called 'GPS Modes'.

4. This is the button of the access menu. It enables controlling filters, to login, to view the map, etc… Device's 'settings' button may also be used to access this menu, if the device has one.

5. These are the information used on the development for SARAS. The sensor values, device's direction and GPS values are shown.

6. In our thesis Google Play Service's GPS library and also Android GPS library are used. Using this layout we can switch between these two methods for obtaining device's GPS coordinates.

4.2.MENU

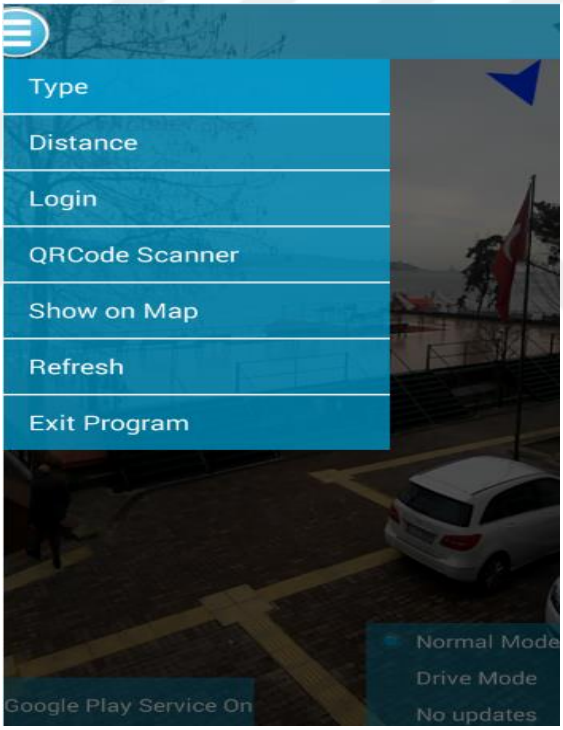

Figure 4.3 Slider Menu

Using the menu on Figure 4.3, the type filter, the distance filter, the login screen, the QR code reader screen, and the map screens are accessible. The local database can be refreshed, and current GPS coordinates can be added as a new POI to the local database or the application may be turned off. It is developed using Slider Menu.

4.3.TYPE FILTER SCREEN

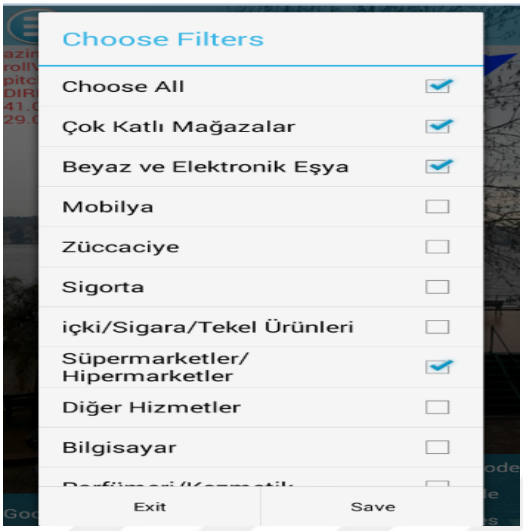

Figure 4.4 Type Filters

Using the type filter screen in Figure 4.4, user can prefer POI types to be listed on screen. This filter is also applied to the map. There is also the 'choose all' option. User can save (*Kaydet*) or discard (*Çıkış*) the changes.

4.4. DISTANCE FILTER SCREEN

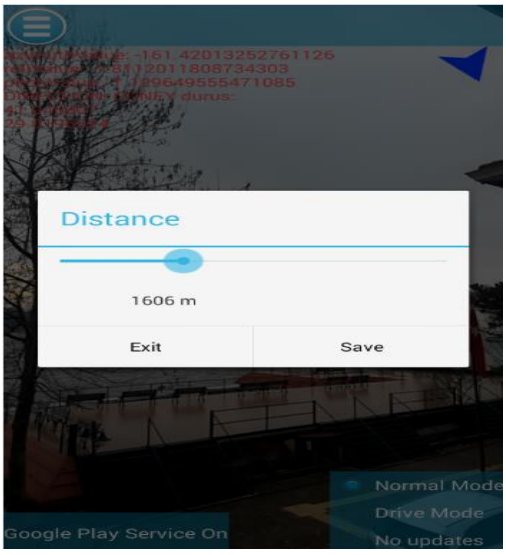

Figure 4.5 Distance Filter

Using this distance filter screen in Figure 4.5, user can filter POIs to be listed on screen depending on their distance. This filter is not applied on map. It has a maximum of 5000 meters value. User can save (*Kaydet*) or discard (*Çıkış*) the changes.

4.5. LOGIN SCREEN

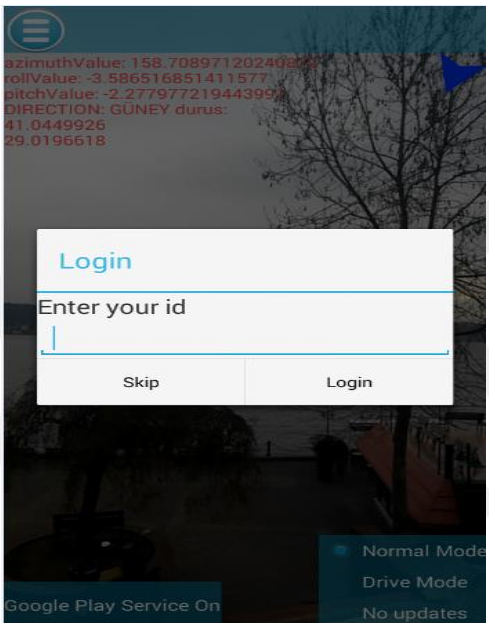

Figure 4.6 Login Screen

Using login screen in Figure 4.6, user can try to login to application. It asks for T.C. id number and controls if this number is registered to the bank database. This login screen can be skipped as it is not necessary to be a bank customer for using SARAS. However, the bank customers and non-customers may be offered different campaigns. If it is the first time that SARAS is launched, it is initiated with this login screen.

4.6. QR CODE SCREEN

Figure 4.7 QR Code Screen

This screen in Figure 4.7 is used for reading QR codes. Because of the poor quality of GPS, SARAS does not list POIs on screen, if user is at an indoor environment like a mall or parking garage. Therefore, he/she can be informed about the campaigns by reading the QR code.

4.7.MAP SCREEN

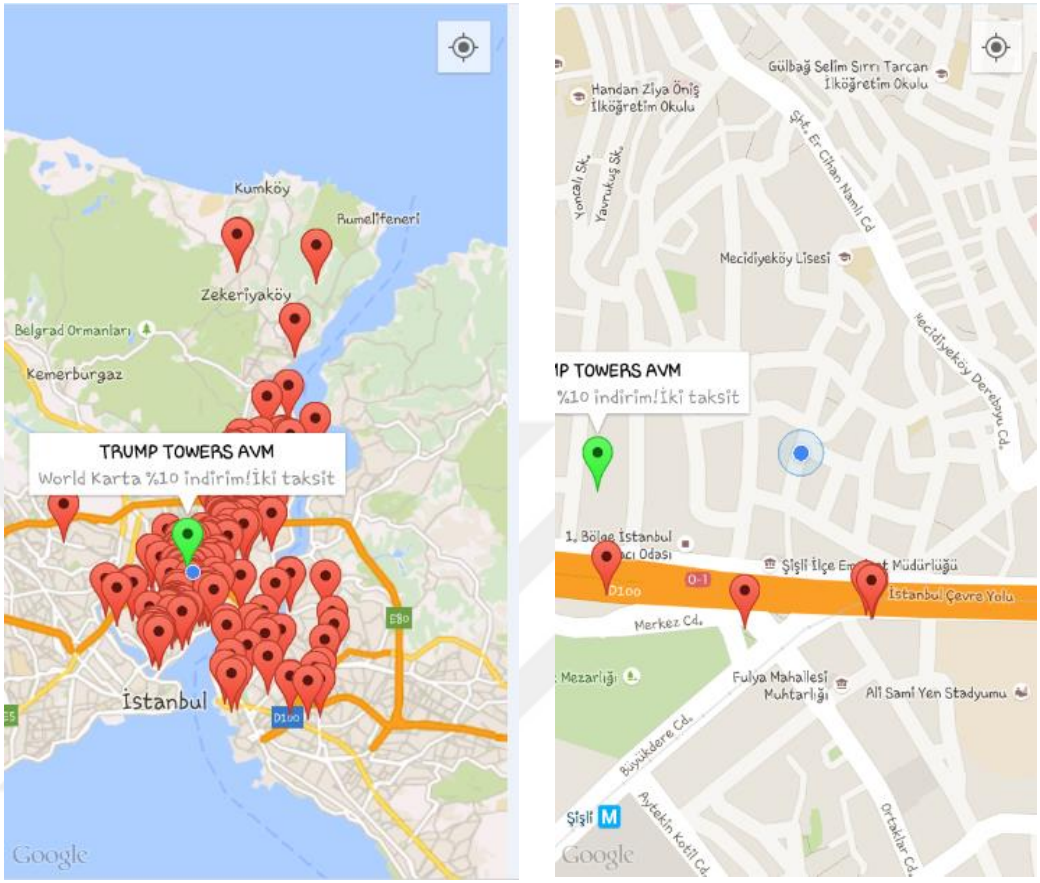

Figure 4.8 Google Map Screen

If the device supports Google Play Services, all the POIs may be seen on the map as in Figure 4.8. Type filter is implemented, but the distance filter is ignored. Markers are in green for merchants who have a campaign, otherwise they are red.

4.8. REFRESH SCREEN

Using refresh button, user can refresh POI values allocated on device's local database. This cleaning and refilling process takes approximately 5 seconds; during which user see an animated wait screen. In Figure 4.9 this waiting screen is displayed.

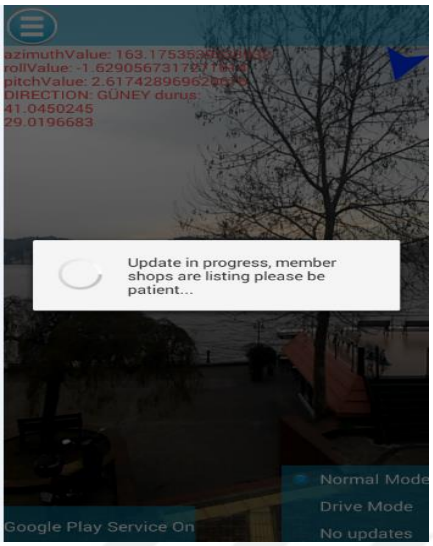

Figure 4.9 Refresh Screen

4.9. ADD A NEW POI

Using add button in Figure 4.10, the current GPS values may be added as a new POI into the local database. It is not sent to database in the server, but only a local change is made. It is used for test purposes, so this option will not be integrated to SARAS.

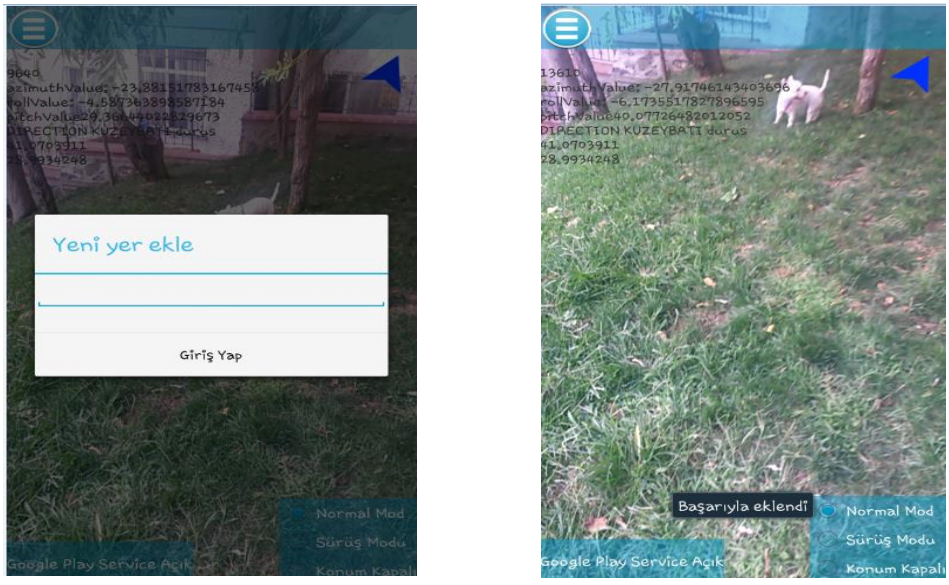

Figure 4.10 Add a New POI Screen

4.10. EXIT SCREEN

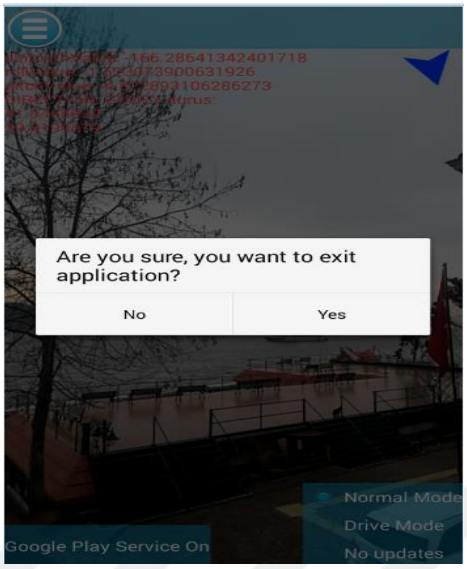

Figure 4.11 Exit Confirmation Screen

If user wants to quit the program, a confirmation message is shown. User can either accept or decline to quit. Figure 4.11 represents this confirmation screen.

4.11. POI DETAIL SCREEN

Figure 4.12 POI Detail Screen

Pressing POI buttons visible on screen you can access to detailed information. You can see campaign information, latitude, longitude merchant name etc. This screen can be seen in Figure 4.12.

4.12. CALIBRATION SCREEN

This screen in Figure 4.13 can be accessed using the compass image on main screen. It shows how to calibrate the device's sensor. This screen can be used when sensor values are wrong, and the compass is not correctly pointing the North.

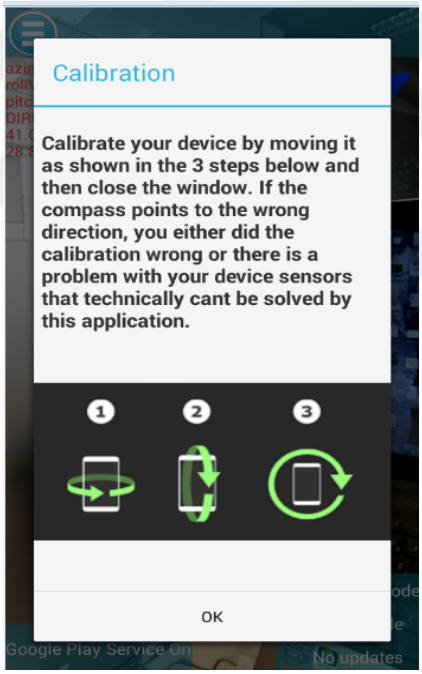

Figure 4.13 Calibration Screen

5. APPLICATION COMPONENTS

A typical AR (Conder et al., 2011) implementation contains two main parts: the “live” data we're augmenting and the “meta” data used for the augmentation.

The augmentation data source can be anything, but often it’s a preloaded database or a web service that can filter to nearby points of interest. The rest of the AR implementation consists of using device camera APIs, graphics APIs, and sensor APIs to overlay the augmentation data over the live data and create a pleasant augmented experience.

For instance, in our application the locations of bank offices and merchants are wanted to be seen in the view finder, the AR “service” must have augmentation data for each merchant, including its latitude, longitude and also altitude. Using this information, and the direction in which the device/camera is pointing, we can approximate the location of each bank office as an overlay on the view finder window and display a little icon in form of a button on or above its location.

In this section, different components of mobile device used in this thesis for achieving this screen positioning are explained. The usage of the mobile device’s camera, motion and position sensors, local storage support, location services will be told and also additional services that our application offers such as: location service choice, compass, error reporting, language support and QR code reading will be represented.

5.1.CAMERA

Displaying the live feed from the Android camera is the reality in AR. The camera (Android Camera, 2015) data is available by using the APIs available within the android.hardware.Camera package.

Our application does not need to analyze frame data, so I started a preview in the normal way by using a *SurfaceHolder* object with the *setPreviewDisplay*() method. With this method, it becomes possible to display what the camera is recording on the screen for use.

For this AR application, I created a camera application following these steps:

1. Identify and reach the camera: Applications should check that the device has a camera and must try to reach the camera if necessary.
2. Create a preview: I created a preview, which includes the surface hold interface extended from surface view.
3. Establish the preview layer: I managed the class through a layer of this preview.
4. Releasing the camera: I release camera after using it for other applications.

For accessing the camera we added following permission to our application:

<uses-permissionandroid:name="android.permission.CAMERA"/>

And also to be able to access camera features we added following feature:

<uses-featureandroid:name="android.hardware.camera"/>

5.2. LOCATION

We need to determine the location of the device (and therefore its user). This way, this application can listen to location events and use those to determine where live items of interest are located in relation to the device.

GPS is in any unobstructed line of sight in the world, four or more satellite locations in all weather conditions and space-based satellite navigation system that provides the time information. For accessing GPS values following permission are added to manifest file:

<uses-permissionandroid:name="android.permission.ACCESS_FINE_LOCATION"/>

In our application two different methods to find the GPS position are implemented.

5.2.1. Google Play Services

We can detect device's GPS coordinate using Google Play Services library.

- **Adding Google Play Services**

As this application needs Google Play Services (Ravi, 2013), we need to setup the play services first.

1. As in Figure 5.1 open Android SDK Manager and install or update the play services under Extras section.

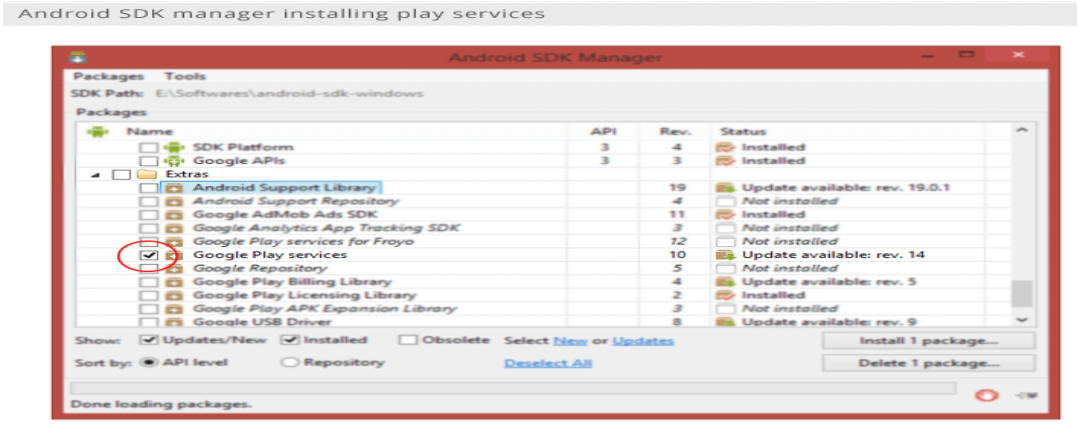

Figure 5.1 Download Google Play Services

2. You should import Google Play Services into your workspace as a library.

3. You should add this library into your project.
4. Create Google Maps API key

Google Maps API key is created as follows:

4.1. You should create Google Maps API fingerprint by writing following statement in Terminal. In figure 5.2 you can see SHA key created after writing following command:

keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android

```
C:\Program Files\Java\jdk1.7.0_09\bin>
C:\Program Files\Java\jdk1.7.0_09\bin>keytool -list -v -keystore "%USERPROFI
.android\debug.keystore" -alias androiddebugkey -storepass android -keypass
oid
Alias name: androiddebugkey
Creation date: 31.Oca.2014
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 24ea487b
Valid from: Fri Jan 31 14:05:51 EET 2014 until: Sun Jan 24 14:05:51 EET 2044
Certificate fingerprints:
         MD5:  9A:28:77:A3:31:4D:92:D5:AC:DE:6C:84:D5:FE:42:2F
         SHA1: 09:3D:BD:9A:FE:3E:64:2C:61:13:ED:EA:AC:A1:2F:40:48:E6:5D:3B
         SHA256: 6E:BF:9D:91:22:F5:5A:12:03:54:73:0B:33:BA:C5:04:C6:85:38:AE
B9:08:9B:15:C0:7D:EA:25:A6:7F:FA
         Signature algorithm name: SHA256withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 55 62 6D 3C DF B0 A9 39   F4 15 AB 7C 11 6F 31 BC  Ubm<...9.....o1.
0010: FD 6D 7F 0F                                        .m..
]
]
```

Figure 5.2 Create SHA Key

4.2 Then connect to Google API console by following address *https://code.google.com/apis/console/?noredirect#project:731857127600:access*
You should allow usage of Google Maps for our project as in Figure 5.3

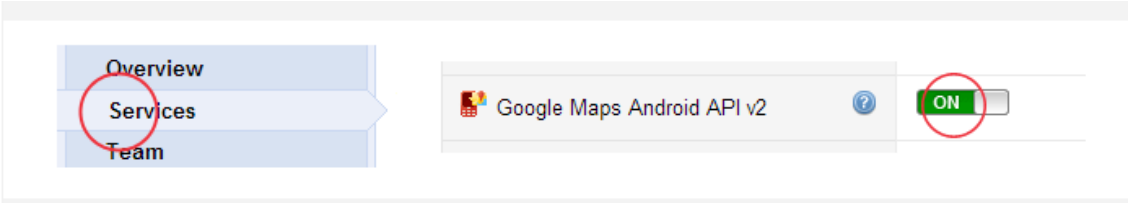

Figure 5.3 API Console

4.3. Open API access tab and create a new Android key using SHA1 fingerprint.

4.4. Finally, we get an API key as you can see in Figure 5.4.

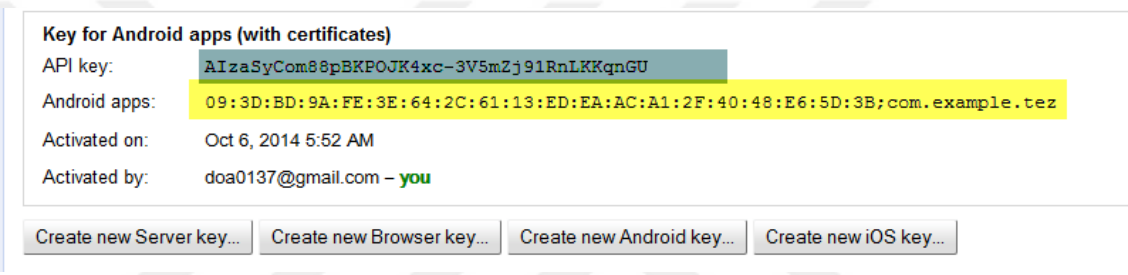

Figure 5.4 API Key

5. Change Android-Manifest.xml file as in Figure 5.5:

```
</activity>
<!-- GOOGLE PLACES -->
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyB6menWM4_MgCcm2vr_Wpwc06FKUPFo_TY" />
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

Figure 5.5 Adding API to Manifest File

Add Google Play Services as a library into your project as Figure 5.6 shows.

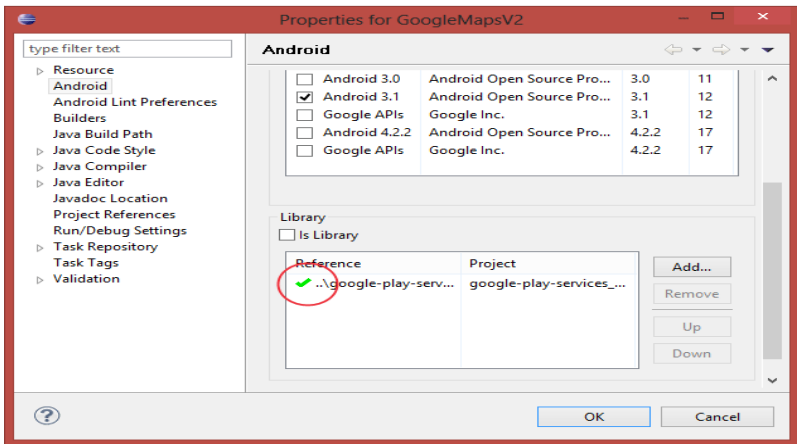

Figure 5.6 Google Play Services Library

Add permissions in Figure 5.7 into your manifest file.

```
<permission
    android:name="info.androidhive.googlemapsv2.permission.MAPS_RECEIVE"
    android:protectionLevel="signature" />

<uses-permission android:name="info.androidhive.googlemapsv2.permission.MAPS_

<uses-sdk
    android:minSdkVersion="12"
    android:targetSdkVersion="17" />

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.RI
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!-- Required to show current location -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Figure 5.7 Map Permissions

- **Google Play Service Usage**

A class called GooglePlayServices.java is created.

Following code snippet can be used to find out if the device support or not Google Play Services:

GooglePlayServicesUtil.*isGooglePlayServicesAvailable*(**context**);

In our main activity, this code block is used and the application gives the possibility to choose whether to use Google Play Services library or Android Location Library.

In our scenario, our application needs location updates periodically. If the user is static, if he is not changing location, it should not be updated. In order to control this, this following code block is added:

Google Play Services:

mLocationRequest.setSmallestDisplacement(1);

Android Location Library:

lm.requestLocationUpdates(LocationManager.*GPS_PROVIDER*, 20000, 1, locationListenerGps);

This means ‘do not take updates if the user has not displaced more than 1 meter’.

If Google Play Services is supported user can also use Google Map and can see all the POIs on the map.

- **Google Map**

We have created a class called MapViewActivity.java for implementing Google Maps Screen. This class extends from f*ragmentActivity* to support the Google Maps feature.

The user's location is shown with a blue dot on the map. Using markers, POIs are placed on the map. Their GPS (latitude and longitude) data is used to place them correctly on the map. As the user press on this marker, a snippet is shown with this POI’s name, GPS location and campaign information on it.

The marker is placed on the map as follows:

Marker marker = googlemap.addmark (new markeroptions (). Position (11));

Type filters are also implemented in here. So, only the POIs for chosen types are located on map. However, distance filter has been ignored; user can see all the POIs available in our server's database.

Only POIs on the visible part of the map are emplaced, and as the user scrolls on the map markers for other POIs are added respectively. This makes our application faster.

Related screenshots are as follows (Figure 5.8). As is visible in Figure 5.8, merchant with campaign is green and others are all red.

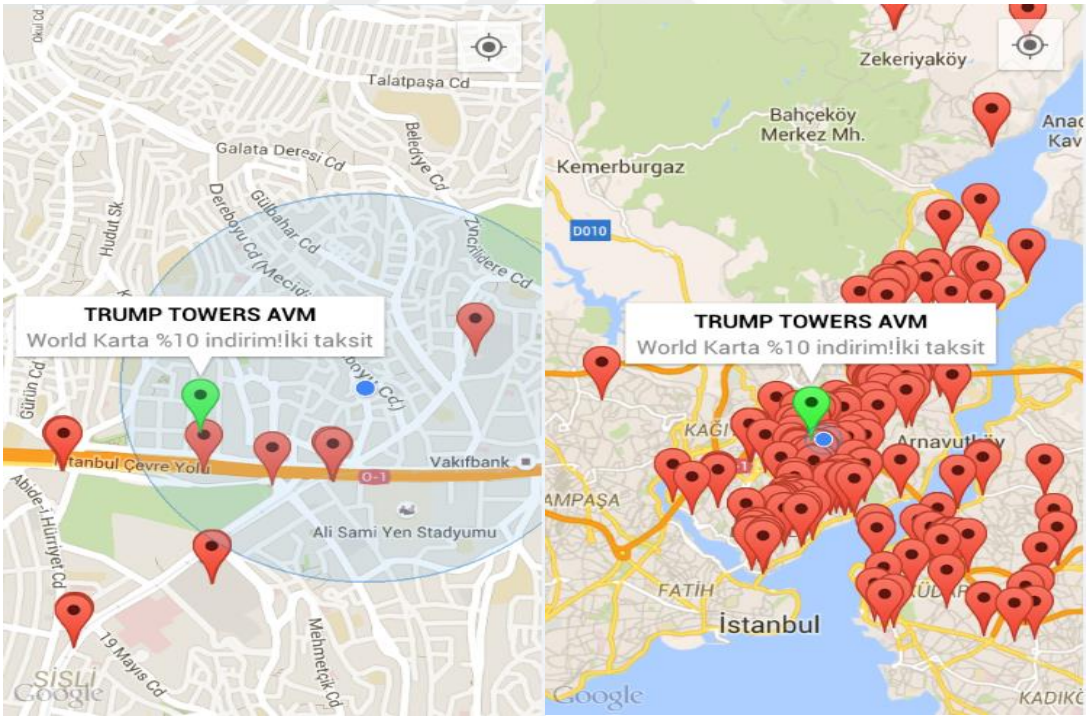

Figure 5.8 POIs on Map

5.2.2. Android Location Library

If the device does not support Google Play Services, we use Android Location Library. In this mode, user can't access map.

For locating user, we use GPS_PROVIDER and if it cannot be detected we use NETWORK_PROVIDER.

5.2.3. GPS Modes

In our application, we take location updates as the user has displaced minimum of 1 meter. But for performance testing purposes this is omitted. These updates are based on time. Three different GPS modes are added on our application for both Google Play Service version and Android Location Library version. These modes are:

- **Normal Mode**

This mode is the default mode. Application starts in this mode. We force the application to take updates every 20 seconds. This should consume less energy. This can be used when the user is walking.

- **Drive Mode**

We force the application to request updates every 1 second. This should consume more energy. This can be used when the user is driving or displacing quickly. All devices are not able to take location updates every 1 second, so this mode's update range changes between devices. For example HTC XLARGE minimum update range is every 10 seconds and on SAMSUNG S4 MINI it is 15 seconds.

- **No Updates Mode**

Application does not request location updates. This should consume less energy. This can be used when the user is stable. It assumes that user is at the same place and makes its operations based on the last GPS taken.

You can see test results on (Karaman, 2015). As mentioned before, these modes are only created for test purposes. Best way to achieve our goal is to take location updates as the user has displaced minimum of 1 meter. So we will deliver our application to the bank using this method.

5.3.SENSORS

Sensor data is often important for AR implementations. For example, knowing the orientation of the phone is usually very useful when trying to keep data synchronized with the camera feed.

To determine the orientation of an Android device, we need to leverage the APIs available in the android.hardware.SensorManager package.

The use of sensors to allow the user to move the device around and see changes on the screen in relation to it, really pulls the user into applications in an immersive fashion. When the camera feed is shown, this is critical.

For this purpose I used following sensors:

5.3.1. Motion Sensors

Motion sensors (Android Motion Sensors, 2015) are useful for monitoring device movement, such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game or a user controlling a

ball in a game), but it can also be a reflection of the physical environment in which the device is located (for example, moving with you while you drive your car). In the first case, you are monitoring motion relative to the device's frame of reference or your application's frame of reference; in the second case you are monitoring motion relative to the world's frame of reference.

All of the motion sensors return multi-dimensional arrays of sensor values for each SensorEvent.

1. Accelerometer

An acceleration sensor measures the acceleration applied to the device, including the force of gravity. The code in Figure 5.9 shows you how to get an instance of the default acceleration sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
  ...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Figure 5.9 Accesses to Accelerometer

Accelerometers use the standard sensor coordinate system. In practice, this means that the following conditions apply when a device is lying flat on a table in its natural orientation:

- If you push the device on the left side (so it moves to the right), the *x* acceleration value is positive.
- If you push the device on the bottom (so it moves away from you), the *y* acceleration value is positive.

- If you push the device toward the sky with an acceleration of A m/s^2, the *z* acceleration value is equal to A + 9.81, which corresponds to the acceleration of the device (+A m/s^2) minus the force of gravity (-9.81 m/s^2).
- The stationary device will have an acceleration value of +9.81, which corresponds to the acceleration of the device (0 m/s^2 minus the force of gravity, which is -9.81 m/s^2).
- In general, the accelerometer is a good sensor to use if you are monitoring device motion. Almost every Android-powered handset and tablet has an accelerometer, and it uses about 10 times less power than the other motion sensors. One drawback is that you might have to implement low-pass and high-pass filters to eliminate gravitational forces and reduce noise.

Figure 5.10 represents *x*, *y* and *z* axis.

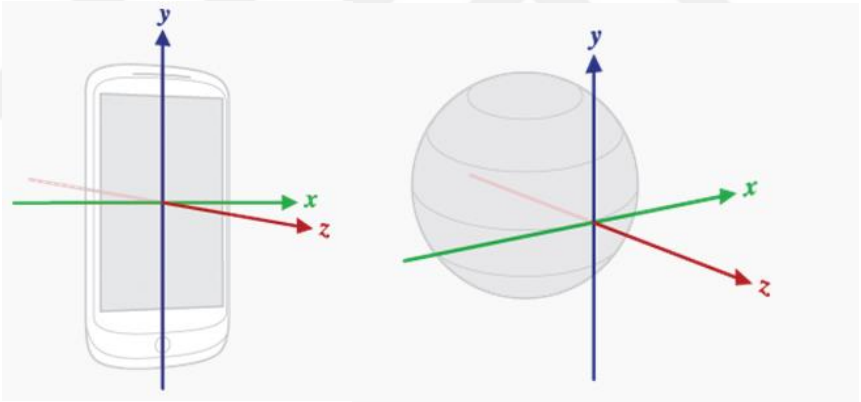

Figure 5.10 Axis

2. Gyroscope

The gyroscope measures the rate or rotation in rad/s around a device's *x*, *y*, and *z* axis. The code in Figure 5.11 shows you how to get an instance of the default gyroscope:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

Figure 5.11 Access to Gyroscope

The sensor's coordinate system is the same as the one used for the acceleration sensor. Rotation is positive in the counter-clockwise direction; that is, an observer looking from some positive location on the *x*, *y* or *z* axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. This is the standard mathematical definition of positive rotation and is not the same as the definition for roll that is used by the orientation sensor.

5.3.2. Position Sensors

Motion sensors by themselves are not typically used to monitor device position, but they can be used with other sensors, such as the geomagnetic field sensor, to determine a device's position relative to the world's frame of reference. Position sensors (Android Position Sensors,) are useful for determining a device's physical position in the world's frame of reference.

1. GeoMagnetic_Field

The geomagnetic field sensor lets you monitor changes in the earth's magnetic field. Figure 5.12 shows you how to get an instance of the default geomagnetic field sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Figure 5.12 Access to Geomagnetic Field

This sensor provides raw field strength data (in μT) for each of the three coordinate axes. Usually, you do not need to use this sensor directly. Instead, you can use the rotation vector sensor to determine raw rotational movement or you can use the accelerometer and geomagnetic field sensor in conjunction with the getRotationMatrix() method to obtain the rotation matrix and the inclination matrix.

2. Orientation Vector

The orientation sensor derives its data by using a device's geomagnetic field sensor in combination with a device's accelerometer. Using these two hardware sensors, an orientation sensor provides data for the following three dimensions:

Azimuth (degrees of rotation around the z axis in Figure 5.13): This is the angle between magnetic north and the device's y axis. For example, if the device's y axis is aligned with magnetic north this value is 0, and if the device's y axis is pointing south this value is 180. Likewise, when y axis points east this value is 90 and when it points west this value is 270.

Pitch (degrees of rotation around the x axis in Figure 5.13): This value is positive when the positive z axis rotates toward the positive y axis, and it is negative when the positive z axis rotates toward the negative y axis. The range of values is 180 degrees to -180 degrees.

Roll (degrees of rotation around the y axis in Figure 5.13): This value is positive when the positive z axis rotates toward the positive x axis, and it is negative when the positive z axis rotates toward the negative x axis. The range of values is 90 degrees to -90 degrees.

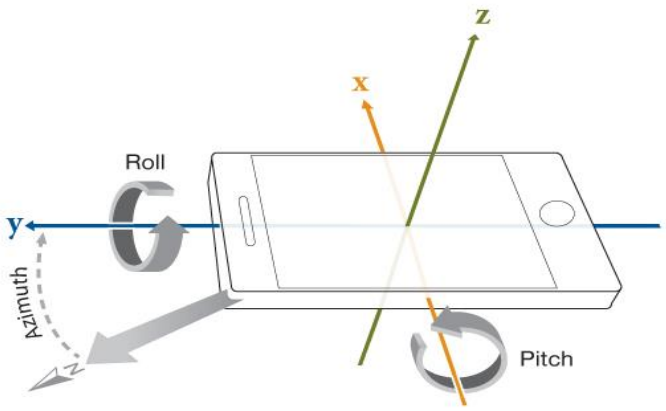

Figure 5.13 Azimuth Roll and Pitch (Azimuth Pitch Roll Image)

In order to define if the user is directed towards the north I calculated this azimuth value. And also I used pitch value to determine if the device is looking down on the ground or up in the sky. And I used these result on my screen positioning algorithm.

The orientation sensor derives its data by processing the raw sensor data from the accelerometer and the geomagnetic field sensor. We manage that using getRotationMatrix() method to compute orientation values. And also we should use the remapCoordinateSystem() method to translate the orientation values to your application's frame of reference.

5.3.3. Rotation Matrix

getRotationMAtrix() method Computes the inclination matrix ***I*** as well as the rotation matrix ***R*** transforming a vector from the device coordinate system to the world's coordinate system which is defined as a direct orthonormal basis, where:

- *X* is defined as the vector product ***Y.Z*** (It is tangential to the ground at the device's current location and roughly points East).
- *Y* is tangential to the ground at the device's current location and points towards the magnetic North Pole.
- *Z* points towards the sky and is perpendicular to the ground.

Using this matrice we can get azimuth, pitch and roll as follows:

```
SensorManager.getOrientation(matrixR, matrixValues);
double azimuth = Math.toDegrees(matrixValues[0]);
double pitch = Math.toDegrees(matrixValues[1]);
double roll = Math.toDegrees(matrixValues[2]);
```

5.3.4. Remap Coordinate System

RemapCoordinateSystem() (Android Sensor Manager) method rotates the supplied rotation matrix so it is expressed in a different coordinate system. This is typically used when an application needs to compute the three orientation angles of the device in a different coordinate system.

5.3.5. Sensor Fusion

During my research I have found an open source project called SensorFusion (Lawitzki, 2014) which uses accelerometer + geomagnetic field sensors. In addition to that, it uses gyroscope sensor to calculate these azimuth, pitch and roll values. Figure 5.14 represents sensors used in sensor fusion project.

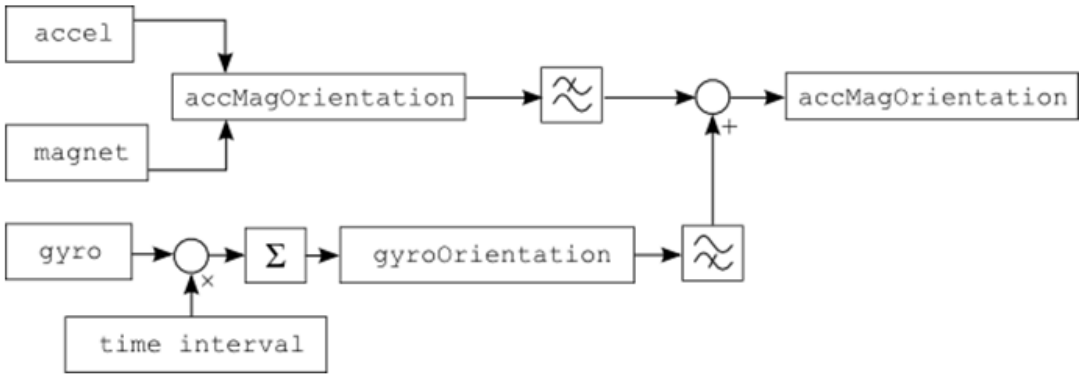

Figure 5.14 Sensor Fusion

As it uses combination of three sensors, its results are more efficient and reliable. It also calculates rotationMatrix and range for azimuth, pitch and roll are as follows:

azimuth: [-180, 180]. -180/180 points South, 0 points North, 90 East and -90 for West
pitch: [-90, 90].
roll: [-180, 180].

5.4.DATA STORAGE ON ANDROID

In our application we need to store data for two reasons:

1. For keeping POI's information.
2. For keeping filters chosen by user.

5.4.1. For keeping POI's information:

POI information will be fetched using web services at the beginning of the program. However, accessing a web service requires so much energy and time. So keeping certain amount of POIs in device's local storage makes the program faster and saves energy. So for this storage we need to save a larger amount of data and also it should be private to our program.

Therefore, we fetch POIs inside a diameter of 5 kilometers via web service, and then keep them in device's local storage. As the user change location, if he gets out of the circle, we reconnect and get data from web service, clean our local database and refill it with the new POIs' information. SQLite database is used for this objective. SQLite is an open source SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

A database called **saras.db** is created using SQLite and also created following tables:

- **SARAS_MEMBER_SHOP TABLE**

This table is used to store all the POIs inside a diameter of 10 kilometers. This table is filled using MerchantServices web service. It gets POIs of all types from our server. Type filter is not added in here because as the use change his choices of type filter we should not be obliged to go to web service. So, when he changes filter for type we connect to our local database and fetch new data quickly.

If it is the first time the application is being used, if the user press refresh button on menu, or if user has changed location and got away 5 km from his first location we refresh this table. The older values are dropped, we reconnect to our server using web service and we refill SARAS_MEMBER_SHOP table with new POIs which are located inside 10 km of diameters. Table 5.1 shows us the table model.

Name
SARAS_MEMBER_SHOP

Column	**Type**	**Not Null**	**PK**	**Purpose**
MEMBER_SHOP_ID	NUMBER	Y	PK	Refers to POI id
MEMBER_SHOP_NAME	TEXT	Y		Refers to POI's name
MEMBER_SHOP_LATITUDE	DOUBLE	Y		Refers to latitude of POI
MEMBER_SHOP_LONGITUDE	DOUBLE	Y		Refers to longitude of POI
MEMBER_SHOP_CATEGORY	TEXT	Y		Refers to category of POI
MEMBER_SHOP_TYPE_ID	NUMBER	Y		Refers to type of POI
COSLAT	DOUBLE	Y		Used for calculating distance
SINLAT	DOUBLE	Y		Used for calculating distance
COSLNG	DOUBLE	Y		Used for calculating distance
SINLNG	DOUBLE	Y		Used for calculating distance

Table 5.1 SARAS_MEMBER_SHOP Table

- **SARAS_MEMBER_CATEGORIES TABLE**

This table is used for feeding category filter on the user menu. As the program is being opened first time, we connect to server using *getMerchants* web service (Chapter 5.8.2.). We get all the possible distinct type available and we fill our table with these values. So we can fill our menu option for type filter using this table.

Table 5.2 is the representation of table model.

Name				
SARAS_MEMBER_CATEGORIES				
Column	**Type**	**Not Null**	**PK**	**Purpose**
CATEGORY_ID	NUMBER	Y	PK	Refers to category id
CATEGORY_NAME	TEXT	Y		Refers to category name
CATEGORY_PRIORITY	NUMBER	Y		Refers to category priority

Table 5.2 SARAS_MEMBER_CATEGORIES Table

5.4.2. For keeping filters chosen by user

User can change distance and type filter, and the program should remember these selections and gather data using these filters. So for this storage we need to save a smaller amount of data and also it should be private to our program.

- **Shared Preferences**

Shared Preferences is used to save a relatively small collection of key-values. A Shared Preferences object points to a file containing key-value pairs and provides simple

methods to read and write them. Each Shared Preferences file is managed by the framework and can be private or shared. In our case we used the private one.
We created following variables with Shared Preferences:

1. **User's first entry to application control: *SARAS_FIRST_ENTRY***

This control is used to bring in front the login screen. If it is the first entry, meaning if this value is null, we show alert dialog for login. This screen asks for Turkish citizenship identity number.

As the user enters his identity number we check if the user is a bank customer or not. The campaign information is different for bank customers.

→If user is a bank customer:

SARAS_CUST_LOGIN parameter will be saved using SharedPrefs with parameter value = 1.

SARAS_CUST_GROUP_ID parameter will be saved using SharedPrefs with parameter value = customer's group information.

→If user is not a bank customer:

SARAS_CUST_LOGIN parameter will be saved using SharedPrefs with parameter value = 0.

SARAS_CUST_GROUP_ID parameter will be saved with parameter value = 0.

2. **Distance Filter**

Distance chosen by user is also saved using Shared Preferences. Parameter's key value is *DISTANCE_VALUE*. Thus we don't lose user's choices even if the program is

terminated. We can get this parameter's value inside different java classes. During the program this parameter's value is used in web services and distance filter alert dialog.

3. Type Filters

Categories chosen by users are also kept using Shared Preferences. For functionality of the program, all distinct categories are fetched at the initialization. They are loaded in our SQLite local database.

Afterwards we feed our type filters with these values kept in our local database. But also we should keep in mind type filters chosen by user. We should not lose them even if the program is terminated.

For this reason we use Shared Preferences. We save all the categories with keys with following format:

TYPE_VALUE_CATEGORY_NAME (for ATM it is *TYPE_VALUE_ATM*)

If it is chosen its value is 1 and if it is unchecked it's value is 0. I implemented these parameters using this generic structure. Therefore, we support the new categories added in program and also we don't lose older values.

4. Location Information

We should update our local database as the user changes his location. Therefore at the time that we fetch data from web service we should save device's GPS values. Using *MYFIRSTGPS_LONGITUDE* and *MYFIRSTGPS_LATITUDE* parameter keys, we save latitude and longitude values. Continuously, we calculate the difference between these first values and our current GPS values. If this distance is bigger (*UPDATE_MIN_VALUE* = 10000 meters), then we recall our web service, clean our table keeping POI information and refill this table with new values. Using Shared

Preferences, we write our new GPS (*MYFIRSTGPS_LONGITUDE*, *MYFIRSTGPS_LATITUDE*) values over old values.

5.5.ERROR REPORTS

This application can contain different errors on different devices. An error report screen is added to our application in order to test it on several devices. If user encounters an error, application can crush down. Therefore, user is led to the mail screen, and application asks him to send us an error report.

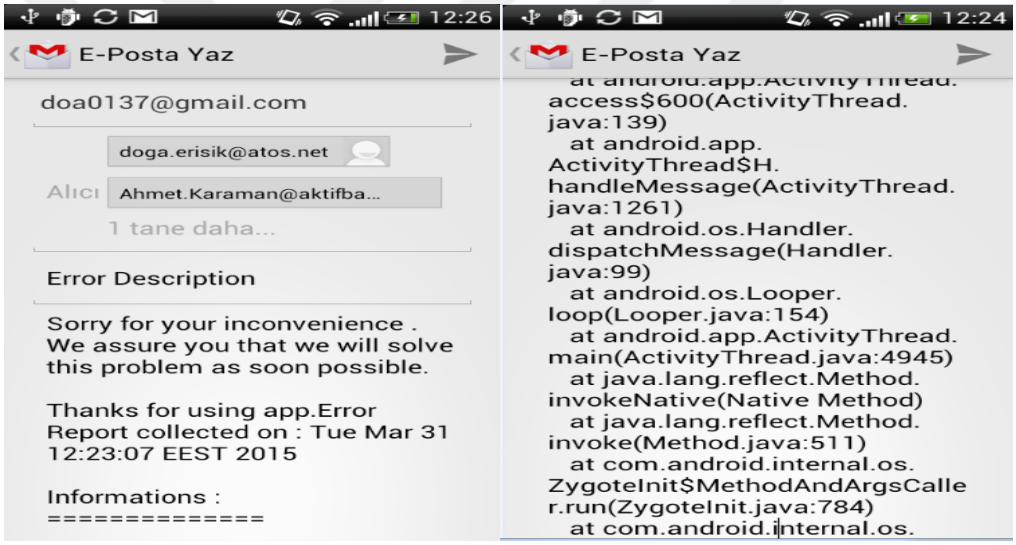

Figure 5.15 Error Report Mail

This error report can be seen in Figure 5.15. It contains error description, cause, and trace. It also contains device's model. The receiver information is filled with mine and Ahmet's e-mail addresses automatically. So, the user can send us this e-mail adding on which operation he/she encountered this error. It will help us to improve and overcome our application and its errors.

For realizing this screen following code block is added to our main thread on MainActivity.java file.

Thread.setDefaultUncaughtExceptionHandler(new ExceptionHandler(this));

ExceptionHandler class extends from java.lang.Thread.UncaughtExceptionHandler class. It contains a method called uncaughtException which originally belongs to UncaughtExceptionHandler class. This method is overwritten in order to assemble device's information and error details. After finding these information user is lead to e-mail screen, filling necessary mail components with these values.

5.6.COMPASS AND CALIBRATION

During the tests, we realized that device's sensors can sometimes calculate false values. For example, it was always showing north even if the device's screen is turned to opposite way. That it is caused because the device has lost its calibration, and it has to be recalibrated. An image from a GooglePlayStore application called Compass (Compass Application, 2014) is implemented to inform the user for recalibrating its device. But, user can not know if sensors calculate false values so a little compass is added in screen as it can be seen in Figure 5.16. It points to North. If it shows false values (for example Is it does not change position) user can understand that the device needs to be recalibrated.

Figure 5.16 Compass

As you can see direction is South (GÜNEY) and the compass's pointy end shows the opposite way as North.

This calibration screen in Figure 5.17 pops out when our application starts, and user can access this information screen by pressing on the compass.

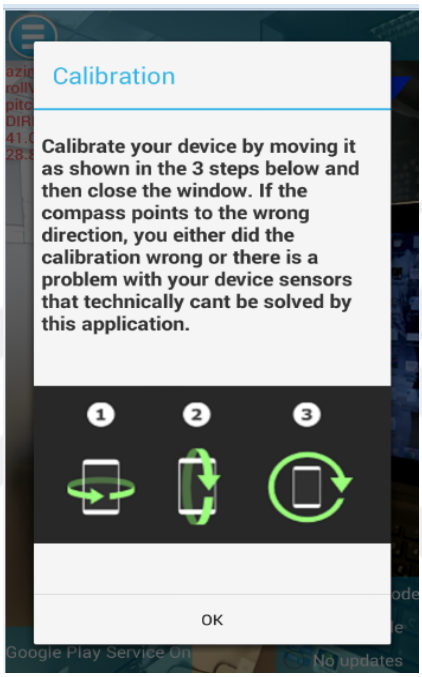

Figure 5.17 Calibration Info

5.7.LANGUAGE SUPPORT

Android makes easy to support different languages in screen components with a resources directory in each Android project. Therefore, in our application we support English and Turkish. So if the device's language setting is set to Turkish, all the warnings and menus are listed in Turkish. Subdirectories and string resource files are created in this purpose. They are located in:

MyProject/ res/ values/ (It is for the English version) strings.xml
MyProject/ res/ values-tr/ (It is for the Turkish version) strings.xml

For example screen warning for no GPS connection is defined:

Strings-tr.xml

<string name="*gpskapali*">

GPS konum servisini açınız, uygulama kapanacaktır.</string>

Strings.xml:

<string name="*gpskapali*">GPS is disabled, application will be closed.</string>

Figure 5.18 represents the case where device's setting is set to Turkish.

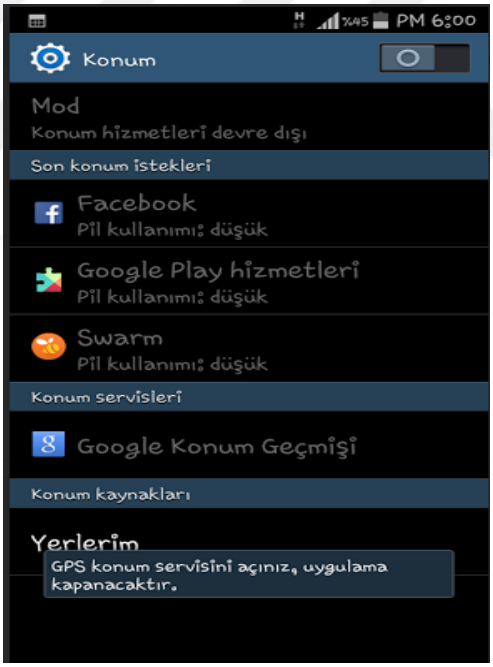

Figure 5.18 Turkish Version

Figure 5.19 represents the case where device's setting is set to English (or any other language different than Turkish).

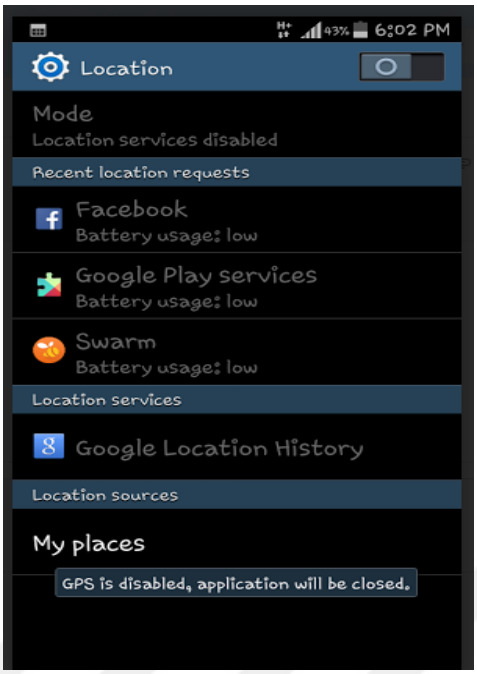

Figure 5.19 English Version

5.8.OTHER COMPONENTS

In this part other components will be explained developed by Ahmet Karaman, which are used in our application. They will be explained shortly and for further information you can look into (Karaman, 2015).

5.8.1. Server

For keeping POI's and customer's values a server and a database were needed. This server has 194.27.192.112 IP value and it is provided by Galatasaray University. Necessary permissions are given to port 80 for accessing via web services. A remote user can access to this server. It has five predefined and password secured user.

The main database system we used is MySQL. A schema called SARAS is created on database. Data model for this schema can be seen in Figure 5.20.

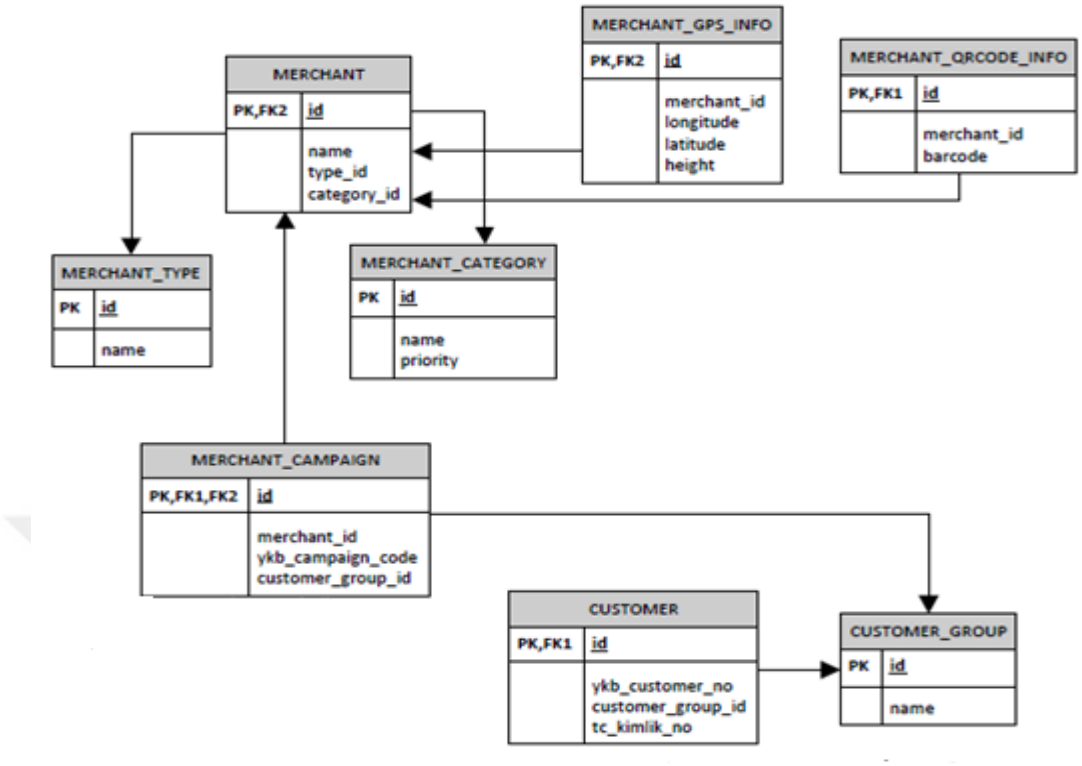

Figure 5.20 SARAS Data Model

5.8.2. Web Services

I already explained that we access to a remote database for POI's and customer's information. After fetching necessary values we keep them in our local database or in our local storage.

This web services are created using a framework called XAMPP. Restful services are chosen because they are faster and more flexible. These restful web services are:

http://194.27.192.112/Services/getTypes.php: Service used to get POI types
http://194.27.192.112/Services/getCategories.php: Service used to get POI categories. These 2 services are unified on Java codes (client side).
http://194.27.192.112/Services/getMerchants.php: Service to get the entire POI on a specified range of distance.

http://194.27.192.112/Services/getCampaigns.php: Service to get campaign information.
http://194.27.192.112/Services/customerLogin.php: Service to control if user is a bank customer or not.

5.8.3. QR Code Screen

Our application should also offer QR Code sticker analyze option to show merchant's campaign information. For this purpose, we used ZXing library. ZXing is an open-source, multi-format 1D/2D barcode image processing library implemented in Java. QRCode Screen can be seen in Figure 5.21.

Figure 5.21 Reading QR Code

If user is in a closed environment with no GPS connection, he can use this feature for obtaining campaign information.

6. SCREEN POSITIONING ALGORITHM

The main purpose of this thesis is to develop an algorithm for positioning POIs at the right point on mobile device's screen. This algorithm should be responsive to user's motions, efficient, and it should be as accurate as it could be.

In this chapter different algorithms and tools that have been encountered during our research is represented. Basically, two different approaches have been used: One from the related literature, the other one is the proposed approach.

6.1.AUGMENTED REALITY SDK

There are several tools which offer us a platform for developing AR applications on Android. Some tools are quite simple and does not require special programming skills and they allow to define POIs (Wikitude and Layar). When these POIs are detected, the user can select them and get more information about them or even perform actions on them. Several other tools, such as ARLab SDK, Mixare require some programming skills and have been developed for more serious AR developers.

Using these SDKs you can add and display POI into your application. These frameworks take care of all the complex functionalities of the AR browser. All the optimizations for energy consumptions has been tested and maintained. You can use these SDKs for developing a location-based AR application. They will do all work such as spotting predefined POIs and reducing battery consumptions and you can focus on designing your application. In our research it was essential to create a new algorithm for tagging POIs with optimum battery consumption, so we did not use one of these SDKs

.

6.2.WORLDPLUS

This algorithm that is found during our literature search (WorldPlus:An Augmented Reality Application with Georeferenced content for smartphones -the Android example by Sérgio Graça, João Fradinho Oliveira and Valentim Realinho) (Graça et al.,2012). It is developed for an Android AR application called WorldPlus. It has an objective similar to ours, to place POIs on mobile device's screen.

In this project, the projection is divided in two different calculations: the horizontal projection, *X* axis, and the vertical projection, *Y* screen coordinate.

The horizontal projection is based on two similar but different values: the azimuth that is provided by the magnetic sensor, and the bearing that is the angle formed between the true North and a given point on Earth that the device is facing. The bearing is calculated in the same way as azimuth but it requires an additional point. In the projection if the azimuth and bearing to a point is the same or very close it means that the device is facing in the direction of that point. In Figure 6.1 these values are represented.

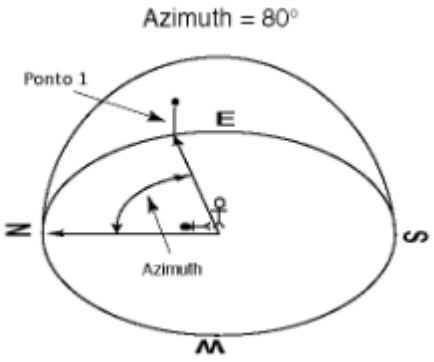

Figure 6.1 Calculating Horizontal Projection (Graça et al.,2012)

In the case that the bearing to a point is greater than the azimuth then the point is in the right half of the screen, or else, if the bearing is less than the azimuth the point is in the left half of the screen. With this they could calculate a horizontal screen coordinate. After calculating this horizontal screen coordinate for the point, they calculate the

vertical coordinate for it, the vertical projection, which consists in calculating the Y coordinate that the point will have on the device's display. Like in the horizontal projection, the vertical angle of view is defined with an angle of 45 degrees.

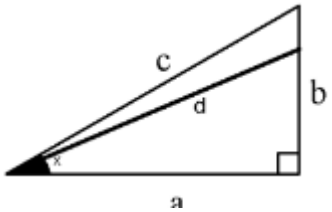

Figure 6.2 Calculating Vertical Projection (Graça et al.,2012)

In Figure 6.2 the position of a point is defined by a line *d* that also represent the angle formed between the position of the user and the position of the point of interest. This is the angle to determine if the point of interest is in the field of view or not, if it will be drawn in the device's display or not.

As it can be seen in the Figure 6.2 they have the distance between the two points (*a*) (for our case between the POI and the user's device), they also have the height of the point in relation to the height of the user position (*b*) (for our case it will be difference of altitudes between POI and user), also the angle formed between *a* and *b* is a 90 degree angle. So using the Law of Cosines they can calculate the angle formed between the user position and the point of interest position.

Finally, and similar to the horizontal projection, they compare these angles, the angle *y* (explained above) and the angle that restricts the vertical field of view represented as *c*, if the angle *y* is lesser than the boundary angle it means that the point is in the angle of view, if it is greater it means that the point is outside the angle of view and it will not be drawn in the device's display.

In this way they determine if the point is inside the field of view and calculate the exact screen coordinate.

6.3.AREA

AREA (Geiger et al.,2013) platform has also a similar objective to ours. The overall purpose of this work is to outline the engineering process of a sophisticated mobile service running on a smartphone. More precisely, we show how to develop the core of a location-based augmented reality engine for the iPhone 4S based on the operating system iOS 5.1 (or higher. he augmented reality engine

AREA shall basically show points of interest (POIs) inside the camera view relative to the current position of the user and the POIs. On the screen, the POIs shall be only displayed if they are inside a visible view of the user, particularly inside the field of view of the device's built-in camera. This is represented on Figure 6.3.

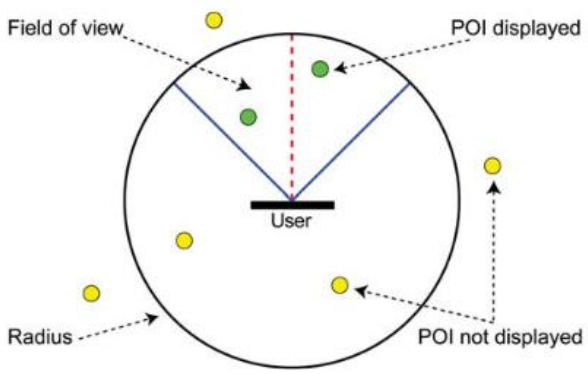

Figure 6.3 Illustration for Visible POIs (Geiger et al.,2013)

On developing their algorithm on screen positioning they used basically the same path to WorldPlus. But they carried forward some calculations such as angle of view and the distance between the device and POI. Their algorithm consists of four steps:

1. **Calculating the Distance**

First, the distance between user and POI location is calculated.

2. Calculating the Bearing

Only POIs being inside the visible field of view shall be displayed on the camera view. Hence, just like WorldPlus, the bearing between user and POI's positions relative to the North Pole is calculated.

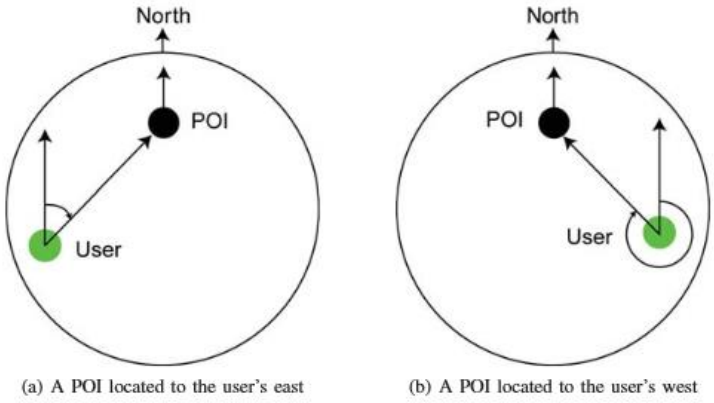

Figure 6.4 Representation of Calculated bearing (Geiger et al.,2013)

They did not used android's bearingTo function but a different formula for calculating the bearing. As it is represented on Figure 6.4 Using this result, it becomes possible, in combination with the smart phone's compass, to determine whether a POI is inside the horizontal field of view and where it must be drawn on the screen

3. Calculating the Elevation Angle

The visible field of view of the smartphone's is not only limited in its width, but also in its height. Therefore, the altitude difference between the user and POI is calculated to determine whether or not a particular POI is inside the vertical field of view. Thus, they used pitch value of the smartphone to determine what area shall be visible on the display.

4. Calculating the Field of View

In WorldPlus, for the angle of view, they defined an angle of 45 degrees without any calculations. But in AREA they calculate this angle of view using the image size, more

precisely to the size of the image sensor, and the focal length of the camera lens of the device. As a consequence of their calculations iPhone has a horizontal field of view of 56 degrees and a vertical field of view of 44 degrees when used in landscape mode.

Now as the size of the field of view of the iPhone camera is known, by additionally using other results it becomes possible to determine whether a POI is inside the vertical and horizontal field of view, and in what distance a POI is located to the user

So as these two examples use bearing, first fully we tried to implement this algorithm into our project using Android's Location library's bearingTo function. But as the outputs were complex to interpret and as we have to develop a new algorithm we decided to develop another which will be easier and more comprehensible.

6.4. LATITUDES AND LONGITUDES SCREEN POSITIONING ALGORITHM (LLSP)

This is our own algorithm that we developed for our thesis. This algorithm is developed using latitudes and longitudes difference between user's device and the point of interest to be positioned on screen. Similar to the algorithm explained above we calculate horizontal and vertical projection. This algorithm is developed for the countries in north hemisphere. For other countries we should add additional conditions.

6.4.1. Finding POI's Direction

LLSP algorithm for finding and interpreting POI's direction is as follows: The LLSP algorithm uses all the possible directions (North, South, East, West, Northeast, Northwest, Southeast, and Southwest). It considers that there is 360 points that the device can point at, because a circle has 360 degrees. As there are eight different directions, 360 are divided to 8 and the device is said to have a vision range of 45 points. Then, the algorithm tries to figure out if the POI is in the given interval of 45

points or not. If it is in the interval, then this POI needs to be positioned on screen. Otherwise, this POI should not be listed on screen.

The steps of the algorithm can be summarized as follows:

1. Image buttons are created on screen for each POI in chosen type and distance.

2. All of the POIs are made invisible. (All of them are created, because it is faster than recreating them on every move of the device.)

3. The difference between POI's and device's latitude is calculated as *diffLatitude* and the difference between POI's and device's longitude is calculated as *diffLongitude*.

4. These differences are compared and it is defined if the POI stays in North or South (if *diffLatitude* > *diffLongitude*) or it is in East or West (*diffLatitude* <= *diffLongitude*).

5. *i.*If (*diffLatitude* > *diffLongitude*), then we should calculate if it is in North or in South using the rule: If (POI's latitude > device's latitude), then POI should be in North, because we are in the North hemisphere and the latitude increase as we go through North; otherwise it should be in South.

   ii. If (*diffLongitude* >= *diffLatitude*), then we should calculate if it is in East or West using the rule: If (POI's longitude > device's longitude), then POI should be in East, because the longitude increase as we go through East; otherwise it should be in West. Therefore, the direction of the POI can be calculated.

This process is schematized on Figure 6.5.

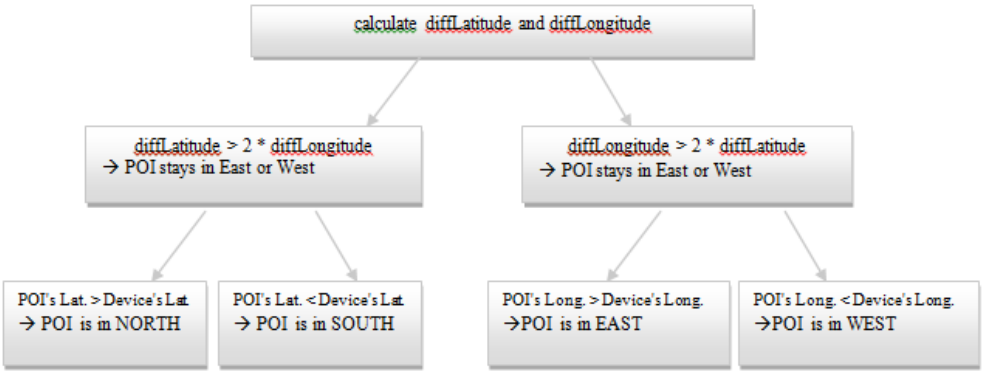

Figure 6.5 Finding POI's Direction

6. The POI's location is also calculated.

   The world is divided into 360 points. Using the azimuth range for directions, these 360 points are divided into the following cardinal and inter cardinal directions:

   North: Interval is 157.5 – 202.5

   Northeast: Interval is 202.5 247.5

   Northwest: Interval is 112.5- 157.5

   South: Intervals are 337.5 - 360 and 0 - 22.5

   Southeast: Interval is 292.5 337.5

   Southwest: Interval is 22.5 67.5

   East: Interval is 247.5- 292.5

   West: Interval is 67.5 - 112.5

Figure 6.6 represents these values in a circular way and the user is located at the center.

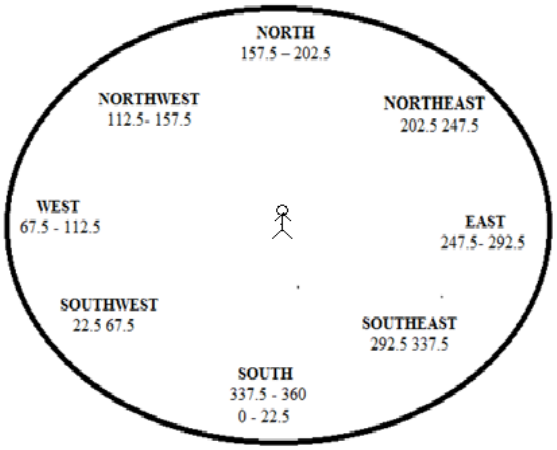

Figure 6.6 Directions Representation

i. If POI is in **North**: Interval is 157.5 – 202.5

1. The middle of North is 180 ((157.5 +202.5) /2). Looking to the North direction, if the POI is in East (POI's longitude > device's longitude), then the POI should be on the right side of screen. So, this point should be between 180 and 202.5. Otherwise, if it is in West, it should be between 157.5 and 180.
2. So, it has a range of 22.5. 22.5 is divided to the *diffLatitude*, and it is multiplied with *diffLongitude*.
3. If the POI is in East, 180 is added to the result (or if it is in West, 180 is subtracted from the result) of step 2. Finally, the POI's point is generated.

ii. If the POI is in **South**: Intervals are 337.5 - 360 and 22.5 – 0.

1. The middle of South is 0 or 360, as it has two intervals. Looking in the South direction, if the POI is in East (POI's longitude > device's longitude), then the POI should be on the left side of the screen. Thus,

this point should be between 337.5 and 360. Otherwise, it should be between 22.5 and 0.

2. It has a range of 22.5. Therefore, 22.5 are divided to the *diffLatitude*, and the result is multiplied with the *diffLongitude*.
3. If the POI is in East, the result of step 2 is subtracted from 360 (or if it is in West, the result of step 2 is added to 0). Finally, the POI's point is generated.

iii. If the POI is in **East**: Interval is 247.5- 292.5.

1. The middle of East is 270. Looking in East direction, if the POI is in North (POI's latitude > device's latitude), then the POI should be on the left side of the screen. So, this point should be between 247.5 and 270. Otherwise, it should be between 247.5 and 270.
2. Thus, it has a range of 22.5. Therefore, 22.5 is divided to *diffLongitude*, and this is multiplied with *diffLatitude*.
3. If the POI is in North, the result of step 2 is subtracted from 270 (or if it is in South, the result of step 2 is added to 270). Finally, the POI's point is generated.

iv. If the POI is in **West**: Interval is 67.5 - 112.5.

1. The middle of West is 90. Looking in West direction, if the POI stays in North (POI's latitude > device's latitude), then the POI should be on the right side of the screen. Thus, this point should be between 90 and 112.5. Otherwise, it should be between 67.5 and 90.
2. It has a range of 22.5. Therefore, 22.5 is divided to the *diffLongitude*, and the result is multiplied with *diffLatitude*.
3. If the POI is in North, the result of step 2 is added to 90 (or if it is in South, the result is subtracted from 90). Finally, the POI's point is generated.

The inter cardinal directions such as Northeast, Northwest, Southeast and Southwest are also considered in the algorithm. If the *diffLatitude* is two times bigger than *diffLongitude* or vice versa, the POI is assumed to be in North, West, East or South. Otherwise, it is assumed to be in Northeast, Northwest, Southeast or Southwest.

7. Screen range is also calculated using the mobile device's azimuth value. This value is assumed to be the middle of the screen, 22.5 is subtracted from this value for finding the beginning point of screen. Next, 22.5 is added to this value for calculating the last point of screen. Therefore, the screen has a range of [azimuth-22.5 – azimuth+ 22.5].

8. If the device is pointed to the POI's direction, the calculated interval at step 7 is compared with the point calculated at step 6. If the POI's point is inside this interval this point becomes visible.

This flow is schematized on Figure 6.7.

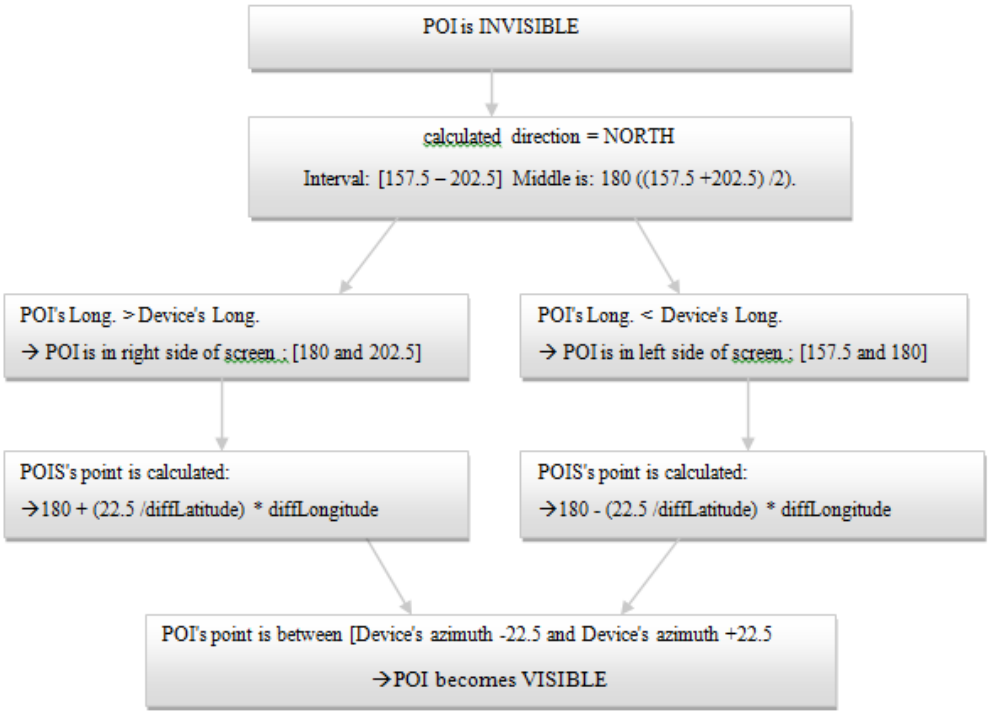

Figure 6.7 Calculating POI's Location Point

As the next step, the POI's point needs to be calculated on mobile device's X axis.

6.4.2. Calculating the point on device's X axis:

9. Device's azimuth value is interpreted as the middle of the screen (same as the step 7).
10. We calculate the azimuth value of the first/last point of the screen (it can be calculated with the following formula: device's calculated azimuth value - + 22.5, since each interval has 45 points) (same as the step 7).
11. The first value calculated in the step 10 is subtracted from the POI's point that is calculated at the step 6.
12. The width of the screen is determined using Android's getWidth() function. Then, it is divided to 45 (the screen range). Therefore, the portion per point is calculated. So as the screen is considered having 45 points, every point should have a range of this calculated portion.
    Portion * 45 = device's screen width.
13. The value that is calculated on the step 11 is multiplied with the one that is calculated on the step 12.
14. The button's size/2 is subtracted from the value that is calculated at the step 13.

Finally, the POI's point on X axis is found.

These steps are schematized in Figure 6.8.

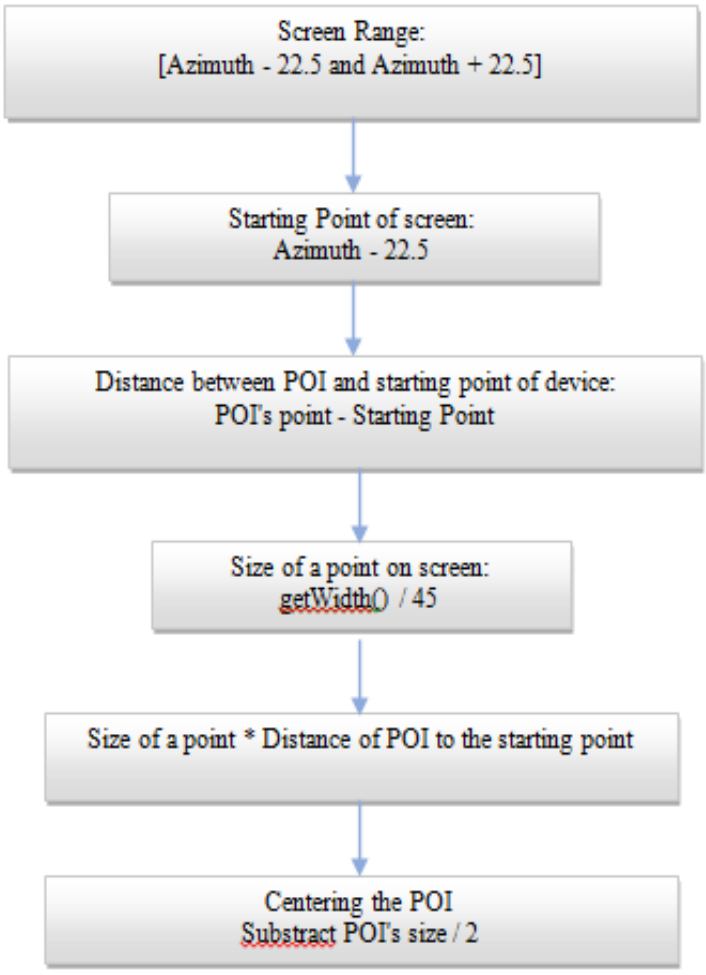

Figure 6.8 Calculating POI's point on X axis of Device

For example:

- Device's azimuth value is 120. And *Xmerchant* is 135 and *sizeMerchant* is 20.

(In this example steps are related to algorithm steps previously explained in "Calculating the point on devices X axis")

Step 9: 120 is interpreted as the middle of the screen.

Step 10: 120 - 22. 5 = 97.5. This is the first point that is visible on the screen.

Step 10: 120 + 22.5 = 142.5. This is the last point that is visible on the screen.

The merchant's location on screen is calculated as following:

Step 6: let’s say that the merchant’s point on real world is calculated as 135.

Step 8: 135 is between 97.5 and 142.5 so it is on range of the screen. So this button should be visible.

Step 11: Subtract merchant’s point from the first point calculated on step 10: 135 - 97.5 = 37.5

Step 12: getWidth() is 480. (480/45 = 10.6)

Step 13: 10.6 * 37.5 = 400.

Step 14: 20 / 2 = 10. 400 - 10 = 390.

Thus, the merchant is at the point 390. Note that the screen range value is between 0 – 480, and this value may be generated using the function getWidth().

6.4.3. Calculating the point on device's Y axis:

For calculating POI’s position on Y axis I use the pitch value. If the device is directed up or down to the floor, nothing is listed on the screen.

Otherwise;

1. Screen is considered having 90 points from up to down.
2. Screen's height is divided to 90 and we obtain value of a portion per point.
3. We divide screen's height to 2 for obtaining the value of the middle of the screen.
4. We multiply pitch value with the value calculated on step 2.
5. We subtract step 3's value from step 4.
6. We subtract button's size divided by 2 from step 5's value.

We did ignore the altitude differences between the user's device and POI as we did not have the altitude values of merchants of Yapı Kredi. The vertical projection calculation of Worldplus (Graça et al, 2012) would be added into our application.

6.4.4. Handling Intersections on the Screen

If multiple POIs are placed at the same point of the screen, they should be repositioned in order to remove these visual intersections. POIs, which intersect are listed as follows on screen as in Figure 6.9.

Figure 6.9 Intersections on Screen

As it can be seen buttons intersects with each other, so the one which stays under cannot be seen. So, I had to handle these intersections using following algorithm:

1. All the available intervals are collected into a hash map.
2. A suitable interval is found for the given POI.
3. After placing a POI, this interval is changed by omitting this POI's occupied place. If it is in the middle of an interval, we obtain two intervals.

For Example:

At the beginning we have [XStartInterval and XEndInterval] and the merchant is placed in Xmerchant with the size of SizeMerchant.

- **If (*XStartInterval* < *Xmerchant*) and (*Xmerchant* + *SizeMerchant* < *XEndInterval*),** the new interval becomes:

  [XStartInterval – Xmerchant] – [Xmerchant + SizeMerchant – XendInterval]

- **If (*XStartInterval* > *Xmerchant*) and (*Xmerchant* + *SizeMerchant* > *XStartInterval*),** the new interval becomes:

  [(Xmerchant + SizeMerchant) – XendInterval]

- **If (*Xmerchant* < *XEndInterval*) and (*Xmerchant* + *SizeMerchant* > *XEndInterval*), t**he new interval becomes:

  [*XStartInterval* - *Xmerchant*]

  The same rules are applied to axis Y.

4. This interval (calculated on step 2) is removed from the hash map. New intervals that are calculated in step 3 are added to the hash map.
5. If the merchant's calculated position is occupied, the nearest interval is chosen.

Therefore, the merchants can be listed on the screen as in Figure 6.10.

Figure 6.10 Handling Intersection

Another Example:

Devices's: Screen width = 480 and Screen height = 800

POI1's calculated X= 40 and Y= 50 and Size= 20

At the begining our intervals for empty places are:

xStart(0) = 0 xEnd(0) = 480, yStart(0) =0 yEnd(0) = 800 → Interval: **[0-480] [0-800]**

We place POI1 on screen and we reform our intervals in hashmap. Finally we obtain these 4 new available intervals.

xStart(0) = 0 xEnd(0) = 40, yStart(0) = 0 yEnd(0) = 800→ Interval: **[0-40] [0-800]**

xStart(1) = 60 xEnd(1) = 480, yStart(1) = 0 yEnd(1) = 800→Interval: **[60-480] [0-800]**

xStart(2) = 40 xEnd(2) = 60, yStart(2) = 0 yEnd(2) = 50 → Interval: **[40-60] [0-50]**

xStart(3) = 40 xEnd(3) = 60, yStart(3) = 70 yEnd(3) = 800→ Interval: **[40-60] [70-800]**

7. THE ANALYSIS OF THE PROPOSED ALGORITHM PERFORMANCE

7.1. Comparison with Other Similar Applications

In this part, I will compare our application with other applications that I mentioned in Chapter 3: Similar Applications, in terms of several performance metrics. The chosen applications are a few of the most frequently used augmented reality applications developed by big software companies. Our primary aim is to keep up their performance level, and then go beyond it. We focused on creating our own application, having different features our own algorithm for screen positioning.

In general, our thesis has following advantages versus these other applications:

- Chapter 5.2. GPS libraries: None of these applications mentioned in Chapter 3 offers user to choose a library for fetching GPS information. This is important for energy consumption.
- Chapter 5.2.1. Map Support: Generally this kind of applications offer either map support or screen positioning support, but our application has both of them.
- Chapter 5.2.3. GPS Modes: None of these applications offers user to choose his own mode for getting GPS information. This reduces energy consumption.
- Chapter 5.6. Sensor Calibration: Sensors can produce faulty data from time to time, so we added a feature for calibrating sensors. This is specific to our application.
- Chapter 5.6. Compass: A compass is added on the main screen.

- Chapter 5.7. Language Support: None of these applications offers user the Turkish language selection. Our application can be used both in Turkish and in English.

Now, the proposed LLSP algorithm will be compared with another screen positioning algorithm that we found during our literature search (Graça et al.,2012). This application implemented on Pro Android Augmented Reality (Raghav, 2012) is an augmented reality world browser that shows data from Wikipedia and Twitter all around.

Normally, the given application has following features:

- It has a live camera preview.
- Twitter posts and topics of Wikipedia articles that are located nearby are displayed over this preview.
- There is a small radar visible that allows the user to see whether any other overlays are available outside their field of view.
- Overlays are moved in and out of the view as the user moves and rotates.
- The user can set the radius of data collection from 0m to 100.000 m (100 km).

With these properties, this is a similar application to ours. Therefore, in order to test the performance of our algorithm, we implemented the given algorithm (Raghav, 2012) into our application.

7.1.1. Energy

First, we focus on the energy efficiency of the algorithms. The results show that our proposed LLSP algorithm spends less energy than the given algorithm (Karaman, 2015). On Table 7.1 our test results with different modes of GPS and with different libraries are shown. As the first step, I compare the results of the best methods on GPS calculation and internet connection. Best results are obtained in normal mode with

Google Play Services library for GPS calculation and using WIFI for internet connection.

SARAS with Google Play Services library using WIFI consumes (Karaman, 2015):

- 978 mw of energy
- %13 of battery
- 896 j of LCD (this depends on how many POI is displayed on screen)

on 20 minutes of uninterrupted running.

Pro Android Augmented Reality using WIFI consumes (Karaman, 2015):

- 978 mw of energy
- %14 of battery
- 892 j of LCD (this depends on how many POI is displayed on screen)

on 20 minutes of uninterrupted running.

The results reveal that our proposed LLSP algorithm is more energy efficient than the given algorithm (Karaman, 2015). It consumes %1 less battery in 20 minutes.

7.1.2. CPU Usage

Secondly, I will compare CPU usage of these two algorithms.

SARAS with Google Play Services library using WIFI consumes (Karaman, 2015):

- 267 J of CPU

on 20 minutes of uninterrupted running.

Pro Android Augmented Reality using WIFI consumes (Karaman, 2015):

- 265 J of CPU

on 20 minutes of uninterrupted running.

The test results reveal that our proposed LLSP algorithm uses almost the same amount of CPU power than the given algorithm.

We can summarize differences on source consumptions as follows:

	SARAS	Pro Android Augmented Reality
Energy	**978 mw**	**978 mw**
Battery	**%13**	**%14**
LCD*	**896 J**	**892 J**
PU	**267 J**	**265 J**

***** this depends on how many POI is displayed on screen

Table 7.1 Resource Utilizations

7.1.3. Screen Positioning Comparisons

- **Standing Still**

As the third step, we test the consistency of the screen positioning. For test purposes, we set the maximum distance for map and for distance filter to 10 km.

For the tests, device stand still and the device's screen is turned for looking around. Figure 7.1 shows the situation of the device and the POI's during the test. There are 11 POIs on the range. Device is in the center of the circle on the intersection of direction divisions. They are depicted in East and North-East, when the mobile device is thought in the middle.

Figure 7.1 Map Representation

POIs which stay in **East** are:

- KasımPaşa Şubesi
- Kasımpaşa Şb Plus-1
- Kasımpaşa Şb Plus-2
- Perşembepazarı Şubesi
- Perşembepazarı Şb Plus

POIs which stay in **North-East** are:

- Atılım Şubesi
- Rahmi Koç Müzesi
- Okmeydanı Bedaş
- Okmeydanı Şb Plus-1
- Okmeydanı Şb Plus-2
- OkMeydanı Şubesi

- **East**

Using LLSP algorithm POIs in East are listed as in Figure 7.2. Using referenced book's algorithm (Raghav, 2012) they are listed as in Figure 7.3.

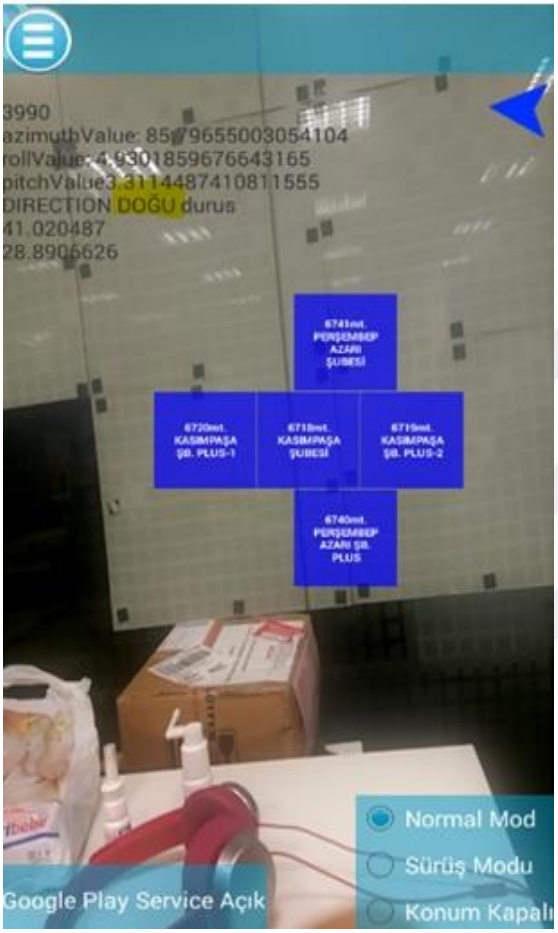

Figure 7.2 Screenshot using our proposed LLSP algorithm - East

Figure 7.3 Screenshot using the given algorithm - East (Raghav, 2012)

As it can be see from the Figure 7.2, all of the five POIs in East are placed correctly on the screen.

But the algorithm of our referenced book (Raghav, 2012) listed all the POIs in East, even though several of them were in North-East.

- **Passage from right (East) to left (North-East)**

Using LLSP algorithm POIs are listed as in Figure 7.4. Using (Raghav, 2012) algorithm they are listed as in Figure 7.5.

Figure 7.4 Screenshot using our proposed LLSP algorithm - Passage from East to North-East

Figure 7.5 Screenshot using the given algorithm (Raghav, 2012) - Passage from East to North-East

Using LLSP algorithm all of the POIs are listed on screen because it is like an intersection for all of them. Moreover the POIs who are in East stays in the right side of the screen and the POIs who stay in North-East are placed in the left side on the screen(As it is a passage from right (East) to left (North-East)).

Given (Raghav, 2012) algorithm showed the same list as in East. It listed all of the POIs.

- **North-East**

Using LLSP algorithm POIs are listed as in Figure 7.6. And using (Raghav, 2012) algorithm they are listed as in Figure 7.7.

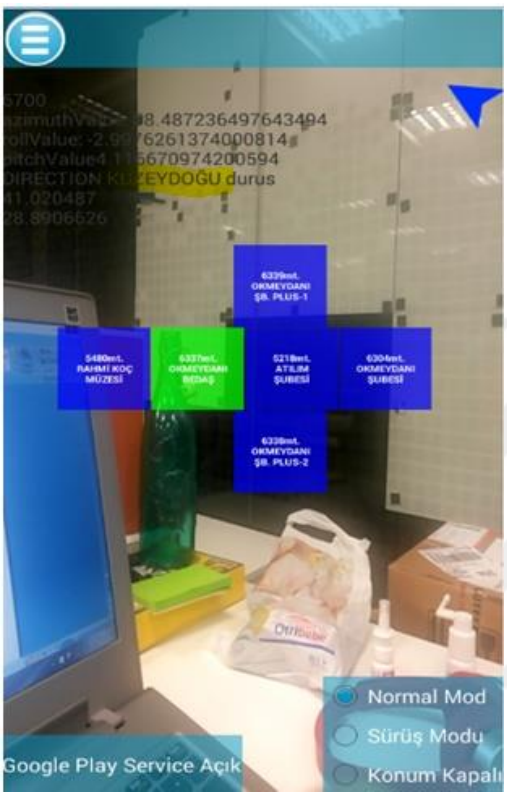

Figure 7.6 Screenshot using our proposed LLSP algorithm North-East

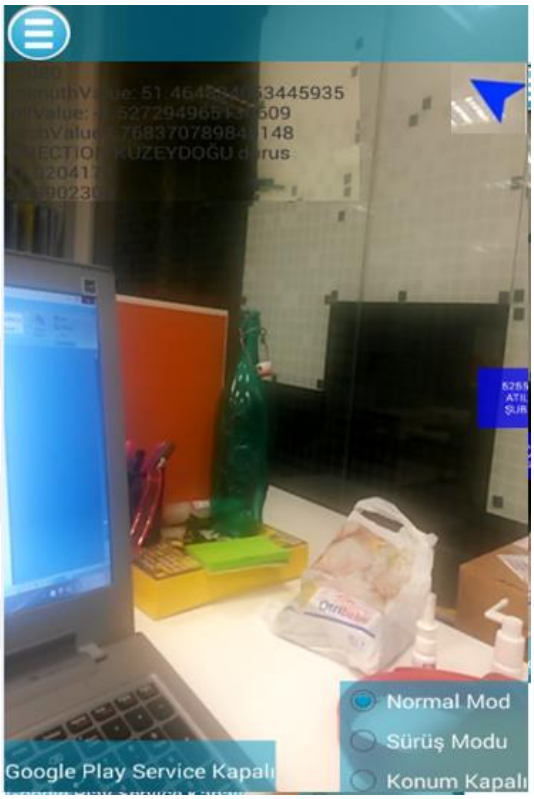

Figure 7.7 Screenshot using the given algorithm (Raghav, 2012) - North-East

Using LLSP algorithm all the 6 POIs in North-East are also listed properly. (Raghav, 2012) listed only one of the POIs which are on North-East. It missed four others. Finally we can conclude that our proposed LLSP algorithm produces more reliable results and more accurate screen positions compared to (Raghav, 2012).

- **Walking**

The same tests are generated for the walking case. The outputs are analyzed while walking. The outputs were reliable and fast. The application was reacting quickly and the outputs were as same as the test in standing still.

- **Driving**

These two algorithms are also compared while driving. The point of interest to show on screen is Çağlayan Adalet Sarayı. LLSP algorithm's outputs can be seen on Figure 7.8.

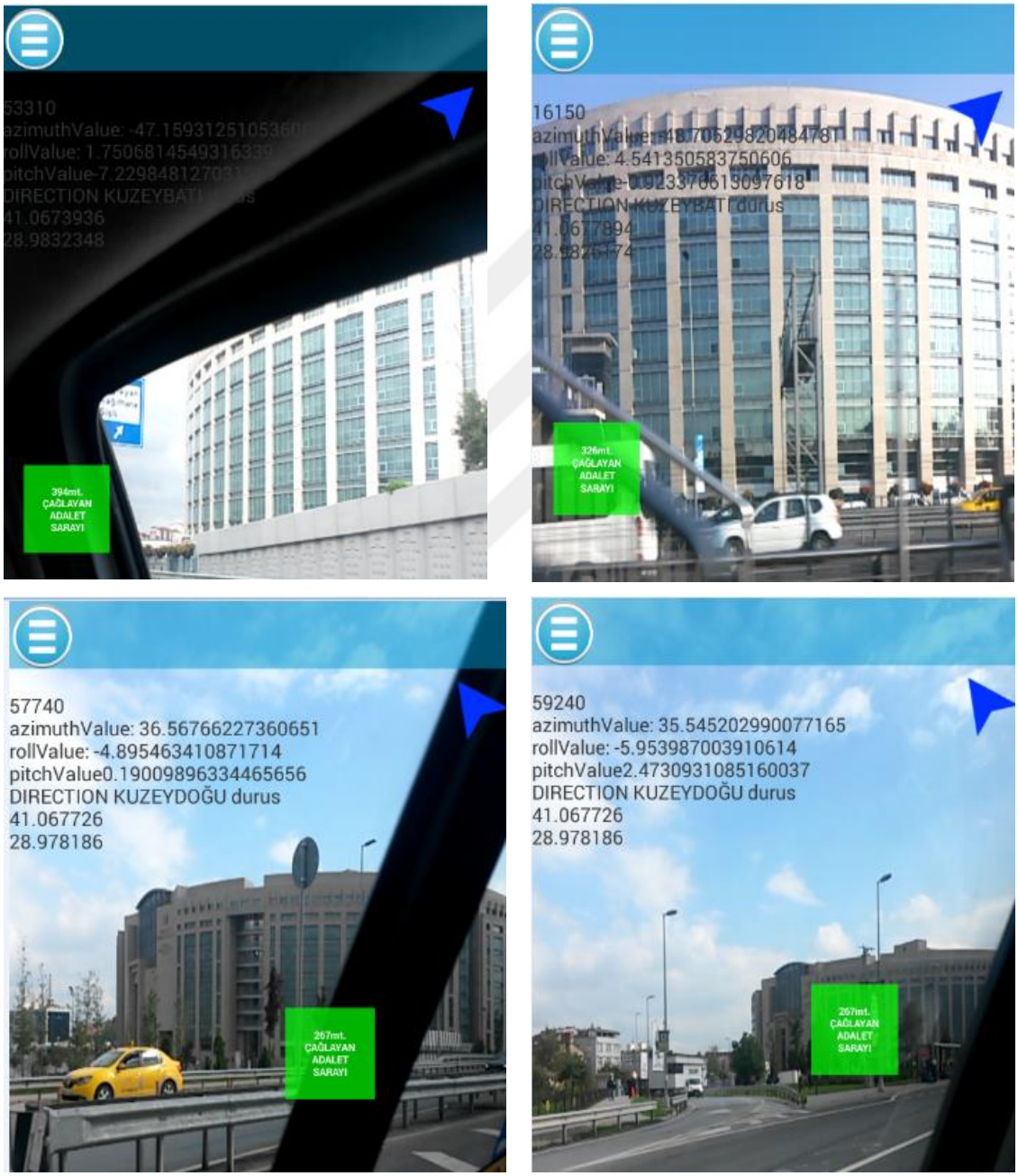

Figure 7.8 Screenshots using our proposed LLSP algorithm while driving

The same building using the proposed algorithm is shown in Figure 7.9.

Figure 7.9 Screenshots using the given algorithm while driving (Raghav, 2012)

Comparing these outputs we can conclude that both of these algorithms have similar outputs. Both of them placed Çağlayan Adalet Sarayı at the correct position.

8. CONCLUSION

The aim of this research is to develop an algorithm for tagging POIs on mobile device's screen. An AR mobile application (called SARAS) is built. SARAS is compatible with Android OS. One of the biggest banks in Turkey (Yapı Kredi Bank-YKB) and Ministry of Industry and Sciences have supported this research. The POIs are chosen as the merchants, offices and the ATMs of the bank. Using this application, users can find the nearest bank and ATMs, along with their way towards them. Besides, bank may use this application as a new channel to inform their potential users on related campaigns.

Since only the benchmarking algorithm (Raghav, 2012) has an open source algorithm, the performance comparisons are done using it. Both algorithms spend almost the same amount of energy. The biggest sources of energy consumption of AR applications are the GPS and camera usage. LCD usage depends on the number of POIs that are displayed on the screen, therefore it can be concluded that both algorithms spend almost the same amount. The CPU usage of these two algorithms is at similar ranges, but the proposed algorithm consumes less battery.

The proposed algorithm places accurate tags while standing still and walking. When driving, it places the tags at accurate locations on the screen in each time, however its response time to position changes need to be improved.

The most significant contribution of this algorithm is its simplicity. It is developed for the countries in North hemisphere, but it is easy to convert it for the South hemisphere. Going further, the performance of the algorithm will be improved to more rapidly react to position changes while driving.

9. THREATS TO VALIDITY AND FUTURE WORK

Going further, the screen positioning algorithm should be improved. During our performance tests, I figured out that it does not give satisfying result, while user is moving rapidly. Therefore, LLSP algorithm should produce better, reliable and faster outputs. Moreover, the direction recognition in sensor fusion algorithm was slow. So, the outputs were coming with latency. This results delay in the screen positioning. We should work on this algorithm for faster information and faster and reliable tagging. We have some shaking, tilting problems on POI projections caused by sensor sensitivity. Therefore, we should implement a better low pass filter to our algorithm.

In our algorithm we did not take into consideration device's and POI's altitude value and device's height limitations. We should also enrich our algorithm by calculating the field of view based on device's height. As an additional feature, a radar could be used to indicate the direction in which invisible POIs are located.

Some screen warnings, such as closing the application if device has not enough battery, are inserted to the application. We also added different modes for fetching GPS values and the possibility to use two different libraries for GPS. However, we had also to search for reducing the battery consumption of the camera and internet connection. But we did not make enough progress in it. We only avoided this consumption by closing the application and by warning user. Therefore, we should make more searches on reducing this energy consumption caused by the camera and internet.

We should unify queue-matic system on bank offices and our application. By using this feature, customers can get a queue number without actually going to bank office. As the bank security policies did not authorize this, we could not integrate our application into the bank's system.

REFERENCES

Android Camera (2015).

URL:*http://www.tutorialspoint.com/android/android_camera.htm.*Google Googles

Android Motion Sensors (2014).

URL:*http://developer.android.com/guide/topics/sensors/sensors_motion.html.*

Android Position Sensors (2014). Android Developers.

URL:*http://developer.android.com/guide/topics/sensors/sensors_position.html.*

Azimuth Pitch Roll Image.

URL:*http://www.mathworks.com/matlabcentral/mlc-downloads/downloads/ submissions/40876/versions/8/screenshot.jpg.*

Android Sensor Manager (2014). Android Developers.

URL:*http://developer.android.com/reference/android/hardware/SensorManager. html.*

Compass Application (2014).

URL:https://play.google.com/store/apps/details?id=com.gn.android.compass.

Geiger, P., Pryss, R., Schickler, M., and Reichert, M. (2013). Engineering an advanced location-based augmented reality engine for smart mobile de- vices. *Technical Report UIB-2013-09*, University of Ulm, Germany.

Google Goggles (2014).

URL:*https://play.google.com/store/apps/details?id=com.google.android.apps. unveil&hl=tr*.

Graça, S., Oliveira, J.F., Realinho, V. (2012) "WorldPlus: An Augmented Reality Application with Georeferenced Content for Smartphones - the Android Example", Proc:. *20th WSCG International Confrence on Computer Graphics and Computer Vision.*

Karaman, Ahmet **(**2015). Analysis Of Energy Efficiency of GPS-Based Augmented Reality Application.*Galatasaray University*

Lawitzki, Paul (2014). Android Sensor Fusion Tutorial.

URL:*http://www.codeproject.com/Articles/729759/Android-Sensor-Fusion-Tutorial.*

Layar (2009)**.** Layar HomePage.

URL: *http://www.layar.com/.*

Shane Conder, Lauren Darcey (2011). *Augmented Reality: Getting Started on Android.*

URL: *http://code.tutsplus.com/tutorials/augmented-reality-getting-started-onandroid -- mobile-4457.*

Sood, Raghav (2012). Pro Android Augmented Reality.

Tamada, Ravi (2013)**.** Android working with Google Maps V2.

URL:*http://www.androidhive.info/2013/08/android-working-with-google-maps- /.*

Tuscany+ (2010)**.** Tuscany+ Home Page.

URL: *http://www.turismo.intoscana.it.*

Tuscany+ Image (2010)**.** Tuscany+ Home Page..

URL:*http://www.turismo.intoscana.it/allthingstuscany/aroundtuscany/files/2010/05/30644_1123958354547_1694785481_213647_377230_n.jpg.*

Wikipedia (2015)**.** Augmented Reality.

URL: *https://en.wikipedia.org/wiki/Augmented_reality.*

Wikitude (2008)**.** Wikitude Home Page.

URL: https://www.wikitude.com/.

Wikitude Image (2008)**.** Wikimedia.

URL:*https://commons.wikimedia.org/wiki/File: Wikitude_World_Browser_ @ Salzburg_Old_ Town_2.jpg.*

Yuen, S., YaoYuneyong, G., Johnson, E. (2011). An overview and Five Directions for AR in Education. *Journal of Educational Technology and Development and Exchange, 4(1), 119-140*

BIOGRAPHICAL SKETCH

The author of this thesis was born in 1986 in Erzincan, Turkey. She has studied in Galatasaray High School between 1998 and 2005, and started her undergraduate education in the Computer Science and Engineering Department of Galatasaray University in 2005-2010 terms. Consequent to the graduation from the undergraduate degree, in 2010 she has enrolled to the Computer Engineering Master's Degree in Université Paul Sabatier (Toulouse) University Institute of Sciences. After that she has continued at the Galatasaray University for finishing her master. Since 2011 she has worked at research, banking and telecom domains in several workplaces.

PUBLICATIONS

- Paper titled "Design Of Sensor-Based Augmented Reality Software (SARAS)" of this thesis has been presented in 23rd IEEE Signal Processing and Communications Applications (SIU 2015).