Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Augmented Reality User Interface Analysis in Mobile Devices

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: José Daniel Guzmán G.

DIRECTOR: Dolors Royo Vallés

DATE: May 08th 2014

Title: Augmented Reality User Interface Analysis in Mobile Devices

Author: José Daniel Guzmán G.

Director: Dolors Royo Vallés

Date: May 08th 2014

Overview

The presence of high-end phones in the telephony market, has allowed consumers to have access to the computational power of mobile smart-phone devices. Powerful processors, combined with cameras and ease of development encourage an increasing number of Augmented Reality (AR) researchers to adopt mobile smart-phones as AR platform.

The same way, Augmented Reality on mobile devices has become increasingly popular for many applications, including search and location, tourism, and shopping. An interesting development platform for implementing an AR application is Unity 3D. Unity 3D has a powerful library that provides access to the integrated sensors in mobile devices.

Implementation of an AR application that utilizes data from sensors of mobile phone device is the primary basis for this project. In late 2011, the API of Google maps stopped being free, this fact has prompted many developers looking for an API with the same features besides free, this option is OpenStreetMap, and the main objective of this thesis is the implementation of a Location-Based Augmented Reality application which in turn has a dedicated interface to mapping services.

Finally, this thesis proposes to test the implemented application to analyze usage preferences between users to map services and location-based augmented reality.

TABLE OF CONTENTS

INTRODUCTION

1.1. Motivation and Research Goals

Augmented Reality on mobile devices has become increasingly popular for many applications, including search and location, tourism, and shopping; this has been due to the development of new hardware technologies and decreasing costs that has enabled devices enter the market with better features.

Additionally new platforms of software development allow creating applications that are compatible with different devices. Unity 3D is an example of software that has become the favorite multiplatform for developers.

Many services those were previously only offered via web and designed exclusively for desktop computers, have taken advantage of this situation to also move to mobile devices. For example the positioning services that use GPS data to locate the position on the map, where Google and its mapping platform have dominion over any other.

The need to venture into these development platforms, touching and testing the offered potential to programmers and opt for using a different mapping services than Google maps, make that the main objective of this thesis will be to create an platform which combines the mapping service offered in this case by OpenStreetMap, and the features of an Augmented reality application.

To accomplish this goal, the application will be implemented on the multiplatform: Unity 3D. Also this application should be able to take readings of integrated sensors into mobile devices as GPS and compass. Finally, this application will be tested by users to analyze their usage preferences.

1.2. Overview of Thesis Chapter

Chapter 2 reviews literature related to Augmented Reality, including essential sensor for an augmented reality system works. Location-based services are also examined, including too Google maps.

Chapter 3 describes the situation about location services and explains what will be the aim of the thesis and its requirements. The system's design will be then demonstrated in Chapter 4, using some functional diagrams and some theoretical explanations with examples' illustrations that will clarify the intended system's design, to be implemented throughout the project.

In Chapter 5, the implementation process of the application will be discussed, illustrating some code's portions for better understanding. Mainly, three major

issues will be discussed in this Chapter; the implementation of Mapping services block, the Augmented Reality service implementation, and at the end of this Chapter, the testing process of the implemented application.

Chapter 6 will be dedicated for analyzing the results of the testing and the interfaces of the application. At the end of this report, a summarizing conclusion will be stated, recapitulating the major results of the project.

2. THEORETICAL BACKGROUND AND RELATED WORK

2.1. Introduction

The purpose of this chapter is to describe how Augmented Reality changes the user perception with things; theoretically which is the process of transition between Location-based services to integration with Augmented Reality applications. Moreover, this chapter specifies the mapping services conception for navigation maps.

2.2. Augmented Reality

Augmented Reality (AR) is the combination of real and virtual environments. The term “Augmented Reality” was coined in 1990 by Tom Caudell while developing a system for Boeing to help workers assemble aircraft with the aid of screens that projected blueprint information on a worker’s view. Application spaces for AR include entertainment, education, art, navigation, visualization, manufacturing, medical and military fields. The processing powers of home computers and mobile devices have steadily increased over the years to enable the use of more and more computationally intensive Computer Vision (CV) techniques. Every year has brought advances in the field of computer vision and Augmented Reality in general.

In recent years, the processing capabilities of mobile smart-phones have reached a stage where they have become a viable platform for consumer level Augmented Reality applications. This increase in processing power combined with integrated orientation sensors present in most new smart-phones [1] is making smart-phones an increasingly popular platform for Augmented Reality research.

Augmented Reality is closely linked to Virtual Reality (VR). Where Virtual Reality is the immersion of a subject into a purely virtual world, Augmented Reality is the immersion of a subject into a version of the real world that is augmented with additional information. Virtual objects are overlaid on top of a real view of the real world in real-time and must be drawn in such a way as to appear a natural part of the world.

Fig. 2.1 demonstrates two examples of Augmented Reality. On the left, the real information of obstacles is showed on the screen of NASA X38, so these obstacles appear to be a natural part of the real world. On the right is an example of a different application for AR, EdiBear, based on SARI engine developed by Samsung Electronics, it’s a game application where a Model 3D is superimposed on top of the view of real world objects.

(a) NASA X38 display showing video map overlays including runways and obstacles during flight test in 2000.

(b) Samsung SARI AR SDK marker less tracker used in the AR EdiBear game (Android OS)

Fig 2.1 Examples of Augmented Reality.

2.3. Essential Sensors for an Augmented Reality System over Mobile Devices

An augmented Reality system depends basically on Orientation and input sensors. Orientation and input sensors, provides the system with visual input, user input and potentially with data that can aid orientation. Therefore, AR system must be aware of its orientation and position within the surrounding environment on order to augment the environment.

Location

A popular way of determining one's position on earth is by utilizing the Global Positioning System (GPS) developed by the United States military and became fully operational in 1994. It was available to civilians in degraded from until 2000 when "Selective Availability" was discontinued and it was made freely available worldwide. Alternatives include Cell phone tower based location determination and Russia's Global Navigation Satellite System (GLONASS), but the former does not provide sufficient accuracy and the latter is not currently fully operational. GPS is sufficiently cheap and accurate for determining location in AR applications, and GPS sensors have been included in mobile phone since 2005.

Orientation

In 3D space, orientation of a rigid body can be uniquely determined by three angles, or Degrees of Freedom (3-DOF): pitch, roll and yaw. In AR, yaw can be seen as the direction an observer is facing while pitch and roll can be seen as the way an observer is tilted. Figure 2.2 is a visualization of these three angles.

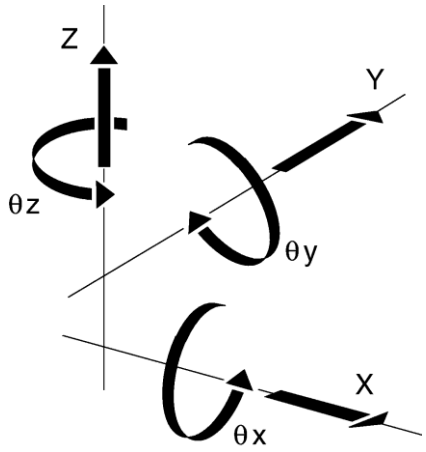

Fig 2.2 Visualization of the three angles of orientation: pitch, roll and yaw.

Magneto-resistive sensors, or magnetometers, are used as an “electronic compass” and are used to determine direction or magnetic north of the planet. Newer generation of electronic compasses use three axes to ensure a correct reading no matter the tilt of the sensor.

Accelerometers measure proper acceleration, which is the acceleration experienced by an object relative to an object in free fall. An accelerometer at rest will register 1g (approximately 9.81 m/s^2) upwards because a stationary object on the Earth’s surface is accelerating upwards relative to an object in free fall. Accelerometers cannot determine yaw (direction), but they can be used to determine pitch and roll.

Electronic gyroscopes can be used to determine all three degrees of freedom of orientation at once. Some cutting- edge smart phone devices are now being released with embedded Micro Electro-Mechanical System (MEMS) gyroscopic sensors. A common problem in visual tracking systems is that sudden movements of the camera can cause image blur which may reduce, or inhibit, further visual tracking. A device with bot a gyroscope and accelerometer will be able to determine direction accurately, as well as detect and compensate for loss of visual tracking when sudden movements occur [2].

Video Feed

AR super-imposes information on the view the user is seeing. For dynamic systems where information must be blended with objects in the user’s view and the user’s view of the real world, a capture device such as a digital camera is required. For AR purposes a real-time camera (video camera) is preferable. Most applications use a single camera and are known as monocular systems. Stereoscopic systems use two cameras to determine depth and some systems may use several cameras to improve depth estimations.

Touch Input

On hand-held AR systems such as smart-phones or tablet PCs, the screen may double as input and an output device. A touch-sensitive layer can capture multiple touch inputs which can be used to manipulate AR objects directly on the screen.

2.4. Location-Based Services and Augmented Reality

The areas of location based services and augmented reality are currently discussed separately. The argument of this separation is purely artificial. From other point of view, applications from both areas solve the equal problem. The key difference is just the information presentation. The location based service can be described simply as an application providing required information with regard to the user location. For instance, in case user stands within the area of a train station, this application is able to detect his/her position and provide automatically train schedule for this particular station. The location works as a kind of input device. Advanced application based on this principle is Google Now. Now provides different information cars that are based on user location, expected events, contacts etc. (see Fig. 2.3)

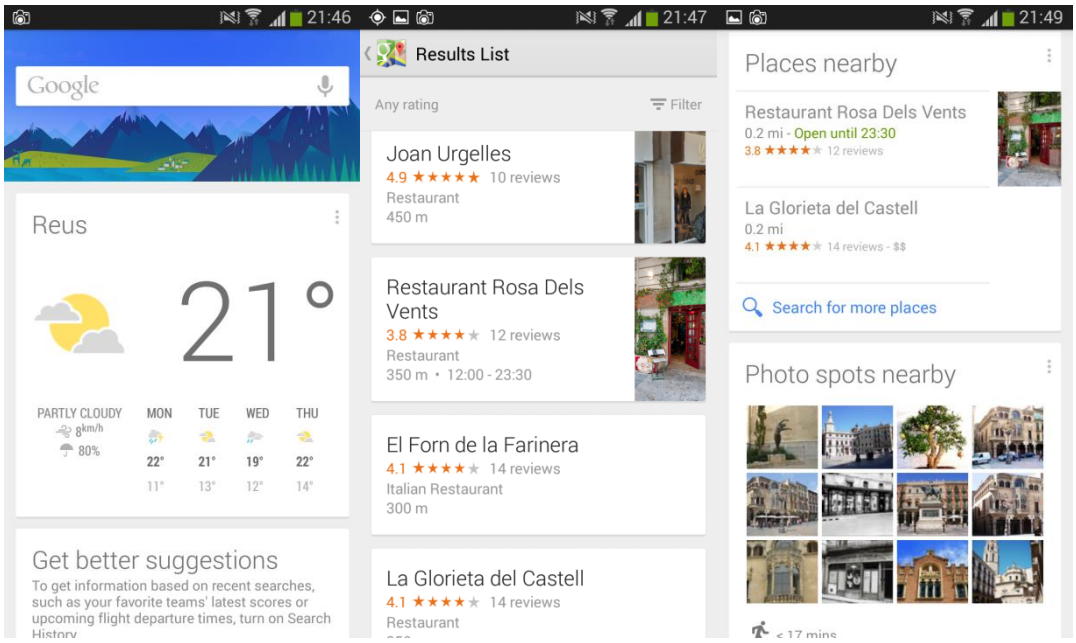

Fig. 2.3 List possible Google cards. Example of location-based information provided by Google Now application.

The Augmented Reality application is usually able to detect objects within the user surrounding using image processing and replace these objects with some connected virtual objects. Nonetheless, the information that is embedded into the scene is not necessarily based just on image processing.

For instance, the augmented reality application design in this thesis uses just GPS. Therefore, it shows the additional information purely on the location basis. If the gist of the augmented reality approach is summarized, the key is not the method used for an object or user detection; it is the way, how information is presented – a combination of a real world image and additional digital information (text, 3D model etc.). Hence, even though there are exceptions on both sides; the location based services and augmented reality applications are substantially overlapping areas.

On the Fig. 2.4, it is possible to see a smooth transition from a common location based application to clear example of an augmented reality application

In last years, there was a substantial emergence of the location based services. Examples can include location based searching, map applications, notes with stored location and automatic transportation schedules. With the occurrence of this area grows also the importance of augmented reality that provides a very straightforward way for information presentation. Therefore, this thesis has focused on the issue of merging these two areas from the architectural point of view.

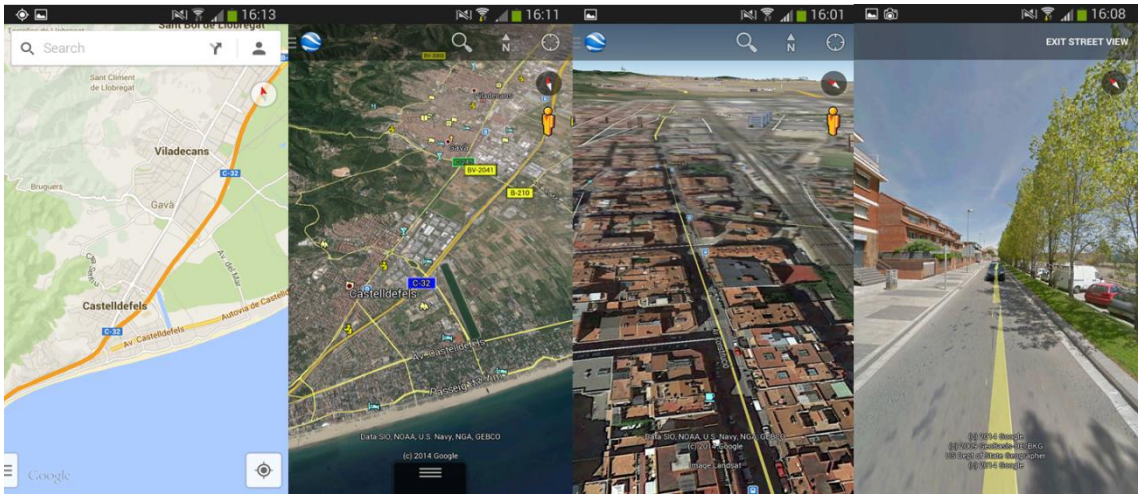

Fig 2.4 Smooth transition from common map with location-based information, through 3D view and common navigation to full AR view using Street View.

Location based services (LBS) applications integrate its user or device location with other information to provide an added value to the user. Nowadays, LBS are used in many applications: military or emergency solutions and commerce. Last group consists especially of mobile technologies such as cellphones and tablets. This Thesis wants to aim at the third category.

Contemporary mobile device is embedded with useful sensors as GPS and GLONASS navigation system receivers, accelerometer, gyroscope, magnetometer and so on. These sensors help detect the user's location (geographical coordinates, altitude) and orientation (cardinal directions from the Earth geomagnetic field). This piece of information allows the developer to

create new contexts in an existing application (e.g. determinate weather in the current location, regardless of user input or movement) or an entirely new service (e.g. position of another person or device on the map).

These location services mainly work outside, whereas inside building GPS works hardly. Also the magnetometer is sensitive to electromagnetic interference. There are several methods for the indoor location detection, mainly based on wireless technologies, such Wi-Fi, cellular phones of RFIDs.

2.4.1. Comparison between Augmented Reality and Location-Based Services Architectures.

One detail distinguishing Augmented Reality and Location Based Services application is the architecture. AR software is often based on a stand-alone paradigm. This thick client representation is made to work out of the box, so all required parts are stored within the device: suitable algorithms, data for augmentation (e.g. 3D objects and sounds) etc. Advantages of this solution are clear; Application works offline, this it does not depend on network connection parameters. Also, there is low security risk, because of application uncommunicative; on the other hand, various problems arise with this approach: At first, resources of the device are frequently limited (very limited amount of data can be stored, computation performance is incomparable with common desktop or server CPUs etc.).

The thick client is habitually created for one platform; therefore, it frequently does not work on other platforms. Very noteworthy is also the fact that data updates have to be done individually on each device.

On the contrary, LBS applications frequently use the service based approach (client-server architecture). Data and even whole applications are stored at one place. Thus being accessible from virtually any device. Users connect to this service usually via thin clients. It provides e.g.: cloud data storage, data sharing among users, simple updates of algorithms and data, automatic feedback from users etc. Moreover, it is easier to create a multiplatform thin client software then create a multiplatform stand-alone application (dependence on other APIs etc.).

The necessary Internet connectivity is the biggest bottleneck of this approach. Besides the costs and power consumption, there exist risks of denial of service. Therefore, this approach requires a subtle technical solution and consideration of possible security risks.

Despite mentioned problems, there is a clear trend in LBS, move the application data or move whole application engine to a remote server or cloud. Important developers such as TomTom or Google link their devices and applications with their servers and offer online traffic reports, searching for persons, point of interests and other information pieces.

2.5. Map Navigation Services

The mapping services was born to respond to a problem: the absence of free, reusable geographic data al accessible prices, previously jealously guarded by private operators (Navteq and Téléatlas, for example) or institutional organizations (the French IGN and Ordnance Survey, before their opened the aforementioned APIs)

Google launched its online mapping service in February 2005. While digital maps had been present online since the early 1990's, Google's service set itself apart from the competition with its user-friendly consultation interface and greater interactivity.

In that same year, 2005, Paul Rademacher working for DreamWorks, hacked the platform of Google maps; While this impromptu hacking placed the spotlight on Google's role in the online mapping sector, a number of other mapping services had made their APIs available online starting in 2005: Microsoft's Bing Maps, the UK Ordnance Survey's OpenSpace, and the collaborative open source mapping project OpenStreetMap (OSM) [8].

The maps of the following mapping services cover the world, but may have insufficient details in some areas:

- OpenStreetMap
- ArcGIS online
- Google Maps
- Bing Maps
- ViaMichelin
- MapQuest
- Mappy
- Others

Briefly, the online mapping services have long been dominated by Google Maps. The rise of Creative Commons-licensed service OpenStreetMap could shake up the established order, threatening web emperor Google.

2.5.1. Google Maps Services and Google Glass

Google Maps is a desktop and mobile web mapping service application, offering satellite imagery, street maps, and Street View perspectives, as well as functions such as a route planner for traveling by foot, car or public transportation.

When Google launched Google Maps, the online mapping service had been solely intended to search for addresses and directions; Google quickly understood its potential for being used in other ways; one of them was the Google Maps API (Application programming interface) was made public, serving as a mapping basis for new applications.

This data library is searched by user's requests in order to display maps on their Web pages – a process that allows for map personalization (size, zoom scale, default localization etc.) and facilitates interoperability between the API's of other services and online applications.

In October 2011, Google decided to start charging fees for access to the Google Maps API once daily usage limits are surpassed. Every time an Internet user visits a site that uses a Google map, a request is sent to the Google Maps API, and the number of requests made by a given site thus equals its number of visits. The more popular a site or application, the more its risks having to pay in order to continue to display a Google map.

In fact, many services using Google Maps already pay usage fees, having adopted the Google Maps API for Business offer. Conversely, the vast majority of sites using Google Maps may continue to do so without paying, remaining below the daily usage limit for free access: the Static Maps API and JS Maps API v3 have the comfortable daily usage limit of 25,000 map views before fees are incurred. Thus, only 0.32% of the API users are affected by this measure. However, this tiny percentage includes major customers such as Apple, whose tens of millions of iPads and iPhones sold worldwide come with the Google Maps application built in: at four dollars per 1,000 map views above the daily limit of 25,000, the bill starts to look steep.

Google justifies its pricing system by its desire to "encourage responsible use" of map data and "secure the long-term future" of the service. This fee hike nevertheless brought about a swift reaction, with a number of applications deciding to abandon Google Maps in favor of OpenStreetMap.

This element in Google's strategy may potentially more drastically transform the power between the two organizations.

2.5.2. OpenStreetMap, another Mapping Services

OpenStreetMap is a collaborative project to create a free editable map of the world. The OSM's project started because most available maps, such Google or Bing, have legal or technical restrictions on their use: lot of content are not exposed due to copyrights, belonging to mapping companies (e.g. NAVTEQ or Tele Atlas) or national agencies (e.g. Ordnance Survey from United Kingdom or IGN-National Geographic Institute). OpenStreetMap is different. All of the quality data contributed is openly available, just like Wikipedia.

Therefore, two major driving forces behind the establishment and growth of OSM have been restriction on use or availability of map information across much of the world and the advent of inexpensive portable satellite navigation devices.

OSM is one of the world's most active open and crowd-sourced projects with over 1.5 million registered editors (a number that has been doubling every year)

[9]. It has grown exponentially faster than anyone could have ever imagined ten years ago.

The quality of the map data has evolved so much that, in the past couple of years, developers like Foursquare, Pinterest and Uber have integrated OSM as a display map into their products; most likely as a way to get access to a more detailed map and to avoid those costly fees from Google.

The following pictures show the differences between Google and OSM maps:

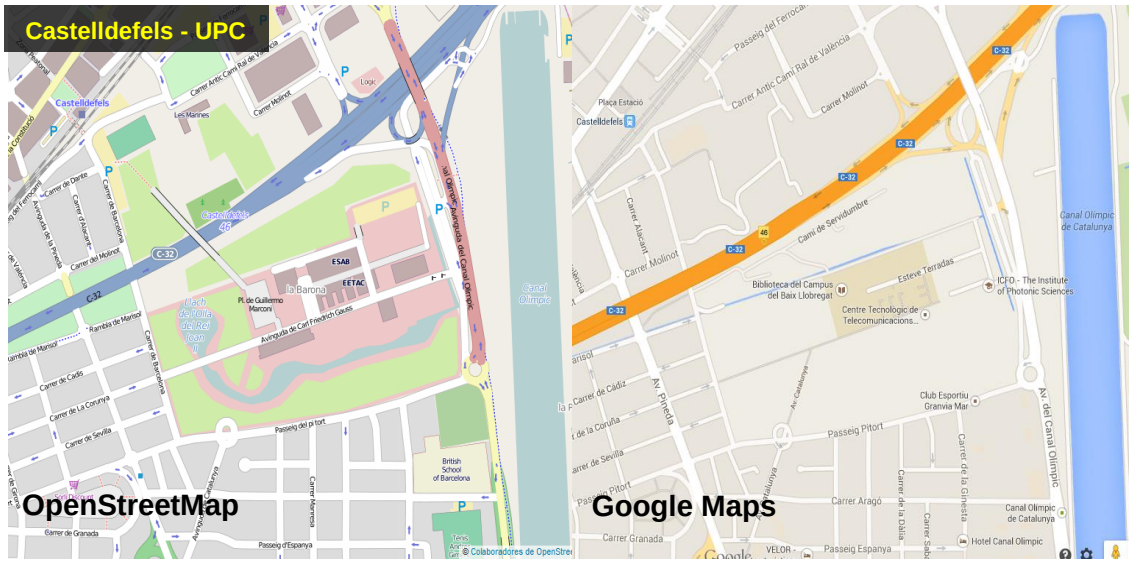

Fig. 2.6 Current OSM vs. Google Map of Castelldefels, Catalonia, UPC buildings.

In parallel to the changes in Google Maps mentioned in the previous subsection, OpenStreetMap currently has the wind in its sails. Political authorities are gaining awareness of the advantages of open mapping. Therefore, there are several applications which have made the change to OpenStreetMap.

2.6. Google Glass Reality Augmented

Google Glass is a wearable computer in the form of a headset that looks like a pair of glasses and lets users accomplish various tasks and receive quick information in a smartphone form. The difference is that Google Glass is used hand-free and does not requires the user to look away to receive information, which is officially called a heads-up display [3]. Google Glass was created to ease the user's daily and social life by taking the bulky and physical part of technology out of the way to help the user connect to the world in a faster way. It is connected to the Internet, and the user can take photographs, record videos, send and receive messages, and stream live recordings, amongst

others [4]. It supports Apps that people can develop for Google Glass using its API, which is called Mirror API.

Mirror API is fully web-based, Glassware is always server-side and can be developed in Java, PHP, Python, Go and .NET. Because of this approach, the only way of interacting with Glass is through the cloud [5]. Cloud computing, as used in Glass, is a technology that allows a large number of devices to access and interact with one or multiple services simultaneously [6].

Glassware is deployed to the Google App Engine which is a type of cloud service known as “PaaS” (Platform as a Service). This distributed development platform allows developers to build and manage their Glassware without having to worry about hosting, scalability and hardware maintenance [7]. The Glassware built within the Google App Engine can be considered “SaaS” (Software as a Service). These applications run within the cloud and can connect to a large number of clients (users) simultaneously.

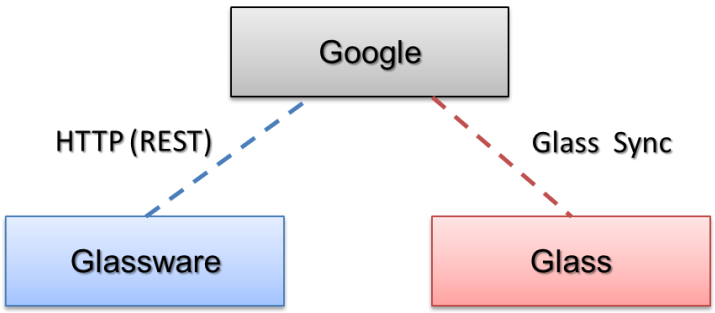

Fig. 2.7 Stack showing the communication between Glassware and a Glass device using Google’s Cloud

Because the applications are approached via the Internet, users don’t have to worry about installing and updating Glassware and can easily manage with which applications they want to share information.

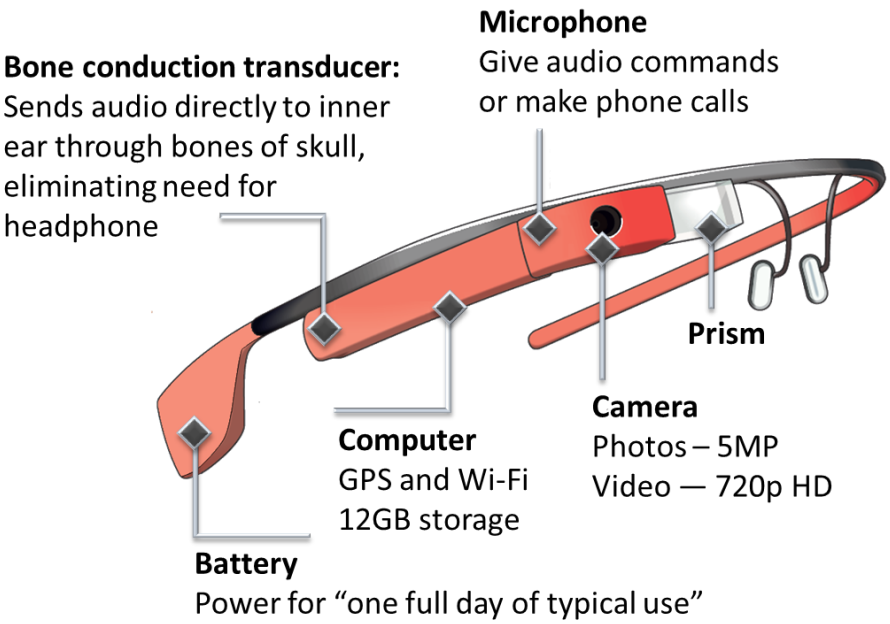

Fig. 2.8 Google Glass augmented realty computer components.

However, Google Glass has weaknesses in web technology and hardware area. Since each piece of Glassware has to run within Google's cloud, Google will always be in control of the content displayed on Glass. Developers have to comply with Google's terms and conditions which don't only result in limitations concerning application functionality and design.

Another weakness of Glass lies within its shared data-connection with a smartphone over Bluetooth or Wi-Fi which is necessary to use the majority of Glassware and build-in functions on the go. This doesn't only require a decent smartphone and sufficient 3G-subscription, but also causes the smartphone's battery to drain rapidly, especially when GPS is enabled as well.

The biggest drawback however is that several developers who had a first glance at the Mirror API were extremely disappointed by the fact that applications are solely web-based and running code on the device itself is impossible. This, in combination with the fact that the prism of Glass only allows a relatively small display, applications which contain full Augmented Reality features such as image registration and object tracking are off the table, resulting in a head-mounted display (HMD) rather than an Augmented Reality device often suggested by the media.

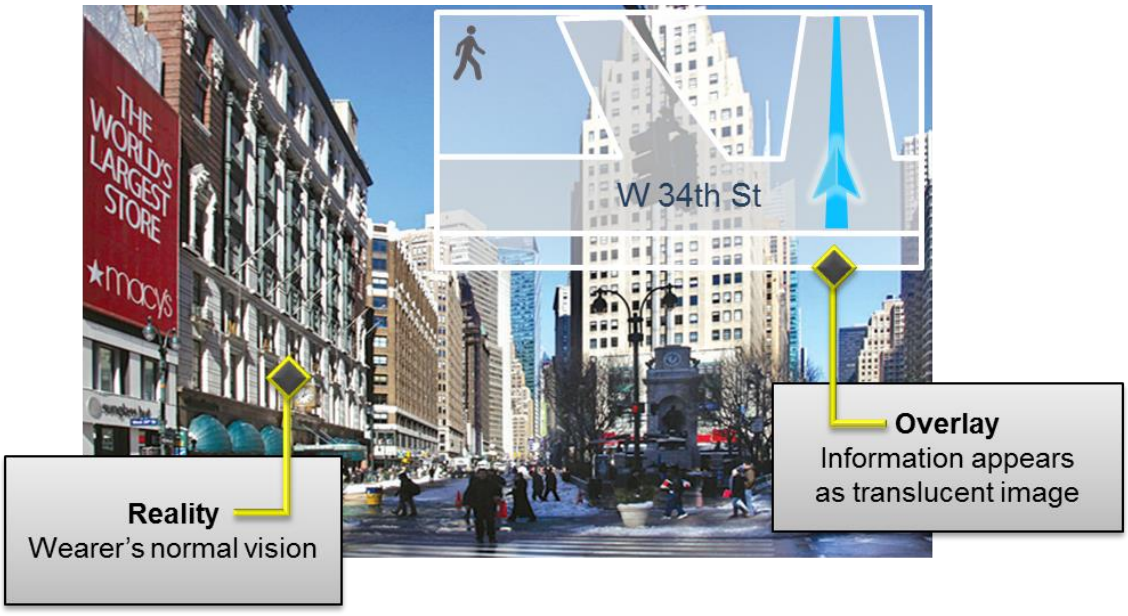

Fig. 2.9 Viewing Experience of Google Glass – Augmented Reality.

From the hardware point of view, in order to provide the Augmented Reality overlay, Glass uses a small projector which beams a 740x360 pixel image into a plastic prism (Fig. 2.8) where the rays of light are directly deflected onto the user's retina; but, the dimensions of the prism and the fact that it only covers one eye can be considered both a strength and weakness. It definitely adds in the unobtrusive nature of Glass, but also makes it technically impossible to display extensive content or implement full Augmented Reality.

3. REQUIREMENTS AND ANALYSIS

3.1. Introduction

In this chapter, the discussions will focus on explaining the requirements of thesis, which are choosing the platform on will run the application and what kind of mapping services will be used, besides definition of interface of the application.

3.2. A Comparative Study of User Experience towards Mobile Location Services

In more recent years, there have been many technological advances in relation to mobile devices; innovations as 3G to 4G networks, more precise sensor reading and their cost much cheaper make to change the conception of users has towards technologies that are within reach.

From the point of view of Location-based services, a new variant has been introduced; it's the fusion between augmented reality application and location-based services. Therefore, a positive results is obtained by users because of effective use of information arrive to them. Consequently, the question is how much useful is an Augmented Reality application as tool of location and which is the preference that users have between both service and architecture?.

3.3. System Requirement

To solve these two mentioned questions in the previous subchapter, an Augmented Reality application is required for doing the role of evaluation tool; how was explained at introduction, the thesis aim is to design a program. The requirement system of that application will be that meets the following functions:

- The application should have one interface of location-based over mapping service and other second interface over augmented reality.
- The mapping service used will be OpenStreetMap; therefore the map navigation has to be implemented with OpenStreetMap's API.
- The Augmented Reality interface will show the buildings are near from real position of users (device position).
- The entire application has to be programed over Unity 3D platform because one of the objectives is to explorer the Unity potential for working with GPS and other sensor of mobile devices.
- Finally both interfaces will use the same information to user.

3.4. Requirement Analysis

The developed application should allow answering some questions about map navigation and location-based over augmented reality, answer which kind of application the users prefers and finally to analyze the potential of Unity to build an application based on GPS, and the usage of mobile device sensors.

Hence, once the application will be finalized, that would allow analyzing the advantages that offers each one of services, therefore the question is the following:

- Which kind of services in mobile devices prefers the users?

For the purpose of this project and due to the only way to discuss the utility of a program based on user reviews, it was decided to test the program with a number of people and finally to conduct a survey to assess their views about the program.

- How well works the Unity3D engine on the android platform, to design an augmented reality application using sensors and positioning a mobile device?

At project completion, should be able to answer this question by assessing whether the program worked correctly on devices and thus evaluate the potential for Free Unity 3D version.

4. DESIGN

The objective of this Chapter is to explain the application design of the developed Software in this thesis; it will be divided in two parts; both parts will reflect the Location Services through Augmented Reality and the use of Maps. Therefore before to specify the respective designs; some of the most important tools are discussed below, to understand their functionality.

4.1. Tools

The main tool to develop the Augmented Reality application, is the cross-platform Unity, nevertheless that development requires a mapping services, in this case OSM will be used; and an helper that supports the conversion of maps to latitude – longitude as Proj.Net tools. So it is necessary to consider the mobile phone sensors.

The following subsections will explain theoretically and practically the used tools in all the process of development of this software.

4.1.1. Unity 3D

Looking at the objectives of this thesis, one of them is working on Unity3D, to know, what is the potential of this platform to develop an application, which uses GPS with the combination of Augmented Reality?, so an introduction of Unity will be useful.

Unity is a cross-platform game engine developed by Unity Technologies. It is used to develop video games for web plugins, desktop platforms, consoles and mobile devices. It grew from an OS X supported game development tool in 2005 to a multiplatform game engine.

Unity3D interface allow working with several Tabbed Windows, called Views. There are several types of views in Unity; they all have specific purposes which are helpful. The Unity views are the following (Fig.4.1):

Fig. 4.1 Unity 3D interface and their respective views.

- Project Browser
- Hierarchy
- Toolbar
- Scene View
- Game View
- Inspector

The game engine's scripting is built on Mono, the open-source implementation of the .NET Framework. Programmers can use UnityScript (a custom language referred to as JavaScript by the software), c#, or Boo (Python syntax).

Unity 3D supports to work with model or art asset of other 3D modelling applications, they are 3ds Max, Maya, Softimage, Blender, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks.

The implementation of the application will be completely in C# and the models or asset worked on Blender and then imported as Prefabs.

The main advantage of his interface it the Hierarchy, it contains every GameObject in the current Scene. Some of these are direct instances of asset files like 3D models, and others are instances of Prefabs, custom objects. The objects can be selected in the Hierarchy and drag one object onto another to make use of Parenting. As objects are added and removed in the scene, they will appear and disappear from the Hierarchy as well.

Unity uses a concept called Parenting. To make any GameObject the child of another, drag the desired child onto the desired parent. A child will inherit the movement and rotation of its parents.

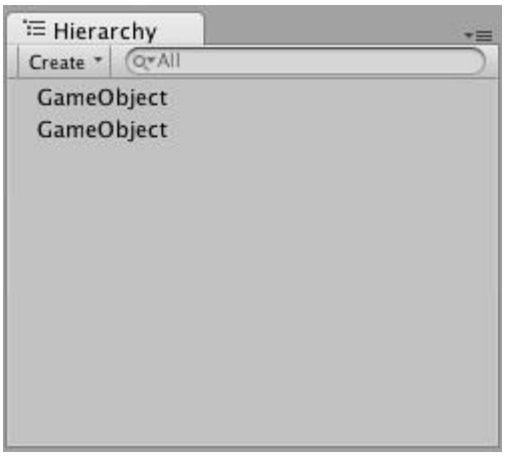

Fig. 4.2 Two unparented objects

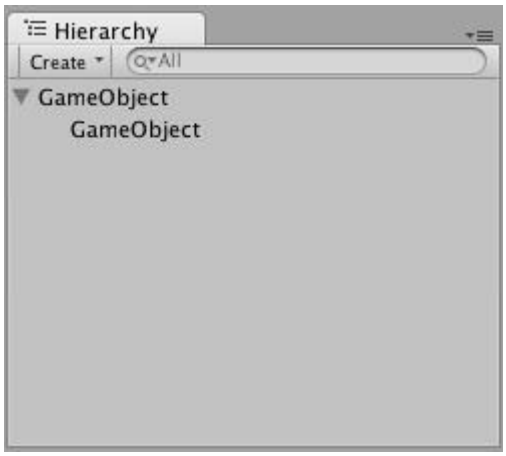

Fig. 4.3. One object parented to another

Every object has a Transform; the Transform component determines the Position, Rotation, and Scale of each object in the scene.

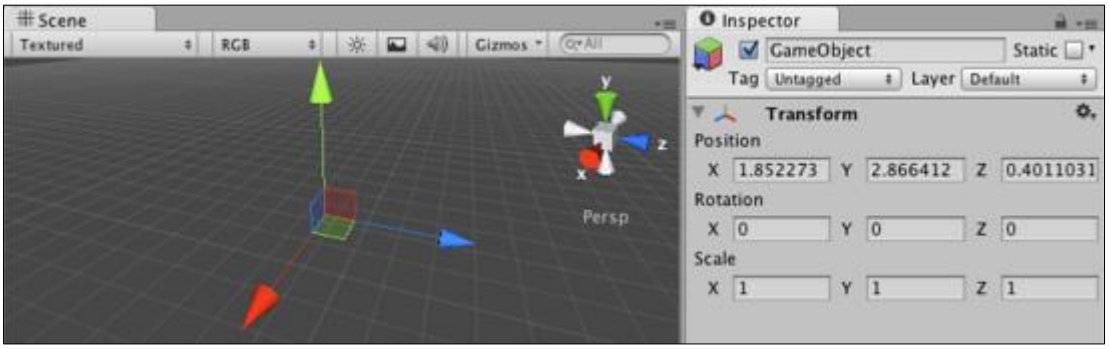

Fig. 4.4 The Transform Component is editable in the Scene View and in the Inspector

4.1.2. Open Street Map

The main feature of OpenStreetMap is that OSM provides a repository of quality map data, as Google do but with more details in their maps. Everyone can access it on their main site or embed it in their own website. So there are other ways to access OSM data which will be explained in this chapter.

4.1.2.1. Available API

Indeed OpenStreetMap provides several API to create, modify and read geographic content. In addition some API provides advanced querying features (e.g. geocoding or reverse geocoding features).

The entire available API is listed below:

- **API**: OpenStreetMap editing API for fetching and saving raw geodata from/to the OpenStreetMap database.
- **Xapi**: OpenStreetMap extended read-only API, based on a modified version of the standard API, which provides enhanced search and querying capabilities, as bounding-box or X-path.
- **Nominatim**: OpenStreetMap search engine, for searching OSM data by name and address and generating synthetic addresses of returned OSM points (i.e. reverse geocoding).

Xapi offers search queries for some common simple use cases and helps to put load off the main API. In particular, it reimplements the standard map request such that it can be performed much faster. Xapi uses a REST style interface with x-path flavouring. More sophisticated queries can be handled with other read-only mirrors such as Overpass API which support more powerful but more complex syntax.

Xapi only ever deals with elements that are current and does not return any elements that are historical or deleted. The source database is a mirror of the main OSM and is updated via the per-minute diff dumps. The data is normally no more than about 10 minutes behind the main database.

Xapi and Nominatim are the most interesting for today. These API are used as one of the sources for the Search box on the OpenStreetMap home page, but also powers other websites like MapQuest.

MapQuest is an online mapping company based in the United States and their company gives their support to OpenStreetMap making MapQuest the first large online mapping service to embrace OSM. They earmarked $1 million in resources to help improve the OSM data in the United States with the stated intention of possibly using OSM data for their maps in the future.

The MapQuest Open Initiative is one of the results of their support efforts. In addition to localized websites it provides an API to read and query OSM data. These data are frequently updated from OSM (Every 15 minutes for map data, every 5 minutes for search data and daily for routing data).

The project will uses Xapi, because provides bounding box of data and the application will need to show bounded information.

4.1.2.2. The Data Model

In order to use API via Overpass API, there is a need to explain the data model used by OpenStreetMap. OSM data is stored as XML. Basically, it is a list of instances of 3 data primitives:

- **Node**, a single geospatial point.
- **Way**, an ordered interconnection of at least 2 nodes that describe a linear feature (e.g. street, footpath, railway line, river,…).
- **Relation**, used to group way or node objects that are geographically related (i.e. connected or adjacent to one another).

To summarize, querying the API should return all or part of the following XML schema:

Table 4.1 Result of querying the API, XML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="Overpass API">
<note>The data included in this document is from
www.openstreetmap.org. The data is made available under
ODbL.</note>
<meta osm_base="2014-05-05T17:12:02Z"/>

  <node id="2542835051" lat="41.1566855" Lon="1.1127319">
    <tag k="amenity" v="bank"/>
  </node>
  ...
  <way id="124097636">
    <nd ref="1382092447"/>
    <nd ref="1382092449"/>
    <tag k="highway" v="residential"/>
  </way>
  ...
  <relation id="3673829">
    <member type="node" ref="245471476" role=""/>
    <member type="way" ref="200188956" role=""/>
    <member type="way" ref="88815384" role=""/>
    <member type="node" ref="2159299141" role=""/>
    <tag k="route" v="light_rail"/>
    <tag k="type" v="route"/>
  </relation>

</osm>
```

A node consists of a single point in space defined by its latitude, longitude and node id. Nodes can be used on their own to define point features. When used in this way, a node will normally have at least one tag to define its purpose.

For example, in the table above a bank may be tagged simply with amenity=bank

A way is an ordered list of nodes which normally also has at least one tag or is included within a Relation. A way can have between 2 and 2000 nodes.

A relation is one of the core data elements that consist of one or more tags and also an ordered list of one or more nodes and ways as members which is used to define logical or geographic relationships between other elements, e.g. bus route.

4.1.2.3. Querying the API

The previous XML result can be obtained by querying Xapi. The map query is identical to the main API map query and returns:

All nodes that are inside a given bounding box and any relation those reference them.

All ways that reference at least one node that is inside a given bounding box, any relations which reference the ways, and any nodes outside the bounding box that the ways may reference.

Table 4.2 Retrieving map data by bounding box: GET

```
GET /api/0.6/map?bbox=left,bottom,right,top
```

Where:

- **Left** is the longitude of the left side of the bounding box, or *minlon*.
- **Bottom** is the latitude of the bottom side of the bounding box, or *minlat*.
- **Right** is the longitude of the right side of the bounding box, or *maxlon*.
- **Top** is the latitude of the top side of the bounding box, or *maxlat*.

For example, given the following bounding box:

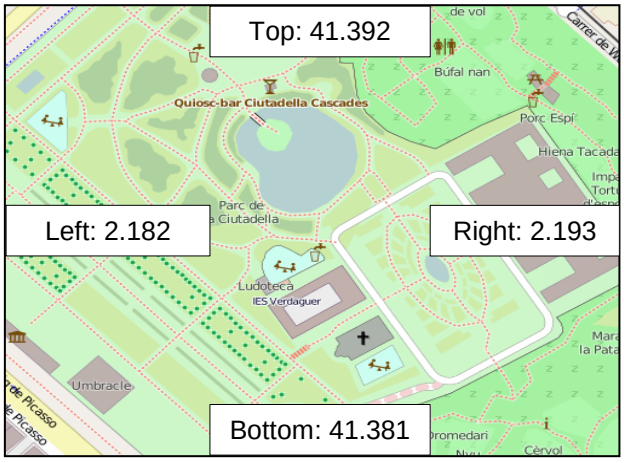

Fig. 4.5 Bounded box with its components.

The URL for this map request has the following form:

Table 4.3 Example of the map request.

```
http://www.informationfreeway.org/api/0.6/map?bbox=2.182,41.381,2.193,41.392
```

The querying engine allows the use of filters for all three kinds of elements; nodes, ways and relations.

For all three elements, the URL request takes the following form:

Table 4.4 Request for each element of Data model.

```
http://www.informationfreeway.org/api/0.6/node[...]
http://www.informationfreeway.org/api/0.6/way[...]
http://www.informationfreeway.org/api/0.6/relation[...]
```

In this thesis, the application will be implemented using Overpass API or (OSM3S) that serves up custom selected parts of the OSM map data.

Overpass API has a powerful query language beyond XAPI, but also has a compatibility layer to allow smooth transition from XAPI. To support small and well scaling OSM main services, Overpass API is run as a third party service. Everyone can use the public visible instances:

- http://overpass-api.de/ with query base address http://overpass-api.de/api/ (4 physical cores, 32 GB RAM).
- http://overpass.osm.rambler.ru/ with query base address http://overpass.osm.rambler.ru/cgi/ (8 cores, 64 GB RAM).
- Overpass API on openstreetmap.fr

Both servers have a total capacity of about 1.000.000 requests per day. Queries to the overpass API are in XML or Overpass QL form.

Overpass API has got a compatibility layer for the most common XAPI queries. The base URLs are:

Table 4.5 Base URL to make a request to overpass API

```
http://www.overpass-api.de/api/xapi?
http://overpass.osm.rambler.ru/cgi/xapi?
```

The only substantial differences are that Overpass API does not deliver changeset, user, version and timestamp data.

Both versions of the map call are supported, where the variant with map has slight performance advantages:

Table 4.6 Variant to querying the bounded box of map.

```
http://www.overpass-api.de/api/xapi?map?bbox=7.1,51.2,7.2,51.3
http://www.overpass-api.de/api/xapi?*[bbox=7.1,51.2,7.2,51.3]
```

The order of the four numbers is *left*, *bottom*, *right*, *top*.

For node, way, and relation it is possible to either combine arbitrarily many tag search criteria with exactly one or no value:

Table 4.7 three different way to get filter data.

```
http://www.overpass-api.de/api/xapi?node[name=Castelldefels]
http://www.overpass-api.de/api/xapi?node[name=Castelldefels][railway=halt]
http://www.overpass-api.de/api/xapi?node[name=Castelldefels][railway=*]
```

4.1.3. Mobile Device Sensors on Unity

In the application design, the Mobile Device Sensors will take the leading role are the GPS and the Compass, the indispensable twin tools of navigation.

These sensors are essentials since GPS sensor will be used to get the lecture of device position and the compass will determine the orientation of objects in general.

Every model of mobile phone has often the same sensor but these sensors are not made by the same manufacturer. So that implies not always we will have the accuracy that the application requires. To access at the data of Sensors, Unity 3D provides of the input class for working with these functions.

One of these classes is the LocationService Class, whose functionality is to start and to stop location service updates, Start function enables to return the last location coordinates and stop function allow saving battery life of the device.

When the start function is enabled, the "*lastData*" function returns the last measured device geographical location. In the following box, an example of use will be showed:

Table 4.8 Example of code for starting the input services of devices.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
   IEnumerator Start() {
      if (!Input.location.isEnabledByUser)
         return;
      Input.location.Start();
      int maxWait = 20;
      while (Input.location.status == LocationServiceStatus.Initializing && maxWait > 0) {
         yield return new WaitForSeconds(1);
         maxWait--;
      }
      if (maxWait < 1) {
         print("Timed out");
         return;
      }
      if (!Input.location.status == LocationServiceStatus.Failed) {
         print("Location: " + Input.location.lastData.latitude + " " +
                              Input.location.lastData.longitude + " " +
                              Input.location.lastData.altitude + " ");
      }
      Input.location.Stop();
   }
}
```

The previous code, checks if the location service is enabled and if it is true, the stars function is called to allow the reading of the "*lastData"*, finally the latitude, longitude and altitude if it exits, is printed in console.

For reading the compass data, the "*trueHeading"* function returns the heading in degrees relative to the geographic North Pole. This value is used in application, to translate and rotate the game objects to the needed form.

Table 4.9 Example of code for using the obtained data of the compass.

```
function Update() {
                // Orient an object to point northward.
                transform.rotation = Quaternion.Euler(0, -Input.compass.trueHeading, 0);
        }
```

The last box shows how to use this value; in that case, the transform of an object will take the compass orientation.

4.2. Transverse Mercator projection

The fact to create an application, which projects geographic coordinate data on a map, it needs a method of map projection. The following explanation about Projection, in particular Transverse Mercator projection helps the design.

The Transverse Mercator map projection is an adaptation of the standard Mercator projection. The transverse Mercator projection is widely used in national and international mapping systems around the world.

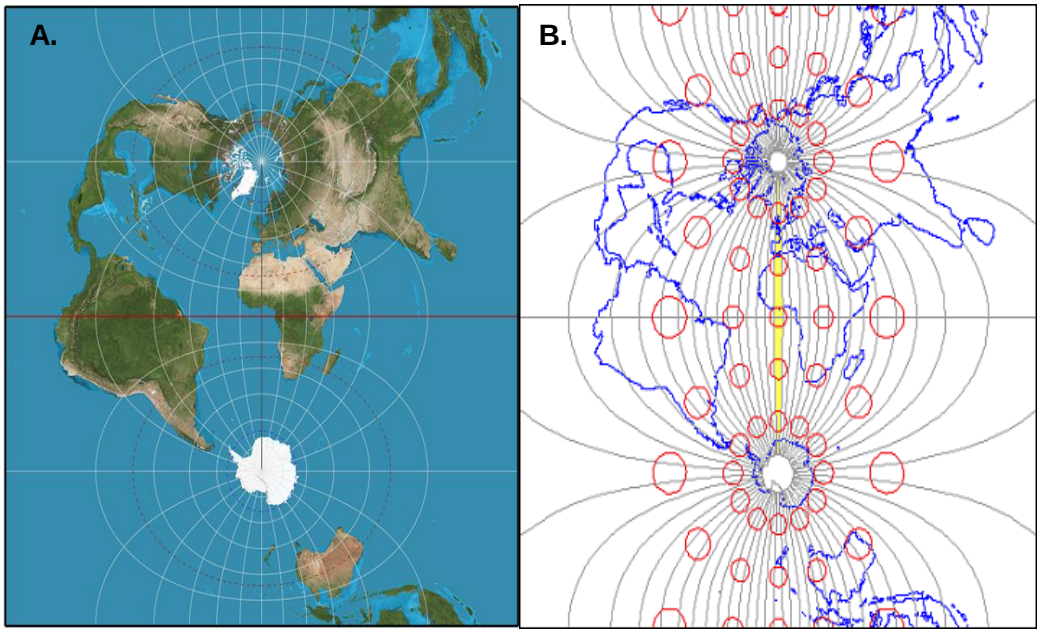

Fig. 4.6 (a) Spherical transverse Mercator (truncated at $x = \pm\pi$ in units of Earth radius.). **(b)** Red circles reveal the scale distortion introduced during the transformation from geographic to projected plane coordinates.

The meridians and parallels of the Transverse Mercator (fig. 4.6) are no longer the straight lines they are on the regular Mercator, except for the Earth's Equator, the central meridian, and each meridian 90º away from the central meridian. Other meridians and parallels are complex curves.

The spherical form is conformal, as is the parent projection, and scale error is only a function of the distance from the central meridian, just as it is only a function of the distance from the Equator on the regular Mercator.

The ellipsoidal form is also exactly conformal, but its scale error is slightly affected by factors other than the distance alone from the central meridian.

Many popular web mapping applications such as Google, Bing, OpenStreetMap, etc. use Transverse Mercator projection, specifically the EPSG 3857 projection.

4.2.1. WGS84 - EPSG 3857

OpenStreetMap uses the WGS84 spatial reference system used by the Global Positioning System (GPS). It uses geographic coordinates between -180º and 180º longitude and -90º and 90º latitude. So this is the “native” OSM format.

This is the right choice to transform the coordinates into some other spatial reference system or projection or geographical coordinates.

The EPSG code assigned to the projection has changed several times and much of the information displayed on the internet, as well as used in spatial applications themselves, is out-of-date.

The actual projection used by Microsoft or Google itself is a type of hack, which can lead to misalignment issues when the code actually does the projection the “correct way”. For example, it is common to find issues raised across a number of tools that conversion between WGS84 and Bing or Google Maps leads to Y values that are “out” by about 20km [12].

Coordinate systems are defined using a set of parameters, including the type of projection used (Mercator, Albers, etc.); coordinate offset (False Easting, False Northing), the unit of measurement (Metre, Foot, etc.), the underlying datum (WGS84, NAD27, etc.) and many more.

The issue is how determine which one of projection to use is the correct option.

The EPSG 3857 projection with corresponding parameters as follows:

Table 4.9 WKT of WSG84 projection / Pseudo-Mercator.

```
PROJCS["WGS 84 / Pseudo-Mercator",
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84", 6378137, 298.257223563, AUTHORITY["EPSG","7030"]],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich", 0, AUTHORITY["EPSG","8901"]],
    UNIT["degree", 0.01745329251994328, AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Mercator_1SP"],
  PARAMETER["central_meridian",0],
  PARAMETER["scale_factor",1],
  PARAMETER["false_easting",0],
  PARAMETER["false_northing",0],
  UNIT["metre", 1, AUTHORITY["EPSG","9001"]],
  AXIS["X",EAST],
  AXIS["Y",NORTH],
```

```
  AUTHORITY["EPSG","3857"]]
```

The projection parameters defined above for EPSG: 3875 describes a Mercator projection of coordinates defined on the WGS84 ellipsoid, with an inverse-flattening ratio of 298.257223563:

Table 4.10 Spheroid definition of the WGS84 projection.

```
SPHEROID ["WGS 84", 6378137, 298.257223563, AUTHORITY ["EPSG","7030"]]
```

The problem is that neither of these really describes the Google / Bing Maps projection, in which the underlying geographic coordinates *are* defined using WGS84 (as in 3857).

In conclusion the problem is that there is no standardized way to represent this “dual”-ellipsoid system, in which a different ellipsoid is used to interpret the geographic coordinates than is used to project them.

The solution is to provide a little bit more information to explain that the projection should occur from the sphere rather than the WGS84 ellipsoid specified in the geographic coordinates system. The additional information can be supplied in different ways.

For this thesis, Proj.Net will be used, replacing the projection from “Mercator” to “Mercator_1SP” and to add an additional “semi_minor” parameter to the projection parameters in the WKT. This parameter allows to manually specifying the semi-minor axis of the ellipsoid used for the projection, which by default is otherwise assumed to be the same ellipsoid specified in the GEOGCS.

To create a projection based on the sphere that has the same radius as the WGS84 ellipsoid, as Google or Microsoft use, it should states a semi-minor axis of 6378137 m. This will override the WGS84 semi-minor axis that would have been used for the projection otherwise 6356752.314245m.

So, the following WKT works in Proj.Net to specify the correct projection to OpenStreetMap.

Table 4.11 WKT of WSG84 – Simple Mercator Projection for a spheroid datum.

```
PROJCS["WGS84 / Simple Mercator",
  GEOGCS["WGS 84",
    DATUM["World Geodetic System 1984",
      SPHEROID["WGS 84", 6378137.0, 298.257223563,AUTHORITY["EPSG","7030"]],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
```

```
    UNIT["degree",0.017453292519943295],
    AXIS["Longitude", EAST],
    AXIS["Latitude", NORTH],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Mercator_1SP"],
  PARAMETER["semi_minor", 6378137.0],
  PARAMETER["latitude_of_origin",0.0],
  PARAMETER["central_meridian", 0.0],
  PARAMETER["scale_factor",1.0],
  PARAMETER["false_easting", 0.0],
  PARAMETER["false_northing", 0.0],
  UNIT["m", 1.0],
  AXIS["x", EAST],
  AXIS["y", NORTH],
  AUTHORITY["EPSG","900913"]]
```

The final option is to forego the general projection routines and write a specific mathematical transformation designed to convert only between OSM maps and WGS84. The following mathematical functions are provided by OSM:

Longitude /latitude to tile numbers:

$$x = \left[\frac{lon+180}{360} \cdot 2^z\right] \quad \textbf{(4.1)}$$

$$y = \left[\left(1 - \frac{\ln\left(\tan\left(lat \cdot \frac{\pi}{180}\right) + \frac{1}{\cos\left(lat \cdot \frac{\pi}{180}\right)}\right)}{\pi}\right) \cdot 2^{z-1}\right] \quad \textbf{(4.2)}$$

Where Z = Zoom,
Lon = Longitude,
Lan = Latitude.

Tile numbers to longitude/latitude:

$$lon = \frac{x}{2^z} \cdot 360 - 180 \quad \textbf{(4.3)}$$

$$lat = arctan\left(\sinh(\pi - \frac{y}{2^z} \cdot 2\pi)\right) \quad \textbf{(4.4)}$$

Where Z = Zoom,
X, Y = Coordinate of Tile.

4.2.2. Proj.NET

In the previous section was explained the solution of the correct projection to OpenStreetMap; Proj.NET is the used tool to get solve the problem.

Proj.NET performs point-to-point coordinate conversions between geodetic coordinate systems to use in Geographic Information Systems or GPS applications. Also Proj.NET supports:

- Datum transformations
- Geographic, Geocentric, and Projected coordinate systems
- Converts coordinate systems to/from Well-Known Text (WKT) and to XML

Projection types currently supported by Proj.NET are Mercator, Transverse Mercator, Albers, Lambert Conformal and Krovak.

To construct a coordinate system (CS) by well-Known Text, it has to be declared of the following form:

Table 4.12 Definition of a WKT projection as string – c#.

```
string wktEPSG900913 = "PROJCS[\"WGS84 / Simple Mercator\", " +
             "GEOGCS[\"WGS 84\", " +
             "DATUM[\"World Geodetic System 1984\", SPHEROID[\"WGS
             ----THE REST OF WKT defined in the previous section----
            "AXIS[\"x\", EAST], AXIS[\"y\", NORTH]," +
            "AUTHORITY[\"EPSG\",\"900913\"]]";
ICoordinateSystem                    epsg900913                    =
CoordinateSystemWktReader.Parse(wktEPSG900913) as ICoordinateSystem;
```

Finally to define the coordinate system transformation:

Table 4.13 Definition of the coordinate system transformation.

```
ICoordinateTransformation            wgs84ToEPSG900913             =
ctFactory.CreateFromCoordinateSystems(GeographicCoordinateSystem.WG
S84, epsg900913);
```

4.3. Field of View (FOV) in Augmented Reality

The concept of Field of View or Visual Perception has high relevance in terms of how to design the application. The FOV is represented by an adjustable

value in many games and directly affects the distance and size of objects displayed on the screen [10].

One part of the program will be use the camera to create the interface of Augmented Reality, and then the question what extent will the field of vision of the application?

To answer the previous question, the focus of this changes to the visual perception of human. The visual field is defined by the area that people are able to see with the eyes fixed on a point.

The limitations of the static eye reaches to see are: 60º in the nasal side, 65º in the upper direction, 75º and in the lower and 100º in the temporal side. These values are approximated and change in order of the face anatomy of the person.

Briefly, the field of view of the application will be 60º of visual perception.

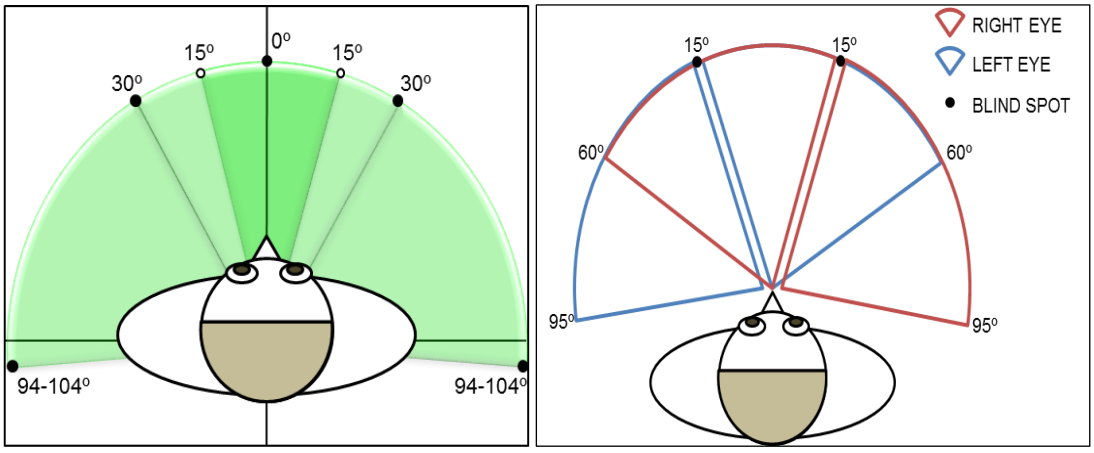

Fig.4.7 Visual perception on Human is limited 60º in the nasal side.

To calculate the angle that the building has respect to the device, a Cartesian plane is considered, the angle formed between the building position and the device position towards X axis, is used to know if that angle is between the FOV angles.

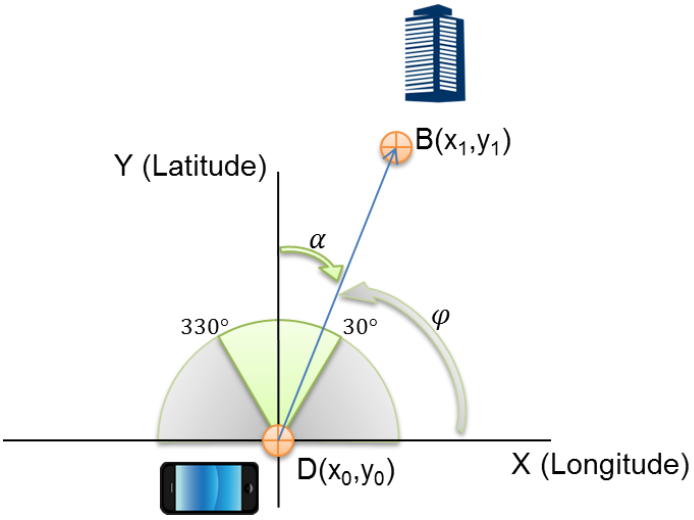

Fig. 4.8 Designed System to get the angle of building towards location device.

The formulas to calculate these angles are the following:

$$\varphi = \begin{Bmatrix} If\ (y_1 - y_0) \geq 0\ \ then\ \varphi = \ \tan^{-1}\left(\frac{y_1 - y_0}{x_1 - x_0}\right) \\ If\ (y_1 - y_0) < 0\ \ then\ \varphi = \ \tan^{-1}\left(\frac{y_1 - y_0}{x_1 - x_0}\right) + 360 \end{Bmatrix} \quad \textbf{(4.5)}$$

$$\alpha = \begin{Bmatrix} \alpha = 450° - \varphi \\ If(\alpha \geq 360)\ \ then\ \alpha = \ \alpha - 360° \end{Bmatrix} \quad \textbf{(4.6)}$$

The figure below shows the recognition concept of building with FOV.

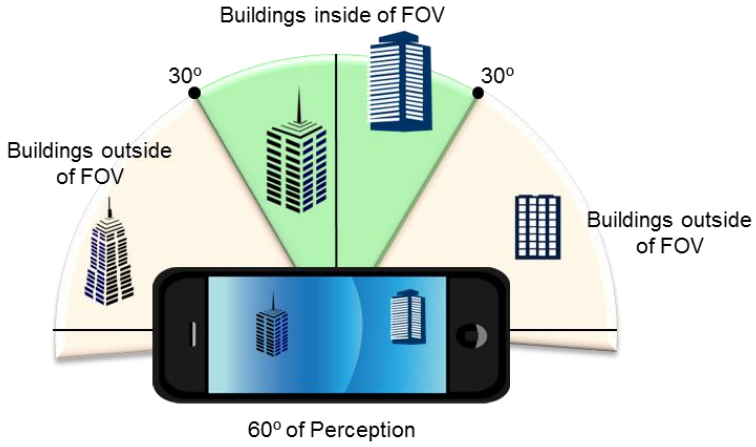

Fig. 4.9 Scheme of the FOV concept, which will be implemented on the Augmented Reality application.

4.4. Schematic Diagram of Application

At first, some key problems of augmented reality applications should be discussed. Certain issues of AR applications can be detected in scope of concurrent applications.

- There is lack of experience with augmented reality user interfaces.
- Semantics of environments artifacts is required; AR will require not only information on geometry and appearance but also meaning.
- Improvement of tracking and mapping environment is necessary.

Moreover, a just few AR applications use the service based approach that is suitable for effective data management. Therefore, data management such as updating or using user data depends on the vendor of the application.

This goes hand in hand with another already mentioned issue of contemporary AR application architecture, whole application runs on the user server device. Data are stored within the application, or the application is connected to single remote database provided by the application vendor. An example of such AR application architecture is shown in Fig. 4.10

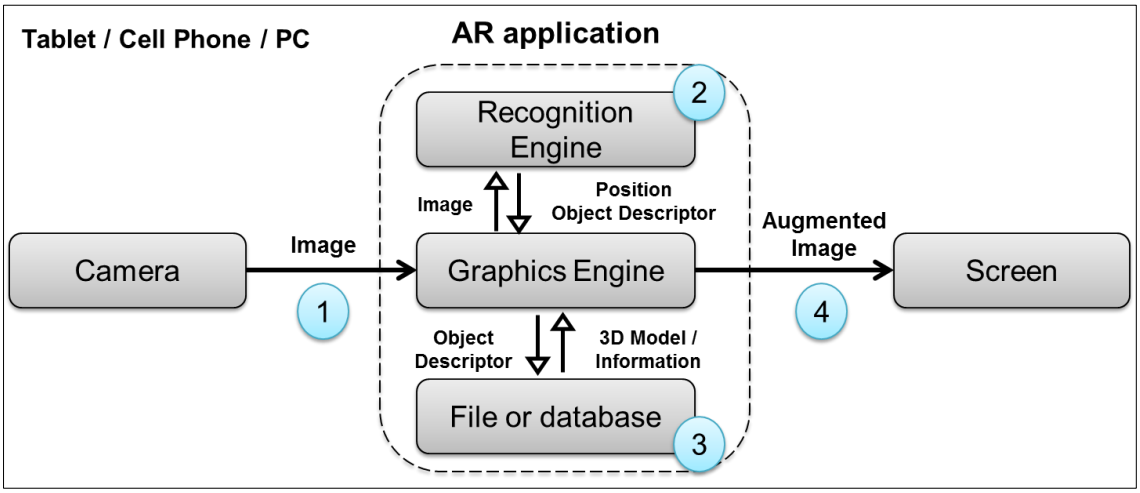

Fig. 4.10 Common AR application architecture.

The first problem the thesis focused on is the open and sharable data structure. As been explained above, both LBS and AR applications store information linked to locations. Therefore, the solution might be based on open standard that is used for LBS data storage – the geodatabases. Geodatabases are widely used, well documented and opened. Communication standards for querying databases, such as widely-used geospatial standards Web Map Service (WMS) or Web Feature Service (WFS) are available and also well

documented. Geodatabases bring the advantages of client-server architecture as well as the possibility to use data from more sources. The solution assumes usage of appropriate OpenStreetMap database, in this case Overpass.

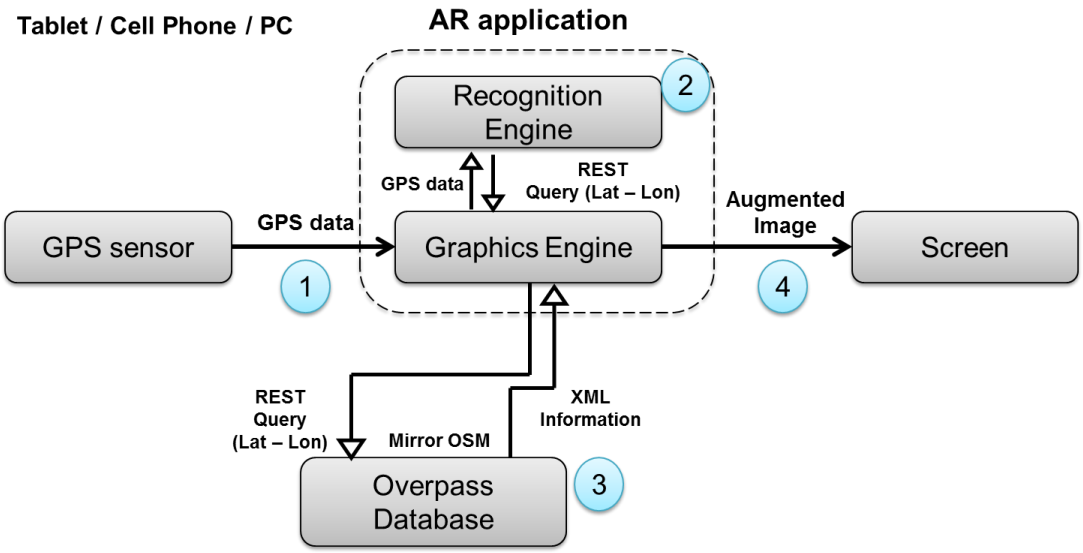

Fig. 4.11 Selected AR Application Architecture

Concerning the AR application architecture, will take the form showed at the figure 4.11, where the camera concept is omitted and replace by only the GPS sensor data; the location is analyzed by the Recognition Engine, which follows the prerequisite: If the location is outside the bound limits, do a new query to the specific mirror get the information XML. Therefore generate through graphics engine the augmented image or model to be displayed on the device screen.

4.4.1. System Design

The entire design of the application will take in account Location based services over Mapping services and Augmented Reality. The functional diagram shown in figure 4.12 summarizes the entire process of application coupled with a local downloader technique.

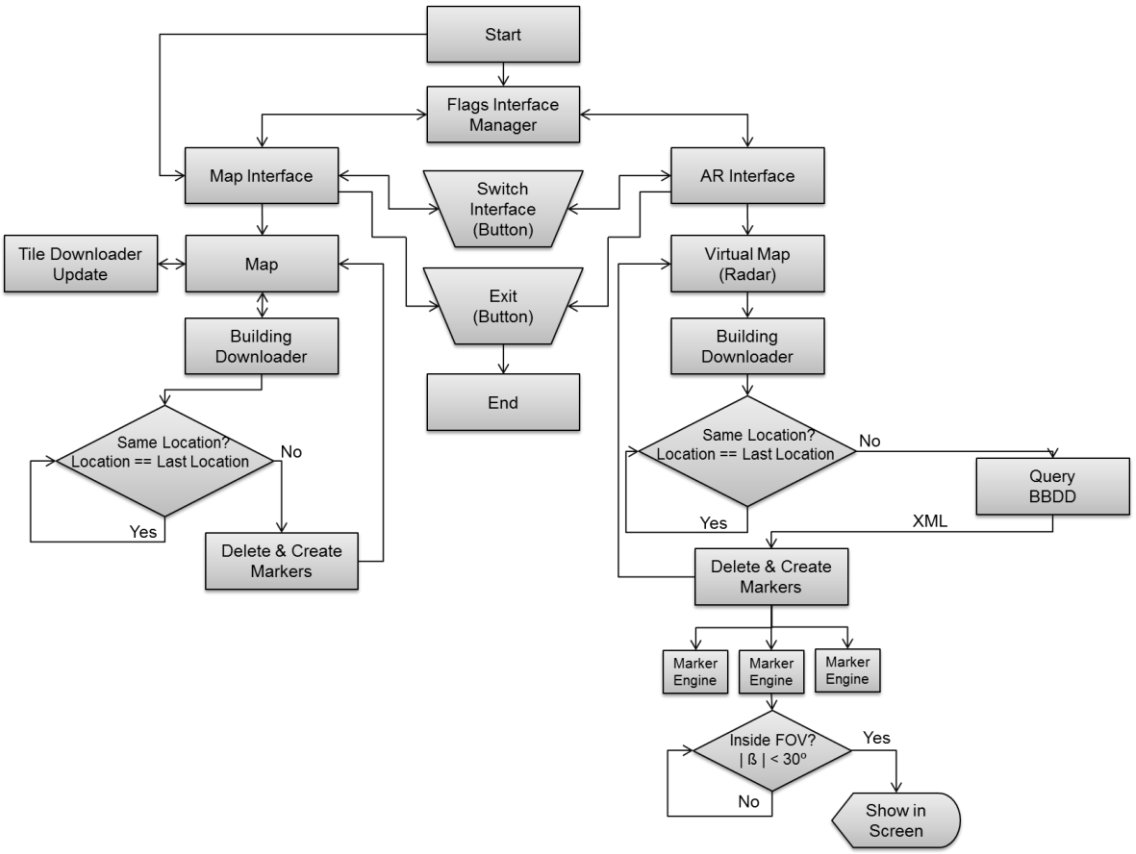

Fig. 4.12 Functional Diagram of the entire Application.

The first step of the application system is loading the flags used by the common interface to their default values; at the same time the map interface is initialized to create the map object which contains the tiles and markers.

The map is automatically updated by the tile downloader module; this module is always alert to any realized event on the interface, when get an event the tiles are update. Moreover the next step of the map is the building downloader functions; it works on the concept of bounded areas, if the location lecture of the GPS is inside of bounded area, the downloader keeps waiting until the location will be outside of area. See figure 4.13 to understand the concept of bounded area.

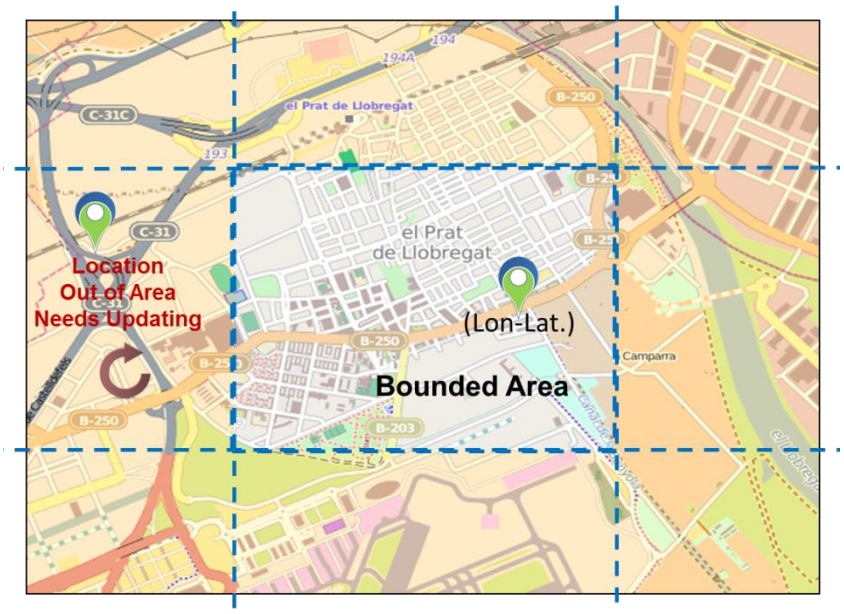

Fig. 4.13 Concept of Bounded area, if the location is outside, the markers need to be updated.

Therefore the markers are updated when the location is outside of range; a new petition is sent to Overpass API and the XML answer is processed to create the new markers on the map.

Now, there is another element in the interface of the application, it is common between both interfaces, it's the switch button which does the function of switch the mode of interface. Hence, it gives access to the AR interface and its functionalities.

The AR part of the application, works exactly same as the Map part. The container of the markers in this case is the virtual map, represented as radar on the interface. When a change of location is detected, the system does a new petition to BBDD of Overpass, the obtained information is processed and the 3D models are created but they are not showed in the interface.

3D models have an engine to calculate when they should be showed in the screen. If the orientation of user is inside of the 3D Field of view (FOV) of the model, the engine enable instantaneously the 3D Model on the screen, merging the model with the camera image.

The 3D models and markers of both interfaces can be related through the flags interface manager, it can save the flag of the 3D model to look for it in the map interface.

Finally to close the application, the interface has a button with this functionality.

4.4.2. Device Selection

For the purpose of this study, a number of mobile smart-phone devices were utilized as tools of development platform. The chosen development environment for this thesis is Android SDKs. In this section a number of Android devices are listed, because the application will run on them, and it's another way of evaluated the AR application, to know how works on different devices. The table 4.14 shows the comparison between these mobile devices.

Table 4.14 Comparison of three High-end Smart-phones.

.

Devices			
	Samsumg Galaxy S3	**Huawei Ascend G510**	**ZTE Blade Apex**
			
CPU	Quad-core 1.4 GHz Cortex-A9	Dual-core 1.2 GHz Cortex-A5	dual-core Qualcomm MSM8930 Snapdragon 1.2GHz
GPU	Mali-400MP4	Adreno 203	-
RAM	1GB	512 MB	1GB
Screen Resolution	1.280 x 720 pixels 4,8" inches	480 x 854 pixels 4.5" inches	540 x 960 pixels 4.5" inches
Camera Resolution	3264 x 2448 pixels	2592 x 1940 pixels	2592x1944 pixels
3-Axis Accelerometer	Yes	Yes	Yes
GPS	Yes, with A-GPS support and GLONASS	Yes, with A-GPS support	Yes, with A-GPS support
Compass	Yes	Yes	Yes
Android API	4.3	4.1.1	4.1.2

5. IMPLEMENTATION AND TESTING

The aim of this chapter is to explain the implementation of the application system. Therefore, explaining the code of every functionality part of the program.

The first step to create a new application is follow a design diagram; this diagram was specified in the last chapter, see figure 4.12, but also it is needed specify the platform where the application will be developed. Hence, the work platform, in this case Unity 3D, has to be configured before programming the application.

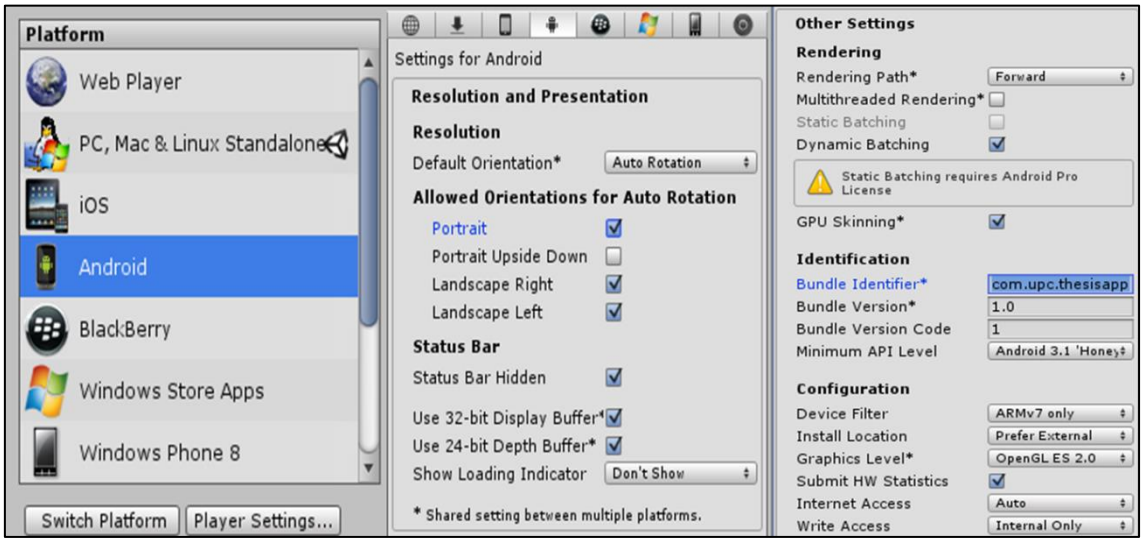

Fig. 5.1 Unity Project Configuration – Selected options for the project on Unity.

To configure the unity project, the Android platform should be chosen; once has been selected the platform, the needed options for the project are checked, in this case the screen orientation will be auto-rotation, allowing the rest type of orientation, less portrait upside down. Also the used buffer options are checked too, without forgetting to select most important part, the selection of Android API.

For developing the application, the project use API level: Android 3.1 “Honey”; and finally internet access is set to auto, that means the application will have always connection automatically when the devices is connected to internet.

5.1. OpenStreetMap Implementation

The fact to create an application that contains a map service, it implies following the same model or structure that map services use, that is, a map with functionality of container and tiles such stored objects on the map, it works

same for markers. Given that, it needs a map able to store objects and maintaining the relation with map position.

To maintain the geographic position with map projection or vice versa, the Proj.Net library is declared at the beginning of the class, see table 5.1.

Table 5.1 Illustrates the code portion that creates the Class Map and usage of Proj.Net library (bold).

```
using ProjNet.CoordinateSystems;
using ProjNet.CoordinateSystems.Transformations;
using ProjNet.Converters.WellKnownText;

public class Map : MonoBehaviour
{
        private static Map instance = null;
        public static Map Instance
        {
                get
                {
                        instance = FindObjectOfType(typeof (Map)) as Map;
                        var go = new GameObject("[Map]");
                        instance = go.AddComponent<Map>();
                        return instance;
                }
        }

        private Map()
        {
        }
}
```

The coordinate systems, transformation and converters WKT are essential to project the coordinates over the map, because the value of the sensor is always in format WGS84 system and to project the coordinates it must be converted to EPSG900913; table 5.2 shows the initialized variables.

Table 5.2 Initialization of system variables coordinate projection.

```
String wktEPSG900913 = WKT projection (see subchapter 4.2.1. – Table 4.11).

   CoordinateTransformationFactory ctFactory;
   ICoordinateSystem              epsg900913;
   ICoordinateTransformation      wgs84ToEPSG900913;
   IMathTransform                 wgs84ToEPSG900913Transform;
   IMathTransform                 epsg900913ToWGS84Transform;

      // initialize the coordinate transformation
      epsg900913 = CoordinateSystemWktReader.Parse(wktEPSG900913) as
                        ICoordinateSystem;
      ctFactory = new CoordinateTransformationFactory();
      wgs84ToEPSG900913 = ctFactory.CreateFromCoordinateSystems(
                              GeographicCoordinateSystem.WGS84, epsg900913);
      wgs84ToEPSG900913Transform = wgs84ToEPSG900913.MathTransform;
      epsg900913ToWGS84Transform = wgs84ToEPSG900913Transform.Inverse();
```

The IMathTransform class makes any transform of the project will be translated on the correct position of the map; it will be used for positioning the markers on the map or center the map.

The attribute of a map are zoom and center location of the map, these variables are also declared in the class map.

Table 5.3 Zoom and Center location variables of the Map class.

```
float    currentZoom = 16.0f;
float    maxZoom = 18.0f;
float    minZoom = 1.0f;

double[] centerEPSG900913 = new double[2];
double[] centerWGS84 = new double[2];

centerEPSG900913 = wgs84ToEPSG900913Transform.Transform(value);
centerWGS84 = epsg900913ToWGS84Transform.Transform(centerEPSG900913);
```

With these parameters the tiles can be downloaded through the OpenStreetMap API. To continue the process is required another object to manage the set of tiles, for this reason is created a second class, Layer class. The created structure is shown in the following figure.

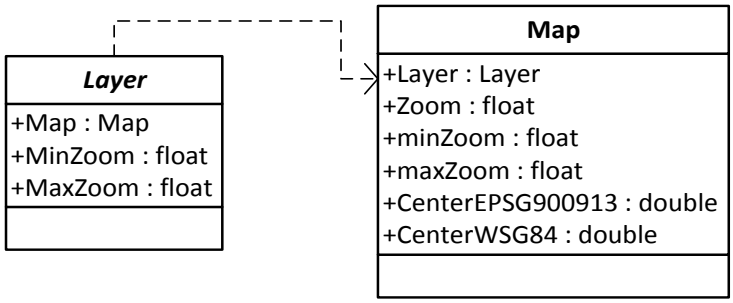

Fig. 5.2 Relation between the Layer and Map Classes.

But the Layer class is abstract, works such the base class for the following class, the class TileLayer; the TileLayer class is the real engine which the tiles are created to generate the entire map of the application. At the same time TileLayer is also an abstract class; the WebTileLayer completes functions are not implemented in TileLayer, functions such request the tiles or cancel this tile request. Finally the last missing class is the OSMTileLayer which completes the functions of get the next tile to be request according to OSM API.

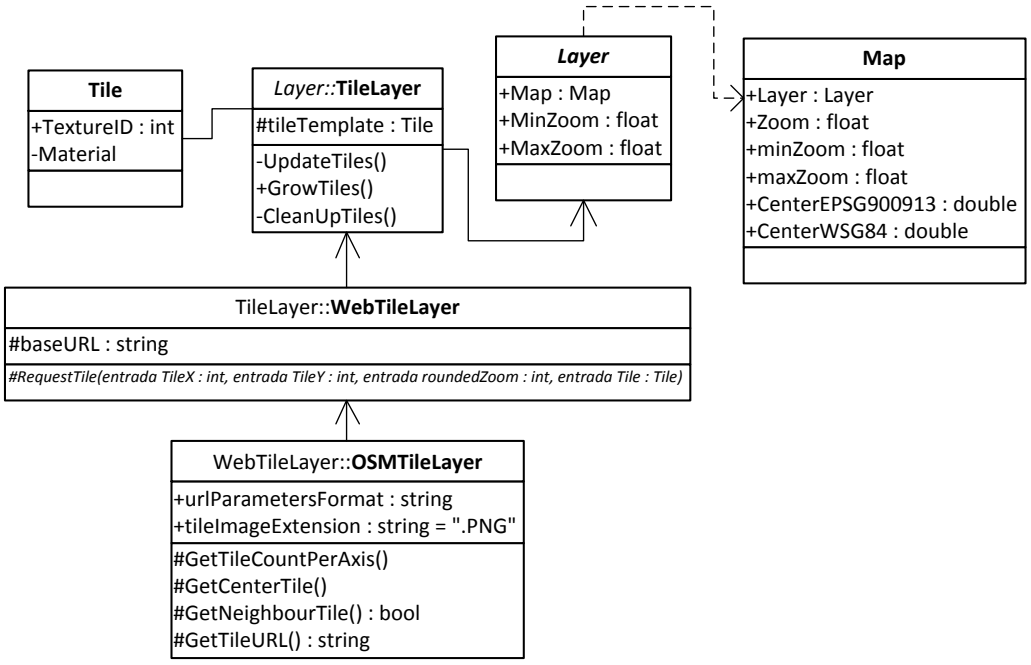

Fig. 5.3 OpenStreetMap Implementation – Significant methods and variables of each class.

The most important part of the previous diagram is the OSMTileLayer class because determines the tiles number of the map according to the actual zoom

of map. The OSM API, provide the formula to calculate the number of tiles. See formula 5.1 and its implemented code.

$$2^n \times 2^n \text{ tiles} \quad \textbf{(5.1)}$$

Table 5.4 Method to establish the number of tiles and getting its URL.

```
Protected override void GetTileCountPerAxis(out int tileCountOnX, out int
tileCountOnY)
{
    tileCountOnX = tileCountOnY = (int)Mathf.Pow(2, Map.RoundedZoom);
}
protected override string GetTileURL(int tileX, int tileY, int roundedZoom)
{
     Return String.Format(Path.Combine
                    (BaseURL,URLParametersFormat).Replace("\\", "/") +
                    TileImageExtension, roundedZoom, tileX, tileY);
}
```

Moreover, the request of tiles is managed through the WebTileLayer class; it implements two methods, one to request tiles and canceling the same. Table 5.5 illustrates the code portion that does the request function.

Table 5.5 The code portion of WebTileLayer that request the tiles.

```
protected override void RequestTile(int tileX, int tileY, int roundedZoom, Tile
tile)
{
        TileDownloader.Instance.Get(GetTileURL(tileX,tileY,roundedZoom),tile);
}

protected override void CancelTileRequest(int tileX, int tileY, int
roundedZoom)
{
        TileDownloader.Instance.Cancel(GetTileURL(tileX,tileY,roundedZoom));
}
```

Finally, the function of the TileLayer class is working as real container of tiles, although the number of tiles is calculated by 2^{zoom} tiles per axis (Formula 5.1), TileLayer restrict to 100 tiles, and download only the requires tiles. To do it, creates a plane object which works such a frustum, used as volume rendering display a camera.

The code is always asking by the next neighbour, if the tile has a neighbour, download the new tile since to complete the plane object.

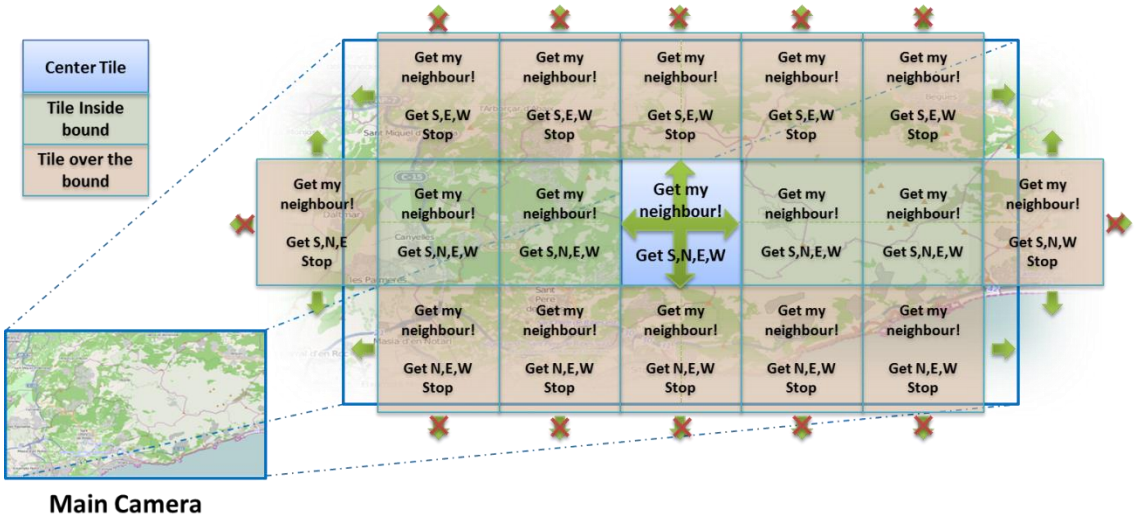

Fig. 5.4 Download System that TileLayer class uses.

Through the last Diagram of implementation (Fig. 5.3.), the concept of Map on the interface becomes real; the following table shows the way of calling the map.

Table 5.6 Start Method of the Basic Map Interface.

```
private Map map;

private void Start () {
        map = Map.Instance;
        map.CurrentCamera = Camera.main;

        OSMTileLayer osmLayer = map.CreateLayer<OSMTileLayer>("OSM");
        osmLayer.BaseURL = "http://a.tile.openstreetmap.org/";
}
```

With the implementation of the map based on OSM finish; the interface implementation is converted in the main objective.

5.2. Map Navigation Service Implementation

The implemented services in the application are GPS navigation function and Static Map service; in both functions the map has markers which show interest places. The diagram of this part is shown in figure 5.5.

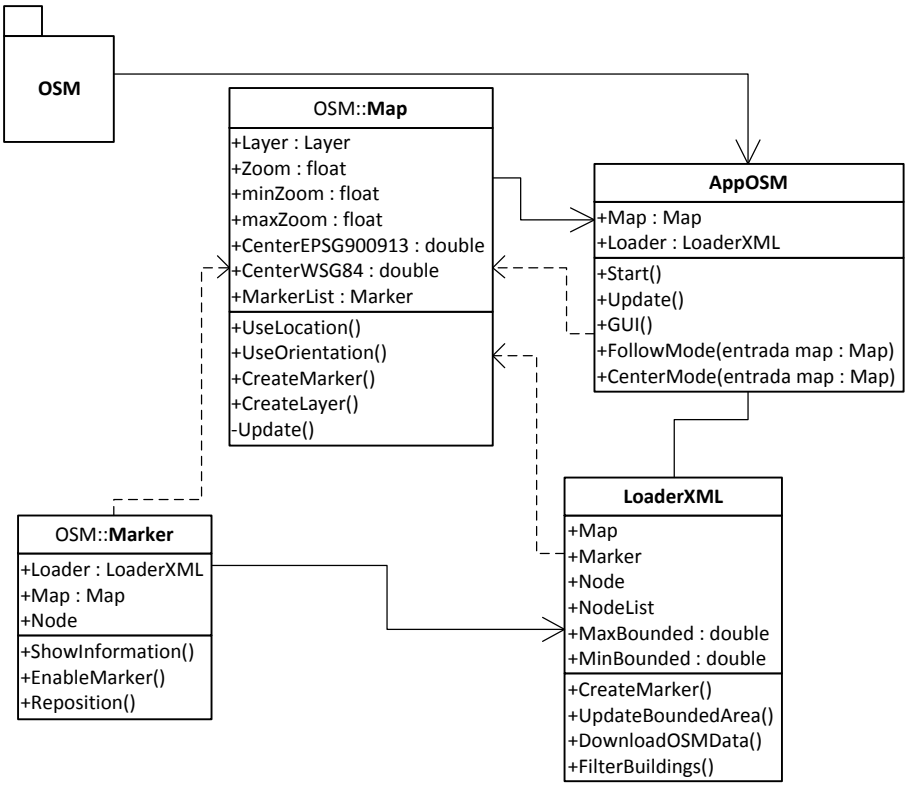

Fig. 5.5 Diagram of the implemented system; a new class is created to manage the download of Markers on the map: LoaderXML class.

The Map class has new implemented methods to give solution to the new services. UseLocation () is a method that enables the location services of the devices, it allow getting the information of the input sensor, in this case GPS sensor and its location data.

Table 5.7 Implementation of UseLocation method.

```
private bool   useLocation = false;
public bool    UseLocation
{
    get { return useLocation; }
    set
    {
        if (useLocation == value)
                return ;
        useLocation = value;
```

```
      if (useLocation)
      {
          if (UnityEngine.Input.location.isEnabledByUser
        &&(UnityEngine.Input.location.status ==LocationServiceStatus.Stopped
         || UnityEngine.Input.location.status == LocationServiceStatus.Failed))
       {
                UnityEngine.Input.location.Start();
        }
      }
      else
      {
        if (UnityEngine.Input.location.isEnabledByUser
        &&(UnityEngine.Input.location.status==LocationServiceStatus.Initializing
        || UnityEngine.Input.location.status == LocationServiceStatus.Running))
        {
                UnityEngine.Input.location.Start();
        }
       }
   }
}
```

The UseLocation function is a Boolean that it's set true when the user press the button with that functionality. The method, its code is illustrate at table 5.7, verify if the Location device was enabled by user, if it's true, the method checks the location service status of the device and start it, allowing the GPS reading.

The UseOrientation function of Map class works exactly same, with the only added of allowing to get the data of compass sensor. Compass data is used to rotate the main camera of the application, to give the sensation of movement towards the direction the person is looking.

Table 5.8 This code is inside the update function,

```
if (useOrientation && cameraFollowsOrientation){
  if (lastCameraOrientation == 0.0f) {
    currentCamera.transform.RotateAround(Vector3.zero,Vector3.up,heading);
    lastCameraOrientation = heading;
  } else {
    float cameraRotationSpeed = 1.0f;
    float relativeAngle = (heading - lastCameraOrientation) *
                          cameraRotationSpeed * Time.deltaTime;
    currentCamera.transform.RotateAround(Vector3.zero,Vector3.up,
     relativeAngle);
  }
```

Moreover, to show markers on the map, LoaderXML class has been implemented. LoaderXML has the functionality of downloading through queries needed information of node to create the markers. LoaderXML implements four important functions:

First function updates the bounded area, it is useful to get markers information, the concept of bounded area was explained at subchapter 4.4.1, see Fig. 4.13, to obtain the true wanted area; the function works under the following formula.

$$Lon = GPS_{Lon} \pm 1433.2 \cdot e^{-0.693 \cdot Zoom} \quad \textbf{(5.2)}$$

$$Lat = GPS_{Lat} \pm 569.86 \cdot e^{-0.693 \cdot Zoom} \quad \textbf{(5.3)}$$

Second Function implemented is Download OSM data, this create the petition of information and receives the answer in XML format from Overpass server. This function is called when the GPS data is outside of bounded area. Its implementation is shown in table 5.8.

Table 5.8 Function used to download OSM data.

```
private IEnumerator DoWWW(string latlonURL, string specificbuild){
  string urlweb = "http://overpass-api.de/api/xapi?node";

  urlweb = urlweb + latlonURL + specificbuild;
  WWW www = new WWW(urlweb);

  yield return www;

  if (www.error == null && www.text.Contains("404 Not Found") == false
      && www.text.Contains("Error in") == false){
       //XML DOCUMENT
       XmlDocument xmlDoc = new XmlDocument();
       xmlDoc.LoadXml(www.text);

       XmlNodeList elemList = xmlDoc.GetElementsByTagName("node");

       foreach (XmlNode xmlNode in elemList){
          CreateMarker(xmlNode)
       }
  }
}
```

The downloading function call inside its code another function, this is a dedicated function to create new markers for each node in XML object. When makers are created, their nodes indicate which kind of building pertains. It's useful when users utilize filters in the interface of application.

The loaderXML has a small class dedicated to managing tags, if they are activated or not, this class is called Tags, which only has the following implemented code.

Table 5.9 Implemented Methods in Tags Class.

```
void Start () {
        KeyList.Add(new Key ("Aeroway","aeroway",true));
        KeyList.Add(new Key ("Amenity","amenity",true));
        KeyList.Add(new Key ("Building","building",true));
        KeyList.Add(new Key ("Historic","historic",true));
        KeyList.Add(new Key ("Leisure","leisure",true));
        KeyList.Add(new Key ("Office","office",true));
        KeyList.Add(new Key ("Public Transport","public_transport",true));
        KeyList.Add(new Key ("Railway","railway",true));
        KeyList.Add(new Key ("Shop","shop",true));
        KeyList.Add(new Key ("Tourism","tourism",true));
        }

public bool isActivatedBuilding(string typeofbuild){
        bool auxbool = false;
        if (!typeofbuild.Contains("other")){
        Key auxkey = KeyList.Find(x => x.keyCode.Contains(typeofbuild));
                        auxbool = auxkey.keyActivated;
                }
                return auxbool;
        }
```

The Start function adds to the list all the key words used such filter and its state, in that case all are enabled (true). The isActivatedBuilding function receives the type of build as parameter and checks its state, returning a Boolean with its respective value.

Finally the LoaderXML class will also be used in the augmented reality interface, and their variables will be managed by another class called SceneManager. This last class is used for the two services, map services and AR service.

5.3. Augmented Reality Service Implementation

The implementation of AR services has the same approach than Map services; the fact the entire diagram is equal without the class of download tiles. The diagram is illustrated in figure 5.6.

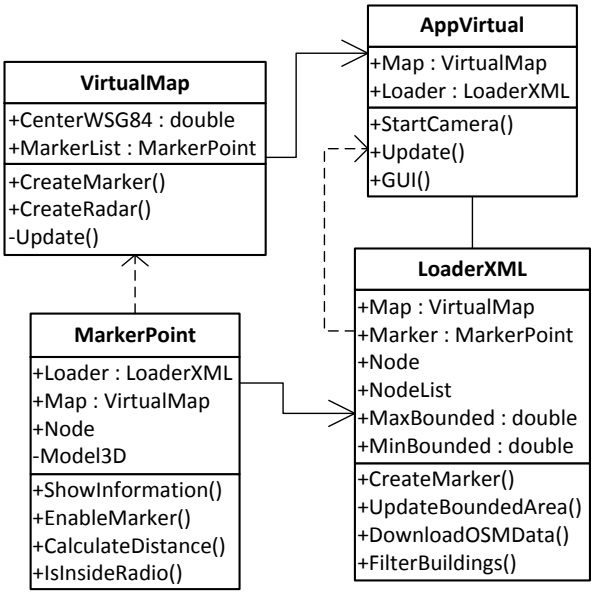

Fig. 5.6 Implementation of AR services.

The virtualMap class is only a container of marker transforms, it doesn't require any tile, the map is represented as Radar, where it projects buildings in form of points, then buildings are near, are shown in the radar.

Table 5.10 Implementation of method that creates the Radar on screen.

```
void OnGUI() {
        angle = Input.compass.magneticHeading;
        Matrix4x4 matrixBackup = GUI.matrix;
        //updating radar's position
        GUIUtility.RotateAroundPivot(angle, pivot);
        GUI.DrawTexture(rect, texture);
        GUI.matrix = matrixBackup;
        GUI.skin = skinScroll;
        GUI.Label(rectlabel1, "Radius");
        radioarea = GUI.VerticalScrollbar(rectScrollbar,radioarea,200.0f,
                        1200.0f, 100.0f);
        radioarea = Mathf.Round((float)radioarea * 10f) / 10f;
        GUI.Label(rectlabel2, ""+ radioarea+" m.");
        }
```

The code above use one property of Unity able to rotate a picture around the pivot, this property is RotateAroundPivot from GUIUtility library. It uses the heading of the compass to rotate the image and to give the sensation of real compass.

The AppVirtual class has the function to create the displayed camera in the interface, this class works basically to implement the interface of the AR services, also create the button to switch between interfaces, Map service interface and AR interface; and the option to exit from application.

Table 5.11 Camera code in the App Virtual.

```
float height = Camera.main.orthographicSize * 2.0f;
float width = height * Screen.width / Screen.height;
transform.localScale = new Vector3(width, height, 0.1f);

WebCamDevice[] devices = WebCamTexture.devices;
deviceName = devices[0].name;

webCameraTexture = new WebCamTexture(deviceName, Screen.width,
Screen.height);

renderer.material.mainTexture = webCameraTexture;

// Starts the camera
webCameraTexture.Play();
```

The code of table 5.10 starts the camera of the device, it calculate the relation between the main camera of scene and the screen size and this relation is used to modify the size of texture where the image of camera is shown, then the code looks for the webcam device and project their image on the resized transform.

MarkerPoint is an important class in the AR system, every Marker manage itself the moment when it has to appear on the screen and if user is looking to one direction, the engine creates an vector with same orientation respect to the marker of the building, if the angle formed is near numerically between the range of field of vision that user has, the marker is shown at the screen. The implementation code of this engine is shown in the table 5.11. See figure 4.8.

Table 5.12 Implementation of the Marker engine.

```
private void CalculateAngle()
```

```
        {
    float difLon = ((float)coordinatesWGS84[0]-
UnityEngine.Input.location.lastData.longitude ); //x
    float difLat = ((float)coordinatesWGS84[1]-
UnityEngine.Input.location.lastData.latitude );  //y

    float ang = 0;

    if (difLat>=0){
        ang = Mathf.Atan2 (difLat,difLon);
        angleLocation = ang * Mathf.Rad2Deg;
    }else{ // diff latitude = 0
         ang = Mathf.Atan2 (difLat,difLon);
        angleLocation = 360 + (ang * Mathf.Rad2Deg);
    }
}

private void CalculateAlpha(){
    angleAlpha = 450.0f - angleLocation;
    if (angleAlpha>=360.0f){
        angleAlpha = angleAlpha -360.0f;
    }
}
```

Finally, with the implementation of Augmented Reality service, the implementation of the application is complete but there is still a class that manages the options of interface for the entire application, it is the SceneManager class. The manager class has neither implemented methods; it only keeps main variables of interfaces and marker identifiers. The complete diagram of the implementation is shown in the last figure, see Fig. 5.7.

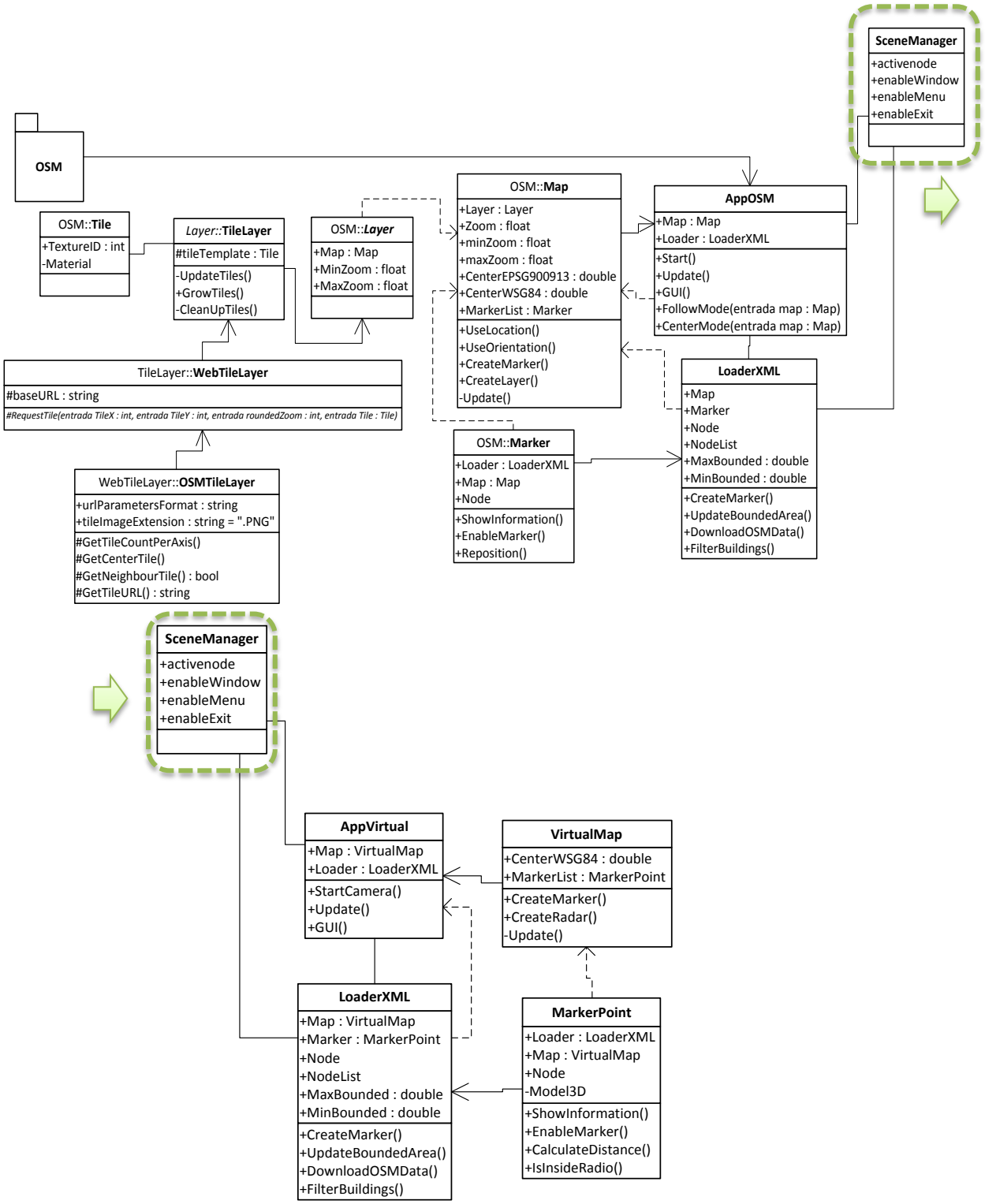

Fig. 5.7 Implementation Diagram of the design application.

5.4. Testing the application with real users

To evaluate the application, the program is delivered to a group of people which experiment, using and trying to understand the application functions. Moreover, they will answer a series of prepared questions with the purpose of analyze the application from their views.

The test consists of ten questions, these are the following:

The questions from the one to number four will answer and provide the background of the concepts that have people about GPS and Augmented Reality.

1. Indicates, how often you use the GPS?
2. Please indicate how often you visit the website of google maps or any other map services web?
3. Do you know or ever used an application of Augmented Reality?
4. Do you Know or have knowledge of what is Google Glass?

From number five to ten will answer specifically questions about the program.

5. What do you like most of the program that you just used?
6. If you need to find a building, would you rather use the map or the camera of the application?
7. Do you think that the information shown on the map is appropriate?
8. Do you think that the information contained in the shown buildings on the camera is adequate?
9. Do you have trouble understanding and using the program?
10. Can you give a punctuation of useful to the program?

The questions number seven and eight have the same information, and it's another way to say which one of both services do you prefer? The results of all these questions are evaluated in the next chapter.

Finally to perform these tests, it is necessary that the user must have a mobile with the following features:

- Android 3.1 "Honey" or higher.
- Internet connection
- Mobile phone that has a compass and GPS sensor.
- Enable the option of allow installation of non-Market Apps, from application settings in the mobile device.

6. RESULT AND DISCUSSION

6.1. Results of Implemented Application

After a long implementation, the obtained application can be appreciated through its finalized interface and its finished functionalities. The following figures show the application running on a Samsung device.

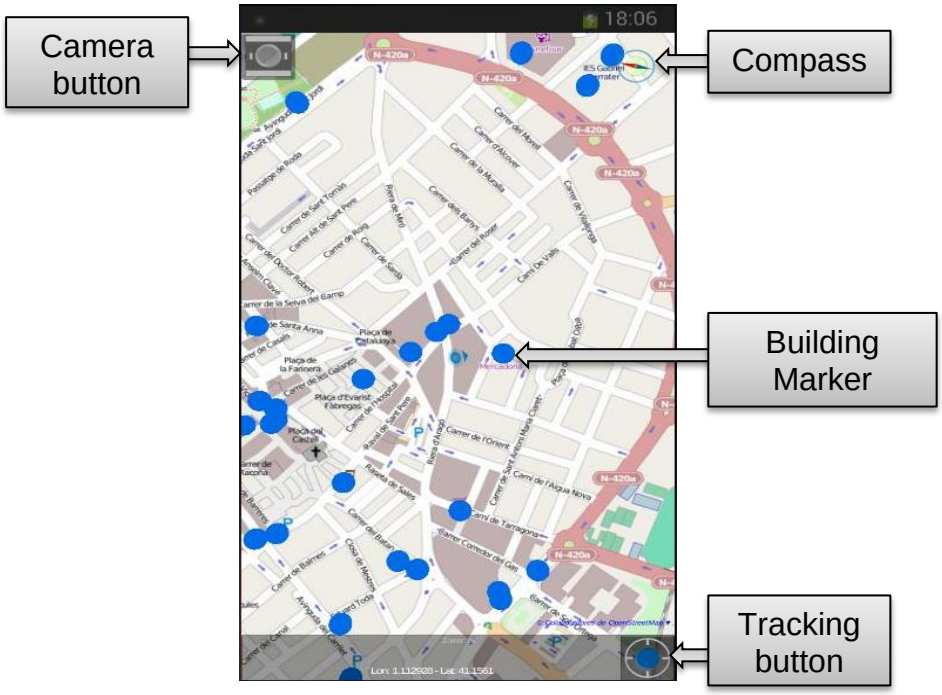

Fig. 6.1 Interface of the application when it is initialized – Portrait Orientation.

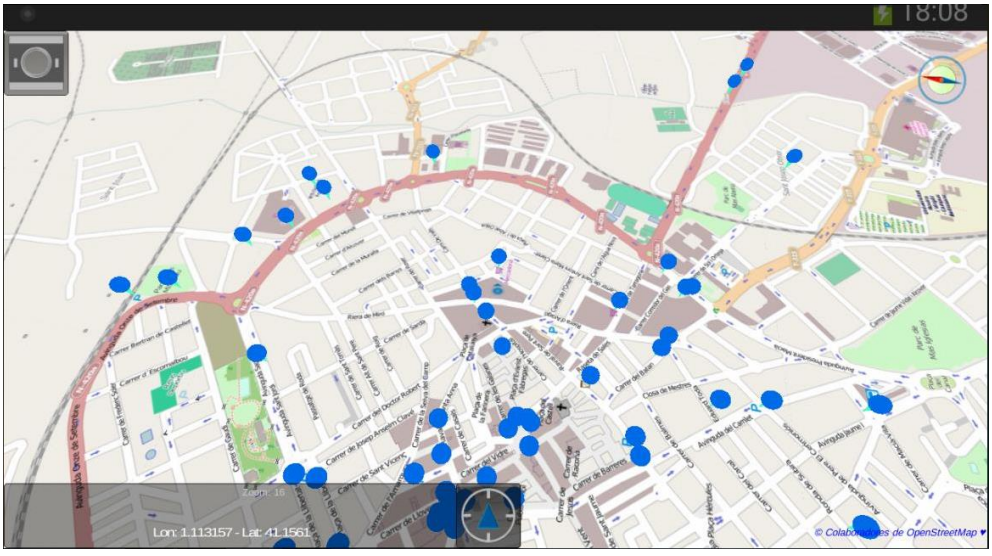

Fig. 6.2 Interface of the application when it is initialized – Landscape Orientation.

The figure 6.1 and 6.2 show as viewed already deployed application. When the application is initialized the program follows the device orientation, and adapts the map to the interface. Furthermore it can be seen that the application starts running the map interface instead of the AR interface.

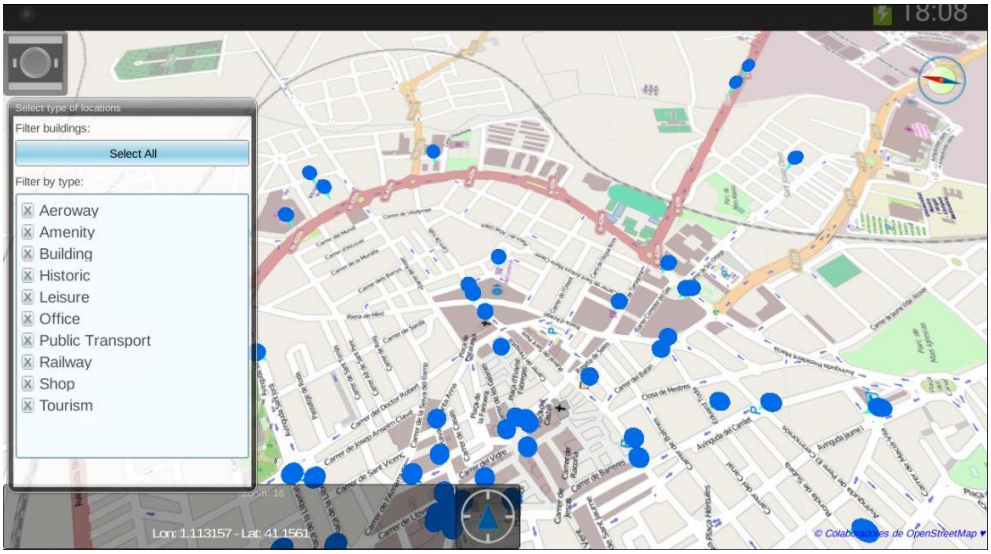

Fig. 6.3 The application shows the filter implemented by types of buildings.

The blue points on the map are markers that represent buildings with their respective positions. These points can be filtered through the filter of the application; it can be accessed via the menu button on the mobile.

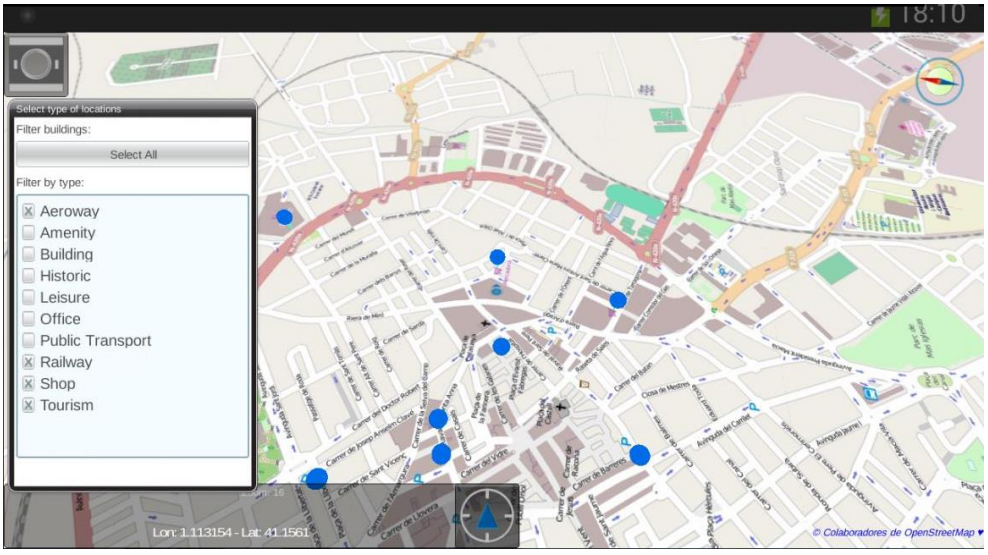

Fig. 6.4 The filter of application with some options disabled.

The figure 6.4 compared to the fig. 6.3 shows the filter used. There are many disabled options in the filter, it can be appreciated in the same figure through the number of blue points that are shown, in this case there is less blue points than before in figure 6.3. However, if the user wants to know if the filter works, the user can access the buildings information and check. The following figure shows some markers with its activated information window.

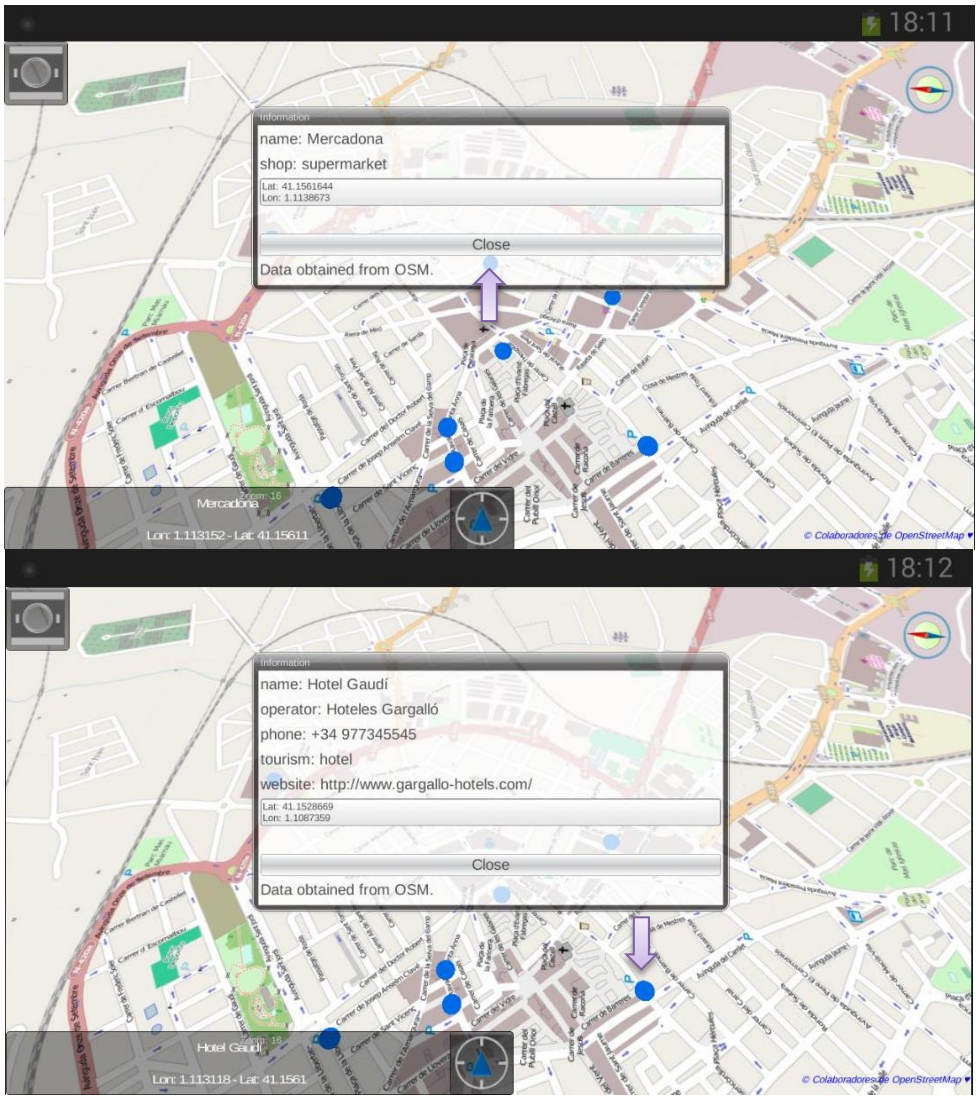

Fig. 6.5 Information windows of the building markers.

One of the requirements of the thesis was designing and implementing a map service using mapping service of OpenStreetMap; this requirement can be assumed as complete. Another requirement was to carry out the implementation of an augmented reality interface; to see or access the AR interface, it can be accessed via the camera button of the map interface. The following figures show the AR interface.

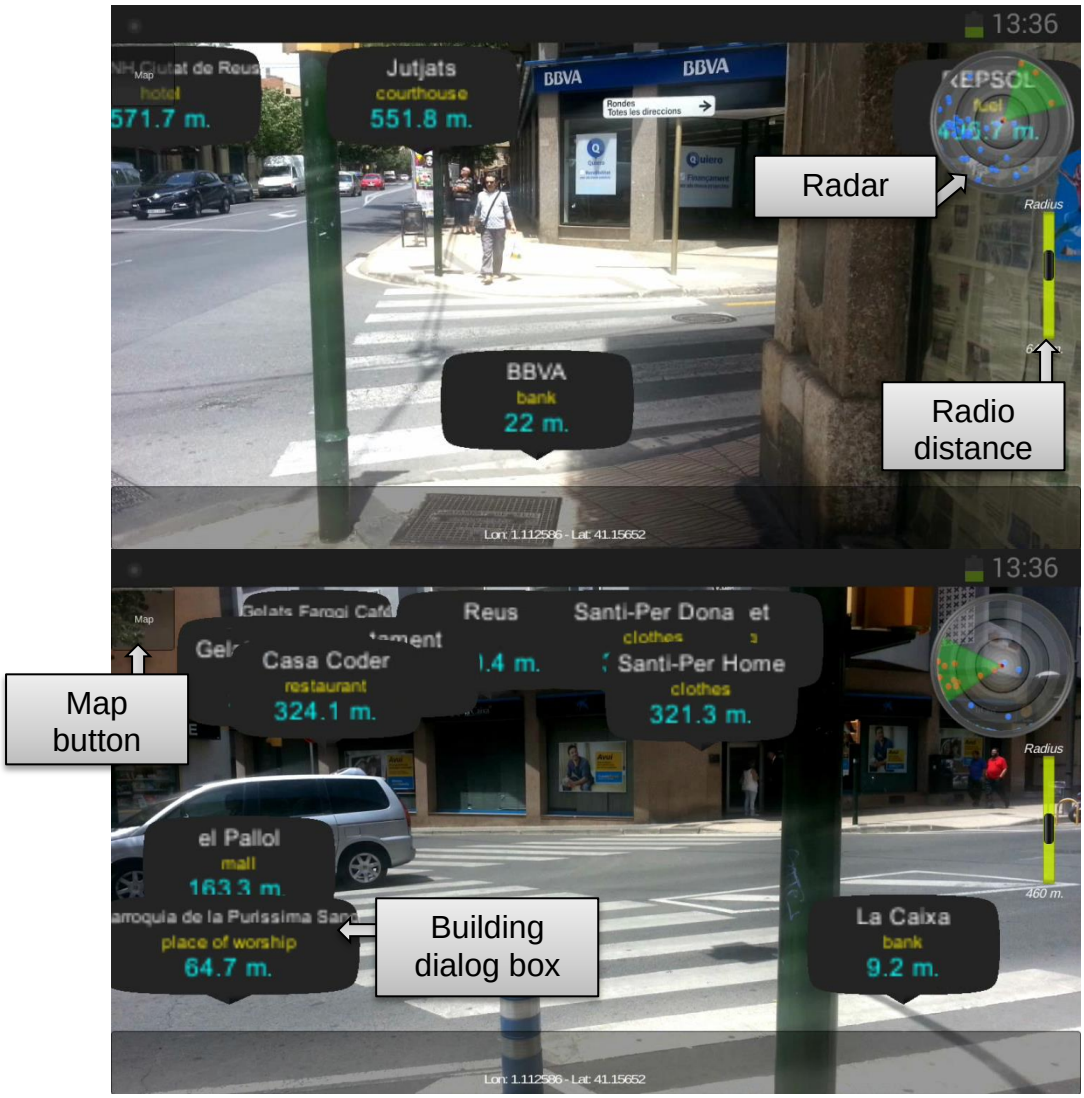

Fig. 6.6 Examples of Augmented Reality interface.

When switching to the AR interface, the data are displayed correctly; models of dialog for buildings show the correct information, however, it have to denote such as it was designed, distance and orientation of the buildings is updated according to the position of the GPS, so if the reading of GPS is outdated or the sensor does not have much precision, distance and orientation may show no very accurate information.

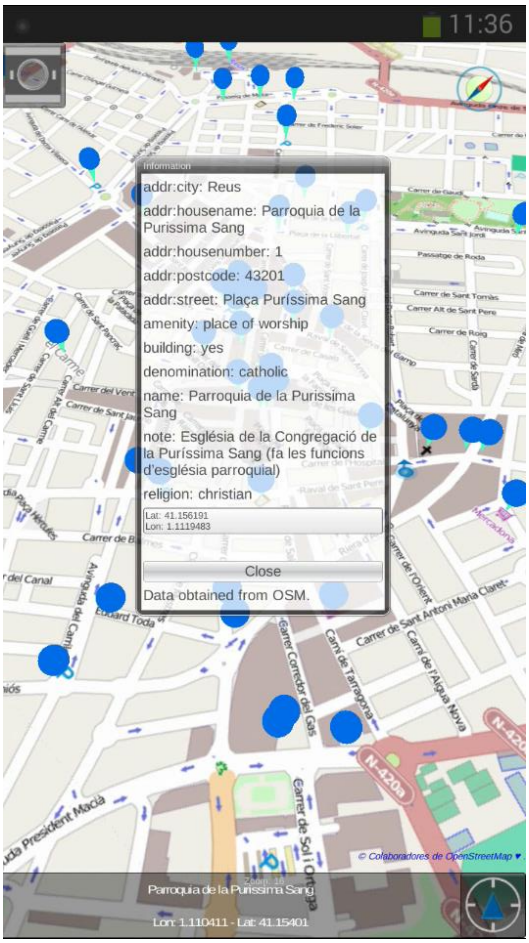

Fig. 6.7 Information of buildings accessed through a touch event – Map Interface.

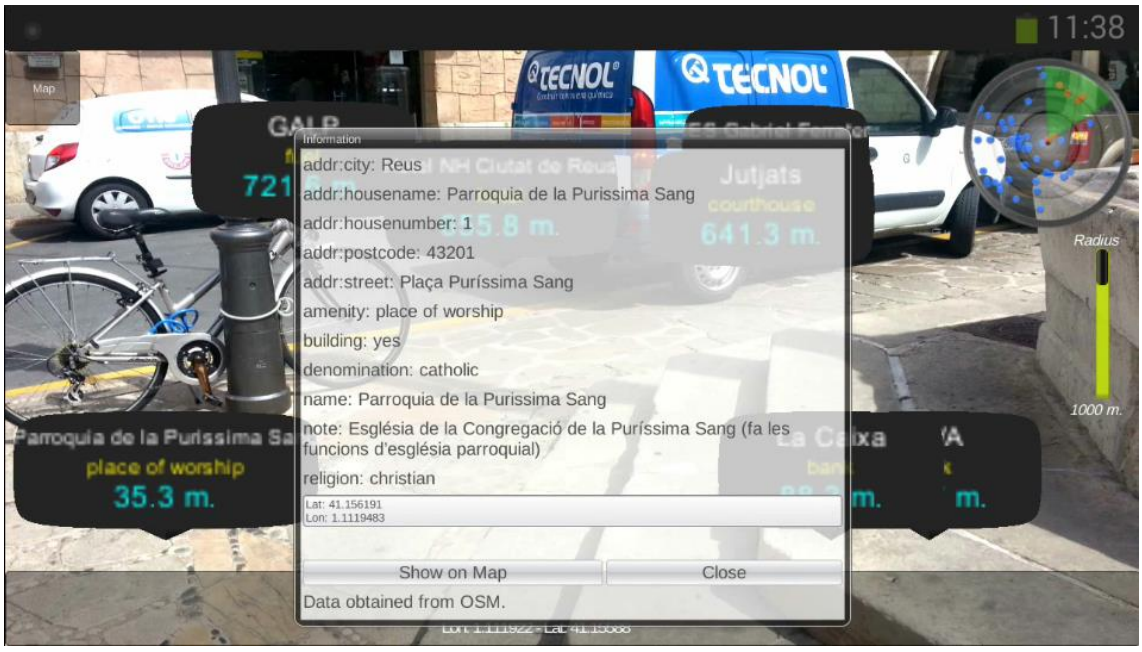

Fig. 6.8 Information of buildings accessed through a touch event – AR Interface.

The information of the buildings was another requirement of design for the application; both interfaces show the same data but in different format information. It can be seen in figures 6.7 and 6.8 that the information is the same; therefore, in this aspect the application satisfies the required conditions.

For the three mobile devices used in the implementation, all the features of the application worked correctly, nevertheless the ZTE device didn't return the exact position; this is probably due to not having a dedicated GPU then the CPU collapsing or just the GPS sensor does not have enough precision.

In summary, the system requirements that were established in subchapter 3.3 have been completed correctly; the application has two interfaces, one with the functionality of the integrated mapping services and another interface based on augmented reality. Both interfaces show the information of near buildings with the same data and finally the entire implementation was realized with Unity 3D platform.

6.2. Answer to the Test Questions and Analysis of the Results

Number of people who have tested the program was fifteen; these fifteen users have done a test consisting of ten questions. The results of these questions include age differentiations that are also reflected in the following charts.

The first four questions are a thermometer to measure the preferences that users have on GPS devices and also it will allow to know if they have used or not an augmented reality application. So the first questions ask for the frequency with which people use the GPS.

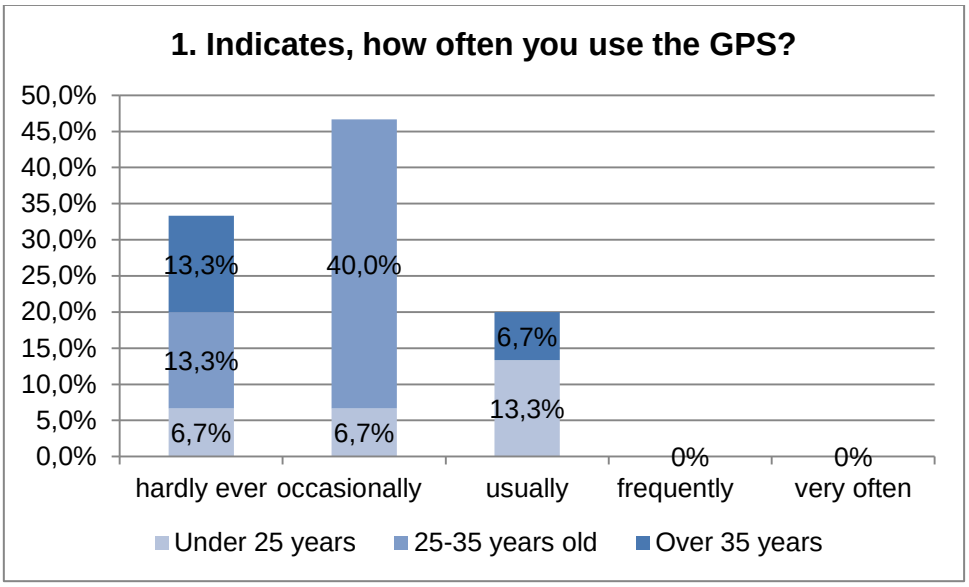

Fig 6.9 Chart of the first question: Indicates, how often you use the GPS?

The result of this question indicates that with a 47%, most of surveyed people use the GPS occasionally, especially those between 25 and 35 years. With this, it is observed that there is no custom of using GPS and it is understandable that people are not traveling all the time; this percentage would have changed for specific people working in the transport sector.

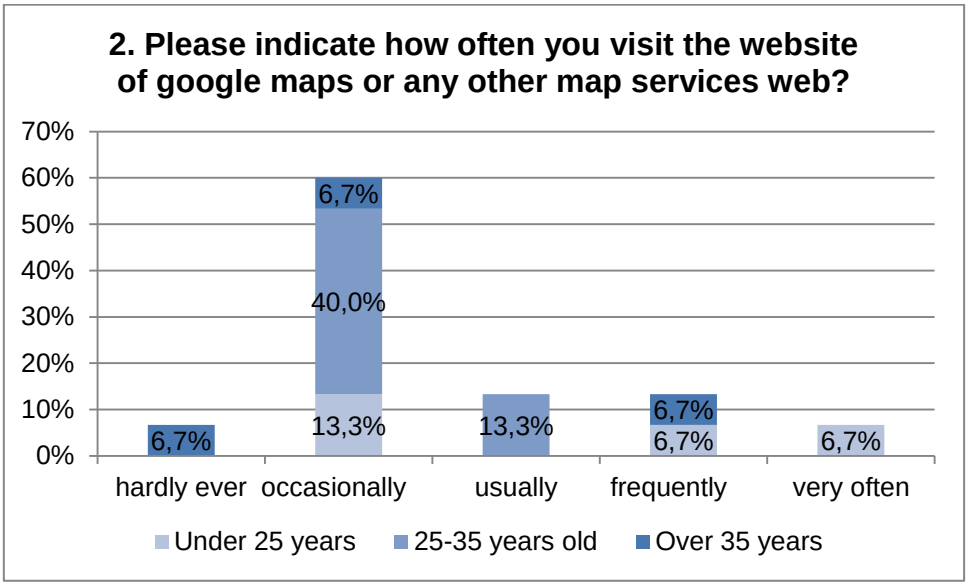

Fig. 6.10 Frequency chart about how much surveyed people use map services web.

The first and second questions confirm that people use map services from time to time, and only when they need it; So the result of second question show that 20% of surveyed use frequently map services by web. The following two questions allow knowing what the perception that people have about augmented reality is.

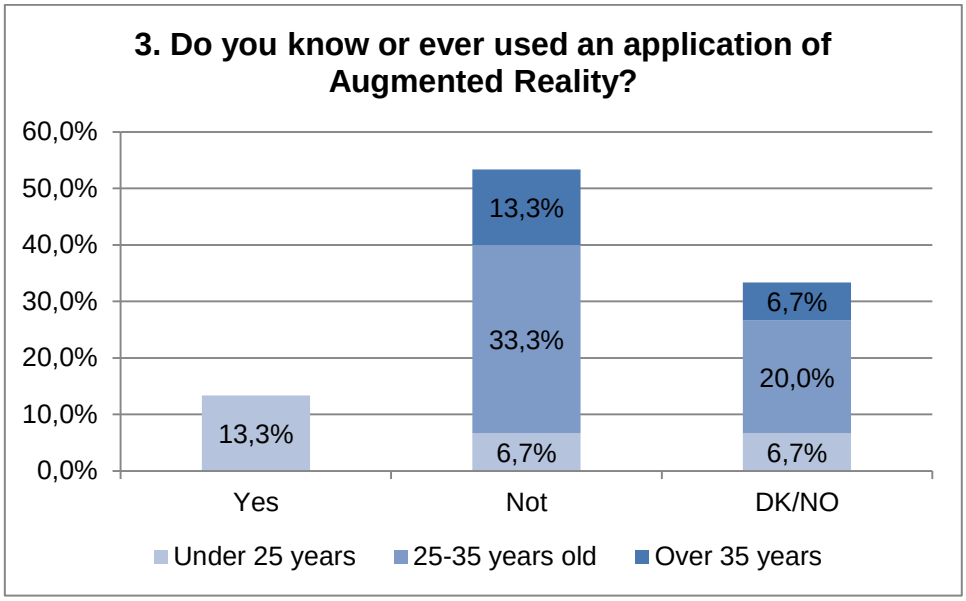

Fig. 6.11 Chart of the usage of AR applications.

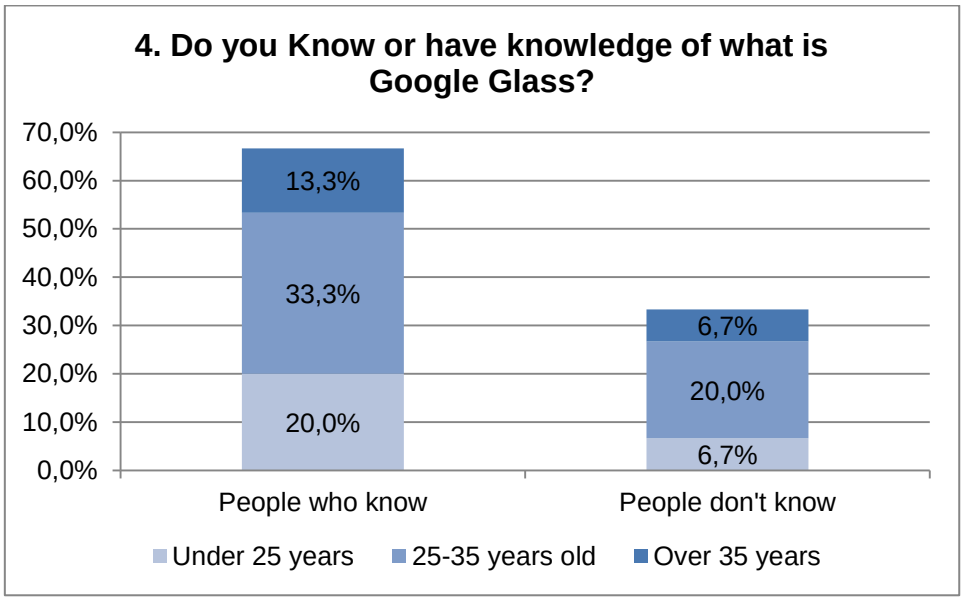

Fig. 6.12 Chart of knowledge that people have about Google Glass.

The obtained results from these two questions are very interesting because most of surveyed people have never used an augmented reality application (53% shown in Fig 6.11) and 33% of people don't know what is an AR application. But curiously they do know what a Google Glass (67% shown in Fig, 6.12). This is easy to understand since google has made a big marketing campaign on the Internet and on TV News. Summarizing, people have the custom to use both GPS and web map services occasionally and also have to add the little use or failure to recognize the augmented reality applications.

The rest of the questions focus on the opinion that user has about the implemented application has a result the following figures:

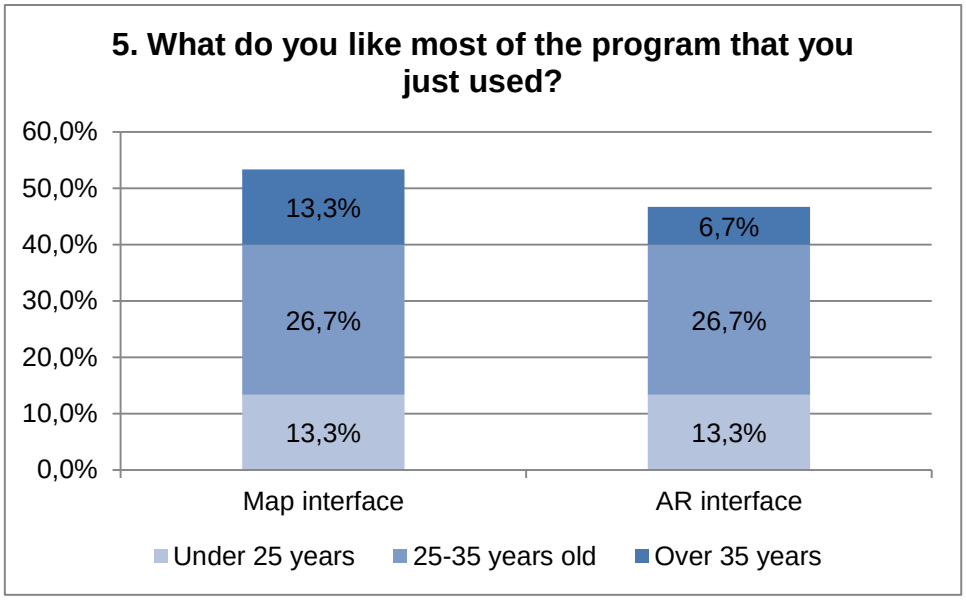

Fig. 6.13 Feedback from users about their preferences in the program interface.

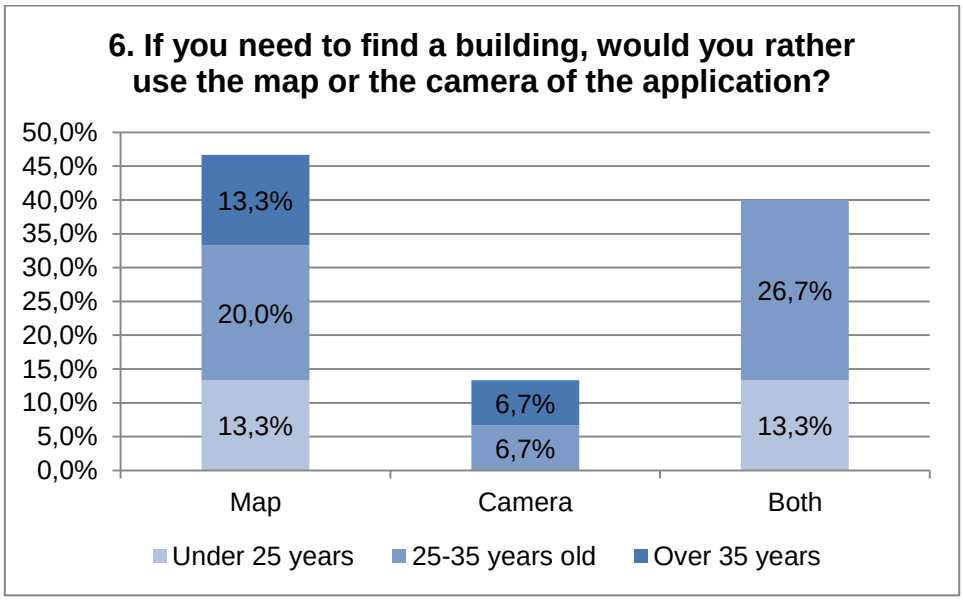

Fig. 6.14 Preference of usage about the services of the application.

The question five and six, have the same purpose, to know which interface, they believe that is more useful. Therefore the obtained results say 53% of people prefer to use the map services than AR services, nonetheless the percentage are similar and the question six helps to clarify the results. In the question six, the percentage for the map interface is similar (~47%), and 40% of users think that the combined use of both services is much better than using only the AR interface (13.4%).

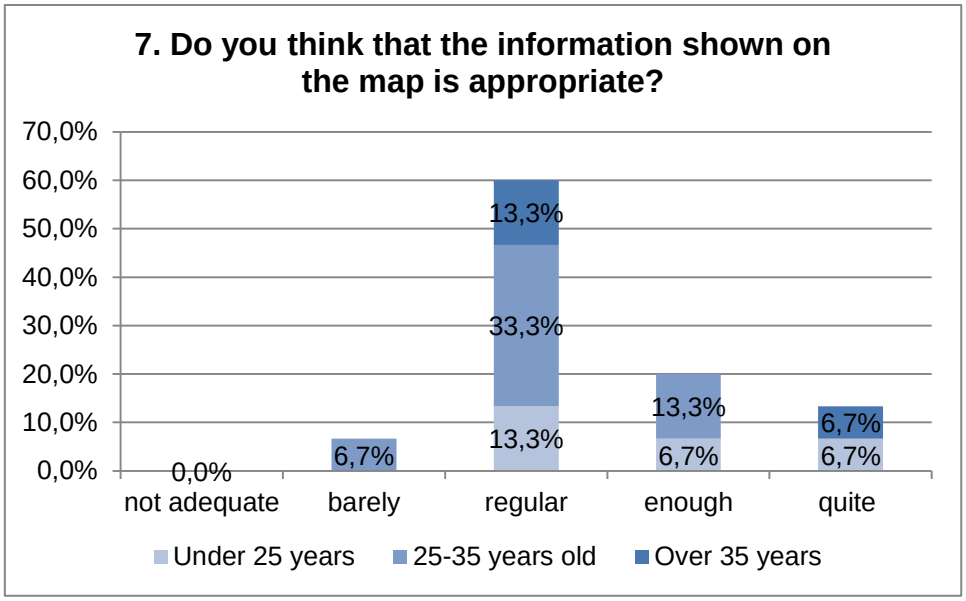

Fig. 6.15 Opinion about the shown information on the map.

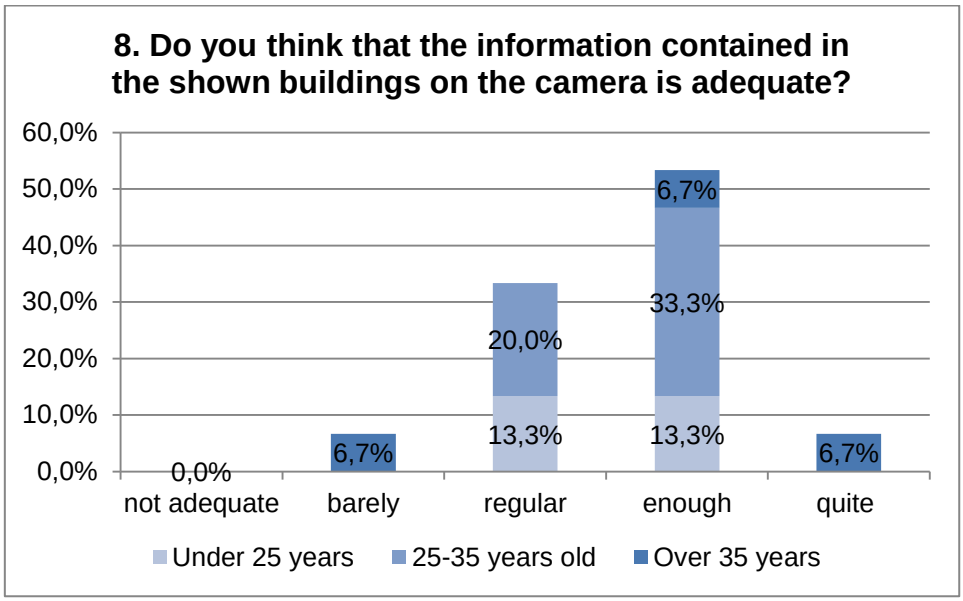

Fig. 6.16 Opinion about the shown information on the map.

The results of the seven-eight questions are shown in the figures 6.15 and 6.16, where most of surveyed think the information on the map with 60% is regular, nevertheless they also think the shown information for buildings in the AR interface is enough adequate; people between 25-35 years old are the ones who most valued the function of augmented reality. These two questions were formulated for this reason the same information is evaluated differently by user. The obtained results related to map interface or services, are the expected because the users are accustomed at much improved maps as Google maps, then it is logical that augmented reality interface was best evaluated, because the data are exposed in a simple way and understandable.

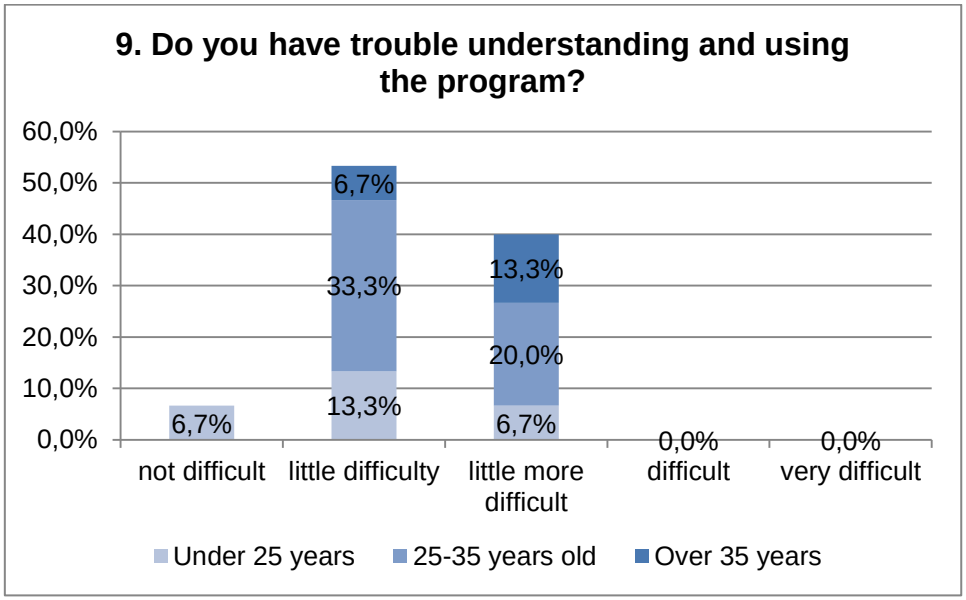

Fig. 6.17 Percentage of trouble understanding the program.

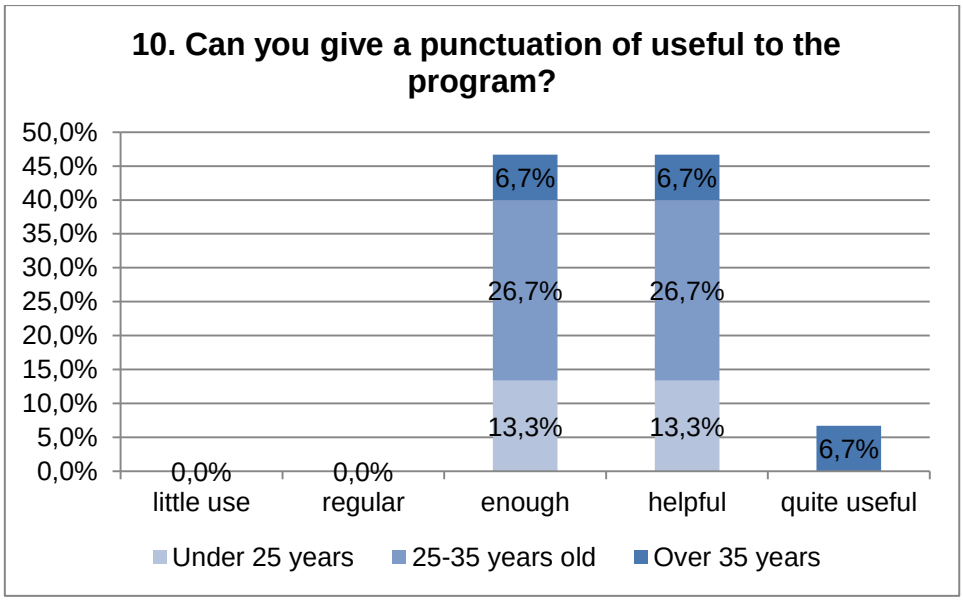

Fig. 6.18 Punctuation of usefulness of the program.

Finally the results of the nine-ten questions shown the punctuation that the application get from users; the usability of the program according surveyed people is little difficulty with 53.3% where 30% corresponds to users between 25-35 years old, 40% users think the program a little more difficult and only 6.7% of surveyed people (under 25 years old) say the program is easy and had no problems using it. The answer of the last question says 46.7% of users believe that the program is enough useful and with the same percentage the rest of users think the program is helpful, for this question is no distinction of

age because the percentage is relatively even, but can say that 6.7% of people over 35 years thought the application is quite useful. Summarizing, the surveyed people believe that the program is useful; a combination of augmented reality and mapping services can improve the location service completely and although people prefer to use a familiar interface, it is likely technologies such as Google glass, helps to change the perception of the usefulness of AR applications.

7. CONCLUSION

The implemented system can be deemed a success in respect of the fact that the main core of the project has been met. The two interfaces proposed on application requirements, mapping service and augmented reality interface, have been successfully implemented, tested and integrated as one.

Respect to offered potential of Unity 3D, the free version is quite robust in terms of programming, however in aspects of image and textures some options are disable but it doesn't prevent the program will be good quality. Therefore, the final program has able to verify that libraries offered by Unity 3D allow working properly with the embedded sensors in the devices, in the case of GPS, the position obtained by the sensor doesn't only depend on libraries, but also the accuracy of GPS sensor of the mobile device.

In the implementation, when the request is performed to obtain the information of buildings, XML data obtained through the request to the OpenStreetMap server are categorized by nodes; however much information classified as way is lost and cannot show. Even so the tiles offered by OpenStreetMap have better quality details than tiles of google maps.

Finally, according to the results obtained through the test, the people who have used the program have a slight preference to use the mapping service; nonetheless they have assessed that the augmented reality interface is quite useful, although most of them occasionally use the mapping services and have not used any augmented reality application. To conclude, the people surveyed have rated that the implemented application is enough useful and the combination of augmented reality and mapping services can improve the location service completely.

8. References

[1] LANE, N., MILUZZO, E., LU, H., PEEBLES, D., and CHOUDHURY, T., “A survey of mobile phone sensing.” *IEEE Communications Magazine*, 2010, Vol. 48, pp. 140–150.

[2] KLEIN, G. and DRUMMOND, T., “Tightly integrated sensor fusion for robust visual tracking.” *Image and Vision Computing,* 2004, Vol. 22, No. 10, pp. 769–776.

[3] Isaacson, B., 5 “Alternative Heads-Up Displays That Aren’t Google Glass”, 4 March 2013 http://www.huffingtonpost.com/2013/03/04/google-glass-alternatives_n_2735818.html

[4] “Google Glass – What It Does” http://www.google.com/glass/start/what-it-does/

[5] Google Developers, “Glass Overview” https://developers.google.com/glass/overview

[6] Wikipedia, “Cloud Computing” http://en.wikipedia.org/wiki/Cloud_computing

[7] Crunch Base, “Google App Engine” http://www.crunchbase.com/product/google-app-engine

[8] Digital Tech, “Google Maps versus OpenStreetMap: charting new territory on the Web?” http://www.inaglobal.fr/en/digital-tech/article/google-maps-versus-openstreetmap-charting-new-territory-web

[9] Steve Coast, “It’s Time to Make OpenStreetMap Your Only Street Map” http://stevecoast.com/2014/01/30/its-time-to-make-openstreetmap-your-only-street-map/

[10] RaceSimOnline, “The Field of View” http://www.racesimonline.com/articulos/El_campo_de_vision.php

[11] CUIJPERS, R.H., KAPPERS, A. M. L. and KOENDERINK, J. J., “Visual Perception of Collinearity”. Perception & Psychophysics 2002, 64(3), pp. 392-404.

[12] Alastair Aitchison - Spatial / A.I. / Game Development, “The Google Maps / Bing Maps Spherical Mercator Projection” http://alastaira.wordpress.com/2011/01/23/the-google-maps-bing-maps-spherical-mercator-projection/