BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

AUGMENTED REALITY BASED ON OPTICAL SEE-THROUGH DEVICE

ROZŠÍŘENÁ REALITA POMOCÍ OPTICKY-PRŮHLEDNÉHO ZAŘÍZENÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR AUTOR PRÁCE	ROBERTO LÁZARO
SUPERVISOR VEDOUCÍ PRÁCE	Ing. VÍTESLAV BERAN, Ph.D.

BRNO 2016

Abstract

This BSc Thesis was performed during a study stay at the Faculty of Information Technology of the Brno University of Technology. The thesis is focused on augmented reality using Unity and optical see-through device. The theoretical part presents augmented reality, its history, application areas and possibilities of further development. The practical part is focused on using glasses for augmented reality and its integration to Unity framework. The designed application contains a renderer of the 3D data from a robot into the scene of the user based on leading markers. The data is obtained in two ways: from a server publishing 3D data measured by robot or from a file. The thesis describes the design, implementation of the application and necessary preliminary work in order to work with the device properly. Finally, the testing of the application and results are described together with possibilities of the future development.

Abstrakt

Tato bakalářská práce byla vytvořena během studijního pobytu na Fakultě informačních technologií Vysokého učení technického v Brně. Práce je zaměřena na rozšířenou realitu s využitím Unity pro opticky průhledné zařízení. Teoretická část pokrývá obecný přehled o rozšířené realitě, její historii, oblastí použití a dalších možností jejího vývoje. V praktické části se práce zaměřuje na práci s brýlemi pro rozšířenou realitu a s jejich integrací do Unity. Navržená aplikace obsahuje zobrazování 3D dat z robota do scény uživatele s použitím vodicích značek. Data jsou získána dvěma způsoby: ze serveru, který publikuje 3D data snímaných robotem nebo ze souboru. Práce popisuje návrh, implementaci aplikace a přípravné práce nutných pro využití zobrazovacího zařízení. Práce na závěr popisuje způsob testování a výsledky řešení včetně možností dalšího vývoje.

Keywords

Augmented Reality, Unity, Vuzix, Optical See-Through, ARToolkit, 3D Frameworks

Klíčová slova

Rozšířená realita, Unity, Vuzix, Opticky průhledné brýle, ARToolkit, 3D frameworks

Reference

LÁZARO, Roberto. *Augmented Reality Based on Optical See-Through Device.* Brno, 2016. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Beran Víteslav.

Augmented Reality Based on Optical See-Through Device

Declaration

I declare that I have created this thesis myself under the supervision of Ing. Víteslav Beran, PhD. I have cited all the bibliographic sources and publications used for the creation of this thesis.

. . . .
Roberto Lázaro
May 18, 2016

Acknowledgements

I would like to thank my supervisor Mr. Beran for all the useful advices and the help that he gave me, and for the trust that he had on me when I faced the most difficult problems. Also special gratitude to my family for all the support and for make true the chance to get here. In addition I would like to thank my lab partners for all the help and collaboration. And finally a special acknowledge to my Spanish family in Brno, that gave me a lot of good moments during this period of my life.

Contents

Chapter 1

Introduction

With augmented reality applications you can mix the real world with computer generated graphics. These kind of applications have to be accurate to obtain good results when rendering the virtual content. In this thesis is explained what is and how to use augmented reality to develop an application that will solve the necessity of knowing what a robot sees with their sensors in the real world. So the main goal of the project is to render the data that the robot is processing into the real world using augmented reality tools.

The final solution of the project has been developed with Vuzix STAR 1200 XL augmented reality glasses, and using Unity as a framework of development in a Windows 10 environment. But it can be reused with other augmented reality devices, with the camera of a PC, or a tablet or even a smartphone. Also this solution can be used in Linux environments with a few changes since Unity also works on them.

In order to achieve the goal of rendering the data in the real world, this thesis solves several problems and has additional information. This information includes how to integrate the glasses into Unity, how to work with the SDK that the Vuzix company provides with the glasses and how to use it to build some application for the glasses. For the last thing i will explain the installation of the dependencies, that are needed to work with the glasses. Because the device is from 2010 and the libraries and dependencies that are needed in most cases are old and give some problems at the time of compilation.

In addition rendering frameworks for AR applications are covered in this these. These frameworks include Unity(that is the one which the final solution is built in), Autodesk 3D Max, Open-Scene-Graph, ARToolkit. All of them can be used to develop AR applications, and all of them can be used for free except 3D Max, also Open-Scene-Graph and ARToolkit are open source. Unity has been chosen for the final solution because other projects that works with the robots are built with it, so in this way the solutions can be mixed and reused in the other projects.

Chapter 2

Theoretical part

2.1 Definition

Augmented reality(AR) is the use of the real world view that surrounds us through an electronic device with components and information that are produced by a computer, this two elements are combined into a mixed view displayed into the device. AR also refers to the elements that are "augmented" (displayed into the real world). It has not to be confused with virtual reality(VR), virtual reality generates a whole new 3D world with the help of some displays called Head-Mounted Displays(HMD), and AR generates virtual content in the real-world. The content that is displayed may be so different, starting with normal text that is showed in the display, or maybe an image or a video, to 3D content that is rendered in specific places of the real world using techniques of tracking.

2.2 Brief History

The first device invented that used AR is from 1968, and was invented by Ivan Sutherland, was the first HMD of the world called the Sword of Damocles and it was not wearable, it was suspended from the ceiling. However the idea of AR was firstly mentioned earlier in the twentieth century by a writer called Frank Baum, the idea was a display that overlay data into the real life, was called 'character display'. Later on, in 1990 Thomas P. Caudell, a boeing researcher, created the term 'Augmented Reality'.

The next steps of AR were developed mainly by US military institutions that created AR systems for soldiers, like HMD that shows intel about munition, enemies, etc.., or AR combat systems for their battleships or helicopters. But that was for private military use, for public use in the nineties the first AR production for a theatre was created in Australia in 1994. And also other important fact is that in 1999 ARToolkit was developed by Hirokazu Kato, this is a free group of libraries based on OpenGL and OpenCV to develop AR solutions.

From 2000 until now the history of AR have been focused on the develop of new HMD systems like Vuzix 920AR, Epson Moverio or the most recent Microsoft Hololens. Also with the arrival of the tablets and smartphones AR have found a new whole group of gadgets to develop applications. At the same time that this inventions were created, the software has been improved to make easier the creation of new programs based on AR.

2.3 Application Areas

One of the biggest benefits of AR is that have a lot of areas where it can be applied, here i will mention few of them:

Archaeology

AR can help archaeologist to see how the places were built in the past, where was the important buildings, draw schemes of founded objects to see patterns. Also it can be used to augment ancient archaeological ruins for the tourists that want to see how the original buildings were. One example is Architip AR (Figure 2.1), an smartphone app that allows the user to view the ancient look of some ruins.

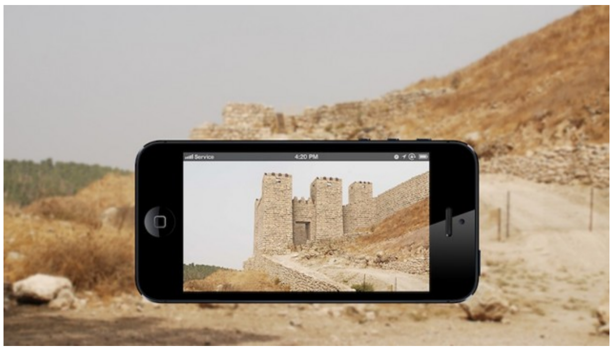


Figure 2.1: Architip AR app

Architecture

In this area AR can help by rendering virtual models of the buildings in the place where they are meant to be. Other application in architecture can be to augment the draws and the designs of the architect to see them in 3D. Also can be used to see the structure of buildings or constructed areas. One example is the application that Trimble and Microsoft are developing(Figure 2.2), this solution renders buildings using Hololens on the planes to help the architects in their work.

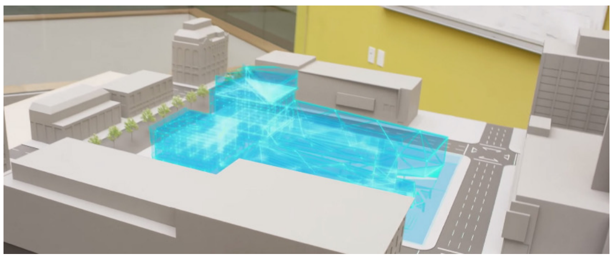


Figure 2.2: Trimble AR app

Art

AR can aid this area, showing information about the pieces to the visitors of the museums. Or allowing the artist to create augmented reality art to use more ways of expression. Also is possible to show art pieces in every part of the world.

Commerce

This is one of the biggest areas where AR can be applied because of the interest of the enterprises. AR can be used to augment the objects in a shop in order to show more info about it, to see the content inside the products without open it, to put promotional content like marketing videos or offers attached to the augmented products, to make an online market with augmented products where you can buy from your home...

Construction

In this area AR is combined with GPS localisation and other georeferential technologies to show buildings, structures, pipelines, cables and other construction stuff, that can help workers to locate problems, to construct better and to consult helpful info. AN example is CityViewAR app(Figure 2.3), a solution that shows buildings destroyed by an earthquake in the city of ChristChurch(New Zealand), in order to reconstruct them properly.

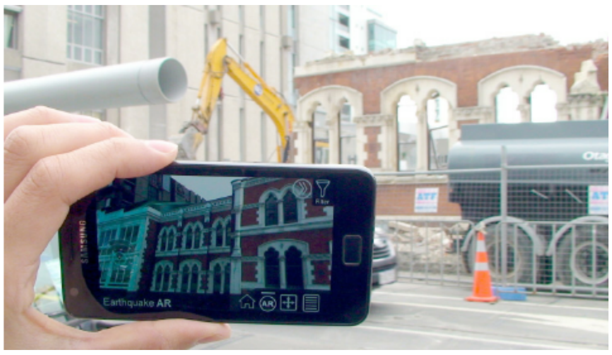


Figure 2.3: CityViewAR app

Education

This is another of the biggest areas of AR application. In this area AR can improve the learning of the students by augmenting their lessons or their books with 3D visualizations of the contents. Even can augment animations and videos or allow the interaction between the student and the contents. AR can aid students and teachers by allowing the collaboration, or the interaction between them in different places. Also is possible to create practical experiences for the students in environments that in other way could be impossible, like seeing how the human body is inside and allowing the student to manipulate the bones and the organs, or showing how the atoms are from a closer point of view.

Brain Scan(Figure 2.4) is an augmented reality app for tablets that allows the student to see how the huan brain works nd to interact with his different parts.

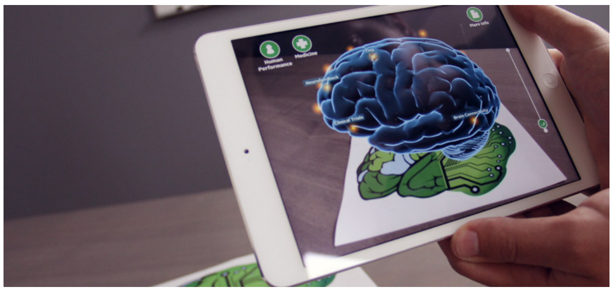


Figure 2.4: Brain Scan app

Emergency management

AR can be used to manage emergencies faster and better, for example, showing the professionals where are the people in danger, what areas are threatened, what's the level of danger and a lot of more info that can save lives and do their job safer.

For example, LandForm+(Figure 2.5) is a geographic augmented reality system used for search and rescue, and emergency management. It renders useful information for the rescue forces, to help them in their job.

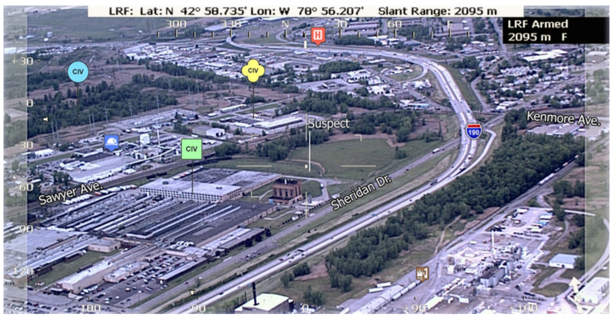


Figure 2.5: LandForm+ app

Video Gaming

Maybe this is the first thing that people think is a good area for AR, and it's true that AR can be a revolution for gaming. Augmented reality allows gamers to experience digital game play in a real world environment. In the last ten years there has been a lot of improvements of technology, resulting in better movement detection and the possibility for the Wii to exist, but also direct detection of the player's movements.

There are many games being developed for AR and in the next years we will see a revolution in this area. One example are the Invizimals(Figure 2.6) series from Sony.

Figure 2.6: Invizimals Game

Industrial design

Here AR can help the engineers by showing the designs of the products in 3D, allowing to look how the pieces and the mechanisms will work together. Also can show simulations of the products before they are created. For example Volkswagen uses AR to make simulations with their designs, or to help the workers in the repairs of the cars(Figure 2.7).

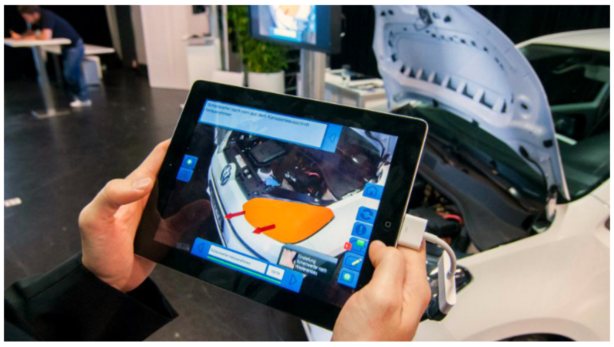


Figure 2.7: Volkswagen AR repair app

Medical

Ar can help doctors providing useful information of their patients when they need surgery or harmful treatments. Also is possible to use applications as X-ray viewers or searchers for tumors.

Since 2005, a device that films subcutaneous veins, processes and projects the image of the veins onto the skin has been used to locate veins. This device is called VeinViewer(Figure 2.8).

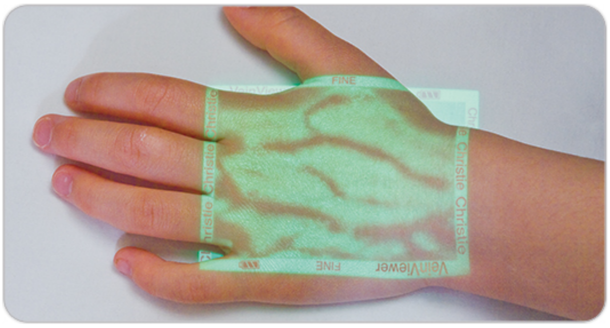


Figure 2.8: Veinviewer system

Military

This area probably provides one of the biggest inversions on the development on AR to help soldiers and improve weapons in the battlefield. For example AR can show useful information on the googles of the soldiers, like ammunition, targets, state of the other soldiers, routes in the battlefield, views of cameras... Also the command centers can monitor the troops with the use of AR.

In 2014 American army developed an AR application for helping the commanders of the troopers to access useful info about the battlefield in real time. This application was developed with his own helmet with an AR device incorporated. The helmet with the application is called ARC4(Figure 2.9).

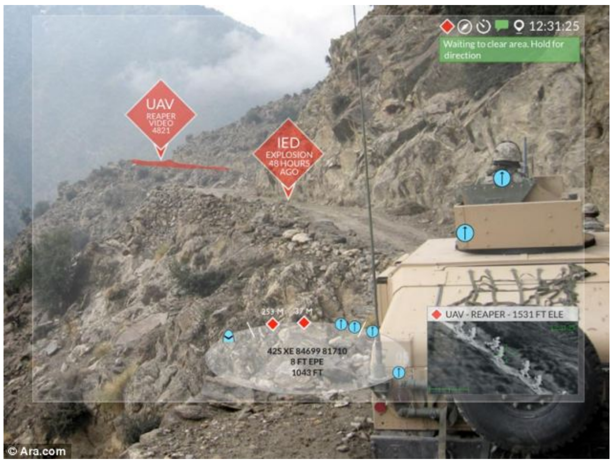


Figure 2.9: ARC4 system

Navigation

Nowadays the new generation of cars is being created using AR to introduce the information about velocity, traffic, state of the road, wind... In the windshield of the car, making driving safer. Also GPS can use AR to augment their routes in the devices or to show dangers in the paths. In general all the ways of transport can benefit of this technology by augmenting their information about the routes or the state of the vehicle.

Office workplace

In this area AR applications can be developed to create virtual conferences between members of a company that are far away ones to others. Also is possible to present augmented information in the meetings to present more clear data or to interact with the projects, or to use in the personal workspace to create a virtual workshop with tools that can optimize the work.

Television

The first augmented reality application for the general public, were the weathercastings on TV. It has become common to show in the display of the weathercasting live videos of the places, 3D content to explain better the predictions, and a lot more 3D stuff that is augmented on the display.

Augmented reality has also become common in sports telecasting. Sports usually overlay information in the telecasting to help the audience to understand what is happening, like the usual lines in football to show the plays, or the lines in swimming competitions that show the record. Also is normal to see commercial banners in the telecasting. All this things are displayed over the real image so they augment the reality to help the spectators.

Also augmented reality is starting to allow Next Generation TV viewers to interact with the programs they are watching. They can place objects into an existing program and interact with these objects, such as moving them around. Avatars of real persons in real time who are also watching the same program. And maybe AR will change the way that we see TV, because we will have the TV with us permanently as Microsoft showed with their Hololens, we will be able to create a TV in any place and see the programs that we want.

2.4 Devices

There are many devices that are designed for augmented reality, normally they are called glasses because of their look, but their technical name is Head-Mounted Displays (HMDs), they are a wearable device, normally with at least one camera for tracking the real world and a pair of lenses, normally semi-transparent.

Also i have to mention that AR can developed for almost every device that have a camera, like a laptop or a smartphone, and there are a lot of applications for this kind of devices but there are not designed with the proposal of develop AR.

I will focus on the most known HMDs and their characteristics, because AR has grown a lot in the last years and there are many of them with interesting data. I have to mention that i will divide the HMDs into the ones that are able to render 3D content and the ones that only augment 2D content.

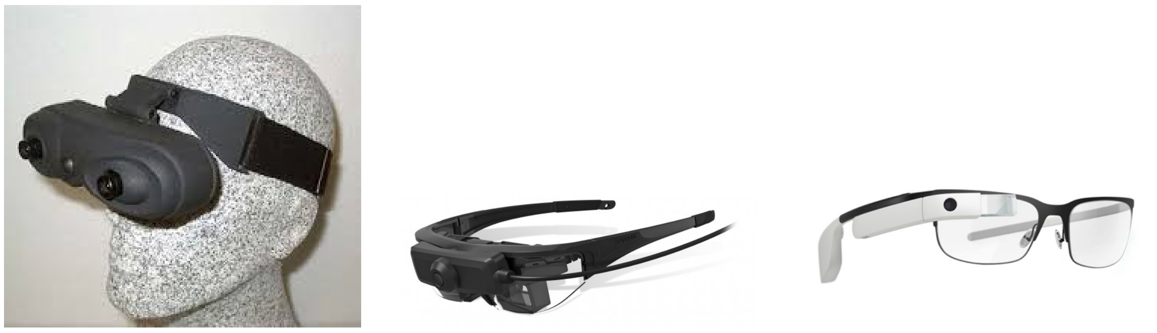

(a) Video See Through (b) Optical See Through for 3D (c) Optical See Through for 2D

Figure 2.10: Examples of HMDs

In the figures on the top is possible to see the differences between the types of HMD. The first one(Figure 2.10a) is a video see through device, so they take video captures from the real world with the cameras and shows them to the user in the displays that are inside the device. This type of HMDs can render 3D and 2D virtual objects, but the user doesn't see the real world. The second and third figures(Figure 2.10b and 2.10c) shows optical see through displays, with them the user sees the real world mixed with another virtual layer where the 3D in case of the second figure and 2D in case of the third one are rendered.

2.4.1 3D Devices

Microsoft Hololens

This is the device that Microsoft is developing right now, in the presentation of his features, was announced that its a optical see through display that projects holograms in the real world and allows the user to interact with them in five ways: with the voice, with the hands, with the movement, with the eyes and with the sound. It's an indoor device, you can't go with it to the street, and there are still some doubts about his development and final features.

Vuzix STAR 1200 XL

Here a special mention need to be done to this device because is the one that is used to develop and test this project. It's a optical see-through gadget that can render 3D content and is designed for be used in interiors. It has a camera that reads the info of the real world and processes it to render the content in the displays. Because it has two displays, is possible to render stereoscopic 3D in order to have depth. Also it has accelerometer, gyroscope and a position tracker to have data about the movement of the glasses. It has the possibility to be in 2D mode to use programs that only needs to show info in the real world.

META 1

This are an optical see-through glasses that can render 3D in the real world and provide interaction by hand and movement recognition, so you can interact with the objects that it renders in the real world with your hands. Also have a big field of view so is possible to see almost the same as if you weren't wearing it. They are designed for interiors not for using it in the street.

2.4.2 2D devices

Epson Moverio BT-200

This is an interesting device because you have to control it with a special controller that is similar to a smartphone but is provided by Epson. This device has access to a marketplace made only for this glasses where you can download and try the apps developed to this glasses. Also is possible to develop and include android apps.

Sony SmartEyeGlass

This device created by Sony is not fully augmented device like Hololens or Moverio, because it const of a small display that is synchronized with your android smartphone via bluetooth and shows info like messages that you receive, tweets, data from the gps to don't get lost, and some other stuff that you want your smartphone to show in that screen. So is AR because it overlays info into the real world, but it can't render 3D data like new objects, it only shows info from your mobile.

Google Glasses

Quite famous and similar to the Sony device, only can display info relative to the phone like calls, info about the weather, videos, messages and some other stuff. Also with this you can shoot some photos and videos with the device camera but this will eat the battery of the glasses. Google is working in a new version because the first one doesn't had much acceptance. You can interact with the device with your voice or with your eye.

Vuzix M100

Is another optical see through device like google glasses. It's connected with your smartphone or with the Internet and shows you info in the real world. It's the direct competitor of the google glasses. You can control it by using your voice, a couple of buttons in the device, or using your smartphone as a virtual mouse.

Recon Jet

This are glasses with a right-eye display as google glasses, with a sport design that are designed for sport environments. They are planned to show important information to the athlete, like velocity, heart rate, temperature... They work with iOS or android devices and use AR applications in both of the platforms. For control they have a touchpad and besides they have microphone, is not possible to control them by voice natively.

Optivent Ora-1

Another device that have a pair of glasses and a optical display in one of them. This device has a few special characteristics that the others does not have. The first one is that you can move the screen of the display so it's possible to have the AR info in the front of your sight or in the corner. The second one is that the lenses are photochromic so they change between clear lenses and dark lenses with the sunlight. And the third one is that this device can function as a standalone android device, so there is no need of use it with your smartphone.

Glass Up

This AR product has less features than the others but it's because is designed as a simple glasses to offer basic information about places and online events like messages or notifications. It only shows in the center of your sight the information in a basic monochromatic way, also it can't be controlled by voice or using a touchpad, also it doesn't had a camera for taking photos. To sum up it only shows basic info during a few seconds in the display.

2.5 Camera tracking

One of the basic things in AR is to locate the camera, specially if you are using a HMD. For doing the tracking, the camera has to recognise real world objects and use their positions to compute his own position[3]. Doing this is not easy because the recognition of objects in the real world is hard and normally inaccurate, so to solve this problem normally the applications use markers. The markers are objects placed into the scene with a black and white special figure drawn on it. These figures are patterns easy recognisable for the camera, making the tracking problem easier, but also if the application has to be used outdoors they aren't practical so for outdoor applications the solution is to use markerless tracking. These two ways of tracking(marker and markerless) are the solutions that can be used nowadays when developing an AR application, it depends on the type of application to use one or another.

In our case this problem is solved by using ARToolkit as a library that handle the tracking and marker recognition. However is interesting and useful to know how the tracking is done, to understand how the solution works. So the next lines will describe how ARToolkit does this tracking[11].

First of all ARToolkit is OpenGL and OpenCV based, so for doing the tracking and recognition will be using OpenCV. After looking at the source code of ARToolkit libraries and his documentation, focusing on the AR tracking, is possible to realize that is done in three steps. The first one is to prepare the camera of the device and start filming live video with it, this is done to catch the frames of the video to analize them. When the framework has the frames, the second step is an algorithm that is in charge of transform these images into binary images(black and white) using the lightning of the environment. Then another algorithm starts to search squares shapes in the images, this search is done by looking for the corners of the squares and locating their coordinates, when the algorithm find one square(four corners positioned as a square)then searches for the image contained inside the square. If the square has an image then the pattern of the image is captured and analized to compare it with the trained patterns. If the pattern of the square matches with a trained pattern, then saves the position of the four corners and calculates the rotation of the marker comparing it with a default position of the marker that is saved, basically compares the relative position of the corners to themselves and compares this positions to the default ones to calculate the rotation of the marker. The third and final step is to locate the camera using the tracked marker, to do this ARToolkit uses a 3x4 matrix that fills with the coordinates obtained from the marker and using that coordinates calculates the actual

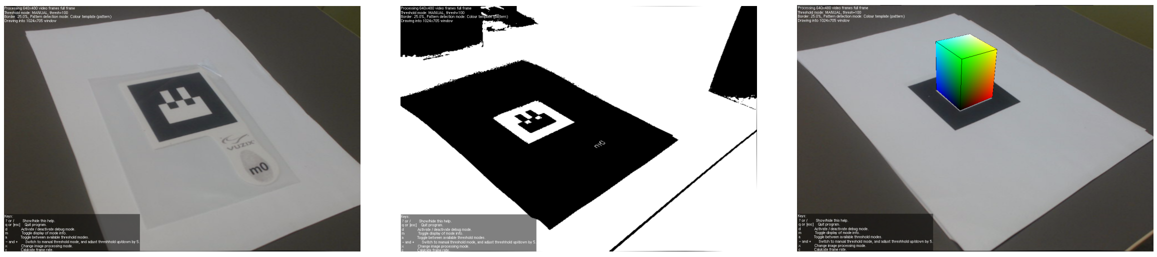

(a) Video See Through (b) Optical See Through for 3D (c) Optical See Through for 2D

Figure 2.11: Marker detection process

position of the camera. Then with those coordinates creates a virtual camera on the same position as the real one is, and uses this camera to display the virtual scene.

In the previous figures is possible to see the process described in this section. Figure 2.11a shows the video frames that are captured initially. Then in Figure 2.11b the video is converted to binary and is analized, searching for the patterns of the markers. Finally in Figure 2.11c when the marker is detected the 3D stuff is rendered using a virtual camera.

2.5.1 Markerless tracking

With this technique you can reach the ideal AR application, because it will only depend of the camera itself and of course of the code of the application. In the future most AR applications will use markerless tracking but nowadays the algorithms used to achieve this technique doesn't provide robust solutions as the marker tracking provides. This is because markerless tracking is far more difficult than marker tracking. In markerless tracking the camera has to be able to recognise patterns in the real world, and these patterns are complex(like cars, people, furniture . . .).

Markerless tracking actually uses several methods like SLAM(simultaneous location and mapping) or algorithms based on basic geometrical shapes[2], but all of them have the problem that only provide acceptable results in planar surfaces which is not the optimal solution for an application.

In the previous section I have explained the camera tracking with a marker. With markerless the difference is that we don't have the patterns to camper with, so the algorithm has to search for figures in the environment that are static in order to use them to obtain the coordinates to locate the camera.

2.5.2 Registration

In order to achieve a good application, one big problem you have to deal with is registration. Registration is the accurately alignment between the real world and the virtual images or objects[1][8]. It has to be the most precise that is possible, because a bad registration will cause bad results and even will cause the user to feel dizzy.

The registration can be divided into static and dynamic. Static registration is when the user remains in the same pose all the time, and dynamic registration is when the user moves the pose. Normally in AR systems you have to support a good dynamic registration, that means that the virtual objects have to be accurately rendered to avoid lag or jitter between them and the real ones.

Registration is done by using the camera created with OpenCV when doing the tracking. This camera has to match the real camera frames so the view that the real camera is filming and the view that the virtual camera is rendering has to be the same. Because if not the virtual objects will appear in different positions from the ones where they has to be.

2.6 Rendering Frameworks

2.6.1 Unity

Unity is a cross-platform game engine developed by Unity Technologies in 2005. A game engine is a framework specialised in developing and rendering computer graphics, normally used to create videogames. Unity can be used in Windows, Linux or OS X, and supports many platforms like android, iOS, Linux, OS X, Playstation, Xbox, Wii, and web. Depending on the development platform Unity will use different graphic libraries; in Windows is Direct 3D and in Linux and OS X is OpenGL. Unity supports two mainly programming languages, C# and JavaScript, but also is possible to use Unity Scripting that is the own language of the engine.

One of the main features of Unity is his asset store, which is a marketplace where you can find many items for building your own applications, this items are developed by the community in his majority, but also others are created by the enterprise itself. Unity can be used for free, but for using certain features(like using external plugins) is necessary to acquire a license.

Is usual to use Unity combined with other graphic generation tools like 3ds Max, Maya, ZBrush, Blender... This is because this tools are specialized in creating the textures and models that Unity will use inside his scenes. It's possible to create this stuff only with Unity but is less powerful. Also is possible to create the scenes with only this tools but it's harder to code the internal logic of the scenes and have less support for the final platforms.

In the case of AR development, there are a lot of libraries or plugins that can be integrated with Unity and helps the developer to create AR applications. In fact some of this plugins or libraries are other frameworks that can be used on its own to develop AR applications, but combining it with Unity result in a more powerful tool for developers.

2.6.2 Autodesk 3ds Max

3ds Max is also a rendering framework created in 1995 by Autodesk that can be used to develop 3D applications, however it has differences in the way it works with the others. 3ds Max is focused on modelling assets, this means that provides useful tools for creating and modelling 3d objects, for creating textures for this objects and for combining the models and the textures.But it didn't provide many tools for developing the internal logic and behaviour of the scenes, because of this 3ds Max is not considered a graphic engine.

Furthermore, is normally used previously to Unity or some other graphic engine, first you create the 3D assets in 3ds Max and then you import them into the graphic engine.

3ds Max can be used only in Windows systems and has its own programming language called MaxScript, but it's also possible to use C++ with the SDK of 3ds Max. Normally 3ds Max isn't used for AR applications but the hardware that is used in this thesis include a special plugin for 3ds Max 2011 that allow the developer to create a fast AR scene and to export it as an executable. This plugin is called MaxReality and basically it works as a library for developing with 3ds Max, and also as a player for the AR scenes that you create with this framework.

2.6.3 OpenSceneGraph

OpenSceneGraph is another graphic engine created in 1998 by a programmer called Don Burns, in 1999 Robert Osfield joined him to continue developing the framework. This engine compared with others that are avaliable nowadays has the benefit that is very light and doesn't need much resources to work with it. But on the other hand the results that you can obtain are worst compared with more complex and heavier frameworks, not in a functionality meaning but in an external aspect. The framework is coded in C++ and also use this language for developing with it.

OpenScenGraph itself can't be used for developing AR, you need to add some additional libraries to start working in AR. The best ones for doing that are the libraries of ARToolkit for OpenSceneGraph, called OSGART. With this libraries you can create a scene and add recognition for markers and AR camera setup.

2.6.4 ARToolkit

ARToolkit isn't a graphic engine or a modelling framework like the others, it is a specific SDK to create AR applications. It was developed in 1999 by Hirokazu Kato. Nowadays is one of the most used libraries for developing AR applications, and has a large community with lots of documentation that help developers to make their applications better. The SDK itself is coded in C++ and to use it alone you have to use this language, but one of the benefits of ARToolkit it's that is integrated with many 3D frameworks like Unity, Unreal Engine 4, OpenSceneGraph, and many more. In the particular case of Unity the libraries are coded in C# to work better.

Chapter 3

Augmented reality device integration and solution design

The main goal of the project is to render robot data into the real world using AR and Vuzix glasses with Unity. At the beginning I tried to use only the SDK of Vuzix to do this. So, when I started the project my first step was to be able to use the glasses to see some sample scenes of the SDK. I faced the problem that the SDK itself was compiled using Visual Studio 2008 and for a Windows 95/XP system, since this version of Visual Studio is deprecated and is no longer available for download, I have to recompile the sources using a newer version of Visual Studio. For some samples and libraries of the SDK, the 2015 version of the IDE was able to recompile them well and I can try them and modify to test how the SDK works but for some other I need to find Visual Studio 2010, which is also out of support but you can still find some installers on the web. With this two versions I was able to compile and run most of the SDK. With these samples I could try the glasses and see how the SDK locates the camera and renders some 3D stuff into the displays, but all that is included in the SDK is for VR not for AR. All the samples and all the functions in the libraries are for render in mono or stereoscopic 3D VR mode, so I started to search for some library of AR in order to be able to use the glasses this way.

Accomplished the goal of using the glasses, the next goal, as I said, was to use it to render some AR basic scene. For doing that there are two basic ways, to use marker or markerless tracking. The robots use marker tracking to detect the objects, also marker tracking is easier to implement, and for the required goals is enough, so I decided to use some libraries or framework with support to marker tracking. The first way that I took was Autodesk 3ds Max, since the Vuzix toolkit comes with a plugin for this framework and some markers for using it. With this framework I could render some 3D scenes in AR in such an easy way, and it worked with Vuzix glasses. However there was two main problems, the first one was that this framework is for making models and scenes but not for making the internal logic and behaviour of the components, this is because the programming language is MaxScript which is the own scripting language of the framework and with that is not possible to access to the robot data and render it in a proper way. And the second one was that the final solution must be integrated with Unity and this framework can't be integrate with Unity since both of them two big frameworks that work with the same type of models but for different purposes. So it's like integrating two similar things and this doesn't make sense.

Then, the next step was to be able to use Vuzix glasses with Unity. Here I found the biggest problem of the project, that was the lack of documentation about this concrete model of Vuzix device. So as Vuzix hasn't any middleware that can be integrated with Unity for AR, I needed some other library that provide the basics of AR an can be integrated into Unity. These basics are support for OpenCV and OpenGL, that are the base libraries to make AR solutions, so at the beginning I tried to develop some sample code to render a 3D AR scene using only those libraries, but for making it work is necessary to get an additional library called ALVAR, and the version of these library that is needed doesn't exists anymore, this means that the dependencies needed to make it work in the glasses was missing. Finally to solve this problem I found ARToolkit, which his SDK is based on OpenCV and OpenGL and also can be integrated into Unity.

When the integration was successful the next goal was the main one, to render the data of the robot. The robot data is saved in a JSON text file, so to render it was necessary to implement some C# code to open the data, read it and render to the 3D AR scene. Those steps can be developed in some kind of controller that does all the rendering pipeline. The first time this was done in local with no connection with the server that has the data, taht implementation will consist of the final step.

The next, and final, step was to be able to retrieve the data of the robot from the server were the robot works. So the solution has to connect to the server get the JSON data and render it, or if this is not possible, to use a local file with data from the robot to show it. Also a final improvement to the solution is to update the scene in runtime if the data changes. For example, if the robot moves an object the scene has to show that this object has moved. For the first goal, the approach was to develop some code that connects, retrieves and parses the data. And for the second one is necessary to modify the rendering controller.

To reach all this goals several system parts needs to be designed. In the research about how AR is made and the kind of applications that already exist, I learnt that normally the AR applications consists of some libraries for realising the tracking with the camera and the markers, and some implemented code for doing the rendering into the scene and to control the logic of the application. Is necessary to take into account that this solution has to be integrated with Unity so the code for rendering has to interact with the rendering framework.

In Figure 3.1 is possible to see the concept of the proposed system with all the parts. The AR library (ARToolkit 5) that will communicate with the hardware, use his camera to do the tracking and use the pose to render properly the scene, the receiver that will be the part that creates a socket to get the online data, parse it and send it to the controller, and the controller that takes data sended by the receiver if possible or if not takes local data, analizes it extracting all the info into parameters and and then use it to create the objects and assign the parameters for the rendering, and finally the 3D framework(Unity) that will be used by the rendering controller to render the scene and by the AR library to establish the position of the virtual camera. This is an initial approach, the implementation will follow this pattern but other functionality can be added to the components.

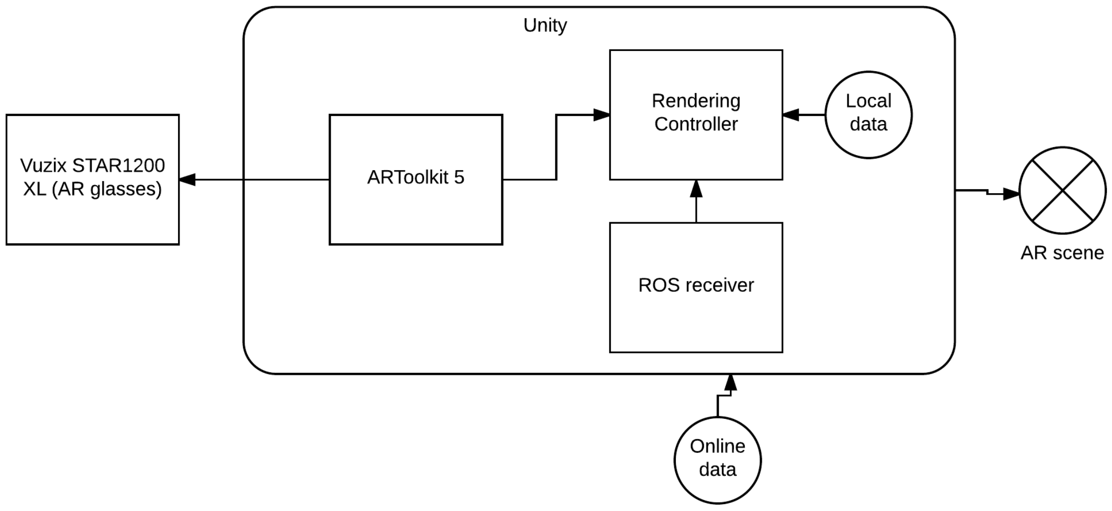


Figure 3.1: Concept of the System

3.1 System Parts

The whole system consists of several parts, including the device(glasses), the markers of the scene and the controllers that process the information between these components and renders the results into the scene. Here these parts are explained in detail.

First of all, is necessary to explain that in Unity when a script is placed in the scene attached to an object, is executed in parallel to all other scripts in the scene. Is possible to establish an order of initialization of the scripts, but in runtime all will be executed in parallel. This is important because in Unity is not possible to use threads in the code, because Unity is not thread safe and will cause errors. However if you know that the scripts are run in parallel is possible to code things that will be executed like if they have threads. Is important to take this into account for understanding the next explanations and how the different parts could work together.

3.1.1 Visual Markers

As explained in chapter 2.5, AR applications could be marker or markerless. For this solution the application will use marker tracking. Markers are an easy way to track the camera and to establish reference points to render the data. In the application only one marker will be used, for referencing the center of coordinates that will allow the application to render the data of the robot around his position, using relative coordinates to the marker. However other markers can be easily added and tracked if necessary, this will be explained in detail in the implementation part. The marker that the application will use as the origin of the coordinates is the one in Figure 3.2.

Normally when you use markers, you can manage to take any image or planar pattern

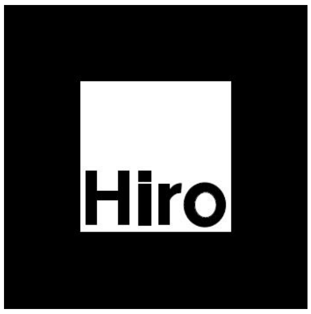


Figure 3.2: Marker of the application

and establish it as a marker. However for a good marker recognition is good to use black and white images called fiducial markers[9]. These markers are similar to QR codes but using less black and white squares, that makes the pattern figures simpler than the QR. Also they are surrounded by a big area of the opposite colour that has more presence in the pattern. All of these properties makes easier for the cameras to detect the marker, and for our application we want to see almost every time the data from the robot, so is needed to have a strong recognition of the marker.

In this solution the marker has a very clear purpose, help the tracking. For doing this the marker will be established as the origin of coordinates of the virtual scene. This means that the coordinates of every object rendered inside the virtual scene will have the marker as origin and Unity will calculate the position of the objects relatively to the marker.

3.1.2 Rendering Controller

For achieving the goals of this project, is necessary to code two main things: the process of getting the data of the robot from the server and rendering it in the AR scene. As we have to work with Unity the way to do this is to implement at least two scripts in C#. This two or more scripts have to be attached to some objects of the AR scene inside Unity, and act like controllers of the scene. Here the meaning of object is not a 3D model that will be rendered, Unity calls object at every part of his scene, in fact a scene is also an object, so everything inside Unity is an object. However there are kinds of objects, in our case the scripts will be attached to an object that acts like a parent for all the rest of objects in the scene, this way is possible to get access from the code to every component created in the scene, and will make easier to control which objects are rendered, to get their parameters and to modify them.

Figure 3.3 shows how the behaviour of the first controller in runtime should be. The objective of the controller is to use the incoming data to render the scene. The implementation will follow this scheme to achieve the goals. In this diagram, firstly is necessary to parse the data to get a data structure from the JSON and make it readable to the part of the controller that will compute it and translate into rendering data, like position, rotation and models. With all of these variables the application can create scene objects that will be like the real ones and render it into the scene. The models will be assigned using the name of the objects that have to be rendered, this is because the robot only works with three objects so there is no database and they can be identified by their names. In the diagram also is included and additional part to update the scene in case the data has changed. To update is necessary that some function keeps listening for updates in the main class and then if and update comes, is necessary to repeat the part of analizing the data and then update the scene with the new parameters.

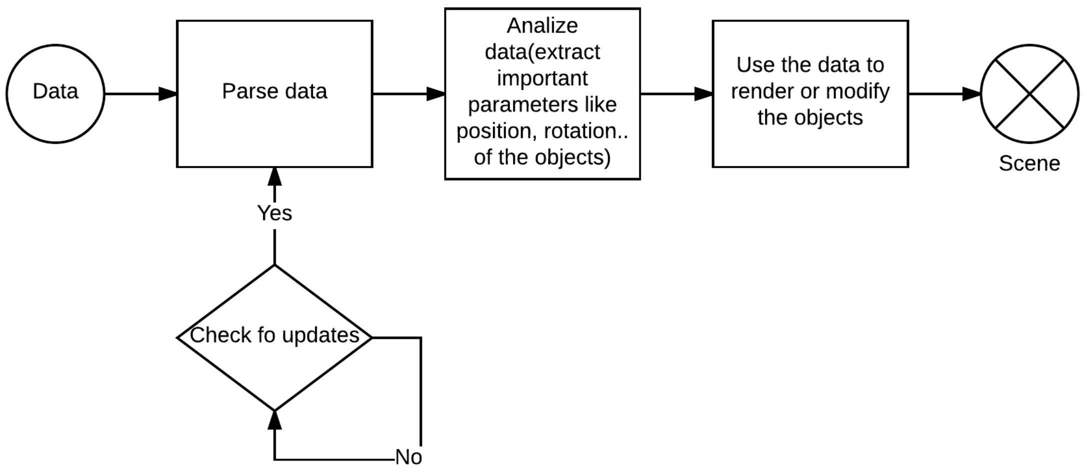

Data
Parse data
Analize data(extract important parameters like position, rotation.. of the objects)
Use the data to render or modify the objects
Scene
Yes
Check fo updates
No

Figure 3.3: Data-flow diagram of rendering controller

3.1.3 ROS receiver

The other part to implement is the receiver of the messages from the server. To connect to the server is necessary to use ROSBridge, because the messages are coded by ROS and is necessary to understand this messages. The way that this part should work is to create a listener that awaits for the message of the server, then the message has to be parsed into JSON, and if all goes well send the message to the rendering controller. Also is necessary to keep listening for posterior messages, because updates to the data can come. In Figure 3.4 this behaviour is showed. The listener, in this case a socket, is created and subscribed to the server, when a message arrives is readed and then parsed into JSON to finally, send it to the rendering controller.

3.2 Virtual layer

In AR applications you combine different layers in order to augment the reality. Can be as many layers as the developer wants(with a resource cost of course) and needs, everyone of them with their own behaviour and components. For this concrete solution will be two layers(which is the minimum number for AR applications), one of them with the images of the real world and the other with the 3D models of the solutions. These virtual stuff that the solution will use, is provided by Unity or some static models that the robot use. This is

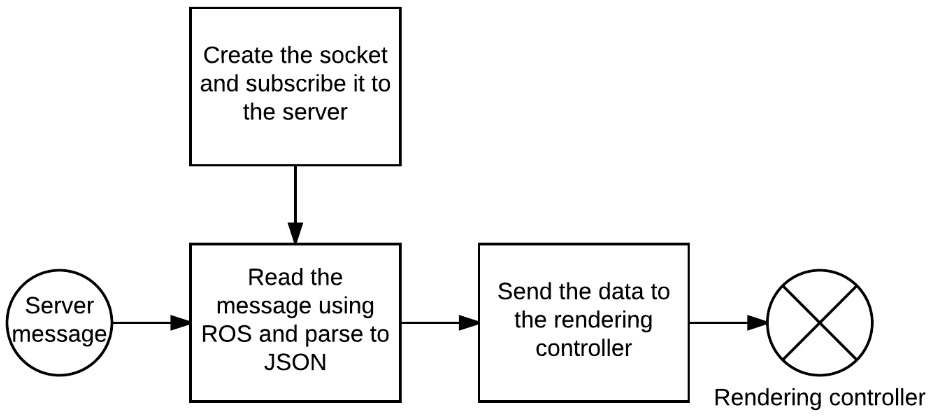

Create the socket and subscribe it to the server
Server message
Read the message using ROS and parse to JSON
Send the data to the rendering controller
Rendering controller

Figure 3.4: Data-flow diagram of ROS receiver

because actually the robot only works with three objects that are modelled into some .dae files, so there is no information about their size, only the model. Also are used the basic geometry figures that Unity provides to test the solution and to show different objects in AR.

3.2.1 Models

Models are 3D representations of real objects, these representations are scalable, normally come with a texture or material attached(like a colour) and are used with any graphics framework to make the setup of the scene.

The models can be saved in a lot of different types of files, depending on the framework or program that has been used to create them. In our specific case the format will be .dae, which is an XML based format for saving 3D files. Since Unity has native support for this kind of 3D files it's a good choice for our models.

For the final solution only three models will be used(apart for the default geometry figures that comes with Unity), since the robot only works with those three. The models are point clouds transformed into solid 3D objects, so their aspect doesn't look like a normal 3D object, but they are clearly recognisable. The nexts figures represent those models: the first one(Figure 3.5a) is the model of a case, the second one(Figure 3.5b) is a juice brick model, and the third one(Figure 3.5c) is a tea box model.

3.2.2 Virtual Scene

The virtual scene refers to the whole group of virtual layers that will be combined with the reality to augment it. In the solution will be only one virtual layer in the virtual scene so is not the scene is not too complex, but also a layer can have as many things as the developer wants. In this case the layer will be the combination of the models, the scripts and the cameras that we will place inside Unity, all of them are game objects. This combination

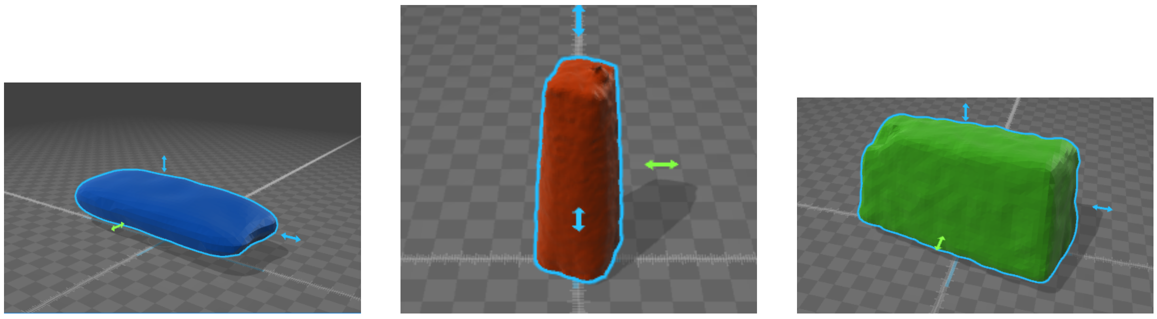

(a) Case model (b) Juice model (c) Tea model

Figure 3.5: Application models

will be saved into a Unity scene, this way if another project needs to reuse some of the parts of this one, has the possibility of reuse the scene or some of its components or the internal logic coded inside the scripts. Because in Unity a scene acts like a parent which all of the objects placed inside the scene inherit of. And if you import and load the scene all the stuff that inherits from it will be placed and have the same behaviour as if they were in the original project.

The representation of the virtual scene inside Unity is the same as in the majority of this kind of graphic frameworks. Is represented by a virtual grid with infinite cells in it, each cell represents four point of coordinates in the grid, placed in their corners. These coordinates are stored as Vector3 variables, so when an object is placed in the grid or rendered in runtime, it has a variable of this type saving his position in the grid. As we are working in 3D and besides the grid is a plane, objects can be placed above and under the grid. Because the grid is in the editor only to help the developer and to have reference points to place the objects inside the scene.

For this concrete solution in the virtual scene, the center of coordinates will be represented for the marker, this way we have a point in AR where the other objects can refer with relative positions.

In the Figure 3.6 you can see the grid of a Unity scene, in this concrete grid the red cube represents the origin of coordinates. The squares in the grid represents point of coordinates, and the axis in the top right corner, show how the axis are aligned because it's possible to rotate the grid or change the perspective.

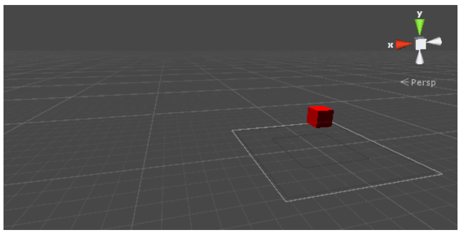


Figure 3.6: Unity Scene Grid

Chapter 4

Realization of the Augmented Reality solution

In this chapter is described how was the implementation process. This process will follow the guidelines and schemes presented on the previous chapter.

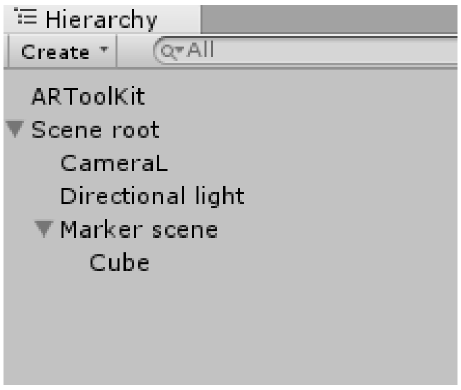
4.1 Implementation Tools

During this project for achieve the final solution, many frameworks, libraries and programs have been tested in order to decide which are better and to acquire strong knowledge of the basics of AR applications. Here is explained in detail which of these tools are included in the final solution and how. It has to be noticed that most of the basics of these tools have been explained in chapter 2.6, so here are only explained their tasks in the final solution and how to use them.

ARToolkit

These libraries are the one that allow the camera and the marker tracking. They can be used alone with raw C++ code or including them in Unity. For using them inside Unity the integration is easy, because they are compiled as a unitypackage which is a type of file that can be imported inside Unity directly. When they are imported is necessary to do a basic setup of the Unity project to make them work properly. This setup consists in two main things: the first one is the calibration of the camera which will be explained in the next chapter, and the second one consists in placing several game objects with some scripts attached to them inside the scene and then to link them by telling them which marker they have to detect and some additional stuff. Basically is necessary to include some game objects as the Figure 4.1 shows. These game objects have attached to them the scripts that come with ARToolkit with the same name, then the last thing is to say to the marker object which marker has to be detected, to do this we have to put in a folder named markers the proper marker and to assign one of the variables of the script with the name of that marker. For the first time is possible to use the default markers of ARToolkit that are in the folder, but is also possible to use external markers.

First of all if only ARToolkit for Unity in a basic way is needed this step is not necessary. But for the solution was necessary to do a proper calibration and to include additional markers, so was necessary to install ARToolkit as a SDK. For doing that the first thing is



ARToolKit
Scene root
CameraL
Directional light
Marker scene
Cube

Figure 4.1: Unity Solution Hierarchy

to install the dependencies of ARToolkit SDK. These dependencies are: OpenGL, libjpeg, GLUT 3.7.6, OpenCV and also is necessary to have installed Microsoft Visual Studio 2013(in the windows environment), some of the dependencies where installed previously in the system but others not. Then only the installation is needed, with the executable that is avaliable to download in the home page of ARToolkit.

Unity

One of the goals of the project is to integrate the solution with Unity, so is obvious that one of the tools is Unity itself. Unity is a framework but also is his own IDE, this means that the solution has to be developed with this IDE. Installation of Unity is easy, just a download of a executable and some normal installation steps and it's possible to start developing with it.

How to use Unity is much more complex than the installation, because of the variety of uses and tools that Unity has. For this solution the use of Unity is placing game objects, attaching them scripts to control the behaviour of the solution and also attaching models to the game objects to obtain the 3D visualization.

ROSBridge

This library is not for AR developing but for obtaining the online data of the robot, because the robot works with this library and the data is codified by it. So for including this library in Unity, the only step required is to download the library and place it inside Unity's project folder and then inside the scripts when is necessary to use some of the functionality of the library is as simple as import it at the top of the script.

SimpleJSON

Unity doesn't come with native JSON support, so to parse the robot data that comes in a JSON format to a format that Unity can handle, is necessary to have something that does that parsing. SimpleJSON are some libraries for multiple programming languages that provides support to work with JSON. In Unity only is needed to download the libraries, put them inside the project folder and import them where they are needed in the code.

4.2 Preliminary Work

For achieving the final solution and results a long way of testing and doing preliminary work has been done. In this section the details of all this preliminary work are described. There are three different parts in this preliminary work, the first one was with Vuzix SDK, the second one with 3ds Max and the last one with ARToolkit SDK and Unity combined with ARToolkit.

4.2.1 Vuzix SDK

At the beginning was made a process of testing how the Vuzix glasses works, how to use the SDK, and how to develop with them. The SDK is from 2010 so it's necessary to have Visual Studio 2010 to compile the libraries and the samples of the SDK. These libraries and samples provide methods and functions to track the camera and to make 3D stereoscopic VR applications. As the SDK does not provide marker tracking, the tracking of the camera is done using the sensors of the device. Basically the SDK uses the information of the accelerometer, the gyroscope and so on to calculate the pitch, roll and yaw of the headset, is possible to access to this data using a function called IWRGetTracking that returns three values between -32768 and 32768. These values need to be converted to degrees in order to get the position for using it. There are some other functions to get information about the headset but that one was the most important. Also the SDK provides stereo VR rendering using OpenGL or DirectX9. The rendering is done initialising the device into stereo mode with IWRStereo_Open then is possible to render 3D stuff using glVertex, matrix, and basic geometry like glutSolidSphere. These functions use vectors with three points of coordinates to render 3D geometry, also with the matrix is possible to use multiple vectors to render planar figures. Then the color is assigned using float variables that refers to the red, green, and blue colours.

It's important to notice that OpenCV, Windows Platform SDK for Windows Server 2003 R2, OpenGL, and freeglut 3 have to be installed. Also when compiling some application may appear some compilation errors because the library dependencies of the SDK are compiled for Windows XP or 95 so if this errors appear is necessary to change the variables WINNT and WINVER to the version of the OS that the developer is using, for example in windows 7 or later is 0x70X where X may change depending on the concrete version.

4.2.2 Autodesk 3ds Max

Autodesk 3ds Max was tested to see how it works with AR solutions and because it comes as one of the Vuzix glasses ways of developing with the SDK. Only is possible to develop with the 2011 version, because is the only one that supports the plugin of Vuzix. So first thing is to install 3ds Max 2011 and then add the plugin of Vuzix called MaxReality. For doing this is necessary to install the plugin in the OS and then add it with the Utilities tool of the IDE of 3ds Max. Once this is done the steps to developing and AR application with 3ds Max are the same as developing a normal application. These steps are: setting up the models and 3D objects inside the grid, adding cameras and additional effects, adding scripts for the behaviour and compiling the application.

Adding the components to the scene

The first step is to add the objects that are going to appear in the scene in a static way. This is because objects can also be added in runtime via scripting but this has a higher cost of resources, so if something has to be on the scene is better to add it before the compilation. Adding the objects to the scene is done in two steps, the first is to add the mesh of the object, the mesh is like the model of the object but without any texture. This mesh can be designed with 3ds Max or imported from a pre-created model. The process of modelling a mesh is like making an sculpture in a virtual 3D environment. The second step is to assign textures to the meshes, this is like attaching some property to the mesh object. Textures are normally plain images that are painted with the computer by a designer, the image contain the unfolding of the mesh painted. When these two kind of objects are assigned one to another and put it together inside the scene, the project is ready to go to the next step.

Putting the cameras

With all the objects inside the scene, is necessary to add the cameras and the post process effects. The cameras are the points of the scene where the user can have a look into the scene, in the special case of AR only one camera is necessary because the user will look around and see the whole scene. The post process effects are the lights and effects that can be placed inside the scene like a explosion, some object breaking into pieces, raining... They can be added at runtime but with more resource cost, or can be added before compilation but hided and activated in runtime with a specific event or at a specific time.

Creating the behaviour

The last thing that is necessary to do for completing the scene is to create the behaviour of the scene itself. It can be done in two ways: graphically or by scripts. The graphical way is using the tools that 3ds Max provide to determine if the objects need to move to somewhere if they have to rotate, add some graphical events like when this objects goes through that point move other object, etc. For doing this the other way is necessary to create scripts using MaxScript which is the scripting language used by 3ds Max, and putting there the behaviour, then attach the scripts to the objects and execute the scene.

4.2.3 Unity and ARToolkit

With ARToolkit was two separate branchs of work, the first one using the SDK alone, and the second one combining ARToolkit with Unity.

ARToolkit SDK

The final task that was made before start developing the final solution, was to practise with ARToolkit SDK and Unity with ARToolkit integrated. Basically that practising consisted of running and modifying the samples, calibrating the device and developing basic AR applications.

When you download and install ARToolkit SDK, the first thing they recommend you to do is to run a sample called SimpleLite, that comes compiled with the SDK, and render a cube in the marker called Hiro. This is for trying if the SDK works fine for your AR device. When I used with Vuzix, the cube was rendered but there was some problems of registration because the cube was rendered in a bad position. So the next thing to do is to calibrate the camera, this is necessary for the SDK alone and for Unity with ARToolkit, because both of them uses the same file with the calibrated camera parameters.

For calibrating the camera has to be printed one marker that seems like a chessboard and comes with the SDK, it's called calibration chessboard, and then run a few configuration files. The main calibrating file to run is called calib_camera, this file must be executed and then look with the device to the chessboard. The device will display forty crosses in the corners of the cells of the chessboard. If those crosses are green, the image is bad because there is too much light or the camera is not tracking well the chessboard so is not possible to use the actual image to calibrate. But if the crosses are red the image is good and can be taken to calibrate the camera. The optimal calibration requires to take ten photos of the chessboard from all the possible angles. When the photos are taken the file that was running will export a file called camara_para.dat, that file contains our calibration settings and has to be placed in a folder called data inside the installation folder of the SDK or in the project folder of Unity. Also the calib_camera program will tell us if the photos taken was good or not, because all the crosses can be red but the image may have errors of calibration, so it's necessary to look at the output of calib_camera, this output will be by console and will tell us the approximately calibration error of every image. If this error is bellow one the photo is fine, but if it's more than one, the image has to be done again. Also the general error has to be bellow one. This measures are taken in pixels so the error represents the error in terms of pixels moved form it's position. To make the calibration the program calib_camera can be used with additional options, for example the resolution of the camera can be changed, but one thing has to be taken into account, the proportions of the resolution used by the camera during the calibration must be maintained in the future. For example if is used a 4:3 proportion it's not possible to use a 1920x1080 resolution because the registration will make mistakes and will cause bad tracking.

In Figure 4.2 a capture of the calibration process is showed. This capture shows the chessboard to calibrate the device camera. All the points are in red so it's a good picture for calibration, also it's appreciable that you have to take at least ten images to calibrate the camera in a proper way.

All the previous calibration process was for one camera, but if the device has more than one camera to support stereo rendering, another calibration process needs to be done. The additional calibration consists of running calib_stereo with every camera separately, the steps are more or less the same as with calib_optical. And then putting the data files produced inside the folder of camera_param. If the process was made well when the stereo sample is ran the rendering has to be well registrated.

Once the device is calibrated the next step is to modify the samples to see how the virtual scene is rendered using ARToolkit. ARToolkit uses OpenGL to render the objects, so to render the 3D scene using the SDK is only necessary to modify the render method putting there all the stuff that is necessary to render. The normal behaviour of ARToolkit program is to have a setup method for the camera that starts recording the images from the real world, a render method that has all the 3D stuff rendered with OpenGL, a main loop that calls the camera method and if the camera is fine then starts calling the render

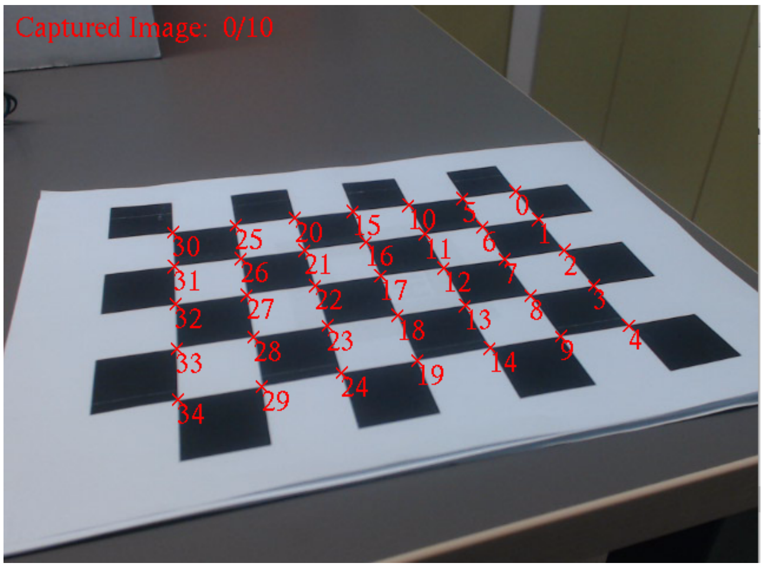


Figure 4.2: Camera calibration

method, and a init method that provides the camera parameters , the kind of application and more initial parameters. Also other methods are provided such utility functions for handling events and displaying help text but the main core of an ARToolkit application are the previous ones. Then only rest to compile and deploy the code which will generate a executable that depending on the initialising parameters will be executed on a specific window size with concrete camera parameters.

And additional feature that was tested was to import some image as a marker and use it on a application. For doing this is necessary to run the bin file mk_patt and then look with the camera to the image that we want to use as a marker. If the image is valid, it will be surrounded with a green and red line, then is possible to capture the image and save the data as .patt file that can be used in any application with ARToolkit. In Figure 4.3 we can appreciate this process, also is possible to see in the upper corner what pattern is identifying the calibration program.

Unity and ARToolkit

At this point having tested all the previous tools, the last thing to do was to integrate the AR library used with Unity as explained in the previous chapter. But before start developing the final solution a training and testing process of the framework was done in order to achieve good skills using the tools and to see how the development work with them.

The first thing necessary to do is to put the calibration files obtained when using the SDK inside the Unity project folder, because ARToolkit works the same way with Unity and needs this parameters to make a good registration. Then a start point are the scene samples that come with ARToolkit, they are samples for using a single marker or multiple ones, to make stereo rendering or to use NFT images, which are special images for markers. Always when modifying the samples or creating new ones the things that is necessary to

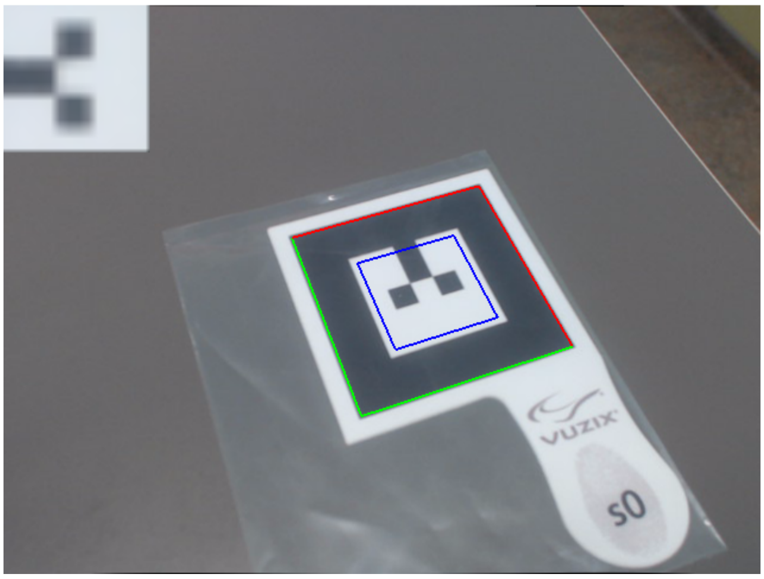


Figure 4.3: Marker training

be are the camera, the light, the marker and the objects. To test how to work with a scene is interesting to add objects into it and then to create scripts that control these objects. For example to make them appear or disappear, to change the models or the position at runtime. Also to add some effects, like particles or an animation to an object. All of this is done like in normal Unity project, but is the way to achieve a good knowledge about how Unity works.

4.3 Implementation Details

The implementation of the solution has three main parts: the setup of the Unity scene, the coding of the rendering controller, and the coding of the online receiver for the data. In this section is explained in detail the process of implementing these three parts of the solution.

It has to be taken into account that scripts in Unity come with two methods by default: the start and the update method. Start method is executed when the script runs for the first time, its like a initialising method. And the update method is executed once per frame, so, as the name says, it's and updating method for the scene. Those two methods come empty when a script is created, and most of the cases they have to be implemented, but is not obligatory. One important thing is that those two methods and some other functions can only be implemented or called in the main thread of the script, this means that if there are other classes implemented in the script, only in the main one can be implemented start and update or called some special functions of Unity. Also this has relation with the fact that Unity is not thread safe so it's not possible to use threads with Unity.

Also a final thing is that for the use of the application the details about the IP of the server and the path of the local file have to be wrote in a file called config.json located in the folder of the executable, or in the folder of the project. This file comes by default with the values of the development but when executing the solution in other computer may be necessary to change the values.

4.3.1 Unity Scene

When working with Unity is necessary to use his IDE, that means that in Unity projects one of the vitals parts is to setup the scene properly. The setup in case of AR applications with ARToolkit consists of the creation of several game objects that have attached AR-Toolkit scripts that allow the marker tracking. The first step is to create two objects, one is the scene root which has attached the script called AROrigin, this sets where the origin of coordinates is for the AR scene, other scripts can be attached to this object because is the father of all the others so the scripts placed here will have access to all the scene. In fact we will attach to the scene root the scripts to render the data and to get it from the server. The other object has to be one called ARToolkit that has attached the main script of ARToolkit in order to be accessible to all the others to use his functionality.

Next step is to put the basic objects of any Unity project, the camera and the light. Both of them will inherit from the Scene root. The light is a basic directional light object that comes with Unity, is necessary to put at least one light in all the Unity scenes if you want to see the objects clear, if a scene doesn't has light the objects will appear dark. The camera is another basic object of Unity, is used to set a point of view in a scene that can be accessed by the user, in the solution will be only this camera in the origin of coordinates with the ARCamera script attached that tells ARToolkit that this is the camera used to do the tracking. A basic configuration of those objects can be made, setting parameters like the brightness of the light, the field of view of the camera... In the solution is not necessary to use another parameters, with the default ones works fine, but for future works could be possible to modify them.

The last objects that is necessary to add, are the marker and a cube. The marker is a game object with the script ARTrackedObject attached, this script tells ARToolkit which particular pattern has to be recognised as a marker. In the solution is used the Hiro marker, which comes by default with ARToolkit, but any image can be used as a marker. Also is necessary to tell where the marker is, in our case will be the center of coordinates. The final step was to add a cube attached to the marker, this has the only purpose of showing where the origin of coordinates is in the application.

4.3.2 ROS Receiver

To face the problem of receiving the data of the robot online, was necessary to code a script in C# that connects with the server and retrieves the data. This data is coded using ROS Indigo which is a framework for developing solutions with robots, so in the same way is necessary to receive it and parse to JSON format. For doing this in the script is imported ROSBridge, this library is used to create a socket that subscribes to the server receives the data and converts it to JSON using SimpleJSON. So in the script the start method is used to create the socket and subscribe to the server using his IP and port. Then the subscriber listens to server in order to receive the message with the robot data, when this is done a function does a parsing to JSONnode format and then to JSONString. Then using an instance of the controller that will be explained on the next section, sends the data to the mentioned controller. Also was implemented a function to disconnect from the server when the application shuts down for make the resource free.

4.3.3 Rendering Controller

The most important part of the implementation, this script is the one that reads the data, process it and then decide what is necessary to render in the AR scene. This was implemented as a C# script attached to the scene root game object, this way the script has control over all the scene and his inheritors, in this case the camera, the marker and the 3D objects. For doing that only the main class of the script was needed, in this class with the start method the scene is initialised with the data that the server has sent, and then with the update method if there is a change in the data that the robot is transmitting then the AR scene is updated. Also the start method is used to create a shared instance of the controller class following a singleton pattern, this is made for allow the other scripts to call this controller and send him the data, because in Unity is not possible to create instances of the classes in external scripts.

So start and update methods are the basic skeleton of the controller, but they only call the functions and receive the values, the implementation of the rendering and the reading of the data is done in other functions, called renderData and readData. ReadData uses SimpleJSON to parse the data into a string that is readable by Unity and C#, the data is stored in a global variable called robotData. Then renderData reads the string in robotData and process it with a loop. This loop basically reads the properties of the objects that are in the string, like the position, rotation and the name, and with those properties creates objects in Unity and assigns the values to the objects, then renders the objects inside the scene. The objects can be created as Unity primitives which are basic geometry objects or using models that are stored in the projects folder.

4.4 Testing

It's clearly evident that testing is a vital part of the development process, and particularly in the development of this solution, testing has been a continuous activity more than a final one. This is because during all the process of creating the solution, tests has been made with different parts of the system, with different parameters, with different objectives... Maybe this is because as we are working with Unity, the system parts can be easily plugged or unplugged, because the objects are placed in the scene and the scripts that control the system are attached to them, so is possible to test multiple combinations of scripts and scene objects, to make the system strong and search for bugs or simply to assure that the solution works well. The main tool for doing these tests has been the Unity console, because in that console appear all the changes that the system is doing in runtime, all the warnings, and all the errors with detailed messages of their origin. Also is possible to write in the code calls to the console with the objective of debug the code and see that everything goes fine. All of this messages are divided on the IDE of Unity into debugging, warning and error messages, is possible to filter them and always contains information about which part of the system has provoked them, what is the reason of the message and the concrete line of code or concrete parameter or configuration in the editor that has failed or launched the message. In the case that you are using the debugger, is possible to add the messages a tag, or to create special messages if you want to filter them in the console.

Most of the testing has been done using the Vuzix device, since the final solution has to work for it. However in some tests that I ran on home I tried the application with the camera of the laptop. These tests where not optimal because the camera of the laptop is static and the objects are seen in strange positions but is enough if the test is about seeing that something is rendered properly or if something moves when updating. Here I can say that the rule is if it works with Vuzix it will work with the camera, but working on the camera does not assure that it will work with Vuzix.

One final thing to take into account is that all the tests have been done in an indoors environment, since is more practical to use markers indoors than outdoors and also the application needs a computer with the HMD connected. However is possible to use the application outdoors, the only necessary thing that maybe needs to be done is to recalibrate the camera with the lightning conditions of an outdoors environment. This is becuse the difference between indoors and outdoors may cause registration issues, that will result in an incorrect rendering.

For testing the system is necessary to modify the incoming data, so to do that I use a file called config.json in the application folder that has the data about the IP of the server and the path to a local file in the computer, changing this file allow to test different files or different servers. In the next section is explained how the system parts were tested.

Rendering Controller

This part of the system, as explained in the previous section, is the one in charge of render the data into the AR scene. It can be tested as an isolated part or with the Data receiver altogether. For testing it as an isolated part, is necessary to use a JSON file that has the same structure of the messages from the server. Using a file like that, this part of the system can be tested in several ways: seeing that the parsing and reading of the data is fine, doing several changes to the parameters of the objects to see if they change in the scene, doing changes at runtime to see if the updating is working fine... In Figure 4.4 we can see a test of this part using a file that has different rotation data of the objects to see if they are rendered properly into the scene. This test was done without the server connection, in order to try if the part that works over the local file works fine.

ROS Receiver

The testing of this part is clearly more simple than the previous one, this is because of his function. The function of this part is very concrete, connect with the server, obtain the data from the robot using a ROS socket and parse it to a JSON format, with the purpose of sending it after to the rendering controller. So for testing is possible to change the IP of the server to try different servers with multiple data. During the development I've had only the chance to test it against one server, because the server is external to this project and there was only one machine running with the data from the robot. In consequence I can only say that this part of the system can establish well the connection and retrieve the data but I haven't tested it with more servers. One remarqueable thing is that this was tested in the lab using an ethernet cable from the internal network of the University,

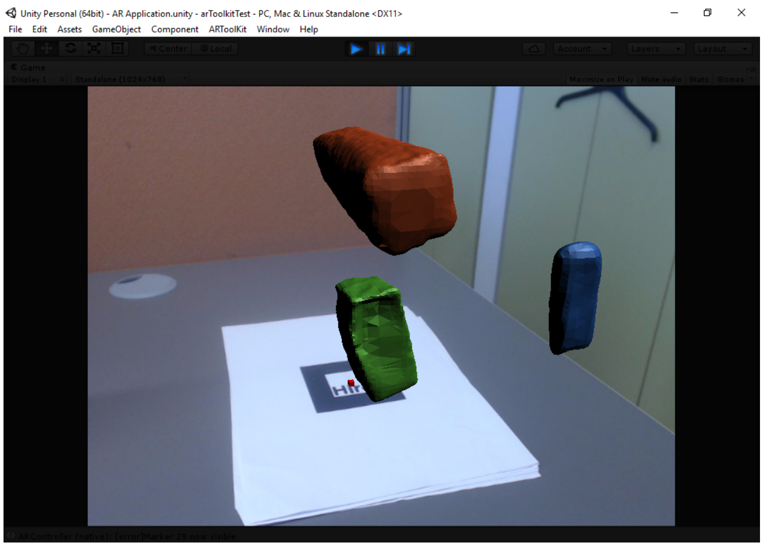

Unity Personal (64bit) - AR Application.unity - arToolkitTest - PC, Mac & Linux Standalone <DX11>
File
Edit
Assets
GameObject
Component
ARToolKit
Window
Help

Figure 4.4: AR Solution test

and the server was in the same network. I say this because i tried to do the same using eduroam, which is a Wifi network of the University and it doesn't work. I asked the person in charge of the server project and he told me that this is because of the security measures of the Wifi network, so for secure results is better to use cable networks in the same net.

4.5 Results

After all the work done in the design, implementation and testing, the application is capable of using the required device, obtain the data from the server, and render it into an AR scene. The solution has been made using marker tracking, so the tracking of the camera is done calculating his relative position to the marker that is placed somewhere in the real world environment. Also to get a good registration with the marker the device has been calibrated using some binaries of the AR library. Finally, the rendering uses augmented reality, which means that all the objects that are obtained from the data of the server will be augmented and displayed around the marker.

All the work of the solution has been done using Unity after integrating it with the device. The application connects to the server through a socket, gets the data and parses it, then the data is interpreted and computed into Unity parameters like coordinates, rotational values, and type of objects and rendered in a scene which will be augmented thanks to the marker. The connection is done using ROS socket, because the robot works with that libraries and the messages with the data are coded with them, so for translating the messages into JSON is necessary the use of ROS libraries.

The solution includes some additional features, like the possibility of use a JSON file instead of connecting to the server to get the data, the runtime updating if the data changes, and a configuration file that allows the user to specify the IP of the server(the port is 9090 by defect using ROS) or the path to the JSON file. Also one thing to take into account is that the rendering is really accurate, but the models have some lacks, because the data that the robot sends hasn't some parameters that are needed when rendering it into Unity like the scale or the default position of the object. So the objects may appear bigger or smaller or looking into another direction.

After all the implementation of the solution, is clearly realizable that it has his strong and weak points. Also it has many potential that can be implemented in future projects to make the application more complete and strong. During all the process of developing I have taken notes of all of this, so here I will specify the limitations of the actual solution and the future work to make it more complete.

Limitations:

- The solution needs a marker to display the data.
- The information about the models may cause differences with how are they seen by the robot.
- In order to achieve a good visualization is necessary to use a HMD.
- The calibration is not optimal so there are differences between the real camera and the virtual one.
- The information about the IP and the sample file is in a file.

Future work and updates:

- Possibility of making a GUI to select from various files of data or to specify the server parameters.
- Optimize the calibration to achieve accurate rendering of the virtual camera.
- Add a database with the specifications of the real objects to render more accurate models.
- Use markerless tracking to remove the limitation of the marker.
- Possibility to combine the solution with an manipulation device like leap motion to interact with the scene.
- Adapt the application to Linux or Mac systems.
- Can be added data of movement of the objects to the data of the robot, to render the translation in the application.

Chapter 5

Conclusion

This thesis has been done focusing on the Augmented Reality area, studying his principles and specializing in the development with Optical See-Through devices. The main objective was to use this knowledge to develop an augmented reality application that uses the device from Vuzix to render some data obtained from a robot, all of this development using Unity. This application will render the virtual layer with all the 3D models that the data contains. The system will consist of a Unity project with all the necessary to develop future AR applications and a demo created with this project.
The Unity solution has been designed with the purpose of allow future developments with augmented reality purposes. It can be changed easily to develop solutions which use the camera of the PC, or the smartphone instead of a Head Mounted Display. The Unity project contains scripts with all the methods necessary to render 3D objects on runtime in the scene, also methods to retrieve online data and patterns to connect some scripts to others. Also the project contains some sample scenes that cover most of the augmented reality cases of development. The application uses ROS to establish the connection and ARToolkit to do the tracking. The solution was tested in the lab using online and local data, also changing this data to update the scene. The models rendered in the scene correspond to real objects computed by the robot.
Finally I have to admit that the development of all the system using Vuzix hardware has been a long and hard process because of the lack of documentation, but it has given me many skills and experience, and I'm proud of it. I've learnt a lot of the Augmented Reality field, beginning with his history, possible usages, future and ways of development. Also I've learnt C# which is a language that I've never used before and I improved my C++ skills. And of course I learnt much about Unity and 3D frameworks because i was a newbie in this area. So, to sum up I've to say that now I'm far more experienced with all this tools and fields of development than when I started. For all of this if I had the chance of doing this project again I will take it for sure.

Bibliography

[1] Adnan Ansar. *Registration for augmented reality.* http://repository.upenn.edu/dissertations/AAI3031635/. [Online; cit. 2016-05-15].

[2] Andrew I. Comport, Eric Marchand, Muriel Pressigout, Francois Chaumett. *Real-Time Markerless Tracking for Augmented Reality.* http://www.irisa.fr/lagadic/pdf/2006_ieee_tvcg_comport.pdf. [Online; cit. 2016-05-14].

[3] Borko Furht. *Handbook of Augmented Reality.* Florida Atlantic University, 2011.

[4] Collective. *Difference Between Marker based and Markerless Augmented Reality.* http://stackoverflow.com/questions/27229465/difference-between-marker-based-and-markerless-augmented-reality. [Online; cit. 2016-05-15].

[5] Daniel Wagner. *Handheld Augmented Reality.* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.245.413&rep=rep1&type=pdf. [Online; cit. 2016-05-15].

[6] Denis Kalkofen et al. *Visualization Techniques for Augmented Reality.* http://imd.naist.jp/imdweb/pub/kalkofen_bookchapter11/paper.pdf. [Online; cit. 2016-05-14].

[7] Dennis Joele. *Development of an Augmented Reality system using ARToolKit and user invisible markers .* http://mmi.tudelft.nl/~vrphobia/RA_final_report_Dennis_Joele.pdf. [Online; cit. 2016-05-14].

[8] Dongdong Weng, Dewen Cheng, Yongtian Wang, Yue Liu. *Display systems and registration methods for augmented reality applications.* http://www.sciencedirect.com/science/article/pii/S0030402611003159. [Online; cit. 2016-05-14].

[9] Eva Hornecker, Thomas Psik. *Using ARToolkit markers to build tangible prototypes and simulate other technologies .* http://www.ehornecker.de/Papers/ExpGestInteract05.pdf. [Online; cit. 2016-05-14].

[10] Georg Klein. *Visual Tracking for Augmented Reality.* http://www.robots.ox.ac.uk/~gk/publications/Klein2006Thesis.pdf. [Online; cit. 2016-05-15].

[11] Hirokazu Kato, Mark Billinghurst, Rob Blanding, Richard May. *ARToolkit manual.* https://www.google.cz/url?sa=t&rct=j&q=&esrc=s&source=web&cd=19&ved=0ahUKEwjN2PuAv9LMAhVGVRoKHZi2Di04ChAWCGgwCA&url=http%3A%2F%2Fwww.cs.vu.nl%2F~eliens%2Fmanuals%2Fart-pc211%2FManualPC2.11.doc&usg=AFQjCNHwti1sjegovEKDQWiohuMUEel_YA&sig2=0WwFPLcHjj7Ii41cUcMSIA&bvm=bv.121658157,d.d2s&cad=rja. [Online; cit. 2016-05-15].

Appendices

List of Appendices

Appendix A

Content of CD

\ARproject	Unity project of the solution with all the sources
\demo	Demo application
\markers	Markers for AR applications
\doc	Source files of the technical report in $\LaTeX$
xlazar09-dp.pdf	Text of the report in pdf
readme.txt	Hints for the content of the CD