VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Bachelor in Computer Sciences

ROBOMR: A MIXED REALITY ROBOTIC GAME

ARTHUR VINCENT CHOMÉ

Academic year 2018–2019

Promoter: Prof. Dr. Beat Signer
Advisor: Payam Ebrahimi, Ahmed K.A. Abdullah
Faculty of Sciences and Bio-Engineering Sciences

VRIJE UNIVERSITEIT BRUSSEL

Afstudeer eindwerk ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Bachelor of Science in de Computerwetenschappen

ROBOMR: A MIXED REALITY ROBOTIC GAME

ARTHUR VINCENT CHOMÉ

Academiejaar 2018–2019

Promoter: Prof. Dr. Beat Signer
Advisor: Payam Ebrahimi, Ahmed K.A. Abdullah

Faculteit Wetenschappen en Bio-ingenieurswetenschappen

Abstract

In this thesis, we propose a framework for the navigation of virtual agents inside a dynamic robotic Mixed Reality environment. It will allow virtual objects to track a physical robot using spacial mapping. We will illustrate the findings of our developed framework by realising a Mixed Reality robotic game where a user can steer a robot with a head-mounted device. Using path-finding algorithms, our application will find the shortest and most efficient path that the robot can use to reach its destination. The framework we want to develop would allow virtual agents to track and follow the robot around. The user can interact with these virtual objects by using the real-life robot. This ensures that there is interaction between the environment's physical and virtual objects.

Acknowledgements

I would like to thank my advisor Payam. He gave me great feedback for completing my research paper every week again and this in spite of his other responsibilities as a PhD student. I had no experience in writing a formal research paper and gathering good references proved to be tricky at times. Luckily, Payam gave me good advice at every step of the process to prevent rooky mistakes. He was a great advisor overall.
The same can be said about Ahmed. He gave good recommendations and showed great patience when mistakes occurred. He also introduced me to LateX[1], Google Scholar[2] and many more handy tools for writing a paper. In short, a very good advisor.

[1] https://www.latex-project.org/
[2] https://scholar.google.be/

Contents

1 Introduction

Video games and computer graphics have come a long way from their inception in the 1950s[1]. Going from simple vector graphics to fully 3D rendered games, the player's gaming experience and immersion has increased dramatically over the past decade. The next step in the search for greater immersion is to completely surround the user in the game space instead of limiting it to the screen. This can be done with Mixed Reality giving the user the impression of being physically present in another world.
Mixed Reality is a set of technologies that involves combining elements of the real and virtual world to create a new world where the two can interact with one another. As indicated in Migram's 1994 paper[14], Mixed Reality can be classified somewhere in between the real and virtual environment on the reality-virtuality continuum. The two sides of the spectrum refer to an environment where everything perceived is part of the real world and an environment where everything perceived is computer-generated (virtual reality) respectively. This makes MR a pretty broad research field.

[1] `https://en.wikipedia.org/wiki/Video_game_graphics`

In recent years, there has been a surge in public availability for Mixed Reality technologies. For Augmented Reality alone, market forecasts predict a total revenue of around 80 billion US Dollars by 2021[5]. Research for MR however has existed for a considerably longer time going back as far as the Virtual Fixture system developed in 1992 at the United States Air Force Research Laboratory [2]

The goal of our thesis is the development of a framework for navigation in an MR environment that lets virtual elements hover around certain physical objects. We will demonstrate our findings by developing an MR robotic game where one uses a Microsoft HoloLens[3] to steer a robot around that is able to interact with the digital world. An example of this can be found in the problem identification section.

[2] https://en.wikipedia.org/wiki/Virtual_fixture

[3] https://en.wikipedia.org/wiki/Microsoft_HoloLens

1.1 Problem Identification

During the Build Developers Conference[4] held by Microsoft in 2015, Alex Kipman demonstrated the HoloLens' cross-space capabilities involving the B15 robot [5]. In the demo, we can see the user control the B15 by indicating waypoints that the robot improves with path-finding. They also managed to overlay a holographic robot on top of the physical robot that follows the real robot around based on the vision of the Microsoft HoloLens. This was done in order to make the experience more immersive.

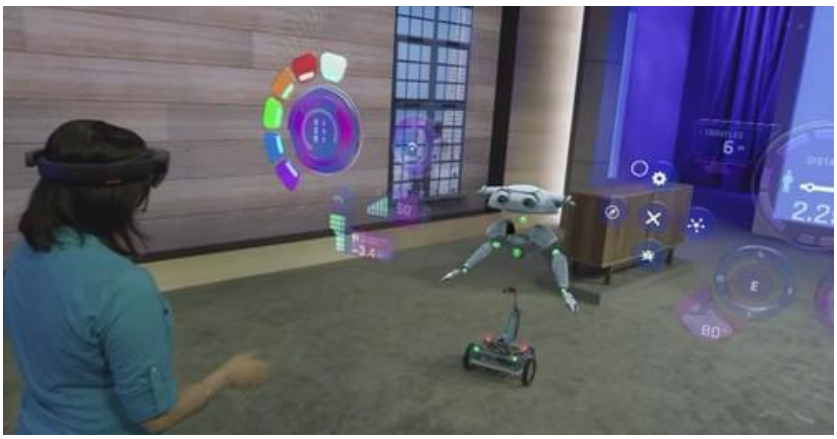

Figure 1.1: Using the HoloLens and a virtual interface, the user can specify a path for the robot to follow. This image was adopted from the Youtube video in footnote.

The issue here is that not all the source code of the project is freely available to the public. The whole goal of the research thesis is to develop a software application inspired by this Microsoft project and make it freely available for researchers to use and build upon it. However, there are some problems we have to account for namely navigation and the interaction between the virtual agents and the robot.

1.1.1 Navigation

To the best of our knowledge, there exists no publicly available framework for navigation inside a dynamic environment. There is —however— a master student who for his final thesis has to develop a similar MR framework with the ability for navigation at the WISE Lab[6]. Unfortunately, his work is not

[4] https://en.wikipedia.org/wiki/Build_(developer_conference)
[5] https://www.youtube.com/watch?v=xnrHFV34PfM
[6] https://wise.vub.ac.be/

ready for release. This is why developing such framework will be the focus of our thesis.

1.1.2 Interaction Virtual Agent and Robot

From the perspective of the user, there should be a seamless interaction between the robot and the virtual objects. The robot should be able to see the game objects and interact with them based on what it sees from the head-mounted device of the user. The perception problem also applies to the game's virtual objects: what vision do they have of the physical robot?

1.2 Objectives

Having dealt with the problems of implementing a Mixed Reality application similar to Kipman's demo, our research question is thus as follows: "Can we make a framework that would display similar features to Microsoft's demo and make it available for researchers?". For solving the stated problems, we have to formulate some objectives.

Navigation Framework

We will have to define a navigation framework for a dynamic Mixed Reality environment. It should allow virtual objects to track the physical robot. It will require some efficient path-finding algorithms as well as spatial mapping and tracking algorithms.

System Design

I would like to explore the feasibility of implementing an alternative system architecture than the classic client-server model[7] for my framework. We could alter the configurations as to make the objects of the virtual space and the objects of the physical space equally important. Making the application peer-to-peer based as to spread the computation over multiple components could help in this objective.

[7] `https://en.wikipedia.org/wiki/Client-server_model`

1.3 Methodology

For this thesis, we have chosen to use the Design Science Research Methodology for information systems applied to Mixed Reality systems[16]. We will clearly elaborate our activities for each step:

- Problem identification and motivation: this has been done in section 1.1.
- Define the objectives for a solution: the objectives for the success of this research effort have been specified in section 1.2. The overall objective would be to develop a navigation framework for MR and explore possible architectures to do so.
- Design and development: the design and expected features for our solution has been established after repeated meetings and discussions. Chapter 3 gives a more high-level explanation to our solution while chapter 4 discusses the more low-level implementation details.
- Demonstration: we will provide a demo of our software application during the final thesis presentation.
- Evaluation: we will assess to which extent the developed product answered to the stated problems. The goal is to disclose the usability of the robotic game and —as a consequence— our framework. We will also discuss development issues that we faced. This can be found in Chapter 5.

The focus of this phase is on the problem identification and defining the objectives. We will also provide a brief overview of the technologies we plan to use in the second phase. The other steps —like the project development— will be covered in that second phase.

1.4 Research Paper Outline

In line with our research methodology, the thesis report has following structure:

Chapter 2: a discussion of the general concepts used for the project by referencing previous research papers. This related work sections helps us in specifying our thesis requirements. It is the basis for the offered solution of chapter 3.

Chapter 3: a presentation of the final mixed reality robotic game with a focus on the developed navigation framework: its features, requirements, relevance for future research by other scientists...

Chapter 4: the implementation of the framework and the subsequent application: this includes its system architecture, design choices, used tools and technologies and a discussion of the programming process.

Chapter 5: a critical reflection on the development process leading to the final application and an evaluation of the final result.

Chapter 6: a conclusion to the paper. We repeat our solution and in which ways it deviates from the original requirements and discuss the possibility of future work.

2
Related Work

Now that we identified our problems and formulated our objectives for the research topic, we will do a literature study to refine them. The found related work represents the current knowledge the scientific research community has about the field. The discussed papers focus on Mixed Reality for robotics, navigation in a Mixed Reality environment, computing architectures, target selection, occlusion in Mixed Reality and the Microsoft HoloLens. This broad selection will allow us to either validate or refine our research question and the objectives. Writing a brand-new paper about something that has already been researched before would be redundant. Our research effort should be useful and allow researchers to build upon it.

2.1 Overview

Mixed Reality technologies allow us to perceive things that others cannot through an enhanced view of our surroundings. Its utilities and applications are very broad from simulating real-life tests for robots[7] to pedagogical uses like educating an assembly line worker[5].
An application we could more relate to, would be the popular game Pokémon GO[1]. It is known for bringing augmented-reality technologies to the public,

[1] https://en.wikipedia.org/wiki/Pokémon_Go

increasing its accessibility. The game allows players to explore the world enhanced with computer-generated Pokémon like Pikachu or Charizard. They can proceed to Pokéstops that function as the game's general stores and to Pokégyms to battle other players. All of this gets displayed on the screen of the user's smartphone. The player can also move to wild Pokémon to try and catch them[2].

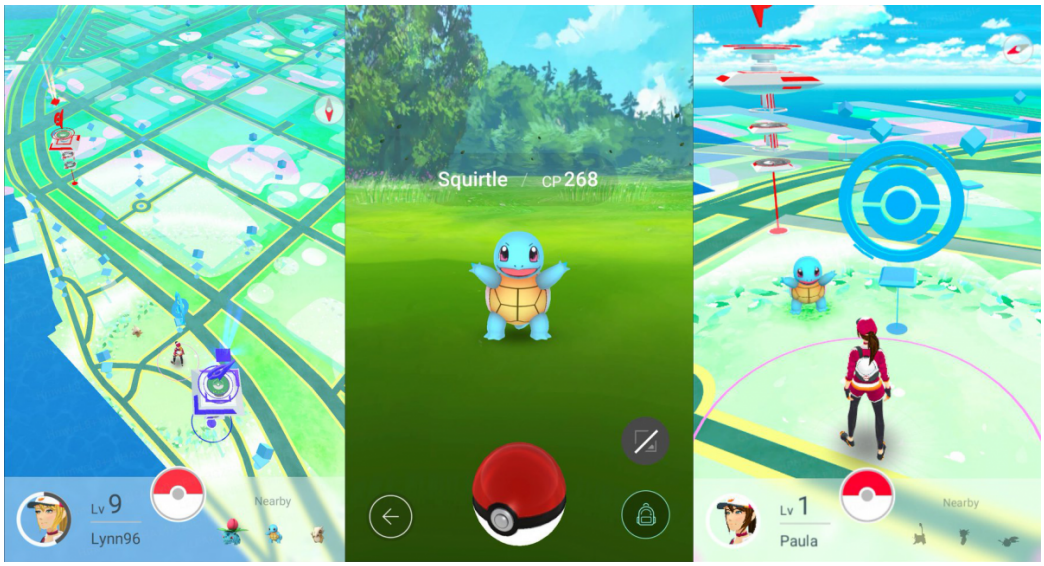

Figure 2.1: These screenshots show different views for the player. To access Pokéstops and Pokégyms, the player must travel the real world. His mobile screen shows his current position allowing for orientation. These pictures have been taken from Edubilla[2].

2.1.1 The Microsoft HoloLens

The Microsoft HoloLens is a head-mounted device that allows for the mapping of virtual agents over the physical world and it will be a key technology for our application. It has been such a success for Microsoft that it spawned a collaboration[3] with technology firms Asus and Samsung. These two firms now develop their mixed-reality software applications around the specifications of the HoloLens.
The paper by Gabriel Evans[5] evaluated the HoloLens on an application

[2] http://www.edubilla.com/articles/play-and-games/people-goes-crazy-behind-the-augmented-reality-game-pokemon-go/

[3] https://en.wikipedia.org/wiki/Microsoft_HoloLens

for educating people to use an assembly line. Users would get directions on where every piece of hardware should go to assemble a product, making for a very educative experience.
The conclusion of the paper is that the HoloLens is a more than capable device to deliver a Mixed Reality experience. The hardware capabilities of the HoloLens "allowed for virtual content to be spatially located correctly enough for assembly instructions". The display of the head-mounted device was also more than sufficient for the very detailed user interfaces programmed for the application to improve ease of use.

2.2 Robotics

Referencing the work of researcher Wolfgang Hönig[7], we can see that Mixed Reality has come a long way in robotics. The combination of these two disciplines has proven to be a great way to come to even more new findings. This follows out of Hönig's statement that "the central contribution of this work is to establish Mixed Reality as a tool for research in robotics".

2.2.1 Benefits

With Mixed Reality, we can create an environment where physical objects seamlessly interact with virtual objects. It would therefore be feasible to make physical robots interact with virtual components. This brought forth some very neat advantages for the involved scientists in the field of robotics improving their research and overall productivity.
Scientists can now witness remote tests performed on their built machines by way of Mixed Reality. They no longer need to travel great distances to the testing grounds as they can follow everything through a head-mounted device. They thereby meet in one virtual environment instead of a physical one. This improves their overall presence and collaboration considerably since there are no more travel distances between collaborators. This situation is described as spatial flexibility[7] and refers to a situation where "interaction between physical and virtual environments in MR allows experiments with robots to be performed remotely". This is all made possible by the recent advancements in internet connectivity.
Another advantage is the enormous safety benefits linked to using virtual environments. No longer will human participants be put in potentially hazardous situations by interacting with experimental machines. These machines are still to be tested and could potentially show unexpected behaviour dangerous for the humans involved. Using humans for testing would be a

thing of the past and experiments would only use virtual models.
MR applications have the ability to add computer-generated information to their environment that can easily be changed at will. This would make for much richer testing grounds for the experiments. “This expedites and simplifies debugging because the difference between implementation and simulation becomes even smaller”.

2.2.2 Experiments

What follows are practical experiments extracted from Hönig’s paper[7]. The research team conducted three experiments using cameras mounted on Unmanned Aerial Vehicles[4] (UAV).These are so called "drones" and are aircraft that do not have a human pilot on board. All control of these devices happens remotely.

UAVs Following a Human Model

The first experiment[7] focused on making drones follow a virtual simulation of a human around. But since human movement is “complex and unpredictable”, researchers believed their whole experiment to be not “very accurate”. The whole purpose of the experiment was to use human models with MR to limit safety risks since no real humans would actively participate with the experiments. It is worth mentioning that the team used the Unity game engine[5] which we will use extensively in the second phase of the thesis.

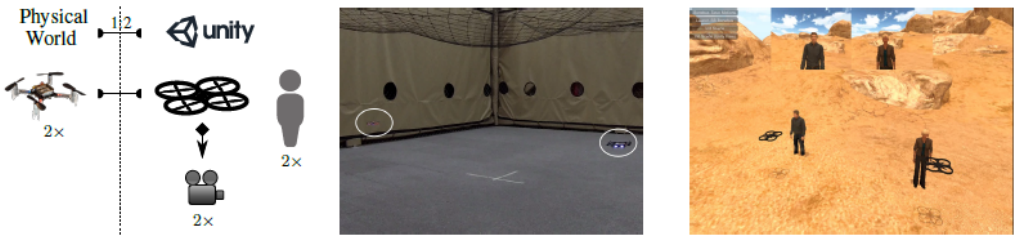

Figure 2.2: Above pictures show 3 views on the experiment. From left to right: abstract image of the used Mixed Environment, a view of the physical environment and the virtual environment generated by the Unity engine. With MR, the two quadcopters follow virtual objects around who simulate human movement. These 3 images have been adopted from Hönig’s paper[7].

[4] https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle
[5] https://en.wikipedia.org/wiki/Unity_(game_engine)

Spreading UAVs on a Wide Area

For the next experiment, the team implemented a swarm algorithm that managed the spreading of drones over a wide MR environment. In the environment, its computer-generated elements continuously changed to challenge the swarm algorithm on its robustness. This way, the team could test how a group of drones coped with an ever-changing environment. It "allows for swarm algorithms to be tested in a simulation that more closely depicts the actual physical motions of a robot".

Figure 2.3: 3 views on the experiment. From left to right: abstract image of the MR environment, the physical environment with the drones spreading out and the virtual environment representing what the drones see. The pictures have been extracted from Hönig's paper[7].

Robot-tracking UAV's

The final experiment involved controlling two ground-based mobile robots based on a drone's camera vision with the purpose of moving a box around in a specific direction. This was realised using technologies such as augmented reality tags[6] that calculate the UAV's camera position relative to the robot and some advanced image processing algorithms. This is quite useful for our paper since it highlights key technologies that we can use for tracking and moving our own robot using some off board camera. The drone would in this case be the user with the HoloLens.

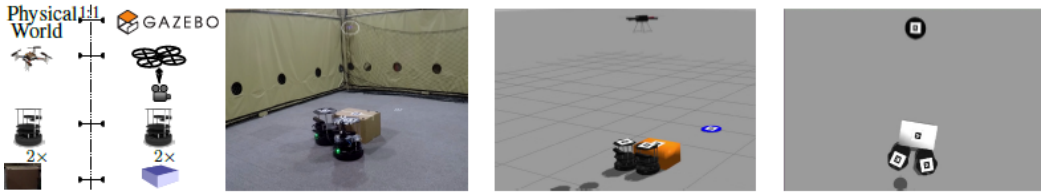

Figure 2.4: 4 views on the experiment. From left to right: abstract image of the experiment, the physical environment with the box and the two robots, the virtual environment and the virtual view of the quadcopter hovering above the robots. These pictures have been extracted from Hönig's paper[7].

2.2.3 Evaluation

The 3 experiments are interesting applications of MR. The first experiment tackles the issue of tracking a virtual object by a physical machine. Be it a drone, be it a Lego robot. The successful tracking could be applied on our project where the robot interacts with the virtual "targets".
The second experiment focuses on ever changing environments. Making our application robust enough to handle such surroundings could improve its reliability considerably. In the context of our MR game, this is handy when there is a lot of virtual movement that could put some strain on the robot. Think about higher levels with a lot of "enemies" to shoot down.
The last experiment involves making ground-robots move a box by way of visual perceptions from a UAV camera. In our application, it must be possible for the user to move the robot to any location he wants. This is one of the key requirements for our final robotic game. Though the controlled robots have the objective to move a box in a certain direction, this can be easily generalised to user-controlled movement in any direction.

[6] `https://en.wikipedia.org/wiki/ARTag`

2.3 Navigation

Navigation[7] involves determining the position of an object in an environment and directing it to another specific position that you would like that object to be at. We will be focusing on the navigation of virtual objects and the user inside an MR environment. To do so, we will analyse what the scientific world already knows in this field and aboard a few papers that might serve our cause. We will conclude by revising our research objectives based on our findings.

2.3.1 Navigation for Humans

Navigation in MR focuses on providing appropriate information to the human users about their surroundings to more easily move across it. This way, users can keep track of the environment without the issues related to disorientation due to the added strain of computer-generated information.
We reference Cheng's paper[2] for further insights. The paper presents a Mixed Reality application that offers a tour of the Next Gene 20 Project[8] in Taiwan. It would allow the users to explore the built projects in a more informative and immersive fashion all while correctly navigating them through it. This would be perfect for our robotic application. A possible feature we could have in our game based on these possibilities, would be of assisting the user of he ever were to lose track of the Lego robot. The system would then give feedback to the user to correctly navigate him or her back to it.

Next Gene 20

The Next Gene 20 Project[8] is an International Architecture Project from a few years back. It was hosted in north-eastern Taiwan and consisted of a group of both local and foreign architects co-operating to build different houses following the landscape architecture[9] style. Similar to the Museum of the Future[8], this project focuses on enhancing the user's experience by adding computer-generated information to the site he is visiting. All this with head-mounted devices. The whole project shows us that current MR technology is more than capable of navigating a human through an environment with added computer-generated information.

[7] https://en.wikipedia.org/wiki/Navigation
[8] https://inhabitat.com/next-gene-20-project-taiwan/
[9] https://en.wikipedia.org/wiki/Landscape_architecture

Figure 2.5: Stourhead in Wiltshire, England. It represents a typical work of landscape architecture. The image was adopted from a Wikipedia article[9].

Added Technology

To add to the user's immersion, the paper questions if an MR interface can do something more than offering the traditional input tools like a mouse, keyboard or joystick. Even for 2008, more technologies surged that added to the user's experience.
The Mixed Reality system introduces a new hand-held device that gives the user more information about his surroundings if he were to place it on certain locations. "Tangible controllers move around on the real navigation map to create perfect connection between the navigation map and 3D virtual reality environment." This technology makes the user's experience more informative as a whole.

2.3.2 Navigation for Virtual Components

The previous section has shown us that our current knowledge allows us to perfectly help a human navigate in a mixed environment. It even showed us that his tools for expressing himself (user input) has come a long way from the traditional mouse and keyboard devices. But how about navigating a non-human, non-physical object? A computer-generated object that does not exist but in the enhanced reality we try to build up?

Tracking Physical Objects: Effective Mapping

To navigate a virtual object in a real-life environment, we must be able to correctly and precisely perceive the physical surroundings through our sensors. We will refer to Richard A. Newcombe's paper[15] that presents a new surface mapping and tracking algorithm. It would allow us to overcome one of the biggest issues for Mixed Reality[10]: the incorrect depth calculation of the environment resulting in poor mapping of virtual objects and a poor performance for the whole application.
The featured KinectFusion method is able to do a real-time mapping of the environment using only a hand-held depth camera. Applying this method on our application with the Microsoft HoloLens would make it more robust since KinectFusion delivers a more complete mapping of its environment. Overall, this seems like a good mapping algorithm for our application.

2.3.3 Path Finding Algorithms

So far, we focused on the perception the virtual agents and the robot must have of each other to effectively cooperate. We also discussed the navigation of the user and the virtual agents relative to the robot together with some basic environment mapping.
We will now go in more detail in the navigation of the robot and more precisely the path-finding algorithms it relies on. In the application, the user must specify a path it has to take that must also be improved if necessary by these algorithms.
We will refer to Abiyev's 2015 paper[1] for further insights on the issue. In it, the researcher and his team compare a set of path-finding algorithms with different complexities and performances.

Experiment: Robotic Soccer Game

The experiment in Abivey's paper[1] resolves around optimising a robotic soccer game[10] for 12 fastly moving robots in teams of 6. This is done —like mentioned above— by comparing a set of different algorithms while presenting an improvement of an already existing algorithm. The visual perceptions are supplied by two overhead cameras positioned 4 meters above the soccer field. Path finding is discussed within the context of a very quickly changing environment where opposing robots obstruct newly calculated paths and steal the ball from one another.
Though some discussed algorithms are slower than other ones featured in

[10] `https://www.youtube.com/watch?v=qNaBUs7gP_A`

the paper, they are less complex to implement and still fast enough for our application. We will be focusing on those slower algorithms. To make our robotic game applicable on even the most difficult environments, we will also discuss the improved algorithm featured by the paper.

Figure 2.6: Using omni-wheel robots and some fast path-finding algorithms, a robotic soccer game gets realised. This image has been taken from a Youtube-video[10] of the Robocup Robot Soccer Finale of 2013 in Eindhoven.

A*-algorithm

Th A*- algorithm discussed in the paper[1] is overall very popular for solving the path-finding problem. The algorithm works by starting at a begin point in the environment and tries to find a route to a specific endpoint with waypoints in between. Every waypoint has a specific distance to cover to reach it from a neighbouring waypoint. During the algorithm's execution, we maintain a tree of paths that all start from a specific begin point. At each iteration, the algorithm adds another node to it until a path to the end location has been found. Though not as efficient as other algorithms in this paper and too slow for a robotic soccer game where the environment changes a lot and quickly, it is still a great fit for our goal. Especially since it does not need any prior knowledge of the surroundings it gets applied to. This is a big selling point for our application since it has to be applicable on any surroundings. Another thing to consider is that the environment is fairly static: there should not be any fast moving entities in our environment that block up paths we created for the robot forcing us to find alternate paths.

RRT Smooth

The abbreviation "RRT" stands for "Rapidly-exploring Random Tree". It is a path-finding algorithm that works by arbitrarily generating trees that fill up our problem space. The original algorithm quickly finds paths between two locations but it is sometimes not the shortest possible path that can be found. Worse still is that the paths found can sometimes be quite long and overall quite bad.

Iterative RRT Smooth

The paper[1] therefore introduces an optimised iterative version that finds “feasible and near optimal solutions in short time and shortens the path length considerably”. It is part of the more complex algorithms featured in the paper. Since the user can move around changing the camera angle and making robot tracking more difficult, it can be useful. Especially when there is a lot of user movement.

2.3.4 Evaluation

We have covered the navigation for both the robot, the user and the virtual agents and discussed some path-finding algorithms for the robot specifically. In the process, we made some interesting findings.

Navigation for Humans

Cheng’s paper[2] showed us that our current knowledge in MR allows us to perfectly navigate a person in a mixed environment. Navigation in our application could be used for assisting the user in finding its robot if he were to lose track of it. Note that this is only a suggestion and not a fixed objective for the second phase of the thesis.

Navigation for Virtual Agents

We found an efficient mapping and tracking algorithm [15] that would allow us to have a better grip of our surroundings. This would make for a better positioning of the mixed environment’s virtual objects and their navigation. In turn improving the application’s performance and reliability.

Path Finding

Abiyev's paper[1] discussed an extreme case of path-finding with two remote overhead cameras applied on fast-moving robots playing a competitive game together. Path finding on a slow Lego robot should be easy in comparison. Another worry here would be to implement efficient methods for the user to express where the robot should move to. In other words: providing user input.

2.4 User Input

The ability for the user to interact with our system is crucial. It has to be able to express such actions as making the robot move around to a certain location and making it interact with the environment including the virtual agents. Some advanced methods for user input need to be discussed to realise such expressiveness for the user. We will be relying on Kyto's paper[11] for an overview of modern techniques of pinpointing which is the "use of multimodal pointing techniques for wearable (AR) interfaces". These would allow users to select virtual or real objects in their environment. The paper focuses on eye gaze and head movement as primary methods for user input and also provides discussions to combine these. This revealed some performance improvements.

2.4.1 Eye Gaze

Aside from Kyto's paper[11] for a broad overview of pinpointing techniques, we will discuss eye gaze more specifically by relying on Lupu's survey[12] that focuses on eye tracking algorithms.
The functioning of eye gaze works by deducing the eye position and calculating the area they are pointing at and then implicitly or explicitly specifying whether you want to select that area. Explicit selection can be done with -for instance- a handheld device. Though eye gaze can be a good technique for the user to express himself within the application, it "suffers from low accuracy due to calibration errors and drifts of wearable eye-tracking sensors"[11]. It also has the problem of unwanted selection since a lot of the user's eye movement happens unconsciously.

2.4.2 Head Movement

A next pinpointing technique in Kyto's paper[11] is head movement. Though more precise than eye gaze for pinpointing, it has proven to be quite tiresome to use since the whole head has to be moved instead of only the eyes. To alleviate the user's efforts, we could use it as a supporting technology for eye gaze.This would be a way to cope with the low accuracy of eye gaze improving the user input's preciseness.

2.4.3 Evaluation

After reading Kyto's paper[11], it became clear that both eye gaze and head movement have their advantages and disadvantages. Eye gaze for instance is —when it comes to input speed— overall faster than head movement but it lacks accuracy because of technological limitations and the very fast nature of eye movement.
Head movement is more accurate but requires more effort of the user to handle since he has to move the whole head and neck area instead of just the eyes. Head movement proved therefore to be a tiresome input method. Moreover —when it comes to user input speed— it is slower than eye gaze.
By combining these two techniques, we could combine the best of two worlds: the accuracy of head movement with the speed and natural feel of eye gaze. This while limiting user fatigue by too much head movement.

Robot Control

The user can use both eye gaze and head movement techniques to control the robot. This would allow him to select the location in the MR environment of where the robot should move to. Any path optimisation would be handled by the path-finding algorithms that we already covered in the previous section.

Target Selection

Under "target", we mean the virtual agents that track the robot. These have to be shot down using the robot. Though for a video game it would be quite precise to use a handheld device for manual input, we are obliged here to use the full functionalities of the Microsoft HoloLens. Since selecting a hovering virtual agent in an MR environment is a work of precision, head movement supported with eye gaze looks like the best option.

2.5 Occlusion

Occlusion in Mixed Reality means that virtual objects that are "further" away from the user than physical objects can hide behind these "real" objects. Occlusion allows the user to have an increased perception of virtual objects by giving him the impression that virtual objects are positioned behind real objects. This would create the illusion that they are part of the environment[4]. A good paper that discusses the occlusion problem would be the paper[17] of researcher David Walton of the University College London[11].

2.5.1 Previous Techniques

Walton's work[17] starts by offering on overview of known methods for occlusion. This is a good starting point to explain what the presented new method does better. The problem with all existing methods is that they typically require some proper prior knowledge of the environment and more specifically on the physical objects on which occlusion must be applied. This must be either inserted by hand or "reconstructed using a dense mapping algorithm". Since our application should be applicable on any environment, we must find a method that does not need any prior information about it.

2.5.2 Cost Volume Filtering Occlusion

The good news is that Walton's paper[17] presents a new method for occlusion that does not need prior environment knowledge. It does —however— need a RGBD camera. Based on the colour levels it perceives and by depth estimations of the environment, improved occlusion is achieved.
It works by using "incomplete data to accurately simulate the occlusion of virtual content by real content in video see-through augmented reality scenes". The algorithm is very efficient, so much so that it is "capable of detecting partial occlusion at a pixel, allowing it to avoid aliasing along occlusion edges".

2.5.3 Evaluation

Immersion is important in Mixed Reality: good applications in that category should make users feel they are in a totally new world[13]. Part of this realization is to make occlusion happen, it would make virtual objects able to —al be it partially— position themselves behind real objects. Virtual

[11] `https://en.wikipedia.org/wiki/University_College_London`

Figure 2.7: The top two pictures present less than optimal results. The lower result was generated using the Cost Volume Filtering Occlusion method. The pictures were adopted from Walton's paper[17].

agents hovering next to the physical robot will sooner or later be "behind" it. The application must then make sure that occlusion happens.

2.6 Mapping of Virtual Agents

We will discuss the appropriate spatial mapping and tracking algorithms to overlay virtual objects on the physical environment. This all the while not having any knowledge about the terrain that gets mapped. This essentially means that an application having this feature can create a new mixed environment out of any environment. This is quite a novelty and is attributed to the emergence of better cameras and other sensors. It has already been discussed —be it briefly— in the navigation section of the paper but will now be treated in more detail. For further insights, we will be referencing Comport's paper[3] that presents a new "3D model-based tracking algorithm".

Moving Edges Algorithm

This algorithm is used for tracking moving physical objects. Globally speaking, "tracking is dependent on correspondences between local features in the

image and the object model. In an image stream these correspondences are given by the local tracking of features in the image". More trivially explained, it is called the moving edge algorithm because it finds the edges of an object with some simple tracking algorithms. This way, it can identify and track a physical object around.

Virtual Visual Servoing

Normal visual servoing[12] is a technique that consists of steering a robot based on the feedback it gets from a visual sensor monitoring all of its movements. This way, it can improve its current task. Virtual visual servoing consists of correcting virtual objects —not robots— based on their current positions in the real world as seen by our head-mounted device. This is important since the mapping of the virtual agents hovering around the robot must be as precise as possible for the sake of usability.

2.6.1 Experiments

The researchers demonstrate the features of their proposed tracking algorithm by way of a series of experiments[3]. They all use a commercial digital camera to perceive their environment. If using such device is possible with the given algorithm, this must mean it is quite efficient for not requiring expensive hardware to operate[10]. Given that the experiment succeeded with a commercial camera, using the Microsoft HoloLens should therefore be no problem. The distances of the real-life objects are calculated relative to the camera. More specifically, it gets calculated by way of the moving edges algorithm discussed above.

Tracking in Indoor Environments

The first experiment involves tracking in an indoor environment. With tracking, we mean that the used visual sensor or camera follows elements in the physical environment. The image estimation here has to be of high precision if we want the tracking algorithm to succeed. The experiment successfully tracked four 3 dimensional circles around.

Tracking in Outdoor Environments

A little bit trickier than previous experiment since outdoor environments provide more natural factors (rain, etc.) hindering the mapping. The experiment had to track a cylinder and correctly place a virtual object relative to

[12] `https://en.wikipedia.org/wiki/Visual_servoing`

it. “Despite very noisy images, tracking was still achieved” making the experiment an overall success. This experiment could be applied to effectively make virtual agents hover around the physical robot.

2.6.2 Evaluation

This article was chosen to tackle the problem of mapping virtual objects on the physical environment. To do this, the virtual objects' positions towards the physical robot must continuously be corrected or checked. If not, the placements of virtual objects can be so bad that the immersive experience would be severely undermined. The principal technology used for this is virtual visual survoing with the moving edges algorithm. The article[3] confirms that we can correctly position virtual game objects by way of a moving camera perspective, in this case the Microsoft HoloLens.

2.7 System Design

There are different system architectures that we could consider for our prototype. Kipman's demo was for instance client-server based but it could be interesting to research for some alternatives. Especially since it could offer some advantages for future development. Hence, we will explore our options.

2.7.1 Client-Server Based Application

The most trivial choice would be a client-server architecture[13]. It organises a system in such a way that part of its components provide computations for other components that request them. The former are called servers and the latter clients.
This architecture can easily be applied to our prototype MR game. Yet, we will not be using a remote server[6] for this but instead use the Microsoft HoloLens to host the application's server. This allows our prototype to work more independently and it eases testing. We also have a Mindstorm[14] robot that can also do some computation all be it to a lesser extent.

[13] https://www.webopedia.com/TERM/C/client_server_architecture.html
[14] https://www.lego.com/en-us/mindstorms

2.7.2 Peer-To-Peer Based Application

The problem of client-server based applications is that having some system components do all the computations can create quite a performance bottleneck. Especially by adding too many clients that could overwork the servers. Luckily, we can divert part of the workload to other components like our Mindstorm robot if need be. It is also important to consider that the application may have more components than originally envisioned in future development.
All these problems —namely scalability and the server-client dependency— could be alleviated by using a peer-to-peer architecture[15]. In it, all system components would be of equal importance and all provide and request computation. Having no real dependency between the nodes could prevent a lot of system crashes since no component would be critical for the system to work.

Shared Environment

A way in which we could expand the system —both in features and components— would be to implement a shared Mixed Reality environment that would allow multiple users to interact with each other. Note that we are considering this feature. Like mentioned above, for the sake of scalability and crash-prevention, using a peer-to-peer architecture seems like a good option. It would potentially allow an unlimited number of participants to the system.

Solaris System

Keller's paper[9] presents the Solaris system[16] that uses a peer-to-peer architecture to create a shared virtual environment. In it, every system component would provide and request computation which solves the performance bottleneck of having centralised servers do all the work by spreading it over all system participants. Solaris is also highly-dynamic adapting very quickly on system occurrences like ever-changing connections between nodes or users dropping in and out as they like.

[15] https://www.techopedia.com/definition/454/peer-to-peer-architecture-p2p-architecture

[16] https://en.wikipedia.org/wiki/Solipsis

User Interaction

A neat feature of the Solaris protocol[9] would be that it allows nodes to communicate their presence to neighbouring nodes as far as their hardware allows it. This means mobile computers like smartphones are still able to contribute to the peer-to-peer network, even if their computation abilities are pretty limited. This only lowers their radius of communication with the adjacent nodes.
Having a radius on which other closely positioned nodes can be in, it calculates which nodes are close enough to interact with. When two nodes are both within communication range, the system makes sure the nodes can interact with each other. It should be noted that if a node's computation capacity is very low, it might not be able to communicate with all nodes that are close enough to it.

2.7.3 Evaluation

We considered both the client-server and peer-to-peer architecture for our application. Having an alternate system architecture or even designing our framework as to be able to use both, could significantly extend its usability. Though the Solaris protocol is mostly applied to Virtual Reality, we could extend its usage to a Mixed environment. In it, the avatars' position would be the physical position of the users with each carrying a head-mounted device. They would all perform and request computation to make sure they can interact with one another and with the virtual elements in the environment. Again, this is only an exploration of what our application could offer.

2.8 Requirements

Using the gathered concepts and techniques from previous research work, we would like to formulate and develop a new application. It will be used as a framework for the navigation of game objects within a Mixed Reality environment.

2.8.1 Tools

To allow for Mixed Reality, we need a device capable of enriching the user's visual perception of the environment with additional computer-generated information. In this regard we have the Microsoft HoloLens[17] at our disposal.

[17] `https://en.wikipedia.org/wiki/Microsoft_HoloLens`

For development and debugging, we're using our own laptop.
We require a remotely controllable robot as an anchor point to place virtual objects in the environment. It must also have mobile capabilities since our navigation framework must also be able to correctly map game objects even when the environment changes dynamically. For this purpose, we have the Lego Mindstorms[18] robot.
To allow communication between these 3 devices
As for the software, we require different IDE's. We need the Unity editor to implement the server, Eclipse for developing the Java client and Microsoft Visual Studio for the deployment of the project on the HoloLens. We also discovered a number of toolkits to help us in the development process such as Microsoft's HoloToolkit[19] and the Vuforia[20] toolkit.

2.8.2 Functional Requirements

Based on these tools, we are able to develop a Mixed Reality application. We still need a list of basic functionalities that the software program should abide to.

Robot Navigation

The previously mentioned robot should be programmed to move in different directions based on the user's indications. This is necessary as to show that the framework can track a moving real-life object around and correctly place game objects on top of it.

User Input

The user must be able to direct the target robot in a certain direction. This can be realised in different ways. Voice commands —for instance— would require identifying certain keywords spoken by the user that would trigger the server into sending instructions in which way the robot should move. More complicated input methods would be specifying waypoints using eye gaze that the robot should follow using a path-finding algorithm. Depending on how the project development goes, different input methods will be developed.

[18] `https://en.wikipedia.org/wiki/Lego_Mindstorms`
[19] `https://github.com/Microsoft/MixedRealityToolkit-Unity`
[20] `https://developer.vuforia.com/`

Object Tracking & Mapping

A target object has to be overlayed with virtual objects. For this to be possible, it must be formally identified in the environment and tracked efficiently. Game objects must still hover on top of the robot by following it in its new course.

Spatial Awareness

This feature along with finding a path for the robot to follow and mapping virtual objects requires some advanced spatial awareness. For this to be possible it would require extensive geometry of the environment to allow for virtual objects to interact seemingly with the real-life world.

3

Navigation Framework

In this chapter we give our theoretical solution for the problem domains discussed in the introduction: a framework for the navigation of game objects inside a dynamic Mixed Reality environment. We start by giving an overview of its expected features followed by a discussion of system design involving the chosen framework architecture compared to possible alternatives and their feasibility. To assess the success of our framework, we will use it to develop a Mixed Reality game to show off its full capabilities. The low-level implementation details of both the framework and the subsequent game will be given in the following chapter.

3.1 Overview

In the related work section we specified our framework's functional requirements. As the development process went along, it became clear that some requirements had low feasibility and had to be modified to still achieve a fully-functioning framework. We mention where the framework deviates in this from our initial expectations.

3.1.1 Spatial Awareness

Using a device that allows Mixed Reality, the framework accesses the data collected by its sensors of the real world surroundings. This happens by way of the UWP application interface when using the HoloLens. The framework uses the collected information to generate a set of meshes representing the environment's geometry. This allows for correct navigation of virtual objects and interaction with real world components.

3.1.2 Object Tracking

A target object in the environment must be identified by the framework and tracked while moving. The user —when building derived applications— must provide a three-dimensional scan of the target. Scanning the environment for the object and tracking it happens with some advanced Object tracking happens by way of deep learning, a technique of artificial intelligence that builds up artificial neural networks. These networks are layers that progressively extract higher level features from raw camera input. Every higher layer in the network tries to identify more advanced diagnostics of the given data compared to the layer directly under it. To identify a target in the environment given its scan, the network would first try to identify the edges of the object and then try to find more advanced characteristics such as its shape, colour, etc. This process makes using Augmented Reality easy and fast.

3.1.3 Navigation of Virtual Objects

Identifying and tracking the target object is the first step to overlay virtual content on it. The framework is able to position virtual objects on top of the target even when it is dynamically moving. The user is able to configure the number of similar looking targets to track in the environment. This possibility can be the basis of future work for a multiplayer extension of the MR environment we would like to built.

3.1.4 User Input

In the thesis' introduction, we teased the idea of using eye gaze and selection for the user to specify waypoints in the environment for a device with displacement abilities to follow. However —due to technical difficulties with the immersion device— we had to use other input methods. This is why we opted for voice commands with certain keywords triggering the server in

sending certain instructions to the connected clients. This requires the host device to be equipped with a microphone to recognize voice commands.

3.1.5 Client-Server Architecture

The server component can be hosted on different devices and —in the context of Unity— can be activated by adding it to the current game scene. By way of a computer network shared by the server, other devices are able to send connection requests to it and receive instructions. There is a clear master-slave relationship relation between the two with the server communicating all the user input to all connected network devices.

3.2 Framework Package

The framework was built using Unity and represents a bundle of assets to help in the development of derived applications. We give an overview of all mayor components of the package and which of the functionality of previous section they implement. This will serve as a starting point to discuss image tracking and voice recognition, two important technologies on which the framework hinges.

3.2.1 Scenes

A scene in Unity is a piece of your game containing a variety of objects to make it display certain behaviour. These scenes can be opened and executed individually and can be extended by users to suit their derived software projects. For our framework, we had some deployment issues on our immersive device and had to make two scenes. The first scene has the object tracking functionality disabled as to allow for deployment on the HoloLens and the second scene has all components for the functionalities advertised in previous section.

3.2.2 Scripts

Our code snippets can be linked to game objects to make them display certain behaviour. Our package has two scripts and the first one 'ServerTcp' represents our framework's server. It can be activated by being added it to a scene object. It displays notifications by changing a 3D text mesh object and sends through all instructions to connected clients that it gets by voice commands.

The task of recognizing voice commands and thus of guaranteeing user input is managed by the second script: 'KeywordManager'. As the name suggests, it manages the recognition of certain keywords being spoken by the user. Keywords in voice recognition are short, distinct messages that trigger certain actions upon being recognized. The manager —when recognizing commands— prompts the server into sending instructions to its clients. The package users can alter these keywords as to make the manager react on other voice commands.

3.2.3 Holograms

To show that object tracking works, the framework must overlay virtual content over the tracked objects. The above mentioned folder contains a bee virtual object to map on the detected objects and a mesh of the manual for when we use the desktop and the HoloLens respectively. The virtual object that gets overlaid over the target can be altered.

3.2.4 Toolkits

These are bundles of software development tools that aided us greatly in developing this package. We used Microsoft's Mixed Reality toolkit and Vuforia in the process of developing our framework. The features of which will be discussed in the implementation section.

3.3 Spatial Awareness

The immersive device had to gain an understanding of the environment it was placed in to allow for the correct placement of virtual objects. This was achieved by way of Microsoft's Mixed Reality Toolkit. It provided us with a system for spatial awareness that could be activated after proper installation and subsequent configuration. The system works by collecting geometric information of its surroundings and representing it as a set of meshed. This would allow for the computer-generated components of our MR environment to interact seamlessly with the real-world environment. In combination with our object tracking software, it would make the whole experience even more compelling.

3.4 Object Tracking

Navigating a game object within a fastly changing environment requires some advanced image tracking algorithms to identify the target in the environment and keep a sight on it. In the process of developing our framework, we made use of the Vuforia[1] engine providing extensive object recognition and tracking capabilities. Being the intellectual property of a private company, the engine's full specifications remain a secret to the public. In spite of this we can discuss some of the used concepts in artificial intelligence[2] for realising object recognition.

3.4.1 Deep Learning

The engine makes use of artificial intelligence and in particular deep learning as to allow object recognition and tracking in a dynamic environment. It works by creating artificial neural networks consisting of multiple layers to progressively extract higher level information from raw visual input. Every higher layer tries to identify more advanced characteristics of the footage compared to the layer directly under it. The neural network is provided with a scan of the real-life object and would first try to identify the edges of this target. It would then go on to match more advanced attributes of the real-life object until it finds a suitable match in the environment. This process makes object tracking easy and allows to map virtual information on the identified target.

3.5 Keyword Recognition

To provide user input on the HoloLens, we make use of voice commands. By recognizing spoken keywords, the application reacts by invoking some methods for it. Unity provides us with a keyword manager to which we can set words that trigger the sending of an instruction to the client. The user is able to make the robot go forward, backward, left, right and stop as well as choosing to which clients instructions would be sent. This means the same instructions can be sent to all clients or just to one client also allowing the user to switch between them. He is also able to change these keywords without altering the code by using the Unity editor. It should be noted that the keywords should be by preference short and understandable. The recognizer is not able to understand a sequence of keywords so inserting

[1] https://developer.vuforia.com/
[2] https://en.wikipedia.org/wiki/Artificial_Intelligence_(series)

sentences is out of the question. If we were to recognize full sentences, Unity's dictation recognizer would be much more suitable.

3.6 Communication 

For our framework, the server can communicate to its connected client components by way of a private network. All devices connected to this network using the Internet protocol suite[3] have their own unique IP address. This allows them to be identified on the network and for other devices to send data packets to them. In our framework, the server manages multiple clients. It first listens on one of the ports of the host device for incoming client requests and —after accepting the connection— the server creates a thread (or task depending on the host) to handle its communication with that client specifically. For each connected client, it keeps a reference to it in a list. This list of references allows the user to select to which client data packets should be sent. We can choose to send the same instructions to all clients or just one selected client. Both the server and the client application are using stream sockets[4] for network communication. This allows data to be transmitted as a sequence of bytes. Before this can happen though, all instructions are encoded using the UTF8 character encoding scheme and —after being through the network to the client's endpoint— are decoded using the same scheme.

3.7 System Design

Based on the functional requirements specified in the literature study, we define our architecture best-suited for the prototype application we would like to develop. Though the client-server model has suitable characteristics in that regard, we would like to explore the feasibility of using the peer-to-peer model. We will therefore discuss both and see how achievable they each are given the resources at our disposal.

3.7.1 Client-Server Model

With this model there exists a clear role between the providers of a service —called the servers— and the requesters of a service —called the clients— with both communicating using an established computer network and hosted

[3] https://en.wikipedia.org/wiki/Internet_protocol_suite
[4] https://en.wikipedia.org/wiki/Network_socket#Stream_socket

on separate devices. This makes for a centralized application with the server providing the majority of computation.

Advantages

The client-server model boosts characteristics that make it suitable for our prototype Mixed Reality application. The framework provides a server component to be hosted on a device with significant computing power and we also had to implement a stand-alone client program that could connect to this server. This clear separation of component roles maps perfectly on the client-server model.
The centralised nature of the architecture makes for a distinct communication flow between servers and clients. This simplifies our prototype application greatly compared to if were to use the peer-to-peer model where servers send packages to each other.
The HoloLens immersive device would host the server and is capable of managing multiple clients. Given that we only have two EV3 bricks at our disposal, it shouldn't be overworked by too many service requesters. This makes the model suitable for our prototype application. If we were to have more resources for our prototype, up-scaling the application would require adding an appropriate amount of servers for the number of connected clients to efficiently share the workload.

Disadvantages

Having clear system roles makes for a master-slave relationship between the servers and clients. The model has system critical components that would slow down or even stop the application if they were to be removed. In terms of reliability, the peer-to-peer model would be a better choice since every system component is equally privileged and the system would dynamically adapt to the removal or addition of peers to the system. Because we're only making a small prototype for our framework with only one server component, this disadvantage is negligible.

3.7.2 Peer-to-Peer Model

This model makes no clear distinction between providers and requesters of services. 'Peers' make up the application and are equally privileged with each providing and requesting services to the system participants. They are said to form a peer-to-peer network of nodes. The communication flow is no longer limited from servers to clients but can be from any two peers in the system.

Existing Technologies

As was mentioned in the literature study, there are already Augmented Reality systems in existence realised with a peer-to-peer model. This shows that the possibility is there to adapt the model on our framework. The Solaris system[9] in that regard allows for a shared Mixed Reality environment where multiple users are able to interact with one another. The system is highly-dynamic adapting very quickly on system components dropping in and out of the established network as they like.

Requirements

Compared to a client-server architecture, the server components in the network are now peers and must be able to send requests to other participants if the need would arise. This would be the case when the workload in the application would be unevenly spread and a participant requests assistance from other nodes to manage multiple clients efficiently. The HoloLens device hosting the server is perfectly capable of sending and listening to service requests having multiple ports to do so.

Advantages

An architecture that does not have clear roles for its participants has some interesting attributes for the prototype we would like to develop. Using the peer-to-peer model makes for a more robust system since we don't have any critical components that would stop or slow down the execution of the application. If a peer would break down, the system would adapt dynamically to this and divert possible service requests to other peers.
Dividing the workload between its service providers would be much more flexible since all nodes are able to communicate with one another. If a network node would be overloaded by too much clients requests, it can still outsource part of it to neighbouring nodes.

Disadvantages

This model provides us with great flexibility for our application with no critical components that could potentially stop its operation. It does —however— come with increased implementation complexity since all nodes in the network must provide and request services between each other. Having each network component send requests puts a significant strain on the network. It would make for a much more complex communication flow.

3.7.3 Prototype

We eventually settled for the client-server model for our application. Though the peer-to-peer model satisfies our requirements, the problem is that our prototype is too small to reap the benefits of the model. We only have one HoloLens at our disposal to host our network node with no neighbours to receive and send service requests. This only adds complexity to the server code for features we wouldn't even be able to test out with our limited hardware and the scale of our prototype. The client-server model is relatively simple to implement —with the servers only having to communicate with the clients— making it appropriate for the prototype application we would like to develop. We conclude by saying that the peer-to-peer model is feasible to implement but should be used on bigger applications for its benefits of flexibility to outweigh the added complexity. Expanding into a peer-to-peer system can be a good idea for future work.

3.8 Mixed Reality Robotic Game

To show the capabilities of our navigation framework, we developed a Mixed Reality application based on it. Our framework was integrated as a package offering different resources for spatial mapping and object tracking with a server component also being provided.

3.8.1 Hardware

Our application has two distinct programs coded in different languages, hosted on different devices and with one using the framework as a basis. The program using the navigation framework can be hosted on a desktop computer or the HoloLens[5], an immersive device developed by Microsoft[6]. The other program represents the client in our architecture and had to be run on a Lego Mindstorms[7] robot capable of receiving data packets through an internet network. It is equipped with tracks to allow for displacement. The communication between these devices was guaranteed by of a private network setup with a router.

[5] https://en.wikipedia.org/wiki/Microsoft_HoloLens
[6] https://en.wikipedia.org/wiki/Microsoft
[7] https://www.lego.com/en-us/mindstorms/build-a-robot

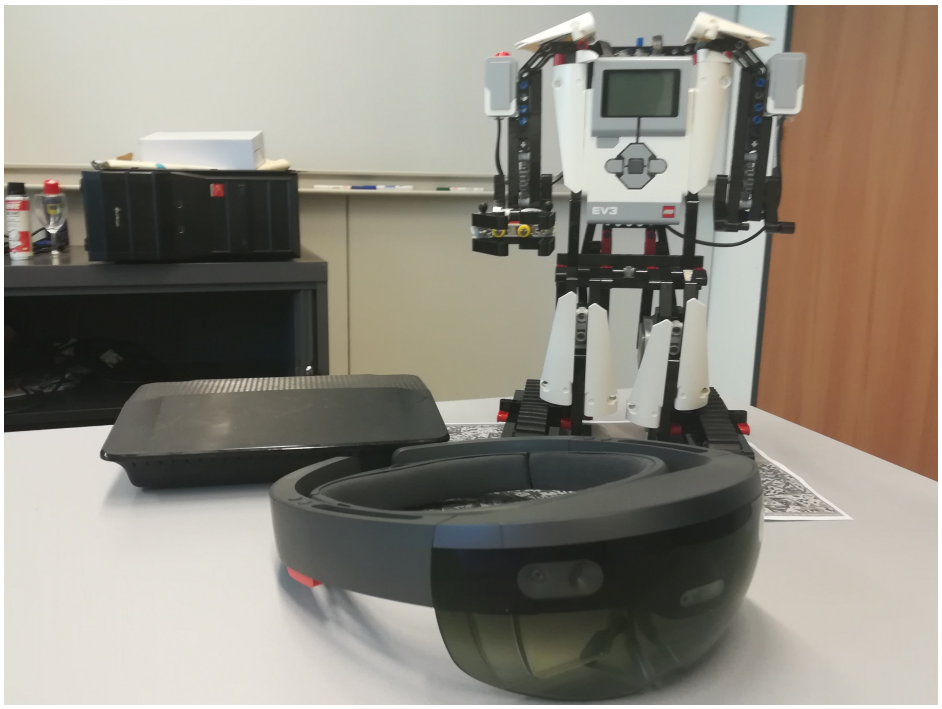

Figure 3.1: Using these three devices, we were able to test our framework and design a prototype application. From left to right: a D-Link router to setup a private network for communication, the Microsoft HoloLens to host the server and the Lego Mindstorms robot to host the client.

3.8.2 Framework Integration

The framework was built using the Unity[8] game engine and is a package with a set resources that includes a server component, spacial mapping component, voice command manager and object tracking component. Developing the application also happened with the help of Unity[3]. All of the previously mentioned components had to be added to the application's game scene in order to be activated.

3.8.3 Server

As mentioned previously the framework's server component is a Unity[3] scene object with a server script attached to it. The established server is capable of listening to new client requests and can handle multiple clients at once.

[8] https://en.wikipedia.org/wiki/Unity_(game_engine)

Within our prototype application, the whole Unity program is hosted on the HoloLens.

3.8.4 Client

The client is hosted on the Mindstorms robot with the EV3 brick —a small computer— functioning as its brain. Having less computation abilities compared to the HoloLens, it only takes on a client role in the application. It waits for the server to send data packets with instructions through the network and moves in the environment in accordance with what was sent. The brick is connected with tracks as to allow for basic movement.

3.8.5 User Interface

The user must be able to send instructions to the application's client robot indicating how it must move in the MR environment. Doing so happens by way of a very simple interface and voice commands.
Because of technical limitations, we were not able to use waypoints in our environment to steer the robot towards them with a path-finding algorithm. We settled for voice commands with a voice manager that reacts on certain keywords being spoken out loud. These keywords trigger the server to send specific instructions to the clients. The environment has a three-dimensional text that changes for certain events such as the server start-up or on certain keywords being detected by the voice manager.

4 Implementation

This chapter discusses the implementation of our navigation framework and of the Mixed Reality application using it. We start by giving an overview of all hardware used for this project. We then go through the implementation of the framework's components. The server is part of it but —due to its importance for the whole appliction— it will be discussed in its own section. We will end on the client which is the only part of the application not developed using the framework.

4.1 Hardware

Our system uses an internet network for the client(s) and server to communicate with each other. Establishing such network required using a D-Link[1] router. The client program runs inside a Lego Mindstorms EV3 brick[2] equipped with tracks as to enable robot movement. As for the server program, it runs inside a desktop computer when using the Unity[3] editor when debugging and the Microsoft HoloLens when deploying.

[1] `https://en.wikipedia.org/wiki/D-Link`
[2] `https://en.wikipedia.org/wiki/Lego_Mindstorms_EV3`

4.2 Network

For the communication between the client and server, it was necessary to setup our own private internet network using a router device. It used a modem for cable internet connection and required further configuration using the router's installation assistant[3] such as setting the password for the network and the admin account. Our network uses the internet protocol suite[4] and associates a network (IP) address to all connected machines as to allow identification. This would also allow all connected devices to forward data packets to each other.

4.3 Navigation Framework

Allowing the proper navigation of game objects within a Mixed Reality environment was the main focus of the thesis. Our built framework can be seen as a package with a collection of resources that programmers can use to build derived applications. In the context of Unity, these resources take the form of game objects with scripts attached to them. They need to be added to the new application's game scene as to activate their functionalities.

4.3.1 Toolkits

During the development of our framework, we made use of a couple of development toolkits. They provided software tools that aided us greatly in building our final prototype application. We used the Vuforia[6] toolkit for the detection of specific target objects in an environment and the tracking of these objects. This would also allow us to overlay them with virtual information. Another important toolkit used is the Mixed Reality Toolkit provided by Microsoft in an effort to boost the development of applications on their HoloLens product for Mixed Reality. It provided us with building blocks to activate spatial awareness and implement voice commands.

4.3.2 Server

This component is the central part for our prototype application. It receives indications by the voice manager of which keywords have been spoken out by the user and sends the appropriate data packets to the clients based on these spoken commands. The server is a script added to a game object in of the

[3] https://eu.dlink.com/fr/fr/support

[4] https://en.wikipedia.org/wiki/Internet_protocol_suite

framework's scenes. It has a three-dimensional text mesh that gets changed by the server based on some events: starting the server, voice commands to go forward or backward, etc. This way, both user and developer get notified of what's happening on the server side.

4.3.3 Voice Input Manager

To allow for user input with voice commands, the scene includes the Voice Input Manager object. It contains the Keyword Manager C# script provided by one of Microsoft's HoloLens sample projects[5]. It is configured as to send instructions to the subscribed clients on certain keywords.

Keyword Manager

We did not implement our own voice recognizer engine but used one provided by Unity. It listens to speech input and attempts to match spoken phrases to a list of registered keywords. Our keyword manager class code is implemented in such a way to allow the user to set the recognizable keywords himself in the Unity settings. This is much simpler than to hardcode it in the code self. He can also set which methods from which game objects should be called when recognizing a keyword.
We implemented a public structure in the code where every element contains a keyword and a function to be called. At initialisation, all structure elements are added to a dictionary using as keys the keywords of each elements. The associated value are the methods that should We also instantiate a keyword recognizer and give as arguments the keywords it should be listening to. We bind to it a function that —for when a word is spoken out— tries to match it with one of our keywords. If a match does occur, it invokes the method that was associated with the matching keyword. The code for the manager can be found in the Appendix.

4.3.4 Vuforia Augmented Reality Camera

Every scene with object tracking enabled contains an object with Vuforia's camera for Augmented Reality. We also use it to configure our tracking engine and to import 3D scans of small objects —such as our robot— that has to be found in the environment.

[5] `https://github.com/kaorun55/HoloLens-Samples`

Target

Another important object in any scene related to the Augmented Reality camera is the target object. It contains the three-dimensional scan of a real-life object that we would like to track in the environment as well as the virtual object that should be mapped to it. The scan was added as a package from Vuforia's developer portal and allows the real-life object to be found in the environment.

4.3.5 Holograms

The Vuforia engine needed some virtual resources to map on the target that had to be found in the environment. This is why we offer a small bee virtual object as a resource. For user friendliness, we provide two user manual meshes that also appear in the package's scenes. One is for when the desktop computer is being used while the other is for the HoloLens. They explain how to use the default controls on their respective host devices. Using other virtual objects is possible by designing a mesh yourself or browsing Unity's asset store.

4.3.6 Scenes

In the Unity game engine it is possible to split the development of a game in multiple pieces called scenes. They can be seen as game 'levels' with every scene containing a variety of objects linked with script that give it specific functionalities peculiar to given scene. Because of deployment issues on the HoloLens device, we had to develop two different scenes with one having Vuforia's object tracking features removed. This way it could still be deployed on the HoloLens.

Sample Scene

The package contains a 'scene' folder in which you can find all scenes for the framework. A Unity scene contains the environments for your game with decorations, obstacles and other components included. It allows you to design you application in pieces contained in these scenes. We mentioned in previous chapter that the Vuforia engine could not deploy on the HoloLens properly, this is why we made a 'demo scene' where all object tracking is disabled and a 'sample scene' with all framework functionalities enabled. We will discuss the latter here with all of its components.

Demo Scene

To realise our demonstration video with the immersive device, we had to make a scene that deployed correctly on the HoloLens. The current scene has no object tracking or game object navigation but still allows for user input, spatial awareness from the Holotoolkit and has a server component to manage clients.

4.4 Server

This system component manages multiple clients and its code can be hosted on both the Unity editor for debugging and the HoloLens for production purposes. Its role would be to listen for incoming clients trying to connect and then forward navigation instructions to the appropriate connected client. These instructions would be given by voice command when using the HoloLens and by way of the desktop keyboard when using the Unity editor.

4.4.1 Incompatibilities

Programming a server that would work on both devices proved difficult because of library incompatibilities. All Microsoft products since 2016 compile their code using the Universal Windows Platform[6] developed as part of the Windows 10 operating system. Its main goal was to allow the development of universal apps that would run on all Microsoft products such as the Xbox One and HoloLens without the need to be re-written for each product. They use a version of the C# programming language that is different from the one used by the Unity game engine. This means that we have to write two different code sections for our server depending on whether we're using the Unity editor or the Microsoft HoloLens. The implementation details of which are discussed further down.

4.4.2 Server Code

For our application, we wrote code snippets —referred to as scripts— that can be attached to game objects in scenes to make them have certain behaviour. The 'ServerTcp.cs' file contains the code for our application server which had to be compatible on both the Unity editor and HoloLens. It uses a private network based on the Transmission Control Protocol[7] to reliably send

[6] https://en.wikipedia.org/wiki/Universal_Windows_Platform#Compatibility
[7] https://en.wikipedia.org/wiki/Transmission_Control_Protocol

data packets to its connected client applications running on different host machines.

Preprocessor Directives

Because of Microsoft's Universal Windows Platform, we had to write two mutually incompatible code sections in the same server file. Depending on the host device, one section would be used and the other one ignored. An issue is that the compiler is unable to tell both codes apart from each other and —to resolve this— we were required to use preprocessor directives. They are a language construct that tell the compiler how to preprocess the source code before actual compilation starts. In this case, we are telling it which code to ignore when using the computer and the HoloLens respectively. This makes the separation of codes explicit.

```
20  #if UNITY_EDITOR
21  using UnityEditor;
22  using System.Net.Sockets;
23  using System.IO;
24
25  //Make the server work
26  public class ServerTcp : MonoBehaviour
27  {
28
29      private String currentInstruction = "N";
30
31      public Text NotificationsText;
32      public String notification = "Standby..";
```

Figure 4.1: Line 20 says that if we were to use the Unity editor, we should use the code between the #if and #endif preprocessor directives. Though we could intermingle these directives to avoid code duplication for our server, it was much cleaner to do a complete separation of Unity editor and UWP code. This made our code snippet overall much more readable.

Multithreading

The server listens on one of its port to client requests to connect. As to allow multiple clients to use the server, the use of threads was necessary. At the time of writing however, the Universal Windows Platform used by the HoloLens does not support threads instead only allowing us to use tasks. In the two code snippets separated by preprocessor directives, one is using threads and the other tasks. To clarify, threads can be seen as workers to be

used to fulfil some amount of work while tasks are just promises to complete a certain amount of work in the future eventually. The big difference when coding is that they have different programming interfaces.

4.4.3 Desktop Input

The server can be hosted on different devices with different methods for user input. If we were to use the Unity editor, we would make use of the host computer's keyboard. Using the arrows buttons, the user can specify in which direction the robot should move. If nothing gets pressed, the robot simply stops because only the stop instruction —character 'N'— would be sent to the client's endpoint in the network.

ReadKeyBoardInput

Some keyboard buttons trigger some actions within our application. We implemented a function that gets called continuously to see if certain buttons were pressed by the user. It is a collection of conditional statements with each getting executed if the required button was pressed.
Scripts in Unity are all derived from the MonoBehaviour base class. It contains a small set of default functions such as Update and Start with the first being called at every frame in the game and the second when enabling the script. To read the keyboard input, it is necessary to check at every frame whether some keys were pressed. Our ReadKeyBoardInput is placed within that Update function as to be called frequently enough to be responsive for the whole system.

```
/* For when the desktop acts as server, we must scan/read the keyboard.
 * Only relevant when using the UNITY editor, it gets ignored otherwise. */
private void readKeyboardInput()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {
        Debug.Log("Up arrow. Mindstorms will go forward. ");
        currentInstruction = "1";
    }
    else if (Input.GetKey(KeyCode.DownArrow))
    {
        Debug.Log("Up arrow. Mindstorms will go backward. ");
        currentInstruction = "2";
    }
    else if (Input.GetKey(KeyCode.LeftArrow))
    {
        Debug.Log("Up arrow. Mindstorms will go left. ");
        currentInstruction = "3";
    }
    else if (Input.GetKey(KeyCode.RightArrow))
    {
        Debug.Log("Up arrow. Mindstorms will go right. ");
        currentInstruction = "4";
```

Figure 4.2: This is the partial implementation of the function 'readKeyBoardInput'. It gets called for every frame and scans the keyboard on specific pressed buttons such as the arrow buttons in the 4 first clauses.

4.4.4 HoloLens Input

Being a pair of wearable computer glasses that enables Mixed Reality, the HoloLens does not have a tangible interface —such as a keyboard— to work with. The user can therefore use voice commands by enabling its microphone. It should be noted that waypoint selection and path-finding was initially opted as methods for moving the robot but that it could not be realised due to deployment issues.

Voice Input Manager

The code for the voice manager is not part of the server code. Instead it is another Unity script linked to a game object altogether. It merely calls the server methods when it detects some keywords being spoken out. The HoloLens server has a set of public fuctions that form its programming interface with the voice manager.

4.4.5 Client-Server Communication

In our prototype, every connected client has an endpoint to which the server sends instructions to. The client sends a connection request to the server and —when approved— it continuously polls its input socket for new information from the server.

Connection Establishment

The server has a TCP listener that waits on a certain port for incoming client requests. When it receives a new client request, it starts a new thread (or task) that handles all communication to it. It also keeps a reference to the TCP client in a list for later use. This way, we can search up a client, identify it on the network using its address and send specific data to it. This is all done by using the client's streamsocket where bytes of data can be send through.

Client Selection

The user can decide to which connected client to send instructions to. Using the keyboard or voice commands, we can decide whether to send the same instructions to all connected clients or just to one client. For the latter option, the user can also select to which of the connected clients he wants to send instructions specifically.

4.5 Client

This system component was developed without making use of our navigation framework. It is a Java application comprising two classes (see Appendix) hosted on the Mindstorms robot that has access to our private network and connects to the server given its IP address. It then continuously polls its input socket stream for new instructions. The server keeps a list of all connected clients and creates a thread —or task when using the immersive device— for every one of them. Each thread handles the communication to exactly one client. This allows more than one client to receive instructions from the server.

4.5.1 Lego Mindstorms Robot

As moving ground unit, we used a programmable robot based on Lego building blocks with the Lego Mindstorms EV3 Intelligent Brick as its brain. It

allows to upload code and functions as the robot's controller and provides its power to run the motors and sensors. There was however a problem with its firmware. Lego only offered the Ev3 Programmer App[8] to be used on the Mindstorms robot to reprogram it. This only consisted of a primitive drag and drop programming interface and it was too limited to allow for some real coding.

leJOS Firmware

The programmable brick's firmware limitations prompted us to use leJOS[9] as a replacement. It includes a Java virtual machine allowing us to program our robot with the Java programming language. The installation involved uploading it on an SD card and inserting it in the EV3 brick. Starting the brick with the card would configure it to use the replacement firmware. Writing the Java code involved using the Java libraries for the leJOS firmware allowing us to have access to the brick's motors and sensors. By simply using the brick's IP address, we could execute our code and upload it to the brick.

Libraries

Aside from firmware, leJOS also offers a variety of Java libraries to support programmers in using the brick's hardware given of course that it uses its firmware. Importing these libraries granted us access to the brick's screen and connected motors. Each brick component —be it the screen or the connected motors— has their own set of interface methods that we can use in implementing class methods. We can go as far as to make extended trajectories for the robot to follow such as doing a full 360 degree circle or just implement a simple instruction to move forward for a second.

4.5.2 MotorOps Class

As to make the code cleaner, all functions involving operating the robot's hardware are contained in one class. In the client class, we instantiate it as an object and use its public methods to operate the brick's hardware. These operations include changing the screen text or moving the tracks in certain directions to go left, straight or turn around. The track directions can be used to implement certain tricks such as pivoting on its axis.

[8] `https://www.lego.com/en-us/Mindstorms/apps/ev3-programmer-app`
[9] `http://www.lejos.org/`

Movement Methods

These methods are used to move the robot based on the instructions received at the client's endpoint. They call the interface methods of the brick's hardware and follow each other in quick succession to move the robot in any way desired: left, right, forward or backward. It also has one 'stopAll' method that stop all motors and that gets called when the client has no instructions from the stream. As the name suggests, it stops all motors.

Other Methods

The brick has a small screen for menu navigation but can be used during our program execution to display text messages. We developed the method 'screenMessage' that takes a string to print on the screen which proved useful for debugging purposes. We implemented some simple tricks on the robot to test how synchronised both tracks work together. One method allows doing a 180 degree spin while the other does a full spin on its axis. It required some calibration of the time that the motors had to run in inverted directions.

Motor Coordination

A motor is a track or claw connected to one of the robot ports. Our application makes only use of the tracks, the arms are purely aesthetic. For every instruction, both or just one of the tracks gets commanded to move forward or backward. Going forward requires both tracks to go forward, the opposite can be said to go backward. Going left requires only the right track (motor C) to go forward. When moving to the left, it is only the left track that should move forward (motor D). For a spin, both tracks move in opposite directions.

4.5.3 Client Class

This class manages communication with the server on the private network and interpreting the messages it gets from the network. At instantiation, it tries to connect to one of the server's ports on a specified network IP address and instantiates a MotorOps object for all robot movement. When connection to the server succeeds, it listens to its network inputs for data packets. Based on the received packets, it summons certain public methods of the MotorOps objects to move the robot.

```
public static void continueRight(){
    motorB.forward();
    /** Stop motor C: it could still be active. */
    motorC.stop();
}

/** Stop all the motors: arms & legs. */
public static void stopAll(){
    motorA.stop();
    motorB.stop();
    motorC.stop();
    motorD.stop();
}
```

Figure 4.3: The two following methods implement robot movement to the right and a general stopping method for all motors. 'ContinueRight' makes sure the right motor goes forward and the other one stops completely. To go to the left, the opposite is applied.

Stream Socket

As to allow the it to receive data over the established computer network, the client is making use of stream sockets. This type of socket provides a connection-oriented and unique flow of data without record boundaries. They are implemented on top of TCP used by the router so that applications can run across any networks using the internet protocol suite[10]. The client itself only sends a client connection request to the server on the port it is listening to. The client afterwards will not send anything to the server with its socket but merely polls its input stream continuously to see if there are new instructions from the server.

Stream Interpretation

Each character that gets passed through the network represents an instruction to the client. It gets interpreted as an instruction to go forward, backward, left, right or just stop moving. They each follow each other up quickly since they get passed as a stream of bytes through the network. This is why every called method for one instruction starts and ends quickly for the next instruction to be applied on the robot's motors. If on the server side no new movement gets specified by the user with voice commands, then a

[10] https://en.wikipedia.org/wiki/Internet_protocol_suite

```
69    public static void readBytes(){
70
71        try {
72            inputServer = socket.getInputStream();
73            in = new DataInputStream(inputServer);
74            byte[] buffer = new byte[4];
75            int read = 0;
76
77            /** If something is received, read it and interpret it. */
78            while ((read = in.read(buffer, 0, buffer.length)) != -1) {
79                /** What message did we receive from the server? */
80                String message = new String(buffer, "UTF-8");
81                motors.screenMessage("Server says: " + message);
82                /** Interpret the number message. */
83                interpretMessage(message);
84            }
85
86        } catch (IOException ex) {
87            ex.printStackTrace();
88        }
89    }
```

Figure 4.4: This function gets called continuously to poll the client's input stream for messages of the server. If something was sent, it gets displayed on the brick's screen and gets interpreted to a relevant movement instruction for the robot's tracks.

stream of stop instructions gets passed through the network. This is to make sure the client will stop moving if no instructions were given since every passed instruction calls a method that activates certain motors to move in a direction.

5 Evaluation and Reflection

In this chapter, we evaluate the prototype application build using our navigation framework. We also reflect on the development process with the encountered difficulties and in which ways our final product diverts from the initial requirements and expectations. We will also discuss the ways in which we tested our prototype for its correct working.

5.1 Evaluation

To assess the success of our final prototype, we wanted to measure the user's general attitudes and emotions when interacting with its provided interfaces. For this, we made use of the User Experience Questionnaire[1] providing a fast and reliable way to measure the user experience of our interactive product. The server is the user's mayor point of interaction and it has two possible host devices with different methods for user input. This is why we applied the questionnaire on five users when interacting with the desktop computer and the HoloLens separately. This is to measure the success of both interfaces independently.

[1] `https://www.ueq-online.org/`

5.1.1 User Experience Questionnaire

The original German version of the UEQ was created in 2005. After several studies and iterations through the years, its final questionnaire consists of 26 questions that are rated on a scale of 1 to 7. Each question is represented by two terms with opposite meanings on the opposite sides of the scale. The final rating happens in accordance with 6 usability scales with each questionnaire item belonging to one of them.

Usability Scales

Each scale is linked to a particular aspect of the interactive experience and —when taken together— gives an idea of the relative success of the user application. The questionnaire's scales are the following:

- Attractiveness: the appearance of the interface: how pretty, friendly and enjoyable it looks for the user.
- Efficiency: the degree in which tasks can be performed quickly, efficiently and pragmatically. The user should perform the least amount of work and achieve the most in as little time possible.
- Perspicuity: the product is easy to understand and simple. The user should pick it up rather quickly and easily.
- Dependability: the whole user experience is secure and predictable. A secure interface means that interaction is safe and controllable for the user.
- Stimulation: the user's excitement and pleasure when using the application. The measure in which the prototype is fun to use.
- Novelty: the product is innovative and creative.

Analysis

The questionnaire was applied on 5 participants and we interpreted the means of the scales because the questionnaire does not produce a percentual rating of the user experience. By analysing the distribution of the results we came to some interesting findings. The data analysis Excel sheets provided on the questionnaire's website[2] contained a dataset to compare our user results with. It contains data from 18483 persons from 401 studies concerning different

[2] `https://www.ueq-online.org/`

products (business software, web pages, web shops, social networks) forming a benchmark to compare our results with.

5.1.2 Desktop Interface

When the server runs on the Unity editor hosted on a desktop computer, the keyboard is used as input method. Since a tangible keyboard is much more familiar to use than voice commands, it can be expected that the general user opinion would give a preference to the desktop interface. To promote ease of use, the Unity editor's scene view has a 3D manual to explain which buttons trigger which actions. The arrow buttons can be used to move the robot. The user can also choose whether to send instructions to all clients simultaneously (A) or one client specifically (Z). It is also allowed to switch between clients using button C on the keyboard.

Mean Results

The results of the questionnaire showed a high standard deviation indicating significant data dispersions and variations. This can be attributed to the low number of participants. Attractiveness, perspicuity and stimulation were the best scoring scales indicating that the interface is user-friendly, easy to pick up and exciting to use. The three-dimensional manual displayed on the Unity editor clarifies to the user which buttons to use and because of the presence of technology these days, it is normal that all participants can get along with the computer's keyboard. The control of the robot is responsive and it shows in the user's stimulation when using the application.

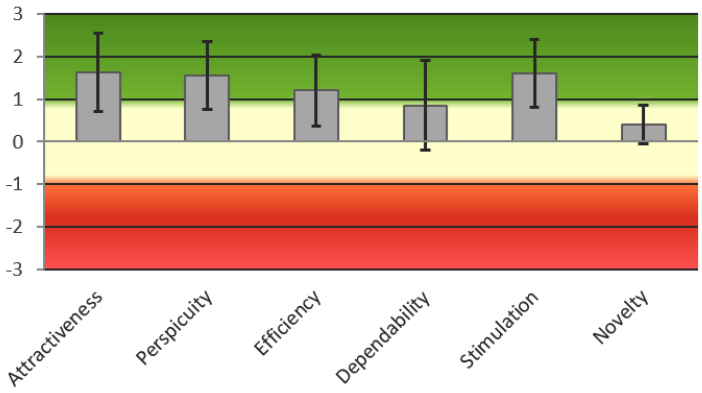

Figure 5.1: Even with great data dispersion, we can see that most means of the scales have a positive evaluation. Novelty scored the poorest.

Novelty on the other hand scored poorly. This can be attributed to the keyboard because is not a new or unconventional method for user input. For the opinions on the dependability of the application, they vary greatly from mediocre to good. An explanation for this is that some users found the controls for client selection not clear. Another explanation would be that they confused the meaning of the word 'secure' with data security instead of safety and controllability.

Benchmark

Comparing our survey with the benchmark dataset indicated that our interface performs above average compared to other software interfaces. Switching and selecting clients was at times a little confusing because the user does not get precise indications of which client in the environment is selected. Novelty was below average because we did not try something creative in terms of user input methods.

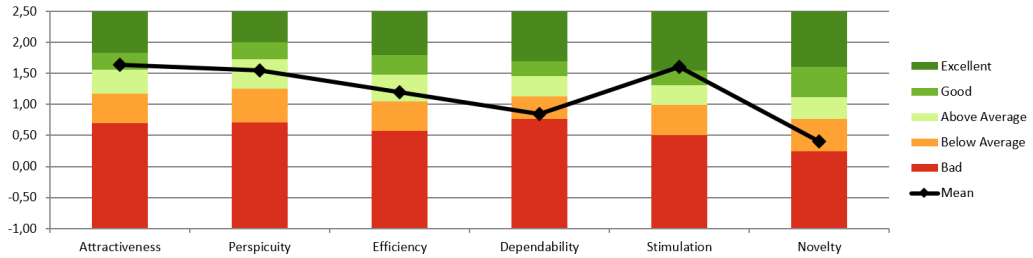

Figure 5.2: Our desktop interface scored below average in terms of dependability and novelty. Other aspects of the user experience scored very well showing that our interface is rated 'above average' for our benchmark.

Verdict

Based on the questionnaire's mean scores and the comparison with the benchmark dataset, we can assume our desktop interface is successful. Though the mean scores for dependability and novelty showed mediocre results —being ranked 'below average' in our benchmark— the users were satisfied in terms of attractiveness, perspicuity and efficiency.

5.1.3 HoloLens Interface

The immersive device has no tangible interface such as a keyboard or joystick. It is just a pair of smartglasses enabling us to develop Mixed Reality applications. This prompted us to use less conventional ways for user input such as voice commands. The user can shout short keywords to let the robot move forward, backward, left and right and to select the clients to which instructions should be sent. This can be all clients or just one allowing to go through all clients to select the right one by also using voice commands.

Mean Results

The results for each scale have lower standard variations compared to the desktop interface results. This can be attributed to the fact that the participants never used an immersive device before. Given answers align more with each other because they have no experience with the hardware to give a more profound opinion about the interactive experience that may vary individually. What also stands out is that our interface scores poorly in efficiency and dependability. Voice commands take some time to be understood by the HoloLens meaning there's a significant delay in sending the instructions to the client. Expressibility with voice commands is lower compared to a more intuitive keyboard requiring more effort from the user in completing certain tasks. The low scores for dependability can be attributed to the voice commands and significant delay between receiving and executing movement instructions. On occasions, voice commands are not well understood by the keyword manager making increasing the unpredictable nature of the experience.

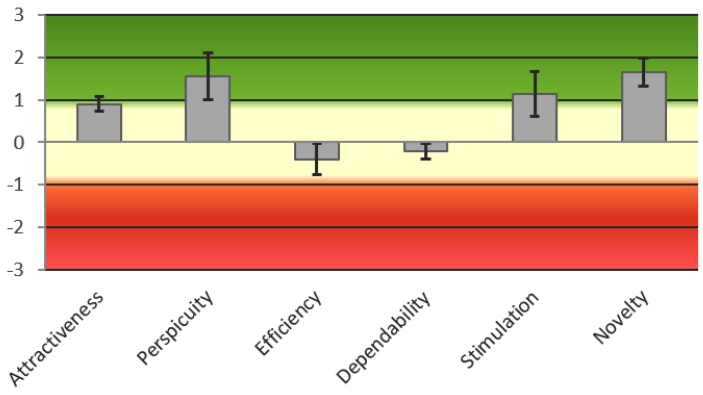

Figure 5.3: Low scores for efficiency and dependability are due to the delay of voice commands and the lack of expressibility with voice commands.

The interface scored well in terms of perspicuity, stimulation and novelty. Voice commands only required users to speak out specific keywords that were short and easy to remember resulting in an interface that was easy to learn and remember. The users thought the whole experience was enjoyable and innovative due to Mixed Reality technology being not very widespread as a technology. Nevertheless, voice commands proved a good choice for the perspicuity of the prototype.

Benchmark

The lower scores also reflect in the benchmark comparison. Efficiency and dependability both score much lower than the benchmark average with a mediocre rating for attractiveness. Due to the futuristic nature of the HoloLens immersive device and the ease of use of voice commands, the interface has good to excellent scores in terms of novelty and perspicuity.

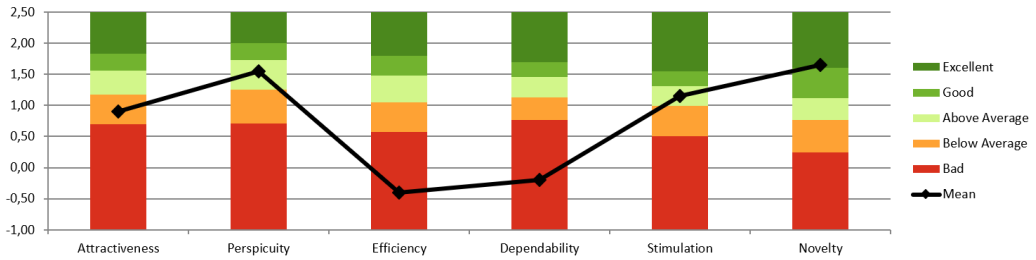

Figure 5.4: The results show that in terms of efficiency and dependability, our interface scores poorly. There is significant delay in processing voice commands on our immersive device and keywords are sometimes not properly understood by the voice manager.

Verdict

The user experience for the HoloLens was mixed. Its interface is innovative and easy to pick up yet voice commands are sometimes not well understood due to poor user voice expression. It also suffers from significant delay in processing voice commands. All these shortcomings can be put on the HoloLens' hardware and not on the interface design itself.

5.2 Reflection

Development of the client application happened smoothly with the installation of the robot's new firmware and the writing of the client code being achieved in the month of April. Developing the server code however proved more difficult. There were library incompatibilities between the two platforms on which the server had to be executed and we faced deployment issues with our immersive device. 

5.2.1 Library Incompatibilities

As was mentioned in the implementation section, the Unity editor and the Universal Windows Platform use different libraries for the C# programming language. We were obliged to write two different codes in the same server file separated by preprocessor directives. This complicated the development process for the server code. Debugging the server for the HoloLens was made more difficult compared to the Unity editor because of long deployment time. Executing the code on the editor happened instantly compared to a dozen of minutes for the HoloLens.

5.2.2 Deployment Issues

The Vuforia engine worked properly on the Unity editor using the device's front web camera to track the target objects. Deployment on the HoloLens immersive device was a different story. It worked well on the Unity editor but crashed when applied on the immersive device. requiring further configurations. This is why for our demonstration, we deactivated all object tracking features to allow for proper deployment.

5.3 Changed Requirements

For our literature study, we said we would like to implement path-finding in our application by allowing the user to select waypoints in the environment with eye gaze and selection. These functional requirements were unfortunately never met because of deployment issues. Given the faced issues, a realisable method to allow for user input to guide the robot around was to use voice commands. It allows us to move the robot in any wanted direction.

5.4 Testing

Our prototype application using our framework required some extensive testing and debugging in order to work properly. It required a variety of tools to be used that we will give an overview of. Most of our tests for the application focused on the connectivity of involved devices and the correct delivery of data packages between them.

5.4.1 Client Side

Testing the client focused on its proper connection with the server and on properly receiving data packets through our network. Its written code contains a stream socket used as an endpoint for receiving packets. Sending packages back to the server was unnecessary since the communication flow of our application goes from the server to its clients.

Unity editor for Debugging

Since the HoloLens was much slower when it came to deployment compared to the Unity editor, it was easier to use the latter of the two to send instructions through the network. By using the host device's keyboard, we could send out instructions to the client's endpoint socket.

Display Screen

To analyse which instructions would be properly received by our EV3 brick[3], we would use its display screen to print out all messages it would receive by the stream socket. We could then configure the robot to properly react on the given instructions on the network. The Unity editor's keyboard allowed us to send specific displacement instructions to the robot to calibrate it even further as to make it more responsive.

Eclipse Breakpoints

Eclipse[4] is a very popular integrated development environment for Java programming among others. It has a base workspace and an extensible plug-in system that would allow programming in other languages giving the right plug-ins. For the development of our client it provides us with breakpoints. This handy tool allows us to do an intentional stopping or pausing of a running program at a certain place for debugging purposes. This helps us to

[3] https://shop.lego.com/en-US/product/EV3-Intelligent-Brick-45500
[4] https://www.eclipse.org/

determine if the program executes in a correct fashion. If a bug or failure would occur at runtime we could pause the program just before its occurrence and analyse every passing instruction to see how we came to that bug.

5.4.2 Server Side

For the same reason as with the client side testing, we primarily used the Unity editor when debugging the server. This proved faster in deploying the server though it did not allow us to test all of its features. Since in the final application the server was supposed to be hosted on the immersive device, we eventually were forced to use it for further testing.

Unity Editor Testing

By running the Unity editor in play mode, the server would be executed in a matter of seconds. Evaluating the server happened in two steps. The first step involved using the Packet Sender[5] app for sending and receiving network packets using the Transfer Control Protocol among others. We first had to specify the IP address and port number of the connected device on the network to which a test packet should be send. If no error message would occur after sending the package, we could be sure that the device was open for connection. The second step involved pressing the arrow buttons on the desktop keyboard and by using the console, we could see which instructions were passed through our network.

HoloLens Testing

Each time we would like to test improvements on the UWP server we had make a new build, compile it on our Microsoft Visual Studio[6] IDE and then upload it on the HoloLens. This whole process would easily take a dozen minutes to complete.
Debugging proved difficult on the HoloLens. At first, using breakpoints on Microsoft Visual Studios did manage us to open the ports of the immersive device but sending out packets to the clients was not that clear using breakpoints. After a lot of trial and error though, we manage to send out data packets to the client.

[5] `https://packetsender.com/`
[6] `https://en.wikipedia.org/wiki/Microsoft_Visual_Studio`

Multiple Clients

To allow for more than one client to connect to the server and request its services, we had to use threads for the Unity server and tasks for the UWP server respectively. In both cases, the server would keep a list of references to the connected clients to allow for client-specific packets to be sent. Testing this out required using a second EV3 brick and seeing to which client specifically data was being sent.

6

Conclusion and Future Work

In this chapter, we conclude our thesis by giving a last overview of the work undertaken in developing our navigation framework. We also discuss possible improvements for our framework to be realised in future work.

6.1 Conclusion

We have worked in accordance with the Design Science Research Methodology for information systems[16]. In the introduction, we identified the problem domains for our research effort and defined our objectives for success. The literature study allowed us to gather concepts and techniques for Mixed Reality as a baseline to define our application's requirements. We ended our related work with the functional requirements for our framework: robot navigation, user input, object tracking, mapping virtual information on the environment and spatial awareness.

In the next chapter, we gave a high-level overview of how we realised each functional requirement from previous chapter. Following this was a discussion of the software package of our framework and how object tracking and communication was achieved. One of our problem domains from the introduction was to explore the possibility of using a peer-to-peer model for implementation instead of a client-server model. This is why we wrote a section about system design arguing which model would be best suited for our framework and subsequent prototype application. We ended this section by stating that implementing the peer-to-peer model was realistic indicated by already existing technology[9] in Augmented Reality using this model. We eventually went for the client-server model because it is more appropriate for the small prototype we would like to build and the lack of hardware for the peer-to-peer model. We ended this chapter by explaining the prototype we built using the framework omitting all implementation details for later.

After giving the high-level explanation of the navigation framework and its subsequent prototype application, we continued on to the low-level implementation details. We started by giving a hardware overview followed by how we set up the private internet network for the client and server communication. We then discussed all system components of the framework package in more detail regarding implementation. The server component — being such an important component of the framework and prototype— got its own section where it was discussed more thoroughly. The client application was developed fully independently from the framework. We talked about its client code, the Mindstorms robot host with its new firmware and how data packets were passed through the server to the client's network endpoint.

Evaluating the prototype happened by using the User Experience Questionnaire[1]. We gathered 5 student participants and made them interact on both the desktop computer and on the HoloLens respectively. We compared results of both which each other and also with a benchmark dataset provided on the questionnaire's website. We noticed that our desktop interface was efficient for performing user tasks and relatively easy to learn but that it lacked creativeness compared to the benchmark dataset and the HoloLens user experience. Using the keyboard is a popular yet very orthodox method for user input so this could be expected. Voice commands on the immersive device were very easy to use and a novelty for the participants. It scored low in terms of efficiency and dependability requiring much more effort from the user to perform tasks compared to a keyboard. Overall, both user experiences were somewhat successful.

The whole thesis revolved around developing a framework that would

[1] `https://www.ueq-online.org/`

allow us to create an application that could mimic Microsoft's demo[2] on the HoloLens' cross-space capabilities. During the development process however, we encountered some technical issues on our immersive device and had to change the requirements a bit.

6.1.1 Difficulties

Object tracking was achieved successfully on the Unity editor and we were able to map virtual information on the target object by using the host computer's webcam. On the HoloLens device, our object tracking engine had trouble deploying. Tracking real-life objects and mapping virtual information on it was therefore not achieved. As a consequence, we could not implement our initial method for user input involving the placement of waypoints in our environment with eye gaze and using path-finding for the robot to follow these waypoints.

6.1.2 Changed Requirements

Our initial methods for user input had to be changed because of the deployment issues faced on the HoloLens. We therefore implemented voice commands as input methods allowing the robot to move in any direction. The navigation of virtual objects on the HoloLens was not realised.

6.2 Future Work

We now discuss aspects of our framework to which we could apply improvements in later development. As was mentioned previously, difficulties encountered during development prompted us to alter our functional requirements in terms of navigating game objects and user input methods.

Introducing our problem domains, we specified that our framework should allow for the correct navigation of game objects in a dynamic environment. The subsequent prototype application developed using said framework achieved this on the desktop computer. In spite of our best efforts, navigating objects on the HoloLens was not achieved because of faulty deployment. If we were to do further development on our framework, this is one of the key problems to solve. A possible reason for the faulty deployment be a faulty integration of the three-dimensional scan of the target by the tracking engine. The scan was made by using the provided application[3] on the engine's developer portal.

[2] `https://www.youtube.com/watch?v=xnrHFV34PfM`
[3] `https://developer.vuforia.com/downloads/tool`

The app was experimental and this might have caused the deployment issues on the HoloLens. A solution would be to develop our own three-dimensional mesh of the target Mindstorms robot. By passing this mesh to the tracking engine, we should achieve object tracking and correct deployment on the imemersive device.

Our framework uses a client-server architecture. In the prototype application, our HoloLens immersive device acts as server and the mobile robot as client. Using such architecture proved convenient for a small application but it's really difficult to scale up. Extending our framework to allow for a peer-to-peer architecture can be interesting for future development. Our system design section mentions other software projects who successfully implemented this architecture in the field of Augmented Reality. Our main issue during the development process was the lack of hardware to debug our potential alternative architecture and to built a prototype showing its capabilities.

User experience on both devices have room for improvements. The desktop interface performed well in efficiency and perspicuity: tasks could be completed efficiently and the interface was easy to learn. Its biggest fault was the lack of creativity and novelty by using the desktop's keyboard for user input. Exploring more innovative methods for input such as handheld controllers can resolve this in future work. The HoloLens experience on the other hand was innovative and creative but lacked efficiency and dependability. Tasks required more effort from the user to be completed and —occasionally— voice commands were not processed correctly. Implementing our initial methods for user input —waypoints and path-finding— in future work could improve efficiency.

A Appendix

A.1 Notable Java Classes

Listing A.1: Motorops class for controlling the robot's motors.

```
import lejos.hardware.BrickFinder;

/** This class provides us control over the motors of the EV3 Mindstorms robot. */
public class MotorOps {

    /** Get the EV3 brick's screen. */
    static GraphicsLCD g = BrickFinder.getDefault().getGraphicsLCD();

    /**
     * Concreticize the motors A, B, C and D of the Lego robot.
     * We built a Lego robot with two claws and 2 tires on its feet.
     * On the brick, following ports have following motors:
     *  A = Right-arm   B = Left-tire   C = Right-tire  D = Left-arm  */
    static BaseRegulatedMotor motorA = new EV3LargeRegulatedMotor(MotorPort.A);
    static BaseRegulatedMotor motorB = new EV3LargeRegulatedMotor(MotorPort.B);
    static BaseRegulatedMotor motorC = new EV3LargeRegulatedMotor(MotorPort.C);
    static BaseRegulatedMotor motorD = new EV3LargeRegulatedMotor(MotorPort.D);

    /** Draw a message on the EV3 brick's screen. */
    public static void screenMessage(String msg){
        /** Clear the screen then write the message */
        g.clear();
        g.drawString(msg, 0, 0, GraphicsLCD.VCENTER | GraphicsLCD.LEFT);
```

```
}


/** Predefined trajectories (tricks) for the robot to follow. */
/** Pivot 180 degrees */

public static void pivot180(){
    motorC.forward();
    Delay.msDelay(6000);
}

/** Pivot 360 degrees */
public static void pivot360(){
    motorC.forward();
    Delay.msDelay(12000);
    motorC.stop();
}


/** These are the primary methods used to move the robot by the client. */
public static void continueForward(){
    motorB.forward();
    motorC.forward();
}

public static void continueBackward(){
    motorB.backward();
    motorC.backward();
}

public static void continueLeft(){
    motorC.forward();
    /** Stop motor B: it could still be active. */
    motorB.stop();
}
```

Listing A.2: Client class for managing the communication with the server and steering the motors based on received messages.

```
import java.io.BufferedReader;

/** This class represents the TCP client who will communicate with the server (Unity &
HoloLens) */
public class Client {

    /** Portnumber the client will be polling (listening) */
    private static int desktop_port = 4002;
    private static int hololens_port = 80;
    /** IP addresses that we will be using: one for the Desktop and the HoloLens */
    private static String IP_Address_Desktop = "192.168.0.100";
    private static String IP_Address_HoloLens = "192.168.0.103";

    /** IP address & Port number in use. */
    private static int port = desktop_port;
    private static String current_IP = IP_Address_Desktop;

    /** Socket for sending information to the server. */
    private static Socket socket;

    /** Streams that we can use to send & receive data from the server. */
    static InputStream inputServer;
    static DataInputStream in;

    /** Instantiate the Mindstorms motor management class. */
    static MotorOps motors = new MotorOps();

    /** Setup a timer to implement the death clock. */
    static Timer timer = new Timer();

    /** Variables that indicate if the client's still trying to connect with the server
     *  and whether we already placed the notification that the client is connected. */
    public static boolean tryConnection = true;
    public static boolean connectNotificationSend = false;

    /** Number of allowed attempts to connect with the server. */
    public static int noOfTries = 10;

    /** Deathclock: will stop the client program after given number of seconds. */
    public static void deathClock(int seconds){
        /** Schedule to end the program in given amount of seconds.
         *  We will exit the program execution after given time. */
        timer.schedule(new TimerTask() {
              public void run() {
                  System.exit(0);
                  }
              }, seconds*1000);}

    public static void readBytes(){

        try {
            inputServer = socket.getInputStream();
            in = new DataInputStream(inputServer);
            byte[] buffer = new byte[4];
            int read = 0;

            /** If something is received, read it and interpret it. */
            while ((read = in.read(buffer, 0, buffer.length)) != -1) {
                /** What message did we receive from the server? */
                String message = new String(buffer, "UTF-8");
                motors.screenMessage("Server says: " + message);
                /** Interpret the number message. */
```

```
                interpretMessage(message);
            }

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    /** This function will interpretate the response of the server */
    public static void interpretMessage(String response){
        /** The messages are one character long, are received in a stream and have different
meanings:
         *  - 1: continue forward.
         *  - 2: continue backward.
         *  - 3: continue to the left.
         *  - 4: continue to the right.
         *  - N: N(o instructions).
         * */
        if (response.startsWith("1")){
            motors.continueForward();
        }
        else if (response.startsWith("2")){
            motors.continueBackward();
        }
        else if(response.startsWith("3")){
            motors.continueLeft();
        }
        else if(response.startsWith("4")){
            motors.continueRight();
        }
        else if(response.startsWith("N")){
            motors.screenMessage("No instructions. Stand-by.");
            motors.stopAll();

        }
        else {
            motors.screenMessage("Not recognized: " + response);
            System.out.println("not recognized: " + response);
        }
    }


    public static void main(String[] args)
    {
        /** Notification that the client was started */
        motors.screenMessage("Client started. ");

        /** Set a death clock for the client: he will die in 60 seconds. */
        deathClock(60);

        /** Try connecting a couple of times with the server. */
        while (tryConnection){
            try
            {   /** Insert the public IP address of the server and the port he is listening
to. */
                socket = new Socket(current_IP, port);

                /** Send a notification that the socket is connected. */
                if (socket.isConnected()){
                    motors.screenMessage("Client connected. ");
                    connectNotificationSend = true;
                }
```

```
                /** Read the bytes the server put on the stream. */
                readBytes();
            }
            catch(Exception e){
                noOfTries = noOfTries - 1;
                if (noOfTries == 0){
                    tryConnection = false;
                    /** After a number of tries, print that the client just could not connect.
*/
                    motors.screenMessage("could not: " + e.toString());
                    /** Stop everything in 5 seconds */
                    deathClock(5);
                }
            }
        }
    }
}
```

A.2 Notable C# Classes

Listing A.3: KeywordManager class to allow for voice commands.

```
using System.Linq;
using UnityEngine.Events;
using UnityEngine.Windows.Speech;
using System.Collections.Generic;
using UnityEngine;

    public partial class KeywordManager : MonoBehaviour
    {
        [System.Serializable]
        /* This structure has two fields for every element in it:
         * a keyword string and a UnityEvent method to bind it with. */
        public struct KeywordsBinding
        {
            public string Keyword;
            public UnityEvent Method;
        }

        public KeywordsBinding[] KeywordMethods;
        private KeywordRecognizer keywordRecognizer;
        private Dictionary<string, UnityEvent> invokeMethods;

        void Start()
        {
            if (KeywordMethods.Length > 0)
            {
```

```
                /* Add the keywords and their associated responses to a
                 dictionary with as key the keywords
                * and as value the response events. */
                invokeMethods = KeywordMethods.ToDictionary
                  (keywordAndResponse => keywordAndResponse.Keyword,

keywordAndResponse => keywordAndResponse.Method);

                /* Instantiate Unity's keyword recognizer, bind a function
                 * for when a keyword is recognized and start it.*/
                keywordRecognizer = new KeywordRecognizer
                  (invokeMethods.Keys.ToArray());
                keywordRecognizer.OnPhraseRecognized +=
                  KeywordRecognizer_OnPhraseRecognized;
                keywordRecognizer.Start();
            }
        }


    private void KeywordRecognizer_OnPhraseRecognized
      (PhraseRecognizedEventArgs args)
    {
        UnityEvent keywordMethod;
        /* Bind the variable keywordMethod with the method associated
          with the spoken word. */
        if (invokeMethods.TryGetValue(args.text, out keywordMethod))
        {
            /* If something got found, invoke it. */
            keywordMethod.Invoke();
        }}}
```

Bibliography

[1] Rahib H Abiyev, Nurullah Akkaya, Ersin Aytac, Irfan Günsel, and Ahmet Çağman. Improved path-finding algorithm for robot soccers. *Journal of Automation and Control Engineering*, 3(5), 2015.

[2] Yen-Ling Chen and Pei-Chien Hung. Intuitive augmented reality navigation system design–implementation by next-gene20 project. In *International Conference on the Association for Computer-Aided Architectural Design Research in Asia*, pages 351–360, 2009.

[3] Andrew I Comport, Éric Marchand, and François Chaumette. A real-time tracker for markerless augmented reality. page 36, 2003.

[4] Emmanuel Dubois, Philip Gray, and Laurence Nigay. *The engineering of mixed reality systems*. Springer Science & Business Media, 2009.

[5] Gabriel Evans, Jack Miller, Mariangely Iglesias Pena, Anastacia MacAllister, and Eliot Winer. Evaluating the microsoft hololens through an augmented reality assembly application. 10197:101970V, 2017.

[6] Thomas A Funkhouser. Ring: A client-server system for multi-user virtual environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 85–ff. ACM, 1995.

[7] Wolfgang Hoenig, Christina Milanes, Lisa Scaria, Thai Phan, Mark Bolas, and Nora Ayanian. Mixed reality for robotics. pages 5382–5387, 2015.

[8] Charles E Hughes, Eileen Smith, CB Stapleton, and Darin E Hughes. Augmenting museum experiences with mixed reality. In *Proceedings of KSCE 2004*, pages 22–24, 2004.

[9] Joaquín Keller and Gwendal Simon. Toward a peer-to-peer shared virtual reality. In *null*, page 695. IEEE, 2002.

[10] Ernst Kruijff, J Edward Swan, and Steven Feiner. Perceptual issues in augmented reality revisited. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 3–12. IEEE, 2010.

[11] Mikko Kytö, Barrett Ens, Thammathip Piumsomboon, Gun A Lee, and Mark Billinghurst. Pinpointing: Precise head-and eye-based target selection for augmented reality. page 81, 2018.

[12] Robert Gabriel Lupu and Florina Ungureanu. A survey of eye tracking methods and applications. *Buletinul Institutului Politehnic din Iasi, Automatic Control and Computer Science Section*, 3:72–86, 2013.

[13] Daniel Mestre, Philippe Fuchs, A Berthoz, and JL Vercher. Immersion et présence. *Le traité de la réalité virtuelle. Paris: Ecole des Mines de Paris*, pages 309–38, 2006.

[14] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329, 1994.

[15] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[16] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[17] David R Walton and Anthony Steed. Accurate real-time occlusion for mixed reality. page 11, 2017.