Title of the project

Subtitle

Project Report

Group: SW805F20

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300
9220 Aalborg East, DK

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9220 Aalborg East, DK
www.cs.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Title of the project

Theme:
Mobility

Project Period:
Spring Semester 2020

Project Group:
SW805F20

Participant(s):
Andreas Stenshøj
Daniel Moesgaard Andersen
Frederik Valdemar Schrøder
Jens Petur Tróndarson
Rasmus Bundgaard Eduardsen
Mathias Møller Lybech

Supervisor(s):
Brian Nielsen

Copies: 1

Page Numbers: 33

Date of Completion:
May 28th, 2020

Abstract:

This is the best abstract ever written

Contents

Abbreviations

Terms and abbreviations used in the report:

Chapter 1

Introduction

1.1 Project idea

The idea for this project is to create a team building game using augmented reality (AR). Two teams will compete against each other to score the most goals using a ball. Each player will be equipped with a Google Cardboard, and these will display the playing field from a top-down 2D view. To achieve this, each player's position needs to be tracked as well as where the ball is located on the field. In the top down view each player needs to see the positions of the other players and the ball. They also need to see their own position on the playing field and where the goalposts are. The players should be able to set an amount of goals they need to score to win before beginning the game.

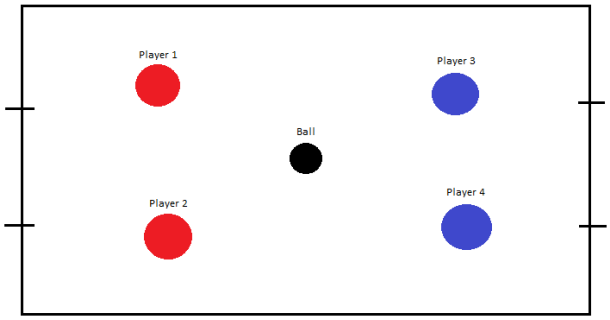

Figure 1.1: An illustration of the playing field

In Figure 1.1, an illustration of the playing field for the game is shown. There are goalposts in each end of the field, and the teams score goals by getting the ball between the goalposts. There could also be another version of the game where, instead of goalposts, there would be goal zones into which the teams need to bring the ball. These zones could even change locations as the game progressed.

1.1.1 Problems to consider

The project idea proposes some problems that will need to be solved for the game to work. We will need to consider which technologies to use for development of the visual aspect of the game which shows the top-down view for each player. As it is something we do not have experience with, it would be preferable if we do not have to build it from scratch. We will also need hardware that is able to track the positions of players and the ball. This must be accurate and update quickly such that the players do not run into each other, otherwise the game will not work. Another problem to consider is how the ball should be displayed in a 2D view. For the players to be able to find the ball on the field, it either has to be quite large to make it easier to find from the top-down view, or the game will need some metric to display how far the ball is from the ground. The game will also need to be able to track when the ball has crossed the goalline and then give feedback to the players. Another problem to solve is how to keep the positional data synchronized across all the players' devices, as it will be difficult to play the game without accurate data.

1.2 Essence

For the process of project development, we have chosen to work with Essence. Previous semesters we have worked with an agile approach inspired by Scrum, however, this semester we are attempting to apply the Essence approach. The basic idea of Essence is to encourage diverse thinking in the team, even though all members of our team share a similar background as bachelors in Software.

Essence uses two strategies to support value creation:

- *A systematic use of diverse viewpoints.* Values, views, and roles are used frequently in Essence. By using different views and roles to represent problems and solutions, Essence tries to facilitate a range of viewpoints on how a problem needs to be understood and solved.
- *A focus on idea maturation more than idea generation.* Essence applies the concept that ideas develop over time and tries to stimulate the team to evaluate and refine ideas [1].

1.2.1 Four variants of innovation

Essence tries to support innovation in software development, and hereby it defines four different variants of innovation, which are: [1]:

- *Product innovation* is new or radically changed software products or services.
- *Process innovation* is software solutions offering the user new or radically improved ways to produce products or services.

- *Project innovation* is fitting software solutions from earlier projects into new application domains
- *Paradigm innovation* is about software solutions coupled with changes in the mental model of what a business is, who the users are or what the market is.

1.2.2 Paradigms

There are two well-established software development paradigms: the document-oriented paradigm, which we know from the waterfall approach and agile paradigm which we know from for example extreme programming and Scrum. The author of Essence considers the new emerging paradigm called the pragmatic paradigm.

The document-oriented paradigm portrayed software developers as being document-oriented. The requirements are static and allow for a top-down waterfall approach to software development, which pays small attention to creativity and innovation.

The agile paradigm sees software development as user-oriented. Requirements are presumed dynamic as customers learn about options and constraints within the course of the project. This leads to incremental software development.

The pragmatic paradigm is problem-oriented. Systems are becoming more complex and it is more difficult to separate systems from each other. The amount of data, software libraries and hardware components available is steadily increasing, leading to hypercomplex software projects. Hypercomplexity is a degree of complexity that makes it impossible to make rational decisions within a reasonable time constraint. The most important features of this paradigm are that requirements are not completely known when the projects start. Ideas evolve in the process of the project, and during the project the requirements for the project are negotiable [1].

1.2.3 Core concerns

All software projects involve these four core concerns:

- Do we know what to build?
- Do we understand the solution?
- Do we understand the problem?
- Should we pivot or persevere?

1.2.4 Team organization

Within the team organization, in Essence, roles are used to create heterogeneity in teams, to ensure diverse points on views and to ensure cohesion despite diversity.

The focus of these roles is to increase learning with personal interaction by sharing insights and experiences. The roles also ensure that the team understands the problem domain, and see potentials in the technology domain.

As a rule of thumb, the roles are persistent meaning that a member will have the same role for the duration of the project. The roles in Essence are compatible with agile software development, making it possible to combine Essence with other processes like Scrum.

There are the four roles in essence:

- Child
- Responder
- Challenger
- Anchor

The role of *Child* can ask any questions and make propositions that are opposite of previous decisions. The rest of the team is not allowed to criticize the child, but they are however allowed to ignore the person's suggestions. The child is one of the main sources of ideas and other perspectives on the project. Outsiders are also allowed to take this role.

Responders are the developers in the team, and they are usually the majority within the team. Responders work closely together with the *Challenger*, so that the most important features are developed first.

Challenger is the customer or customer representative. The challenger can be compared to the *Product Owner* in Scrum. This role formulates and explains the Challenge, prioritizes features and accepts the solutions. There can be more than one Challenger, but if there are they must agree on the product vision.

Anchor is the one responsible for leading evaluations but does not decide the consequences. If necessary, the anchor can intervene and remove threats to the team's ability to develop ambitious responses. A potential threat could be something that results in productivity issues.

Chapter 2

Sprint 1

2.1 Prototypes

To come to a common vision about the layout and functionality of the system, a series of prototypes were created. The primary focus of these prototypes is not to be used for implementation, but rather for comparing opinions about how the flow in the system should be created.

Since the user interface is mostly focused on the mobile devices that the players will be wearing, it was decided that the host computer should simply have a text-based interface, as seen on Figure 2.1.

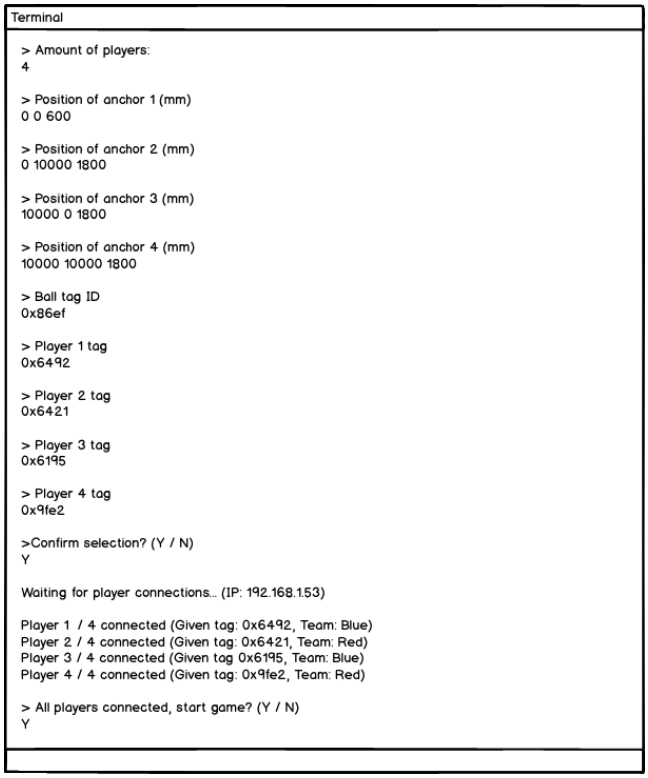

Figure 2.1: Prototype of hosting interface

When a user starts the application, they will be greeted with an input field, where they will specify the IP address of the host machine, as seen on Figure 2.2.

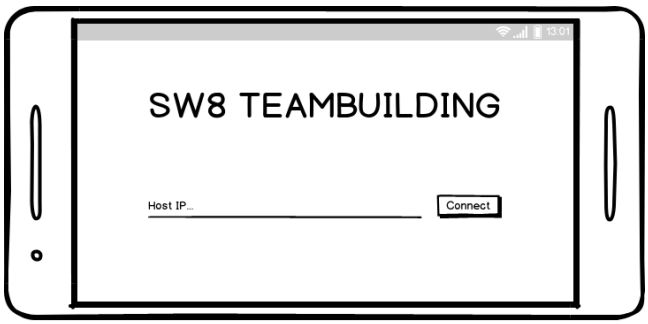

Figure 2.2: Prototype of game menu

After inputting an IP and confirming, they will be redirected to a page where they can see how many users have connected to the host, as seen on Figure 2.3.

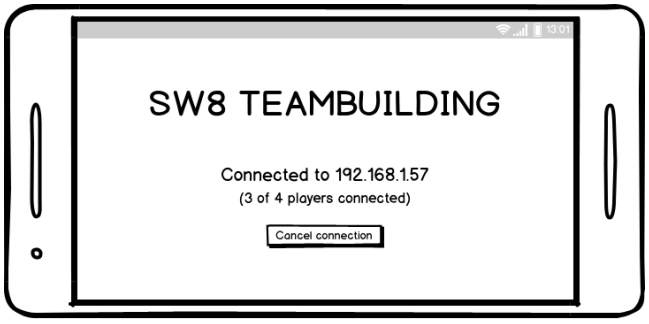

Figure 2.3: Prototype of screen where a user has connected to the host

When all users have connected, the host can start the game, and they will now see the virtual game field, as seen on Figure 2.4. In this prototype, the player's icon is highlighted by having a solid color, whereas the other players are just shown as outlines. In the middle of the screen is the ball in a designated starting area to make the game fair for both teams.

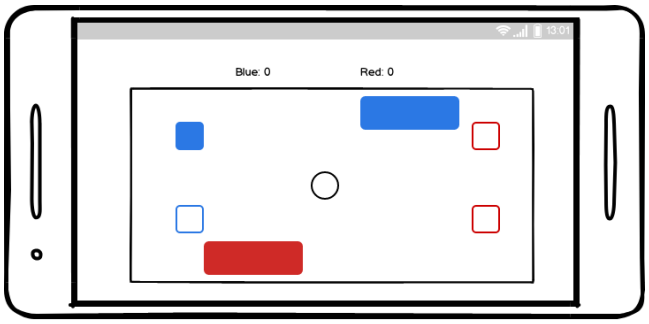

Figure 2.4: Prototype of in-game screen

2.2 Pozyx

Pozyx is a hardware/software solution that is used to provide positioning with an accuracy of down to 10 cm [4]. It makes use of ultra-wideband in combination with machine learning for positioning, which according to their documentation is more precise and efficient than traditional positioning systems such as WiFi, Bluetooth, RFID, and GPS.
Since the two major requirements for positioning in this project are precision and a high update rate in order to ensure that the players can have reliable data available, the Pozyx system seems like a good place to start.
The Pozyx tags support update rates of up to 125 Hz for a single tag [4]. The Creator system from Pozyx is sold with 4 anchors and 5 locatable tags. An anchor is a stationary sensor used by the moveable tags to get their exact position.

2.2.1 Finding the location of anchors

A trilateration method is used for finding the position of a given tag using the anchors. This method uses basic geometry to estimate the position by measuring the distance to the anchors of which we know the position. With this data, it is possible to draw a circle with a given radius. If we use two anchors, we will have two intersection points which are the possible positions of the tag. This means that to find a two-dimensional location, we will need at least three anchors, which will lead to only a single point where all three circles intersect, as seen on Figure 2.5

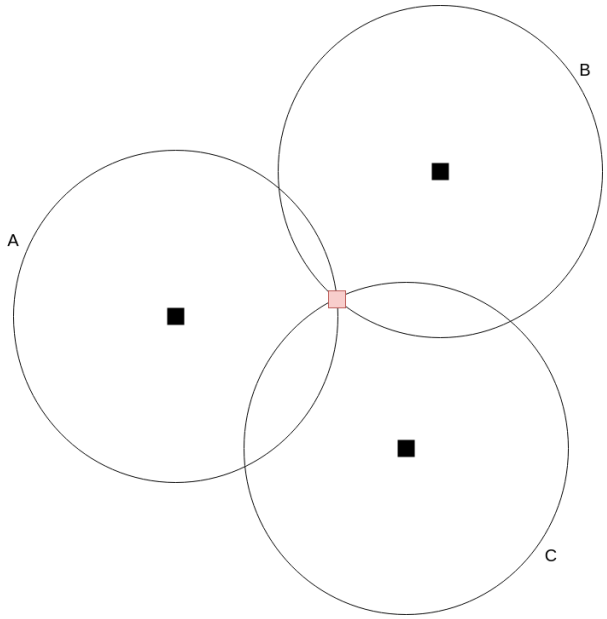

Figure 2.5: An example of trilateration using three anchors

The issue with this approach is that the measurements are not perfect, which might cause the circles to not intersect at exactly one point which will make the positional data seem to jitter.

2.2.2 Using UWB

To find the position of the tags Pozyx makes use of radio waves. Radio waves travel at the speed of light, so by dividing the time of travel between anchors with the speed of light, the distance between them can be found. Because the speed of light is so fast the time measure needs to be very accurate to get the correct distance. To achieve this the anchors make use of ultra-wide bandwidth (UWB) [5]. The ultra-wideband signals that the Pozyx devices utilize has a bandwidth of 500 MHz. This makes the wavelength very short and by detecting the peak of a narrow "pulse", an accurate time can be found. High bandwidth means faster data transfer, which most people would prefer, but if everyone were to use the same frequency the signals

would interfere with each other, therefore the use of high-frequency signals is tightly regulated. Pozyx can use 500 MHz because it transmits at very low power.

2.2.3 Two-way-ranging

We are using the Pozyx Creator Kit Lite which uses the Two-way-ranging (TWR) protocol for positioning [6]. A tag calculates its position by communicating with the anchors one by one, getting the distance from the anchor to itself. Once it has the distance from 3 anchors it can compute its position utilizing trilateration.
If multiple tags are being used at once, one tag is made the master tag and the other tags become the slave tags. The master tag instructs the slave tags to report their position to the master tag one by one. The master tag is then usually connected to a computer that can use the position data. This technique does not scale well as all the slave tags have to be within the radio range of the master tag so spreading them across huge areas is not possible. Instead of a tag being the master it is possible to use an anchor. This makes it easier to have a computer attached, as the anchors are stationary, unlike the tags.

2.3 Unity introduction

As defined in FUK, this project aims to create a team-building augmented reality game. This means the project has to have a game component - an application to display the objectives of the game, the play area and the players. To create this, a game engine can be used, such as `Unity`. A game engine is a piece of software that provides creators with the necessary set of features to build games quickly and efficiently[10]. This means that a game engine is a collection of reusable components, abstracted away from the game developer. This can include tools to help with, for example, graphics, physics, networking or audio. These tools would expose certain functionality to a developer to make use of, and hide the specific implementation details for that functionality, ensuring the developer can focus on more pressing issues. Unity supports the C# language for development[7].

Add ref to project idea section

The Unity game engine supports development for different game platforms. Of particular interest to this project is the support for both `Android` and `iOS` devices, as well as `Google Cardboard`[8]. We chose to use Unity for the development of the game aspect of this project. This facilitates that a greater amount of time can be spent on the other aspects of the project rather than the low-level details of game development, and it allows for easier inclusion of multiple platforms.

2.4 Networking

The following section will examine the different possibilities regarding transmitting player position data to the Unity applications used to play the game.

2.4.1 Possible networking solutions

Unity will be used for the creation of the game aspect of this project as described in section 2.3. Unity includes a proprietary networking solution known as UNet [9]. This solution allows developers to use a high-level API, giving access to commands that cover many common requirements for multiplayer games, without worrying about the low-level details. Since the solution is developed alongside the actual game engine, it has a higher level of integration with the Unity Editor and Engine, which allows for certain components and visual aids to aid the building of the game. As of the beginning of this project, the UNet solution has been deprecated for a while, and the Unity developers are actively working to create a new system to replace it. The current UNet iteration is usable but will be removed in the future. Other third-party solutions for Unity-based games also exist, such as Photon Engine. Photon provides functionality for the developers to make use of to create multiplayer games in the same way as UNet, exposing higher-level functionality. Photon supports multiple platforms outside of just Unity, with both Android and iOS support [3].

ZeroMQ is also a possible solution. ZeroMQ is an asynchronous messaging library. It can carry messages across various transport formats and is available in many different programming languages [11]. It aims to be a high-performance library to be used in distributed or concurrent applications that are reliable. According to the getting started guide provided by ZeroMQ, certain issues tend to arise when developers attempt to create a networking solution using sockets [12]. These are:

- How to handle I/O?
- How are dynamic components handled? What happens if a component disappears temporarily?
- How are messages represented? Different sizes and different content can change representations
- How are messages that cannot be delivered immediately handled?
- Where should message queues be stored?
- How are lost messages handled?
- What if the network transport changes, for example, TCP to UDP?
- How do messages get routed? Can the same message be sent to multiple peers?
- How to write an API for another language?
- How to represent data such that it can be read between different architectures? How much of this should be the messaging system's job?
- How do network errors get handled?

	Pre-existing	Custom	ZeroMQ
Customizability	Consists of a set of pre-defined functionalities	Can have any functionality implemented	Has pre-defined functionalities, but these are lower level than a pre-existing solution
Requirements	Familiarity with the solution	Familiarity with the knowledge required to implement a usable solution	Needs familiarity with a mix of pre-existing and custom solutions
Optimization	Lower-level details are obscured, optimized for general use	Lower-level details are freely available, can be optimized for a specific purpose	Focuses on performance, but the solution is general

Table 2.1: A comparison of the pros and cons of the possible solutions

These issues are mostly applicable to general solutions that need to accommodate changing requirements or be reusable. However, for this project, not all of these issues are relevant. In terms of problems to overcome, this project should only be concerned with handling dynamic components, handling lost messages, routing messages and handling network errors. If a player closes the game application it can lead to dynamic component issues. A message can be lost during the playing of the game. Messages should be delivered to all players to ensure that they all have the same information. Finally, a player might suddenly disconnect from the network.

The alternative to making use of a pre-existing solution is creating a custom solution. A custom solution entails a need to establish a familiarity with the required knowledge to construct such a solution. A custom solution would involve sockets, which are a network API that allows programs to communicate with each other [2].

2.4.2 Choosing a solution

There are certain pros and cons associated with both approaches of using either a pre-existing solution or a custom solution. Table 2.1 shows some of the considerations made when deciding an approach for this project. Based on these considerations, it was decided that a custom solution should be created to handle networking in this project. This choice was based on the lack of transparency in a pre-existing solution as well as the need for fast communication. For the game to be playable and enjoyable, the location data collected by the Pozyx system needs to be transmitted

to all the clients as quickly as possible such that they always have an up to date view of the positions of the players. To achieve this, it would be preferable to build a solution capable of performing the minimum amount of work as quickly as possible. Pre-existing solutions cannot be guaranteed to do the minimum amount of work as lower-level details are obscured from the developers. With a custom solution, the data sent across the network can be guaranteed to be exactly what is needed. ZeroMQ was also a possible choice based on the performance needs, but its generalized approach concerning itself with reusability and issues unlikely to be a big factor in this project meant it was dismissed, in favor of a custom solution in which the problems defined in the previous section are handled.

2.5 Experiment with Pozyx

We have created an experiment to test how accurate the tags in Pozyx are. The primary goal of the experiment was to test the accuracy, but a biproduct of the experiment was to see how many update frequencies there were with each tag.

2.5.1 Setup

The current settings goes for best precision, but gives a smaller amount of updates. The setup of the experiment as seen on Figure 2.6. The experiment was conducted indoors in Novi 9. The anchors `0x632b` and `0x676e` was mounted on a wall 240 centimeters apart and the anchors `0x6738` and `0x676c` was mounted on a bulletin board. The number of cm above the hexadecimal number is the height the anchor was mounted. There was chosen different heights as Pozyx documentation suggests that not all anchors should have the same height.

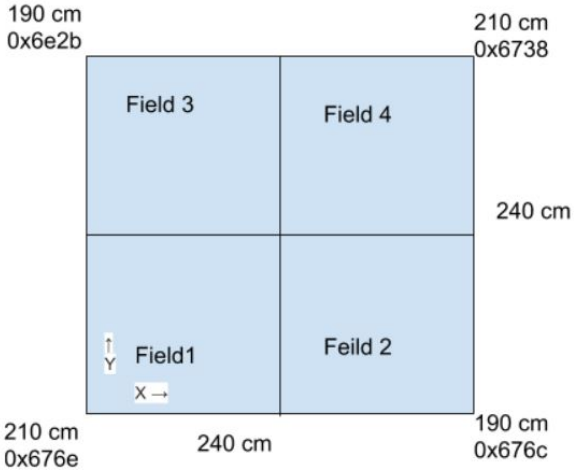

Figure 2.6: The setup of the experiment with the anchors and the height in the corners.

The fields was a blackboard where lines were drawn for each 10 centimeters to know the actual position as seen on.

Figure 2.7: The blackboard with the drawn positions.

2.5.2 Precision with 1 tag

As can be seen on subsection 7.1.1

Update frequency

2.5.3 Precision with 3 tag

24622 was sometimes fairly inaccurate due to there sometimes being few position updates. A few times there were no updates in a 2 seconds time spand.

Update frequency

2.5.4 Precision with 5 tag

Update frequency

2.5.5 Possible influences on the test

One thing that could have affected the tags is that we used a blackboard as a measure. As there is metal in the blackboard this could have affected the precision on the tags.

In the test with 1 tag, the 0 x coordinate was close to the wall. This could have been an influence so that it will be higher than it otherwise would be, as it was difficult to center the tag over the x coordinate where x was 0.

2.5.6 Conclusion on the experiment

Chapter 3

Sprint 2

Chapter 4

Sprint 3

Chapter 5

Sprint 4

Chapter 6

Conclusion

Chapter 7

Appendix

7.1 Results for experiment

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(120, 0)	(141.1, -9.0, 272.2)	(134.5)	(147.7)	(-15.1)	(1.6)	(79.1)	(325.5)
(120, 30)	(141.9, 29.3, 161.3)	(134.1)	(150.4)	(23.1)	(37.5)	(77.5)	(306.8)
(120, 60)	(133.8, 60.6, 89.0)	(132.3)	(135.3)	(59.9)	(61.2)	(87.5)	(90.5)
(120, 90)	(129.6, 72.8, 161.5)	(123.8)	(137.5)	(55.0)	(89.2)	(85.7)	(308.4)
(120, 120)	(123.8, 107.5, 91.7)	(114.7)	(128.9)	(90.9)	(120.3)	(80.2)	(105.0)

Table 7.2: Table with grids for experiment with 1 tag. Tag: 26467

7.1.1 Experiment with one tag

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(0, 0)	(0.9, 1.1, 117.9)	(0.0)	(34.0)	(-23.10)	(18.8)	(0.0)	(346.50)
(30, 0)	(36.4, 9.0, 156.6)	(26.6)	(51.2)	(1.3)	(21.30)	(55.10)	(336.7)
(60, 0)	(67.0, 11.3, 203.3)	(55.7)	(79.8)	(0.7)	(28.1)	(63.2)	(203.35)
(90, 0)	(99.2, 4.5, 248.6)	(83.4)	(108.2)	(-37.3)	(16.3)	(79.3)	(322.6)
(120, 0)	(133.0, 4.5, 249.0)	(109.4)	(172.1)	(-54.5)	(14.0)	(48.2)	(305)
(0, 30)	(22.8, 41.3, 220.7)	(4.6)	(35.2)	(10.0)	(62.7)	(24.5)	(36.2)
(30, 30)	(24.4, 38.5, 225.2)	(8.3)	(57.0)	(26.0)	(67.7)	(28.4)	(33.6)
(60, 30)	(66.8, 33.0, 149.2)	(55.4)	(85.4)	(19.3)	(50.1)	(52.5)	(33.1)
(90, 30)	(99.9, 26.5, 292.9)	(89.2)	(105.8)	(18.5)	(33.0)	(84.9)	(321.2)
(120, 30)	(122.1, 33.9, 283.4)	(104.8)	(136.8)	(14.1)	(53.3)	(64.3)	(31.3)
(0, 60)	(22.9, 41.3, 220.8)	(4.6)	(35.2)	(10.0)	(62.7)	(24.5)	(362.3)
(30, 60)	(34.6, 64.0, 64.0)	(-33.3)	(84.6)	(46.9)	(110.1)	(7.8)	(334.9)
(60, 60)	(62.1, 78.0, 282.4)	(47.0)	(81.5)	(59.0)	(97.5)	(70.5)	(338.1)
(90, 60)	(90.2, 63.9, 106.6)	(80.7)	(97.1)	(56.8)	(70.1)	(81.8)	(317.1)
(120, 60)	(105.6, 50.3, 150.0)	(62.2)	(135.5)	(13.8)	(74.3)	(76.2)	(336.5)
(0, 90)	(18.4, 98.1, 193.8)	(4.9)	(42.1)	(73.8)	(122.9)	(47.9)	(347.2)
(30, 90)	(33.0, 91.8, 261.7)	(-5.2)	(54.4)	(69.7)	(106.5)	(61.4)	(330)
(60, 90)	(64.0, 88.5, 120.6)	(54.1)	(76.7)	(76.2)	(98.8)	(68.9)	(317.6)
(90, 90)	(88.3, 92.5, 99.2)	(80.0)	(103.3)	(75.2)	(107.7)	(82.1)	(313.5)
(120, 90)	(114.1, 86.3, 124.0)	(88.1)	(136.4)	(69.7)	(112.7)	(65.0)	(314.8)
(0, 120)	(22.7, 118.1, 77.4)	(11.6)	(58.9)	(62.2)	(133.8)	(61.8)	(365.4)
(30, 120)	(34.9, 119.5, 197.5)	(15.9)	(56.4)	(96.8)	(138.6)	(82.1)	(333.1)
(60, 120)	(67.0, 124.2, 242.3)	(45.1)	(83.2)	(108.3)	(143.1)	(82.2)	(313.6)
(90, 120)	(91.3, 121.5, 104.5)	(79.4)	(105)	(114.3)	(130.0)	(76.5)	(315.2)
(120, 120)	(130.7, 117.9, 70.0)	(119.9)	(142.4)	(105.6)	(126.7)	(629.0)	(819)

Table 7.1: Table with grids for experiment with 1 tag. Tag: 26895

7.1.2 Experiment with three tags

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(180, 0)	(184.1, -2.0, 242.2)	(172.6)	(192.7)	(-10.7)	(18.2)	(86.6)	(304.9)
(180, 30)	(185.3, 18.6, 165.4)	(174.5)	(202.1)	(9.3)	(29.1)	(79.7)	(295.4)
(180, 60)	(181.4, 56.6, 173.2)	(173.7)	(192.9)	(48.7)	(62.8)	(84.9)	(307.2)
(180, 90)	(176.2, 94.0, 175.1)	(167.0)	(190.3)	(85.8)	(104.6)	(85.7)	(320.5)
(180, 120)	(175.5, 118.3, 82.4)	(168.6)	(180.5)	(112.8)	(122.7)	(74.0)	(94.5)

Table 7.3: Table with grids for experiment with 1 tag. Tag: 26895

Actual grid	Average grid (x,y,z)	x min	x max	y min	y max	z min	z max
(220, 0)	(228.4, 2.7, 270.8)	(210.8)	(236.1)	(-7.9)	(10.2)	(58.2)	(325.9)
(220, 30)	(228.8, 31.9, 261.7)	(219.0)	(232.0)	(25.3)	(51.3)	(96.7)	(298.2)
(220, 60)	(224.2, 54.0, 193.1)	(222.5)	(226.0)	(51.3)	(56.7)	(97.1)	(289.1)
(220, 90)	(226.9, 53.1, 139.5)	(205.8)	(255.1)	(49.6)	(61.0)	(76.5)	(276.5)
(220, 120)	(206.2, 101.4, 230.0)	(192.8)	(213.4)	(83.8)	(125.6)	(89.0)	(326.8)

Table 7.4: Table with grids for experiment with 1 tag. Tag: 24622

Bibliography

[1] Ivan Aaen. *Essence. Problem Based Digital Innovation*. 2020.

[2] Bill Fenner, Andrew M. Rudoff, and Richards W. Stevens. *UNIX Network Programming, Volume 1. The Sockets Networking API*. 3. edition. Addison-Wesley Professional, 2003.

[3] Exit Games. URL: https://www.photonengine.com/en-US/PUN (visited on 02/14/2019).

[4] *Home: Pozyx*. URL: https://www.pozyx.io/.

[5] *How does ultra-wideband work?* URL: https://www.pozyx.io/technology/how-does-uwb-work.

[6] *Positioning protocols explained*. URL: https://www.pozyx.io/technology/positioning-protocols-explained.

[7] Unity Technologies. URL: https://unity.com/how-to/programming-unity (visited on 02/13/2019).

[8] Unity Technologies. URL: https://unity3d.com/unity/features/multiplatform (visited on 02/13/2019).

[9] Unity Technologies. URL: https://docs.unity3d.com/Manual/UNet.html (visited on 02/14/2019).

[10] Unity Technologies. *Game engines - how do they work?* URL: https://unity3d.com/what-is-a-game-engine (visited on 02/13/2019).

[11] ZeroMQ. URL: https://zeromq.org/get-started/ (visited on 02/18/2019).

[12] ZeroMQ. URL: https://zguide.zeromq.org/page:all (visited on 02/18/2019).

List of Figures

List of Tables

Listings