Bachelor's thesis

Information and Communications Technology

2021

Niko Laivuori

EYE AND HAND TRACKING IN VR TRAINING APPLICATION

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS| ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2021 | 29 pages

Niko Laivuori

EYE AND HAND TRACKING IN VR TRAINING APPLICATION

The Varjo VR-2 Pro virtual reality head-mounted display has high resolution and in-built hand and eye tracking features. The purpose of this thesis was to research eye and hand tracking technologies used in VR applications, and add the Varjo head-mounted display to an existing virtual reality simulation meant for training sea captains in commanding a freighter ship. In the simulation the user is on the command bridge trying to navigate in an archipelago environment. The simulation is using older VR technology, resulting in low resolution and unintuitive user experience. These problems are tackled with the Varjo VR-2 Pro HMD. This thesis mainly focuses on the eye tracking features but also touches upon the hand tracking.

The work was carried out using Unity3D game engine, the existing command bridge simulation and Varjo libraries. In the end the Varjo HMD was integrated to the project and as a result the command bridge application was using eye tracking to evaluate the user performance to a small extent. Varjo hand tracking was also used in the project. The finished project works as a good starting point for more complex development ideas with Varjo VR glasses.

KEYWORDS:

virtual reality, Unity, Varjo, eye tracking, hand tracking

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

TIETO- JA VIESTINTÄTEKNIIKKA

2021 | 29 sivua

Niko Laivuori

SILMÄN JA KÄSIEN SEURANTA VIRTUAALITODELLISUUS KOULUTUSOHJELMASSA

Varjo VR-2 Pro -virtuaalilaseissa on korkea resoluutio ja sisäänrakennetut silmän- ja käsienseuranta ominaisuudet. Tämän työn tarkoituksena oli lisätä Varjon lasit olemassa olevaan virtuaalitodellisuusprojektiin, jonka tavoitteena on kouluttaa merikapteeneja rahtilaivan ohjaamisessa. Sovelluksessa käyttäjä ohjaa rahtilaivaa komentosillalla ja pyrkii navigoimaan saaristossa. Sovelluksen ongelma oli vanhan virtuaalilasiteknologian heikko resoluutio sekä intuitiivisen käyttökokemuksen puute. Näitä ongelmia pyrittiin ratkaisemaan Varjon modernimmilla laseilla. Tässä työssä keskityttiin erityisesti silmänseurannan käyttöön, mutta myös käsienseurantateknologiaa selvitettiin.

Työ toteutettiin komentosiltasimulaattorin tavoin Unity-pelimoottorissa. Työn toteutuksessa hyödynnettiin vanhaa komentosiltaprojektia ja Varjon tarjoamia kirjastoja. Lopputuloksena Varjo VR-2 Pro -virtuaalilasit saatiin toimimaan komentosiltaprojektissa, ja saatiin aikaan komentosiltasimulaattori, joka hyödyntää silmänseurantateknologiaa simulaattorin käyttäjän arvioimisessa. Työssä tutkittiin myös Varjon tarjoamaa käsienseurantateknologiaa. Työ toimii hyvänä alustana monimutkaisemmille jatkokehitysmahdollisuuksille Varjon virtuaalilasien kanssa.

ASIASANAT:

virtuaalitodellisuus, Unity, Varjo, silmänseuranta, käsienseuranta

CONTENTS

FIGURES

PICTURES

TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AR	Augmented Reality
CPU	Central Processing Unit
CSV	Comma-Separated Values
FIT	Future Interactive Technologies
FOV	Field of View
FPS	Frames Per Second
GPU	Graphics Processing Unit
HMD	Head-Mounted Display
MarISOT	Maritime Immersive Safe Oceanites Technology
MarSEVR	Maritime Safety Education with VR
PC	Personal Computer
PPCR	Pupil Center Cornea Reflection
TGL	Turku Game Lab
VR	Virtual Reality
VRS	Variable Rate Shading

1 INTRODUCTION

Virtual reality has been around for years now, and it has been mostly used to create immersive gaming applications. However, in recent years more companies outside the games industry have realised the potential use of VR applications in training and preparing their employees for their tasks. VR training simulations give practical experience to trainees, and it's been argued for a long time that practical learning is the most useful way to learn, allowing users to retain more information. VR training simulations have been found especially useful when it comes to physically dangerous or possibly costly and challenging tasks. A good example is medical professionals preparing for surgeries via VR training, learning form their mistakes and thus reducing the chance of repeating those mistakes with real patients [1].

There are problems when it comes to VR training, mainly the use of VR controllers in place of hands, and often the low resolution of HMDs (Head-mounted displays) compared to human eyes causes problems that might not be present in real-life situations. Both of the aforementioned elements also severely break the immersion when using VR application, which in turn reduces the usefulness of Virtual Reality in training purposes. After all, if the training scenario does not reflect real-life situations then there's really no point doing it in the first place.

A Finnish company called Varjo is known for their high quality HMDs, which have a resolution comparable to the human eye. Many Varjo HMDs have integrated accurate eye-tracking capabilities and also provides hand tracking features using technology from the UltraLeap company. Both of these tools combined with the high resolution can be used to enhance the effect of virtual reality training applications.

This thesis focuses mainly on eye tracking in virtual reality, but hand tracking is also researched. Turku University of Applied Sciences FIT (Future Interactive Technologies) research group has a virtual reality project called MarSEVR (Maritime Safety Education with VR), where the user is in control of a freighter ship and is navigating through an archipelago. MarSEVR is a VR safety training application meant to help reduce maritime accidents. It is part of MarISOT (Maritime Immersive Safe Oceanites Technology). MarISOT has several applications integrated to it trying to prevent maritime accidents, which often lead to human casualties, sea pollution and other environmental damages.

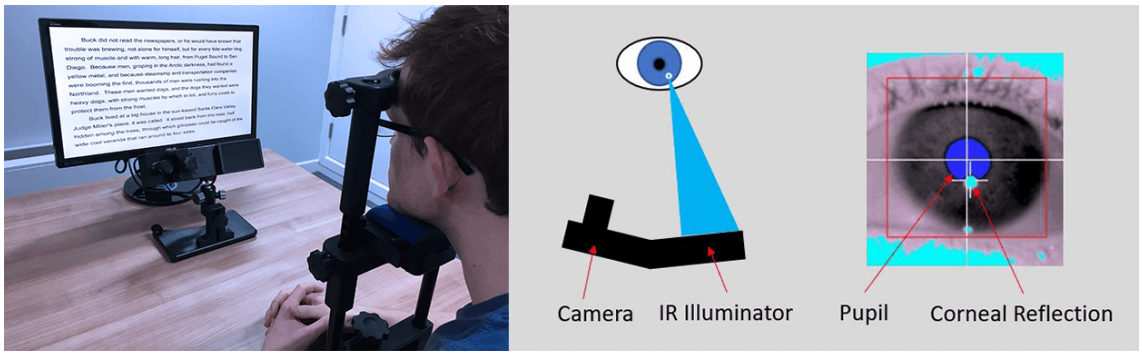
MarSEVR was using a low resolution virtual reality headset with controllers, and gathering no data about the user behaviour. The practical goal of this thesis was to implement Varjo VR-2 Pro and it's eye-tracking features into the existing VR application used for training purposes, and also to develop a small demo in Unity game engine to demonstrate some of the hand-tracking capabilities of Varjo VR-2 Pro. The finished simulation can then be used as a solid starting point for further development, with the Varjo features already implemented.

2 EYE AND HAND TRACKING THEORY IN VR

2.1 Eye Tracking

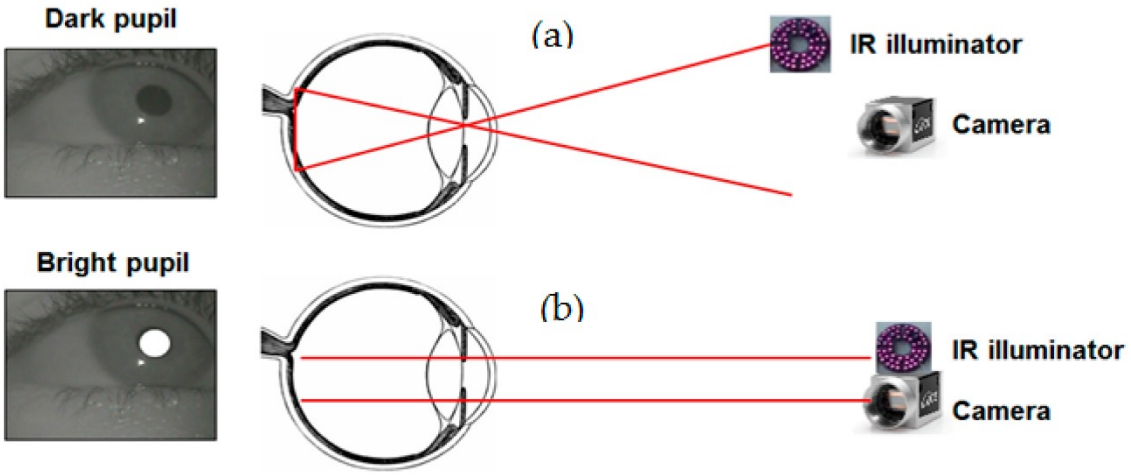
Eye tracking is measurement of eye activity. Some examples of eye activity would be the duration of us looking at something, how our pupils react to visual stimuli and also blinking and saccading (rapid eye movement when fixation point changes). In recent years more companies have started to research and find use cases for eye tracking, whether it is in a gaming environment, some kind of behavioural research or maybe user experience testing on a website.

One of the most popular techniques for eye tracking is Pupil Center Cornea Reflection (PCCR). PCCR uses a camera to track the pupil center and the light reflected from the cornea, also known as “glint”, using near-infrared light (Picture 1). Near-infrared light is not visible to human-eye and it creates much clearer reflection than light on the visible spectrum [3]. Cornea is the eye’s outer-most part on front covering the iris and pupil, and it is transparent [4].



Picture 1. Cornea reflection and pupil center tracking [2].

PCCR can be implemented in different ways. Dark and bright pupil tracking are two sides of the same coin, where bright pupil tracking places an infra-red illuminator close to the camera’s optical axis, which causes the pupil to lit up. Dark pupil tracking does the opposite, placing the illuminator away from the camera’s axis and showing the pupil dark (Picture 2)[5].

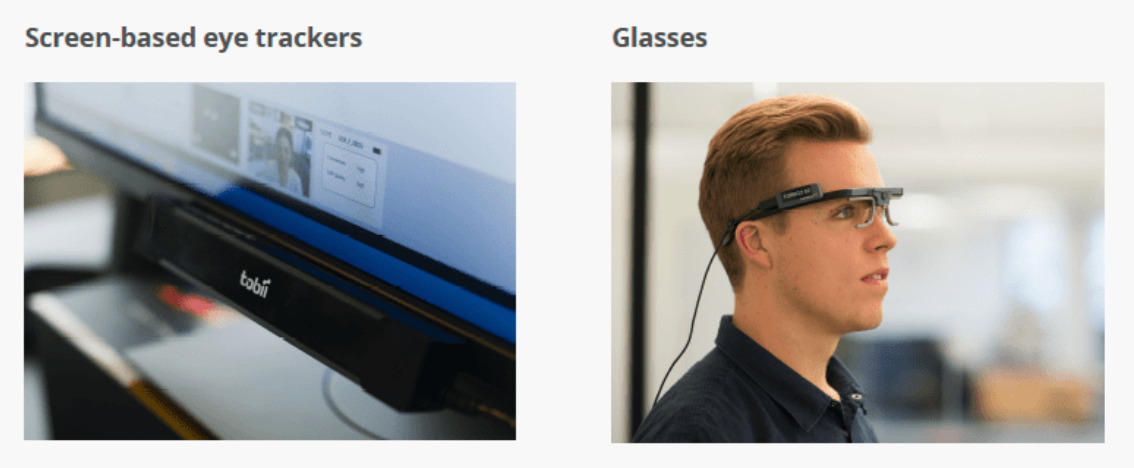


Picture 2. Bright and dark pupil tracking [5].

At the most basic level, eye tracking sensors produce coordinates called gaze points. Gaze points are usually in 2D (computer screen) or 3D (virtual reality). Typically, when a few gaze points are close to each other both in time and on the coordinate plane, it is somewhat safe to assume the user is fixated on something. If, on the other hand, the gaze points are varying heavily both in time and on the coordinate plane, the eyes are probably saccading. Fixations and saccades can be used to create many types of metrics, like heatmaps and fixation sequence maps which are generally speaking very useful for evaluating users [6].

There are two types of eye trackers, screen-based and glasses (Picture 3). Screen-based trackers are usually something resembling a webcam, and they are useful for observing the users reactions to stimuli on computer screen, such as images, videos or websites. Drawbacks for screen-based options are that usually the user is not free to move around and the tracking is less accurate compared to glasses [3]. However, screen-based tracking is usually cheaper and easier to setup.

Eye tracking glasses may refer to glasses that are specifically created for eye tracking or, for example, virtual reality headsets that have eye tracking integration. Glasses have few benefits compared to stationary screen-based trackers. First, they are very close to the eyes which allows them to be extremely accurate. Second, they are mobile and allow the user to move around completely or at least much more freely compared to screen-based trackers. This movement option expands the number of use cases for glasses enormously, for example, in virtual reality applications where it is quite usual for the user to move and turn around constantly.



Picture 3. Screen-based and glasses eye tracking [3].

Eye tracking has many applications in the world, and due to technology becoming easier to use and more accurate there is an increased number of applications using eye tracking as time passes. One of the most common fields that utilize eye tracking is cognitive psychology and neuroscience where eye tracking can be used to evaluate intellectual processes like perception, attention and even language comprehension [7]. Another important field for eye tracking is market research. There are multiple case studies that show how eye tracking has been used to analyze customer behaviour when they're shopping in physical stores and on the web. Eye tracking has helped companies with their product placement and package and web design. In many industries like manufacturing and construction, eye tracking has been used for training and onboarding staff, and making workplaces safer and more optimized for human performance [8].

2.2 Hand Tracking

Advances in the field of computer vision and artificial intelligence have made it possible to develop algorithms that allow computers to recognize hands and gestures from digital images, and have given birth to hand tracking.

Hand tracking is essential when it comes to achieving immersion within virtual reality. It includes terms such as finger tracking and gesture recognition, which are often confused with each other. However, there is much overlap between these terms, since they are methods that allow the user to interact with computers without physical touch or controller. Using hands is more natural and intuitive for most users compared to

controllers, and hand tracking is considered the default way to interact in AR Applications [9].

Although virtual reality is just now starting to gain momentum and see commercial success, hand tracking is by no means a new concept. The first attempt at commercial grade hand tracking was in 1987 with Nintendo Power Glove (Picture 4), which was based on DataGlove, or wired glove, technology patented by Thomas G. Zimmerman in 1982. The Power Glove was modified to work with consumer-grade hardware and it was affordable for consumers [10].

Picture 4. Nintendo Power Glove [10].

Naturally, hand tracking has evolved over thirty years. Modern hand tracking in virtual reality relies on sensors which are usually inside-out cameras on the HMD. Even though most of the work is done by software, having poor quality cameras can easily become a bottleneck for hand tracking performance. Very often HMDs have two cameras or sensors dedicated to hand tracking in some capacity. Two cameras allow for stereoscopic view, which enables better depth perception. Also the cameras should have a larger FOV than the actual display used by the HMD, because it makes it makes tracking easier and smoother, having the hands already being tracked when they enter the display FOV. Having at least the same framerate than the display reduces latency, and providing some illumination can be very helpful in hand tracking, because sometimes the environment might be too dimly lit for tracking to work properly. Like eye tracking, many hand tracking cameras rely on infrared light.

The images captured by the cameras/sensors are sent to the tracking engine. Tracking engine is a software that takes the raw hand tracking data, applies advanced artificial intelligence algorithms to it in order to recognize the hands. Ultraleap, the hand tracking provider for Varjo, uses a neural network to process the raw data and generate hand models which are then sent to applications which need them, for example a game engine like Unity.

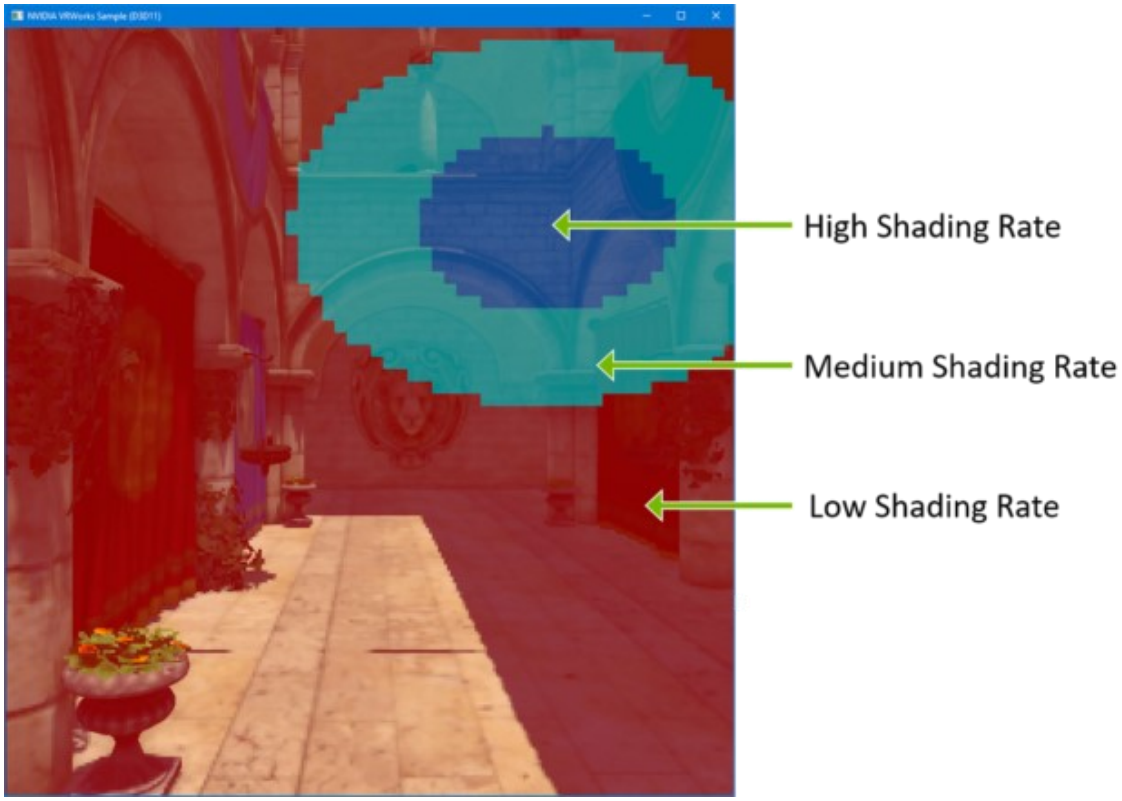
This brings us to the third and final stage, gesture recognizition. If a user is, for example, pointing their index finger at something, or their hand is clenched in a fist, the software should react some way. Gesture recognizition is something that is usually solved in the game engine by a plugin. Ultraleap provides a Unity plugin which contains what they call “Interaction Engine” that interprets the gestures and provides functionality based on them in the game engine [11].

3 EYE TRACKING IN VIRTUAL REALITY

This thesis focuses on eye tracking in virtual reality. In VR, eyes can be used to interact with the virtual world and also smoothen the experience with, for example, dynamic foveated rendering.

Dynamic foveated rendering is one most promising new technologies in VR. It tries to create more realistic method to observe the virtual reality scene, while simultaneously reducing the load on hardware (Picture 6). Human eye doesn't see the the entire field of view in same detail. Only the fovea, which is a very narrow area in FoV, is seen in high detail while the rest is sort of blurred. To put it simply, dynamic foveated rendering sharpens the image based on where the user is looking, and reduces the image quality in the peripheral, meaning the area on the field of view that the user is not fixated on [12]. It is able to do this based on gaze data.

One of the popular techniques to implement foveated rendering is variable rate shading or VRS (Picture 5). VRS means that we are changing the number of pixels, usually referred to as a block, that is sent to the pixel shader for processing [13]. In picture 7 the blue high shading rate area is processed pixel by pixel, or rather in 1 by 1 blocks. The medium rate area is processed in 2 by 2 blocks, which means that a single pixel shader process handles 4 pixels at once, reducing the total amount of processes needed and improving performance. Finally, the red area is processed in 4 by 4 blocks, significantly reducing the number of operations.



Picture 5. Variable rate shading [13].

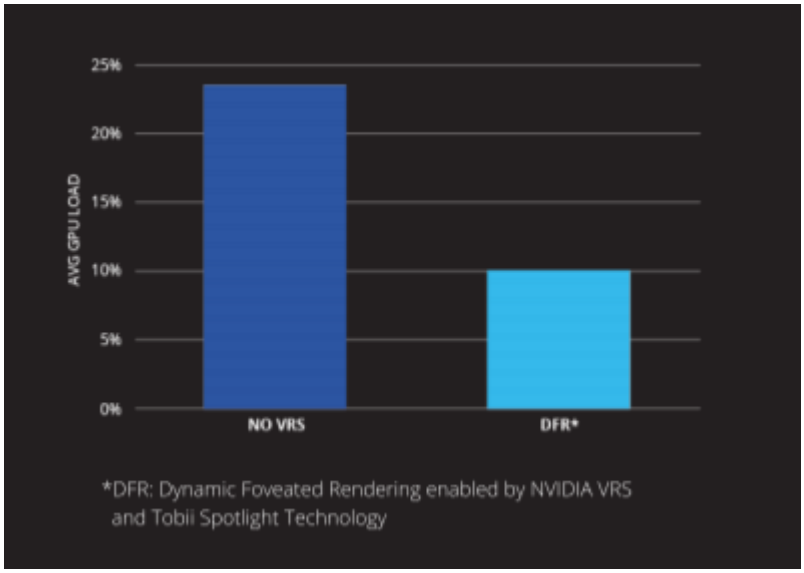


Figure 1. Tobii foveated rendering reducing GPU load [14].

First VR HMD with eye tracking feature built-in was called Fove 0, created by a Japanese company and released in 2017. It was already capable of doing dynamic foveated rendering. In 2021, Varjo has just released their VR-3 and XR-3 HMDs. Meanwhile, Tobii

eye tracking technology is used in HP Reverb G2 Omnicept Edition and HTC Vive Pro Eye. All of these HMDs have sub degree gaze accuracy, however the Varjo VR-3 and XR-3 HMDs have tracking frequency of 200Hz, compared to 120Hz with HP Reverb G2 Omnicept Edition and HTC Vive Pro Eye.

Some important metrics when it comes defining a good eye tracker are accuracy and precision, which are measured in degrees. The sampling rate is also very important. It means the frequency at which eye tracking data is captured.

Picture 6. Eye tracking accuracy and precision [15].

Virtual Reality and eye tracking are both utilized in many fields, and more companies are starting to see their usefulness. Both Varjo and Tobii have many case studies on their websites where clients ranging from medical professionals to astronauts, aviators to construction companies and many others are demonstrating benefits that they have gained from virtual reality applications and eye tracking used together. IMotions also have a lot of case studies on their website, and many of them involve psychological therapy. VR and eye tracking have been used for example in treating social anxiety and post-traumatic stress disorders [16].

4 PRACTICAL IMPLEMENTATION

4.1 Prerequisites

The previous version of the simulation was tested by maritime industry experts. According to them, one can determine the level of confidence of the commander through their eye movements, which can be an important factor in virtual reality training [17]. They were also concerned about hand controllers, which could cause unnecessary cognitive load for the user, ultimately resulting in confusion and coordination mistakes. They proposed that hands-free approach could potentially provide better results [18]. Varjo VR-2 Pro was chosen for this project because it has integrated eye and hand tracking features, provided by industry leaders like iMotions and UltraLeap.

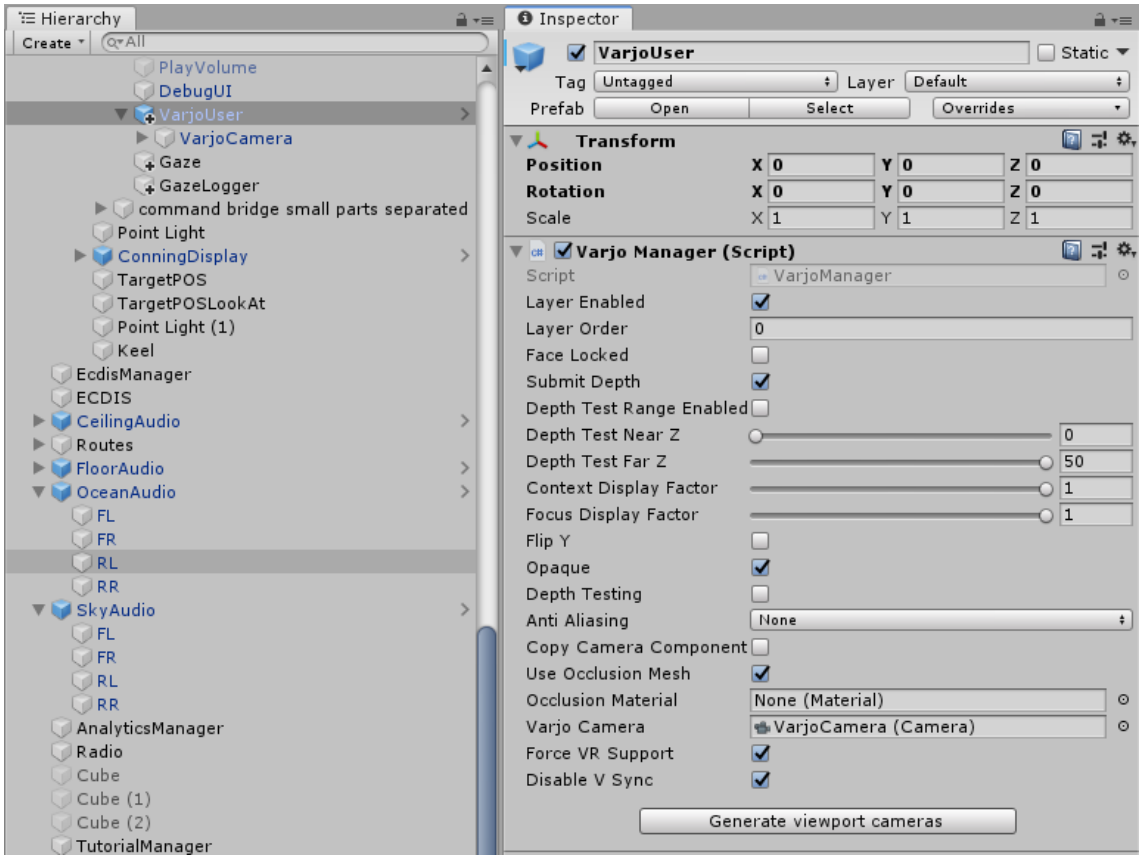
Varjo headsets require quite a lot of computing power. Thankfully TGL was able to provide a PC that could run Varjo without any issues.

First thing needed was Varjo Base software (Picture 7). Varjo Base is a companion software that is needed on a Windows 10 computer. It provides utilities like screen capturing and simulating button presses within the headset. It's main purpose is to configure the headset itself and to link it with SteamVR. SteamVR is Valve's virtual reality platform, consisting of both hardware and software. It allows the user to use different controllers and define a "play-area", a border where the user can move freely. At least one SteamVR base station is required to operate this. The base stations track the user's position in real life, which is then translated to virtual environment.

Picture 7. A look inside Varjo Base.

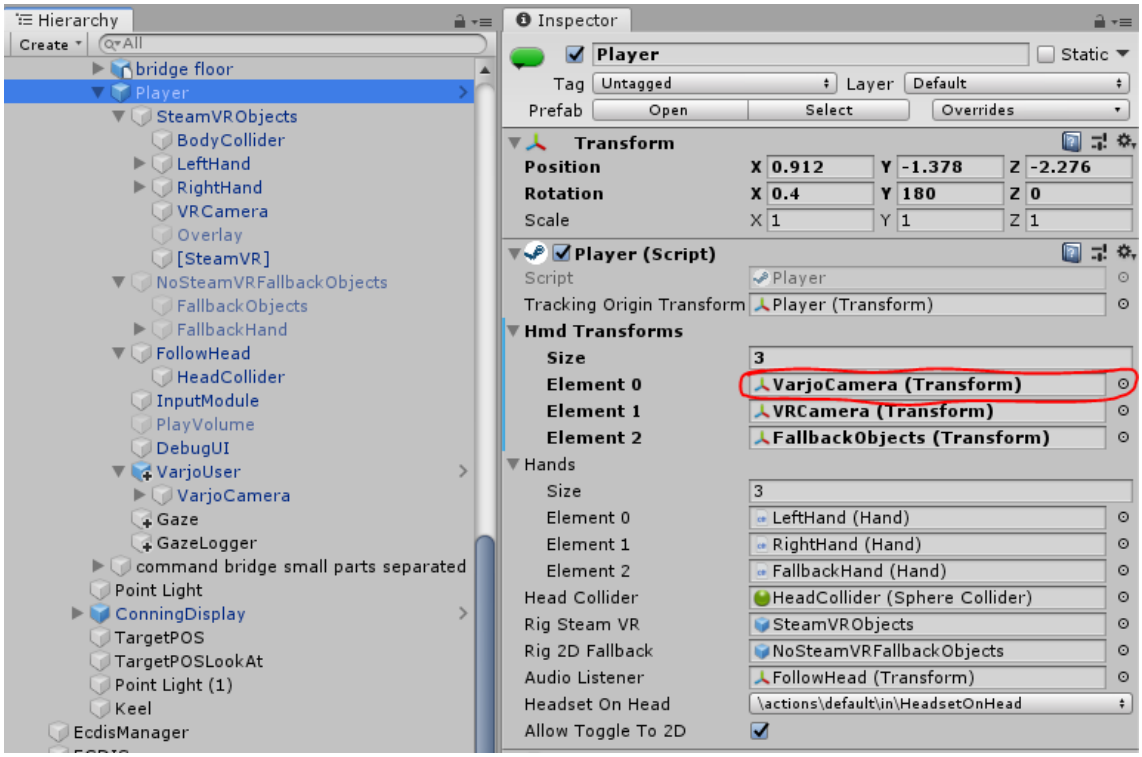
Before starting to work with eye tracking, Unity plugin for Varjo, that enables rendering to Varjo headsets in Unity, had to be installed in the project. The package also comes with example scenes and very important prefabs that are used in this thesis. The package version used in this thesis is 2.3.0, which at the time of writing has been converted to a “legacy” version since more modern versions are available. The SteamVR plugin for Unity was also needed, but it was already installed in the previous version of MarSEVR. There are many things about the SteamVR that could be talked about, like the way it works as an API for different virtual reality devices, or the way it can be configured inside Unity to handle different interactions. However, in this thesis we’re focusing mostly just on the Varjo-related topics instead of diving deeper into SteamVR.

After bringing in the Varjo package, which was downloaded from the official Varjo downloads page, the scene needed the “VarjoUser”-prefab (Picture 8), provided by the Varjo package.



Picture 8. Screenshot of VarjoUser gameobject in the scene.

VarjoUser is a gameobject with the VarjoManager-script attached to it, and a Camera component attached to it's child object. That Camera component needs to be added to the "Player"-script in the "Player" gameobject in the scene (Picture 9), which is actually a prefab from the SteamVR plugin. This is done so that SteamVR knows to follow the correct camera in the scene.



Picture 9. Camera component from Varjo gameobject added to Player gameobject.

Now our scene should render to the Varjo headset and it's time to implement eye tracking.

4.2 Implementing Eye Tracking

Goal of eye tracking implementation is to get the data you see in Table 1 and Table 2. Before going into implementation, a few words about the Gaze data format. What is shown in the Table 1 is pretty much the generic gaze data from Varjo headsets. The first three columns measure time by frames, current real time and time since logging started. Next two columns record the headset position and rotation. After that, there is a lot of information about the eyes, like statuses indicating whether or not the eyes could be properly tracked. For example, the eyes can't be tracked while blinking and in those situations the status column would record "INVALID". We also get information about pupil sizes, gaze positions and gaze forwards. Gaze forward is a vector that starts from gaze position and is directed towards where the eye is looking at. It should be pointed out that all coordinate information about the gaze is presented in left-hand coordinate system. Last two columns we have are "FocusDistance" and "FocusStability". Focus

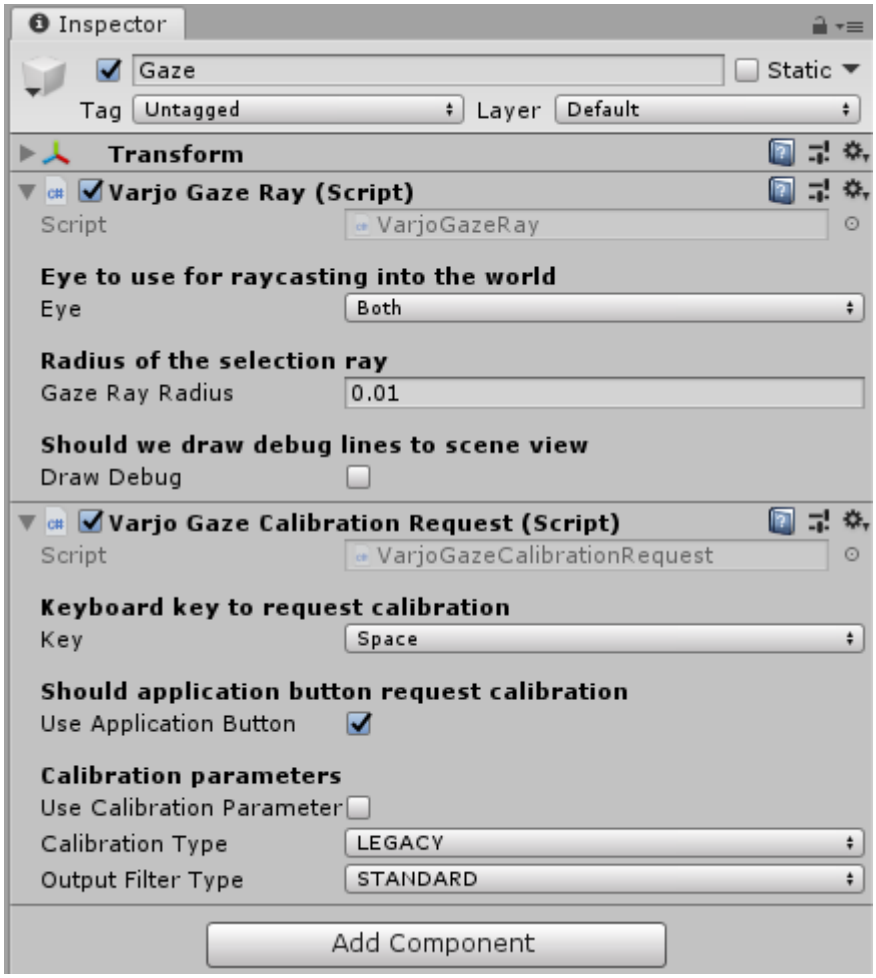
distance is a value between zero and two meters while stability is a decimal value between zero and one, the latter meaning that focus is stable.

Table 1. Eye tracking data from GazeLogger.

A	B	C	D	E	F	G	H	I
Frame	CaptureTime	LogTime(Seconds)	HMDPosition	HMDRotation	GazeStatus	CombinedGazeForward	CombinedGazePosition	LeftEyeStatus
9373	1,00129E+18	0	(-1024.668, 52.716, 1470.639)	(23.682, 135.328, 356.013)	VALID	0.0388509668409824, 0.00759701291099191, 0.999216139316559	0, 0, 0	VALID
9375	1,00129E+18	0	(-1024.668, 52.716, 1470.639)	(23.729, 135.369, 356.022)	VALID	0.0387067943811417, 0.0077048004604876, 0.999220907688141	0, 0, 0	VALID
9377	1,00129E+18	0	(-1024.668, 52.716, 1470.639)	(23.837, 135.363, 356.031)	VALID	0.0384908579289913, 0.00788727216422558, 0.999227821826935	0, 0, 0	VALID

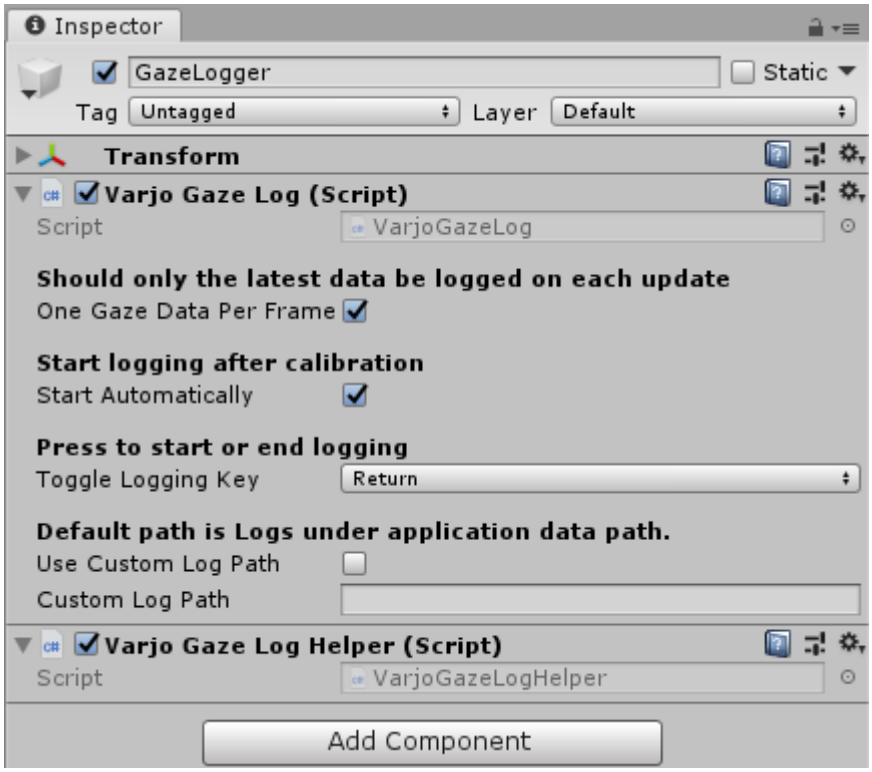
J	K	L	M	N	O	P	Q	R
LeftEyeForward	LeftEyePosition	LeftEyePupilSize	RightEyeStatus	RightEyeForward	RightEyePosition	RightEyePupilSize	FocusDistance	FocusStability
0.027602646499872	-0.0318530015647411, 0, 0	0.1498089283704	VALID	0.0500933602452278, 0.00249429745599627, 0.998	0.0318530015647411,	0.179402694106102	2	1
0.027298878878355	-0.0318530015647411, 0, 0	0.1499307900667	VALID	0.0501086972653866, 0.00269305985420942, 0.998	0.0318530015647411,	0.180709153413773	2	1
0.026791874319315	-0.0318530015647411, 0, 0	0.1499105691909	VALID	0.0501835756003857, 0.00280368770472705, 0.998	0.0318530015647411,	0.181379839777946	2	1

In order to get the data we want we need to start by creating 2 game objects under the “Player”-object, “Gaze” (picture 10) and “GazeLogger” (picture 11). Gaze is going to need two components from the Varjo package called “Varjo Gaze Ray” and “Varjo Gaze Calibration Request”. “Varjo Gaze Calibration Request” allows the user to calibrate eye tracking inside the Unity application, instead of Varjo Base. “Varjo Gaze Ray” is responsible for using the Unity physics system for spherecasting into the virtual environment. In order to create the spherecast, it needs a point of origin (the eyes) and a direction vector to know where to cast. It gets these parameters from the Varjo plugin. Spherecast-function returns detailed information if it hits objects in the scene, and we are using it to log gaze data to a csv-file for user behaviour analysis, more about this later. There are few customization options already available, like which eye origin , or both, to use for spherecasting, how big the ray should be and which method to use for calibrating the eyes.



Picture 10. Gaze object with Varjo components.

Let's talk about GazeLogger next. It has only one component by default, "Varjo Gaze Log"-script. It's main task is to handle logging the eye tracking data and storing it to a location in the file system. Most of the customisation options which are immediately available are pretty self-explanatory, although the "Should only the latest data be logged on each update" is pretty interesting. Varjo VR-2 Pro has gaze tracking frequency of 100 Hz, which is usually different from the frames per second in Unity and you can choose to log data based on frames per second inside Unity or the internal tracking frequenzy of Varjo HMD.



Picture 11. GazeLogger object with Varjo gaze log and custom helper script.

Last thing we need to do in order to make use of Varjo eye tracking, is to add “Varjo Gaze Target”-component to all objects in the scene we want to make trackable. The script has “OnHit”-method that gets triggered by a spherecast from the “Varjo Gaze Ray”-component in Gaze-object. This method is sending data about the gaze target to “Varjo Gaze Log Helper”.

“Varjo Gaze Log Helper” is a custom script to produce different eye tracking data (Table 2). It’s a simple script that generates a list of gameobjects that have been looked at by the user, counts the framecount of the gazes and finally calculates the percentage of time in frames relative to the total framecount during time played. However, the fact that the framerate inside Unity may fluctuate means that counting frames isn’t necessarily counting real time since the time between frames might not be constant.

Table 2. Gaze data from Varjo Gaze Log Helper.

A	B	C
ObjectName	FramesCount	Percentage
backwindows	401	9.130237
rightwindows	103	2.345173
Gas_Levers	245	5.578324
left_gas_lever	1	0.02276867
wheel1	481	10.95173
rear_side_engines_stick	67	1.525501
front_side_engines_stick	132	3.005465
autopilot_l_top_part	52	1.183971
autopilot_r_top_part	124	2.823315
right_gas_lever	13	0.2959927
frontwindows	318	7.240437
OverheadPanel	95	2.163024
leftwindows	148	3.369763

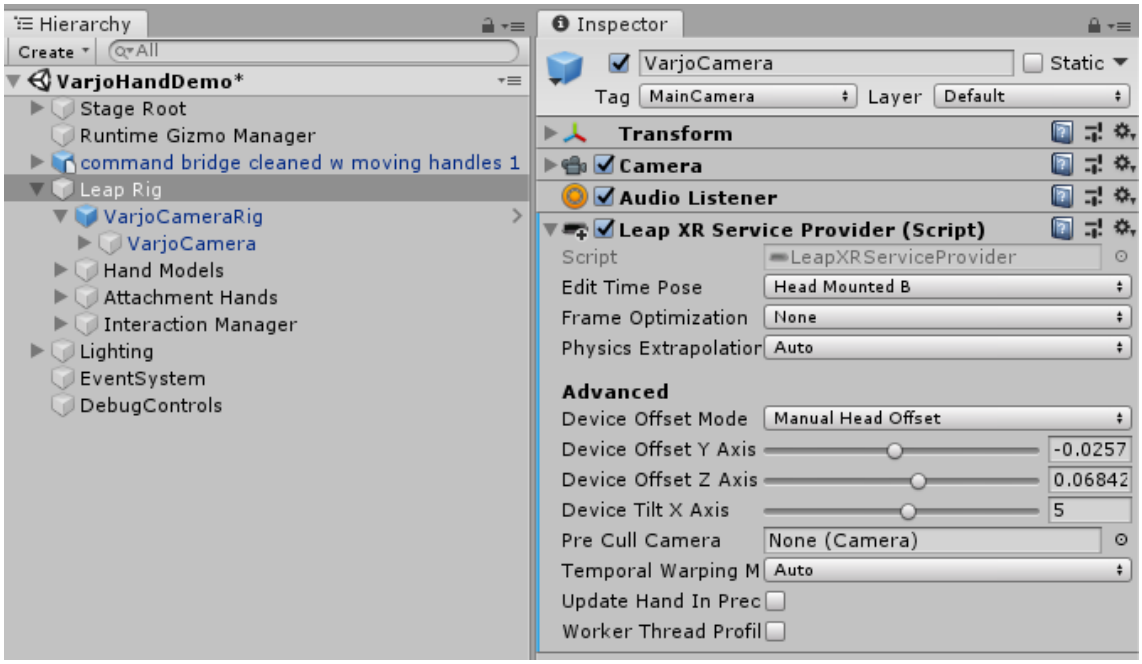
4.3 Implementing Hand Tracking

The main purpose of this thesis was to study and implement eye tracking into MarSEVR application. That being said, a small scene was created to demonstrate some of the hand tracking capabilities that Varjo VR-2 Pro has. In the scene we have a stripped down version of the actual command bridge used with eye tracking. For example all the radars are left blank because they would have been excess considering the purpose of this demo scene. Only a few objects are actually made interactable, namely the steering wheel, the gas levers and engine levers (Picture 12).

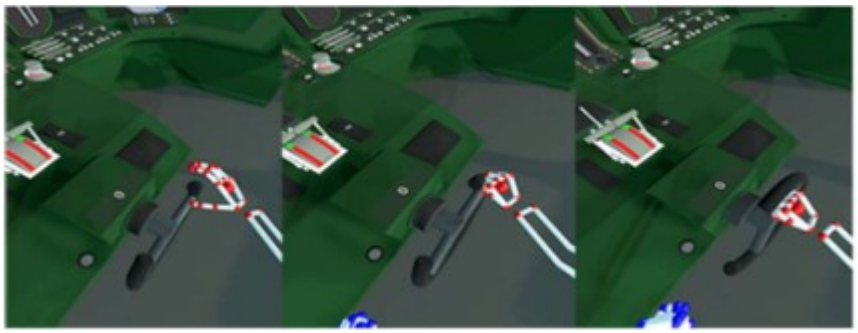
In this thesis we don't talk about the hand tracking implementation in detail. However, few things worthy of note: Varjo has been collaborating with UltraLeap, a company behind Leapmotion hand tracking software. The hand tracking sensors have been integrated into the Varjo HMD, and all we needed to do was to download a Leapmotion developer kit and a Leapmotion package for Unity. Both the developer kit and Unity package have a Varjo version that has been specifically tailored to fit the Varjo HMDs needs.

Picture 12. View inside Hand tracking Demo scene.

After installing the developer kit and importing the Leapmotion package to Unity, all we needed to do was use the “Leap Rig” prefab provided by the Leapmotion package. Leap Rig has the basic Varjo Camera child object, although we are not using SteamVR this time, so we don’t need to have the “Player”-object like we did in eye tracking. VarjoCamera has one major addition, the “Leap XR Service Provider”-component (Picture 13). This component is responsible for fetching the hand tracking data from sensors inside the Varjo HMD. “Hand Models” is parent object for all the different types of models you want to use as hands in the scene. “Attachment Hands” are not hand models, but they provide joint transforms and interaction points that are used by the hand models. “Interaction Manager” handles the way you can interact with objects in the scene. By default Leapmotion allows you to interact with objects by grasping, hovering and contacting (Picture 14).



Picture 13. LeapXRServiceProvider attached to VarjoCamera, child object of Leap Rig.



Picture 14. Rotating the wheel with hand tracking.

5 RESULTS AND DISCUSSION

During the project development phase, testing was left to a minimum. The author worked on the project alone and no outside testers were utilized. Most of the testing done by the author was about finding best ways to gather eye tracking data and later to determine whether or not hand tracking was usable in the MarSEVR -application. After testing Varjo hand tracking solutions the author decided that implementing hand tracking would have required remaking the entire ship physics system, which would have been too much work for this thesis which prioritizes eye tracking. Therefore the author created another scene in Unity to demonstrate some of the hand tracking features.

A VR development professional who had previous experience with the original project was recruited to test the new version after it was done. The professional gave feedback about the author's eye tracking implementation and data collection. Some of the feedback was about the code readability, which the author feels like could have been solved with more focus on polishing the project. However the issues with code readability were not directly related to performance and were therefore pushed aside. The most important feedback was about the logging process. In the project eye tracking data is captured and written to a file every frame and the way "Varjo Gaze Log Helper" gathers info about the gazed gameobjects is by using the Unity physics system. In the feedback there were some concerns about the performance aspect when this logging process happens every single frame, which means with Varjo VR-2 Pro around 90 times per second (or atleast that's the desirable rate). Proposed solutions to solve this problem were to either reduce the logging rate or implement a multithreaded system where logging is done on another CPU thread. The author thinks that the logging rate could be easily reduced, but without having a clear research goal in terms of what the eye tracking data is gathered for, reducing the logging rate might be counter-productive and at this point unnecessary. When it comes to multithreading we run into some problems. For example, the StreamWriter -class we use for data logging is not thread safe [19]. This means that multiple threads are able to write to the same location in memory at the same time [20], in this case our logging file. This could lead to unexpected results and bugs. All this being said, the author agrees with the test results that multithreading could be a useful feature and it should be added to the project and tested in the future.

6 CONCLUSION

In this chapter the author of this thesis discusses his feelings and opinions about the work that was carried out in this thesis.

Before working on MarSEVR project in this thesis work, the author had no experience whatsoever with eye or hand tracking in virtual environment, nor the Varjo equipment. Much research was conducted about how the Varjo headset works and what it needs in order to be utilized in Unity game engine. This took probably the most amount of time, compared to implementing the eye and hand tracking features, or doing general research about the eye tracking techniques. Many mistakes were made and time was wasted. However, the author learnt many new things about virtual reality development that are very likely going to be useful in the future, so overall the working experience was quite positive.

Much more could have been done by the author, especially performance testing and more focus on the user experience. When it comes to user experience though, for example developing advanced user interfaces and giving detailed instructions to the user, the author felt that since this work is more of a stepping stone instead of the final product, focusing on polishing the application would have been unnecessary.

As mentioned in the introduction, this work was never thought to be the finished product. On the contrary, implementing Varjo features to the MarSEVR-project is a starting point for further research that is already taking place. The eye tracking implementation provides a solid base for research in cognitive neuroscience and it will probably be used in future MarISOT projects [17]. The work carried out in this thesis was also featured in two IEEE conference papers titled *Eye Tracking in Maritime Immersive Safe Oceans Technology* [17] and *Finger tracking and hand recognition technologies in virtual reality maritime safety training applications* [18], presented at the IEEE International Conference on Cognitive Infocommunications, 2020.

REFERENCES

[1] Thompson, S., VR for Corporate Training: Examples of VR already being used. [Cited May 2021]

https://virtualspeech.com/blog/how-is-vr-changing-corporate-training

[2] SR Research: about Eye Tracking [Cited May 2021]

https://www.sr-research.com/about-eye-tracking/

[3] Farnsworth, B., Eye Tracking: The Complete Pocket Guide, 2018 [Cited May 2021]

https://imotions.com/blog/eye-tracking/

[4] Wikipedia: Cornea, https://en.wikipedia.org/wiki/Cornea [Cited May 2021]

[5] Tobii, Dark and Bright Pupil Tracking [Cited May 2021]

https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/what-is-dark-and-bright-pupil-tracking/

[6] Farnsworth, B., 10 Most Used Eye Tracking Metrics and Terms, 2020 [Cited May 2021] https://imotions.com/blog/10-terms-metrics-eye-tracking/

[7] Wikipedia: Cognitive Neuroscience [Cited June 2021]

https://en.wikipedia.org/wiki/Cognitive_neuroscience

[8] Tobii: Applications for eye tracking [Cited June 2021]

https://www.tobiipro.com/applications/

[9] Smart VR Lab Blog, Hand Tracking in VR [Cited June 2021]

https://www.smartvrlab.nl/hand-tracking-in-vr/

[10] Wikipedia: Power Glove [Cited May 2021]

https://en.wikipedia.org/wiki/Power_Glove

[11] UltraLeap, 2021, What is Hand Tracking in VR? [Cited June 2021] https://www.ultraleap.com/company/news/blog/hand-tracking-in-vr/

[12] VIAR, The Role of Eye-Tracking and Foveated Rendering in VR Learning [Cited June 2021] https://www.viar360.com/the-role-of-eye-tracking-and-foveated-rendering-in-vr-learning/

[13] Bhonde, S., Easy VRS Integration with Eye Tracking, 2019 [Cited June 2021] https://developer.nvidia.com/blog/vrs-wrapper/

[14] Tobii: Tobii Spotlight Technology [Cited June 2021]

https://vr.tobii.com/wp-content/uploads/2019/07/Tobii-Spotlight-Fact-Sheet.pdf

[15] Eye tracker accuracy and precision [Cited June 2021]

https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/what-affects-the-accuracy-and-precision-of-an-eye-tracker/

[16] Farnsworth, B., The Future of Therapy – VR and Biosensors, 2019 [Cited June 2021] https://imotions.com/blog/vr-therapy/

[17] Luimula, M., Markopoulos, E., Kaakinen, J., Markopoulos, P., Laivuori, N. and Ravyse, W. (2020). *Eye Tracking in Maritime Immersive Safe Oceans Technology.* [Cited June 2021]

https://www.researchgate.net/publication/344379901_Eye_Tracking_in_Maritime_Immersive_Safe_Oceans_Technology

[18] Markopoulos, E., Markopoulos, P., Laivuori, N., Moridis, K., Luimula, M. (2020). Finger tracking and hand recognition technologies in virtual reality maritime safety training applications. [Cited June 2021]

https://www.researchgate.net/publication/344380036_Finger_tracking_and_hand_recognition_technologies_in_virtual_reality_maritime_safety_training_applications

[19] Microsoft docs: StreamWriter Class [Cited June 2021]

https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-5.0

[20] Ershad, G., Thread Safety in C#, 2015 [Cited June 2021]

https://www.c-sharpcorner.com/UploadFile/1c8574/thread-safety369/