



Bachelorthesis

*Leveraging SLAM to bring Positional Tracking to
phone-based VR*

Christoph Geske



Examiner

Dr.-Ing. Sven Wachsmuth, Bielefeld University

Dr. rer. nat. Thies Pfeiffer, Bielefeld University

*Bielefeld University
Faculty of Technology
Bioinformatics and Genome Research*

31.10.2019

STATUTORY DECLARATION

I Christoph Geske declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Bielefeld, 31. October 2019

place, date

signature

Abstract

This thesis examines the latest developments in phone-based virtual- and augmented reality (VR/AR) and investigates how these technologies can be merged to enhance current state-of-the-art phone-based VR. An overview over the available hard- and software will be given, and limitations and chances discussed. Experiments measuring different quality characteristics of the required hard- and software will be presented. The focus of this work lies in the detailed investigation of the available tracking software and in the filtering process for improving the tracking quality. Various approaches used to validate and measure the tracking software and the developed filtering methods will be described and compared. The latest research on human perception is presented and the Importance of this thesis and future developments in this area emphasized. The limitations of the presented tracking software will be described, and future research directions discussed in the final chapter. Additionally, the outlook will give an impression on how releasing the here presented tracking software can benefit the open source community and how it influenced this thesis.

Contents

Abstract	II
List of Abbreviations	V
List of Figures	VI
1 Introduction	1
1.1 Motivation	1
1.2 Goals	2
2 Virtual-, Augmented- and Mixed Reality	4
2.1 Virtual Reality (VR)	4
2.2 Augmented Reality (AR)	5
2.3 Mixed Reality (MR)	5
3 Hardware Technology	6
3.1 Head Mounted Displays (HMDs)	6
3.1.1 Phone-based VR HMDs	7
3.1.2 Stand-Alone HMDs	10
3.2 Sensors	10
4 Software Technology	12
4.1 Camera Tracking	12
4.1.1 Visual Odometry (VO), Visual Inertial Odometry (VIO)	12
4.1.2 Simultaneous localization and mapping (SLAM)	12
4.2 Software Development Kit's (SDKs)	12
4.2.1 Virtual Reality SDKs	12
4.2.2 Augmented Reality SDKs	13
4.3 Development Platform	15
5 Perception and Sensation	16
5.1 Latency and how it is perceived	16
5.2 Immersion	16
5.3 Motion sickness	18
6 Head Tracking	19
6.1 Leveraging ARCore for Positional Head Tracking	19
6.2 Evaluation Of Tracking Latency	29

6.3	Interpolation on tracking data to improve visuals	32
6.4	Tracking Accuracy and quantification	33
7	Filtering	36
7.1	Hard Coded Filter	36
7.2	Kalman Filter	36
8	Outlook	37
	References	38

List of Abbreviations

- **HMD** head mounted display
- **IMU** Inertial Measurement Unit, in general made up of accelerometer, magnetometer and a gyroscope
- **RGB camera** red-green-blue camera
- **SDK** .. Software Development Kit, collection of software and libraries necessary for developing an application.
- **SLAM** Simultaneous Localisation And Mapping
- **VIO** Visual-Inertial Odometry
- **VO** Visual Odometry
- **3DoF** Three Degrees of Freedom
- **6DoF** Six Degrees of Freedom

List of Figures

1	Reality–Virtuality Continuum. Reprinted from “Augmented reality: A class of displays on the reality-virtuality continuum.” by Milgram et al. (1995)	4
2	Overview phone-based VR and stand-alone VR HMDs	7
3	Comparing ARCore ARKit and Tango AR SDKs. Reprinted from “AD-VIO: An Authentic Dataset for Visual-Inertial Odometry” by Cortés et al. (2019)	14
4	Effects of latency on sense of presence. Reprinted from “Effects of Head-Display Lag on Presence in the Oculus Rift.” by Kim et al. (2018)	17
5	Effects of latency on perceived scene instability. Reprinted from “Effects of Head-Display Lag on Presence in the Oculus Rift.” by Kim et al. (2018)	17
6	Position data in x direction of a Galaxy S7 fixed to a skateboard. Curve shows deceleration over roughly 4 meters.	20
7	Using interpolation leads to a more uniform distribution of the tracking datapoints. Also, visible occasional discrepancies in the curve most likely caused by ARCore internal loop closer and drift correction algorithms. .	21
8	Screen capture recording running in parallel to the phone-based VR HMD tracking its position in space. Running both processes in parallel required more processing power than the phone can deliver resulting in the tracking not being able to update the position at 60fps.	22
9	Running an app that combines the VR functionalities of the Google VR SDK with the positional tracking functionalities of the ARCore SDK. . .	23
10	Visible the excellent performance when using the “HelloAR sample app” and no VR mode is enabled. Using Unity 2018.2.20.	23
11	Performance measurements of the GearVR and ARCore setup in the first half and the ARCore and DIYVRCam in the second half. The comparison shows the impact the “EarlyUpdate.TangoUpdate” function has on the performance.	25
12	Vuforia powered positional tracking VR app running on the Galaxy S7 showing acceptable performance characteristics.	26
13	The graph shows the x position values of an app build using the Vuforia AR SDK. The phone is attached to a pendulum. The curve describing the pendulum motion shows significantly more jitter compared to curves recorded with ARCore. The sinewave describes the ground truth of the pendulum movement.	26

14	Performance measurement of the combination ARCore + GearVR running on the Note 8.	27
15	Benchmark comparing performance of Note 8 and S7 [23]	28
16	Unity profiler showing the performance of a GearVR + ARCore app and in blue the performance impact of multiple Debug Log processes on the performance of the app and especially on the effect the “EarlyUpdate.TangoUpdate” process has on the overall performance. The Debug Log processes are not activated at the start of the app but instead after a few seconds into the app running.	29
17	Schematic experimental setup to measure the latency of the ARCore tracking.	30
18	Using the VLC media players and its single frame-by-frame function to go through every frame. As soon as the phone starts to move the camera frames are counted and noted and the number on the screen which describes the current position is listed. The frames it takes the tracking software to notice a change in its position can then be used to determine the system latency of the ARCore tracking software.	31
19	List of the position values displayed on the phone screen at 60fps. The first column shows the distance from the starting point which is displayed on the phone’s screen. The second row shows the number of frames that have passed since the beginning of the measurement. The third column shows if the phone is moving and in which direction. The fourth column calculates the distance from the previous position. The fifth column uses colors to indicate areas of speeding up and slowing down either towards the right (green) or the left (red) side. The sixth column indicates when the tracking software recognizes the change in position either stating, stopping or reversing the direction. The seventh column shows the number of frames it takes from the high-speed camera recognizing a change in position to the tracking software displaying a change in position. . . .	32
20	Schematic how interpolation can be used to generate a 60fps tracking solution from tracking data that only updates with a rate of 30fps. . . .	33
21	Schematic on how the interpolation introduces lag by not immediately applying the new position data.	33
22	Schematic on how the linear extrapolation could be used to generate a 60fps tracking solution without lag but possibly introducing erroneous position data.	34

23	The 3 dimensional positional tracking data of the Vive tracker and the phone running the ARCore tracking solution where transformed into a 2D representation and plotted on top of each other. This 2D representation was generated by calculating the vector length between adjacent points and the length was plotted without scaling or compressing.	35
24	The positional tracking data of the Vive Tracker and the phone where plotted on top of each other after the transformation, scaling and compressing steps where performed.	35
25	GitHub page showing an already released version of the tracking software presented in this thesis.	38

1 Introduction

Within the last decade, driven by Moore's law, high demand and the yearly release cycle of phone manufacturers, phones improved significantly in the area of processing power, display technology and sensor fidelity. The accumulation of all these developments made it possible to use regular smartphones for virtual reality (VR) applications (apps) using only the built-in sensors, software and a headset with two optical lenses. The sensors in the first smartphones were not chosen with the requirement of responsive head tracking for VR apps in mind, but with better sensors becoming available in regular phones dedicated phone-based VR came to market able to track the rotation of the head very precisely. In recent years companies which were leaders in the field of phone-based VR shifted their focus away from these systems to developing dedicated stand-alone hardware optimized for VR. At the same time phones continued to evolve and computer vision software got more efficient. Especially in the development of augmented reality the software made great strides allowing phones to leverage the camera to get an understanding of the environment, calculate their own position in space and extend the camera image with virtual objects. First proof of principle projects showing that the AR platforms ARKit and ARCore can be leveraged to extend phone-based VR with positional tracking were released in 2017. All these projects rely solely on the tracking capabilities provided by the AR platforms and do not seek to improve the quality of the tracking. In this thesis the feasibility of extending phone-based VR with positional 6 degrees of freedom (6DoF) head tracking by using current AR software tools which only rely on a single red-green-blue (RGB) camera for tracking will be evaluated. The latest literature will be reviewed to determine which characteristics a VR positional tracking system must fulfil and how tracking errors and latency affect the user's experience. Implementing a phone-based VR HMD, extending it about a positional tracking component and quantifying the result with different measurement approaches to determine what such a system can offer and where its weaknesses lie will be some of the main goals of this thesis.

1.1 Motivation

Modern smartphones have the necessary hardware for delivering good quality 3 degrees of freedom (3DoF) VR experiences and with the technology of today's high-end phones quickly finding its way into cheaper models which are sold into every corner of the globe even more people will have access to phones which are able to deliver a good VR experience. Dedicated stand-alone hardware which is optimized for VR purposes can deliver a much better overall performance when compared to phone-based VR,

even when similar hardware is used but the advantage that only a cheap headset is needed provided the phone is already available makes this technology still relevant. With millions of headsets already in consumer's, educator's and creator's hands, offering a 6DoF tracking which allows for more immersive interactions with the technology without the need to purchase new hardware seems like a goal worth pursuing. The lack of 6DoF head and hand tracking is one of the biggest drawbacks of current phone-based VR compared to standalone and wired HMDs and offering solutions which don't require additional hardware would be significant. Offering a low-cost open source VR headset as an alternative to expensive commercially available stand-alone VR devices can also be seen as a great equalizer allowing more people especially with a low income to have access to a more immersive VR technology. Amin et al. (2016) see in phone-based VR the potential to be used in the healthcare sector as cost effective pain relieve for domestic use by many patients and by enhancing the immersion of the technology the effectiveness will increase as well according to Parsons et al. (2009). Since the main phone manufacturers stopped offering new phone models supporting phone-based VR platforms other than Cardboard VR developing new tracking solutions for these already well established platforms together with help of the open source community can perhaps make a contribution in moving the field forward and offer a free open source alternative to currently available stand-alone VR. By combining multiple open source modules for VR and positional tracking as well as pointing to even more modules for hand, eye and body tracking, this project gives students and developers great flexibility to extend and modify the presented software and develop their own VR projects with it. Part of the code might also be used in areas different from VR like robotics or the related field of augmented reality (AR). Offering an open source project as an alternative to the proprietary hardware and software solution available on the market might be interesting for some users concerned about their privacy or companies working with sensible data interested in an open source code base.

1.2 Goals

The goal of this thesis is to explore the possibilities and limitations of the latest AR tracking technology for bringing positional tracking to phone-based VR. Making available a comprehensive list of the requirements that a positional tracking system for a VR HMD needs to fulfil will help with the process of testing current AR tracking technologies for their suitability in use as a tracking solution for phone-based VR. Furthermore, knowing these will benefit future research concerned with further improving the here presented software solutions. By concentrating on the most widely distributed VR headsets more precisely Google Cardboard, Daydream View and GearVR the goal is to give as many

developers as possible access to a cost-effective VR HMD with 6DoF positional tracking. Even when hardware and software companies now offer good stand-alone HMDs millions of capable phone-based VR which would benefit greatly from 6DoF positional tracking exist in schools, universities, companies and households or can be purchased for a low price on the secondhand marketplace. Significantly extending the capabilities of current phone-based VR might also prolong the lifetime of these systems having a financial and environmental benefit. With phones constantly improving especially in the area of positional tracking and environmental understanding for AR applications researching possible solutions now with current hardware might help with developing appealing software solutions faster in the future when the hardware has further improved.

2 Virtual-, Augmented- and Mixed Reality

When describing technology that allows users to immerse themselves into a computer generated virtual world or enhance the real world with computer generated content it is useful to look at the reality–virtuality continuum Figure 1 proposed by Milgram et al. (1995) which illustrates how the technologies are interconnected and merge seamlessly. In recent years the term extended reality (xR) is often used to describe all three terms whereby the letter x is a place holder for the initial letters (V, A and M).

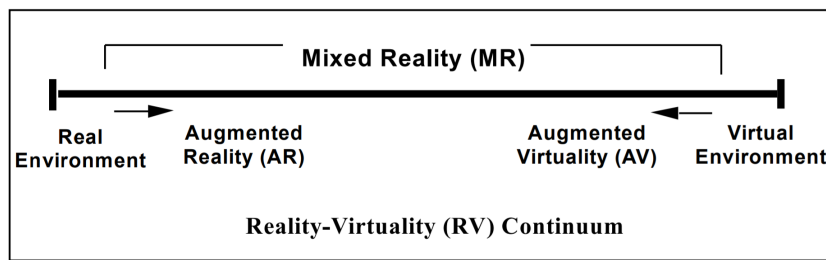


Figure 1: Reality–Virtuality Continuum. Reprinted from “Augmented reality: A class of displays on the reality-virtuality continuum.” by Milgram et al. (1995)

2.1 Virtual Reality (VR)

The definitions for VR differ from one source to another but one common aspect is that a simulated environment is first generated with the help of a computer and the user is then enabled to be immersed in and engage with that simulated environment. Vision is arguably the most important sense affected by VR technology and some definitions rely exclusively on visual stimulation when describing VR: “Computer graphics technology called virtual reality (VR) creates a visual scene, in which the user feels immersed. As the user makes active head movements, the computer determines the new direction of gaze and recreates the scene from the new point of view.” (Akizuki et al.) “VR occurs when computer generated data completely obscures the real world.” (Nowacki et al. (2019)) “Virtual Reality (VR) – defined as three-dimensional, stereoscopic, interactive computer graphics – is a computer-generated environment that can simulate physical presence in virtual worlds by engaging human sensory experiences.” (Amin et al.) Other sources have an even narrower definition requiring multiple senses to be stimulated to speak of VR: “Virtual reality (VR) enables interactions that satisfy the five senses of users—particularly the visual, auditory, and tactile senses—in order to provide an experience similar to reality.” (Lee et al. (2017)) In their review article Roche et al. (2019) point out that a strict consensus regarding the definition of Virtual Reality (VR) is missing and give the following definition: “VR has been defined as a form of technology

that permits advanced, dynamic interaction between humans and a computer interface. VR is often additionally described as being immersive and/or interactive.” whereby the terms ‘immersive’ and ‘interactive’ have various meanings throughout the literature and we will explore what immersiveness means and how it is effected by technology in the chapter about Perceptions and Sensation. All current phone-based VR HMDs can be placed on the right side on the reality–virtuality continuum since they only use internal sensor information for head tracking and do not make use of information of the environment to augment the virtual scene.

2.2 Augmented Reality (AR)

AR in comparison to VR profits from a good understanding of the environment for tasks like placing virtual objects into it and make them appear to be part of the real world. Despite an understanding of the environment not being necessary for VR, a VR experience can benefit from the advancements made in AR technology since both technologies require to react to the movements of the user and both VR and AR have come up with different tracking solutions. According to Arth et al. (2015) the history of mobile AR technology can be traced way back to 1989 but tracking technologies like visual-inertial odometry (VIO) and Simultaneous Localization and Mapping (SLAM) running on mobile phones only started to be possible in recent history with technologies like “PointCloud” the first commercially available SLAM system to run on a mobile phone being released in 2011.

2.3 Mixed Reality (MR)

“Mixed Reality can be defined as both the real and virtual are mixed, where the virtual augments the real and the real augments the virtual.” (Fast-Berglund et al. (2018)) When adopting the obstacle detection, object classification or hand tracking functionalities of an AR software to augment the content displayed in a phone-based VR HMD one would move to the left on the reality-virtuality continuum becoming a Mixed Reality technology. Using more features developed for AR the systems will result in a new term phone-based MR but since this thesis is primarily concerned with adding positional tracking to the phone-based VR HMDs and not augmenting the virtual world with information about the real world the term phone-based VR will be used.

3 Hardware Technology

The advantage of phone-based VR is that no external hardware is required except a headset with two lenses magnifying the display and breaking the light so that a wide field of view is possible.

3.1 Head Mounted Displays (HMDs)

There are many VR HMDs on the market but since this thesis focuses on extending phone-based VR with positional tracking capabilities only the most relevant mobile HMDs which also rely on a mobile processor will be discussed. Anthes et al. (2016) distinguish between mobile and wired HMDs and define mobile HMDs as being wireless and not requiring an additional PC. Anthes et al. (2016) introduces three subcategories for mobile HMDs: “simple casing”, “ergonomically designed” and “stand-alone systems”. The subcategories “simple casing” and “ergonomically designed” both describe HMDs that are assembled from two parts the phone as the display and processing unit and the headset for holding the phone and containing the lenses. The “ergonomically designed” HMDs differ only in terms of better optics and more comfort. The subcategory “stand-alone systems” describes HMDs that have all the hardware build in not requiring an additional phone. However, with the wide range of available headsets to insert the phone into, this sub categorization breaks apart for some HMDs. The Google Cardboard for example can be combined with multiple headsets of varying quality making the experience on some headsets even more ergonomic than on GearVR or Daydream headsets. To distinguish the sub-category “stand-alone systems” from the other two is relevant since these systems are optimized for VR while phone-based VR HMDs are multipurpose computing devices with VR only being one of many possible applications. In this thesis only two subcategories will be used to describe mobile HMDs: “phone-based VR HMDs” and the term “stand-alone systems” introduced by Anthes et al. (2016) will be changed to “stand-alone VR HMDs” to distinguish between VR, AR and MR HMDs. The new subcategory “phone-based VR HMDs” unites the two terms “simple casing” and “ergonomically designed” and describes HMDs that rely on a phone but doesn't categorizes them by the quality of the headset or lenses. A tree structure similar to the one used by Anthes et al. (2016) can give a quick overview over the relevant hardware. An important feature of all VR HMDs is head tracking and two categories are common for mobile VR HMDs: three degrees of freedom (3DoF) head tracking where the environment responds to head rotations and six degrees of freedom (6DoF) head tracking where the virtual environment responds to translational and rotational head movements.

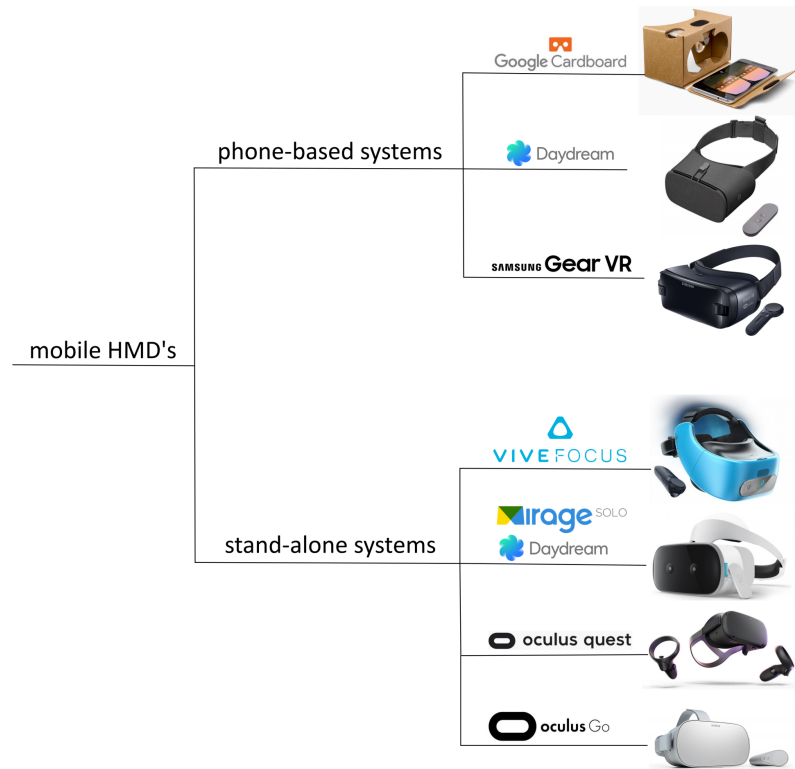


Figure 2: Overview phone-based VR and stand-alone VR HMDs

3.1.1 Phone-based VR HMDs

Phone-based VR HMDs rely on the processing power, the high-resolution screens and built-in sensors of regular smartphones to deliver an immersive experience. The phones are in general not designed with VR in mind however some sensors might have been chosen so that they allow the phone to perform better when used for VR purposes. The biggest advantage of phone-based VR is that many people already own a phone that has all the necessary hardware build in to deliver a compelling 3DoF VR experience and that the cost for a headset to put the phone into is considerably less compared to a complete stand-alone system presuming the user already owns a capable phone. The three phone-based VR HMDs: Google Cardboard, Daydream and GearVR dominate the market and which system a phone supports depends on the hardware, but to a greater extend on the support provided by the platform operators. As an example, the Samsung Galaxy S9 supports all three platforms while an even more powerful successor phone like the Samsung Galaxy Note 10 only supports the Google Cardboard platform. The first widely available phone-based VR headset was the Google Cardboard which was developed by Google engineers in 2014. In 2015 Samsung and Oculus released the GearVR headset which contained a highly accurate Inertial Measurement Unit (IMU) allowing for the precise measurement of acceleration and rotation of the headset. In

2015 Google unveiled the Daydream platform which had no additional IMU build in the headset but only worked in combination with selected phones having a precise and well calibrated IMU sensor. Both platforms came together with special SDKs providing the necessary software allowing developers to create VR apps for use together with these high-quality headsets. Now for the first time consumers with a supported smartphone had the chance to experience VR with low latency 3-degrees of freedom (3DoF) head tracking and a wide field of view under the headset. The headsets from Oculus and Google were more expensive than the Cardboard viewer but with around 100 still much more affordable compared to VR HMDs available for desktop PCs. The improvements in smartphone technology and computer vision software also made it possible to combine the sensor data of an IMU and a camera to track the phone's position in 6DoF with high precision and allow for appealing AR experiences running on millions of phones. In 2016 the first phone running the Tango platform was released which made use of a time of flight depth sensor, an build in IMU and the SLAM technology for determining the device position in 6DoF with high precision. The Tango platform was primarily used for AR applications but Bhandari et al. (2017) used it to develop a cable free phone-based VR HMD with 6DoF head tracking. However, the low number of specialized smartphones supporting the Tango platform led to its termination in 2018 and the focus shifted to only relying on the red-green-blue (RGB) cameras found in most regular phones to obtain depth and positional information of the surrounding for positional tracking. In the last years Google and Apple focused primarily on bringing AR to the masses by relying on the already widely distributed mono cameras in smartphones. Both companies released free to use but closed proprietary positional tracking software based on the SLAM technology called ARCore in 2018 and ARKit in 2017. In October 2017 Roberto Lopez Mendez published a blog post describing the idea of using ARCore in combination with a GearVR headset. Building on Mendez's initial idea this thesis will dive deeper in the technology which is allowing to enhance phone-based VR by adding 6DoF head tracking using ARCore and the underling SLAM technology.

Google Cardboard The Google Cardboard v1 released in 2014 is the oldest phone-based VR HMD available for consumers and by far the most affordable option. It has Cross-platform support for both Android and iOS. For many people the Google Cardboard headset was the first encounter with VR as a new technology and at the time of this thesis the Play Store shows 10,000,000 – 50,000,000 downloads of the associated Google Cardboard app. Cutting the cardboard yourself and ordering plastic lenses online for less than 1 per pair is the cheapest option to build a phone-based VR HMD when a smartphone with the required hardware is already available. The small lenses

on the first Google Cardboard version v1 were not ideal and the later version Google Cardboard v2 improved on the input method and lenses. But despite the improvements using Google Cardboard v2 for prolonged VR sessions is not practical.

Many other headset designs and solutions are available online in all price ranges and most of them are very cheap with 13 of 15 of the first smartphone holding headsets on amazon being priced below 30. The Google Cardboard app only has very low requirements on the phone (Android 4.4 'KitKat' or iOS versions 8.0) and should run on 96% of all Android and 99% of all iPhones available today. But even if a phone is capable of running a Cardboard app the gyroscope in older phones were not chosen with fast response time or high accuracy in mind [24].

With release of the Google VR SDK the Cardboard SDK was combined with the Daydream SDK improving the quality of Cardboard experiences for Daydream ready phones.

Google Daydream The Google Daydream platform was released in 2016 and the View 1 headset was the first headset meant to be used in combination with a Daydream ready phone. The Daydream platform only supports phones that fulfil specific requirements mainly in the area of chipset, display and sensors. The phone recognizes the headset with a near field communication (NFC) chip installed in the lid of the headset and no physical connection with the headset is required making the setup very simple. A noticeable disadvantage of the Daydream View headset is that the front lid obscures the camera and users interested in making use of the 6-dof tracking solution presented in this thesis either need to permanently modify the Daydream Viewer or build their own solution to keep the phone in place without the lid covering the camera.

GearVR The GearVR was developed in a partnership between Oculus and Samsung and released in 2015. The GearVR only supports a subset of Samsung phones, is available in different sizes to optimize the fit and might require an adapter because of different connector. Since the headset has no cover which could obscures the camera it is better suited for developing a camera based 6DoF tracking solution when compared to the Daydream headset. Publishing software on the GearVR store is also possible when some conditions for example running at 60 fps and for at least 45 minutes without overheating are fulfilled. Apps using similar camera-based tracking solutions as the one presented in this thesis can already be found in the GearVR store.

3.1.2 Stand-Alone HMDs

Stand-alone headsets impress with great performance on limited hardware. By having more control offer the hard and software manufacturers can achieve better results than phone-based VR HMDs for example the Oculus Go uses the same processor as the S7 used in this thesis but can offer higher frame rates and better performance partly because of optimized software and partly because the hardware is better suited for removing heat from the CPU compared to a regular phone. Stand-alone VR HMDs allow manufacturers to install sensors and hardware components that are perfectly suited for VR applications reducing costs by avoiding unnecessary components and improving performance by matching the components to each other.

3.2 Sensors

The inertial measurement unit (IMU) and the camera are the two sensors that play a role in this project and will be described briefly. An IMU found in current smartphones consists of an accelerometer, a gyroscope and a magnetometer. In phones they are primarily used for:

- making changes to the user interface e.g. reacting to changes in the phone's orientation
- gaming
- image stabilization
- navigation
- augmented reality
- virtual reality

Accelerometer The accelerometer measures forces acting on masses. This way the accelerometer can measure gravity and forces resulting from accelerating the phone. The accelerometer can however not distinguish between the gravitational force and forces created by linear acceleration. On the physical level a phone accelerometer is comprised of a semiconductor material that form beams that flex when forces are applied. The flex changes the distance of capacitor plate to one another which can be precisely measured. <https://patents.google.com/patent/US4736629A/en> Since earth's gravity stays constant this sensor does not suffer from drift over long periods of time.

Magnetometer The magnetometer can detect magnetic fields but can not distinguish between magnetic fields created by earth's iron core or other electric devices in or near the phone. In general, 3-axis magnetometer are used that can measure the magnetic field independent on how the phones orientation is. Together with the accelerometere measuring gravity it can be determined where the earth's surface is and which of the 3 magnetometers needs to be used. On a physical level the magnetometer work by current being deflected from the interaction with magnetic fields through the so called Hall-Effect. Since earth's magnetic field stays constant this sensor does not suffer from drift over long periods of time.

Gyroscope Gyroscope measure angular velocity which describe how fast an object is rotating. Angular velocity is thereby defined by the rate of change of the angular rotation over change in time. On a physical level they often work by an oscillating mass which is affected by changes in orientation and the movement is converted into an electric signal that can be amplified and measured. Triple axis gyroscope able to measure change in all three directions are commonly found in current consumer electronics. Important to note is that acceleration or linear velocity do not affect the measurements of the gyroscope which only measures angular velocity. Since the absolute angular position is calculated via integration from the angular velocity the gyroscope tends to drift. By combining the gyroscope with the accelerometere or magnetometer which are stable over longer time periods the drift can be reduced in a process called sensor fusion. The camera which does not tend to drift can also be used to correct for drift that accumulates when integrating IMU sensor data for positional tracking.

4 Software Technology

4.1 Camera Tracking

The tracking technology used for this thesis requires a camera to eliminate the drift of the responsive but error prone IMU sensors. Since the phone only has one camera 3D dept perception is derived from motion.

4.1.1 Visual Odometry (VO), Visual Inertial Odometry (VIO)

Yousif et al. ((2015)) describe Visual Odometry (VO) as follows: “VO mainly focuses on local consistency and aims to incrementally estimate the path of the camera/robot pose after pose, and possibly performing local optimization.” The process of combining the data of IMU and a camera to track the position is called Visual Inertial Odometry (VIO). The frame rate at which the position can be updated is limited by the speed by which the camera image can be processed, position information can be extracted and then used to correct for drift by the IMU.

4.1.2 Simultaneous localization and mapping (SLAM)

In recent years it became possible to run SLAM algorithms which extend the pure motion tracking about map building directly on the phone. Yousif et al. (2015) describe SLAM as follows: “SLAM aims to obtain a globally consistent estimate of the camera/robot trajectory and map.”

4.2 Software Development Kit's (SDKs)

4.2.1 Virtual Reality SDKs

Beside the AR SDKs that bring 6DoF positional tracking to phones this project relies on SDKs which allow supported smartphones to be used as HMDs and display convincing 3D virtual content. Google and Oculus both developed SDKs for their respective platforms with the Oculus Mobile SDK for GearVR compatible phones and the Google VR SDK for Daydream and Cardboard compatible phones. These SDKs target different phone models and together bring phones-based VR to almost all phones currently in use worldwide. By targeting all three SDKs the reach of this thesis is maximized and more people should be able to benefit from this research. The two VR SDKs are provided for native development and for the two most widely used game engines Unreal and Unity. The SDKs provide developers with all the necessary software to build full 3DoF VR applications. The Oculus Mobile SDK provides all the necessary tools for

developing for the GearVR platform and the Google VR SDK does likewise for developing for Google Cardboard and Daydream HMDs. Together these two SDKs bring phone-based VR functionalities to most modern phones. The VR and AR SDKs can both be imported into Unity and used in combination to complement each other. Despite the Cardboard/Daydream and GearVR platform offering similar functionalities the corresponding VR SDKs differ significantly in naming conventions or how and what kind of sensor data can be accessed which complicates the joint development for both platforms.

4.2.2 Augmented Reality SDKs

The two most prominent Augmented Reality SDKs offering motion tracking via the built-in RGB camera are ARCore (more than 400 million devices) as the successor of the Tango platform and ARKit (more than ?? million devices) which runs on some of the latest apple smartphones and tablets. ARCore and ARKit are backed by Google and Apple, offer great performance, good quality motion tracking, are compatible with many phones, are specially calibrated/certified for the hardware they are running on which improves the tracking quality and are free of charge. All these characteristics make those SDKs best suited for this thesis and influenced the decision to choose them over other prominent AR SDKs like Vuforia, Wikitude or MaxST which offer similar functionalities like SLAM, motion tracking, image/object detection, plane finding and so on. Since this thesis is interested in extending phone-based VR with 6DoF head tracking, a VIO tracking solution would in principle be enough, but ARKit and ARCore rely on SLAM or concurrent odometry and mapping respectively which extends the pure tracking part with an on the fly map building process. For AR applications who are used to add virtual objects to the image of the real world it is essential to have a consistent understanding of the surrounding which they build using the SLAM software. Having a map of the surrounding available in addition to the pure position information opens up other use cases that could be explored in the future projects like building a representation of the surrounding as a sort of safety system that warns the user when he comes too close to an object. Relying on proprietary systems like ARCore and ARKit for an open source project is not ideal since the code providing the positional tracking capabilities is not fully accessible. Open source tracking solutions are also available and in their 2018 paper Cortés et al. (2019) compare five different tracking solutions: three proprietary systems ARCore, ARKit and Tango and two open published academic tracking solutions “Robust Visual Inertial Odometry (ROVIO)” and “Probabilistic Inertial-Visual Odometry (PIVO)”. Cortés et al. (2019) compares the tracking performance over longer periods of time and come to the conclusion that the

Tango device offers the best tracking results, followed by ARCore and ARKit. The academic solutions showed some weaknesses and diverged significantly from the ground truth data.

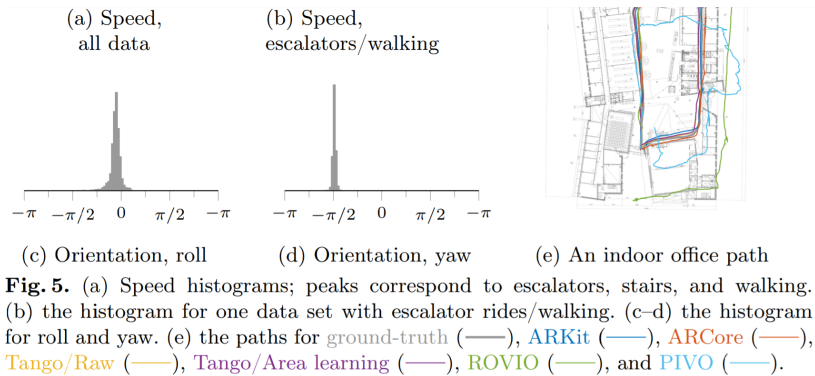


Figure 3: Comparing ARCore ARKit and Tango AR SDKs. Reprinted from “ADVIO: An Authentic Dataset for Visual-Inertial Odometry” by Cortés et al. (2019)

Cortés et al. (2019) choose these two academic tracking solutions since they showed similar results to the Tango tracking but when running on an iPhone with an inferior IMU sensor and a camera with a small field of view they performed worse than ARCore, ARKit and Tango. But even when the two open published academic tracking solutions are not well suited to run on the chosen iPhones with cheap sensors and tracking the position reliable over large distances, that doesn’t mean that they would not work well over shorter periods of time which are more important for good positional tracking on phone-based VR. Cortés et al. (2019) list multiple reasons for the lower performance of the academic tracking solutions compared to the proprietary systems. One possible reason for the differences according to Cortés et al. (2019) is that ARCore and ARKit can rely on factory-calibrated IMU data. Comparing open source academic tracking solutions against proprietary systems like ARCore and ARKit for the use of bringing positional tracking to phone based VR on different phone models with better IMUs and wider field of view cameras could be subject to future research especially when being interested in an alternative to the proprietary systems. Cortés et al. (2019) also compare the tracking results of the three proprietary systems with one another describing Apple’s ARKit as performing well in most situations but showing drift in some outdoor settings, ARCore as being more aggressive when performing visual loop-closure which can lead to jumps and Tango as performing best and close to ground truth because of the specialized hardware. Nowacki et al. (2019) also made comparison measurements but looked specifically at the differences between ARCore and ARKit on different phone models and measured more characteristics like CPU load, memory usage, loading time of models, accuracy of plane detection and performance under different lighting conditions.

Just as Cortés et al. (2019), Nowacki et al. (2019) also recognized that ARCore performs slightly worse when used indoors but is ahead in outside environments. Additionally, Nowacki et al. (2019) showed that ARCore loses tracking after fast movements about three times as often as compared to ARKit but both need roughly 2s to recover. On the other hand, ARCore recovers with just above 0.5s significantly faster after full occlusion of the camera compared to ARKit which requires 5s. Nowacki et al. (2019) explain this difference in recovering time with the different frame rates ARKit (60FPS) and ARCore (30FPS) have access to. With ARCore version v1.11.0 now supporting 60FPS on some devices the described difference in recovery rate is likely less pronounced or disappeared completely which should be investigated in future research. An extensive list on all the differences and advantages of the two platforms can be found in the corresponding literature. Despite ARKit offering better indoor positional tracking results ARCore is better suited for bringing 6DoF tracking to phone-based VR since only ARCore supports all the available VR SDKs.

4.3 Development Platform

When choosing a platform to support the development of this project one can either use a game engine that supports cross-platform development or develop natively for Android and iOS with the promise to build a more performant application. When developing natively one should consider that offering cross-platform support and developing 3D virtual environments is more elaborate.

Unity is a game engine of choice for this thesis since it is well suited for developing VR applications for mobile, stand-alone and wired VR HMDs, supports all the mentioned SDKs and a broad range of other computing platforms. The Unity game engine is well documented, the community is active and regular releases, updates and bug fixes support the development process. A possible alternative when looking for another cross-platform game engine would be the Unreal game engine. Unity also has developer tools like the profiler built in that support developers with the optimization process by showing which processes need what amount of resources.

5 Perception and Sensation

5.1 Latency and how it is perceived

A term often used when describing the quality and responsiveness of a VR tracking solution is latency. Mania et al. (2004) defines the term as “the time lag between a user’s action in a virtual environment and the system’s response to this action.”. Jerald et al. defines latency simply as “the time required for a system to respond to a user’s actions”. Raaen et al. (2015) points out that the perception of latency varies and can be as low as 3-4 milliseconds (ms) or as high as 100 ms. Jerald et al. recommends an end-to-end system latency of around 5ms to be imperceptible. However according to Niehorster et al. (2017) and Kim et al. (2018) modern wired VR HMDs like the HTC Vive and Oculus Rift CV1 run with a higher latency of around 22ms. Mania et al. (2004) and Jason J. Jerald (20..) mention the hypothesis that not the latency by itself is detected but the consequences latency has on the virtual environment is perceived by the user making it appear to be unstable.

Jason J. Jerald in his PhD thesis (20..) coins the term “scene motion” which describes the effect when the virtual environment shows visual motion while the head is not moving in the real world. He also introduces the terms “unintentional scene motion” which is scene motion caused by the technology including hardware specifications, latency and imprecise calibration. Jason J. Jerald recommends that HMD systems should reduce latency to a point where no scene motion is perceivable by the user.

To understand what these latency values could mean for the sense of presence the research of Kim et al. (2018) on the “Effects of Head-Display Lag on Presence in the Oculus Rift” is very insightful. Kim et al. (2018) used an Oculus Rift (CV1), added varying amounts of latency and measured the perceived scene instability and sense of presence in seven participants. The results of his research were significant and showed a strong correlation which can be seen in Figure 4 and Figure 5

5.2 Immersion

Roche et al. (2019) defined three levels of immersion VR HMDs can offer whereby the lowest level is 360-degree content for example panoramic video, the intermediate level is 3DoF tracking and the highest level is full 6DoF head tracking. Amin et al. also points out that resolution and field of view play an important role in the level of immersion and that more immersive experiences are more distracting which is relevant for applications like pain relieve for example. In Roche et al. (2019) review paper it is also pointed out that the quality of the VR technology may influence the results of a study and that the

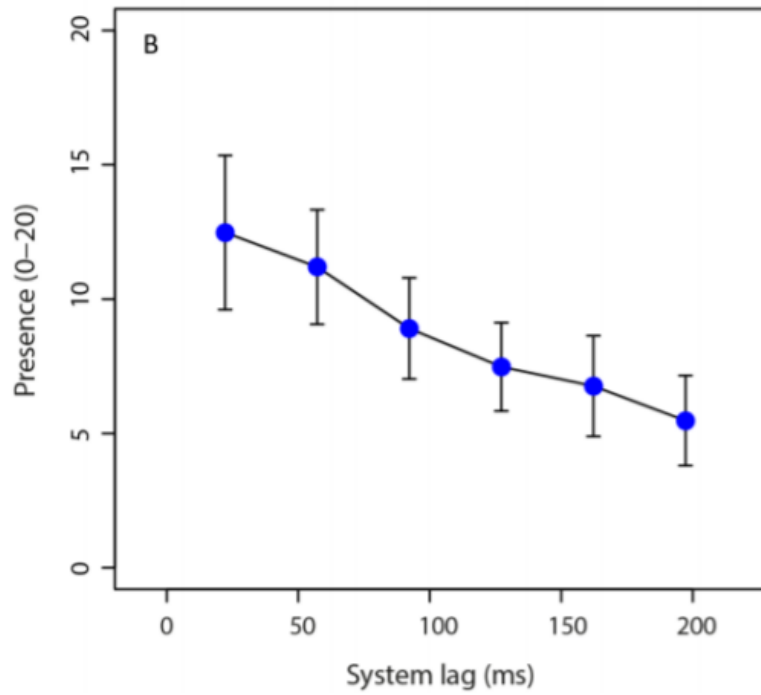


Figure 4: Effects of latency on sense of presence. Reprinted from “Effects of Head-Display Lag on Presence in the Oculus Rift.” by Kim et al. (2018)

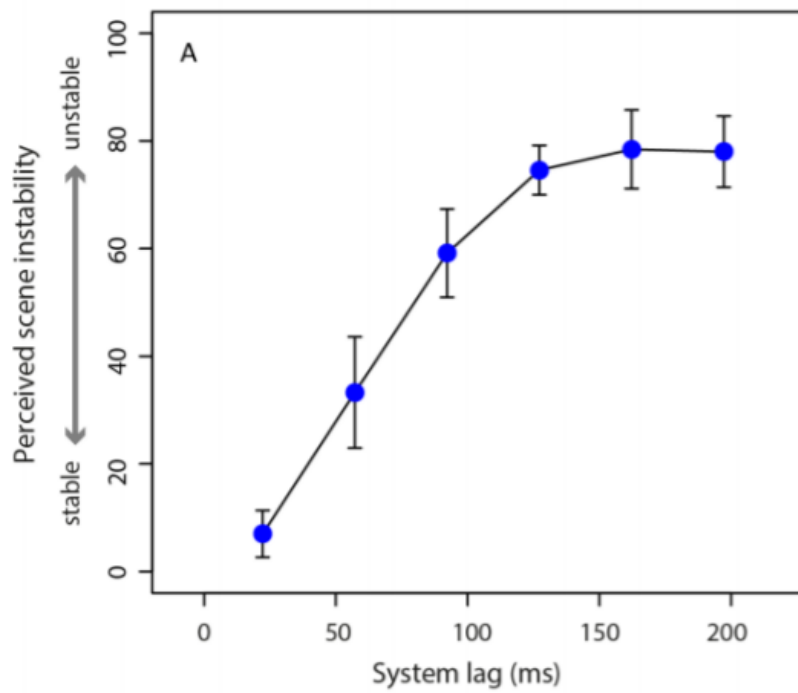


Figure 5: Effects of latency on perceived scene instability. Reprinted from “Effects of Head-Display Lag on Presence in the Oculus Rift.” by Kim et al. (2018)

effectiveness of immersion in improving mental wellness like self-reported distress and mood could be shown. According to Lee et al. (2017) the immersion and presence in VR applications can be improved further by stimulating more senses and using natural interactions.

5.3 Motion sickness

According to Muth et al. (2018) prolonged use of VR technologies can cause side effects like decreased task performance and cognitive deficits similar to the effects of alcohol intoxication whereby the exact causes are unknown. Muth et al. (2018) cite different proposed causes like a narrow field of view, occlusion of peripheral vision or system latency. Brooks et al. (2010) mentions that the “sensory conflict theory” is the theory most widely accepted to be the main cause of motion sickness.

According to Brooks et al. (2010) many factors like age, experience, gender, illness and others can have an effect and lead to motion sickness. Since system latency and motion sickness are strongly correlated this thesis will aim to measure and explore solutions to reducing the latency introduced by the newly developed tracking method. The frame rate will also be measured to avoid introducing additional latency by overloading the system and dropping frames as a result.

According to Langbehn et al. (2018) the inner ear is sensitive to acceleration and orientation whereby the eyes only perceive motion. An evaluation of the responsiveness of the developed system to changes in acceleration is described in the chapter on evaluation of tracking latency. When users perform lateral head movements wearing a phone-based VR HMD with 3DoF tracking the virtual scene doesn't react at all to this sort of head movement. According to Langbehn et al. (2018) the major cause for VR induced discomfort is when the eyes perceive different or no visual movement cues whatsoever compared to the vestibular sense and proprioception perceiving head motion.

6 Head Tracking

In order to feel immersed and present in the virtual world low latency head tracking plays a crucial role. In general VR headsets deliver either rotational head tracking also known as 3DoF or full translational as well as rotational head tracking abbreviated as 6DoF. All currently available phone-based VR HMDs only deliver 3DoF tracking while most of the currently available stand-alone and wired systems offer full 6DoF head and controller tracking. Powell et al. (2016) points out that current phone-based VR is generally used for passive entertainment or viewing 360 degree media and that it is not well suited for the active exploration of virtual environments. Extending the tracking to full 6DoF could open up many more possible applications for phone-based VR. The tracking software developed for this thesis targets all three phone-based VR platforms, but the focus lies on the GearVR in combination with ARCore.

For 3DoF tracking only an IMU sensor found in most modern phones is necessary since the gravity measured by the accelerometer always points down and can be used to prevent drift and the accumulation of errors over time. In principle the IMU would allow for 6DoF tracking as well but since all sensors suffer from measurement errors calculating the change in position in 6DoF from the IMU alone is not practical since drift accumulates quickly even over short time periods. The camera of the phone can be used together with Visual Odometry (VO) algorithms to eliminate the drift that occurs when using the IMU for positional tracking. The IMU has the advantage of being highly responsive and detect quick changes in the HMDs orientation and acceleration, the camera on the other hand can be used to calculate the change in position over longer periods of time without experiencing drift.

Using ARCore for tracking comes with other challenges beside latency. Errors in the position data provided by ARCore are likely caused by ARCore's aggressive loop closure process as described by Cortés et al. (2019) and drift between IMU and camera data which needs to be resolved by the ARCore software internally. When ARCore is used for AR application intended to be used an arm length away and only displays the virtual content in 2D such errors are hardly noticeable. When using ARCore for tracking a HMD one must consider that VR applications place stricter requirements on the tracking technology as is discussed in the chapter on Perceptions and Sensation.

6.1 Leveraging ARCore for Positional Head Tracking

In this sub chapter the detailed process of exploring the available AR tracking software mainly ARCore for bringing 6DoF tracking to phone-based VR will be described and methods used for quantifying tracking quality will be discussed. The process of finding

the right tools and methods was iteratively refined based on new results and measurements. Initial tests were carried out directly in VR and it was immediately noticeable that the tracking is not error free. The reasons were not clear at this point and possible reasons for the tracking errors spanned from general inaccurate tracking by the ARCore software, performance bottlenecks on a rather old phone like the Galaxy S7 used, wrong setting in the game engine or incompatibilities between the multitude of leveraged SDKs or the SDKs with the game engine.

Hypothesis One: ARCore suffers from tracking errors. To test the most plausible hypothesis that ARCore suffers from tracking errors, the position data were recorded at 60fps with a Unity script, one of the 3 tracked direction x,y,z were chosen and data points plotted as shown in Figure 6. To not influence the results by holding the phone in the hand the camera was mounted on a skateboard and pushed once resulting in a curve shown in Figure 6. The blue points are the position values in x direction of the phone. The position values come in pairs since the phone records at 60fps and tracking data by ARCore is provided at 30fps. Sometimes position values come in triplets suggesting that the software was not able to deliver a new position value on time and the old value is returned again.

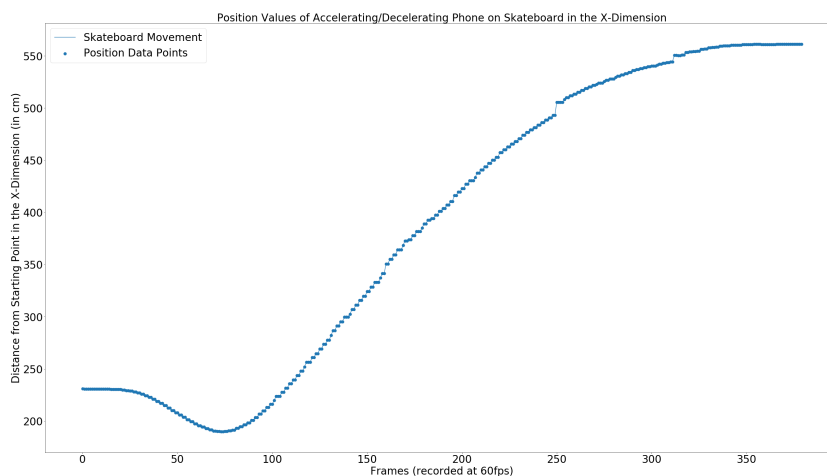


Figure 6: Position data in x direction of a Galaxy S7 fixed to a skateboard. Curve shows deceleration over roughly 4 meters.

To rule out that the jumps detected in Figure 6 are caused by unevenness in the track a different setup was chosen whereby the phone was attached to a pendulum. This way the movements was still based on a physical movement which helped with the interpretation of the data and reduced the likelihood that bumps in the floor falsify the results. The tracking results with the phone attached to the pendulum are shown in Figure 7. It can be noticed that the position values in Fig. 11 no longer comes in pairs as was the

case in Fig. 10 but each frame now has a different position value. This is due to using an interpolation approach explained in chapter 9.5. The interpolation just generates a new data point in between an old and a new position value resulting in a more balanced distribution of data points having a positive effect on the visual appearance of the scene. To determine if an uneven track caused the displacements seen in Figure 6 the focus was put on Figure 7 and especially on the areas which seem not to fit the otherwise smooth curve and deviate from the other well-spaced data points. These irregularities indicate that the displacements in the curve are still present and hence it can be concluded that the way the measurements is performed is not the reason for these displacements.

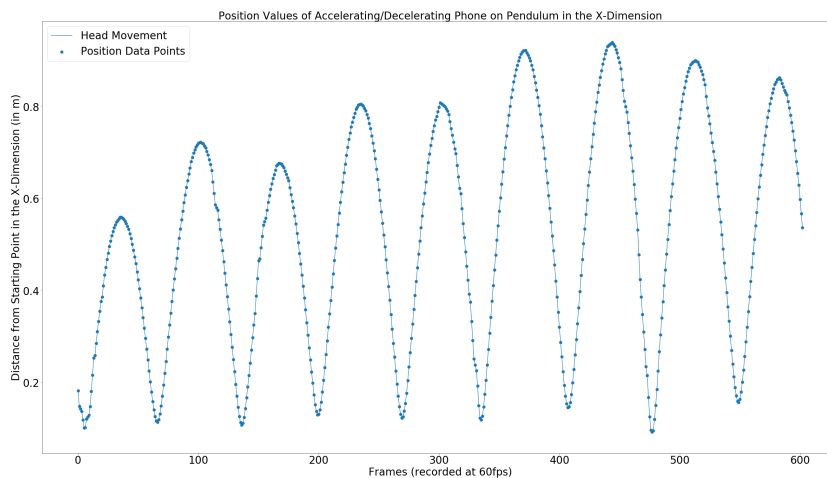


Figure 7: Using interpolation leads to a more uniform distribution of the tracking datapoints. Also, visible occasional discrepancies in the curve most likely caused by ARCore internal loop closer and drift correction algorithms.

The occasional displacements seen in Figure 6 and 7 are not the only indicators for tracking errors. Using this tracking version inside a VR headset results in occasional jumps of the camera movement which are noticeable by the user. If the occasional jumps in the headset and the displacements seen in the curve are the same phenomena was not clear. To record the display while running the tracking application the free “Screen Recorder – XRecorder” software found on the Google Play Store was used. For simplifying the measurements process the phone was no longer attached to a pendulum resulting in a less regular curve seen in Figure 8. Clearly visible is that the updates of the position data are delayed significantly compared to previous measurements taking up to 6 frames until a new position value is available for tracking. This experiment made clear that performance has a significant effect on the tracking quality of the developed app and indicates that focusing on the performance first before performing further tracking measurements is the right approach.

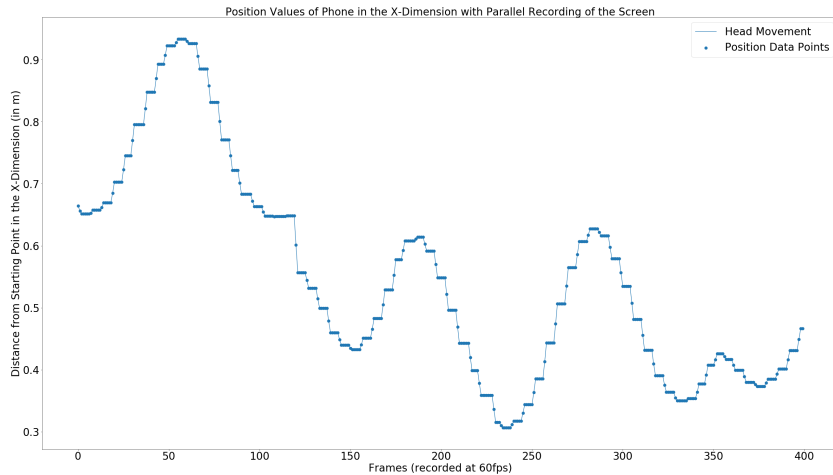


Figure 8: Screen capture recording running in parallel to the phone-based VR HMD tracking its position in space. Running both processes in parallel required more processing power than the phone can deliver resulting in the tracking not being able to update the position at 60fps.

Hypothesis Two. Bad performance leads to errors in tracking. To evaluate the performance of the application the phone was connected to the PC where the Unity profiler was actively recording and measuring multiple performance indicators of the application running on the phone. To analyze the performance, the Unity profiler was set up to display the frame rate and list the most demanding processes in descending order as can be seen in Figure Tango Destroyed. The green-blue graph shown in Figure 9 indicated that the app runs close to the 60fps limit and dropped below this threshold regularly. This 60fps limit also seen in the screenshot as a white line indicates the overall performance of the application and a drop below 60fps results in the app not being able to render a new image to the screen which refreshes at 60fps and hence results in a noticeable freeze of the whole application. As described in chapter 5 Perception and Sensation having a refresh rate of 60fps is necessary to deliver a coherent experience in VR that does not show noticeable stutter by frames not being updated.

The results of the Unity profiler seemed to confirm the second hypothesis that bad performance is to blame for the noticeable stutter and the discontinuities in the plotted position data. Looking at the list of the most demanding processes one process “EarlyUpdate.TangoUpdate” was especially troubling since it accounted on average for more than 1/3 of all the available processing power. This process seems to be part of ARCore since “Tango” is the predecessor of ARCore and uses large portions of the Tango code. The “EarlyUpdate.TangoUpdate” function itself cannot be accessed directly since ARCore is a partly closed SDK which makes troubleshooting some issues difficult. To evaluate what impact ARCore has on the performance of the app and to further investigate the “EarlyUpdate.TangoUpdate” function a pure AR app was implemented. The

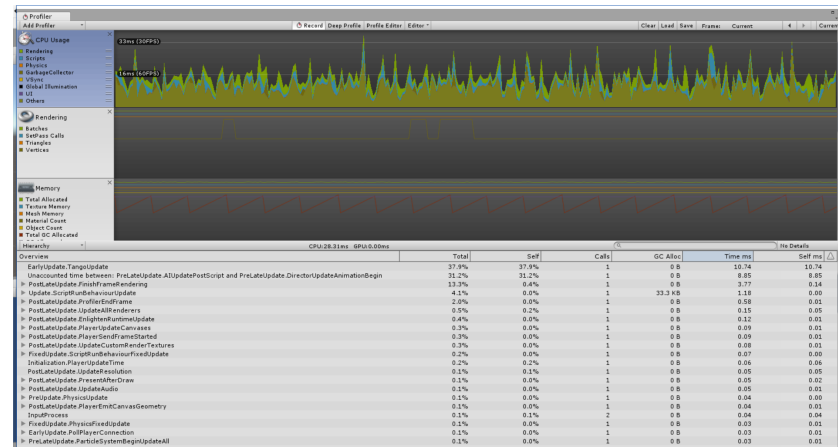


Figure 9: Running an app that combines the VR functionalities of the Google VR SDK with the positional tracking functionalities of the ARCore SDK.

AR app was installed on a Galaxy S7 and the profiler used to display the performance as shown in Figure 10.

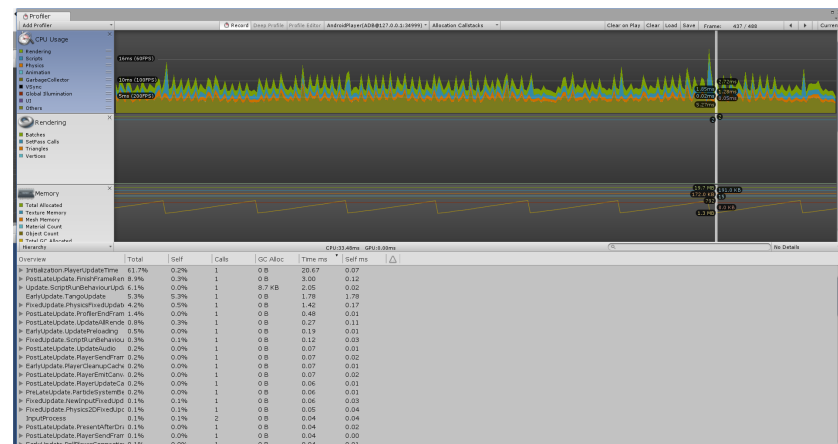


Figure 10: Visible the excellent performance when using the “HelloAR sample app” and no VR mode is enabled. Using Unity 2018.2.20.

The performance of the AR app was much better than expected with framerates staying often well above 100fps and the “EarlyUpdate.TangoUpdate” which in previous measurements required more than 30% of the available CPU performance was down to 5%. This result indicated that adding the VR SDK to the mix results in the performance dropping from above 100 frames, when the ARCore app is running on its own, to below 60fps when a VR camera is added to the scene. To investigate this phenomenon the performance of the AR app was constantly measured and different VR SDKs added. It turned out that as soon as a VR SDK independent of whether the Google VR SDK or the GearVR SDK was used the performance dropped sharply. These results again seemed to confirm the second hypothesis that bad performance is to blame and that

the combination of VR and AR SDKs is too demanding for an old phone like the S7 even when VR apps and AR apps on their own work fine combining them is too much. Since there are no other VR SDKs available for Android except those from Google and Oculus a custom VR setup was developed just for testing if a custom VR camera could be used in combination with the well performing ARCore SDK to build an application which does not rely on any of the VR SDK from Google and Oculus. Making use of two in-game cameras, spacing them apart at the human pupillary distance which is in general between 54-68mm and splitting the screen in two halves worked well for setting up a custom VR camera delivering a different view of the virtual world for the left and right eye. To add rotational head tracking to the custom VR camera the IMU sensor was used and the change in rotation transferred to the two in-game cameras. For adding positional head tracking the tracking capabilities of the ARCore SDK were used as described before. With this custom VR camera, the performance was evaluated shown in Figure 11. The first half of the profiler screenshot in Figure 11. shows the performance of an app using the GearVR SDK for generating a 3D image and delivering rotational head tracking and ARCore for positional tracking. The second half of the performance curve shown in Figure 11 shows a different app using the custom build VR camera consisting of two side-by-side cameras and again using ARCore for positional tracking. Surprisingly the performance of the custom camera is excellent compared to the GearVR + ARCore combination. This result either meant that the GearVR camera is much more demanding than the custom VR camera or that combining the VR SDKs with the ARCore SDK leads to incompatibilities which manifest in the “EarlyUpdate.TangoUpdate” process. That the GearVR camera is more demanding than the custom camera was rather unlikely since Oculus uses many software optimizations to reduce the performance impact of the VR camera. A second argument against this explanation was that the “EarlyUpdate.TangoUpdate” process with the high impact on the performance was somehow related to the ARCore SDK not the GearVR SDK. The most likely explanation therefore was that ARCore and all the available VR SDKs are incompatible or that the Unity game engine does not support both SDKs running in parallel.

Hypothesis Three: Unity does not support running both ARCore and the available VR SDKs To test this hypothesis a different version of Unity called AR Foundation was used. AR foundation differs from the regular Unity engine by targeting a multi-platform API instead of ARCore or ARKit directly allowing for an easier development of multi-platform AR applications. Building an app either using the VR SDK or an AR SDK alone was possible but combining both into one app resulted in the app freezing

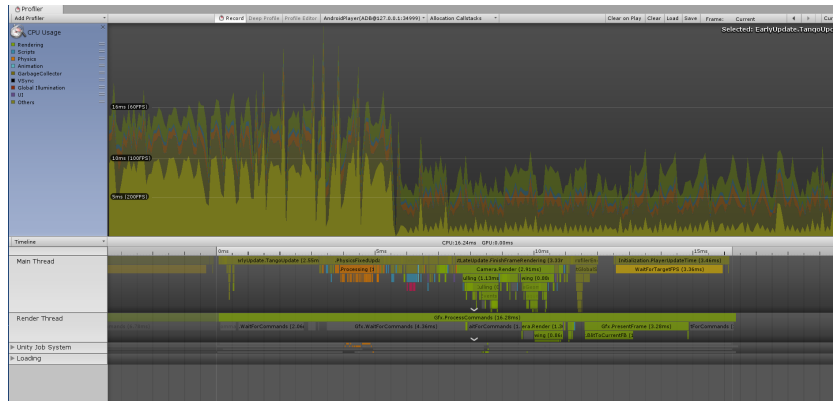


Figure 11: Performance measurements of the GearVR and ARCore setup in the first half and the ARCore and DIYVRCam in the second half. The comparison shows the impact the “EarlyUpdate.TangoUpdate” function has on the performance.

when starting on the Android device. For the AR Foundation version of Unity the Hypothesis that the game engine itself is to blame for the incompatibility can therefore be assumed but it was still unclear if the regular Unity version is also to blame for the performance hit seen when combining the different SDKs. Developing the app with one of the alternative game engines Unreal would also be an option but was not performed since this would have been beyond the scope of this work.

Hypothesis Four: ARCore is incompatible with the VR SDKs independent of the game engine used. This hypothesis seemed to be confirmed by comments made by Google engineers on the official GitHub page of ARCore stating that: “Hi, ARCore + Cardboard is an unsupported configuration and we won’t be fixing this.” (ARCore GitHub page Unity issue 102) Since the ARCore SDK is not fully accessible further investigating the “EarlyUpdate.TangoUpdate” method to determine what might cause this incompatibility was not possible. This seemingly insurmountable hurdle seemed to rule out the use of ARCore for extending phone-based VR with a positional tracking function. At least when trying to build the app with Unity and apply on Android devices. The Vuforia SDK with similar functionalities to ARCore was chosen as an alternative to realize the tracking after all. The setup process was more complicated compared to ARCore nonetheless it was possible to combine the Vuforia SDK with the VR SDKs to build an application of which the performance Figure 12 and tracking quality Figure 13 could be measured.

The sharp tracking quality differences between ARCore and Vuforia were not just noticeable in the recorded data Figure 13 but also when wearing the headset seriously impacting the quality of the experience questioning the usefulness of Vuforia tracking

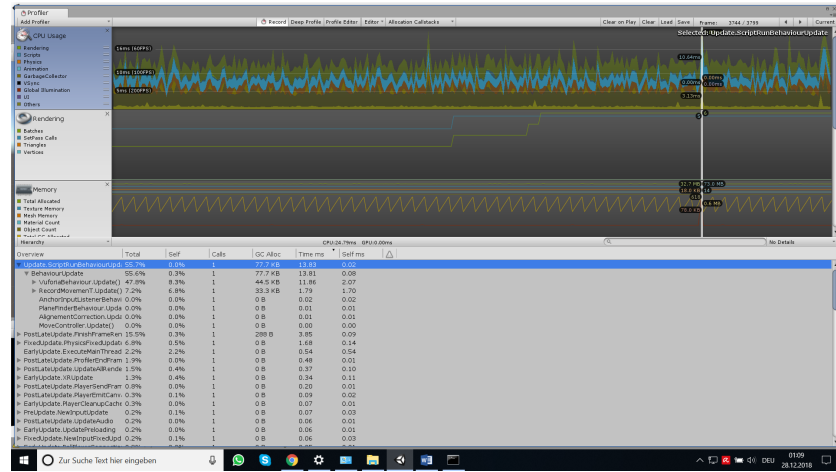


Figure 12: Vuforia powered positional tracking VR app running on the Galaxy S7 showing acceptable performance characteristics.

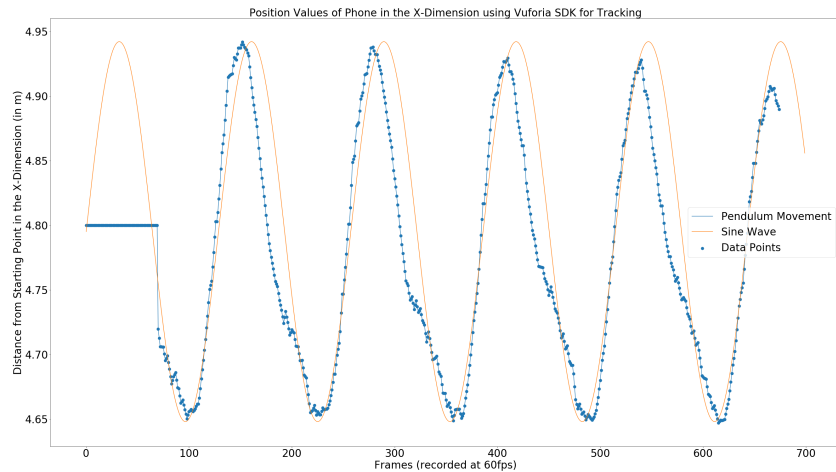


Figure 13: The graph shows the x position values of an app build using the Vuforia AR SDK. The phone is attached to a pendulum. The curve describing the pendulum motion shows significantly more jitter compared to curves recorded with ARCore. The sinewave describes the ground truth of the pendulum movement.

for the goal of bring positional tracking to phone-based VR systems. With no alternative to using ARCore for getting acceptable positional tracking another approach was taken using an old Google Cardboard SDK which at the time of release was not part of the Unity engine but had to be imported separately. Using this old version allowed to disable the VR support in the Unity engine and import all the necessary software via the old Google Cardboard SDK. Unfortunately, the old Google Cardboard SDK could not be imported into the latest Unity version since it was incompatible and importing it caused lots of incompatibility errors. Using an older Unity version allowed for the Google Cardboard SDK to be imported correctly but caused lots of incompatibility

when importing the latest ARCore SDK. After no alternative to using ARCore and the VR SDKs built into Unity could be found the decision was made to accept that the “EarlyUpdate.TangoUpdate” is requiring a large amount of the available performance betting on future updates to resolve the issue. In the hope that the performance impact of the “EarlyUpdate.TangoUpdate” function will be less relevant when using a more powerful device, a Note 8 was used to repeat the measurements and the results are shown in Figure 14.

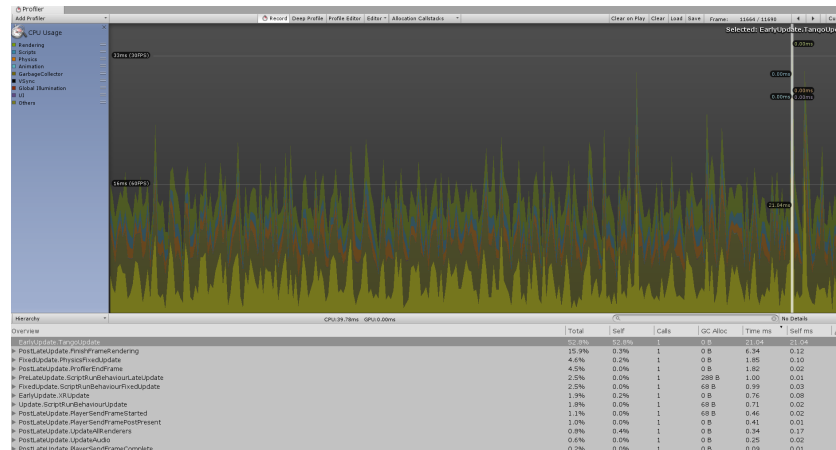


Figure 14: Performance measurement of the combination ARCore + GearVR running on the Note 8.

When comparing Figure 14 showing the app running on a Galaxy Note 8 with Figure 9 showing the app running on a Galaxy S7 no significant differences can be identified. Trying to explain this result the performance benchmarks of the two phones were compared shown in Figure 15 indicating that the Note 8 should perform better compared to the S7. Since the benchmark data show a significant difference it is not clear how much they contribute to the performance of an VR app. It was therefore concluded that the differences might be too small to notice a difference in the VR app tested on the two phones.

Without a roadmap on how to avoid the performance issues introduced by the “EarlyUpdate.TangoUpdate” function the focus shifted to the exact cause of the displacements seen in all the position recordings. Looking at the raw tracking data indicated that these displacements could be filtered out since the outlier data points differed significantly from the transitions seen between the previous and subsequent data points. The exact process of developing a filter for these displacements is described in chapter 10 Filtering.

While developing the filter extensive use of debug logs was made. These debug logs caused the app to lag significantly during the measuring phase and the profiler was used to investigate the causes. The profiler showed (Figure 16) that the debug logs have





















 Samsung Galaxy Note 8 Samsung Exynos 8895 Octa @ 1.7 GHz	365	
 Samsung Galaxy S8 Samsung Exynos 8895 Octa @ 1.7 GHz	358	
 Samsung Galaxy S8+ Samsung Exynos 8895 Octa @ 1.7 GHz	355	
 Samsung Galaxy Note 8 Qualcomm Snapdragon 835 @ 1.9 GHz	346	
 Samsung Galaxy S7 Samsung Exynos 8890 @ 1.6 GHz	342	
 Samsung Galaxy S8+ Qualcomm Snapdragon 835 @ 1.9 GHz	341	
 Samsung Galaxy S8 Active Qualcomm Snapdragon 835 @ 1.9 GHz	341	
 Samsung Galaxy S8 Qualcomm Snapdragon 835 @ 1.9 GHz	339	
 Samsung Galaxy A50 Samsung Exynos 9610 @ 1.7 GHz	336	
 Samsung Galaxy S7 edge Samsung Exynos 8890 @ 1.6 GHz	322	

Figure 15: Benchmark comparing performance of Note 8 and S7 [23]

an especially large impact on the performance and not removing them when building and testing the app resulted in the noticeable lag. The profiler screenshot shows three segments. The first 1/10 segment is the GearVR app with ARCore tracking starting up. The second large segment shows the performance of the app running and tracking the position clearly visible the “EarlyUpdate.TangoUpdate” process requiring most of the processing power. The last 1/10 of the curve shows a blue area which is the CPU usage taken up by the debug logs running in the background. The debug log processes are delayed because they are positioned after an if statement which becomes true after a few seconds. Compared to the the “EarlyUpdate.TangoUpdate” which seems to be added on top of other processes the Debug Log functions suppress the “EarlyUpdate.TangoUpdate” function reducing its influence on the overall performance. This indicates that the “EarlyUpdate.TangoUpdate” process is some sort of pseudo function just pretending to require all the performance left over making the app appear to always run at the performance limit at least when measured with the Unity profiler.

At this stage in the development of the tracking solution it was clear that performance was not an issue and that the cause for stutter under the HMD and displacements visible curves of the plotted position measurements were only caused by errors in the tracking caused by imprecisions in the tracking process of ARCore and possible realignments and loop closure processes performed internally by the ARCore software. How these tracking errors were eliminated is described in chapter 10 on filtering.

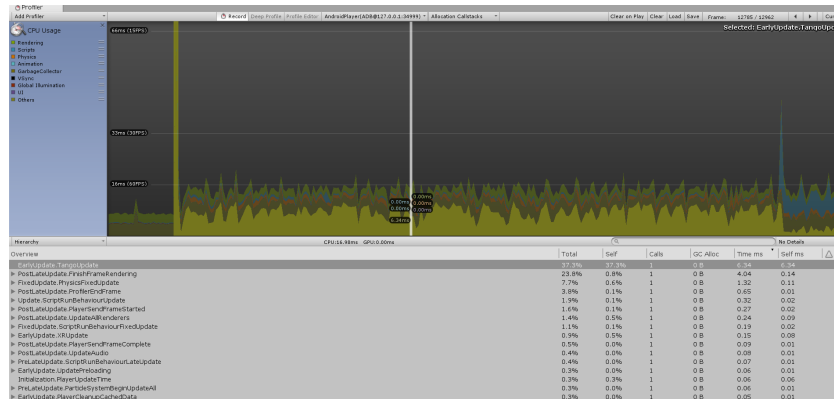


Figure 16: Unity profiler showing the performance of a GearVR + ARCore app and in blue the performance impact of multiple Debug Log processes on the performance of the app and especially on the effect the “EarlyUpdate.TangoUpdate” process has on the overall performance. The Debug Log processes are not activated at the start of the app but instead after a few seconds into the app running.

6.2 Evaluation Of Tracking Latency

When talking about tracking and latency in VR the first term coming to mind is “frame-rate” which is described by Raaen et al. (2015) as “a measurement of how fast frames are sent through the rendering pipeline” or in other words the number of frames per given time period coming through the rendering pipeline. Another term often used is the display refresh rate describing how often the display can redraw the image and is measured in frequency (Hz). Looking at phone-based VR HMDs they all have a display refresh rate of 60Hz which results in a display latency of $1/60$ s or 16.67ms. For comparison the HTC Vive has a display refresh rate of 90Hz and a system latency of 22ms. When intending to deliver a tracking solution to phone-based HMD’s that track with no noticeable latency the positional tracking data should be updated with the same rate as the display refresh rate of 60Hz. ARCore and ARKit both claim to offer 60fps tracking whereby ARCore currently only offers 60fps on the Pixel 2 and Pixel 3. If the claimed values of 60fps positional tracking are correct as soon as the phone is moved, and the next frame is shown ARCore/ARKit should also recognize the change and return a new position value. If the display refresh rate and the tracking refresh rate are in sync which is an option ARCore offers the tracking should work with no perceivable latency. In the case of ARCore running on phones that only support 30fps it would be possible that the first position update is not available at the first frame. In the second frame the new position value should be delivered but the system latency would have increased by 16.67ms at 60fps to 33.34ms at 30fps on a phone like the Galaxy S7 used for this thesis. The time ARCore takes to react to movements of the phone was measured using the high-

speed camera function of a Xiaomi Mi 9T Pro recording at 240fps. The values shown on the phone display Figure 18 are written down every fourth frame when going frame-by-frame through the 240fps high-speed video. Since the high-speed camera records at 240fps always four frames of the video footage show the same value. The four frames are only added once to the list. The distance of each new datapoint to its predecessor is calculated and when it gets larger that indicates that the tracking software started to update its position and move away from the previous position. By comparing the time point when the video showed that the phone running the tracking software started moving to the time point when the position values on the phone screen changed can be used to determine the system latency. In this experiment the latency of the ARCore software was measured. No filter or interpolation processes were added resulting in the quickest possible response time of the software. As shown in Figure 19 the position data is not updated as quickly as expected from a 30fps tracking solution. Even when the ARCore data sheet mentions a frame rate of 30fps the measurements showed that an AR app using ARCore's tracking solution, running on a Galaxy S7, takes 10-12 frames or 167-200ms to recognize a change to its position. An end-to-end latency of 167-200ms would according to Kim et al. (2018) research suggest that users will rate the sense of presence rather low and perceive the virtual scene as unstable independent of how well the filter is able to remove errors made by the tracking software. To understand what these latency values could mean for the user have a look at chapter 5.3 where the research of Kim et al. (2018) and the effects of latency on test subjects is described.

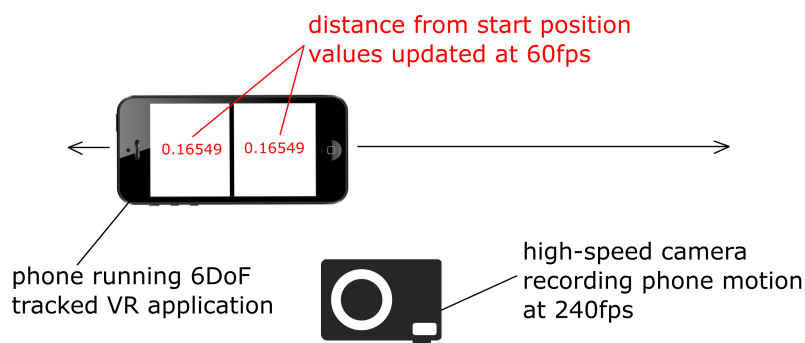


Figure 17: Schematic experimental setup to measure the latency of the ARCore tracking.

According to an ARCore engineer ARCore's visual-inertial-odometry algorithm only updates the position at 10fps and forward-integration using the IMU sensor is used to end up with the claimed 30fps. Since this statement is rather old and new versions of ARCore now support 60fps on some devices it would be a logical next step to measure the latency of such an ARCore version on an appropriated device. It would also be interesting to repeat the experiment using an app developed natively and compare the

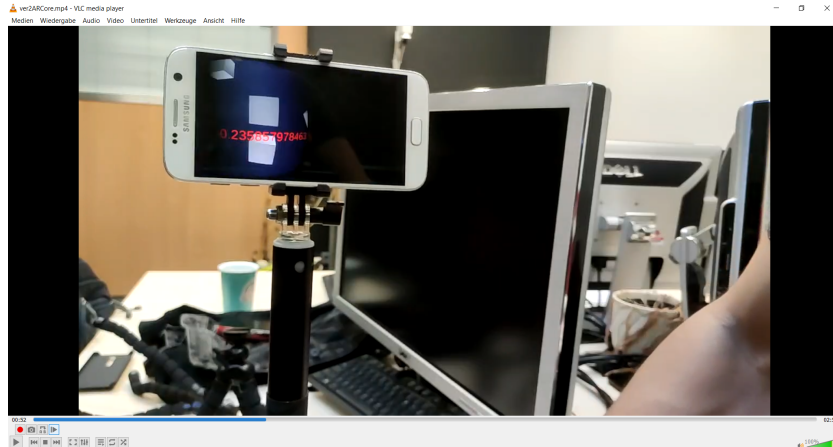


Figure 18: Using the VLC media players and its single frame-by-frame function to go through every frame. As soon as the phone starts to move the camera frames are counted and noted and the number on the screen which describes the current position is listed. The frames it takes the tracking software to notice a change in its position can then be used to determine the system latency of the ARCore tracking software.

latency to the results observed in this thesis by measuring an app developed with a game engine. Since many phones do not yet support 60 fps tracking by ARCore another possible solution to reduce latency is making use of the IMU of the phone or HMD which are known to respond quickly to changes in acceleration. The Daydream ready phones have a capable IMU sensor built in and getting data from them for improving the positional tracking responsiveness might be possible. The Galaxy S7 used for this thesis is not certified as Daydream ready and the built in IMU sensor might not be as good as the ones in other devices but further experiments should be performed to find out whether the IMU is capable enough to improve the responsiveness of the ARCore tracking software. Using the IMU in the GearVR headset instead of the one in the phone would also be an option but Oculus does not provide access to the IMU data of the GearVR headset.

A Google engineer also made a recommendation on how to increase the responsiveness of ARCore by forward integrating the IMU data to get the velocity. He recommends to create an anchor at the position of the camera so that the next frame can be compared with the anchor position in the previous frame. This way one can check if the ARCore tracking did a jump in position due to ARCore updating the IMU data because of drift over time with data from the VIO or SLAM processes. Such readjusting can happen when the understanding of the world due to loop closures for example changes [27]. This might be a solution for improving the responsiveness of the here presented software in the future.

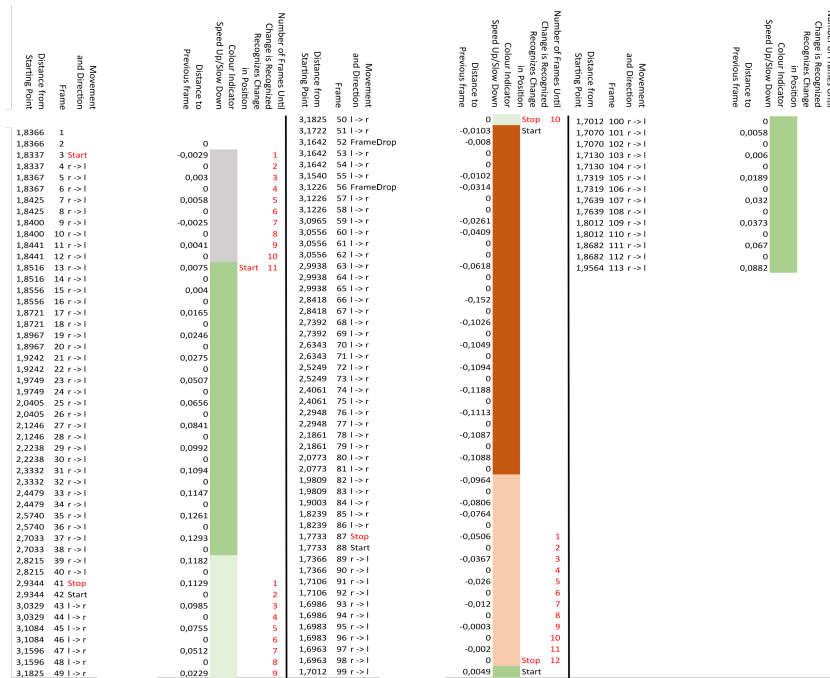


Figure 19: List of the position values displayed on the phone screen at 60fps. The first column shows the distance from the starting point which is displayed on the phone’s screen. The second row shows the number of frames that have passed since the beginning of the measurement. The third column shows if the phone is moving and in which direction. The fourth column calculates the distance from the previous position. The fifth column uses colors to indicate areas of speeding up and slowing down either towards the right (green) or the left (red) side. The sixth column indicates when the tracking software recognizes the change in position either stating, stopping or reversing the direction. The seventh column shows the number of frames it takes from the high-speed camera recognizing a change in position to the tracking software displaying a change in position.

6.3 Interpolation on tracking data to improve visuals

ARCore only delivers new tracking results at 30fps on all devices except the Pixel 2 and Pixel 3 which support 60fps. For human perception to perceive a movement and images as continuous a 60fps update of the position is necessary. To improve the visual quality despite only having a 30fps tracking system available an interpolation approach was chosen. An additional tracking point is thereby generated from the given 30 fps tracking data. For generating a new tracking point the distance from the current position to the new position value is calculated and divided by two. The current position is then set to the calculated midpoint. In the next frame when no new position value is made available by ARCore the position is set to the value received during the last update as illustrated in Figure 20.

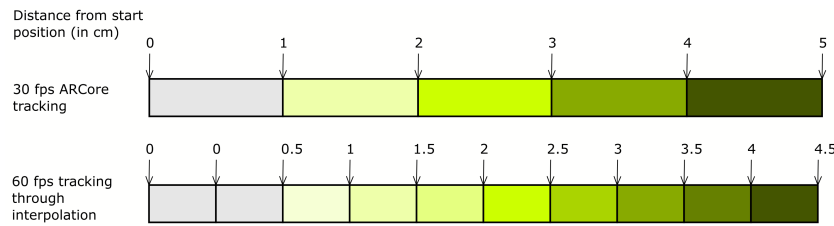


Figure 20: Schematic how interpolation can be used to generate a 60fps tracking solution from tracking data that only updates with a rate of 30fps.

Using this kind of interpolation, the latency of the system is increased by one frame or one could argue by only half a frame. Figure 21 shows that this kind of interpolation can also introduce additional scene motion whereby the virtual environment keeps moving while the head has already come to a standstill.

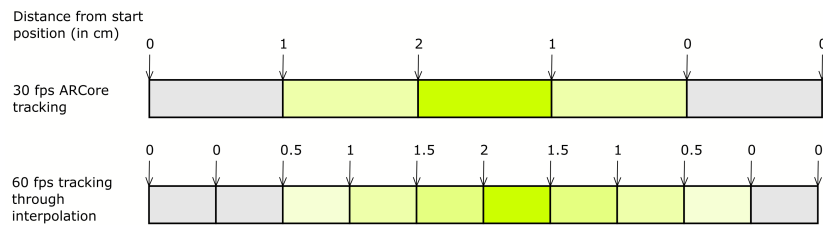


Figure 21: Schematic on how the interpolation introduces lag by not immediately applying the new position data.

An alternative approach which would keep the responsiveness high is linear extrapolation which uses the new position data as soon as it is available and predicts the next position based on the changes seen in the last update. Figure 22 shows this variant schematically. Notice that this approach can lead to overshooting when the prediction is wrong. But when focusing on the critical areas like initial acceleration of the head and coming to a stop this approach shows no added latency and could lead to less experienced scene motion. As the research of Langbehn et al. (2018) on redirected walking has shown small impressions in tracking are not perceivable by users and the errors in position updates introduced by this approach might not be noticed by users.

6.4 Tracking Accuracy and quantification

Beside measuring latency, the overall quality and accuracy of the tracking in 3D space was measured and recorded using a vive tracker. Tracking was slightly different from the comparative measurement, but this should be less of an issue compared to the latency we looked at before. The reason that such differences are less noticeable was described by Langbehn et al. (2018) and is justified by the fact that the sense of vision dominates the sense of

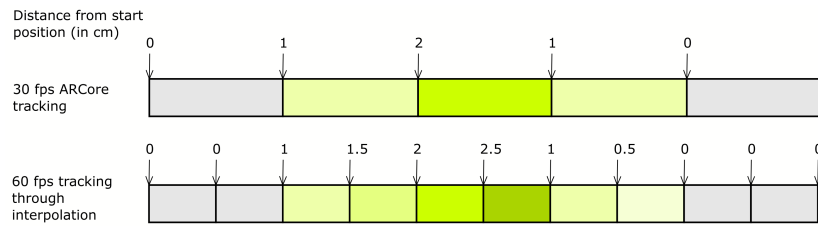


Figure 22: Schematic on how the linear extrapolation could be used to generate a 60fps tracking solution without lag but possibly introducing erroneous position data.

in 3D space should be measured. In order to get the ground truth, a HTC Vive tracker which is tracked via the HTC Vive lighthouse system was used. A script was written in Unity and attached to the Vive tracker GameObject. Recording was initialized by switching to Play Mode and recording performed directly in the Unity game engine. The x, y and z position were recorded and saved in a text file on the PC. On the phone the x, y and z position were recorded with a similar script, saved in a text file and stored on the internal memory. Since two different tracking methods were used the resulting tracking data could not be simply aligned. Three issues needed to be addressed before the data could be aligned.

Compress data that is recorded at a different frame rate

Rotate the coordinate system to align the two datasets

Rescale the dataset to bring it to the same scale

The HTC Vive tracker recorded the data at a faster rate possibly because the target framerate of the Unity engine was set to a higher value compared to the phone. The phone tracked and recorded the position data with 60fps. The PC recorded the data roughly 1.8 times faster than the phone suggesting that the frame rate was set to 100fps. To align the two data sets, one of them must be compressed so that they both have the same dimension.

To simplify the orientation, compression and scaling process the 3D datasets were transformed into 2D datasets shown in Figure 23. This was achieved by calculating the vector between adjacent data points describing the change in position after each frame. By performing this transformation, two lists of vectors were generated. The change in position is the same for the phone and the Vive tracker and by doing the transformation the orientation of the underlying coordinate system became irrelevant. With only two 2D lists left the compression and scaling steps also became easier. The compression and scaling steps were performed with python. Additionally, the Vive tracker dataset was cleaned up because it contained strong outlier points. Figure 24 shows the result of the described steps and that the curves can be aligned and compared.

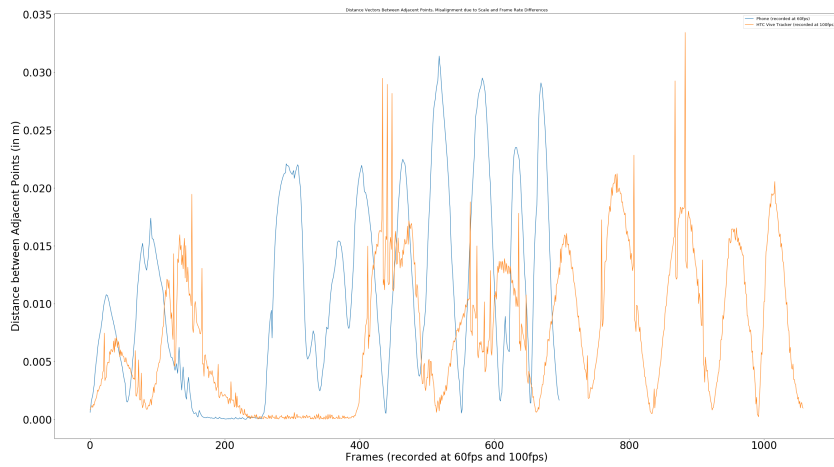


Figure 23: The 3 dimensional positional tracking data of the Vive tracker and the phone running the ARCore tracking solution where transformed into a 2D representation and plotted on top of each other. This 2D representation was generated by calculating the vector length between adjacent points and the length was plotted without scaling or compressing.

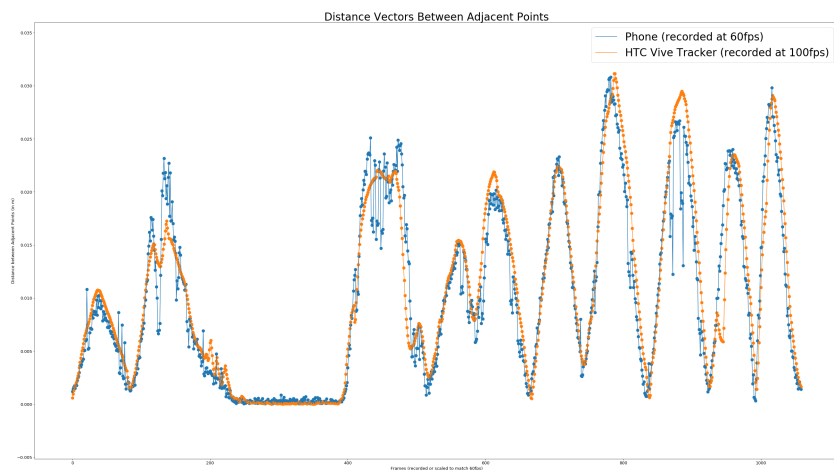


Figure 24: The positional tracking data of the Vive Tracker and the phone where plotted on top of each other after the transformation, scaling and compressing steps where performed.

7 Filtering

7.1 Hard Coded Filter

The hard-coded filter used in this thesis consists of several if statements that if true activate the filter. The if statements are checked for every new position data point provided by the ARCore tracking software. The if statements rely on previous data points and compare changes that occurred in the 2 to 3 preceding updates with the latest change in position. If the new change differs significantly from the previous changes the filter either slightly modifies the new position data or if the change is too large discards the new position data and replaces it with a prediction generated from previous changes. An upper limit for position changes accepted helps filter out extreme jumps in the tracking data which are not humanly possible. Despite the hard-coded filter working well for some cases in practice it requires a lot of fine tuning and needs to be adjusted depending on the speed the head is moving. Distinguishing between large and small head movements is important to adjust the sensitivity of the filter. In this thesis the hard-coded filter is only used for detecting and filtering out movements that are too large to be reasonable and don't need to be considered.

7.2 Kalman Filter

The Kalman filter used in this thesis was released by Johnathon Selstad on GitHub in 2017 and modified to work on the 3-dimensional tracking data delivered by ARCore. One of the main advantages of the Kalman filter compared to the hard-coded filter is that it can automatically adjust to changes in the intensity of the head movement. Another advantage is that the function predicting future changes is updated constantly and refined depending on the incoming data. One of the biggest disadvantages of using the Kalman filter is that the fine tuning of all the variables is cumbersome. For the Kalman filter to work optimally the exact distribution of all the errors of the sensors and the incoming data is required. The Kalman filter version used in this thesis makes an educated guess on these distributions regarding the variance and expected value of the measured data and already delivers promising filter results better than the results that would be possible with a hard-coded filter.

8 Outlook

It is expected that the tracking system presented here will not meet the requirements placed on a consumer product, therefore a warning is appropriate to inform about the danger of suffering motion sickness when using this system in the current state, even when for some users this high latency might be tolerable. Future research should improve the latency of the tracking solution first and then perform a survey like the Motion Sickness Assessment Questionnaire listed by Brooks et al. as one of the most common surveys for measuring motion sickness to evaluate the quality of the VR experience. The filters used in this thesis are not well optimized for reducing latency and by further optimizing the parameters better tracking results should be possible.

The project described in this thesis was parallel to development published on GitHub and enjoyed great popularity among the community. Sharing information on the process on reddit, YouTube and answering comments of interested developers on the GitHub page led to many people downloading the software and asking questions, presenting ideas for improvements and developed their own apps with the provided software. The Youtuber AlitaQuils showed of his own experiments in a video on his YouTube channel [3].

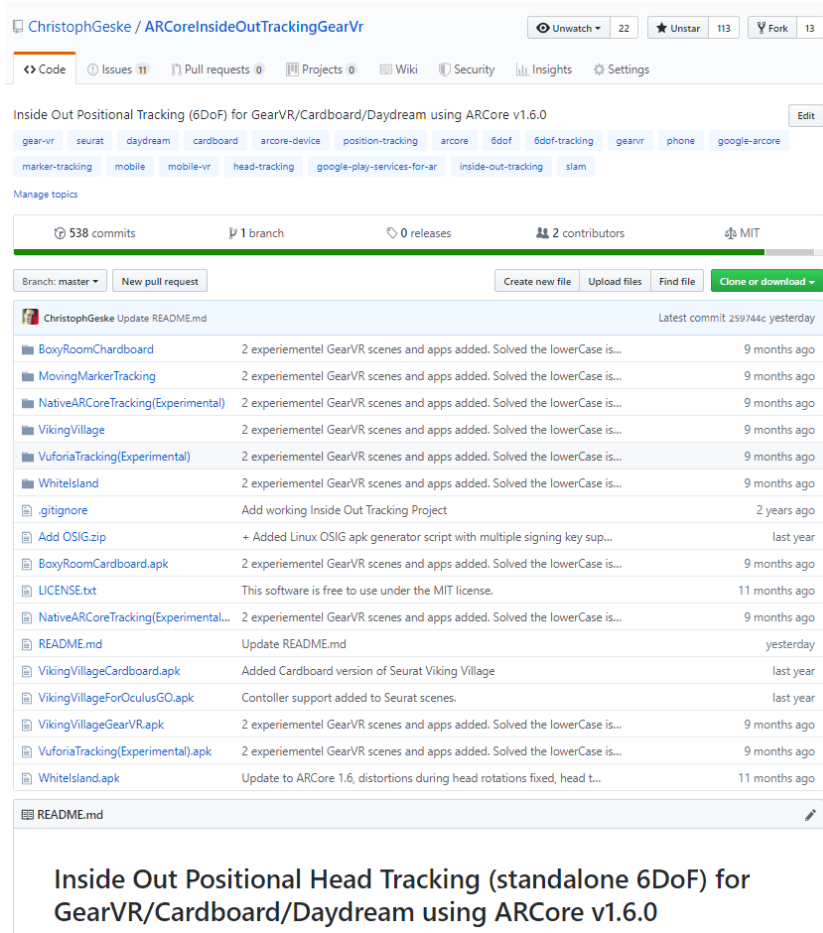


Figure 25: GitHub page showing an already released version of the tracking software presented in this thesis.

References

- [1] Anthes, Christoph, et al. "State of the art of virtual reality technology." 2016 IEEE Aerospace Conference. IEEE, 2016.
- [2] Arth, Clemens, et al. "The history of mobile augmented reality." arXiv preprint arXiv:1505.01319 2015.
- [3] AlitaQuils's YouTube Video making use of the here developed tracking solution URL: <https://www.youtube.com/watch?v=ov-MBoYOt9A> , Last accessed 31.10.2019.
- [4] Bhandari, Jiwan, Sam Tregillus, and Eelke Folmer. "Legomotion: Scalable walking-based virtual locomotion." Proceedings of the 23rd ACM symposium on virtual reality software and technology. ACM, 2017.

-
- [5] Brooks, Johnell O., et al. "Simulator sickness during driving simulation studies." *Accident Analysis Prevention* 42.3 (2010): 788-796.
- [6] Cortés, Santiago, et al. "ADVIO: An authentic dataset for visual-inertial odometry." *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018. URL: <https://arxiv.org/pdf/1807.09828.pdf>,
Last accessed 31.10.2019.
- [7] Fast-Berglund, Åsa, Liang Gong, and Dan Li. "Testing and validating extended reality (xR) technologies in manufacturing." *Procedia Manufacturing* 25 (2018): 31-38.
- [8] John C. Cole, Patent, Micro-miniature accelerometer, Patent id US4736629A, (1985) <https://patents.google.com/patent/US4736629A/en>
- [9] Johnathon Selstad, GitHub, MathUtilities, 2017
URL: <https://github.com/zalo/MathUtilities>,
Last accessed 04.11.2019.
- [10] Kim, Juno, et al. "Effects of head-display lag on presence in the oculus rift." *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*. ACM, 2018.
- [11] Langbehn, Eike, Paul Lubos, and Frank Steinicke. "Evaluation of locomotion techniques for room-scale vr: Joystick, teleportation, and redirected walking." *Proceedings of the Virtual Reality International Conference-Laval Virtual*. ACM, 2018.
- [12] Lee, Jiwon, Mingyu Kim, and Jinmo Kim. "A study on immersion and VR sickness in walking interaction for immersive virtual reality applications." *Symmetry* 9.5 (2017): 78.
- [13] Mania, Katerina, et al. "Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity." *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*. ACM, 2004.
- [14] Mendez, Roberto Lopez. "Mobile inside-out VR tracking, now available on your phone." *ACM SIGGRAPH 2018 Appy Hour*. ACM, 2018.
- [15] Mendez, Roberto Lopez Blog Post
URL: <https://community.arm.com/developer/tools-software/graphics/b/blog/posts/mobile-inside-out-vr-tracking-now-on-your-phone-with-unity>,
Last accessed 31.10.2019.

-
- [16] Milgram, Paul, et al. "Augmented reality: A class of displays on the reality-virtuality continuum." *Telemanipulator and telepresence technologies*. Vol. 2351. International Society for Optics and Photonics, 1995.
- [17] Mladenov, Boyan, et al. "A Short Review of the SDKs and Wearable Devices to be Used for AR Application for Industrial Working Environment." *Proceedings of the World Congress on Engineering and Computer Science*. Vol. 1. 2018.
- [18] Muth, Eric, et al. "Discussion Panel: Motion Sickness in Virtual Environments." *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 62. No. 1. Sage CA: Los Angeles, CA: SAGE Publications, 2018.
- [19] Niehorster, Diederick C., Li Li, and Markus Lappe. "The accuracy and precision of position and orientation tracking in the HTC vive virtual reality system for scientific research." *i-Perception* 8.3 (2017): 2041669517708205.
- [20] Nowacki, Paweł, and Marek Woda. "Capabilities of ARCore and ARKit Platforms for AR/VR Applications." *International Conference on Dependability and Complex Systems*. Springer, Cham, 2019.
- [21] Parsons, Thomas D., et al. "Virtual reality in paediatric rehabilitation: a review." *Developmental neurorehabilitation* 12.4 (2009): 224-238.
- [22] Powell, Wendy, et al. "Getting around in google cardboard—exploring navigation preferences with low-cost mobile VR." *2016 IEEE 2nd Workshop on Everyday Virtual Reality (WEVR)*. IEEE, 2016.
- [23] Primate Labs Inc., Samsung Galaxy Note 8 Benchmarks - Geekbench Browser, URL: https://browser.geekbench.com/android_devices/387, Last accessed 04.11.2019
- [24] Raaen, Kjetil, and Ivar Kjellmo. "Measuring latency in virtual reality systems." *International Conference on Entertainment Computing*. Springer, Cham, 2015.
- [25] Roche, Kayla, Stephen Liu, and Steven Siegel. "The effects of virtual reality on mental wellness: A." *Ment Health* 14 (2019): 811-818.
- [26] Yousif, Khalid, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. "An overview to visual odometry and visual SLAM: Applications to mobile robotics." *Intelligent Industrial Systems* 1.4 (2015): 289-311.
- [27] GitHub, Inc., ARCore Unity SD Issue 604K
URL: <https://github.com/google-ar/arcore-android-sdk/issues/604>,
Last accessed 04.11.2019.