TU WIEN Informatics

Foot Tracking in Virtual Reality

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Media and Human-Centered Computing

eingereicht von

Alexander Bayer, BSc
Matrikelnummer 00726255

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Hannes Kaufmann
Mitwirkung: Projektass. Dipl.-Ing. Dr.techn. Christian Schönauer

Wien, 14. Oktober 2021

Alexander Bayer	Hannes Kaufmann

TU WIEN Informatics

Foot Tracking in Virtual Reality

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media and Human-Centered Computing

by

Alexander Bayer, BSc
Registration Number 00726255

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Hannes Kaufmann
Assistance: Projektass. Dipl.-Ing. Dr.techn. Christian Schönauer

Vienna, 14th October, 2021

Alexander Bayer	Hannes Kaufmann

Erklärung zur Verfassung der Arbeit

Alexander Bayer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 14. Oktober 2021

Alexander Bayer

Kurzfassung

Die Visualisierung der Gliedmaßen in Virtual Reality (VR) Umgebungen erlaubt den Benutzern eine höhere Immersion und gibt ihnen auch mehr Vertrauen in ihre eigenen Bewegungen. Leider werden diese Visualisierungen oft weg gelassen. Ein Grund liegt darin, dass große VR Umgebungen und solche mit mehreren Benutzern schwer mit sogenannten „outside-in“ Trackingmethoden zu realisieren sind, aufgrund von einer möglichen Verdeckung der Sensoren. Ein anderer Grund ist, dass Entwickler nicht ihre ohnehin kleine Benutzergruppe noch mehr eingrenzen wollen, indem sie ihnen teure Zusatzhardware aufzwingen, die so viel kostet wie normale Hand-Controller aber nur in einer Handvoll von Applikationen Verwendung finden.

Diese Diplomarbeit will diesem Problem Einhalt gebieten indem eine einfache Trackingmethode vorgestellt wird, die nur eine Kopfposition von außen erhält und daher sowohl für „outside-in“ als auch „inside-out“ Trackingmethoden geeignet ist. Das System wird mittels einer RGB-Tiefenkamera realisiert, die auf dem VR Headset montiert wird. Fiducial-Marker- und Tiefen-Tracking sowie Inertialsensoren werden gemeinsam zur Abfrage der Fußposition verwendet. Diese, voneinander unabhängigen Systeme, werden dann zusammengeführt um die Vorteile der einzelnen Methoden zu vereinen. Die so erhaltenen Fußpositionen werden dann verwendet um einen virtuellen Avatar mithilfe von inverser Kinematik zu animieren.

Abstract

The visualisation of limbs in Virtual Reality (VR) helps to get a better immersion in the virtual world and it creates better confidence in movement. Sadly a lot of VR applications omit the visualisation of limbs. One reason lies in technical difficulties with bigger scale VR environments and multi-user VR environments where you can not rely on outside-in tracking methods because of the size and possible occlusion that hinders accurate tracking data. Another reason is that developers do not want to exclude parts of their already small user base by demanding special hardware for foot tracking that costs as much as the hand controllers but is only usable in a small number of applications.

This thesis tackles these problems by generating a lightweight tracking system that only relies on the correct tracking of the head position so that either inside-out or outside-in tracking can be used with it. To achieve this, a RGB depth camera is mounted on the VR headset. A combination of fiducial marker tracking, depth tracking and inertial measurement units (IMUs) are used to track the user's feet. These individual tracking signals are then fused to one signal that combines the advantages of the single tracking systems. This tracking information can then be used to animate the feet of a virtual avatar with an Inverse Kinematics (IK) algorithm.

Contents

CHAPTER 1

Introduction

1.1 Motivation

As Virtual Reality (VR) equipment gets cheaper and more widespread on the consumer market, more people get a chance to immerse themselves in VR environments. Products like HTC Vive and the Oculus Rift are among the first examples for affordable VR solutions that are still technically advanced enough to deliver an immerse VR experience. A lot of factors are important for a good experience in VR. Some of them are already in place, like removing latency and getting screen resolutions where you can't distinguish every pixel. These advancements in hardware help gradually in smoothing out these factors. This also minimises the risk of motion sickness, because the experience feels more real.

Another factor for a better VR immersion can be found in the VR software itself. Adding a virtual avatar for the user that represents his real life body adds a lot to the immersion and the natural movement of the user. The animation of arms and hands are relatively easy because there are already controllers in the user's hands that are tracked by the system. So for the animation a simple reverse kinematic algorithm can be used. But animating the user's feet is not as trivial.

1.2 Problem Statement

In a lot of modern VR games and applications, the representation of feet is omitted. The reason for this is most of the times that additional hardware is needed to get tracking for the feet. This additional hardware is an extra cost for the consumer that is as high as the standard controllers for the hands, but with a lot less games and applications that support them. So to not exclude some parts of the already small user base, most developers omit the tracking of the feet. The hardware that is already available, like the

HTC Vive Tracker, relies on additional hardware in the room, so called base stations, for the tracking signal. Because of hardware limitations, the usage of outside-in tracking with base stations is not possible in bigger VR environments. The reason for that is the maximum distance between the base stations that limits the size of the trackable space. So a tracking device is needed that also functions with inside-out tracking. This kind of setup is also often used for multi-user VR environments because of the possibility of occlusion that hinders reliable tracking data with inside-out tracking methods.

1.3 Aim of the Work

In this thesis, I want to develop and evaluate a lightweight method for foot tracking with an inside-out tracking system that does not rely on any stationary hardware in the room. To be independent from the tracking system, the only tracking point that should be available to this system is the head position. A RGB-Depth camera will be mounted on the VR headset to enable tracking of the feet through fiducial marker tracking. If the signal from the marker tracking is lost because of partial occlusion of the markers, an approximation of the foot position should be found with the help of the depth camera. In case fiducial marker tracking is lost completely, inertial sensors, mounted on the feet, should take over and approximate the foot position. To get better overall tracking data, all three tracking methods should get fused if they are available. Using the position of the feet, the program can then animate the feet of a virtual avatar using reverse kinematic algorithms. Finally, I will evaluate the developed system and its effect on immersion and compare the performance and usability against the state of the art.

1.4 Methodology and Approach

The first step is to get a state of the art overview of the topics needed to create a prototype. The first topic is inertial navigation that is needed to get a reliable position signal from the inertial sensors that are mounted on the user's ankles. The second topic, sensor fusion, shows methods on how to fuse the tracking signals from different sources to one reliable signal. The topic on inverse kinematic has some of the algorithms to animate the feet as a whole, especially the position of the knees. The last topic about VR immersion shows the importance of feet for a better experience in the VR world.

The setup for the prototype is as follows: For the VR environment a HTC Vive is used. The fact that this system uses outside-in tracking instead of inside-out tracking is irrelevant for this thesis because only the head position and rotation is used from the HTC Vive. So this setup can easily be used with an inside-out tracking environment as well. An Intel Realsense Depth Camera D435 (Intel Realsense) is mounted on the headset of the HTC Vive pointing downwards to the feet. Two fiducial markers are mounted on the tip of the user's toes as well as two Bosch BNO080 inertial sensors that are mounted somewhere at the side of the user's ankles. With this setup in place, the software uses the fiducial marker tracking as it is reliable but a bit jittery and slow source for the foot

position. When the fiducial marker tracking is lost because of occlusion, then the values of the depth camera are used to approximate the marker position with the previously known depth values. The position data of the inertial sensors tends to drift a lot over time but it has a high refresh rate. Therefore, it generates a smooth position value over time. All of the three advantages of the three signals are combined and the disadvantages are cancelled out by fusing the signals. So, the final position value should be reliable and smooth over time. This value is used to animate the virtual avatar in the Unity engine using a reverse kinematic algorithm.

To evaluate the prototype, it was compared to the HTC Vive Tracker that uses an outside-in approach to track the feet. Values like the accuracy of the position signal, the jitter of the signal, the drift and the overall latency of the system got evaluated.

A user study was conducted where users tested the implemented prototype by doing little movements. Movements like single steps in different directions and movements where an occlusion of the feet is possible were done by the users. The performance of the prototype was then compared to the HTC Vive Tracker. This user study gave more real life data than the technical evaluation. Due to the COVID-19 pandemic, the number of participants was limited to guarantee the safety of the users.

1.5 Structure of the Work

In chapter 2 the research about the parts of the tracking system are presented. This contains literature about VR immersion, inertial navigation system (INS), fiducial marker tracking, depth camera tracking, sensor fusion and Inverse Kinematics (IK). Chapter 3 shows the overall design of the system and its used algorithms. The implementation chapter 4 goes into detail about the implementation of the system itself and its used libraries. In chapter 5 the system is tested against the HTC Vive tracking system, and a user study is done. In the last chapter, chapter 6, the proposed system is summarised and an outlook to future work is given.

CHAPTER 2

State of the Art

In this chapter, I present the fundamental technologies needed for this research project. First I want to present the impact of virtual avatars and especially virtual feet to the immersion in a Virtual Reality (VR) world. The second part about inertial navigation contains information about how to use inertial sensors for position estimation and how to detect and minimise errors. Next I want to give an overview on fiducial marker tracking methods and the usage of a depth sensor. Then I present some algorithms to fuse signals from different sources to one signal that combine the advantages of its sources and minimises its disadvantage. The section about inverse kinematic contains algorithms to calculate the joints of a kinematic chain when start and end position are predetermined.

2.1 VR Immersion

To find the importance of foot tracking for the immersion in VR, we have to first look at the definition of immersion. The immersion can be described as the Sense of Embodiment (SoE). Fribourg et al. describe the SoE as a composition of 3 parts: appearance, control and point of view [FALH20].

The appearance of the avatar influences the Sense of Ownership of the user. The structure of the virtual body, the shape and dimensions of the body parts and the render method all contribute to the avatars realism. The Sense of Ownership gets higher if the avatar is modelled after the user's real body and clothes. A possible approach to realistic avatar models is the use of 3D scanning, although this requires complex 3D capture equipment.

The Sense of Agency is impacted by the control of the avatar. The actions the avatar can perform are judged by the user with several factors. They check if the avatar can do what they want it to do and if their actions are related to the actions of their avatar. Animation techniques like Inverse Kinematics (IK) can impact the Sense of Agency with their more realistic avatar movement.

And lastly, the point of view represents the spatial relationship of the user and its virtual avatar. The placement of the point of view alters the Sense of Self-Location.

These three factors of SoE are deeply correlated. For example, the usage of a less detailed hand model for the avatar would reduce the appearance and therefore the Sense of Ownership but it also can increase the control of the avatar because the user would not overestimate the functionalities of its limbs. Another example would be that extra body parts would get a higher Sense of Ownership by the user if they actually have control over this new body part. The exact impact of these three components on SoE can not be easily described. That is why a lot of studies only focus on one subcomponent at a time and describe its impact on the SoE as a whole.

Some studies tried to show the importance of foot tracking for VR immersion. Kosmalla et al. made a user study to show the impact of virtual hands and feet for VR climbing [KZT+20]. The users had to test a virtual climbing wall with different types of limb visualisations: With no limbs at all, with feet only, with hands only and with both, feet and hands at the same time. The visualisation of both, hands and feet, was of course the preferred visualisation by the users, but on the second place was the visualisation of the feet alone. The users said that feet visualisation was more important for the task of climbing than the visualisation of the hands.

A study by Pan and Steed tests the advantages of foot tracking when solving jigsaw puzzles where the tiles are scattered around the room and the user had to move back and forth from the puzzle to the tiles [PS19]. They had 3 types of participants: one without virtual avatars at all, one with a virtual avatar but without feet and one with a virtual avatar with feet. Participants with a virtual avatar where faster at solving the puzzle compared to participants without a virtual avatar. Participants with virtual feet where not faster at solving the puzzle, compared to the participants without virtual feet, but they had better foot placement and therefore could avoid obstacles more easily. Also participants with tracked feet felt more immersed in the virtual world than the participants without tracked feet.

There are also other VR use cases where foot tracking is indirectly needed to create better immersion. Boletsis and Cedergren compared three VR movement methods to each other: Walking-in-place (WIP), Controller/joystick and Teleportation [BC19]. WIP, the method that uses foot tracking to generate movement in the VR environment as long as the user is moving in place, has the highest immersion rate of the three tested methods. Sikström et al. present a within-subjects study where they are testing different audio feedback when walking on a virtual bridge [SNdS16]. There are two types of bridges in their study, one that looks perfectly safe and one that looks damaged and unsafe. The user not only sees the bridge they are standing on, but also gets audio cues with every step. This is again only possible with foot tracking.

Foot tracking enables the usage of the feet for use cases like rehabilitation and training in an VR environment. But a lot of times somewhat bulky hardware is used for these applications. For example Tierney et al. show VR can be used in gate rehabilitation to

better serve the needs of the patient [TCG+07]. They are using a treadmill to detect the movement of the user. They claim to use relatively inexpensive and unobtrusive hardware. With using a more compact foot tracking solution, this project could be even easier to use.

2.2 Inertial Navigation

An inertial navigation system (INS) uses accelerometers and gyroscopes for position, orientation and velocity calculations that depend on formerly known values. The work by Woodman gives an overview of the techniques and hardware needed for such an INS [Woo07]. There are different types of gyroscopes and accelerometers but only the micro-electromechanical systems (MEMS) variants are small enough and cheap enough to be feasible for inertial navigation in our context. The problem with this smaller and cheaper hardware is that it is not as accurate as its mechanical or optical counterparts. These errors get worse when double integrating the signals to get the position data. When we classify the errors, we could then set countermeasures to minimize the impact on the position data. According to Hussen and Jleta there are five main types of errors: [HJ15]

- **Angle or Velocity Random Walk** occurs because of noise that fluctuates at a higher frequency than the sample rate of the sensor. This noise becomes noticeable as white-noise. The angle random walks standard deviation grows with the square root of time, and the velocity random walks standard deviation grows proportionally to time to the power of three halves.
- **Rate Random Walk** has an uncertain origin. It is random noise that has the characteristics of Brownian noise.
- **Bias Instability** arises from components that are susceptible to random flickering. This flicker noise is mainly observable at lower frequencies as it is overshadowed by white noise at high frequencies.
- **Quantization Noise** is the error that is introduced because of the conversion from analogue data to digital data. It arises from the differences from the actual amplitude of the sampled points and the resolution of the digital values.
- **Drift Rate Ramp** is a deterministic error that represents a slow and monotonic change of the signal over a long time period.

These 5 main types of errors can be measured with the Allan variance analysis. This is done by fixing the inertial measurement unit (IMU) in place and collecting the data for several hours. In Figure 2.1 Hussen and Jleta show where the errors can be located in the results plot of the Allan variance [HJ15].

To get the position estimation from the IMU data in an ideal environment without any errors, one has to subtract the gravity from the acceleration data with the help of the

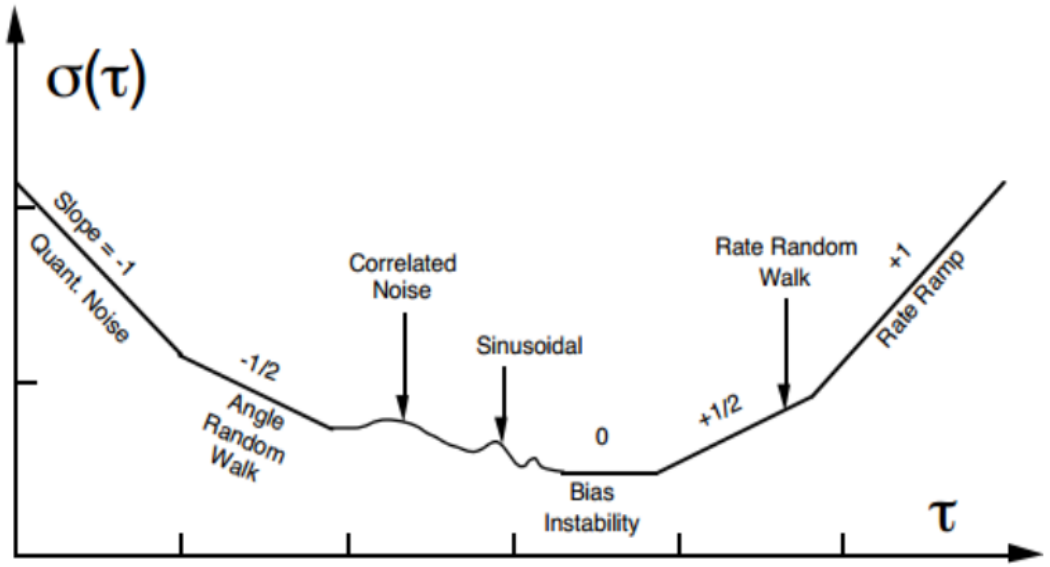

Figure 2.1: A sample plot of the Allan variance results
Source: [HJ15]

orientation data and then the remaining acceleration needs to be integrated twice to get velocity and displacement [Woo07]. This is easily done with the rectangular rule as shown in equation 2.1 and equation 2.2 [Bak15a].

$$v_{n+1} = v_n + a_n * dt \tag{2.1}$$

$$p_{n+1} = p_n + v_n * dt \tag{2.2}$$

When used in real world environment the earlier described errors from the IMU grow quadratically with time because of the double integration. The biggest source of error is the fluctuation of the orientation which leads to acceleration from gravitation leaking into the acceleration measurements. Position estimation with IMU alone leads to a drift of hundreds of meters after less than a minute [Woo07].

A way to counter this gigantic drift is by not using the position estimation from the IMU as is. It is possible to use the accelerometer data to detect a step of the user and with that estimate the user's position. Abadleh et al. propose a novel step detection method where the error is fixed and will not grow with the travelled distance [AAAA17]. The steps are being detected by peaks on the accelerometers acceleration data. Peaks that are falsely detected as steps are filtered out. Every detected step is then categorised as shorter, equal or longer than the average step length. The moved distance is then calculated by

adding together all the steps. The variable step length allows for better results when the user changes its walking style. Yang and Huang use the basic principles for step detection as Abadleh et al. but they add the usage of gyroscope and magnetometer so that the exact placement of the device on the user's body is not necessary any more [YH15].

2.3 Fiducial Marker Tracking

Fiducial marker tracking works on the premise of finding reference points of an object in an image and estimate its position relative to the camera position and orientation. To achieve that, the camera has to be calibrated so that possible distortions from the lenses are taken into account as well as to get a sense of the size of the objects seen in the image. Daftry et al. give an overview of current calibration methods and propose their own method to simplify the process for the user [DMWB13]. The process of calibrating the camera is well defined. There are a lot of different methods of calibration, each with their own advantages and drawbacks. The most common method to calibrate the camera is with a chequerboard printed on a piece of paper with known dimensions. This method requires only a few images of the chequerboard and is very robust. The drawback is that because of the usage of the chequerboard the algorithm can not detect correspondence of the detected features between views. The proposed method by Daftry et al. tries to overcome this drawback by using fiducial markers for the calibration process. There are three major problems with the chequerboard methods. The precision of the edge detection is susceptible to lens distortion effects, which results in a lot of pictures that are needed for a good camera calibration. The second problem is the sensibility to partial occlusion of the chequerboard. The uncontrolled environments where user calibrate their camera can result in bad illumination that results in bad detection of the feature points. The last problem is that traditional methods are bad for cameras with a wide viewing angle. Because of the constraint to detect the whole chequerboard image, the feature points are not well distributed over the image. The method by Daftry et al. eliminates these problems by using fiducial markers that are printed on several sheets of paper and placed on the floor in an approximate grid placement. So not all markers have to be visible in the final image and every feature point is unique. This means the feature points can be detected from different views. This method also has a better distribution of the markers, because the user does not have to check that every marker is in the image; it is sufficient when the image contains most of the markers.

When the camera is properly calibrated and the image is deskewed, the detection of the marker in the image takes place. Kato and Billinghurst describe a general method of how to detect the marker and get its transformation matrix [KB99]. First the image gets segmented into regions with a threshold function. Then the algorithm tries to fit four line segments around the regions to find the marker regions. After a normalisation of the found marker regions, a template matching with the known pattern from a dictionary is done. The normalisation is done with a perspective transformation. With a estimated transformation matrix, the marker can be transformed from image space to camera space. This transformation includes some errors that can be reduced by optimising the

rotation components. It would be possible to optimise all six independent variables from the transformation matrix, but due to computational costs the algorithm limits this optimisation to rotation.

For robust marker tracking, the fiducial marker have to be reliable. In the work of Kahn et al. they show the importance of robust markers and how to create them [KUY+18]. A reliable marker should have the following properties:

- it should be distinct from the context background.
- it should be unique in the marker library.
- it should be passive (e.g. not electronically enhanced).
- it should be detectable in a fast manner.
- it should be robust in low and high light conditions.
- it should be detectable in noisy environments.

It is a big challenge in Augmented Reality (AR) applications to meet all these criteria. That is why Kahn et al. propose a method to rate the markers reliability and to take steps to strengthen the robustness of the marker. For every marker they test for four criteria: the black to white ratio of the marker; the information complexity from black objects in the white area; the edge sharpness which is the abruptness of intensity change; and lastly the inter-marker confusion from the markers currently in the marker library. Their experiments show that the measurements to strengthen the marker design increased the correct identifications of the enhanced marker in contrast to the original marker. It also reduced the false detections of markers.

2.4 Depth Camera

A depth camera adds an additional layer of information to the marker tracking. Langmann et al. provide an overview of the used technologies of depth cameras and they also compare the performance of consumer products with professional ones [LHL12]. The classical depth cameras measure the Time-of-Flight of a light source. There are two methods: A pulsating light where the time is measured when the light hits an object. This method has to calculate very short time intervals, especially when the object is near to the camera. The other method is a continuous wave modulation approach where only a phase shift between the emitted and received light is measured. This phase shift directly corresponds to the Time-of-Flight and therefore to the depth of a pixel. For a long time these sensors had a low resolution or could not provide affordably high frame rates for real time image analysis. In contrast to these classical approaches, cheaper consumer products like the Microsoft Kinect had a totally different approach: An irregular point pattern is projected into the room with an infrared laser LED. The depth of the objects is calculated by the displacements of the known dot pattern. When the displaced dots are identified, the distance to the object can be triangulated.

There exist a lot of tracking approaches for objects that use depth cameras. For example Akkaladevi et al. propose a real-time tracking method for rigid objects using only depth

data [AAFP16]. Their tracking approach uses only the depth data from a RGB depth camera. The algorithm combines a slow global object localisation algorithm with a fast local object tracking algorithm. RANdomized Global Object localozation (RANGO) is the random sampling algorithm that is used for the global localisation of the object in the depth data. The algorithm approximates the scene with a 3D grid and every voxel of the 3D grid is hashed into a single 32bit number. This 3D voxel grid is used to verify candidate transformations. A filtering approach is used to sort out all candidate solutions to determine the best fitting candidate solution. The local tracking algorithm used is a multi-forest tracking algorithm that is modified to already return good results with only a training set of six random forests. To get better results in the tracking approach, the movement of the last frame is used as a starting guess. This allows for better tracking when the objects move fast between two frames. The tracking framework uses the two algorithms as follows: Initially the global object localisation is used with the input depth data to find the object. Then the fast object tracking is used until tracking of the object is lost. If the tracking is lost the algorithm reverts back to the global object localisation and repeats these steps. The framework is robust against occlusion and capable of real-time usage.

Another tracking approach with a depth camera is presented by Zhou and Koltun [QK15]. They extract contour cues from noisy and incomplete depth inputs to better track smooth surfaces that are normally prone to drift. They limit their approach to depth camera input without colour image to also work at low lighting and be more versatile. The foundation of the approach is that occluding contours have a known normal that is orthogonal to the view ray. The framework is build upon KinectFusion. From the received depth data, a volumetric representation of the scene is generated. Then the occluding contours in the depth image are detected, but first the depth image is prepared to fill in missing depth information. In the next step, the corresponding contours in the volumetric representation of the scene are searched. To identify the corresponding contours, the normals of the points of the synthesised scene have to be estimated. They evaluated their method against KinectFusion and an extension of KinectFusion. Their approach was significantly better than the output of prior techniques on various benchmark data. This approach is also better against some approaches that also use colour images as a second data source.

2.5 Sensor Fusion

To get better results from the position estimation with IMU a filtering algorithm is needed [KHS17]. Especially a combination of more than one source of information improves the position estimation significantly. With the Kalman Filter (KF) it is possible to fuse more than one source of data by describing the underlying system. This allows it to combine the advantages of the sensors while minimizing their disadvantages. The KF is an optimal linear estimator which means that it will return an optimal estimation of the system when certain assumptions are taken into account [May79]. The algorithm uses its knowledge of the system with its dynamics, the description of the system noises, errors and uncertainties, as well as any initial information of the variables to estimate the

current value. The system is described by linear functions that describe how to get from one state to another. Noises, errors, and uncertainties are represented in these functions. The KF combines all this information that is given to it to estimate the desired variable while statistically minimizing the errors. The assumptions made by the KF are that the underlying system is linear, that the noise functions has to be some type of white noise and has to be normally distributed.

Because most relevant systems are not linear in its transitions, the Extended Kalman Filter (EKF) was implemented [WB06]. The EKF no longer has the restriction that its transition functions have to be linear but the assumptions for the noise functions are the same as with the normal KF. The algorithm linearises about the current mean and covariance. The biggest flaw of it is that distributions of the random variables are no longer normally distributed because of the non linear transformation functions. Contrary to the KF that gets the optimal solution for its linear system, the EKF only approximates the optimality of Bayes' rule by linearisation.

The EKF and some other KF variations are the de facto standard filter for pose estimation [HKS+15]. To use EKF for position estimation, the transition states of the algorithm would be acceleration, velocity and position with a modelled noise function for every transition. The algorithm performs "time update" where the next estimation is calculated and "measurement update" where the next sensor data is fed to the algorithm. But this alone is not enough to get a reliable position estimation for inertial navigation that is accurate enough for a longer time period. With sensor fusion it is possible to merge the fast but drifting IMU with another sensor that is more reliable over time. With the proposed EKF model by He et al., the fusion of fiducial marker tracking and IMU tracking allows to get a reliable pose estimation with long term partial occlusion and a relatively reliable pose estimation with short term total occlusion. This also results in a position estimation with a higher update rate than with the optical system alone. The IMU gets bias corrected on the fly with the information provided by the fiducial marker tracking system.

2.6 Inverse Kinematic

To calculate the foot position when the location and rotation of knees and hips are known, is a relatively trivial problem solvable with Forward Kinematics. The hard part is the IK problem where the opposite is wanted. You know the starting point and the endpoint of a kinematic chain, consisting of rigid bodies connected by joints, and want to know the positions of all joints positioned between start and end. In our case this would be the known head position and foot positions and we want to know the positions of the knees and the hip.

Aristidou and Lasenby show various methods to solve the IK problem [AL11]. The first family of IK solvers is a numerical approach that uses the Jacobian matrix to find a linear approximation of the IK problem. These approaches provide a smooth and realistic posture but the problem is that they have a relatively high computational complexity.

Although there exist some approaches which are improved in terms of computational complexity, they are still a bit slow for real time applications. The second family of IK solvers is a minimisation problem based on the Newton method. They generate smooth motion without discontinuities but are hard to implement and are computational heavy for every iteration of the algorithm. The next family of IK solvers is based on the Cyclic Coordinate Descent algorithm. These solvers are fast but can lead to unrealistic animations even if joint constraints are defined. Another difficulty of these solvers are that they are hard to extend to kinematic chains with multiple end effectors because they are designed to handle only serial chains. The next approach is a Sequential Monte Carlo Method which is a statistical method that performs well but is again computational heavy. An algorithm that uses the cosine rule to calculate each joint angle is very fast because it can solve the problem in only one iteration but it generates very unnatural poses.

Forward And Backward Reaching Inverse Kinematics (FABRIK), the approach implemented by Aristidou and Lasenby, is a lightweight and fast method to the IK problem that produces relatively realistic poses [AL11]. FABRIK uses a simple approach that only has to calculate point positions on a line. The algorithm sets the current endpoint of the kinematic chain to the new desired endpoint and drags all other points beneath it with the constraint to not exceed the length of the edges. Going back and forth, the algorithm stops when the endpoints are close enough to the desired positions. Figure 2.2 shows a complete iteration of the FABRIK algorithm: Point t is the new wanted endpoint of the chain. The current endpoint P4 is moved to t, forming P4'. Then P3 is dragged along on the line between P3 and P4'. In the last step of the first iteration at (d), the starting point gets dragged too. So the next step is to move the starting point P1 back to its original position and go on with the algorithm, now in the other direction. These steps are repeated until the endpoint is close enough to the desired point t.

When using an IK algorithm on humanoid figures for motion tracking, it has to be possible that the algorithm can work with multiple end effectors, like fingers. FABRIK can handle such kinematic chains by splitting the algorithm into two stages. In the first stage, the algorithm is applied as usual but with every end effector separately and only to the next sub-base. This will create as many positions for the sub-bases, as there are end effectors. A sub-base is a joint of the kinetic chain with more than two links and therefore the joint that splits the kinetic chain into multiple ends. The algorithm uses the centroid of all generated sub-base positions as the new sub-base position. From there the algorithm goes on from the new sub-base position back to the root node, or possible additional sub-bases. After that follows stage two, where the algorithm is applied from the root node back to the sub-base and from there to each end effector separately. The algorithm goes on with these two stages until all end effectors are close enough to its desired targets.

To get even more realistic postures with FABRIK, it is possible to restrict the movement and rotation of the joints to imitate humanoid joints. A single joint can normally be represented by 2 rotations: a joint rotation and a joint orientation. This means that a

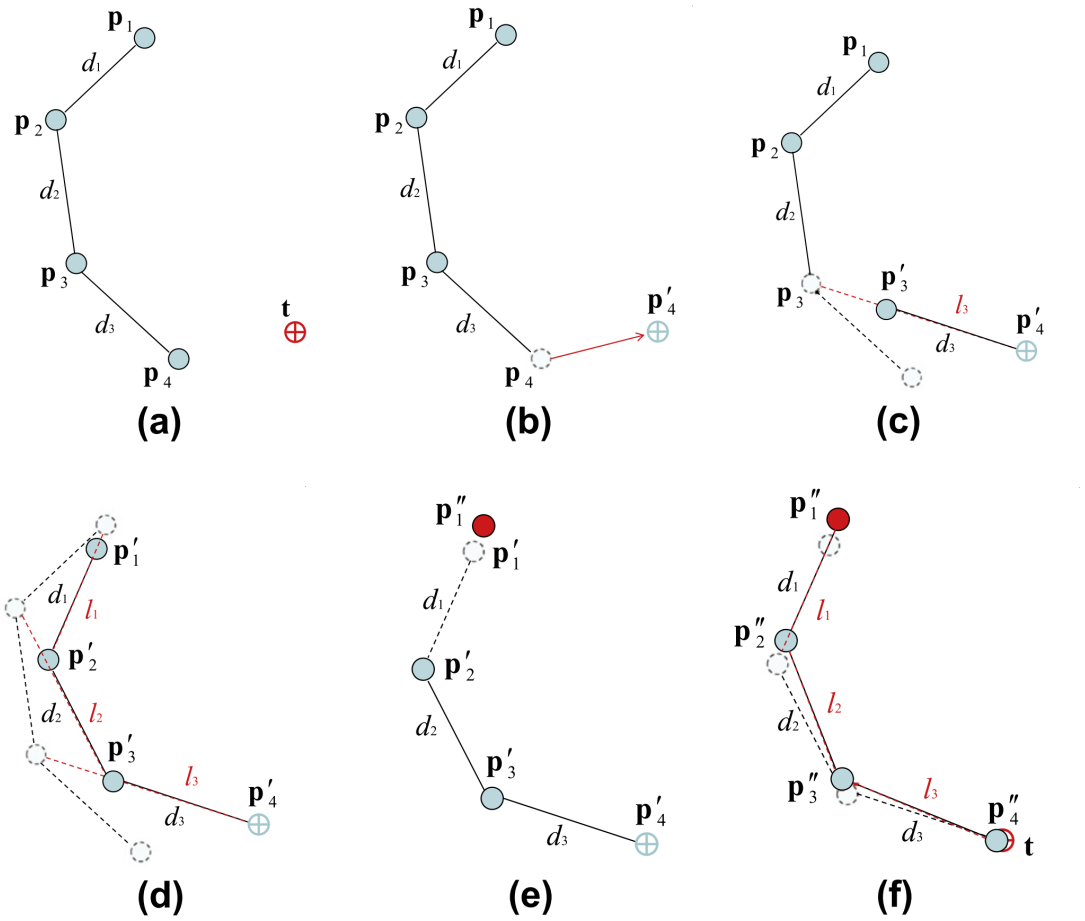

Figure 2.2: Full iteration of FABRIK
Source: [AL11]

enforcement of joint restrictions can be done in two stages. With FABRIK this is done by checking the validity of the joints after every step in the algorithm. The 3D problem is here simplified to a 2D Problem which decreases the complexity and the processing time.

CHAPTER 3

Methodology

In this chapter, I will present the design and structure of this project. First I will give an overview of the project components and their interactions. In the section about inertial navigation, I describe how the inertial navigation system (INS) is structured and which micro-electromechanical systems (MEMS) are used. Then the algorithms used for marker tracking are described. The next section is about the usage of the depth camera and how it should minimise the effect of partial occlusion. The structure of the sensor informations that get fused is part of the next section. Then I describe how Inverse Kinematics (IK) is used to animate an user's avatar.

3.1 System Architecture

The bases of the system architecture are the requirements set by the aim of the work. The most important requirement is that the whole program should be independent of the underlying tracking system used by the Virtual Reality (VR) headset. There are two basic tracking types for VR headsets: inside-out and outside-in tracking. With inside-out tracking, the VR headset contains all the sensors to estimate the position of the head of the user. For outside-in tracking, the sensors that track the VR headset are placed somewhere in the room. These tracking solutions usually come together with devices that are capable of tracking hands or feet from the user. But to be independent from the underlying system used by the VR headset, only the head position is used. The second requirement is that the only output of the program should be position data from feet, knees and hip so that it can work independently from any graphics engine. That means that the rendering of the players avatar is not part of this program. For the overall flow of the program, the system can be divided into four stages. In the first stage, the data from the different sensors is measured. For the second stage the position data is calculated from the different sources of data. In stage three, the position data from the different devices gets fused to combine the advantages from the different tracking methods. With

the last stage, the IK algorithm calculates the positions for the knees and the hip that can later be used to animate the user's avatar.

The components of the program are designed so that they are working mostly independent of each other. This means that components can be swapped relatively easy in the future if I should decide to use, for example, different tracking methods or different algorithms for fusing the position data. The parts of the system are designed as follows:

- **The inertial sensor** that tracks acceleration and rotation of the user's ankle and sends it to the inertial navigation component.
- **The inertial navigation component** calculates the position of the feet from their acceleration and rotation and sends the data to the sensor fusion component. The calculated position is relative to the last known marker position.
- **The VR headset** sends the head position as a reference point to the inertial navigation, marker tracking, depth tracking and IK components. The head position is the only information that is needed from outside of the system.
- **The marker tracking component** tracks the foot marker with the RGB image and calculates the final foot position with the help of the depth image. The calculated positions are then sent to the sensor fusion component. Also the last known marker position is sent to the depth tracking component and the inertial navigation component as a reference point.
- **The RGB depth camera** streams a normal RGB image and a depth image directly to the marker tracking and depth tracking components. The camera itself is mounted on the user's VR headset facing downward to the feet.
- **The depth tracking component** uses the depth image and the last marker position to find the foot marker position by looking for similar shapes at a similar depth value. This prevents the loss of the tracked marker positions with partial occlusion in the RGB image. The position is then sent to the sensor fusion component.
- **The sensor fusion component** combines the information received by the inertial navigation, marker tracking and depth tracking components to generate a more reliable position estimation than one single component. The foot position is then sent to the IK component for pose estimation.
- **The IK component** uses the foot position and head position to estimate the user's pose. The calculated knee and hip position is then sent to a graphics engine to render the user's avatar. This is done outside of this program.

Figure 3.1 visualizes the systems architecture with all its components.

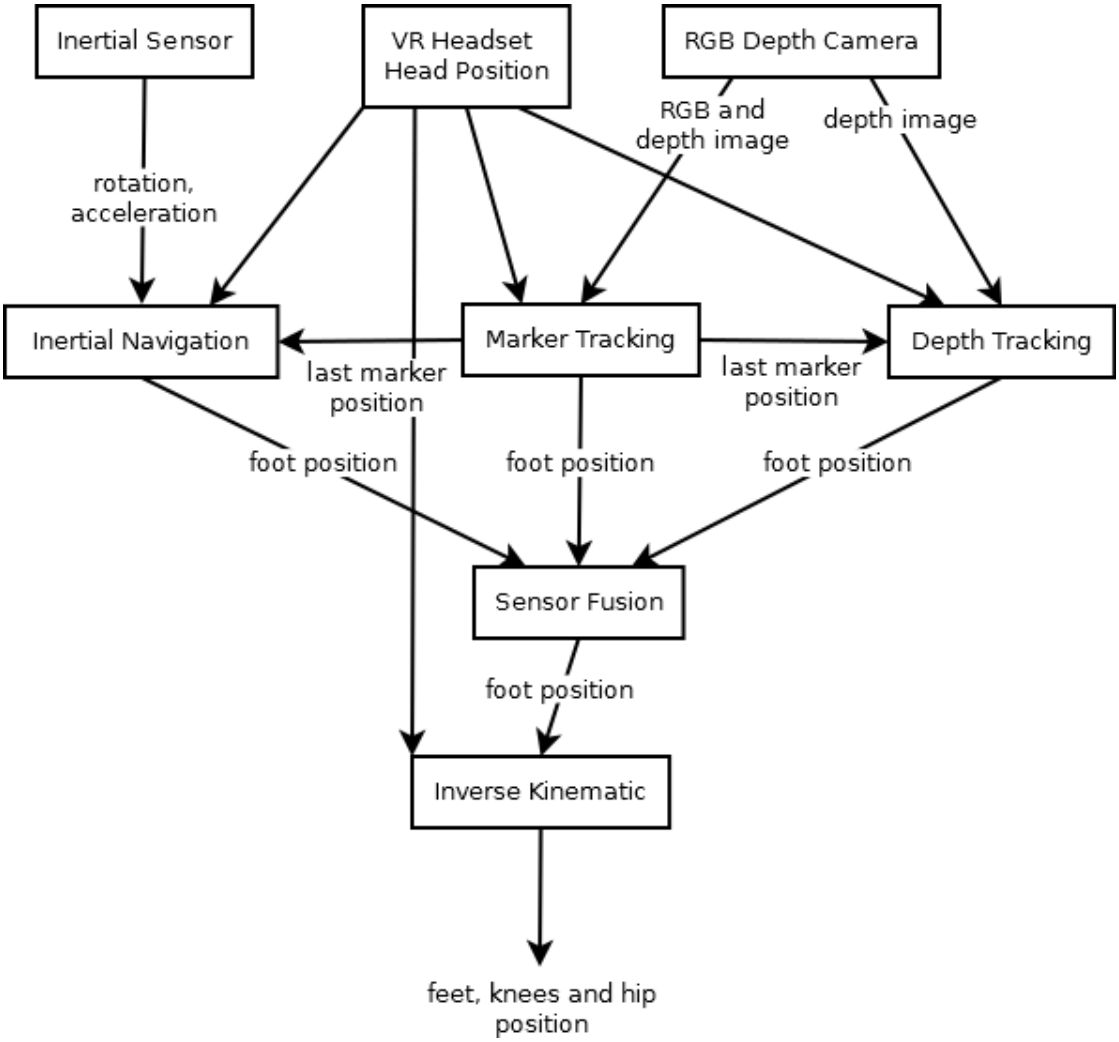

Figure 3.1: The Project Structure

The program is designed as a stand alone library that can be used in any project where foot tracking is needed. When the library is used only the head position has to be fed into it and the foot positions are returned for animation.

3.2 Inertial Navigation

For inertial navigation an inertial measurement unit (IMU) is needed that has a small enough form factor to be mounted on the user's ankle. The IMU should have a gyroscope, accelerometer and a magnetometer. Because the bandwidth from the sensor to the PC is limited, the sensor fusion should already be done on the sensor itself. So less data has to be sent; only quaternions for rotation, vectors for acceleration and timestamps. The

additional magnetometer data is not needed in the software itself.

I had to choose between two IMU that where available to me, the BNO080 development kit for nucleo by CEVA (formerly Hillcrest Labs) and the MetaMotionC by mbientlab. More about which IMU I chose and why follows in the Implementation chapter 4.2. The chosen IMU does not matter here, because both use similar algorithms. They both use their own software to fuse their sensors. The BNO080 uses the MotionEngine by Hillcrest Labs and the MetaMotionC uses the BSX software by Bosch. Both software use variants of the Kalman Filter (KF) for the sensor fusion. The KF algorithm is already covered in the state of the art chapter 2.5.

The IMU sends the final rotation and acceleration data to the inertial navigation component. Now the algorithms 2.1 and 2.2 are used to calculate the velocity and the position of the sensor. For better accuracy, the last known marker position is used as a reference point to correct the fast growing drift of the position estimation. So every time a new marker position comes in, the algorithm uses this marker position as its starting point. Because the performance of this algorithm is very bad when the marker tracking is lost entirely, the algorithm assumes that the sensor is only moving when the acceleration is above a minimum level. This ignores the fact that there is no acceleration when an object moves with a constant velocity. But since the sensor is mounted on the user's feet, a constant velocity is very unlikely and mostly only present when the feet are not moving. Abib did a research of the characteristics of limbs when moving, and in his experiment results, feet had only an acceleration near zero when standing still on the ground [ABI18]. Figure 3.2 shows the curve of acceleration during a normal walk cycle. This proves that ignoring constant velocity has no big impact on accuracy but it helps removing drift when the feet are not moved. The final position data is then saved with its timestamp for usage in the sensor fusion component.

3.3 Fiducial Marker Tracking

Since it is hard to detect feet from a colour or depth image without any context, markers are mounted on the feet of the user. So the marker tracking not only tracks the position of the feet, but also provides context to where the feet are in the colour image and its corresponding depth image. The marker on the foot is internally seen as the same position as the IMU on the user's ankle. This is possible, because the inertial navigation component only calculates relative positions of the feet with the marker position as its base position. The movement between the tip of the feet and the ankle is small enough to be ignored for our use case. This also means that the marker does not have to have the same position on the feet every time. The only constraint is, that the marker has to face towards the RGB depth camera on the user's head.

The fiducial markers used for tracking are ArUco markers based on the works of Garrido-Jurado et al. ([GJnSMCMJ14], [GJnSMCMC16]) and the works of Romero-Ramirez et al. ([RRMSMC18]). ArUco markers are square markers with black outlines and a binary pattern in the middle. The minimum size of the pattern has to be three by three and the

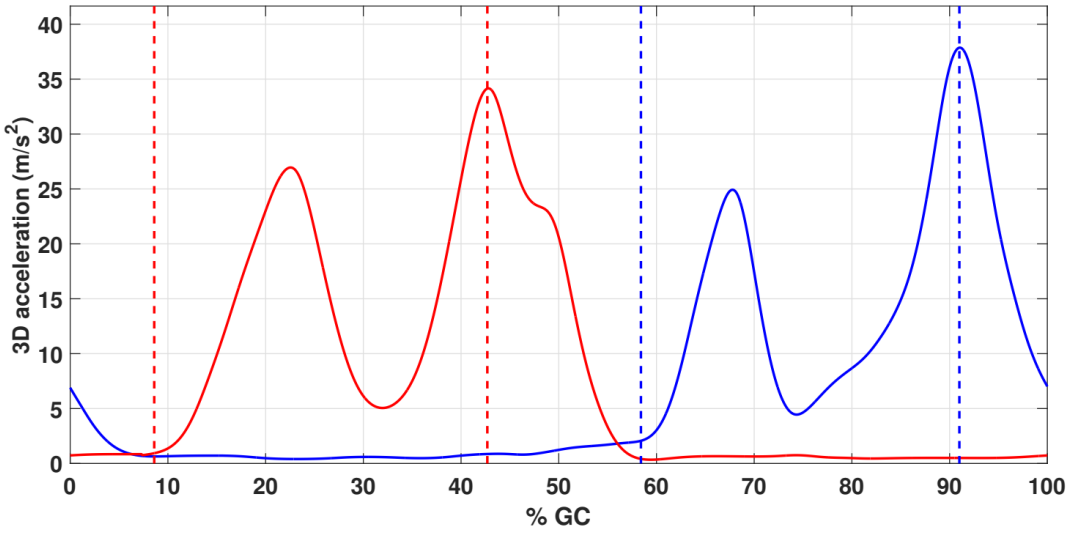

Figure 3.2: The acceleration curve of feet during a walk cycle.
right foot is blue; left foot is red
Source: [ABI18]

maximum size is in theory limitless. The used tracking algorithm works as follows: The RGB image from the camera is used to detect the marker from the known dictionary. When the marker is detected, the position and orientation is calculated. For better depth prediction, the depth camera is used to calculate the Z value of the marker position. This calculated marker position is then sent to the sensor fusion component as well as to the depth camera component as a reference point and to the inertial navigation component to correct the drifting position calculation.

Calculating reliable marker positions from the RGB image is only possible if the camera is calibrated beforehand. The calibration is done with a chequerboard image as described in the state of the art chapter 2.3.

The marker tracking is an essential part of this system. It labels the feet in the RGB image and its corresponding depth image. It also helps to find precise foot positions that are used to correct the drift of the inertial navigation component. The calculated marker position is sent to the sensor fusion component to create the final foot positions.

3.4 Depth Camera

The marker tracking easily loses the marker when it gets occluded. To counter this, the depth camera component takes over and tries to track the marker with the depth image when the fiducial marker tracking gets lost. The algorithm tries to find the marker if either a previous marker tracking position is known or the algorithm itself has not yet lost the marker within the depth image.

The algorithm works as follows: The depth image is filtered into blobs that consists of the old known depth of the marker and a range where the algorithm is looking for the marker. The centres of these generated blobs get calculated and the closest to the old marker position is chosen as the current marker position. This marker position is only valid if it is within a distance threshold to the old marker position. If it is not within this threshold, the marker tracking is lost completely and the algorithm has to wait until the marker is found again through the marker tracking component.

The algorithm is a relatively easy and lightweight method to get an estimated marker position while the marker is partially occluded in the RGB image. It is not necessary to use a more complex tracking method with the depth camera because it is only meant to create a more reliable marker tracking. There is still the inertial navigation component for the case when a total occlusion of the marker happens. The depth cameras calculated marker position is only used for the sensor fusion component if the normal marker tracking is not available. This is because of the inaccuracy of this component in comparison to the marker tracking component as well as the fact, that it does not add new position information to the marker position from the RGB camera. The marker position from the RGB camera can be seen as the ground truth. The depth camera component only has an approximation of this ground truth.

3.5 Sensor Fusion

The sensor fusion is a crucial part of this project. It fuses the signals from two very different sources: the fiducial marker tracking and the tracking with an INS. The advantages of both signals can be combined by fusing them. The fiducial marker tracking has the advantage, that it is highly accurate and can therefore be seen as the ground truth of the current foot position. The disadvantage of the fiducial marker tracking is that it is very slow compared to the signal rate of the IMU. This means that the advantage of the INS is the high data rate of the signal but its disadvantage is the high drift of the calculated position. The signal can drift more than 100 meter in less than a minute. By combining INS and fiducial marker tracking, we can get the high data rate of the IMU corrected by the accuracy of the fiducial marker tracking as one reliable signal.

As described in the state of the art chapter 2.5, the KF or one of its variants are the de facto standard algorithm to fuse different sources to one signal. The used Extended Kalman Filter (EKF) model is based on the work of He et al. [HKS+15]. One cycle of the EKF algorithm consists of a prediction phase and an update phase. In the prediction phase the algorithm estimates the current state variable and its uncertainties. In the update phase the estimate is updated with a new measurement using a weighted average. More weight is given to the measurement or the estimate, depending on their certainty.

The state vector of the EKF consists of the position p, the velocity v and the acceleration a.

$$p = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \quad (3.1) \qquad v = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \quad (3.2) \qquad a = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \quad (3.3)$$

The underlying state transition model is defined by the following functions:

$$\hat{X}_k^- = f(p, v, a) \tag{3.4}$$

$$p_k = p_{k-1} + \Delta t v_{k-1} + \frac{1}{2}\Delta t^2 a_{k-1} + \frac{1}{2}\Delta t^2 w_k \tag{3.5}$$

$$v_k = v_{k-1} + \Delta t a_{k-1} + \Delta t w_k \tag{3.6}$$

$$a_k = a_{k-1} + w_k \tag{3.7}$$

Δt refers to the time that passed since the last calculation step. w_k is the process noise and is assumed to be zero mean Gaussian noise with covariance Q_k.

The corresponding observation model Z_p and Z_a are simple as they directly map the observed position or acceleration value to the transition vector with the function h_p and h_a. They both have their own measurement Noise $v_{p|k}$ and $v_{a|k}$ that is also assumed to be zero mean Gaussian noise with the covariance $R_{p|k}$ and $R_{a|k}$.

The EKF linearises the non linear functions around the current estimate with Jacobian matrices that contain all partial derivatives of the function f, h_p and h_a. These Jacobian matrices are defined as follows:

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}} \tag{3.8}$$

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}} \tag{3.9}$$

$$H_{p[i,j]} = \frac{\partial h_{p[i]}}{\partial x_{[j]}} \tag{3.10}$$

$$H_{a[i,j]} = \frac{\partial h_{a[i]}}{\partial x_{[j]}} \tag{3.11}$$

$$V_{p[i,j]} = \frac{\partial h_{p[i]}}{\partial v_p[j]} \tag{3.12}$$

$$V_{a[i,j]} = \frac{\partial h_{a[i]}}{\partial v_a[j]} \tag{3.13}$$

The final matrices are:

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

$$W = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

$$H_p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.16}$$

$$H_a = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$V_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

$$V_a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

The Kalman a priori covariance estimate is calculated by:

$$P_k{}^- = AP_{k-1}A^T + Q \tag{3.20}$$

The Kalman gain is defined by:

$$\hat{K_k} = \frac{P_k{}^- H^T}{HP_k{}^- H^T + R} \tag{3.21}$$

The last parts of the EKF are the updated a posteriori state estimate $\hat{X}_k$ and the updated a posteriori covariance estimate P_k:

$$\hat{X}_k = \hat{X}_k^- + \hat{K}_k(Z_k - H\hat{X}_k^-) \tag{3.22}$$

$$P_k = (I - K_k H_k)P_k{}^- \tag{3.23}$$

After the EKF fuses the two signals, the newly predicted foot position can be used to animate the user's avatar.

3.6 Inverse Kinematic

The IK component should be able to animate the user's avatar with a relative realistic posture using only the head position and the calculated foot positions. With the head position we can approximate the height of the user because the user will stand on the floor most of the time. With the approximated height, the avatar can be scaled to an appropriate size that represents the user in the best way. The IK algorithm has to position primarily the knees, the hip and the upper body of the user. The arms placement is not relevant for us, because they have too little influence on the placement of the other body parts. The biggest influence the arms have on the body, is the rotation of the upper body that can be ignored for us. Also, the elbow placement is not trivial, as shown by Parger et al. [KZT+20].

A good IK algorithm for our task that is fast and creates realistic postures is Forward And Backward Reaching Inverse Kinematics (FABRIK). It works well with the skeleton of the human body, as it can process multiple end effectors. With modifications it is also possible to limit the movement of single joints to better simulate the human body. FABRIK is discussed in more detail in the state of the art chapter 2.6. For best results the rotation of the feet, that we get from the inertial navigation component, is also taken into account when placing the knees. This allows for better immersion because the approximated knee position will be closer to the user's real knee position.

The skeleton information that is calculated by IK algorithm is then used by the rendering engine to animate the user's avatar.

CHAPTER 4

Implementation

This chapter contains all the information on the implementation of the project. First I give an overview of the used hardware and used libraries in the project. Then I go into the details on the implementation of the components in their separate subchapters.

4.1 System Architecture

The system is implemented as a generic C++ library that is independent of the graphics engine later used for animating the user's avatar. Because of a lack of an easy to use Inverse Kinematics (IK) library, I decided to program the IK part of the program outside of the C++ library in the Unity engine version 2020.3.1f1. Because Unity can not use C++ libraries natively, a C# wrapper class is implemented that exposes the C++ functions to Unity. The main library used for maths calculations and fiducial marker tracking is OpenCV version 3.4.3 [Bra00]. All data received from sensors is converted into the coordinate system that is used by OpenCV, where $-y$ is up, x is right, z is forward and positive rotations are counter clockwise. Figure 4.1 illustrates the coordinate system.

Internally the C++ library starts independent threads for every component and their sensors. The main thread manages the calculated data and waits for library calls to forward the current foot position data. One thread manages the RGB depth Camera with the fiducial marker tracking as well as the tracking with the depth image. For every inertial measurement unit (IMU) one thread is used for the communication with the sensor and one is used to process its acceleration and orientation data. The communication between the threads and the main thread is synchronised by using the Mutex class to lock the data that is read or calculated and prevent simultaneous data access. The locks are divided into groups to not lock all variables at once. The groups consists of independent parts that access the head data, camera data, sensor data, serial communication data and the Extended Kalman Filter (EKF) data.

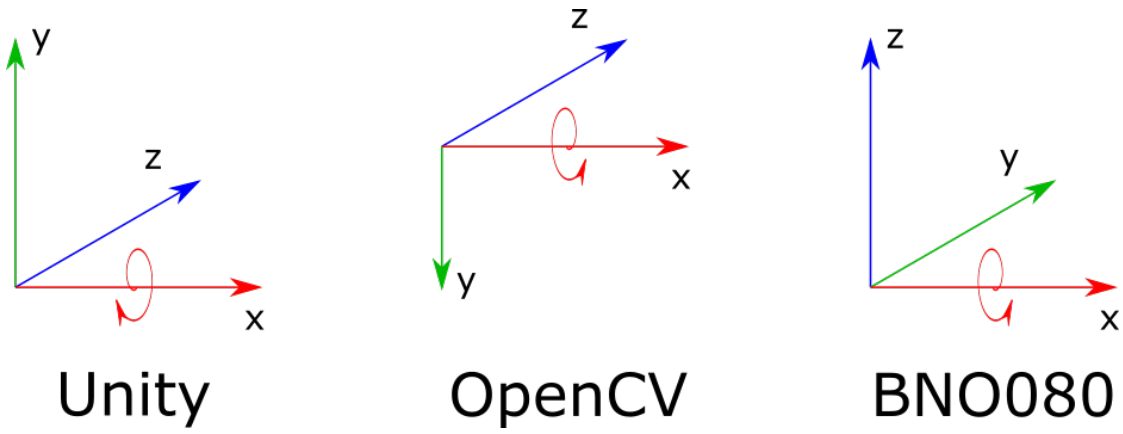

Figure 4.1: Overview of the used coordinate systems by Unity, OpenCV and BNO080

A graphical overview of the system architecture is given in Figure 4.2. It gives a simplified view of the data flow and shows all used libraries.

The usage of the library is as follows:
There are getter and setter methods for returning the foot positions and for feeding the head position and its rotation to the library. Before the library can be used, the user has to set the serial port numbers of the IMU. Then the library can be started with a function call of *StartThread()*. Now the head position has to be provided to the library by setting the current position with periodically calling the setter function. The foot position can be pulled at any time, as it gets updated automatically when a new acceleration value or marker position value is measured. Unity can use the foot positions directly to animate the avatar. The library can be stopped by calling the *StopThread()* function.

4.2 Inertial Navigation

For this project I had to find an inertial sensor that would work well in the context of this lightweight library. In the end I had two candidates that where promising: The BNO080 development kit for nucleo by CEVA and the MetaMotionC by mbientlab. They both have about the same specifications. The BNO080 has a refresh rate of 500 Hz for its accelerometer, 400 Hz for its gyroscope and 100 Hz for its magnetometer. The rotation vector that is already fusing the sensors has a refresh rate of 400 Hz and the gravity corrected linear acceleration has a refresh rate of 400 Hz. The MetaMotionC's sensors only have a high refresh rate when logging the data on the device itself. The refresh rate for logging accelerometer and gyroscope are 800 Hz but for streaming the refresh rate is only 100 Hz. The magnetometer of the MetaMotionC has a refresh rate of 25 Hz. The fused signals for rotation and linear acceleration also have a refresh rate of 100 Hz. The important parts of the specification for both devices are that both fuse their acceleration, gyroscope and magnetometer data on board to provide more accurate gyroscope data and gravity corrected acceleration data. The MetaMotionC

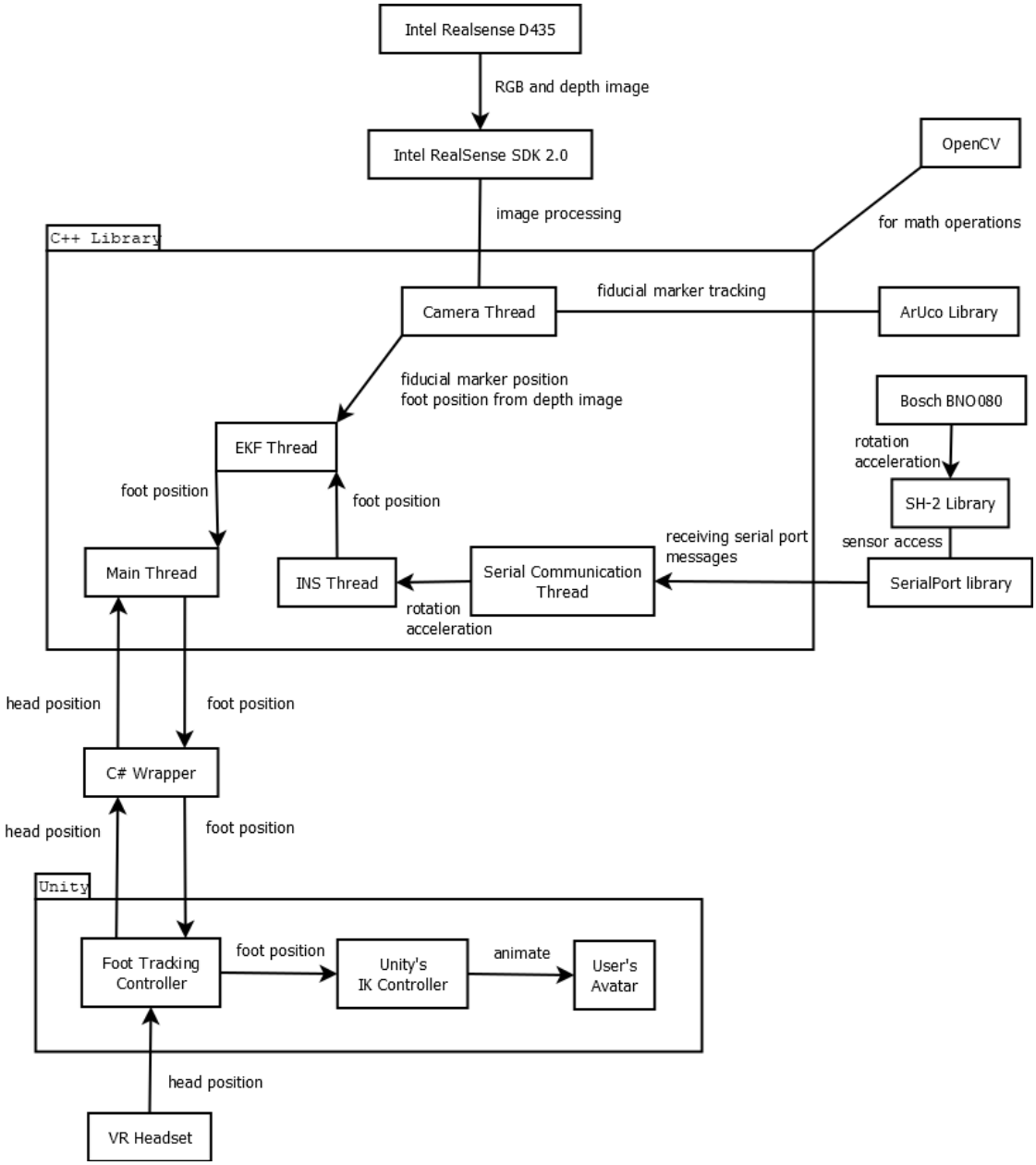

Figure 4.2: An overview of the system architecture

uses a Bluetooth Low Energy connection to communicate with the device. The BNO080 uses a wired serial interface for communication. The serial communication limits the amount of data that can be sent from the device. A data rate of 200 Hz is possible without problems, a higher data rate can cause lost or corrupted data packages. When sending linear acceleration and rotation vector, both can run at 100 Hz. This sets the refresh rate of the BNO080 on a par with the MetaMotionC. In the end I chose the

BNO080 for this project because the Bluetooth library needed for the MetaMotionC was only available for the Universial Windows Platform (UWP). It was not easily possible to use the Bluetooth functionality provided by the library in a C++ library project. The wired serial communication protocol provided by the BNO080 can natively be used in C++ without any extra libraries. The BNO080 also has a way higher refresh rate for its sensors, but is limited by the speed of the serial communication.

The BNO080 runs on a Nucleo development board, shown in Figure 4.3, and is programmed in C. The SH-2 library version 1.0.0 by Hillcrest Labs is used to access and control the sensors. It is programmed to print its sensor data as a string to the serial port. The string consists of a timestamp, an identifier and then the acceleration value or the rotation value in the form of a quaternion. The identifier is either "L" for linear acceleration or "R" for rotation. The messages are separated by a semicolon. The sampling rate is set to 100 Hz, as a faster sampling rate is too much data to transfer over the serial port. It would be possible to get a bit higher sampling rate by sending the data as binary packages, but 100 Hz is sufficient for the purpose of foot tracking. An additional functionality added to the sensor program is the possibility to manually calibrate the acceleration, gyroscope and magnetometer module by doing a calibration routine. This has to be activated on compile time. For the calibration routine one has to place the device on 3 different sides for some seconds and then perform a slow pitch, roll and yawn motion. The calibration is saved with a press of the unit's reset button. When the reset button is pressed 5 times, the saved calibration data is completely wiped and the unit has to be calibrated again. Normally this calibration procedure is not needed, as the BNO080 has a dynamic calibration of its sensors that tweaks its calibration on runtime, when the unit is used.

A slightly modified version of the SerialPort library by Mandal is used to receive the messages of the sensor over the serial port [Man16]. The only edits on the library are compatibility related. The library is designed to receive the messages from the serial port as fast as possible, without big buffering of the data. So every time some parts of the data strings are received they get merged. When a complete message is received this way, it gets processed. The rotation is received as quaternions that consist of r, i, j and k components. They are first normalized by dividing every component of the quaternion by its length. The quaternion is then converted to axis–angle format with the equations 4.1, 4.2, 4.3, and 4.4 described by Baker [Bak15b]. Also the singularities of axis–angle at *angle* 0 and *angle* 180 have to be taken into account. The BNO080 uses a coordinate system where z is up, x is right, y is forward and positive rotations are counter clockwise. This coordinate system is illustrated in Figure 4.1. To accommodate the converted axis–angle to the different coordinate system used by OpenCV, the y and z value have to be swapped and the new y value has to be multiplied by -1 4.5. This rotation value is then saved as it is needed to rotate the acceleration values from the sensor.

$$angle = 2 * \arccos(r) \tag{4.1}$$

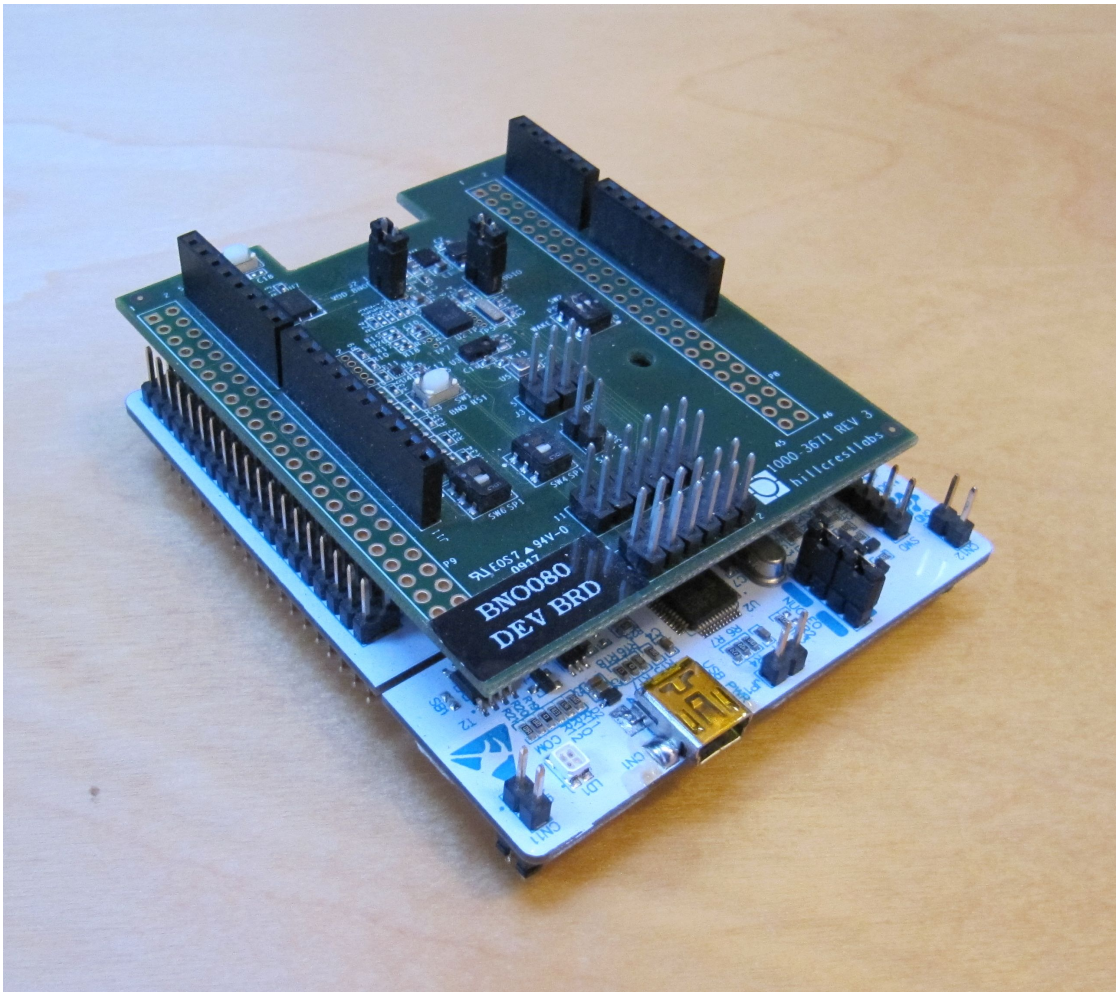

Figure 4.3: Bosch BNO080

$$x = \frac{i}{\sqrt{1 - r^2}} \tag{4.2}$$

$$y = \frac{j}{\sqrt{1 - r^2}} \tag{4.3}$$

$$z = \frac{k}{\sqrt{1 - r^2}} \tag{4.4}$$

$$AxisAngle_{OpenCV} = \begin{pmatrix} x \\ -z \\ y \\ \theta \end{pmatrix}_{BNO080} \tag{4.5}$$

When the acceleration value in the format x, y and z is received from the BNO080, the conversion to the OpenCV coordinate system is done by swapping the y and z

values and multiplying the new y value by -1 4.6. Then it is checked if the foot is stationary by calculating the magnitude of the acceleration value and checking it against a threshold. When the acceleration is below the threshold of 0.5, then the acceleration value is discarded, the velocity value is set to zero and the measured position is set to the old position. This can be done, because of the very unlikely possibility of the feet moving in a constant velocity as described in the methodology chapter 3.2. If the foot is not stationary, then the acceleration value has to be rotated by the orientation of the sensor, to get the acceleration in world coordinates. The acceleration and rotation vectors of the BNO080 have the north pole as their forward vector, meaning that an additional rotation is needed to get the acceleration vector relative to the Virtual Reality (VR) tracking system's forward vector. This offset rotation is done after the rotation of the gyro sensor is applied. The rotations are done by converting the axis–angle rotations into rotation matrices. Then the rotated acceleration vector is given by multiplying the offset rotation matrix with the sensor rotation matrix and with the acceleration vector 4.7. This rotated acceleration vector is then forwarded to the EKF.

$$a_{OpenCV} = \begin{pmatrix} a_x \\ -a_z \\ a_y \end{pmatrix}_{BNO080} \tag{4.6}$$

$$a_{world} = OffsetRotationMatrix * RotationMatrix * a_{local} \tag{4.7}$$

4.3 Fiducial Marker Tracking

The fiducial marker tracking component uses two libraries to fulfil its tasks: The ArUco Library([GJnSMCMJ14], [GJnSMCMC16], [RRMSMC18]) that is included with the OpenCV library is used for the marker tracking itself and the Intel RealSense SDK 2.0 version 2.16.0 is used to receive the RGB image data from the RGB depth camera as well as the depth data from the RGB depth camera. The RGB depth camera that is used is the Intel Realsense Depth Camera D435 (Intel Realsense). Figure 4.4 shows an image of the camera and Figure 4.5 shows the two ArUco fiducial marker that are used. The Intel Realsense has a maximum depth image resolution of 1280 by 720 pixels and a maximum RGB image resolution of 1920 by 1080. The diagonal field of view is a bit over 90 degrees and the vertical field of view of the depth camera is some degree higher than the vertical field of view of the RGB camera. A maximum of 90 frames per second is possible for the depth camera and the working range of the depth camera is 0.2 metres to over 10 metres. The RGB camera has a maximum of 60 frames per second. To use the higher frame rate of the Intel Realsense, a lower resolution has to be used and vice versa, as the amount of data that can be sent is limited trough the USB 3 standard and the quality and length of the used USB cable.

At first the Intel Realsense has to be initialized. The resolution of the camera has to be set to a lower resolution to allow for a faster frame rate. The resolution chosen is 848 by 480 with a frame rate of 60 frames per second. Both, the image stream of the RGB

Figure 4.4: Intel Realsense Depth Camera D435

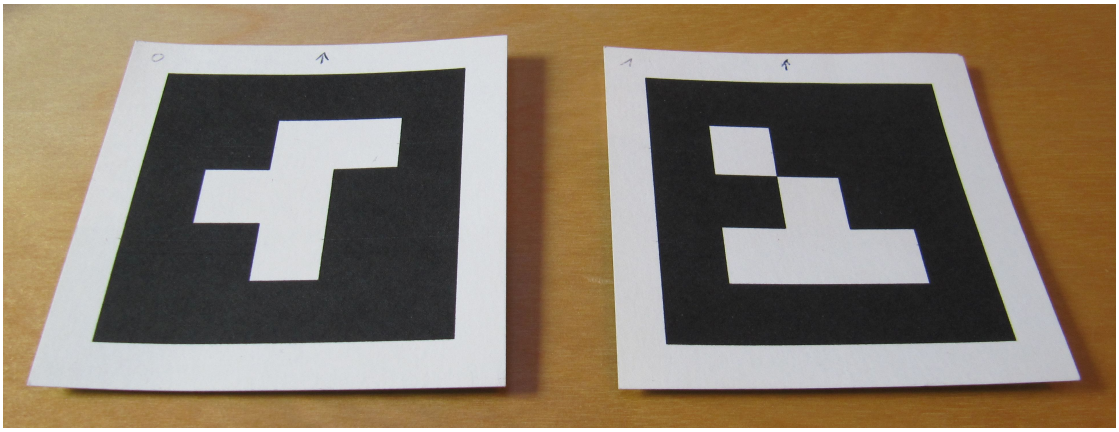

Figure 4.5: ArUco fiducial marker

image and the image stream of the depth image, are initialised with these parameters. To get a better depth image, the following filters are used with the depth image stream: A decimation filter that reduces the depth frame density by downsampling the image, a spatial filter that smooths the image and a temporal filter that smooths the image with information from past frames. Lastly the beforehand generated marker dictionary, the camera matrix and its distortion coefficient are loaded.

In a while loop, the realsense library returns the next RGB and depth image from the camera as soon as it is available. After applying the filter to the depth image, the colour image and the depth image get aligned so that the pixel positions in both images are

the same. This is necessary as the depth camera has a higher field of view than the colour image. An example of a RGB image with a corresponding depth image, with and without apllied filters, is shown in Figure 4.6. The OpenCV library is then used to find the marker in the image. The images have to be converted into matrix format to get useable in OpenCV. With the function *cv::aruco::detectMarkers()* every marker seen in the image is returned by the OpenCV Aruco library. For every marker found in the image, the middle of the marker is calculated, as the library only returns the four edges of the marker. With this position information from the OpenCV library, a lookup in the depth image is possible. The function *rs2_deproject_pixel_to_point()* from the realsense library automatically transforms the 2D image coordinate from the marker to a point in 3D space with the help of the depth information from the depth image. The marker position is now in camera space. This means that it has to be rotated by the rotation of the VR headset to get the marker position in world space. The rotation is saved in axis-angle format. To convert the rotation into matrix form, it first has to be converted from axis-angle to a rotation vector by multiplying the normalized vector of the axis-angle with its angle *theta*. Then the function *Rodrigues()* from the OpenCV library is used to convert the rotation vector to the rotation matrix. The Intel Realsense is mounted in a slight downward angle of 20 degrees on the VR headset. This means that this 20 degrees downwards rotation has to be applied to the marker position before the head rotation is applied. As a last step the head position has to be added to the marker position to get the final position 4.8. The final marker position is then fed into the EKF.

$$p_{markerWorldSpace} = p_{camera} + (RotationMatrix_{camera} * 20^{\circ}_{pitch} * p_{markerCameraSpace})) \tag{4.8}$$

4.4 Depth Camera

The depth image is mainly used when the marker tracking is lost by partial occlusion. The Intel Realsense is used to get a depth image that corresponds to the RGB image used for marker tracking. The calibration is the same as mentioned in the fiducial marker tracking chapter 4.3.

After the marker tracking is done in the camera thread, the depth camera tracking is done for every marker that could not be found in the colour image. The last known marker position and its depth value is the basis for the blob detection. But first the matrix of the depth image is filtered with the *cv::Range()* function. With this function, all values that are too far away from the last marker depth value are set to black, and the rest of the image is set to white. So the resulting image is a binary image with only values of zero and one. Then gaps in the image get closed. To do this, a *cv::getStructuringElement()* is created. The function *cv::morphologyEx()* performs a morphological transformation. With the parameters *cv::MorphTypes::MORPH_OPEN()* an opening operation is performed on the image. There the black parts of the image get enlarged, and after that the white parts of the image get enlarged. This effectively removes noise in the form of small white

holes and gaps. Then the function *cv::findContours()* is used to find the contours of all white blobs in the binary image. For every found blob, a mask is created for this blob by drawing the contour of the blob into a new matrix. Then a rectangle with a minimal area is fitted around the blob, to best approximate the foot shape. Now the blob that is nearest to the last marker position from either the marker tracking algorithm or the depth tracking algorithm is chosen. Figure 4.7 shows the generated blobs with the final marker position marked with a dot. If the new approximated marker position is too far away from the last approximated position, the new position is ignored and the tracking is marked as lost. This means that the inertial sensor has to track the foot until the fiducial marker tracking finds the marker again.

The middle of the chosen blob is saved as the marker position and is used to get the new marker depth. But the middle of the blob is not necessarily a white pixel that has a valid depth value. That is why a function is written, that finds the nearest white pixel in the given image. This function works as follow: To be efficient, the distance calculations are done with matrices. First all positions of white pixels are written into a matrix with the function *cv::findNonZero()*. This matrix is then split into two where one contains all x values and the other contains all y values. Now all vectors from the starting pixel to every white pixel are calculated by subtracting the x and y value of the start pixel from all respective values in the x and y matrix. With the *cv::magnitude()* function it is now possible to calculate the length from all vectors. The pixel of the vector with the shortest length is then chosen for the depth value of the currently approximated marker position. The final position is then fed into the EKF.

4.5 Sensor Fusion

As described in the state of the art chapter 2.5, the sensor fusion component uses an EKF for fusing the different signals from the different sources. The used implementation of the EKF is the KFilter library version 1.3 by Zalzal [Zal08]. The KFilter library itself uses optimized algorithms by Bierman that are translated from Fortran [J77]. The KFilter library is used by creating a new class for the EKF and inherit the EKF class from the library. All the matrices of the EKF have to be created by overwriting the functions *makeBaseX()* and *makeX()*, where X is the name of the matrix. The *makeBaseX()* function only contains constant values that never change. Changing values like Δt are put into the *makeX()* function as it will be newly created before the library predicts and updates the EKF state. The matrices that have to be filled in are the Jacobian matrices A, W, H and V that are described in the methodology chapter 3.5, and the noise matrices R and Q. The values of the process noise Q and the two measurement noises R_p and R_a are adapted values based on the work of He et al. and the actual data sheet of the IMU [HKS+15]. The process noise Q consists of the variance values of the state values of the EKF. Q is a diagonal matrix, as the variances of the states are all independent. The variance of position p and velocity v are based on the paper by He. They are $\sigma_p = 0.0002m$ and $\sigma_v = 0.00006m/s$. The variance of acceleration a is based on the datasheet of the BNO080 and is $\sigma_a = 0.02m/s^2$. The measurement noises R_p and R_a

are also diagonal matrices, as the variables are independent. The variance of the position during measurement is also based on the paper by He and is $\sigma_{mp} = 0.00015m$. The variance of the acceleration during measurement is from the datasheet of the BNO080 and is $\sigma_{ma} = 0.35m/s^2$. Also a different variance has to be used for measuring positions through the depth sensor component. A variance of $0.35m$ is chosen, as the estimated position of the depth sensor can be a bit jittery. This higher variance means that the measured signal has a higher uncertainty.

$$Q = \begin{bmatrix} \sigma_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_v & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_v & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_v & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_a & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_a & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sigma_a \end{bmatrix} \tag{4.9}$$

$$R_p = \begin{bmatrix} \sigma_{mp} & 0 & 0 \\ 0 & \sigma_{mp} & 0 \\ 0 & 0 & \sigma_{mp} \end{bmatrix} \tag{4.10}$$

$$R_a = \begin{bmatrix} \sigma_{ma} & 0 & 0 \\ 0 & \sigma_{ma} & 0 \\ 0 & 0 & \sigma_{ma} \end{bmatrix} \tag{4.11}$$

The last two functions from the KFilter library that have to be overwritten are *makeProcess()* and *makeMeasure()*. The function *makeProcess()* is an implementation of the state transition function of the EKF. The state transition function is already described in the last chapter with the functions 3.4, 3.5, 3.6 and 3.7. The *makeMeasure()* function tells the EKF how the measured value can be mapped to the state vector. As the values that are measured are either position or acceleration values, they can directly be mapped to the state vector. The type of the measurement has to be set before the update and measure methods of the EKF are called. This is done with the functions *setPositionMeasurement()*, *setPositionDepthMeasurement()* and *setAccelerationMeasurement()*.

To accommodate the fact that the feet can be stationary while measuring the acceleration the *makeProcess()* function of the EKF library is changed. When the stationary variable is set by the inertial navigation component, the used transition model is changed to set acceleration and velocity to zero and the position to the value of the old position. This

ensures that the next measurement from the accelerometer gets discarded and the foot position will not change while not moving.

The now implemented class of the KFilter library is used in the code as follows: The first thing that has to be done is the initialisation process for the EKF. The *init()* function sets the initial state vector $X_{initial}$ and the initial covariance matrix $P_{initial}$. The initial state vector is set to all zero, as the position, velocity and acceleration set to zero are a good starting point 4.12. The initial covariance matrix chosen defines the uncertainty with the initial state vector. As the movement of the user at the beginning can not be known, the variance values have to be chosen in a way, that the initial position and movement of the user is within this variance margin. The chosen variance values for position, velocity and acceleration are based on the average walking motion of a human being [ABI18]. The values itself do not have to be very accurate, as the EKF converges relatively fast to a correct solution. So it is better to overestimate the values than to underestimate them, as underestimation sets too much confidence in the initial state value.

$$X_{initial} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.12}$$

$$P_{initial} = \begin{bmatrix} 1^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & (\frac{3}{4})^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (\frac{3}{4})^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & (\frac{3}{4})^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15^2 \end{bmatrix} \tag{4.13}$$

After the *init()* function of the EKF is called with the initial state vector and the initial covariance matrix as parameters, the EKF is ready to use. Now every time a new value is reported from the fiducial marker, depth or inertial tracking component, these values get fed into the EKF. This is done by setting the delta time since the last measurement, setting the type of the measurement, either position through marker tracking, position

through depth tracking or acceleration, and then calling the *step()* function of the EKF. The measurements get in at a high rate of 100 Hz, but for a better representation of the current foot position, the current state of the EKF is predicted every time the IK component wants to read the foot position. This is done with the same procedure as when measuring a new value, but it is done with an empty measurement vector. This skips the update step of the EKF that would normally correct the predicted vector by the measured vector.

To get the current position from the EKF the current state vector can be retrieved from the EKF. After the prediction of the position, the current state vector is returned by the function *getX()*. The first three numbers of the vector contain the x, y and z coordinate from the foot position. To further smoothen the output from the EKF, an additional exponential moving average filter is used 4.14. This filter interpolates between the new and the old value. An alpha value between zero and one defines how far the filtered value is to the new value. With alpha equals zero only the old value is used and with alpha equals one only the new value is used. The alpha value is chosen to be 0.2 so that the new values wont affect the overall position too fast. The filtered position is then forwarded to the inverse kinematic component for the animation of the avatar.

$$\bar{x}_k = \bar{x}_{k-1} * (1 - \alpha) + x_k * \alpha \tag{4.14}$$

4.6 Inverse Kinematic

As mentioned in the system architecture chapter 4.1, the lack of a good IK library written in C++, led to the usage of the Unity engine for the IK part of the system. The usage of the built in IK library of the Unity engine is straight forward. The prerequisite for this is that a humanoid avatar is used. Then an animator component has to be attached to the avatar. A simple animator controller has to be attached to the animator component. A controller script that handles the IK library calls can then be attached to the avatar. The inputs for the IK controller script are both foot positions as well as the user's head position from the VR headset. The foot positions get pulled from the C++ library with a managed C# wrapper library. This is necessary to use a C++ library directly in Unity. After initialisation of the C++ library by setting the serial ports of the IMUs and starting the individual threads, the foot positions get updated in the Unity *update()* method, once every frame. This guarantees that the avatar gets updated as often as possible.

The IK controller attached to the avatar implements the *OnAnimatorIK()* function that is called every time before Unity updates the IK system. In this function the foot position, the head position as well as their rotations get applied to the avatar model. In the *Update()* function of the IK controller, the camera position of the scene is set to the head position of the avatar, so that the user has the right view from the VR headset. Figure 4.8 shows the user's avatar animated by Unity's IK system.

As Unity uses another coordinate system as used in the C++ library and used by the OpenCV library, a conversion has to be done before using the data from the C++ library. The coordinate system is illustrated in Figure 4.1. With the Unity coordinate system the y axis is up, the x axis is right, the z axis is forward and positive rotations are clockwise. This means that for the position conversion only the y axis has to be multiplied by -1. For the axis-angle representation of the rotations, the conversion is done by multiplying the x and z coordinate by -1. Then the *Quaternion.AngleAxis()* function of Unity can be used to convert the axis-angle to a quaternion that is used by Unity.

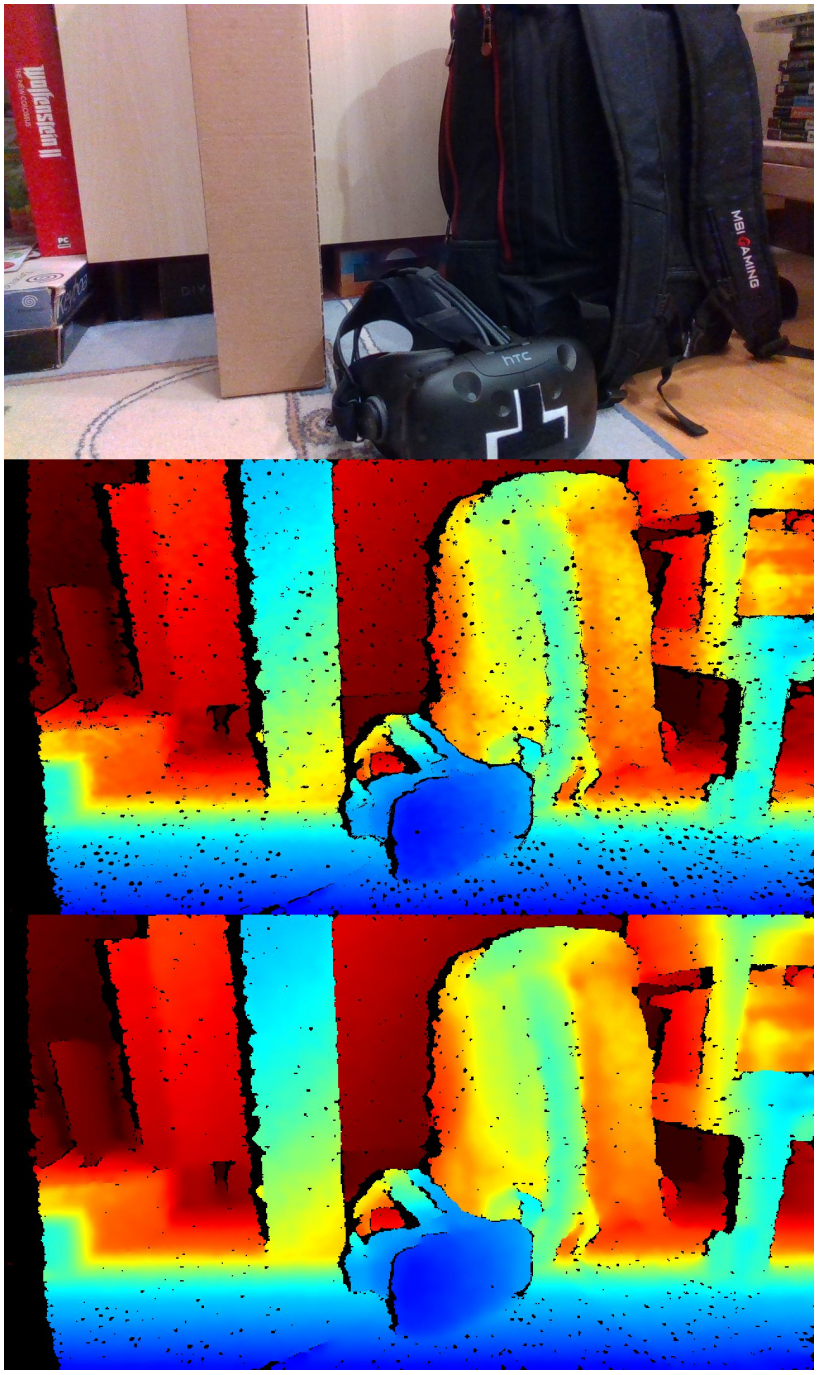

Figure 4.6: The colour image with its corresponding depth image. The top depth image is without applied filters and the bottom depth image is with applied filters. The depth images are coloured for better visualisation with blue for near values and red for far values.

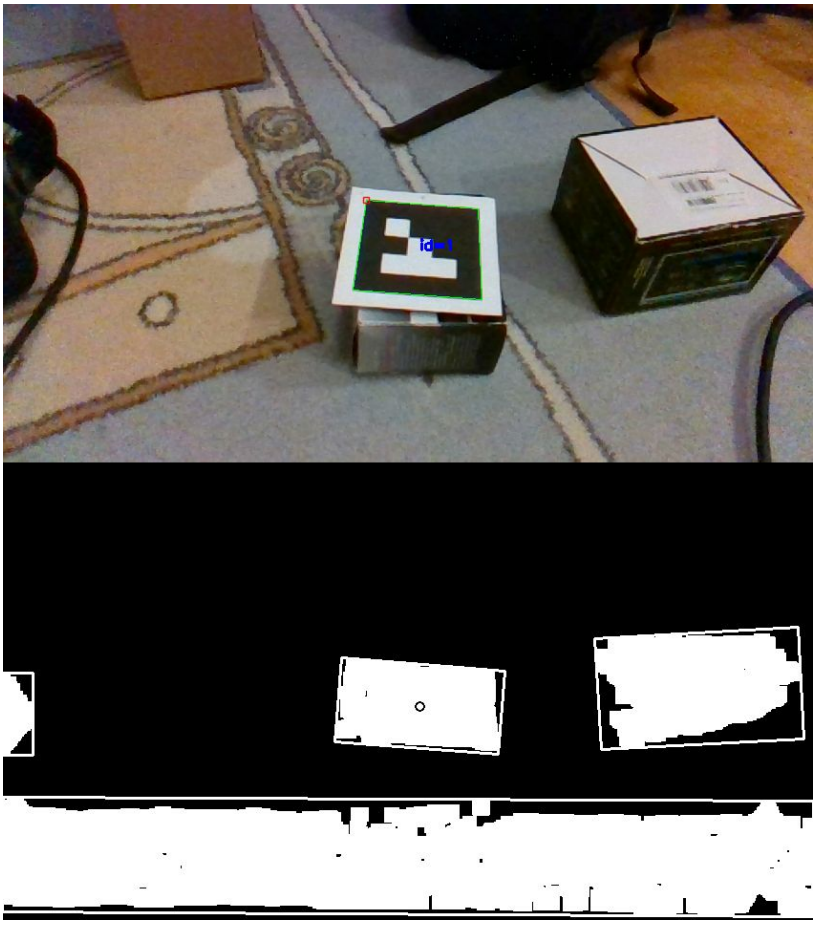

Figure 4.7: Visualisation of the depth tracking algorithm. The bottom image shows the chosen blob.

Figure 4.8: IK animation of the avatar. The green and blue cubes represent the foot positions.

CHAPTER 5

Evaluation

To evaluate the proposed foot tracking system, the following steps have been done: Another reasonably accurate tracking system has been found. This system was then compared to the proposed foot tracking system. Additionally a user study was conducted to test the accuracy of the system in more real life conditions.

Another accurate tracking system is the HTC Vive by HTC and Valve. A study by Niehorster et al. provides an analysis of the accuracy of this system [NLL17]. They conclude that the HTC Vive has some pros and cons that allows for usage in scientific studies under some circumstances. The pros are that the system is affordable compared to equivalent systems, the system latency is low at 22 milliseconds and the noise level is low. The downsides of the system are the position and orientation measurements that are relative to a floor plane that is slightly tilted to the actual real world floor. This results in slightly tilted roll and pitch measurements as well in changing height measurements on the tracking area. Depending on the needed accuracy, this can be countered by calibrating the floor plane with a measurement of the tilted floor plane. The bigger problem is that the orientation of the tilted floor plane changes when the tracking of the HTC Vive is lost, even when it is only lost for a very short moment. This makes the system only viable when used at a smaller tracking area where occlusion is less likely. As the accuracy test was done in a relatively small area, there was no hindrance in using the HTC Vive for the accuracy test.

5.1 Technical Evaluation

The proposed tracking system has three stages of combination of the tracking methods used. All three tracking methods are used when fiducial marker tracking is available. Depth tracking and inertial navigation system (INS) tracking is used when the fiducial marker is partially occluded. Only the INS tracking is used when both, fiducial marker

and depth tracking, is lost. Each of these tracking stages were compared to the ground truth in form of the HTC Vive.

The hardware was set up in the middle of the room where the HTC Vive was not occluded and had a good tracking signal. The Virtual Reality (VR) headset was mounted on a fixed place about 1.5 metre above the ground and was pointed downwards. The tracking hardware of the proposed system was fixated to the HTC Vive controller so that the difference in position could be measured. The setup is illustrated in the figures 5.1, 5.2, and 5.3.

Figure 5.1: **Technical Evaluation:** The VR headset was mounted about 1.5 metres above ground

Figure 5.2: **Technical Evaluation:** The VR headset with the Intel Realsense Depth Camera D435 (Intel Realsense) pointed downwards to the ground

The four tests that were done with every tracking stage were:

- The difference in position when the tracker and the VR headset are stationary for about one minute.
- The difference in position when the tracker is stationary and the VR headset is moved.
- The difference in position when the tracker is moved and the VR headset is stationary.
- The latency of the system when the tracker is suddenly moved in a fast manner.

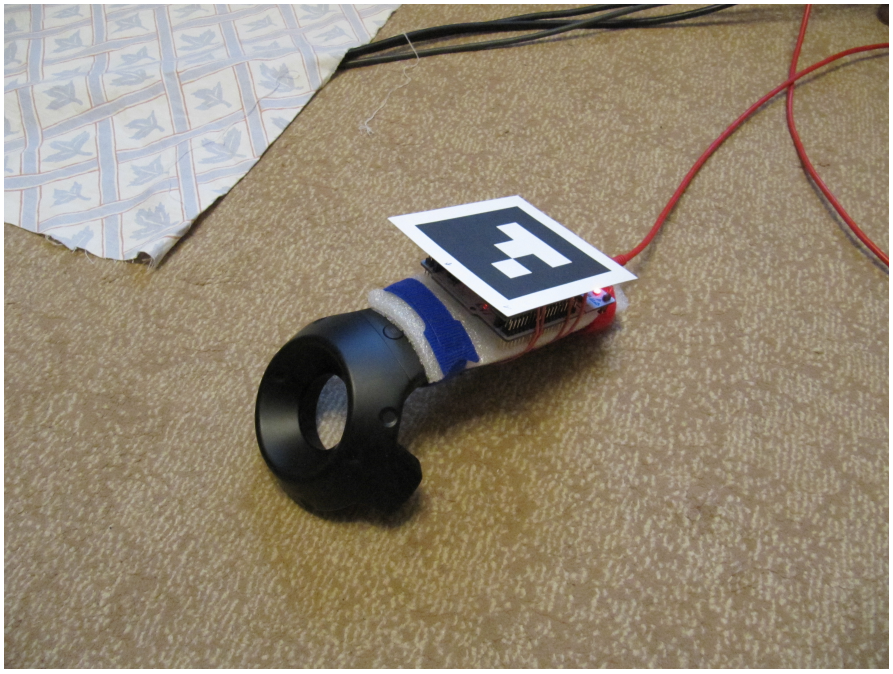

Figure 5.3: **Technical Evaluation:** The HTC Vive controller was fixed to the Bosch BNO080 and the fiducial marker

The tracker movement as well as the headset movement done in the tests were a movement left, right, forwards, backwards, up and down as well as a movement in a circle.

The test results comparing the HTC Vive to the proposed system are as follows:

- **Marker and VR headset were stationary:** Figures 5.6, 5.10 and 5.14 show the distance between the systems over time for the marker, depth and INS tracking stages. The average distances for the stages were 2.9562 millimetres for marker, 2.9738 millimetres for depth and 0.64646 millimetres for INS tracking. The biggest distance measured during the tests were 6.8182 millimetres for the marker stage, 12.568 millimetres for the depth stage and 1.0723 millimetres for the INS stage. For the average case the small jitter only moved the position about 3 millimetres and for the INS tracking only 0.6 millimetres. But with the depth tracking stage the jitter could go up to 1.3 centimetres. This test showed that the position of the proposed system wont drift over time.
- **Marker was moving and VR headset was stationary:** The distances while moving the fiducial marker are shown in the figures 5.7, 5.11 and 5.15. The average variation in distance was 33.645 millimetres for marker, 40.959 millimetres for depth and 169.07 millimetres for INS tracking. The maximum distance was 123.74 millimetres for marker, 103.15 millimetres for depth and 480.95 millimetres for INS tracking. In the worst cases of the test, the marker and depth tracking stage

had a bigger gap from about 10 centimetres but only for a short amount of time from about two to three seconds. The INS tracking had a big gap from nearly 0.5 metres and that for about 5 seconds but it went back to a smaller level of about 5 centimetres after these 5 seconds. The other two stages went back to a distance of about 2 centimetres at the end of the test, when the movement stopped. This means that the system lost accuracy while moving but still generated accurate results after the movement ended.

- **VR headset was moving and marker was stationary:** The distances for this test are shown in the figures 5.8, 5.12 and 5.16. The average distances of this test were 35.35 millimetres for marker, 12.133 millimetres for depth and 0.17528 millimetres for INS tracking. The biggest measured distances for the tracking stages were 129.88 millimetres for marker, 21.989 millimetres for depth and 0.85232 millimetres for INS tracking. For INS tracking, a distance change was nearly non existent with a maximum of under one millimetre. The marker tracking stage was comparable to the test where the marker was moved and the VR headset was stationary. The depth tracking stage had results that were as good as the results for the test where marker and VR headset were stationary.
- **Latency:** Figures 5.9, 5.13 and 5.17 show the distance change of this test. The latency of the tracking stages could be measured by measuring the time from the start of the distance change to the maximum distance change. That is because the system then starts to react to the marker position change and thus the distance to the correct HTC Vive controller gets smaller again. The latencies for the stages were 0.3137 seconds for marker, 0.33622 seconds for depth and 0.36908 seconds for INS tracking. All three stages had about the same latency, but the INS tracking state was a bit slower than the rest.

The tests showed that the proposed system had only a small inaccuracy as long as the marker was not occluded or out of the field of view for too long. When the INS tracking stage had to take over, then the accuracy depleted with longer distances but stayed constant when the foot was not moved. Also the accuracy was worse while moving but got better again when the movement had ended. Not shown in the data, but observed while recording the tests, was the fact that the tracking had a small but constant shift near the edge of the camera's field of view. Most of the bigger spikes in the distance from the two systems came from the latency of the proposed system.

5.2 User Study

The user study was done to get some data that is more to real life conditions than the technical evaluation done at laboratory conditions. For the user study, the participants got an additional HTC Vive controller mounted on their feet in addition to the fiducial marker and the inertial measurement unit (IMU). With this setup the difference in position between the proposed system and the HTC Vive tracking could be measured.

The two systems were placed near enough on the foot, so that the distance to each other should not change. This means that the distance between the two systems should give information on how good the proposed system performed.

The setup of the study was as follows: A VR scene was created in Unity that allowed for the users to move freely in a small space. The scene contained a room that had a wooden floor and was filled with some plants and some furniture that was placed around the moveable area. This was done to increase the user's immersion and to better perceive distance in the room. The users were represented by an avatar in the form of a humanoid robot. The feet were tracked with the proposed system and then visualised by Unity's Inverse Kinematics (IK) system. To track the user's head for the proposed system, the HTC Vive VR headset was used.

The hardware setup of the user study was as follows: Two fiducial markers with a 3 by 3 pattern were placed on the users feet near the toes. The Bosch BNO080 IMU was placed on the ankle of the user's right foot. The left foot had no IMU attached, to test the difference in accuracy of the system with and without the inertial navigation component. Two additional HTC Vive controllers were mounted on the lower part of the shin, in a way where it could not occlude the fiducial markers. The setup of the feet is illustrated in figure 5.4.

The usable walking space for the user study was 2.2 metres by 2.3 metres. Two HTC Vive Lighthouse tracking stations that are required for the tracking of the HTC Vive controller and VR headset, were placed in the diagonals of the room, about one metre outside of the useable VR space, to guarantee optimal tracking conditions. The HTC Vive headset has a maximum field of view of about 110 degrees, a refresh rate of 90 Hertz and a resolution of 1080 by 1200 pixels per eye. The Intel Realsense was mounted on the front of the VR headset, where it not occluded the headset's sensors. The camera was mounted in a slight downward angle of 20 degrees to better align the camera's and headset's field of view. This is shown in figure 5.5.

The sample size of the study consisted of 6 people ranging from ages 20 to 70. Two of the participants were experienced VR users that used VR environments regularly, two had used VR environments before, but not very often and two were completely new to VR systems. The small sample size was due to safety constraints regarding the COVID-19 pandemic.

For the experiment, every participant got instructions on how to put the tracking systems on. Then they were advised to look at their feet during the test, as the proposed system only tracks the feet when they are seen by the user. During the experiment, every user had to do tasks that got recorded. The recorded data was the foot position as calculated by the proposed system and by the HTC Vive system. The tasks every participant had to perform were the following:

- They should go one step forward and then one step backwards.
- They should go one step to the left and then one step to the right.

Figure 5.4: **User Study Setup:** The HTC Vive controller were mounted on the side of the ankles. The fiducial markers were placed near the toes. The IMU was mounted slightly above the HTC Vive controller on the right foot.

- They should go some steps on a straight line with a length of about two metres.
- They should walk in a circle with a diameter of about two metres.
- They should bend their knees a bit to lower their body.
- They should only rotate their upper body left and right while their feet are standing still.

The results of the movement experiment are shown in table 5.1. For every foot, the distances from the proposed system to the HTC Vive were calculated. A smaller distance means a better performance of the proposed system. The distance is not the same as the error of the proposed system, as the HTC Vive itself has a small error margin.

The left foot, which had no INS sensor attached, had a total average distance of 134.82 millimetres and a maximum average distance of 1.04434 metres. The median of the distances from the left foot to the HTC Vive controller was 36.84 millimetres. The right foot with the IMU attached had a total average distance to the HTC Vive controller of 93.09 millimetres. The maximum value average was 55.414 centimetres and the median of all the tests was 27.41 millimetres. This worst case value was from one participant, where the foot position was falsely detected as 6.5 metres away from the real position.

Figure 5.5: **User Study Setup:** The VR headset had the Intel Realsense mounted on it, placed on the front where it could not occlude the headset's sensors.

The median of the maximum distances of the tests was 47.832 centimetres for the left foot and 29.035 centimetres for the right foot.

The comparison between the average and the median value of the feet show that the peak values only appeared for a very short time period. There are some reasons why these distances spikes happened. It happened when the fiducial marker tracking detected something else as the marker. This could happen if the lighting of the room is not ideal. Then dark objects in the background could falsely be detected as the marker. Another reason for the wrong foot position was that the system calculated wrong values in the moment when the fiducial marker left the field of view of the camera. These wrong values had a great influence on the left foot as it did not have the IMU attached to it. The values from the right foot show, that the maximum values were smaller than the ones from the left foot, as the INS tracking helped avoiding such high distance spikes. The spikes were reduced as the IMU wont detect this sudden movement to the wrong position and thus helps to correct the wrong value.

The median values show that the system had most of the time a reasonable distance to the HTC Vive. The total median value of the right foot was 27 millimetres. The Sense of Agency, as described in the state of the art chapter 2.1, is influenced by the accuracy of the tracking system. In this context, it was a reasonable small difference to the real foot position, as the foot position does not have to be as precise as for example a hand position.

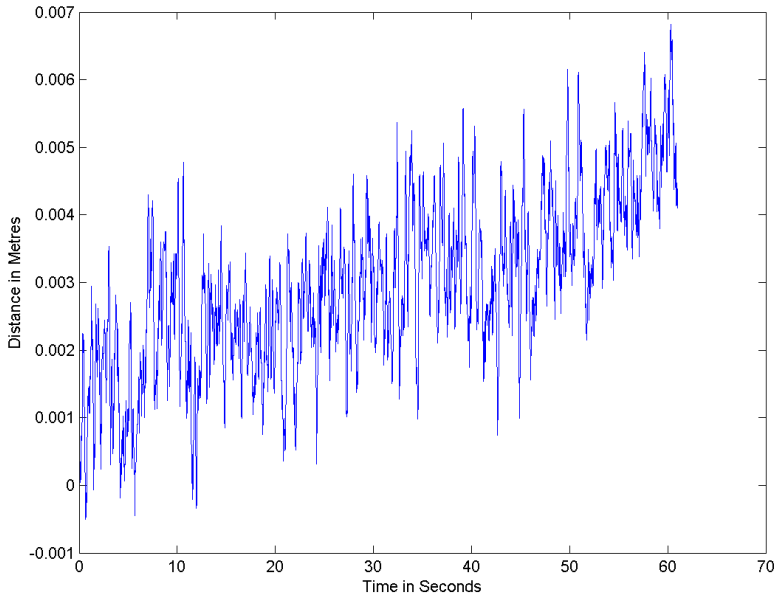

Figure 5.6: **Technical Evaluation:** Marker Tracking Stage: The tracker and the VR headset are stationary

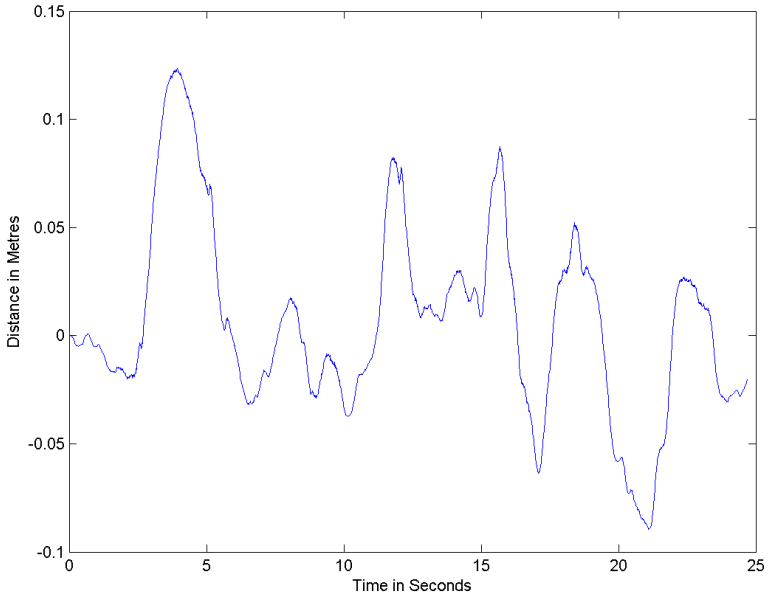

Figure 5.7: **Technical Evaluation:** Marker Tracking Stage: The tracker is moving and the VR headset is stationary

Table 5.1: Results of the user study: shown are distances between the proposed foot tracking system and the HTC Vive

	Left foot average	**Left foot median**	**Left foot max value average**	**Right foot average**	**Right foot median**	**Right foot max value average**
Forward and backward steps	0.19164 m	0.03392 m	1.82836 m	0.05144 m	0.02637 m	0.28215 m
Sideways steps	0.13455 m	0.04378 m	0.53833 m	0.06457 m	0.03246 m	0.32961 m
Walking in a straight line	0.08470 m	0.02526 m	0.62755 m	0.06185 m	0.02516 m	0.39436 m
Walking in a circle	0.21039 m	0.08842 m	1.95595 m	0.22300 m	0.08039 m	0.98367 m
Bend the knees	0.04622 m	0.03232 m	0.16635 m	0.04317 m	0.02379 m	0.16468 m
Rotate upper body	0.14143 m	0.04907 m	1.14953 m	0.11419 m	0.02401 m	1.17036 m
Total	0.13482 m	0.03684 m	1.04434 m	0.09304 m	0.02741 m	0.55414 m

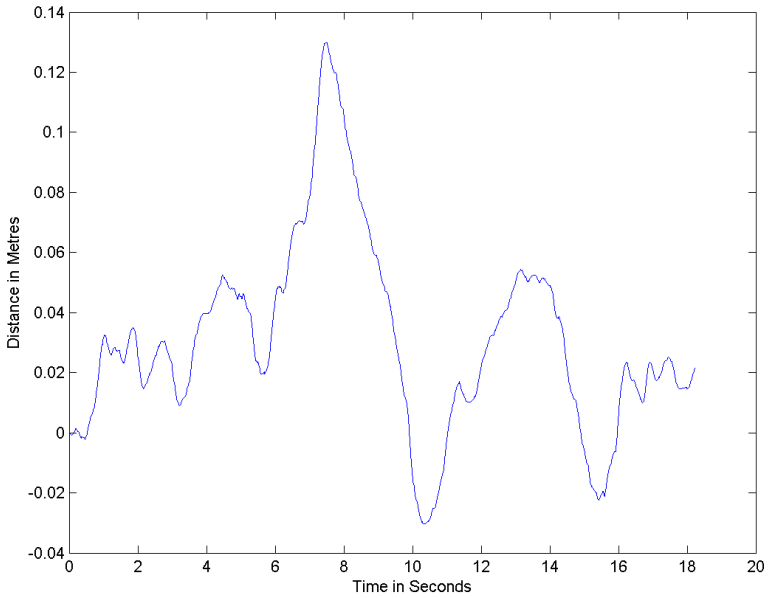

Figure 5.8: **Technical Evaluation:** Marker Tracking Stage: The tracker is stationary and the VR headset is moving

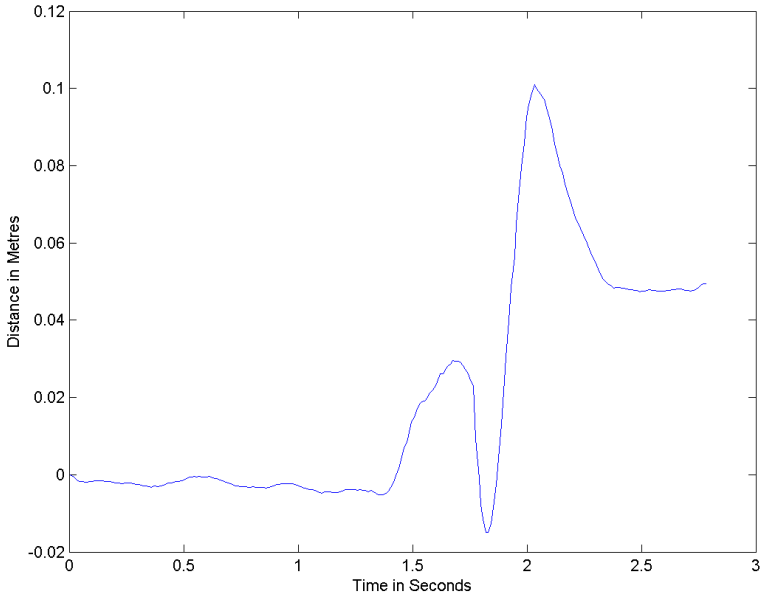

Figure 5.9: **Technical Evaluation:** Marker Tracking Stage: Latency test

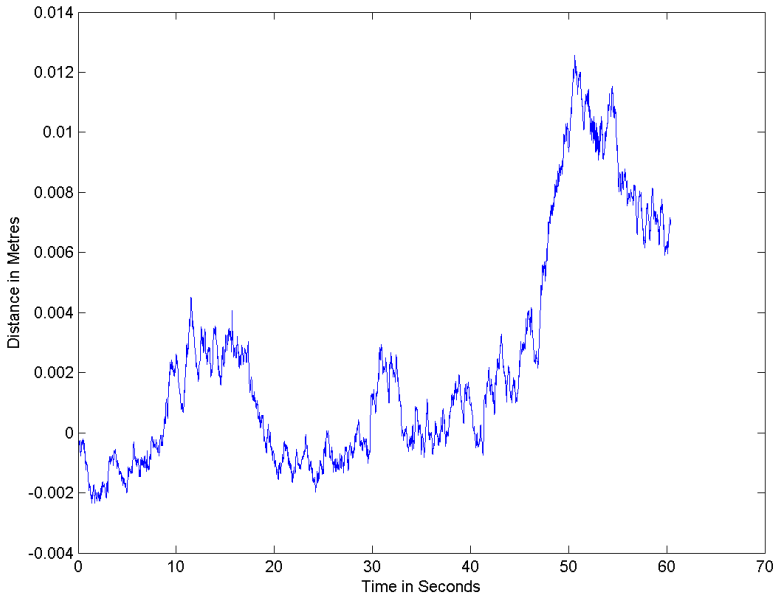

Figure 5.10: **Technical Evaluation:** Depth Tracking Stage: The tracker and the VR headset are stationary

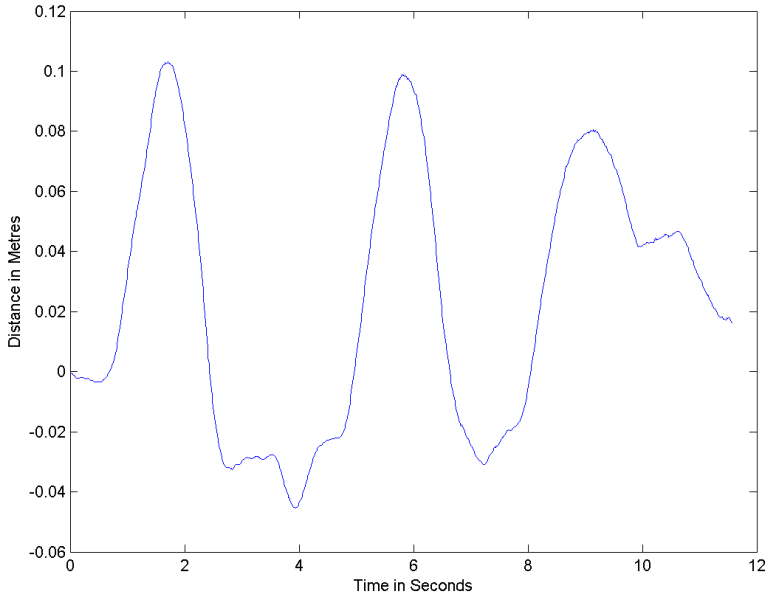

Figure 5.11: **Technical Evaluation:** Depth Tracking Stage: The tracker is moving and the VR headset is stationary

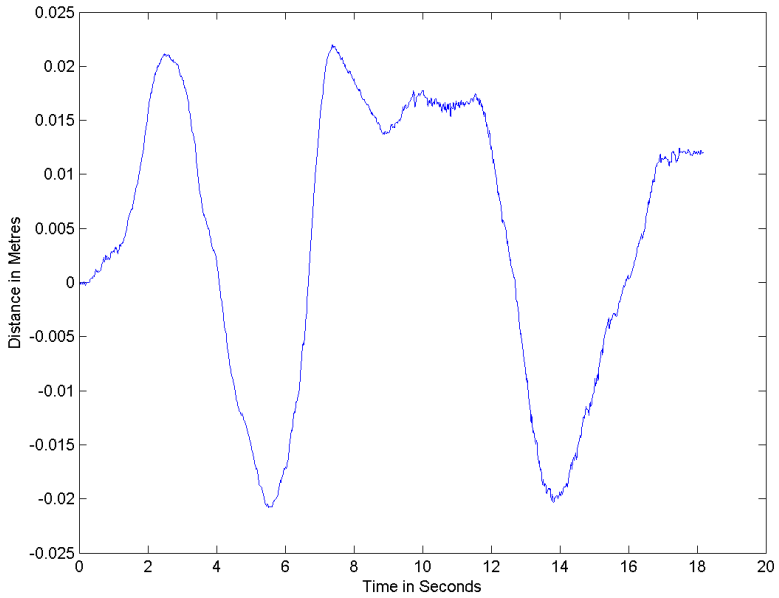

Figure 5.12: **Technical Evaluation:** Depth Tracking Stage: The tracker is stationary and the VR headset is moving

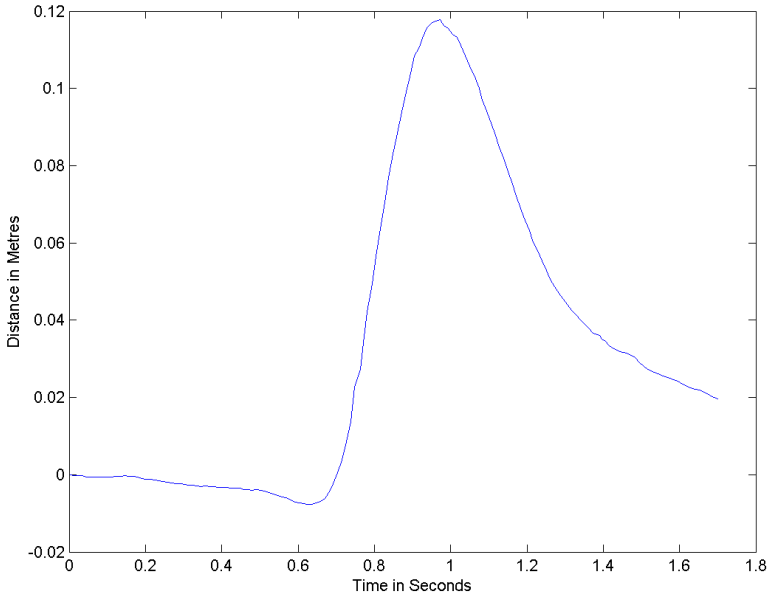

Figure 5.13: **Technical Evaluation:** Depth Tracking Stage: Latency test

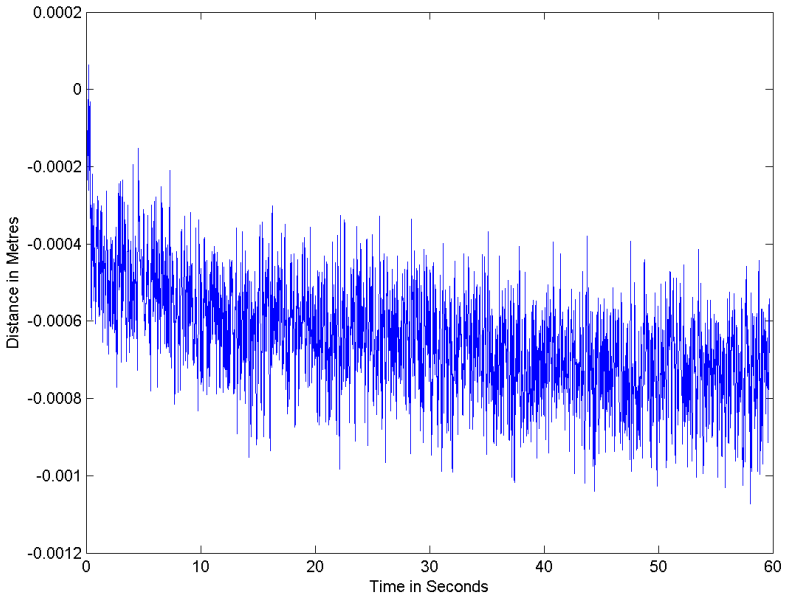

Figure 5.14: **Technical Evaluation:** INS Tracking Stage: The tracker and the VR headset are stationary

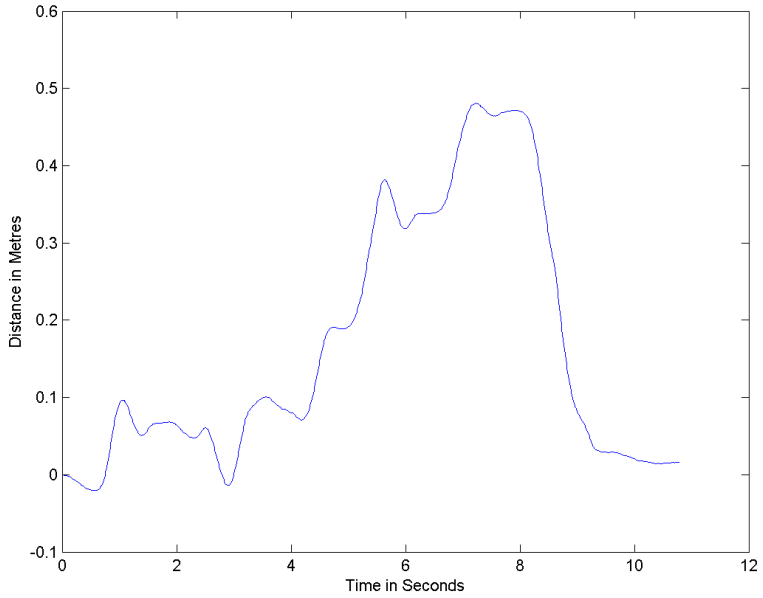

Figure 5.15: **Technical Evaluation:** INS Tracking Stage: The tracker is moving and the VR headset is stationary

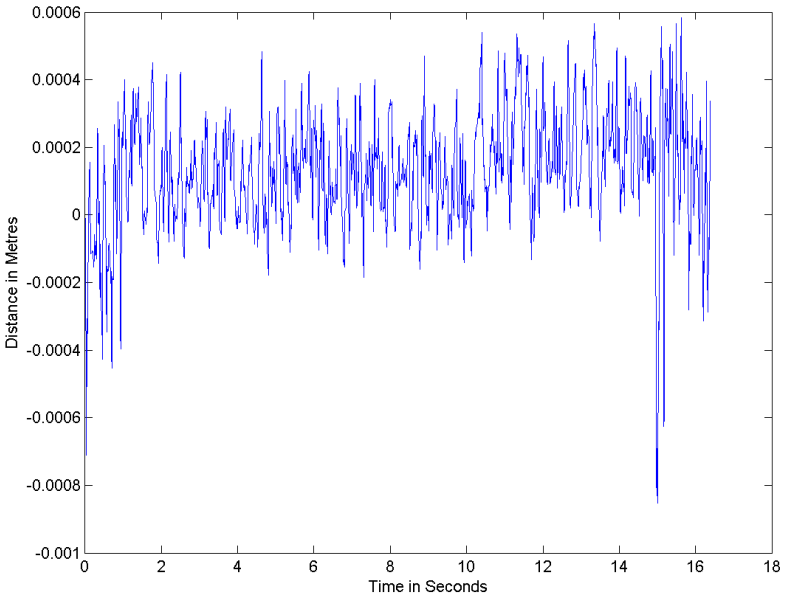

Figure 5.16: **Technical Evaluation:** INS Tracking Stage: The tracker is stationary and the VR headset is moving

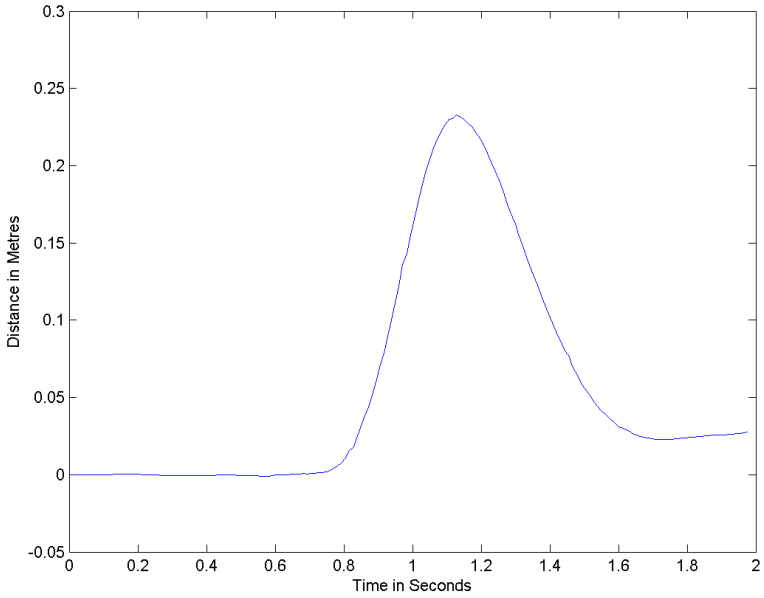

Figure 5.17: **Technical Evaluation:** INS Tracking Stage: Latency test

CHAPTER 6

Summary and Future Work

In this chapter the proposed system and its evaluation are summarised. Then an outlook on future work and possible improvements to the system are given.

6.1 Summary

I proposed a lightweight foot tracking system that works independent from the used Virtual Reality (VR) system. It does not matter if an inside-out or outside-in VR system is used, that means that the proposed tracking system is also suitable for bigger VR environments where occlusion would be a problem when outside-in tracking is used. The proposed system has a RGB-depth camera mounted on the VR headset and uses fiducial marker tracking as its main tracking system. The fiducial marker are mounted on the user's feet. The depth camera is used to track the feet while the marker is occluded. Additionally inertial measurement units (IMUs) are mounted on the user's feet to maintain a tracking signal while the feet are not in the field of view of the camera and to get a higher tracking frequency. These three mostly independent tracking systems are fused with an Extended Kalman Filter (EKF). The calculated position is then used to animate the user's avatar with the usage of Forward And Backward Reaching Inverse Kinematics (FABRIK), an Inverse Kinematics (IK) algorithm.

The evaluation of the proposed system was done by comparing the calculated foot positions with the HTC Vive tracking system. It showed that the system works very well when the feet are standing still, even when the markers are occluded or not in the field of view of the camera. Due to a system latency of about 0.3 seconds, a deviation of about two to ten centimetres to the real foot position is normal while walking. Even bigger gaps are possible due to wrong fiducial marker positions when the marker leaves the field of view of the camera or when other objects are detected as the marker due to bad lighting. But as the system has no time related drift, the foot positions go back to their real positions in a fast manner when movement stops. The user study showed the

importance of the inertial navigation system (INS) component of the proposed system, as it helps to correct the bigger position errors from the fiducial marker tracking component. The IMU wont detect the big position change from a faulty marker detection and thus hinders the EKF to drift as much as without the INS component.

6.2 Future Work

Future improvements could be done by improving the system latency. There are some ways to achieve this: By using a RGB depth camera with a higher frame rate at the same resolution of 1280 by 720 pixel or higher. A higher resolution would reduce jitter from the fiducial marker tracking and a higher frame rate would mean, that less foot positions between the frames have to be calculated by the INS. This would result in even smoother movement and less latency. What has to be improved to eliminate extreme outliers, is a detection when the marker leaves the field of view of the camera. It would also be possible to use a camera with a higher vertical field of view. Then the marker would less often leave the field of view and the range of the trackable area would increase. This would also improve the system, as the tracking would then also be possible, when the user is not directly looking at their feet.

The proposed system could be extended by implementing a way to track the user's hands as well. But this would be more difficult than the feet, as the hands can be rotated by 180 degree. This would make a reliable single marker placement for the hand hard. It would also be less useable than foot tracking, as it would be nearly impossible to track the fingers with markers. But maybe the hands could be tracked with markers and the fingers could then be tracked with the depth data. A system built this way must then have a bigger focus on the depth tracking, as the fingers would have no fallback tracking system when occluded.

List of Figures

List of Tables

Acronyms

AR Augmented Reality. 10

EKF Extended Kalman Filter. 12, 20, 21, 23, 25, 30, 32–36, 57, 58

FABRIK Forward And Backward Reaching Inverse Kinematics. 13, 14, 23, 57, 59

IK Inverse Kinematics. ix, 3, 5, 12, 13, 15, 16, 23, 25, 36, 40, 46, 57, 59

IMU inertial measurement unit. ix, 7, 8, 11, 12, 17, 18, 20, 25, 26, 33, 36, 45–48, 57–59

INS inertial navigation system. 3, 7, 15, 20, 41, 44, 45, 47, 48, 54, 55, 58, 60

Intel Realsense Intel Realsense Depth Camera D435. 2, 30, 32, 43, 46, 48, 59

KF Kalman Filter. 11, 12, 18, 20

MEMS micro-electromechanical systems. 7, 15

RANGO RANdomized Global Object localozation. 11

SoE Sense of Embodiment. 5, 6

UWP Universial Windows Platform. 28

VR Virtual Reality. vii, ix, 1–3, 5, 6, 15, 16, 30, 32, 36, 42–46, 48, 49, 51–55, 57, 59, 60

WIP Walking-in-place. 6

Bibliography

[AAAA17] A. Abadleh, E. Al-Hawari, E. Alkafaween, and H. Al-Sawalqah. Step detection algorithm for accurate distance estimation using dynamic step length. In *2017 18th IEEE International Conference on Mobile Data Management (MDM)*, pages 324–327, 2017. doi:10.1109/MDM.2017.52.

[AAFP16] Sharath Akkaladevi, Martin Ankerl, Gerald Fritz, and Andreas Pichler. Real-time tracking of rigid objects using depth data. In *OAGM & ARW Joint Workshop on "Computer Vision and Robotics"*, 05 2016.

[ABI18] Mahdi ABIB. *Walking gait features extraction and characterization using wearable devices*. Theses, L'ÉCOLE CENTRALE DE NANTES, October 2018. URL: https://hal.archives-ouvertes.fr/tel-01969674.

[AL11] Andreas Aristidou and Joan Lasenby. FABRIK: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5):243–260, September 2011. URL: http://dx.doi.org/10.1016/j.gmod.2011.05.003, doi:10.1016/j.gmod.2011.05.003.

[Bak15a] Martin John Baker. Physics - kinematics, 1998-2015. [Online; accessed 1-June-2020]. URL: http://www.euclideanspace.com/physics/kinematics/index.htm.

[Bak15b] Martin John Baker. Physics - kinematics, 1998-2015. [Online; accessed 1-June-2020]. URL: http://euclideanspace.com/maths/geometry/rotations/conversions/quaternionToAngle/index.htm.

[BC19] Costas Boletsis and Jarl Erik Cedergren. VR locomotion in the new era of virtual reality: An empirical comparison of prevalent techniques. *Advances in Human-Computer Interaction*, 2019:7420781, Apr 2019. doi:10.1155/2019/7420781.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[DMWB13] Shreyansh Daftry, Michael Maurer, Andreas Wendel, and Horst Bischof. Flexible and user-centric camera calibration using planar fiducial markers. In *Proceedings of British Machine Vision Conference (BMVC)*. BMVC, September 2013.

[FALH20] R. Fribourg, F. Argelaguet, A. Lécuyer, and L. Hoyet. Avatar and sense of embodiment: Studying the relative preference between appearance, control and point of view. *IEEE Transactions on Visualization and Computer Graphics*, 26(5):2062–2072, 2020.

[GJnSMCMC16] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and R. Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481 – 491, 2016. URL: http://www.sciencedirect.com/science/article/pii/S0031320315003544, doi:http://dx.doi.org/10.1016/j.patcog.2015.09.023.

[GJnSMCMJ14] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014. URL: http://www.sciencedirect.com/science/article/pii/S0031320314000235, doi:http://dx.doi.org/10.1016/j.patcog.2014.01.005.

[HJ15] A. A. Hussen and I. N. Jleta. Low-cost inertial sensors modeling using Allan variance. *International Journal of Electrical and Computer Engineering*, 9(5):1237 – 1242, 2015. URL: https://publications.waset.org/10001443/low-cost-inertial-sensors-modeling-using-allan-variance.

[HKS+15] C. He, P. Kazanzides, H.T. Sen, S. Kim, and Y. Liu. An inertial and optical sensor fusion approach for six degree-of-freedom pose estimation. *Sensors*, 15:16448–16465, 2015. URL: https://www.mdpi.com/1424-8220/15/7/16448.

[J77] Bierman G J. *Factorization methods for discrete sequential estimation.* Academic Press, New York, 1977.

[KB99] H. Kato and M. Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pages 85–94, 1999.

[KHS17] M. Kok, J. D. Hol, and T. B. Schön. *Using Inertial Sensors for Position and Orientation Estimation.* 2017. URL: https://ieeexplore.ieee.org/document/8187588.

[KUY+18] D. Khan, S. Ullah, D. Yan, I. Rabbi, P. Richard, T. Hoang, M. Billinghurst, and X. Zhang. Robust tracking through the design of high quality fiducial markers: An optimization tool for ARToolKit. *IEEE Access*, 6:22421–22433, 2018.

[KZT+20] Felix Kosmalla, André Zenner, Corinna Tasch, Florian Daiber, and Antonio Krüger. The importance of virtual hands and feet for virtual reality climbing. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, page 1–8, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3334480.3383067.

[LHL12] B. Langmann, K. Hartmann, and O. Loffeld. Depth camera technology comparison and performance evaluation. In *International Conference on Pattern Recognition Applications and Methods*, 2012.

[Man16] Manash Kumar Mandal. Serialport. https://github.com/manashmandal/SerialPort, 2016.

[May79] P.S. Maybeck. *Stochastic Models, Estimation and Control*. Number pt. 1 in Mathematics in science and engineering. Academic Press, 1979. URL: https://books.google.at/books?id=eAdRAAAAMAAJ.

[NLL17] Diederick C. Niehorster, 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):2041669517708205, 2017. PMID: 28567271. arXiv:https://doi.org/10.1177/2041669517708205, doi:10.1177/2041669517708205.

[PS19] Ye Pan and Anthony Steed. How foot tracking matters: The impact of an animated self-avatar on interaction, embodiment and presence in shared virtual environments. *Frontiers in Robotics and AI*, 6:104, 2019. URL: https://www.frontiersin.org/article/10.3389/frobt.2019.00104, doi:10.3389/frobt.2019.00104.

[QK15] Qian-Yi Zhou and V. Koltun. Depth camera tracking with contour cues. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 632–638, 2015.

[RRMSMC18] Francisco J. Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76:38 – 47, 2018. URL: http://www.sciencedirect.com/science/article/pii/S0262885618300799, doi:https://doi.org/10.1016/j.imavis.2018.05.004.

[SNdS16] E. Sikström, N. C. Nilsson, A. de Götzen, and S. Serafin. Is this bridge safe? evaluation of audiovisual cues for a walk on a small bridge over a canyon. In *2016 IEEE Virtual Reality (VR)*, pages 285–286, 2016.

[TCG+07] Nigel W. Tierney, J. Crouch, H. Garcia, M. Walker, B. V. Lunen, G. DeLeo, George Maihafer, and S. Ringleb. Virtual reality in gait rehabilitation. 2007.

[WB06] Greg Welch and Gary Bishop. An introduction to the Kalman filter. *Proc. Siggraph Course*, 8, 01 2006.

[Woo07] Oliver J. Woodman. An introduction to inertial navigation. Technical Report UCAM-CL-TR-696, University of Cambridge, Computer Laboratory, August 2007. URL: `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf`.

[YH15] X. Yang and B. Huang. An accurate step detection algorithm using unconstrained smartphones. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 5682–5687, 2015. `doi:10.1109/CCDC.2015.7161816`.

[Zal08] Vincent Zalzal. KFilter - free C++ extended Kalman filter library, 2006-2008. [Online; accessed 24-February-2021]. URL: `http://kalman.sourceforge.net/index.php`.