BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE IN ENGLISH

DIPLOMA THESIS

I.M.U HAND TRACKING IN VIRTUAL REALITY GAMES

Scientific supervisor: Lect. Ph.D. Radu D. Găceanu

Author: George Vele

2019

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ ENGLEZĂ

LUCRARE DE LICENŢĂ

MONITORIZAREA INERŢIALĂ A MÂINII ÎN JOCURI DE REALITATE VIRTUALĂ

Conducător ştiinţific: Lect. Ph.D. Radu D. Găceanu

Absolvent: George Vele

2019

Abstract

This thesis is about an inertial method of tracking and integrating the human hand in a mobile VR environment. The main purpose of our device is to demonstrate that solely inertial sensors are viable for interacting with a virtual environment in a precise and reliable manner. The main disadvantage of inertial tracking is the error in measurements that accumulates over extended use, also known as drift, that appears when they are used for positional tracking. Therefore, we sought to solve this by creating a device, comprised of three sensors, wrapped around the user's arm that capture it's motion using forward kinematics.

In the recent years, virtual reality has become a field with real capabilities and potential. Although it is very good at providing the people with visual stimuli, there are still many technologies needed to be developed and perfected in order to immerse the player into a virtual world. This is why there is the need for methods that can enable people to interact with it in a more natural manner, that detect the movement of their bodies and translate them on the screen. In computer science, this is called motion tracking, and one way to do it is by using inertial tracking.

Inertial tracking refers to the use of sensors such as accelerometers, gyroscopes and even magnetometers in order to detect the target's position and orientation. This method is usually combined with optic devices for avoiding the drift that occurs.

The inertial device presented in this thesis performs well in tracking the hand using only its orientation. To demonstrate the capabilities of the system, a mobile VR ping pong game was created. The game was implemented to recognize collisions with the arm, as well as what forces should be applied when the player shoots the ping pong ball with the paddle. This paper looks at some of the challenges needed to overcome when implementing such a system, and details on the methods in which we have solved them.

The first chapter discusses about the papers domain, sub-domain and provides an overview on the existent technology relevant to our work. The second chapter entails theoretical background and introduces concepts necessary for the understanding of this paper. The third gives an overview about the technologies used and the reasoning behind using them. The fourth chapter describes our approach to motion tracking, the proposed architecture and it's implementation. The fifth chapter discussions on the challenges we have faced and presents the results we have obtained.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

George Vele

Contents

List of Figures

List of Tables

Introduction

This thesis is about an inertial approach to tracking and integrating the human hand in a mobile VR environment. We will study if a system based solely on inertial sensors can be viable for interacting with a virtual environment in a precise and reliable manner.

The main purpose of our system is to be used to detect and interpret the user's arm motions and transpose them accurately on the screen, by moving a virtual hand in the same way as the user's. In recent years, virtual reality has become a field with real capabilities and potential. Although it is very good at providing the people with visual stimuli, there are still many technologies needed to be developed and perfected in order to immerse the player into a virtual world. This is why there exists a domain that treats one of this problems, transposing the human body into the virtual experience by controlling elements on the screen using the motions of said body. This domain is called motion tracking, and there are many ways in which it can be implemented. Those methods include optical tracking, which makes use of cameras and computer vision, acoustic tracking, which makes use of sound waves, magnetic tracking, which makes use of magnetic fields based on the same principle as a theremin, and the method we are going to elaborate on, inertial tracking.

Inertial tracking refers to the use of sensors such as accelerometers, gyroscopes and even magnetometers in order to detect the target's position and orientation. The main disadvantage of this method is the error in measurements that accumulates over extended use, also known as drift. This is the reason inertial tracking is often combined with an optical method, and at the same time, the reason we wanted to demonstrate that a good system can be created using solely this method.

The way we sought to create this system is by attaching three inertial measurement units to the user's arm, on the hand, the wrist and upper arm. These units are combinations of accelerometesrs, gyroscopes and magnetometers that can work together to generate the position and orientation of the target, to a micro-controller, that in turn establishes communication with a mobile VR game.

The game we have created to demonstrate the capabilities of this system is a ping pong simulation. It determines the virtual hand orientation according to the information received by the motion tracker by mapping it to each segment of the arm and setting up their relative position in a hierarchical manner, method known as forward kinematics.

To this end, we have used the following hardware and technologies: 3 Adafruit BNO055

sensors, connected to a Raspberry PI 3 model B, Python, in order to create the connection between the Raspberry PI and the sensors, Unity, C# and the Google VR library for creating the game, as well as establishing communication with our tracker.

The thesis is structured in five chapters as follows.

Chapter 1, **Virtual Reality**, defines the domain of our paper, virtual reality. It presents the headsets that make it possible, and talks about motion tracking and the methods it can be achieved. I takes a look at the inertial system and presents the solutions that others have developed before us, which make use of inertial tracking for various purposes.

Chapter 2, **Technical Background**, introduces key concepts and technologies, giving a detailed explanation on how they work. Here we present the concept of micro-controllers, what are they good for and why are they so widely used. Also here we are talking about sensors, how micro electro mechanical systems differ from the normal ones, how does that help them in device integration, as well as how accelerometers, gyroscopes and magnetometers, methods of combining them, and what ways exist to make them communicate with a micro-controller or each other. Virtual reality is also presented here.

Chapter 3, **Hardware, Software and Technologies Used**, details and motivates the usage of various technologies. The Raspberry PI micro-controller was used as the brain of the motion tracker, it aggregates the orientations receivd from the three attached sensors and passes them on to our game.The Adafruit BNO055 inertial measurement unit has been used in order to detect the rotation of the hand in the three segments we are interested in. The I2C has been used in order to transfer that data from the BNO055 to the Raspberry PI, and from here TCP sockets were used to transfer it to the game. The Unity 3D game engine was used in order to create this ping pong game, and the Google VR library was used to enable VR functionality in it. During the process of development, VNC Viewer was used to interact with te Raspberry PI.

Chapter 4, **Development and Implementation**, provides the step by step process of creating our motion tracker. Here we are describing the architecture of the system. Here we explain the hardware connections, providing diagrams that show the way I2C looks in practice, wire by wire, and how to verify that it is hooked up properly. We show the algorithms responsible for data retrieval from the sensors and the ones that implement the TCP socket. We are providing a tutorial on how to recreate and use the ping pong game.

Chapter 5, **Results and Discussions**, puts forward some of the challenges that came up in the development of this system as well as the results we have obtained from the current version. A major challenge was to make it so that the paddle the player holds in hand will collide properly with the ping pong ball, applying forces based on the paddle speed.

Finally, there is the conclusions and future work. This chapter emphasizes the main points of the paper, while also providing the conclusion, the consequences of this study. Likewise, it also gives some ideas we are considering as a future work.

List of publications:

- **George Vele**. Hand Motion Tracking for Mobile VR Games. Sesiunea de Comunicari Stiintifice ale Studentilor, Cluj-Napoca (Romania), June 16, 2019. [42] — accepted

Chapter 1

Virtual Reality

In this chapter we will present this papers domain, virtual reality, and present methods for creating input based on the human body, i.e motion tracking. Among other methods, we will talk about inertial systems and provide an overview on the work already done in this field. In sections 1.1 and 1.2 we define virtual reality and then take a look at the different kinds of headsets that make it possible. In section 1.3 we provide an overview of some of the systems that can be used to detect the orientation and position of a target object and transfer it in a virtual environment. In section 1.4 we will discuss about some of the works that have been done on a similar subject.

1.1 What is VR?

Virtual reality refers to the computer-generated simulation of a 3D environment, with which it is possible for the user to interact to, with the help of head mounted displays, and all sorts of peripherals such as sensor fitted gloves [6].

Other people define it differently, but the most important aspects to be taken from this is that it refers to a simulation, and users can interact with that simulation in immersive ways. The level of interaction is not mentioned, so anything will do. For example, even 360° videos are considered virtual reality, as long as they have controls such as play, stop, volume and one can turn their head to see different parts of the video [5].

This definition does not really include the senses of a person, we experience the world through our senses and the more they are taken into account, the more the virtual world will feel real.

1.2 VR Headset

VR headsets allow people to immerse themselves in a 3D virtual environment or a 360° video. There are many types of them, with hardware specification ranging from a pair of glass lenses to intricate motion tracking [34].

Usually, they need to be connected to additional devices in order to make them work. These devices range from smartphones to expensive computers or gaming consoles.

There are known to be three distinct tiers of VR headsets, competing in power and price. The cheapest of the bunch are the ones like Google Cardboard, which only come with two lenses, one for each eye, and simple cases that act as support for them and the smartphone from which the media will be consumed. Those offer little interaction with the medium, the majority relying on gaze input and some having an extra button for clicking the screen.

When talking about mid-range headsets, we look at the likes of Samsung Gear VR, headsets that offer a little more comparing to the low-end, such as tracking sensors, build-in controls, focus wheels and even their own screen in some cases.

High-end headsets are the top of the line, as good as good goes. They offer the best VR experience when connected to a powerful computer. The main names here are HTC Vive, Oculus Rift and PlayStation VR, which offer controllers for both hands, with precise and very fast tracking.

(a) Google Cardboard V2 [34]

(b) Samsung Gear VR with controller [6]

(c) Oculus rift with controllers [2]

(d) HTC Vive with controllers [34]

Figure 1.1: Different types of VR headsets

1.3 Tracking Systems in VR

Motion tracking, or just tracking, refers to the process of following and capturing the position and orientation of an object, with the purpose of converting it to input for a VR application

[43]. This kind of input is usually used to control the representation of the users in the virtual space, and act as a basis for all their interaction with the application.

1.3.1 Optical Systems

Optical tracking refers to the process of acquiring the orientation and position of the users by means of interconnected cameras [43]. Those cameras are paired with a computer, usually via cables. In order to obtain correlated and correct information about the subject, they need o be calibrated. Depending on the elements used to keep track the object in the space, these systems can be classified as:

1.3.1.1 Optical Systems with Passive Marker

Those systems make use of a number of high-speed cameras, fixed around the measurement area, permitting the triangulation of a marker position. In this system, the markers are nothing more than little spheres covered with a reflective material. The cameras infrared light attached to them, shooting beams of light into the scene and detecting the spots where it hits a reflexive object [43].

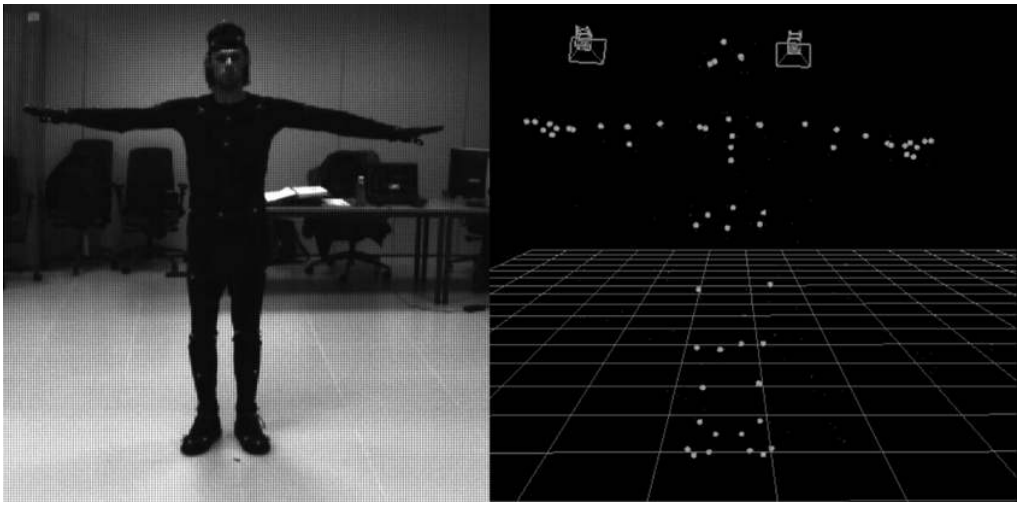


Figure 1.2: An actor in a MoCap suit with passive reflective markers and the 3D points detected by the cameras [44]

This method works very well when there is a direct line of sight between the marker and the camera. When an object obstructs this line of sight, the subject cannot be tracked [39]. To avoid this, the use of many cameras at very different locations is necessary, as the probability of all lines of sight to be obstructed reduces greatly. Because all markers are the physically identical, the cameras produce sets of unlabeled three-dimensional points. To make use of this information, algorithms are needed to identify, classify and provide meaning to those 3D points. The main disadvantage of using this method is because of the high computation needed to make sense of the data received and the probability for points to not be continuously detected. In spite of this, the accuracy of this system is incredible, with an error in the order of sub-millimeter.

1.3.1.2 Optical Systems with Active Marker

These systems are very similar in scope and implementation to the one we've talked about in the previous section. The main difference in approach resides in the markers. While the ones before were nothing more than objects coated in reflexive materials, these are equipped with infrared emitting diodes [43]. Each marker emits light that is detected by the stationary cameras, which detects it and marks its position. Because the infrared light on each marked can be set to different frequencies, it is no longer necessary to use complex post-processing in order to identify which points is which. Despite this, these systems does suffer from the same issues found in the one with passive markers, obstruction of the line of sight is still possible, as well as the data loss that comes with it.

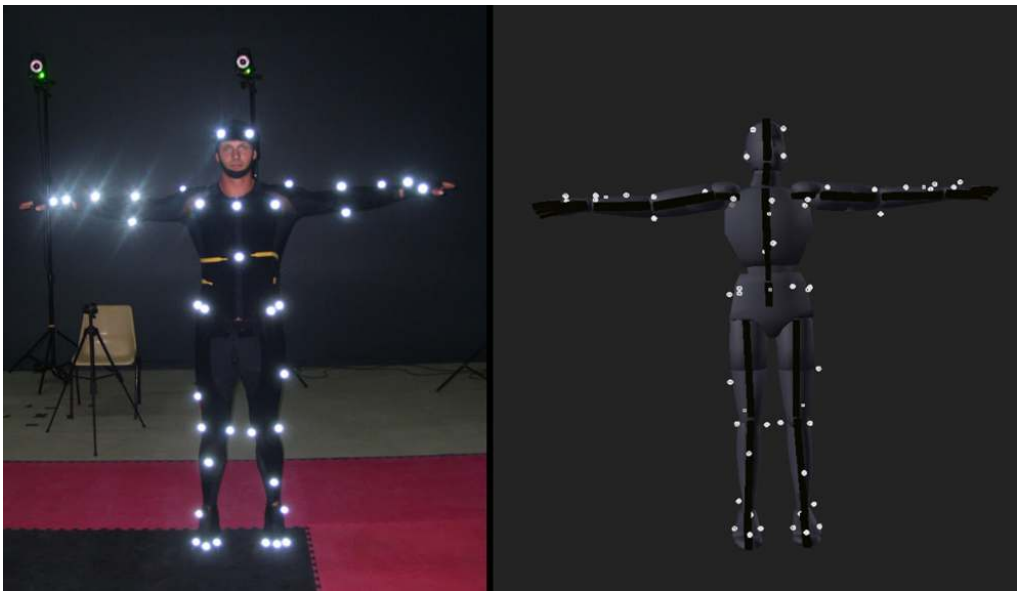


Figure 1.3: An actor in a MoCap suit with active reflective markers and the 3D points detected by the cameras [19]

1.3.1.3 Markerless

Between the three optical systems presented here, these are the ones need the least amount of cameras, requiring just a single camera in order to work. Despite this, they require the most effort in post-processing, as they need to use complex image detection algorithm to make sense of the targets figure. The accuracy of this kind of systems is reasonable, and it is said to have great potential [43].

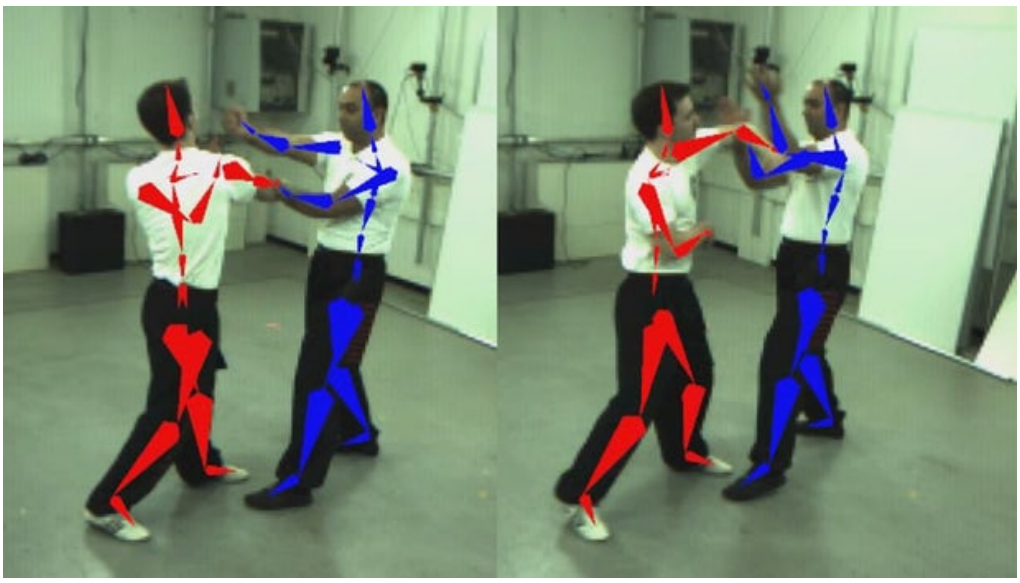


Figure 1.4: Two people being detected by a markerless system [19]

1.3.2 Non-Optical Systems

These kinds of systems do not rely on cameras, but on alternative methods to detect the targets orientation and position. There are a number of these systems in literature, some of which include:

1.3.2.1 Electromagnetic Systems

The implementation of these systems rely on electromagnetism. Those are the theories which express how magnetic fields interacts with electricity.

The way these work is that they use devices that generate magnetic fields in the three axis, known as transmitters, and sensors that detect those fields, called receivers. By having the voltage level on each axis that the sensor intercepts a magnetic field, an accurate orientation an position can be inferred [36].

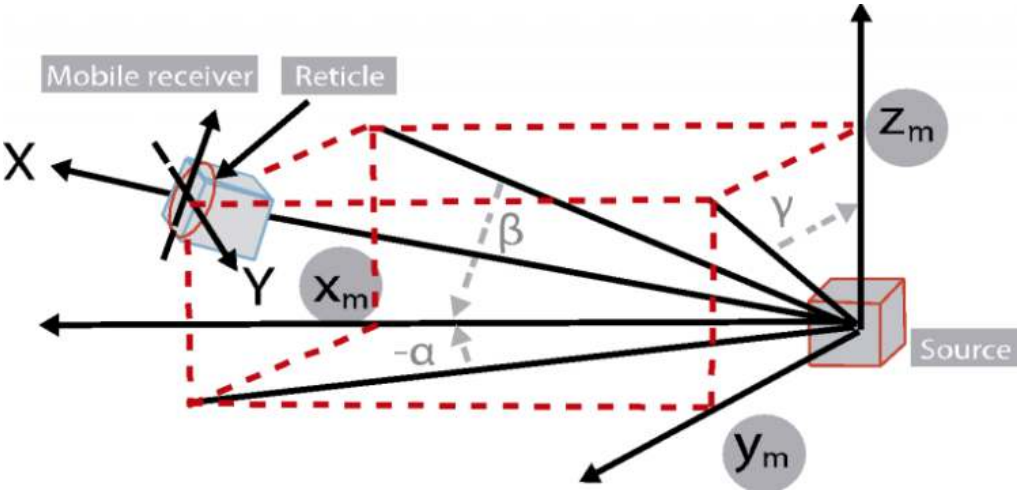


Figure 1.5: The interaction between a sensor and the electromagnetic field of a transmitter [36]

This method resolves the occlusion problems we have encountered with optical systems, as

objects cannot normally obstruct electromagnetic fields as they can with line of sight. However, the detection radius is limited to a small area around the transmitter, as the electromagnetic fields fades with distance, and there are still some objects that can interfere with the signal, especially metals [39].

1.3.2.2 Mechanical Systems

Mechanical tracking systems rely on a physical connection between the target and a fixed reference point. They were among the first tracking systems ever used. An example of a mechanical tracking device is the BOOM (Binocular Omni-Oriented Monitor) developed by Fake Space Labs [29]. This device is a robotic arm that the user must wear, that detects orientation an position based on the angles measured with gears or potentiometers.

As one can imagine, there are movement restrictions that come with these systems, so using it poses a challenge on the user. On the other hand, they are very fast [39].

1.3.2.3 Inertial Systems

These systems make use of inertial sensors such as accelerometers and gyroscopes. These systems are very small and can be placed on the target with ease to detect its orientation [43]. With the help of sensor fusion algorithms such as the Kalman filter and the complementary filter they are able to accurately combine information from those sensors and get reliable results.

While working with these sensors, a cumulative error in the orientation estimation can occur, this being the reason these systems are often paired with other tracking techniques.

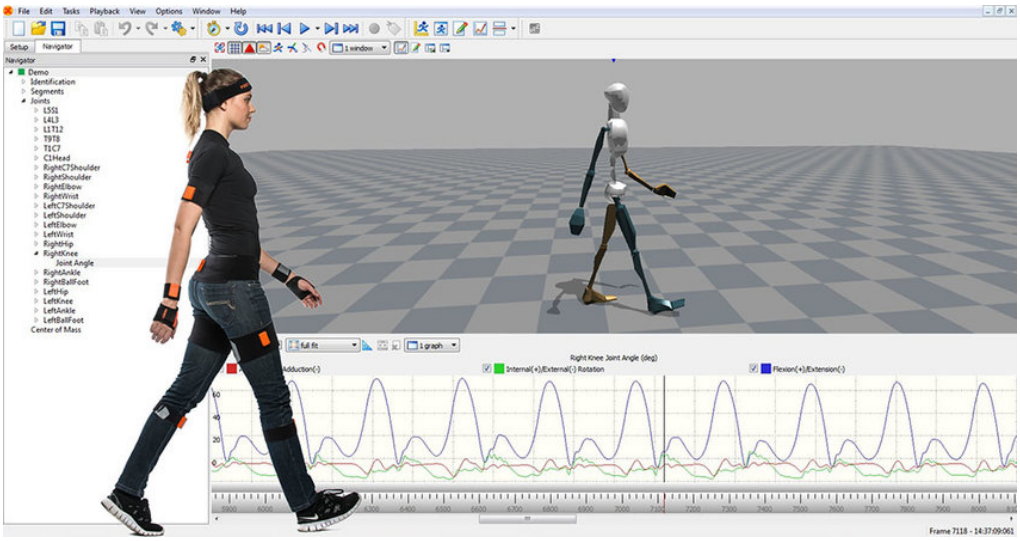


Figure 1.6: An actor wearing an inertial motion tracker [23]

1.4 Related Work

At the moment of writing this paper, the literature on exclusive inertial measurements for VR entertainment systems is scarce. Although in the recent years more and more methods

of capturing motion arise from start-up companies all over the world, most of them use a combination of inertial and optical systems.

There are a number of commercially available motion trackers on the market created for games, although most of them are not fully inertial, and because most of them are hand-held, they will only detect the hand motion, not the motion of the entire limb.

In [12] they created their own version of a dance game using four worn Wiimotes on the ankles and wrists. The way they recognized gesture was by setting up poses to be checked when the beat reached a certain point. Other dancing games have been created using inertial systems, such as [21] and [46]

In [4], a full body tracking system was created using 10 WIMU sensors attached to major joints such as upper arm, forearm, thigh, shin, trunk and pelvis, one of each on both sides. The system uses tracking information measured by the sensors and moves a virtual avatar in real time. In order to test this system, a VR game was created, where the player's goal is to shoot a target by pointing the right arm towards it . If the player hits the target, this explodes and another target is randomly generated in a different locacion. When the player hits the third target, the ones that follow after begin to move, increasing the challenge of the game. In comparison with our work, they were able to track more body limbs than our system, having used 10 sensors instead of 3.

In [45], a virtual game of Quidditch was created. Quidditch is a sport from the Harry Potter universe where the competing wizards have to score more points than the opposing team by throwing balls into floating circles, much like football, only they fly their magic brooms at all times. To recreate this, they used two IMU sensors on the player's arm, and one on both their prop broom and club. The player steered by moving the broomstick and using their club to hit incoming balls. They increased immersion by setting up multiple screens projecting multiple sides of the player. They do not, however, give many specific details on how exactly the tracked body interacts with the environment.

Both [47] and [31] created games that use a single sensor attached to the player's ear or foot respectively. In [47] they used an accelerometer to classify the player movements, if he was was jumping, running, walking, or leaning. Each of these actions was translated in the movement of a virtual dog, whose objective was to avoid all oncoming obstacles.Even though they had a simple setup, high recognition accuracy was achieved, with almost 99%. In the meantime, [31] uaed accelerometers and pressure sensors combined to control actions in a soccer game. The player still used the standard controller for some of the inputs, the system being used in running, passing and shooting. It was very important to them that one cannot cheat when using this system, as players are capable with the Wii.

In addition to those already presented, there are a number of papers that touch on the subject of hybrid systems, combining optical and inertial information to achieve said motion-tracking.

As an example of this, [1] presented a VR soccer game for physical rehabilitation. It uses an Intel RealSense sensor for motion capture. In this soccer simulation, players are put in

the shoes of a goal keeper and can use hand motions to defend against incoming balls. The rehabilitation session can be performed both supervised and automatic. Supervised in this case means that a physiotherapist decides from where the balls are thrown, when, and at what speed. Faster frequency of launches will need faster response time from the patient, and by adjusting the difference in position between consecutive shots they can more or less challenging for the patient. The automatic way is represented by an algorithm that decides those things instead of the physiotherapist, starting slow and making it harder based on the player's score. This score is calculated by recognizing key postures, as the trunk being straight and perpendicular to the ground, with the arms fully extended.

Chapter 2

Technical Background

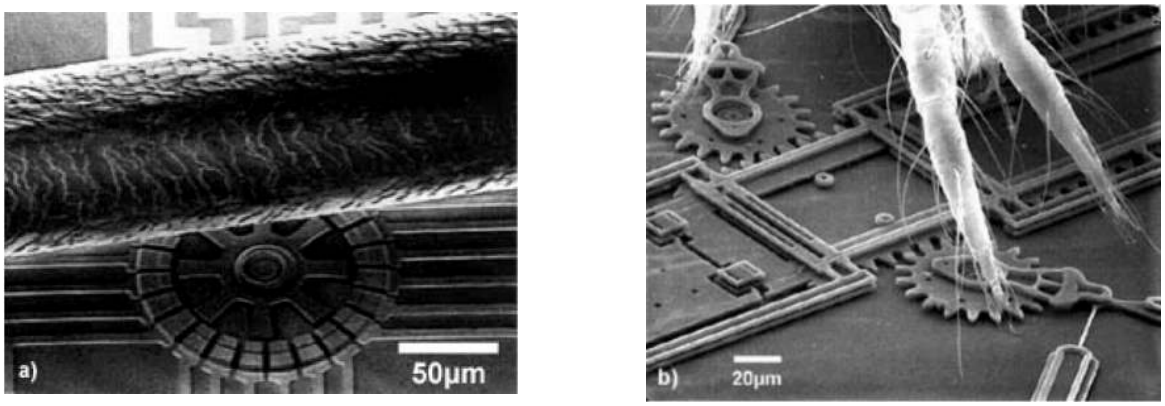
In this chapter we are taking a look at the concepts and technologies we will be referencing throughout the paper. Section 2.1 gives a detailed overview of the types of sensors that are of interest to us, namely accelerometers, gyroscopes, magnetometers, which together form inertial measurement units, as well about the technology that makes it possible to have these devices at a very small scale and easy to incorporate into gadgets. Section 2.2 presents the concept of sensor fusion, why it is needed and two very different methods in which it can be achieved. Section 2.3 talks about three of the more popular communication protocols that allow sensors to transfer information to microcontrollers, explaining their inner workings, advantages and disadvantages.

2.1 Sensors

2.1.1 Micro Electro Mechanical Systems (MEMS)

Micro-electromechanical systems (MEMS) is a process technology that in the broadest terms can be defined as scaled down mechanical and electrical components that are made utilizing micro-fabrication techniques [30]. The basic physical components of MEMS gadgets can vary from well beneath one micron on the lower end as far as dimensions go, all the way up to several millimeters. Similarly, the types of MEMS devices can vary from fairly basic structures having no moving elements, to amazingly complex electromechanical systems with multiple moving parts, under the control of incorporated microelectronics. The principal criterion of MEMS is that at the minimum there are some components having mechanical functionality, regardless of the movement capabilities of these elements. The term that defines MEMS is not globally consistent, this acronym being predominantly used in the US, while there are parts of the world in which it is called "Microsystem Technology" or "Micromachined Devices" [10].

Regardless of the nomenclature, the first thing to notice, and most important one, is the incredibly minuscule sizes these devices operate at.



(a) A MEMS silicon motor together with a strand of human hair [35]

(b) The legs of a spider mite standing on gears from a micro-engine [25]

Figure 2.1: Demonstrating the minuscule size of MEMS technology

2.1.2 Accelerometers

An accelerometer is an electromechanical gadget that measure acceleration forces, which can be static, like the constant force of gravity pulling us towards the earth or they can be dynamic, meaning movements or vibrations of the accelerometer [3].

Measuring the amount of static acceleration due to gravity, we can tell the angle the device is tilted at with respect to the earth.We can analyze the way it is moving by sensing the amount of dynamic acceleration.

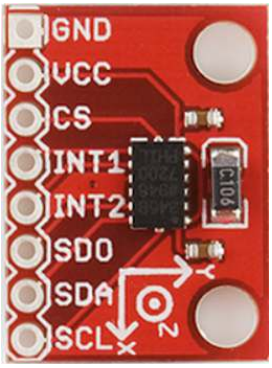


Figure 2.2: ADXL345, 3-DOF MEMS Accelerometer [38]

Acelerometers have may usages in both industry and science. They are components of inertial navigation systems for aircrafts and missles. They are used to detect and monitor vibrations in rotary machineries. Accelerometers are also used in tablets, phones, digital cameras, so that we know their orientation and display images in the right direction. Another interesting use is in drones, for flight stabilization.

Nowadays there are laptops that have accelerometers integrated in their case, that detect when a sudden freefall occures, and commands the hard drive to shut down, preventing loss of data [27]. In the same manner, accelerometers are used in cars to detect a crash and deploy airbags.

2.1.3 Gyroscopes

Gyroscopes are devices capable of measuring angular velocity and rotation motions. The units of angular velocity are measured in revolutions per second (RPS) or degrees per second. Angular velocity refers to the measurement of speed of rotation.

They can be used to in determining orientation and are found in most autonomous navigation systems. Many use cases revolve in rotational stabilization, such as keeping a robot on its feet, keeping a drone upright, stabilizing digital cameras on supports that do not jiggle when carried around. Most of these are implemented by measuring rotation with them and applying in some manner an force inverse to it, thus maintaining the orientation at the gadget's normal axis.

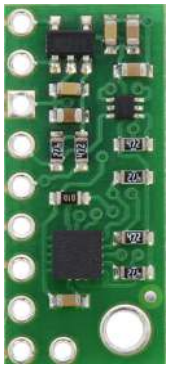


Figure 2.3: L3GD20H, 3-DOF MEMS Gyroscope [14]

2.1.4 Magnetometers

Magnetometers are devices that are capable of detecting the Earth's magnetic field along the three axes X, Y and Z. They utilize solid state technology in order to create miniature hall-effect sensors which in turn provide the sensors with the mean to detect fields.

This hall-effect sensor creates voltage proportional to the strength and polarity of the magnetic field along the axis it is directed towards. This voltage is then converted into digital signal, representing the intensity of the magnetic field. There are other technologies used for creating a magnetometer, which include resistive devices that change the measured resistance based on the fluctuations in the magnetic field.

A small electronic chip encloses the magnetometer, which often times incorporates other sensors, usually an accelerometer that is used for correcting the raw magnetic measurements of the magnetometer using their tilt information.

The magnetometers is vital in the orientation detection relative to the Earth's magnetic field of any device that needs it.

Figure 2.4: BMM150, 3-DOF MEMS Magnetometer [16]

2.1.5 Inertial Measurement Units

An inertial measurement unit, commonly referred as IMU, represents a sensor created from the combination of two or three other specific sensors. This means that it is comprised of a 3-axis accelerometer, 3-axis gyroscope, and sometimes even with a 3-axis magnetometer. When only containing the first two, we say about it that it is a 6-axis IMU, and when having all three we say that it is a 9-axis one. There are also manay IMUs that have more than these sensors, most often a thermostat. Although the reunion of these sensors, that give separate measurements through the common interface of one sensor is what most IMUs offer, there exist those that also incorporate micro-processor, loaded with a sensor fusion algorithm. This makes that the IMU can calculate valuable data such as the stabilized orientation of the device in quatrnions. This combination of sensors can also be found in the literature as referred to as AHRS, or Attitude Heading Reference System.

IMUs are commonly used in determining directions in GPS systems, motion tracking in various devices such as phones, video game remotes, virtual reality headsets and more. People have found many uses for them, especially in the IoT field. As one IMU serves basically as three other sensors, each use that we have listed for accelerometers, gyroscopes and magnetometers can naturally be achieved with an IMU.

Figure 2.5: MPU6050, 6-DOF MEMS IMU (Accelerometer + Gyroscope) [15]

2.2 Sensor Fusion

Combining information measured by two or more sensors, with the intent of obtaining higher quality data, is what is known in literature as sensor fusion [17]. It is done when the sensors we are using give related information that can be used to generate something better when combined than we could ever have hoped to get by using them individually. Most often, this is the case with accelerometers, gyroscopes and magnetometers, whose fusion can lead to excellent orientation measurements. For this purpose, there are many fusion algorithms, ranging from simple but acceptable, to complex bur very good. Two of these algorithms, finding themselves at the extremes of this spectrum are the complementary filter and the kalman filter respectively.

2.2.1 Complementary Filter

The complementary filter uses the data from the gyroscope, because it is very precise and not susceptible to external forces. On the long term, we use the data from the accelerometer, as it does not drift. In it's most simple form, the filter looks as follows:

$$angle = 0.98 * (angle + gyro * dt) + 0.02 * acc$$

Here the formula must be applied for the angle on each axis, where *gyro* and *acc* represent the angles of the gyroscope and accelerometer on that axis, and *dt* represents the sample rate, for example 0.01 for a $10ms$ sample rate [13]. The gyroscope data is integrated (added) every timestep with the current angle value. After this it is combined with low-pass data from the accelerometer (already processed with atan2). The constants (0.98 and 0.02) have to add up to 1 but can of course be changed to tune the filter properly

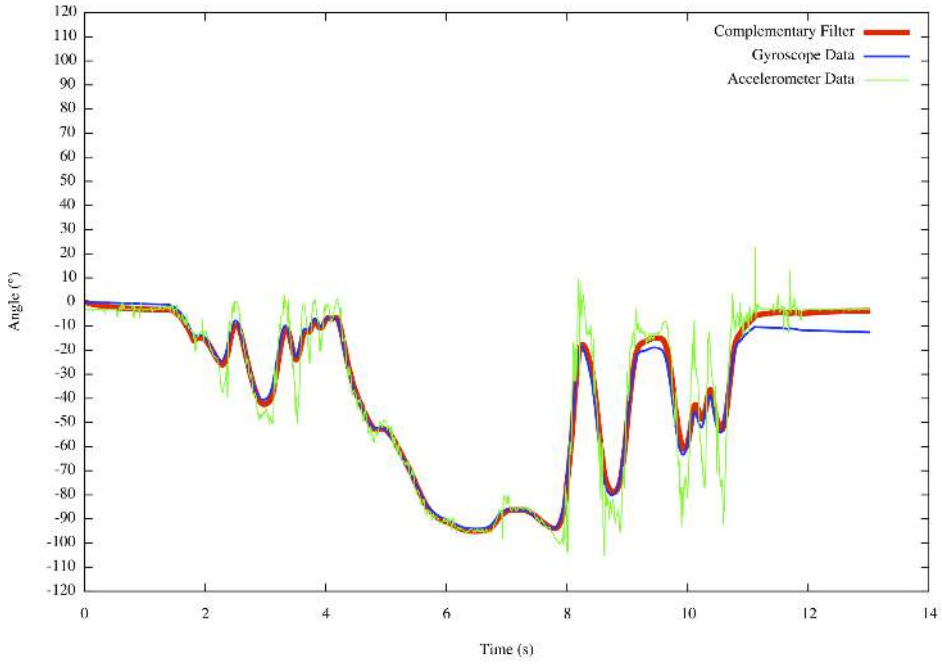


Figure 2.6: Complementary filter using Gyroscope and Accelerometer [13]

Every step the pitch and roll angle values are refreshed with the new gyroscope values by means of integration over time. The filter then checks if the magnitude of the force seen by the accelerometer has a reasonable value that could be the real g-force vector. If the value is too small or too big, we know for sure that it is a disturbance we don't need to take into account. Afterwards, it will refresh the pitch and roll angles with the data from the accelerometer by taking 98% of the current value, and adding 2% of the angle calculated by the accelerator. This will not ensure that the measurement won't drift, but rather that it will be very accurate on the short term.

2.2.2 Kalman Filter

Kalman filtering, also known as linear quadratic estimation in some circles, in statistics and control theory is an algorithm that uses a series of measurements observed over time, with inaccuracies such as statistical noise, in order to produce estimates of unknown variables that tend to be more accurate than those based solely on single measurements. It estimates a joint probability distribution over the variables of each time frame. This filter was named after Rudolf E. Kálmán, one of the primary developers of its theory.

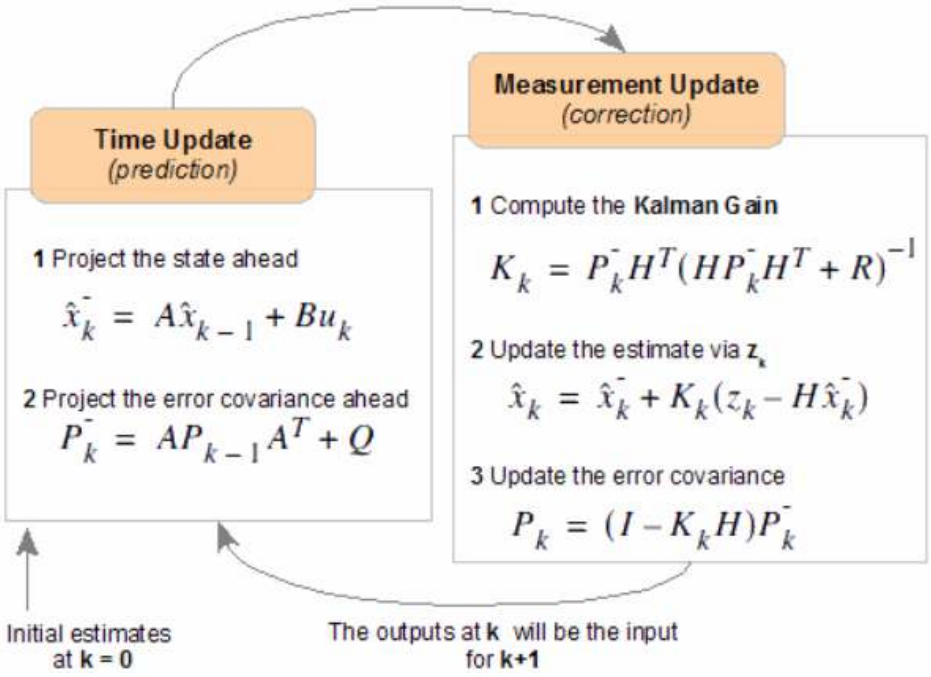


Figure 2.7: Kalman filtering alrogithm [18]

The algorithm works in a two-step process. In the prediction step, the it generates estimates of the current state variables. Once the outcome of the next measurement, which is certain to have some amount of error, random noise included, these estimates are updated using a weighted average, with more weight being given to the estimates with higher certainty. This is a algorithm. It it capable of running in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix, requiring no additional past information. This can be summed up in the following formula:

$$\hat{X}_k = K_k * Z_k + (1 - K_k) * \hat{X}_{k-1}$$

Here $\hat{X}_k$ represents the estimate of the signal that we wish to find, k being the time stamp of that signal. To find it we need Z_k (the measurement value), the Kalman Gain K_k and the previous estimation $\hat{X}_{k-1}$. Notice how if $K_k = \frac{1}{2}$, the current estimation will be the average between the current measurement and the last estimation, so the problem that Kalman solves is finding a suitable K_k for each iteration [18].

2.2.3 Calibration

Not all IMU's are made equal. Each and every one of them differs through microscopic imperfections that alter their functionality and measurement. Luckily to us, we can easily detect and make up for those imperfections by setting an offset specific to each sensor. This process is called calibration [37].

Without the calibration process, sensors would register different data even when placed in the same context as another one.

2.3 Sensor communication with Raspberry PI

Electronic devices communication much resembles the communication between people. In order for it to take place, they need to speak the same language. The equivalent of languages in electronics are communication protocols. Those need to be understood by both parties before any data transfer takes place, to make sure that the information is received and interpreted as expected. There are many protocols that electronics can follow, much like there are numerous languages humans can learn. Here we will take a look at three protocols that are commonly used in most household electronic projects. These are the Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), and Universal Asynchronous Receiver/Transmitter (UART) driven communication [8].

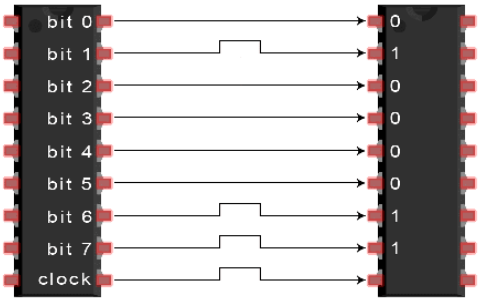
Besides them, thre are protocols like USB, Ethernet, Bluetooth and WiFi, which even though are a bit faster than the ones we will be talking about, they are also more complex and use more hardware and system resources. SPI, I2C and UART are ideal for inter-communication between microcontrollers, and between microcontrollers and sensors, where we are talking about smaller data [8].

2.3.1 Serial vs Parallel Communication

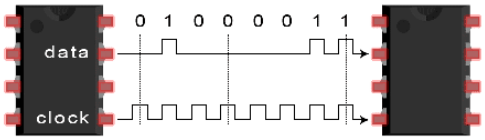
Electronics talk to each other by sending sequences of bits through physical wires, connected between them. Messages comprised of bits can vary in length, depending on the protocol being used, and also have special information such as check sums, acknowledgments that are indicated at key locations in the message. The bits are transferred from one device to another using rapid changes in voltage. In a system operating at 3.3V for example, a 0 bit is represented by short

pulse of 0 V, and a 1 bit is encoded as a short pulse of 3.3V. This is usually the case with most sensors that can be connected to a Raspberry Pi [8].

Messages comprised of bits can be sent either in parallel, where the bits are sent all at the same time, each through separate wires, either serial, where a single wire communicates all the bits one after another.



(a) Parallel [8]



(b) Serial [8]

Figure 2.8: Parallel and serial transmission of the letter “C” in binary (01000011)

2.3.2 Serial Peripheral Interface (SPI)

As common communication protocol, the SPI is used in many different systems. As an example, RFID card reader modules, 2.4 GHz wireless transmitter/receivers and SD card modules all utilize SPI in order to relay information to microcontrollers and vice versa.

In SPI, data can be transferred without interruption. We can send or receie information in a continuous stream, which is a unique benefit of this protocol. With the other two protocols that we are describing, data can only be sent in packets, with limited number of bits. In order to define these packets, start and stop conditions are set up, basically interrupting the data during transmission [8].

SPI’s model relies on the master-slave architecture. The master is the device that pulls the strings, usually a microcontroller, while the slave, which is a sensor, display or memory chip, takes instructions from the master. The base configuration of the SPI is comprised of a single master connected to a single slave. This design is salable, a master being allowed to control more thn one slave.

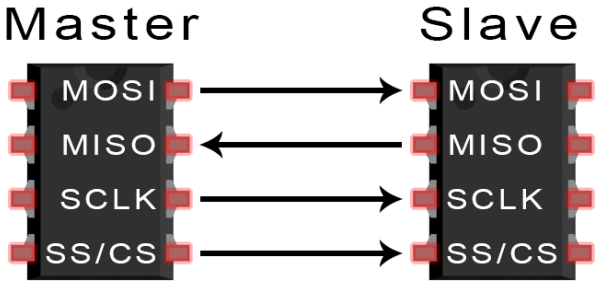


Figure 2.9: SPI connection with 1 master and 1 slave [8]

- MOSI (Master Output/Slave Input) – Used by the master to send data to the slave
- MISO (Master Input/Slave Output) – Used by the slave to send data to the master
- SCLK (Clock) – used to keep the clock signal
- SS/CS (Slave Select/Chip Select) – Used by the master to select to which slave it will send data to

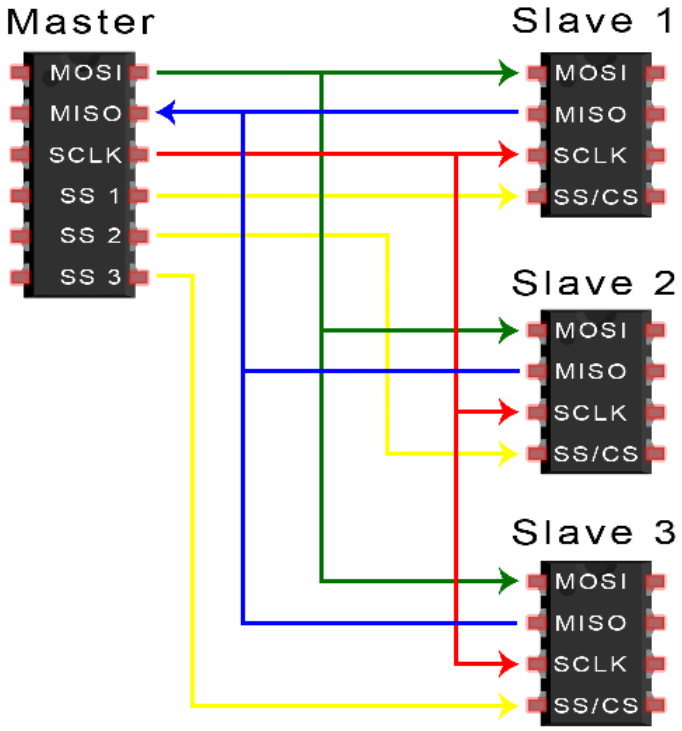


Figure 2.10: SPI connection with 1 master and 3 slave [8]

Advantages:

- Continuous stream of data, no packages and no start and stop bits
- Uses a simple selection system using the SS/CS line
- High data transfer rate, almost twice as fast as I2C
- Data can be sent and received at the same time due to separate MISO and MOSI lines

Disadvantages:

- Uses four wires, double comparing to I2C and UART
- No mechanism set up for checking if the message was successfully received, unlike I2C
- No method to check for errors, unlike UART
- We can only have a single master

2.3.3 Universal Asynchronous Receiver/Transmitter (UART)

UART, unlike SPI and I2C is not a communication protocol. It is a physical circuit, often integrated in microcontrollers, or found as a stand-alone IC. Its most important purpose is to transmit and receive serial data.

The UART acts more like a transformer, taking parallel information from a controlling device such as a CPU, converts it into a serial message and transmits it to another UART receiver, that in turn converts the serial message back to its original form for the receiving device. They only need two wires/lines in order to do all of this. Those are the Tx, the transmitter line, and the Rx, the receiver line. By connecting a transmitter pin to a receiver pin a communication line is established uni-directional from one UART to another. To make it bi-directional, the same process is applied from the other UART to the first.

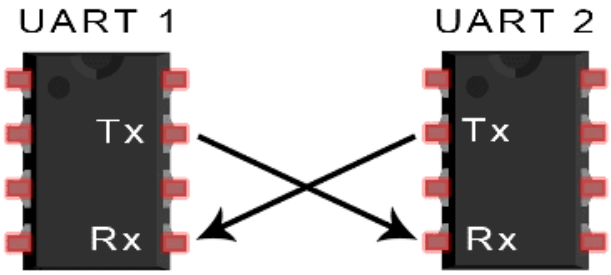


Figure 2.11: Connection between 2 UART devices [9]

Because they have two independent lines for sending and messages in both directions, UARTs communicate asynchronously. This means that they have no need for a clock signal to synchronize the data transfer. Instead of this, the UART that sends information adds a start and stop bit to the data packet it relays. Those bits mark the beginning and the ned of the packets, and so they know when to start and stop reading a message.

In order for two UARTs to communicate, they have to function at a specific frequency, also known as the baud rate. This is a measure of the speed with which data is being transferred, expressed in bits per second (bps). They use the baud rate to know at which frequency to send the bits and at which frequency to read them. Both must work at almost the same rate. This rate between them can only differ by about 10% before data is corrupted by errors caused when the timing is too far off.

In UART, beside the baud rate, one may want to customize the format of the data packets. This can be done, although not to a great extend, and both devices must be set to the same configuration. The usual packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional parity bit, and 1 or 2 stop bits.

In order for a device to be compatible with UART communication, it must feature a data bus with which the UART can interact. The information that the device wants to send passed through the data bus in a parallel manner, and then follows the same transformation as stated above. The UART adds a starting bit, a parity bit and a stop bit, creating and sending the data packet through Tx to the Rx of the other UART, from where the extra bits are checked

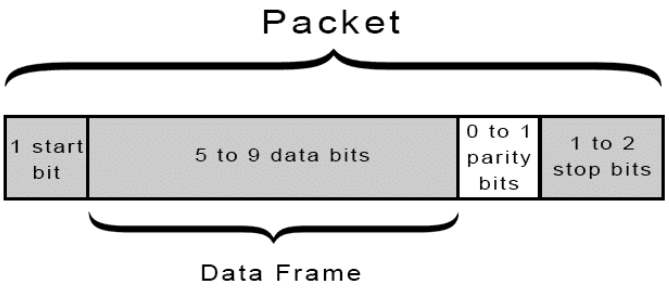


Figure 2.12: UART packet [9]

and discarded, leaving the message in to be transformed back and intercepted by the other data bus at the receiving end.

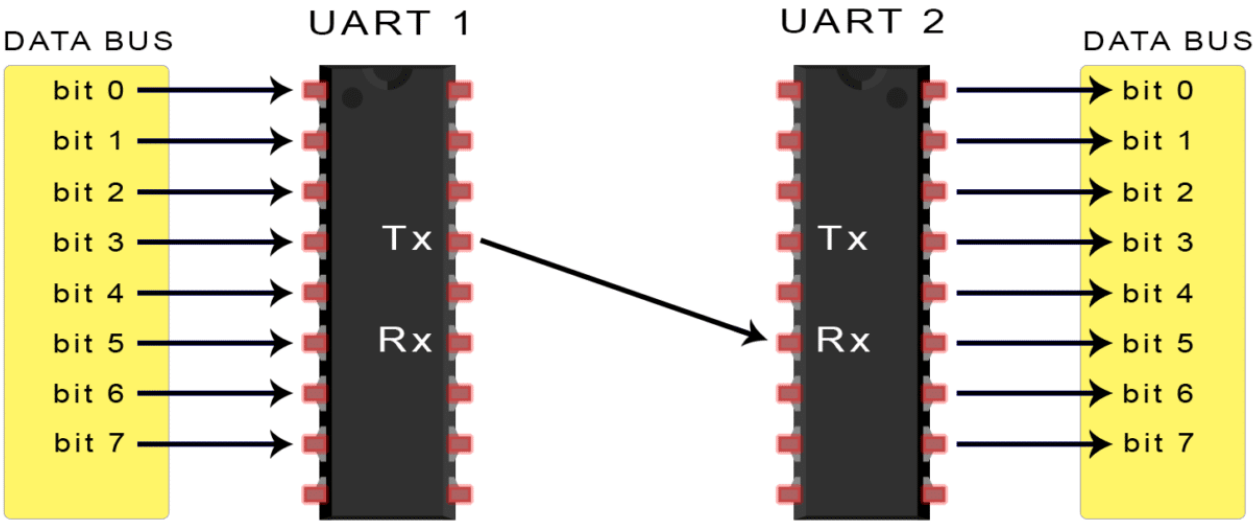


Figure 2.13: UART connection to the data bus [9]

Advantages:

- Only 2 wires are needed
- Data is send asynchronously, with no need of a clock signal
- Can check the message for errors using a parity bit
- Customizable packet format, as long as both sides use it
- Common method, with lot of documentation

Disadvantages:

- Message/data frame limited to a maximum of 9 bits
- Limited to a simple connection between 2 devices
- Both devices must have approximately the same baud rate

2.3.4 Inter-Integrated Circuit(I2C)

This protocol is best suited for creating complex systems that can have multiple masters with multiple slaves. I2C is a serial communication protocol, meaning that data is communicated bit by bit along a single wire, the SDA line. Resembling the SPI, I2C is synchronous, that means the data sent makes use of a clock line shared between the masters and the slaves. The clock signal is always controlled by the master.

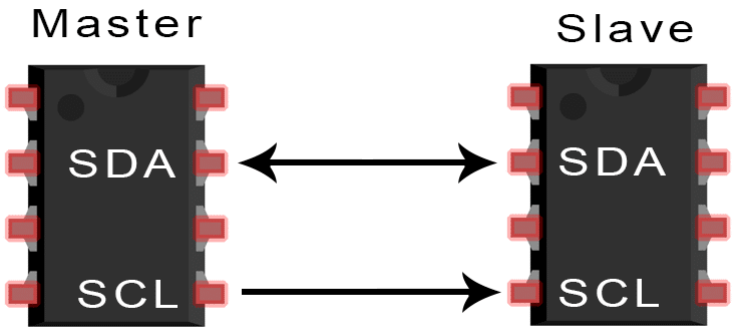


Figure 2.14: I2C connection with 1 master and 1 slave [7]

- SDA (Serial Data) – Used by both masters and slaves for transferring data
- SCL (Serial Clock) – Used by both masters and slaves to synchronize input/output

I2C relays its information in packets. Those packets are split into data frames. Each packet has an address frame that stores the address of the device it is intended to, and another in which the actual message is kept [7]. This package also makes use of the start and stop conditions, much like UART, the difference being that these conditions are created using a combination between the bit on the clock and the bit on the data line. Although it does not have a parity bit, it has acknowledgments after each frame, making the communication both more reliable and slower.

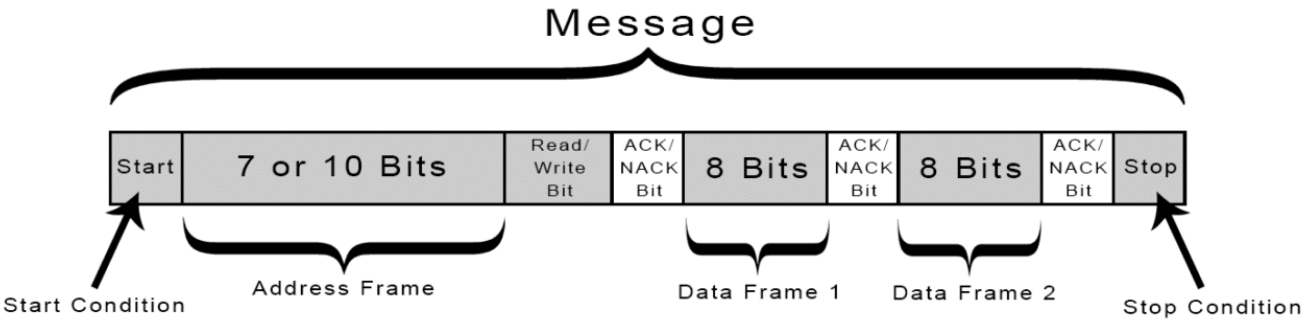


Figure 2.15: I2C packet [7]

Because the I2C does not use a slave select line like SPI, it needs another method to allow the master to pinpoint the slave it wants to send a message to. For this purpose, it implements

an addressing system. This system works much like addressing in computer networks. The master sends a package with the receiver's address on it which is transmitted to all the sensors connected. When the sensors whose address is contained in the package sees it, it sends a low voltage acknowledgment back to the master. The other slaves that do not match the address simply ignore it [7].

Due to this addressing mechanism, we can have multiple slaves connected to a single or multiple masters. The address frame is between 7 and 10 bits long, which can provide between 128 and 1024 unique addresses. Those numbers can even be increased by adding masters in the system, making the number of sensors one can connect with together with this protocol more than enough for almost anything.

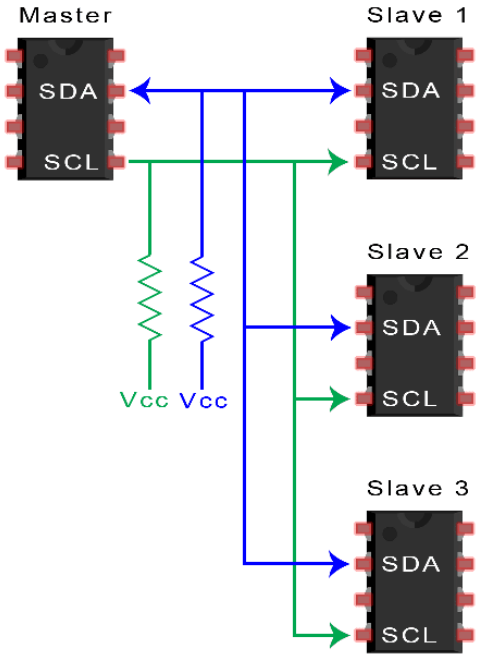


Figure 2.16: I2C connection between 1 master and 3 slaves [7]

Advantages:

- Needs only two wires
- Can connect multiple masters with multiple sensors
- It makes sure that each frame is transferred successfully, due to its multiple acknowledgments
- Requires less complicated hardware than UART
- Very common protocol with very good documentation

Disadvantages:

- Data transfer is slower than with SPI, due to the addressing system and acknowledgments

- Limited data frame, only 8 bits
- Harder to implement than SPI

Chapter 3

Hardware, Software and Technologies Used in the Thesis

In this chapter we are taking a look at all the elements that we have used in the development process of our motion tracker. Section 3.1 talks about the hardware elements, namely the Raspberry Pi and BNO055 IMU sensors, their capabilities and uses. Here we also state the type of VR headset we have used. Section 3.2 touches upon the software and technologies used and gives a brief description of the TCP protocol, TCP sockets, VNC protocol and the Unity 3D game engine.

3.1 Hardware

The hardware used may vary greatly, depending on the budget and skills of the person using it. One can create their own IMU by manualy combining the information of 3 different sensors, which is great because they come extremely cheap. On the other hand, there are IMUs that give out everything we ever wanted and more, like the BNO055, which come with all the sensor fusion necessary out of the box.

3.1.1 Raspberry Pi

Raspberry PI is an overall known brand that picked up in popularity after the success of it's single board computers. Here we will provide some of its history and give an overview of the inner workings of this device. In the development of this thesis, we have used the Raspberry PI version 3 model B, so this is the device we will be talking about.

The Raspberry is a powerful tool with capabilities similar to a desktop computer, although the hardware is not nearly as powerful.

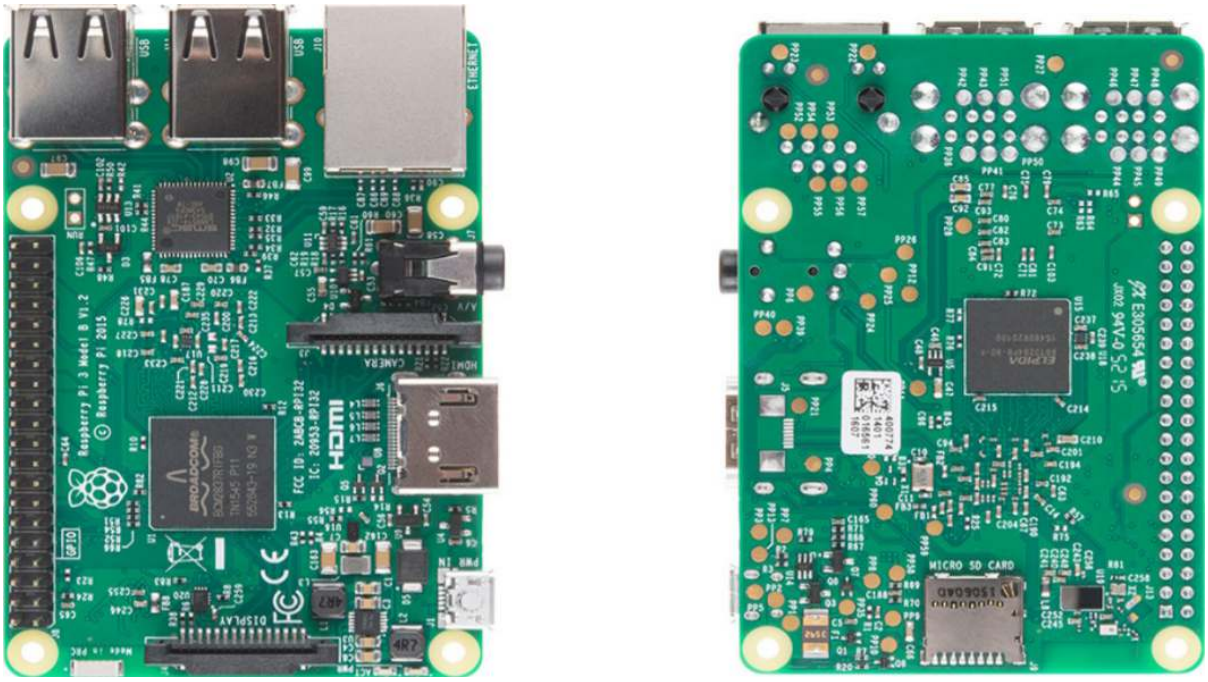


Figure 3.1: Rasperry Pi 3 model B [40]

3.1.1.1 History

The Raspberry PI Foundation project was initiated by a British engineer by the name of Eben Upton. He studied physics and engineering at the Cambridge University. Then he worked for prestigious companies, like Broadcom, Intel and IBM [32].

In the span of five years, While working ab Broacom, Eben spent most of his evenings and weekends working on his single board computer project.In the time that Eben Upton worked and created his prototypes, he became aware of one of the education problems in the UK, being the high prices for computers [32].

Because of this, many young minds did not learn computer sciences and in turn avoided the computer science professions which resulted in companies based in UK having a shortage of professionals [32].

By the year 2009, Eben has already created the Raspberry PI foundation, a registered educational charity foundation based in UK, to structure the Raspberry PI development. This device was to help young students to learn to program with a very low costs, that was his decision. This is the reasoning behind the Raspberry PI's low price.

Many computer manufacturers choose fruits as their name and logo, and the Raspberry is no different. By the time it came around, Apple, Acorns and Apricot were taken also by computers. It is no surprise that this foundation is named this way, looking at the trend of the time. But that is not the only reason, the name also being a funny reference to the expression "blowing a raspberry", which Elton said to be an allusion to the project at that time.

The "Pi" part comes as a nod to Python, this is because the first models started with a terminal prompt that required Python code in order to do work. It was the first of its kind running python, and this differentiation from its peers, which used BASIC, is also hinted in the choosing of "Pi" [32].

This single-board computer has some features that a single-board micro controller has, for instance the GPIO pins, see figure 1. The GPIO are a set of input and output digital pins and they have an important role to the Raspberry Pi. Those pins are exactly what we are going to use in order to physically connect sensors to the Raspberry via copper wires.

3.1.1.2 Technical specification

System on chip (SoC)

Built specifically for the new Pi 3, the Broadcom BCM2837 system-on-chip (SoC) includes four high-performance ARM Cortex-A53 processing cores running at 1.2GHz with 32kB Level 1 and 512kB Level 2 cache memory, a VideoCore IV graphics processor, and is linked to a 1GB LPDDR2 memory module on the rear of the board [40].

Figure 3.2: Broadcom BCM2837 [40]

General Purpose Input/Output (GPIO)

The Raspberry Pi 3 includes the same 40-pin general-purpose input-output (GPIO) header as all the Pis going back to the Model B+ and Model A+. Every existing GPIO hardware will work out of the box; the only change is a switch to which UART is exposed on the GPIO's pins, but that's dealt with internally by the OS [40].

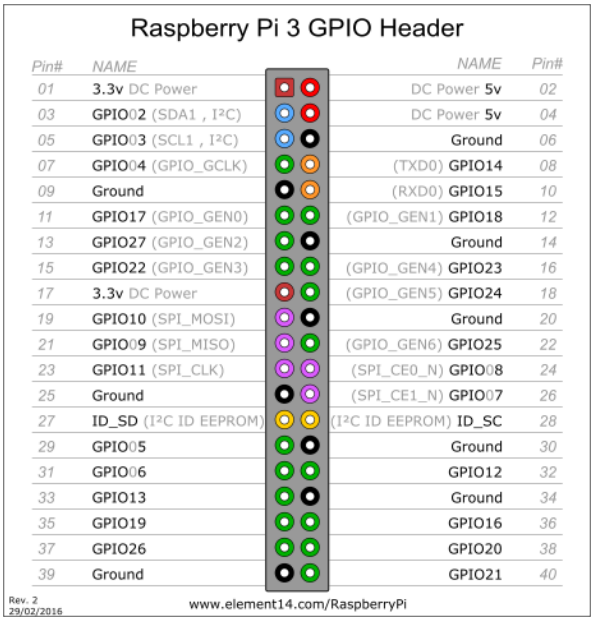


Figure 3.3: Raspberry 3 model B pins layout [26]

USB chip

The Raspberry Pi 3 has the exact same SMSC LAN9514 chip as its predecessor, the Raspberry Pi 2, adding 10% Ethernet connectivity and four USB channels to the board. As before, the SMSC chip connects to the SoC via a single USB channel, acting as a USB-to-Ethernet adaptor and USB hub [40] .

Antenna

The radios on the Raspberry Pi 3 are connected to a chip antenna soldered directly to the board, as to keep the size of the device very small. Due to this, there is no need for an external antenna to be connected. Despite its reduced size, this antenna is capable enough to pick up wireless LAN and Bluetooth signals, even with walls standing in the way [40].

Figure 3.4: Antenna [40]

Improvements from PI 2 Model B to Pi 3 Model B

The fundamental differences are the quad core 64-bit CPU and on-board Wi-Fi and Bluetooth. The RAM remains 1GB and there is no change to the USB or Ethernet ports. Be that as it may, the updated power the board should mean the Pi 3 can utilize more power hungry USB gadgets. For Raspberry Pi 3, Broadcom have supported us with a new SoC, BCM2837. This retains the same architecture as its predecessors BCM2835 and BCM2836. It is for this reason that all the projects and tutorials relying on the details of the Raspberry Pi hardware will continue to work. A custom-hardened 1.2GHz 64-bit quad-core ARM Cortex-A53 has replaced the old 900MHz 32-bit quad-core ARM CortexA7 CPU. As far as sizes go, the Pi 3 model B remains identical to the B+ and Pi 2, even the connectors and mounting holes stayed in the same place making all existing add-ons, HATs and cases fit just fine. As an exception, however, the power and activity LEDs have been moved to make room for the WiFi antenna. The the Pi 3 is roughly 50-60% faster than the Pi 2, making it ten times faster than the original Pi [40].

Software

Raspberry Pi by default runs on a Linux based Operating System, meaning that it allows the use of all the Linux compatible programs in it. Also, there are many Hardware on Top (HAT) that can be added directly to the Raspberry Pi.

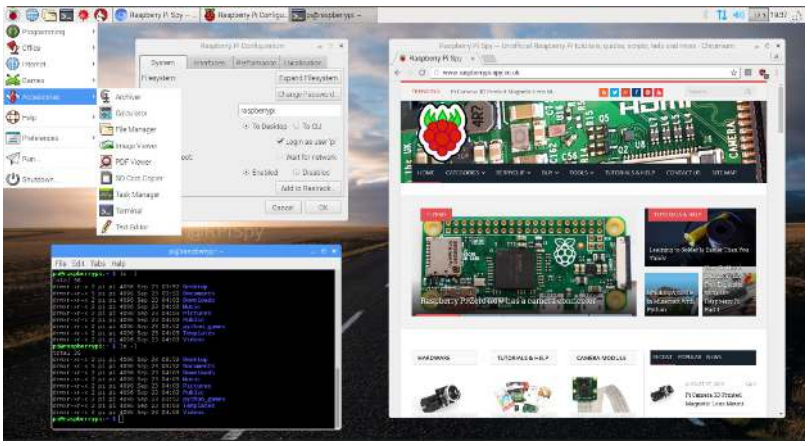


Figure 3.5: Raspbian, the most common operating system for Raspberry PI [28]

3.1.2 Adafruit BNO055

The sensor we have chosen is Adafruit's BNO055. This is a 9-DOF IMU, composed of an Accelerometer, Gyroscope and Magnetometer, together with a micro-processor loaded with a sensor fusion agorithm that computes very useful information out of the box [11].

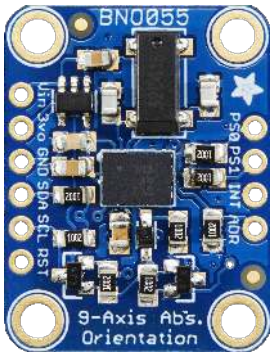


Figure 3.6: BNO055 [11]

The BNO055 can output the following sensor data:

- Absolute Orientation (Euler Vector, 100Hz)

  Three axis orientation data based on a 360° sphere

- Absolute Orientation (Quaterion, 100Hz)

  Four point quaternion output for more accurate data manipulation

- Angular Velocity Vector (100Hz)

  Three axis of 'rotation speed' in rad/s

- Acceleration Vector (100Hz)

  Three axis of acceleration (gravity + linear motion) in m/ŝ2

- Magnetic Field Strength Vector (20Hz)

Three axis of magnetic field sensing in micro Tesla (uT)

- Linear Acceleration Vector (100Hz)

  Three axis of linear acceleration data (acceleration minus gravity) in m/s$\hat{2}$

- Gravity Vector (100Hz)

  Three axis of gravitational acceleration (minus any movement) in m/s$\hat{2}$

- Temperature (1Hz)

  Ambient temperature in degrees Celsius

3.1.3 VR Headset

The headset we used is a spin-off the Google Cardboard V1. The mobile phone is placed inside it is ready to go. It has a magnetic trigger that can be used to click in game, but we won't use that functionality, mainly because it was deemed obsolete by the Google VR Plugin, about which we will talk about later.

3.2 Software and Technologies

The software can again vary greatly depending on the skills and budget of the person using it. We needed a way to connect to the Raspberry Pi in an efficient manner, and for that we used VNC. We also needed a game engine that supports mobile VR, and we chose Unity because it is very used friendly and highly documented. For transferring information from the Raspberry Pi to the mobile VR game we used TCP sockets, as explains in the following section.

3.2.1 TCP Protocol

TCP is a transport layer protocol. it is used mainly by applications for which the integrity of the packets delivery is important. This is because it provides ordered, reliable and error-checked delivery of streams of data [24].

3.2.2 TCP Sockets

Processes running on various machines speak with one another by sending messages into sockets, utilizing a TCP connection. Information can be exchanged as though there was a direct virtual pipe between the client and server. The TCP protocol will ensure that all information sent will be received, in the same order that it was sent [22].

3.2.3 VNC Protocol

Virtual Network Computing (VNC) is a graphical desktop sharing protocol used to control remotely another computer [33]. It can transmit keyboard and mouse events from our PC to the one that is sharing its desktop to us. There can be more PCs that share their desktop to us, and there can also be more PC that those computer share to, basically creating a many to many relationship between them.

3.2.4 Unity

Unity 3D is a video game engine for 2D / 3D development that has been supporting virtual reality and augmented reality since December 2015 [20]. Using Unity, it is possible to deploy over 28 platforms, including Android and IOS. Unity is free but also offers paid subscriptions that offer additional features. Those features range from analytics, cloud storage and more [41]. Unity has a big community that provides different assets or plugins in its marketplace, the Unity Asset Store, free or paid.

Figure 3.7: Unity logo [41]

Unity has been pushing its VR / AR development and is the most used platform for virtual reality development. Using the newest version (2019.1.2), Unity VR offers deployment for Oculus Rift, HTC Vive, PSVR, Android & IOS, HoloLens and Windows Mixed Reality (Unity for VR and AR, s.d.).

Unity also supports C# as a programming language which is a very common scripting language.

3.2.4.1 Google Virtual Reality for Unity (GVR)

GVR is a plugin for Unity that allows the usage of VR headsets in development and deployment of games. It comes with custom cameras, scenes and classes, and needs to be imported into Unity as any other asset would.

Chapter 4

IMU Hand Tracking for a Ping Pong Simulator

In this chapter, we present the steps taken in order to create the motion tracker, application, and how to connect them. Section 4.1 gives an overview on the proposed architecture and explains the role of each part. Section 4.2 highlights some we wished to achieve before starting implementing, such as setting up VNC, to be able to control the Raspberry remotely and creating the physical I2C connections. Section 4.2.0.2 shows how we have gone about receiving data from the sensors to Raspberry, the process of calibration, initialization an usage of the algorithm that controls the sensors. Section 4.4 details the server/client model used in communicating between the Pi and our PC. Section 4.5 presents the game implementation step by step, from setting up the project to support mobile VR, implementing an enemy and most importantly creating the virtual arm and making it match our motion.

4.1 Architecture

The architecture we propose follows the schematic below. We will connect 3 IMU sensors to a micro-controller, using a communication protocol that will allow us to transfer data fast and safe between the two of them. The 3 sensors will be strapped on pieces of fabric designed to be wore on the back of the hand, the wrist and the upper arm.They will be responsible for detecting the orientation of their corresponding arm segment and translating it into quaternions, that will then be transmitted by wire to the micro-controller. The micro-controller will then connect wireless to a mobile device that runs our VR game, to whom it will transmit the data received from those sensors. Once this information is in the game, we will rotate a virtual arm that will correspond to the right hand of the player. At this level, collisions are detected and resolved using techniques that blend in with the game engine that we use.

We will design the game in the Unity 3D game engine, using C# as the scripting language and the Google VR (GVR) library for handling camera controls regarding the head movements and preparing the game to be viewd in a virtual reality headset. As our micro-controller we've chosen the Raspberry PI 3 model B. Sensory data are received and transmitted through scripts

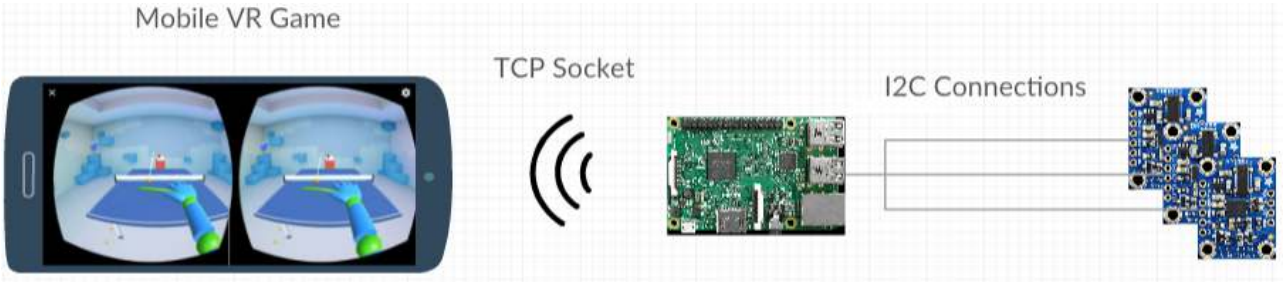


Figure 4.1: Architecture of the system

written in Python. About the sensors we will talk in the next section.

4.2 Preparation

Before starting to work on the game there are some things that must be done, such as establishing a connection to the Raspberry, connecting the sensors to the raspberry and creating a new scene in unity for our game.

4.2.0.1 Setting up VNC to support remote control of Raspberry Pi

Regardless of whether another screen, keyboard and mouse are available, one may still wish to interact with the Raspberry through VNC, because of just how easy it makes it to switch from the computer to the Raspberry with the mouse, without any externals. Although, if a screen is present, connecting it to the computer and placing the VNC window on it will give each device it's own space, and the transition from one to another with the mouse is nothing different than what one would be used to working in a dual monitor environment. It may seem like a detail, but the faster the interaction with the Raspberry, the faster the development time we can expect.

To connect through VNC, which comes preinstalled on the Raspbian, we start it on both devices, and then insert the PI's IP into the VNC on our computer.

There are multiple ways of finding the Raspberry's IP. The first and most straight forward of them is to have a HDMI monitor available and a USB mouse, and we can see the ip in the VNC interface on the PI. Another method we often used is to use a program to search all the IP addresses on the network. We can't recommend this method on just any network, as it can be attract unwanted attention. A third method is to look in the router's DHCP table, if we have access to it.

4.2.0.2 Connecting sensors to Raspberry Pi over I2C

We had 3 BNO055 IMUs which we wanted to connect to the Raspberry using I2C. The BNO055 supports only 2 addresses to be used at the same time on one I2C bus, so for connecting the third one we opened another bus.

In the image above is presented the wiring done in order to achieve the I2C connection. As

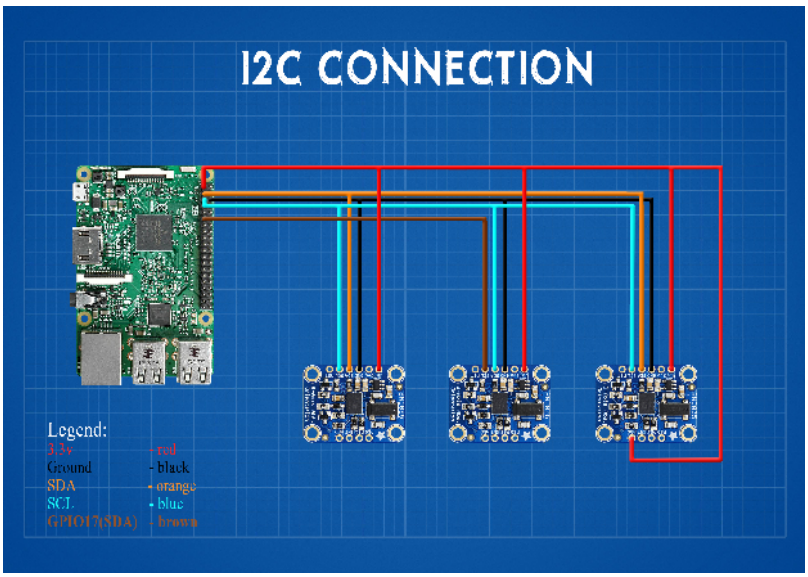


Figure 4.2: I2C connection schematic

stated above, we had to open 2 different buses to connect the sensors. One can see those 2 buses in the wiring. Each bus requires 4 wires going from the sensor to the raspberry. Those are the 3.3v and the ground lines, which are used for powering up the sensors, SCL, which represents the clock, as a measure to synchronize the sensors with the raspberry. SDA, or the data line, is used to transfer information. The only difference between the buses is the data line, bus 1 uses the default SDA pin, while the second bus uses GPIO17 as a SDA. One can see that the 3.3v line as 2 connections to the third sensor, this is because this sensor changes its I2C address if connected this way, assuming address 0x29.

4.3 Receiving Sensor Data from BNO055

4.3.0.1 Calibrating sensors

In order to calibrate the sensors, we can use the algorithm implemented by Adafruit for BNO055. By starting the web example given and doing specific motions with the IMU, we can get the values necessary to calibrate each component sensor. The accelerometer, gyroscope and magnetometer that composes the IMU each have certain motions one has to go through in order to get a good calibration value. Those are:

- Gyroscope: One can leave this device standing still, in any position
- Magnetometer: Normal movement of the sensor should be enough to calibrate it. By moving it to create an infinity sign in the air we can be sure that it will calibrate.
- Accelerometer: This is the most challenging one to calibrate. The best method to do it is to get a cube of some sorts, and place the sensor on each face of the block. Even if this is not perfectly calibrated, the BNO055 should still give quality data.

4.3.0.2 Using Adafruit Algorithm

The easieset way to read sensor data from raspberry is do dowonload the Adafruit Python BNO055 from GtHub and make use of their BNO055 class. Each BNO055 object must be constructed using the bus number and address. We will then set the calibration of the objects using the files we obtained in the calibration phase. The quaternion of each sensor can then be read using a simple 'read_quaternion' method.

```
from Adafruit_BNO055 import BNO055

# initialize the sensors
# hand and shoulder are on bus 1, elbow is on 4
bno_hand = BNO055.BNO055(rst=18, busnum=1)
bno_elbow = BNO055.BNO055(rst=18, busnum=4)
bno_shoulder = BNO055.BNO055(rst=18, busnum=1, address=BNO055_ADDRESS_B)

if not bno_hand.begin():
raise RuntimeError('Failed to initialize hand BNO055!')

if not bno_elbow.begin():
raise RuntimeError('Failed to initialize elbow BNO055!')

if not bno_shoulder.begin():
raise RuntimeError('Failed to initialize shoulder BNO055!')

# calibrate sensors using calibration files
with open('calibration_hand.json', 'r') as calibration_file:
data = json.load(calibration_file)
bno_hand.set_calibration(data)

with open('calibration_elbow.json', 'r') as calibration_file:
data = json.load(calibration_file)
bno_elbow.set_calibration(data)

with open('calibration_shoulder.json', 'r') as calibration_file:
data = json.load(calibration_file)
bno_shoulder.set_calibration(data)

while True:
  x1,y1,z1,w1 = bno.read_quaternion()
  x2,y2,z2,w2 = bno_elbow.read_quaternion()
  x3,y3,z3,w3 = bno_shoulder.read_quaternion()

  with open('data.txt', 'w') as file:
    file.write('{} {} {} {} {} {} {} {} {} {} {} {}'
      .format(x1,y1,z1,w1, x2,y2,z2,w2, x3,y3,z3,w3))
```

Listing 4.1: sensors.py

4.4 Transferring Data from Raspberry PI

4.4.0.1 Raspberry Client

In the client.py, the ip of the server must be manually set up. The script then connects to that ip through a tcp socket connection and sends data from sensors to the server, that is the game.

```
import asyncio
import json

# Asynchronous communication with Unity 3D over TCP
async def tcp_echo_client(loop):

# open connection with Unity 3D
reader, writer = await asyncio.open_connection(ip, 8080, loop=loop)

message = ''

with open('data.txt', 'r') as file:
message = file.read()

# convert JSON to bytes
encoded = message.encode(encoding='UTF-8',errors='strict')

# send message
writer.write(encoded)

# close the socket
writer.close()

loop = asyncio.get_event_loop()
while True:
  try:
    loop.run_until_complete(tcp_echo_client(loop))
  except:
    pass
loop.close()
```

Listing 4.2: client.py

4.4.0.2 Unity Server

Using the server script, the game receives positional tracking data from the client. It will then pass the information to the script controlling the hand, each hand part taking the rotation that the sensors tell it to take.

```
private void ListenForIncommingRequests () {
  try {
    // Create listener on localhost port 8080.
    tcpListener = new TcpListener(IPAddress.Parse("0.0.0.0"), 8080);
    tcpListener.Start();
    Debug.Log("Server is listening");
    Byte[] bytes = new Byte[1024];
    while (true) {
      using (connectedTcpClient = tcpListener.AcceptTcpClient()) {
        // Get a stream object for reading
        using (NetworkStream stream = connectedTcpClient.GetStream()) {

          int length;
          // Read incomming stream into byte arrary.
          while ((length = stream.Read(bytes, 0, bytes.Length)) != 0) {

            var incommingData = new byte[length];
            Array.Copy(bytes, 0, incommingData, 0, length);
            // Convert byte array to JSON message.
            String clientMessage = Encoding.UTF8.GetString(incommingData);

            lock (handTracking.input)
            {
              handTracking.input = clientMessage;
            }
          }
        }
      }
    }
  }
  catch (SocketException socketException) {
    Debug.Log("SocketException " + socketException.ToString());
  }
}
```

Listing 4.3: server.cs

4.5 Creating the Game

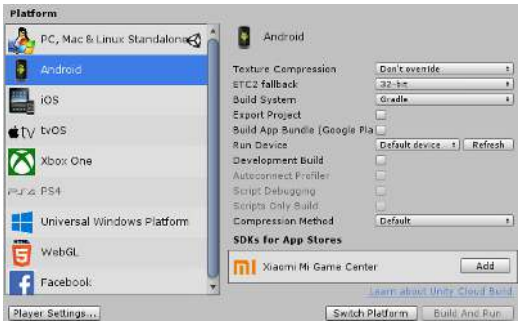
4.5.1 Setting Up the Project

The first thing we are going to do after creating a new 3D project from the Unity launcher, is to change the platform we are building this game for. To to this, we go to *File→Build Settings*. We will be greeted with the setting windows where we switch the platform to Android. After this we want to open the *Player Settings ...*, go to *XR Settings*, and check the *Virtual Reality Supported* checkbox and add the Cardboard SDK. VR SDK is supported only since Android 4.4 KitKat, so we need to change the minimum API level this game is build for. To do this,

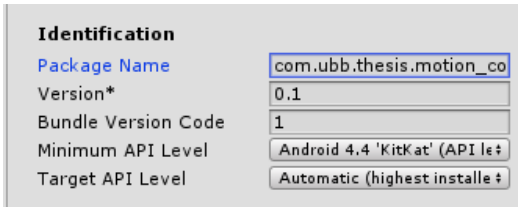
from the same interface where we've found *XR Settings* we scroll up and enter *Other Settings*. Here, under the *Identification* label we should change the minimum API level to 19. Also here we change the Package name of our application to something unique.

Now the project uses Unity's default SDK that will enable us to bulid VR games, but for faster development and more features we install Google's VR Kit. To do this search for it on Google and download the file with the "unitypackage" extension. Then, from Unity go to *Assets* → *Import Package* → *Custom Package...* and choose the downloaded file.

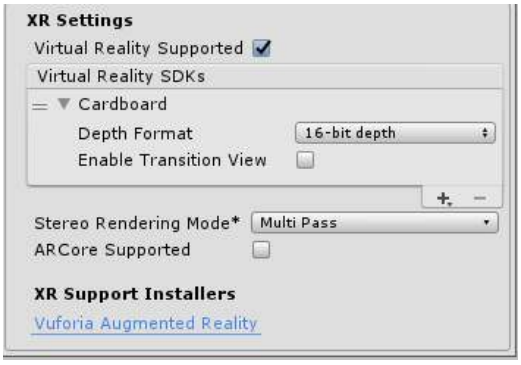
Now everything is set up and we can begin to work on the game.



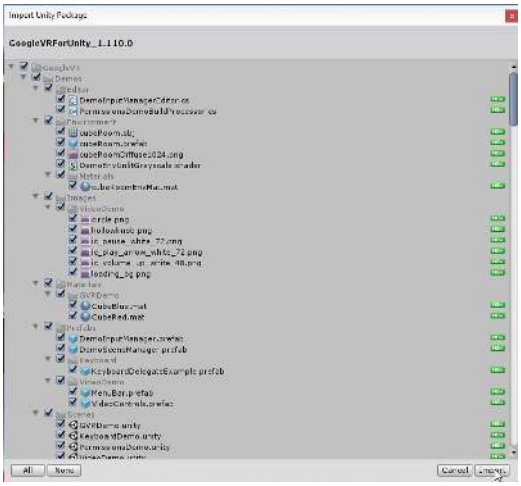
(a) Platrofm selection



(b) Select Package name and API level



(c) XR Settings



(d) GVR import

Figure 4.3: Steps for setting up the project

4.5.2 Implementing the Enemy

The first thing after placing a room and a ping pong table in the scene, we sought out to make the enemy. We thought the complexity it should have and we settled on a stationary enemy, or cannon, that shots balls at the player, much like in a training exercise. We wanted the ping pong balls it shoots to look and feel accurate, so we toyed around with its physics, giving it a *Rigidbody* component, a *Sphere Collider* and setting its mass to 1, drag to 0.2 and angular drag to 0.05 and, which are the values we felt would most accurately represent the ball.

4.5.3 Making the Virtual Arm

After creating a scene with a ping pong table and an enemy that shoots balls, the next step was to create and attach an arm to our player and use the measurements received from the Motion Tracker in order to animate it accurately in the 3D space. Since the scope of this project does not include finger motion, the paddle was fixed to the hand as to mimic it's default position. As long as the player does not move his/her fingers a lot, the immersion should still hold.

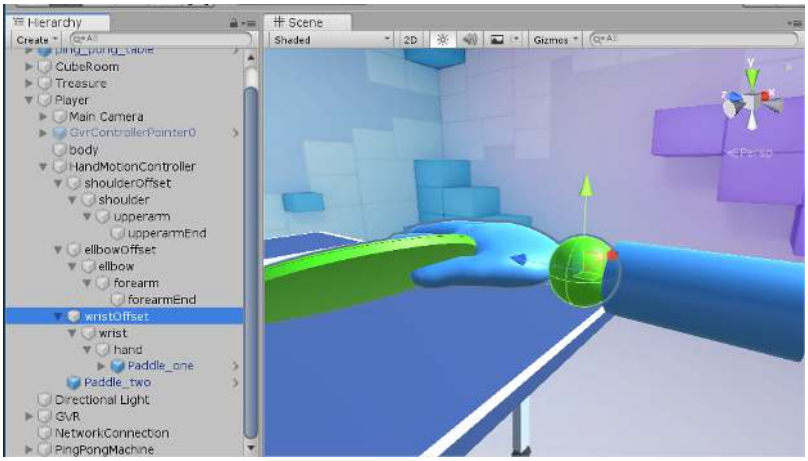


Figure 4.4: On the left one can see the hierarchical structure of the hand, while on the right side we have the offset object selected

Our approach was to use forward kinematics in order to achieve this. Forward kinematics consists in having multiple segments connected to each other in a hierarchical manner, where the motions of the children are determined by the motions of their ancestors.In this manner,we separated the virtual arm into 3 parts: hand, forearm, upper arm.Each part receives specific rotational information from the sensor corresponding to their body part, and their final position and rotation in the 3D space is calculated based on the parent segment. Each segment has an endpoint, at which the next should start, and each frame the child segment transform will be set to that position. The rotation of each segment is globally set to the rotation received as one can see in the code snippet bellow.

```
public void Rotate()
{
  wristRotation = new Quaternion(x1, -z1, y1, w1);
  elbowRotation = new Quaternion(x2, -z2, y2, w2);
  shoulderRotation = new Quaternion(x3, -z3, y3, w3);

  wrist.transform.localRotation = wristRotation;
  elbow.transform.localRotation = elbowRotation;
  shoulder.transform.localRotation = shoulderRotation;

  wristOffset.transform.position = forearmEnd.transform.position;
  elbowOffset.transform.position = upperarmEnd.transform.position;
}
```

Listing 4.4: Rotation of the segments based on received orientations

Because the rotation on the z axis of each sensor is calculated based on their starting point, they are prone to give different measurements even if they stay in the same positions. Due to this, after starting the game and connecting the motion tracker, the arm will bend in unnatural/impossible ways. To accommodate for this problem, we have created a wrapper for each segment, an empty object parents that will have the rotation equal to the offset necessary to make the whole arm point forward. This offset will be set once the player looks at a sphere in the left side of the room and, assuming the player points their arm forward before doing this, the virtual arm will overlap with their own and start behaving naturally. The code bellow shows how exactly the offset is set for each segment.

```
1  public void Straighten()
2  {
3    wristOffset.transform.localRotation
4       = Quaternion.Inverse(wristRotation);
5    elbowOffset.transform.localRotation
6      = Quaternion.Inverse(elbowRotation);
7    shoulderOffset.transform.localRotation
8      = Quaternion.Inverse(shoulderRotation);
9
10 }
```

Listing 4.5: Straightening segments i.e setting offsets

Chapter 5

Results and discussions

When we started the game, we expected that the we would move our hand, hit the ball with the paddle, and Unity would detect the necessary collisions and apply the corresponding forces to the ball. This however was not happening as we expected. If we kept the paddle stationary in front of the ball, the collision would be detected correctly, however if we moved the paddle in an arc motion , the faster we moved it, the greater the chances of the ball passing through it. This was because the incompatibility between the way we were rotating the hand and the way Unity physics are calculated. In Unity, there is a physics engine that is responsible for collisions, applying forces, and other physics-related functions. This engine will update at fixed time intervals, such as 0.02s not being affected by the frame-rate at which the game runs. Unity provides a way to interact with this engine, via the FixedUpdate() method, which will execute the code every time the engine updates. Because moving the hand is not a physics related task, should not be called within this method, but from the Update() method. The difference between the two being that the latter will execute the code once per frame, being totally dependent on the frame-rate of the game. Here we manually set the rotation of the segments to the one received by the motion tracker.

Although to the player, the movement of the hand seems pretty smooth, to the game, a hand motion would equal it being in a point in one frame, and in another in the next, without knowing anything about the road it took from one point to another. The faster the speed, the greater the gaps between the positions from frame to frame, and greater the chance of the ball missing the paddle completely. Normally, within the Update() method, one would call Rotate() or Move() in order to change the transform of an object in a physics specific way. Those functions however have 2 major disadvantages that prevented us from using them. First, they require an amount to rotate an object by, but we had only the final rotation we wanted our object to take, and needed to calculate every frame the difference between the current rotation and the desired rotation, which is tricky using quaternions, and error prone using euler angles. Secondly, they require a time in which the rotation / movement should take place, if we set it to 0, it is the same as manually setting the rotation, if we set it higher, the player will notice a lag between his/her movements and the virtual movements.

The solution we came to is to have an invisible cylinder following the movement of the

paddle, having it handle all the ball collisions. We set up the cylinder in a manner that covers the entire surface of the paddle, equipped with a 'rigidbody' component and a script that uses FixedUpdate(), in which it tells the cylinder to move to move to the paddle position and rotation with a certain sensitivity. This method not only lifts the unnecessary rotation calculations that would be performed if we set the entire hand to update in the FixedUpdate(), but also prevents the hand from being laggy, the lagginess being transferred to the invisible cylinder. As a bonus, we could split the cylinder into smaller parts and treat each one as a separate object, with their own velocity and colliders. This would have the effect that different forces would be applied to the ball, depending on which side of the paddle it would hit, improving the immersion of the game.

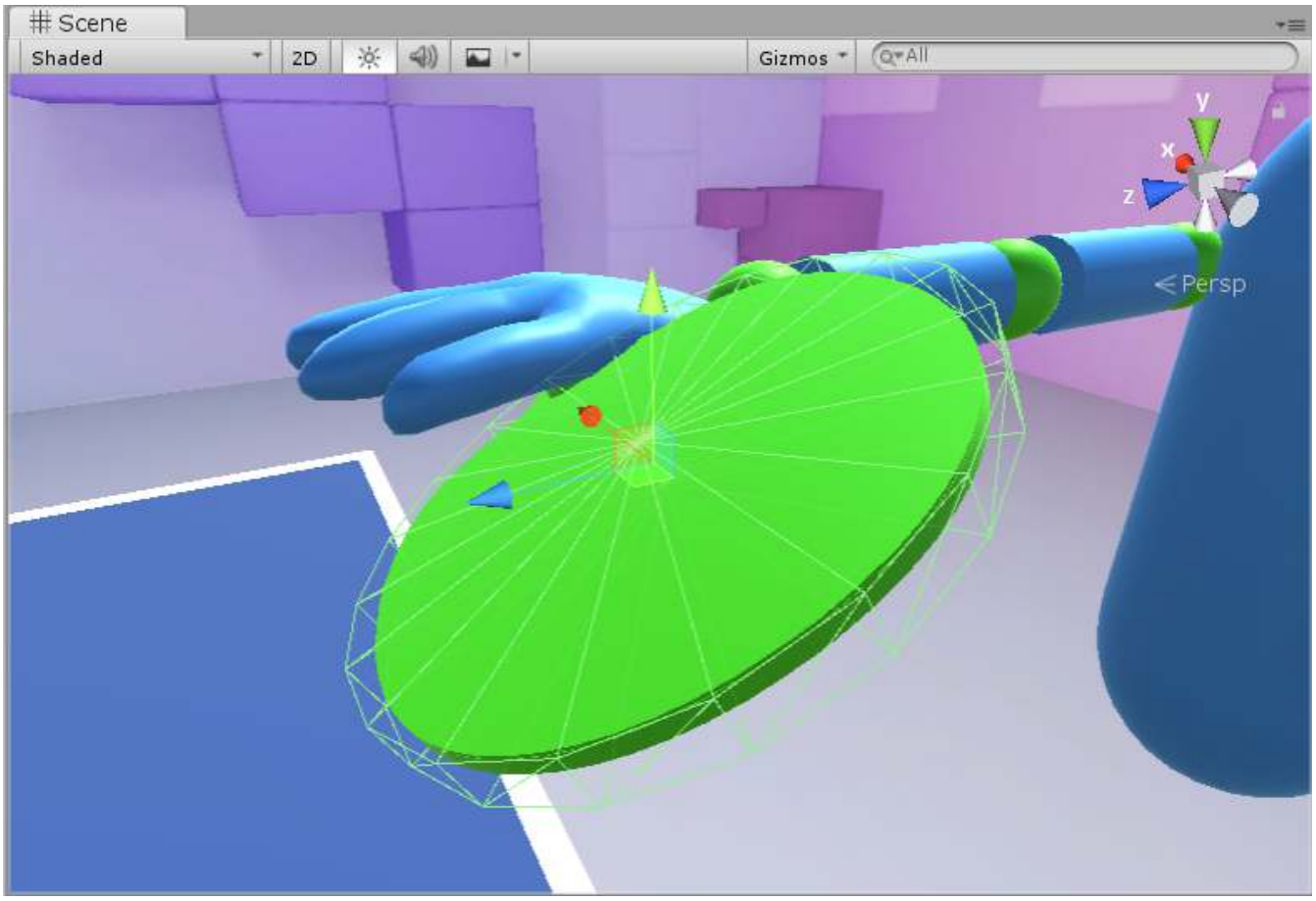


Figure 5.1: Here we can see the cylinder's wire-frame, this is what balls collide into when hitting them

At this stage in the project, we have successfully created an intuitive motion tracker that is fun, easy to use and precise.It behaves very well in several areas. The design is comfortable and does not hinder at all the player's ability to move. The accuracy of the captures is very good, fully capturing all the rotations the arm can make and translating them on the screen in a believable manner.

As previously stated in this paper, one of the constant challenges of motion tracking is the speed at which one can compute the measurements. In the inertial approach, this relates to the speed of the sensor fusion and sensor transmission. We have measured this speed in the number

	Min	Avg	Max
FPS	9	21	50

Table 5.1: Arm frame-rates

of frames the virtual arm receives new rotational information from the sensors per second.

Using this measurement, we have seen that the average speed is around 21 frames per second (FPS).This being said, the frame-rate tends to vary, at times hitting around 50 FPS and other times reaching as low as 9 FPS.

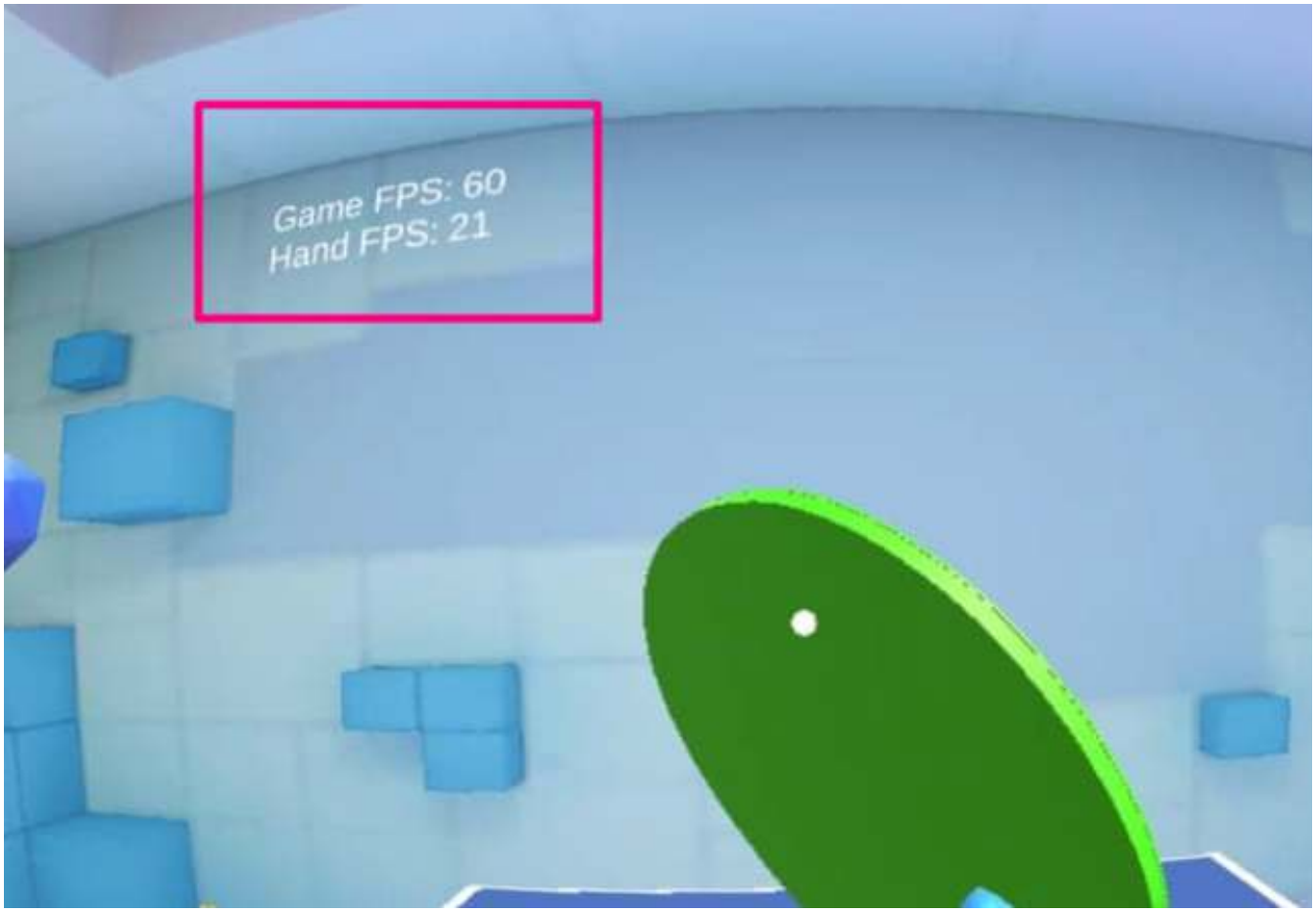


Figure 5.2: The game FPS and the hand FPS displayed in the upper left corner of the room

Future work and Conclusions

With virtual reality being used in more and more industries, such as physical rehabilitation and games, it is very important that we have methods to interact with it. Because the ultimate purpose of virtual reality is to fully immerse the user in its environment, more effort and research is being invested in fields that can make that dream come true. One of this fields is motion tracking, based on which we can create systems that act as an interface between the human body and the virtual world. Our goal was to provide a fully inertial method for detecting, interpreting and transposing the user's arm in such a virtual environment.

In conclusion we can say that this can be achieved by creating a system composed of 3 IMU sensors strapped to the player's arm, a Raspberry PI and that it can be integrated successfully with a mobile VR ping pong game made with Unity. We have provided detailed explanations regarding the connection protocols, collision handling, the range that this system is capable of measuring as well as the speed at which it does it.

It was shown based on our own implementation, as well as other people's work, that tracking human body parts with inertial measurement unis not only can be done, but is also working well.

Although the device functions well, there exists a lot of space or improvement. Below are presented some of the ways in which we think it can be upgraded in the future:

- Using SPI as a protocol for communication between micro-controller and sensors
- Optimizing the frequency at which the communication protocol operates
- Using a faster language, such as C or C++ for measurements retrieval
- Using UDP sockets instead of TCP ones
- Operating Raspbian from a console line interface, to optimize the resource allocation to the sensors
- Cost reduction can be achieved using a microcontroller like Arduino instead of Raspberry, and using a cheaper IMU, such as the MPU6050 and combining it with a custom sensor fusion algorithm

Bibliography

[1] A.Baldominos, Y. Saez, and C.G. del Pozo. An approach to physical rehabilitation using state-of-the-art virtual reality and motion tracking technologies. *Procedia Computer Science*, 2015.

[2] Alison. Oculus rift vr headset: Immerse into the world of virtuality. Online: https://boringportal.com/oculus-rift-vr-headset/, Accessed on 22.06.2019.

[3] Matej Andrejašic. Mems accelerometers. *Marec*, 2008.

[4] D. L. Arsenault and A. D. Whitehead. Quaternion based gesture recognition using worn inertial sensors in a motion tracking system. *Games Media Entertainment(GEM)*, 2014.

[5] S. A. Aseeri, D. Acevedo-Feliz, and J. Schulze. Poster: Virtual reality interaction using mobile devices. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 127–128, March 2013.

[6] Joe Bardi. What is virtual reality? Online: https://www.marxentlabs.com/what-is-virtual-reality/, Accessed on 22.06.2019.

[7] Circuit Basics. Basics of the i2c communication protocol. Online: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/, Accessed on 22.06.2019.

[8] Circuit Basics. Basics of the spi communication protocol. Online: http://www.circuitbasics.com/basics-of-the-spi-communication-protocol, Accessed on 22.06.2019.

[9] Circuit Basics. Basics of uart communication. Online: http://www.circuitbasics.com/basics-uart-communication/, Accessed on 22.06.2019.

[10] Irad Ben-Gal. An introduction to mems. *PRIME Faraday Partnership*, 2002.

[11] Created by Kevin Townsend. Adafruit bno055 absolute orientation sensor. Online: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bno055-absolute-orientation-sensor.pdf, Accessed on 22.06.2019.

[12] E. Charbonneau, A. Miller, C. A. Wingrave, and J. J. LaViola Jr. An exploration of 3d spatial interfaces for dancing. *Symposium on 3D User Interfaces*, 2009.

[13] Pieter-Jan Van de Maele. Reading a imu without kalman: The complementary filter. Online: https://www.pieter-jan.com/node/11, Accessed on 22.06.2019.

[14] Optimus Digital. Giroscop pololu cu 3 axe l3gd20h. Online: https://www.optimusdigital.ro/ro/senzori-senzori-inertiali/2981-giroscop-pololu-cu-3-axe-l3gd20h.html?search_query=giroscop&results=43, Accessed on 22.06.2019.

[15] Optimus Digital. Mpu6050 accelerometer and gyroscope module. Online https://www.optimusdigital.ro/en/inertial-sensors/96-mpu6050-accelerometer-and-gyroscope-module.html, Accessed on 22.06.2019.

[16] dzduino. Bmm150 magnetometer compass geomagnetic sensor. Online: https://www.dzduino.com/bmm150-magnetometer-compass-geomagnetic-sensor, Accessed on 22.06.2019.

[17] Wilfried Elmenreich. An introduction to sensor fusion. Research Report 47/2001, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2001.

[18] Bilgin Esme. Kalman filter for dummies. Online: http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies), Accessed on 22.06.2019.

[19] Jacek Hordyj. Inverse kinematics of anthropomorphic structures for vision systems applications. 10 2015.

[20] Jason Jerald, Peter Giokaris, Danny Woodall, Arno Hartbolt, Anish Chandak, and Sebastien Kuntz. Developing virtual reality applications with unity. pages 1–3, 03 2014.

[21] A. Kailas. Basic human motion tracking using a pair of gyro + accelerometer mems devices. *International Conference on e-Health Networking*, 2012.

[22] Limi Kalita. Adafruit bno055 absolute orientation sensor. *International Journal of Computer Science and Information Technologies*, 2016.

[23] Angelos Karatsidis. *Kinetic Gait Analysis Using Inertial Motion Capture: New Tools For Knee Osteoarthritis*. PhD thesis, 09 2018.

[24] Gary C. Kessler. An overview of tcp/ip protocols and the internet. Online https://www.garykessler.net/library/tcpip.html, Accessed on 22.06.2019.

[25] Sandia National Laboratories. Online: http://www.mems.sandia.gov, Accessed on 22.06.2019.

[26] Dr. Lawlor. Raspberry pi gpio via mmap. Online https://www.cs.uaf.edu/2016/fall/cs301/lecture/11_09_raspberry_pi.html, Accessed on 22.06.2019.

[27] Dimension Engineering LLC. A beginner's guide to accelerometers. Online: https://www.dimensionengineering.com/info/accelerometers, Accessed on 22.06.2019.

[28] Matt. Introducing pixel the new raspbian desktop. Online https://www.raspberrypi-spy.co.uk/2016/09/introducing-pixel-the-new-raspbian-desktop/, Accessed on 22.06.2019.

[29] Tomasz Mazuryk and Michael Gervautz. Virtual reality - history, applications, technology and future. 12 1999.

[30] MEMS and Nanotechnology Exchange. What is mems technology? Online: https://www.mems-exchange.org/MEMS/what-is.html, Accessed on 22.06.2019.

[31] B. Mortazavi, K. C. Chu, J. Tai X. Li, S. Kotekar, and M. Sarrafzadeh. Nearrealistic motion video games with enforced activity. *Ninth International Conference on Wearable and Implantable Body Sensor Networks*, 2012.

[32] raspberrytips. The awesome story of raspberry pi. Online: https://raspberrytips.com/raspberry-pi-history/, Accessed on: 22.06.2019.

[33] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. Virtual network computing. *IEEE*, 1998.

[34] Robert Lee Seligmann. Creating a mobile vr interactive tour guide. *Haaga-Helia ammattikorkeakoulu*, 2018.

[35] Berkeley Sensor and Actuator Center. Online: http://bsac.eecs.berkeley.edu, Accessed on 22.06.2019.

[36] Antonio Serrano Sillero. How is electromagnetic motion tracking used in vr/ar? Online: https://3dcoil.grupopremo.com/blog/electromagnetic-motion-tracking-virtual-reality/, Accessed on 22.06.2019.

[37] Isaac Skog and Peter Handel. Calibration of a mems inertial measurement unit. *XVII IMEKO WORLD CONGRESS*, 2006.

[38] SparkFun. Sparkfun triple axis accelerometer breakout - adxl345. Online https://www.sparkfun.com/products/9836, Accessed on 22.06.2019.

[39] JONATHAN STRICKLAND. How virtual reality gear works. Online: https://electronics.howstuffworks.com/gadgets/other-gadgets/VR-gear6.htm, Accessed on 22.06.2019.

[40] terraelectronica. Raspberry pi 3 model b. Online: https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FT%252FTechicRP3.pdf, Accessed on 22.06.2019.

[41] Unity. The world's leading real-time creation platform. Online Online:https://unity3d.com/unity/, Accessed on: 22.06.2019.

[42] George Vele. Hand motion tracking for mobile vr games. *Sesiunea de Comunicari Stiintifice ale Studentilor, Cluj-Napoca (Romania)*, June 16, 2019.

[43] Lucía Vera. Tracking systems in virtual reality: which is the best choice? Online: https://3dcoil.grupopremo.com/blog/tracking-systems-virtual-reality-the-best-choice/, Accessed on 22.06.2019.

[44] Adam Wojciechowski and Piotr Napieralski. *Computer Game Innovations*. 12 2016.

[45] C.-H. Wu, Y.-T. Chang, and Y.-C. Tseng. Multi-screen cyber-physical video game: An integration with body-area inertial sensor networks. *International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2010.

[46] C. Yang, C. Yang J. Hu, C. Wu, and N. Chu. Dancing game by digital textile sensor, accelerometer and gyroscope. *International Games Innovation Conference (IGIC)*, 2011.

[47] K. Zintus-art, V. Pongparnich S. Saetia, and S. Thiemjarus. Dogsperate escape: A demonstration of real-time bsn-based game control with e-ar sensor. *Knowledge, Information, and Creativity Support Systems*, 2011.