Hand Motion Tracking for Mobile VR Games

George Vele
Department of Computer Science
Babes-Bolyai University
Cluj-Napoca, Romania
vrie2243@scs.ubbcluj.ro

***Abstract*—This work examines the development of a motion tracking device based solely on inertial sensors. The system is comprised of a total of 3 sensors and uses quaternion output to map the player's arm motions to a virtual arm on the screen. 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 we've faced while implementing such a system, and details on the methods in which we have solved them.**

***Index Terms*—motion-tracking, virtual-reality, inertial measurement units**

I. Introduction

When talking about Virtual Reality (VR), we refer to the interactive simulated environment that we can experience with the help of head-mounted displays, in combination with other motion tracking devices and input peripherals.

Along with the advances made in the computation capabilities of modern computers, both in terms of hardware and software, the quality and immersion experienced in VR have increased. Consequently, VR has gained popularity and market value, pushing the research made in this field further and further.

The aim of this paper is to show how arm motion tracking can be achieved using inertial measurement units, and how this tracking can be integrated into a mobile VR ping-pong game as a primary input method for the game. We present methods through which the tracker can be created at a physical level and, at the same time, how to model and synchronize the virtual representation of the arm with the player movements.

This paper is organized according to the following structure:

Section II presents the problem statement in a detailed manner, as well as the motivation behind this paper's existence.

Section III gives an overview of some of the methods already implemented in this subject.

Section IV dives into the methodology of solving the problem. It presents a short technical background defining the notions required to understand the solution. Here we find detailed explanations of the proposed approach, it's implementation, challenges, and technologies used in order to achieve results.

Section V addresses some observations we had during the development of this system, and the results we have achieved.

Section VI presents the improvements that can be made in a future release as well as the conclusions of this paper.

II. Problem Statement

Developing a technology that can seamlessly act as an interface between the human senses and computers has always been a challenge for virtual reality. In this section, we present the concept of motion tracking and a number of techniques that are used to implement it.

Motion Tracking in VR refers to the technologies and methods used in order to provide this interface. It can be defined as the process of aiming to capture, follow and get information about the object's orientation and position, with the purpose of being used by the application for further processing [1]. Immersion into VR refers to the degree in which we can trick our senses and perceive the presented non-physical world as being real.

Since the human brain can perceive large amounts of information at incredibly high speeds, in order to be considered immersive, among many things, the responsiveness and accuracy of motion tracking must also be extremely high. There are numerous methods by which one can implement motion tracking. These are divided into two categories: Optical and Non-Optical Systems.

Optical System tracking is based on the information provided by a collection of cameras, connected between them and to a computer, in order to detect the motion of an element within a limited area [1]. These cameras need calibration to obtain correct information on their relative position and also to be able to correctly detect the object's movement [13].

Non-Optical Systems rely on alternative methods and components that enable the recognition, following and obtaining the position of the tracked object [1] [13]. One such system is the inertial one.

Inertial tracking consists in using very small sensors, based on micro-electromechanical systems (MEMS) technology [12]. As such, accelerometers, gyroscopes and magnetometers are combined into Inertial Measurement Units (IMU), in order to accurately detect the object's position and rotation. An advantage of this method consists in the high frequency at

which information is received. On the other hand, it is susceptible to Dead Reckoning, which leads to errors accumulating in the computation, also known as drift. To compensate for this drift, there can be used a number of estimation techniques, such as Kalman and Complementary Filtering [14] [15].

The motivation behind creating such a system is to observe the practicality of such a device. We are curious about the ease of use, the precision with which our movements are tracked, the latency obtained and the overall immersion given to the player.

III. Related Work

At the moment of writing this paper, the literature on exclusive inertial measurements for VR entertainment systems is scarce. Although in 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 handheld, they will only detect the hand motion, not the motion of the entire limb.

In [6] they created their own version of a dance game using four worn Wii remotes on the ankles and wrists. Their way of gesture recognition for the game being preset poses to be checked on when the beat reaches a certain point. Other dancing games have been created using inertial systems, such as [7] and [8]

Paper [5] developed a body tracking system using 10 WIMU sensors attached to major body joints (upper arm, forearm, thigh, shin, trunk, and pelvis) on the right and left sides. This system tracks these joints to move an avatar in real time. To test this system, they created a VR game where the player's goal is to shoot a target by stretching and pointing the right arm toward it. When the player hits the target, it explodes and the game creates another target in a random location. After the players hit three targets, they begin to move continuously, increasing the game challenge. In comparison with our work, they were able to track more body limbs than our system, having used 10 sensors instead of 3.

Paper [9] created a virtual game of Quidditch, a sport from the Harry Potter universe. They used two IMU sensors on the player's arm, as well as one on both their prop broom and club. The player steered by moving their held broomstick and used their arm and club to attack incoming balls. To increase immersion the setup used multiple screens to project the view on multiple sides of the player. However, they do not give any specific details on the sensor based interactions or gameplay.

Both [10] and [11] created games that use a single sensor attached to the player's ear or foot respectively. In [10] they used an accelerometer to determine if the player was running, walking, jumping, or leaning. These actions controlled an on-screen dog character and the player's goal was to avoid oncoming obstacles. Though a very simple setup, they achieved very high recognition accuracy results at almost 99%, with recognition times under a second. Meanwhile, [11] used an accelerometer and pressure sensor combination to control actions in a soccer game. Though the player still made use of a standard controller for some inputs, the worn sensor was involved in the running, passing and shooting actions. They also put significant consideration into protecting the system from being cheated in the same way as 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, [3] presented a VR soccer game for physical rehabilitation, which uses an Intel RealSense sensor for motion capture. In this game, players take the role of a goalkeeper and use hand gestures to defend balls. There are two ways to conduct rehab sessions, which can be supervised or automatic. In the supervised version, a physiotherapist decides when the ball is thrown and its height in relation to the ground. A faster frequency of launches will require faster movements to be performed by the patient, and a greater difference in height between consecutive launches will require a wider angle in the player movements. On the other hand, the automatic session starts with a longer interval between the launches with a slight difference between the angles and increases or decreases the difficulty level based on the score. In this way, the game analyzes the user's posture and infers the position of the upper trunk joints(shoulder, elbow, and hand). Therefore, the score is increased only if the movement is performed correctly, which requires that the trunk is straight and perpendicular to the ground, and the arm fully extended. The Intel RealSense depth camera should be fixed in front of the user so that the field of view can cover your entire body. Thus, we observed that although it allows capturing the joints of the upper limbs (shoulder, elbow, and hand), it has the common limitation so fusing a fixed depth camera in one position.

IV. Methodology

This section presents our approach for creating the motion tracker: the types of sensors used, the way we connected them to a Raspberry PI, how we communicated the information to our VR game and how we've treated collisions as to have your hand interact with the virtual environment.

A. Technical Background

1) Accelerometers: An accelerometer is a device that measures proper acceleration [17]. Proper acceleration, being the acceleration (or rate of change of velocity) of a body in its own instantaneous rest frame, is not the same as coordinate acceleration, being the acceleration in a fixed coordinate system. For example, an accelerometer at rest on the surface of the Earth will measure an acceleration due to Earth's gravity, straight upwards (by definition) of $g = 9.81m/s$ [16]. By contrast, accelerometers in free fall (falling toward the center of the Earth at a rate of about $9.81m/s^2$) will measure zero.

Accelerometers have multiple applications in industry and science. Highly sensitive accelerometers are components of inertial navigation systems for aircraft and missiles. They are used to detect and monitor vibration in rotating machinery, are used in tablet computers and digital cameras so that images on screens are always displayed upright and even used in drones for flight stabilization. Coordinated accelerometers can be used to measure differences in proper acceleration, particularly gravity, over their separation in space; i.e. the gradient of the gravitational field. This gravity gradiometry is useful because absolute gravity is a weak effect and depends on the local density of the Earth which is quite variable.

2) Gyroscopes: Gyroscopes, or gyros, are devices that measure or maintain rotational motion. MEMS gyros are small, inexpensive sensors that measure angular velocity. The units of angular velocity are measured in degrees per second or revolutions per second (RPS). Angular velocity is simply a measurement of the speed of rotation.

Gyros can be used to determine orientation and are found in most autonomous navigation systems. For example, if you want to balance a robot, a gyroscope can be used to measure rotation from the balanced position and send corrections to a motor.

3) Magnetometers: The magnetometer sensor utilizes the modern solid-state technology to create a miniature Hall-effect sensor that detects the Earth's magnetic field along three perpendicular axes X, Y, and Z.

The Hall-effect sensor produces a voltage which is proportional to the strength and polarity of the magnetic field along the axis each sensor is directed. The sensed voltage is converted to a digital signal representing the magnetic field intensity. Other technologies used for magnetometer may include magnetoresistive devices which change the measured resistance based on changes in the magnetic field.

The magnetometer is enclosed in a small electronic chip that often incorporates another sensor (typically a built-in accelerometer) that help to correct the raw magnetic measurements using tilt information from the auxiliary sensor. In addition to general rotational information, the magnetometer is crucial for detecting the relative orientation of your device relative to the Earth's magnetic north.

4) Inertial Measurement Units: An IMU is a specific type of sensor that measures angular rate, force and sometimes magnetic field. IMUs are composed of a 3-axis accelerometer and a 3-axis gyroscope, which would be considered a 6-axis IMU. They can also include an additional 3-axis magnetometer, which would be considered a 9-axis IMU. Technically, the term "IMU" refers to just the sensor, but IMUs are often paired with sensor fusion software which combines data from multiple sensors to provide measures of orientation and heading. In common usage, the term "IMU" may be used to refer to the combination of the sensor and sensor fusion software; this combination is also referred to as an AHRS (Attitude Heading Reference System).

Common applications for IMUs include determining the direction in a GPS system, tracking motion in consumer electronics such as cell phones and video game remotes, or following a user's head movements in AR (augmented reality) and VR (virtual reality) systems. This motion and orientation information also apply to maintain a drone's balance, improving the heading of your robot vacuum cleaner, and other IoT and connected home devices.

5) Sensor Fusion: Sensor Fusion is the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually [18]. There are multiple implementations of a sensor fusion algorithm, ranging from simple but acceptable, to complex but very good. Two of these implementations, finding themselves at the extremes of this spectrum are the Complementary Filter and the Kalman Filter.

6) Complementary Filter: The complementary filter uses the data from the gyroscope, because it is very precise and not susceptible to external forces [14]. On the long term, we use the data from the accelerometer, as it does not drift. In its 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. The gyroscope data is integrated (added) every timestep with the current angle value. After this, it is combined with the 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

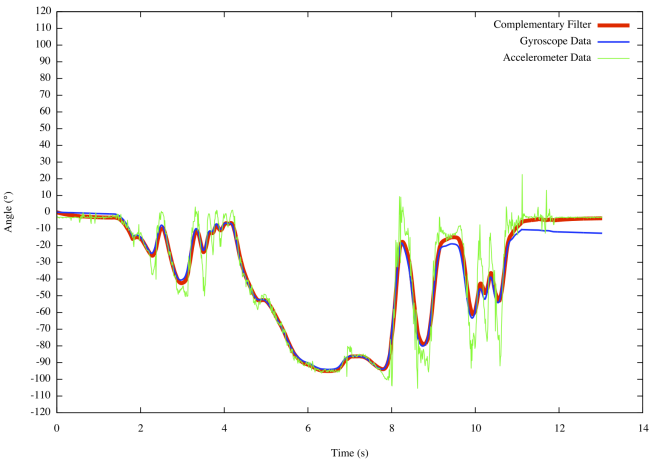

Fig. 1: Complementary filter using Gyroscope and Accelerometer

Every iteration the pitch and roll angle values are updated 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. Afterward, it will update the pitch and roll angles with the accelerometer data 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.

7) Kalman Filter: In statistics and control theory, Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe [15]. The filter is named after Rudolf E. Kálmán, one of the primary developers of its theory.

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization. The Kalman filter also works for modeling the central nervous system's control of movement. Due to the time delay between issuing motor commands and receiving sensory feedback, use of the Kalman filter supports a realistic model for making estimates of the current state of the motor system and issuing updated commands.

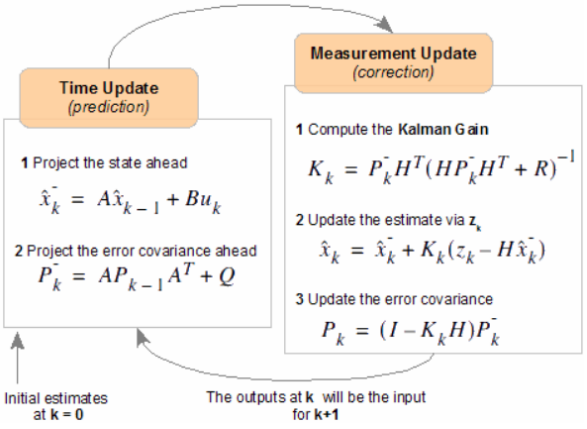

Fig. 2: Kalman filtering alrogithm [15]

The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required. 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 [15].

B. Proposed Approach

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 worn 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.

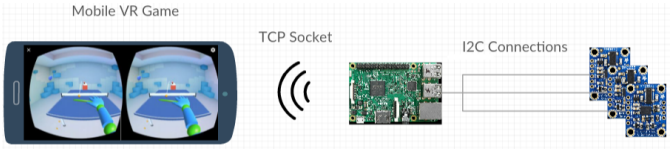

Fig. 3: Architecture of the system

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 viewed in a virtual reality headset. As our microcontroller, we've chosen the Raspberry PI 3 model B. Sensory data are received and transmitted through scripts written in Python. About the sensors, we will talk in the next section.

2) Sensors: 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 microprocessor loaded with a sensor fusion algorithm that computes very useful information out of the box.

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/s^2
- 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^2
- Gravity Vector (100Hz)
  Three axis of gravitational acceleration (minus any movement) in m/s^2
- Temperature (1Hz)
  Ambient temperature in degrees Celsius

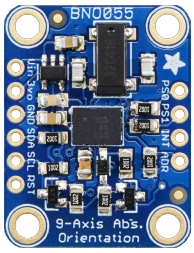

Fig. 4: BNO055

3) Connecting sensors to Raspberry PI: The I2C protocol was used in order to achieve this connection.

The I2C is a well known and widely used protocol. It supports multiple slaves systems and the communication is done through only 2 lines, one for data transferring (SDA) and one for keeping the clock (CLK). The data is sent in messages that contain the address of the device to which it is addressed, as well as multiple acknowledgments and start/stop conditions. Because of this, the devices can confirm that each information frame has been transferred successfully. At the same time, this confirmation takes time. This and the addressing systems make I2C to be significantly slower than another protocol, the Serial Peripheral Interface (SPI).

We've chosen to use I2C, because we've found that the simplicity of the wiring is something that we look for in the Motion Tracker, and although it is slower than SPI, it is by no means slow. Because the sensors used are identical, their internal address is also the same. The hardware specification tells us that their addresses can be changed to another one by powering one of the pins, but that only gives you 2 addresses to work with. We've discovered that you can open multiple I2C buses on the Raspberry, so a workaround is to have 2 buses and to physically change the address of one of the sensors so we can have it at the same bu with another one with default address.

4) Communicating measurements to game: Communication is done via a TCP Socket, where the game acts as a server and actively listens for connections. The Raspberry acts as a client and will create a connection to the game, subsequently sending packets with measurements from the sensors.

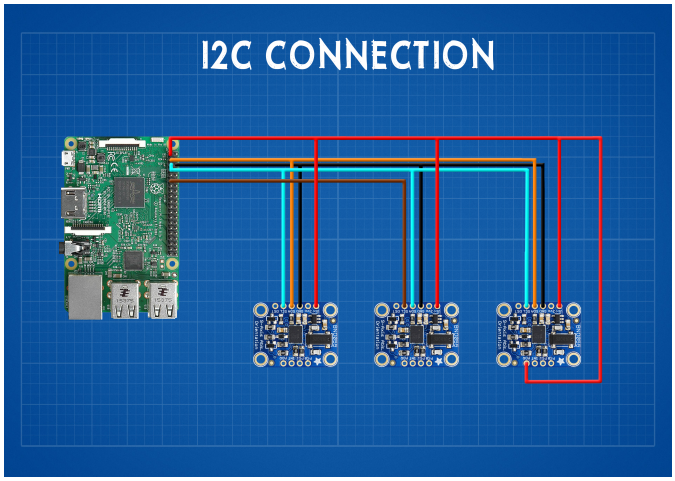

Fig. 5: Raspberry Pi connected to 3 x BNO055 using I2C

5) Creating the game: In order to create the game in Unity, the first step was to create a scene. This scene was then populated with a room that contains the ping-pong table, a cannon that shoots balls in different directions and, most importantly, the player character.

In order to make it a mobile VR game, we had to specify to Unity that this game is to be built for Android and import the Google VR library into our project. This library contains the necessary scripts and cameras that make the player's view rotate along the axes the phone rotates. We attached such a camera to our character.

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 its default position. As long as the player does not move his/her fingers a lot, the immersion should still hold.

Our approach was to use forward kinematics in order to achieve this. Forward kinematics consists of 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 are 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.

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.

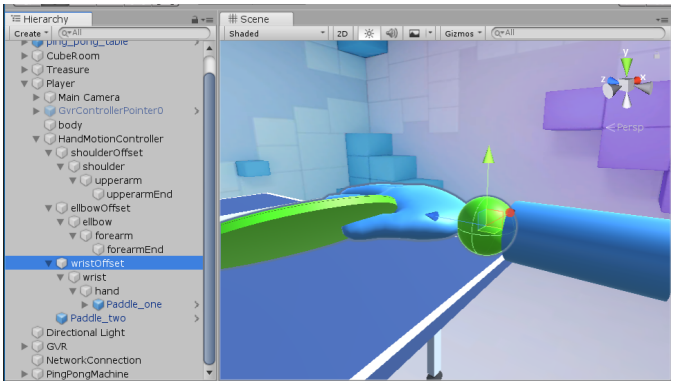

Fig. 6: On the left you can see the hierarchical structure of the hand, while on the right side we have the offset object selected

V. Results and Discussions

When we started the game, we expected that 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 of 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 your 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 is that the latter will execute your code once per frame, being totally dependent on the frame-rate of your 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 the greater the chance of the ball missing the paddle completely. Normally, within the Update() method, you 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 the amount you want 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 'rigid body' component and a script that uses FixedUpdate(), in which it tells the cylinder 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, If we chose, 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.

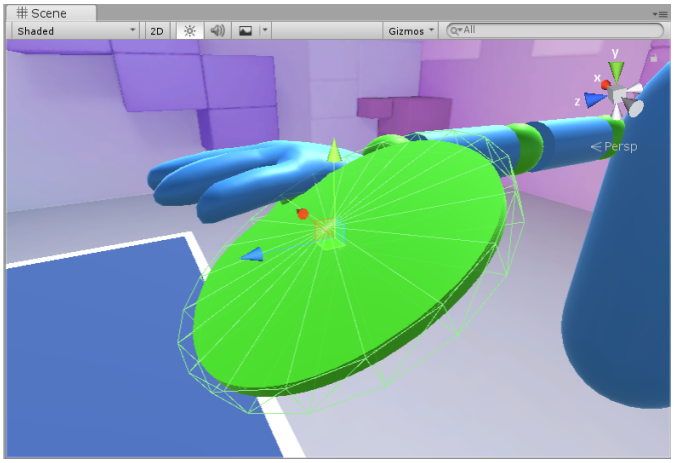

Fig. 7: Here we can see the cylinder's wire-frame, this is what balls collide into when hitting them

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 you can compute the measurements. In the inertial approach, this relates to the speed of the sensor fusion and sensor

	Min	Avg	Max
FPS	9	21	50

TABLE I: Arm frame-rates

transmission. We have measured this speed in the number 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.

Fig. 8: The game FPS and the hand FPS displayed in the upper left corner of the room

VI. Conclusions and future work

In this article, we have shown how arm motion tracking can be achieved by creating a system composed of 3 IMU sensors strapped to the player's arm, a Raspberry PI and how it can be integrated with a 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.

Although the device functions well, there exists a lot of room for 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

References

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

[2] R. Atienza, R. Blonna, M. I. Saludares, J. Casimiro, and V. Fuentes. Interaction techniques using head gaze for virtual reality. In Region 10 Symposium (TENSYMP), 2016 IEEE, pages110–114.IEEE, 2016.

[3] 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, 64:10–16, 2015.

[4] D. Holmes, D. Charles, P. Morrow, S. McClean, and S. McDonough. Usability and performance of leap motion and oculus rift for upper arm virtual reality stroke rehabilitation. In Proceedings of the 11th International Conference on Disability, Virtual Reality & Associated Technologies. Central Archive at the University of Reading, 2016.

[5] D. L. Arsenault and A. D. Whitehead. Quaternion based gesture recognition using worn inertial sensors in a motion tracking system. InGamesMediaEntertainment(GEM), 2014IEEE, pages1–7.IEEE, 2014.

[6] E. Charbonneau, A. Miller, C. A. Wingrave, and J. J. LaViola Jr, "Poster: RealDance: An Exploration of 3D Spatial Interfaces for Dancing, " in IEEE Symposium on 3D User Interfaces, 2009, pp. 141–142.

[7] A. Kailas, "Basic human motion tracking using a pair of gyro + accelerometer MEMS devices, " in IEEE 14th International Conference on e-Health Networking, Applications and Services, 2012, no. 1, pp. 298–302.

[8] C. Yang, J. Hu, C. Yang, C. Wu, and N. Chu, "Dancing game by digital textile sensor, accelerometer and gyroscope, " in 2011 IEEE International Games Innovation Conference (IGIC), 2011, pp. 121–123.

[9] C.-H. Wu, Y.-T. Chang, and Y.-C. Tseng, "Multi-screen cyber-physical video game: An integration with body-area inertial sensor networks, " in 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010, pp. 832–834

[10] K. Zintus-art, S. Saetia, V. Pongparnich, and S. Thiemjarus, "Dogsperate Escape: A demonstration of real-time BSN-based game control with e-AR sensor, " in Knowledge, Information, and Creativity Support Systems, vol. 6746, Berlin Heidelberg: Springer, 2011, pp. 253–262.

[11] B. Mortazavi, K. C. Chu, X. Li, J. Tai, S. Kotekar, and M. Sarrafzadeh, "Nearrealistic motion video games with enforced activity, " in 2012 Ninth International Conference on Wearable and Implantable Body Sensor Networks, 2012, pp. 28–33.

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

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

[14] 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.

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

[16] Matej Andrejasic. Mems accelerometers. Marec, 2008.

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

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