Informatyka techniczna i sztuczna inteligencja
Technical informatics and Artificial Intelligence
Engineer of XXI Century'2024

Legs tracking for VR environment using sensors

Szymon Białek [1], Marcin Bernaś [2*]

[1] *Faculty of Mechanical Engineering and Computer Science, University of Bielsko-Biala, Willowa 2, 43-300 Bielsko-Biala, Poland, iamszymonbialek@gmail.com*
[2] *Faculty of Mechanical Engineering and Computer Science, University of Bielsko-Biala, Willowa 2, 43-300 Bielsko-Biala, Poland, mbernas@ubb.edu.pl*
* *Corresponding author, mbernas@ubb.edu.pl*

Abstract: The article presents an original system for collecting data from the human lower limbs using sensors placed in smartphones, and then transmitting and using them to support the tracking of the lower limbs in a virtual reality environment. The system includes two applications. The first one, designed for mobile devices, allows you to collect data from many sensors attached to the user in the form of quaternions, and then send it via UDP packets to the appropriate application. This application simulates a virtual environment and maps the position and movement of a particular limb. The correctness of the assumptions and the developed solution was confirmed in a virtual environment through numerous tests on a group of users. The advantages and disadvantages of the solution are indicated, as well as the possible directions of development of this technology.

Keywords: VR environment; UDP transmission; movement tracking; sensors;

Śledzenie kończyn dolnych w środowisku VR za pomocą sensorów

Szymon Białek [1], Marcin Bernaś [2*]

[1] *Uniwersytet Bielsko-Bialski, Wydział Budowy Maszyn i Informatyki, Willowa 2, 43-300 Bielsko-Biala, Polska, iamszymonbialek@gmail.com*
[2] *Uniwersytet Bielsko-Bialski, Wydział Budowy Maszyn i Informatyki, Willowa 2, 43-300 Bielsko-Biala, Polska, mbernas@ubb.edu.pl*
* *Corresponding author, mbernas@ubb.edu.pl*

Streszczenie: W artykule przedstawiono autorski system zbierania danych z kończyn dolnych człowieka za pomocą sensorów umieszczonych w smartfonach, a następnie przesyłania i wykorzystywania ich do wspomagania śledzenia kończyn dolnych w środowisku wirtualnej rzeczywistości. W skład systemu wchodzą dwie aplikacje. Pierwsza, przeznaczona na urządzenia mobilne, pozwala na zbieranie danych z wielu sensorów dołączonych do użytkownika w postaci kwaternionów, a następnie przesyłanie ich za pośrednictwem pakietów UDP do odpowiedniej aplikacji. Ta aplikacja symuluje środowisko wirtualne i mapuje pozycję i ruch określonej kończyny. Poprawność założeń i wypracowanego rozwiązania została potwierdzona w testach w środowisku wirtualnym na grupie użytkowników. Wskazano zalety i wady rozwiązania, a także możliwe kierunki rozwoju tej technologii.

Słowa kluczowe: środowisko VR; transmisja UDP; momement tracking; czujniki;

1. Introduction

Virtual reality (VR) is an advanced technology that allows users to interact with a computer-generated, three-dimensional environment in a way that mimics reality [1]. It allows you to be transported to virtual worlds that can reflect both real places and completely fictional scenarios. VR is widely used in many fields, each of which benefits from its unique technologies [2]. In the entertainment and gaming industry, VR enables the creation of highly immersive and

DOI: https://doi.org/10.53052/9788367652292.17

interactive experiences that engage the user on an unprecedented scale [3]. Players can move to fantastical worlds, take part in dynamic actions and adventures, all with a sense of reality and presence in a virtual environment. In education, VR offers new opportunities for experiential learning. Students can virtually visit historical sites, explore remote corners of the world, or conduct complex laboratory experiments without having to leave the classroom [4]. Virtual simulations and models allow for a better understanding of complex concepts and theories through the practical application of knowledge. A key element that determines the quality and effectiveness of the VR experience is the precise representation of the user's movements. It is the accurate tracking and mapping of movements that allows for full immersion in the virtual world and natural interactions with the environment. The realism of these interactions directly affects the degree of immersion, i.e. the feeling of being present in a virtual environment. In the case of lower limb movement, precise reproduction of leg movements is particularly important for applications such as VR games, sports training, or motor rehabilitation.

Mapping lower limb motor skills in virtual reality (VR) systems is a key element in ensuring a realistic and immersive user experience. There are many different approaches to tracking body movements, which vary in terms of technology, accuracy, and cost. One popular solution is the Driver4VR app, which offers free Full Body Tracking using camera readings [5]. This application allows you to map the user's body movements without the need to use expensive sensors or specialized equipment. The disadvantages of the solution are the dependence on the camera - the effectiveness of motion tracking depends on the quality and position of the camera, which can be problematic in low light conditions. Additionally, camera readings can be less accurate and have higher latency compared to specialized motion sensors. The design solution, using smartphones, offers a more reliable and precise alternative to lower limb tracking Vive Trackers Vive Trackers are advanced tracking devices developed by HTC that enable precise mapping of the user's body movements in a VR environment. These trackers are compatible with the HTC Vive system and offer a wide range of applications, from VR games to professional training and simulation applications [6]. Thanks to advanced infrared tracking technology, Vive Trackers offer very high precision tracking for realistic reproduction in VR. The disadvantage of the solution is the price and dependence on the ecosystem. Vive Trackers are relatively expensive, which can be a barrier for individual users. Another solution is the Full Body Tracking SDK developed by Oculus is an advanced solution for tracking the user's body movements using cameras built into Oculus devices. SDK to umożliwia integrację funkcji śledzenia ciała z aplikacjami VR, zapewniając realistyczne odwzorowanie ruchów górnych partii ciała w wirtualnym środowisku [7]. The SDK offers precise tracking of upper body movements, which can significantly improve the realism of VR experiences. The limitation of the solution is a rough estimation of the position of the upper limbs. Another solution is SlimeVR. SlimeVR is a project that offers solutions for tracking body movements in VR, mainly based on IMU (Inertial Measurement Unit) sensors. The project is known for providing detailed instructions on how to create trackers yourself, and also runs a shop with the components needed to build them [8]. IMU trackers can be integrated into multiple VR platforms, making SlimeVR a universal solution. The disadvantage of the solution is the complexity of the construction. For some users, building and configuring trackers on their own can be challenging, requiring technical knowledge and manual skills. A smartphone-based solution with built-in motion sensors, it offers a more versatile and easier alternative to precise tracking of lower limb movements in VR. In conclusion, there are already solutions on the market that offer the possibility of tracking with sensors or cameras, but they have a number of indicated drawbacks, such as cost, the requirement of electronic knowledge or the environmental limitations of their use [16, 17].

The paper proposes a proprietary solution based on sensors in phones to support VR systems in accurate tracking of the lower limbs. The chapters describe the proposed project and the main elements of the implementation. The scope of work also includes testing and validating the system to assess its accuracy and usability.

2. Proposed Method of lower body tracking

The application design presented in Figure 1 allows you to map the movement of the user's lower limbs in a VR environment using sensors mounted on smartphones. The choice of smartphones as a platform for motion sensors is due to their widespread availability and advanced measurement technologies. Smartphones are equipped with various types of sensors, such as accelerometers, gyroscopes, and magnetometers, which can precisely monitor the movements and orientation of the device. The work involves using these sensors to collect data on the user's leg movements and send them to the VR system using the User Datagram Protocol (UDP) [9].

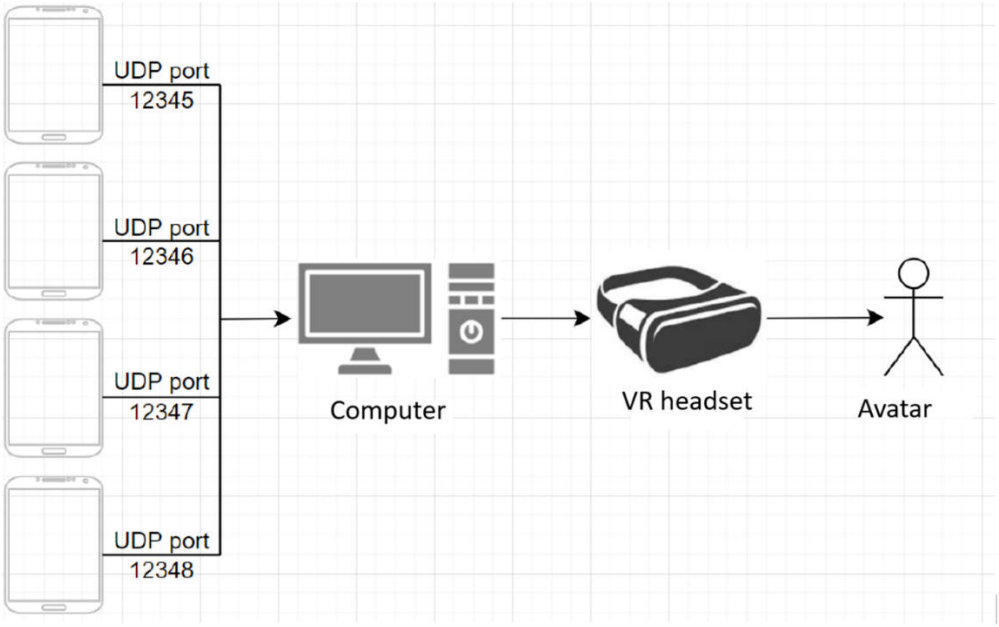

Figure 1. Overall system scheme

The system consists of two apps: the first, running on smartphones, which collects data from the sensors and sends it to the second app on the computer. A second app receives this data and uses it to animate the movement of the avatar's legs in a virtual environment. This arrangement allows for the simulation of natural movements of the lower limbs, which increases the realism and immersion of the VR experience. The implementation of this project aims not only to investigate the technical aspects of using motion sensors built into smartphones to track lower limb movements, but also to assess the practical possibilities and limitations of this solution in the context of VR applications.

The solution is based on a loop of operations carried out in the following steps:

```
As long as the app is enabled:
1. Data Collection: Smartphones collect data from accelerometer, gyroscope, and Game Rotation
Vector sensors.
2. Data transfer: Data is transmitted via UDP to the appropriate ports of the computer.
3. Data reception: The computer receives data from different ports, deserializes and processes
it.
4. Display the application in VR: The computer displays the project in the VR space.
5. Avatar Update: The processed data is used to update the position and rotation of the relevant
avatar body parts in the VR environment.
```

The design application allows for a realistic representation of the user's lower limb movements in a VR environment thanks to the precise collection, transmission and processing of data from sensors mounted on smartphones. The system is scalable and can operate with a different number of sensors, allowing flexible adaptation to the user's needs. In order to implement individual elements of the application, an analysis of the available technologies was performed.

2.1. Motion sensors and their applications

Motion sensors in VR and AR technologies are divided into several main categories, which are necessary for precise tracking and user interaction with the virtual environment [10], [11], [12]. For the purposes of the study, smartphone sensors were analyzed, among others: accelerometers (allow to measure linear acceleration in different axes), gyroscopes (used to measure and maintain the orientation of the device by detecting rotation around the axis), magnetometers (to determine the orientation of the device in relation to the Earth's magnetic field), as well as cameras and optical sensors. All of these are used in advanced VR systems to precisely track the user's movements and interactions with the environment.

Motion sensors are widely used in mobile devices, where they play a key role in a variety of applications, from simple utility functions to advanced augmented reality (AR) and virtual reality (VR) interactions. Smartphones are

typically equipped with a combination of accelerometers, gyroscopes, and magnetometers, which allows you to accurately track the movements and orientation of the device.
Accelerometers in smartphones measure linear acceleration, which allows them to detect movements such as shaking or changing the position of the device. Gyroscopes provide rotation information, which is essential for tracking more complex movements. Magnetometers allow orientation relative to the Earth's magnetic field, which is useful for navigation applications. The integration of these sensors enables the creation of AR and VR applications that respond to the user's movements in real time, offering more immersive and interactive experiences.

Sensor Game Rotation Vector
One of the advanced sensors used in smartphones is the **Game Rotation Vector sensor** described in the Android documentation [13]. This is a sensor that combines data from an accelerometer and gyroscope to determine the orientation of a device in space. It works on the principle of rotation analysis in three axes, providing values corresponding to the rotation around the X, Y, and Z axes.
The Game Rotation Vector works as follows:
1. **Data Collection**: The accelerometer provides linear acceleration information, and the gyroscope measures the angular velocity of the device.
2. **Sensor fusion**: Data from the accelerometer and gyroscope are combined using a sensor fusion algorithm. The accelerometer is good at detecting static orientation, but prone to interference for dynamic movements. A gyroscope is very precise in a short period of time, but its accuracy can decrease due to drift. The sensor fusion algorithm takes advantage of the advantages of both sensors to provide accurate and stable orientation results.
3. **Use of quaternions**: The result of fusion is a set of quaternions that describe the orientation of the device in three-dimensional space. Quaternions allow you to avoid gimbal lock problems by providing smooth and stable rotation tracking.
4. **Data output**: The sensor returns values that are directly used to map orientation in VR applications.

The project will use Game Rotation Vector sensors installed in smartphones, they will be used to monitor the movements of the user's lower limbs. The leg rotation data collected by these sensors will be transmitted to the VR system using the UDP protocol.

2.2 UDP protocol and its use
In the context of motion tracking in virtual reality (VR) and augmented reality (AR) systems, effective data transmission is crucial. The User Datagram Protocol (UDP) is widely used due to its speed and low latency. Unlike the more complicated and reliable Transmission Control Protocol (TCP), UDP allows data to be transmitted without the need to establish a connection [9], which speeds up communication. This is especially important in real-time applications, where data transfer speed is prioritized over reliability. User Datagram Protocol (UDP) is a transport layer communication protocol that is part of the Internet Protocol (IP) suite. UDP is a simple, lightweight protocol that offers no error correction mechanisms or acknowledgments, which means that applications that use UDP must be able to tolerate packet loss, errors, and duplication. UDP is often used in applications that require low latency, such as video streaming, VoIP (Voice over IP), and online gaming [14].

These properties make UDP an ideal choice for VR applications, where low latency is crucial for realistic representation of user movements.

2.3 Quaternions and their use in computer graphics
Quaternions are an advanced mathematical representation of rotation in three-dimensional space that offers many advantages over traditional methods such as Euler angles. Quaternions, introduced by William Hamilton in 1843, are four-dimensional numbers consisting of one real and three imaginary components. They enable the representation of rotation without the risk of problems with the so-called gimbal lock and ensure smooth and stable tracking of object orientation [15].

Quaternions can be represented in the form of:

$$q = ae + bi + cj + dk, \tag{1}$$

where a,b,c,d are real numbers, and (i, j, k) are imaginary units. Rotations in three-dimensional space can be described by unit quaternions, which have a norm of 1. Based on the description in [15], the rotation is represented by a quaternion of the form:

$$q = \cos\left(\frac{\theta}{2}\right) + sin\left(\frac{\theta}{2}\right)(\mathrm{xi} + \mathrm{yj} + \mathrm{zk}), \quad (2)$$

where θ is the angle of rotation and (x,y,z) is the unit vector of the axis of rotation. The division by 2 results from the fact that quaternions describe half of the rotation angle, which is important for rotation operations.

Quaternions vs. Euler angles and gimbal lock

Euler angles describe the rotation around three axes (X, Y, Z), however, they can lead to a situation in which two of the rotation axes become collinear, resulting in the loss of one degree of freedom (such a case called gimbal lock). This problem can be avoided by using quaternions, which describe rotation in three-dimensional space using four parameters: q0 (scalar) and q1, q2, q3 (vector).

In VR and animation systems, quaternions are used to represent the orientation of objects and avatars in three-dimensional space. They are especially useful in motion tracking applications, where precision and stability are key. Quaternions allow you to reproduce complex movements without sacrificing accuracy, which is important for a realistic user experience.

3. Implementation of tracking system

The implementation of the project involves the creation of two applications in the Unity environment: one for smartphones, which collects data from motion sensors, and the other for a computer, which maps the user's movements on an avatar in virtual reality (VR). These applications communicate with each other using UDP to ensure fast and reliable data transfer.

3.1 Phone app

The mobile application will be responsible for collecting data from motion sensors installed in smartphones. These sensors include an accelerometer, gyroscope, and Game Rotation Vector sensor, which allow you to precisely track the orientation and movement of the device in space.

The mobile application allows you to collect data from sensors and send them. The app collects data from the accelerometer, gyroscope and Game Rotation Vector sensor in real time. The sensor data is then sent via UDP to a computer application, using the appropriate ports for each limb according to Figure 2 (port: 12345 - left thigh, 12346 - left ankle, 12347 - right thigh, 12348 - right ankle). It is worth noting that the desktop application will also be adapted to the use of less than four phones, e.g. the phone marked in figures 1 and 2 will be able to simulate a leg on their own without the need to connect the 3rd and 4th phones.

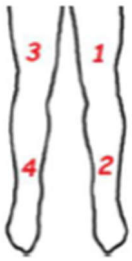

Figure 2. Assigning sensors to the leg position

3.1.1 Initialization and Sensor Reading

By using Unity [16] and its InputSystem, we can easily access position and rotational sensors in the script. The following code shows how to turn on the sensors and access their measurements. In the Start() method, we activate the sensors, and in the FixedUpdate() method, which is responsible for spontaneous refreshing in Unity, we retrieve the current data from the sensors.

```
private void Start()
```

```
{
Screen.sleepTimeout = SleepTimeout.NeverSleep;
InputSystem.EnableDevice(Gyroscope.current);
InputSystem.EnableDevice(Accelerometer.current);
InputSystem.EnableDevice(AttitudeSensor.current);
}
private void FixedUpdate()
{
InputSystem.Update();
angularVelocity = Gyroscope.current.angularVelocity.ReadValue();
acceleration = Accelerometer.current.acceleration.ReadValue();
attitudeSensor = AttitudeSensor.current.attitude.ReadValue();
```

Then the data from all three sensors is stored in the form of a single text string, where the first data is for the accelerometer, the next for the gyroscope, and the last data for the game rotation sensor. Each packet is timestamped to determine its timing.

3.1.2 Device-to-Device Data Transfer

To transfer sensor data from the mobile app to the desktop app, you need to create a UDP connection. This process includes setting up the connection, transferring data, and maintaining stable communication. A simple, intuitive interface (Figure 3) has been developed to control the application, which allows you to define the basic communication parameters.

Figure 3. Data collection application interface

All key methods are included in Annex 1 of the paper. The transmission process is started by the "Send Data" button. First, the port and connection to the server are opened. Once the connection is established, the application starts sending data at a specified frequency. This process is accomplished by the *StartSendingData method*, which creates a new thread responsible for sending data. Next, the *SendDataLoop method* is responsible for continuously sending data to the server at set intervals. The frequency of sending data is specified by the user in the user interface.

3.2. Desktop Aplication

The desktop app is responsible for receiving data from the phone app, deserializing it back to the Vector3, Vector3, Quaternion type, and then processing that data and assigning it to rotate the corresponding avatar body parts in the game. It can work with even one smartphone as a sensor, but the more smartphones are used (up to a maximum of four), the better and more accurate the tracking of the user's movements. The application was also written in Unity (Figure 4), which allows for easy integration with various VR components and provides high performance and flexibility.

Figure 4. An application that maps movement on a character to an avatar

The main tasks of the desktop application are: receiving data, processing it and then updating the avatar in the game. The most important implementations of the methods for work transparency are included in Annex 2.

3.2.1 Data Reception

Receiving data in a computer application is done by creating a server that listens on specific UDP ports (*UdpServer method*). The server receives the data sent by mobile applications and then distributes it to the appropriate scripts based on the port from which it was sent (*SendData method*). Each script responsible for moving limbs listens for a specific port and checks for new data received on that port. Data subscription is done by assigning the data handling method to the appropriate port - *the Subscribe method*. Through this process, the desktop app can receive data sent by a pre-written phone app.

3.2.2. Data deserialization

After obtaining data from mobile applications, this data is passed to *the DataConverter*, which converts the string into the appropriate types: Vector3 for accelerometer data, Vector3 for gyroscope data, Quaternion for Game Rotation Vector Sensor data, and DateTime for the timestamp when the data was sent. The deserialization process is done using regular expressions (regex) - Appendix 2. This allows the data to return to a form accepted by the Unity engine. It is worth noting that the data also has a timestamp, which contains the date and time on which it was sent. This can later be used to verify that all data has been received correctly.

3.2.3 Data processing and avatar update

In the data processing process, we focus on attitude data, which is the data from the Game Rotation Vector sensor. This data is in the form of Quaternion, which allows for precise mapping of the rotation of the device in three-dimensional space. By properly adjusting the rotation of the phone, we can obtain a rotation corresponding to the current rotation of the corresponding part of the leg in reality. The application performs the following operations:

- **Data deserialization**: The resulting data is processed by the DataConverter, which converts it into the appropriate types, including the Quaternion for attitude data.
- **Rotation adjustment**: The rightCoordToUnityCord method converts the orientation from the phone's coordinate system to the coordinate system used by Unity. This is necessary to ensure that the rotation is mapped correctly in the VR environment.
- **Initial Rotation Initialization**: The first time we receive data, we set the initial rotation of the phone and the player's character. This initialization allows you to specify the base position from which further rotations will be calculated.
- **Relative Rotation Calculation**: Relative Rotation is calculated as the product of the inverse of the phone's initial rotation and the current rotation. This operation allows you to get the relative change in rotation from the moment you start tracing.
- **Adjusting the player's character orientation**: Finally, the calculated relative rotation is applied to the player's character rotation in the game. As a result, the movements of the user's lower limbs are precisely reproduced in the VR environment.

By using a bool variable called first, we can re-set the initial rotation of the phone and the player character at any time, which is later used in the position reset (pre-calibration) button.

4. Experiments

Tests of correct operation were carried out both in terms of the number of collected data and the possibility and accuracy of motion mapping.

Transmission quality tests were used to check how many data packets would be lost when transmitting data from sensors to a PC application. The tests were carried out using two phones: HUAWEI Mate 20 Lite (device 1) and Huawei P20 Lite (device 2). A series of tests were carried out. The first test was carried out with data sent at 50 Hz and a permitted delay of 2 seconds. The results are shown in Figure 5 (interface of the application for testing).

Figure 5. Testing interface in application

As you can see, the number of lost packets does not exceed 0.1% for any of the phones. The tests were repeated multiple times for different frequencies (50 and 100 Hz) and acceptable delay (2 and 5 seconds). The results are presented in Table 1.

Table 1. Packages transmission experiments for two smartphones

Parameters (frequency , max delay)	Phone 1 (lost packeges %)	Phone 2 (lost packeges %)
50Hz, 2s	0.0163% (0.5526%)	0.1484%
50Hz, 5s	0.0397%	0.0389%
100Hz, 2s	0.1992%	0.2325%
100Hz, 5s	0.0786%	0.0623%

Intuitively, extending the delay time increases the number of packets reaching the application. Similarly, reducing the frequency from 100Hz to 50Hz reduces the number of packets lost. However, it is worth noting that network traffic is also affected by other transmissions both in the WiFi network and the computer itself, which can significantly affect the result of the analysis – the value in brackets in the table.

4.1 Motion Mapping Accuracy Test

These tests focused on assessing how accurately the system maps the user's movements on the avatar and identifying potential issues related to the precision and stability of the mapping. The sensors used in the system have a 360-degree rotation range. However, during normal use, when the sensors are strapped to the legs, the full range of rotation is not anticipated. The rotation ranges of the sensors have been carefully checked to ensure that the system reproduces the movements accurately in typical usage scenarios (Forward Leg Swing, Side Leg Swing, Leg Kick and Squat). The figure (Figure 6) shows what the tested tilting of the leg to the side, in this case the right leg, looks like.

Figure 6. The experiments of movement possibilities

These tests included several repetitions of the following movements: leg forward backwards, leg tilts to the side, leg kick and squat. Based on these tests, it can be concluded that the application reproduces movements precisely, even with multiple repetitions. Observations have shown that: minimal deviations in rotation, sometimes after the movement the rotation may be slightly higher or smaller than it should be, but this is automatically corrected in subsequent sensor data transfers. This means that the leg, despite the lack of movement of the user, can slightly correct its rotation.
Some errors in accuracy were caused by the phones being loosely attached to the leg, which caused the phone to rotate in unexpected ways. Correct mounting of sensors is crucial to maintain high precision of motion mapping.

During these tests, it was observed that the sensor data smoothly mapped movements, even at times when the camera could not keep up with the movement (Fig. 6). This is proof of the system's high precision, despite minor accuracy issues mainly due to the mechanical aspects of sensor mounting.

4.2. Testing the use of the solution in the game

This test consists in assessing the accuracy of the user's movement reproduction. The test is done by turning on the program and subjectively assessing by the user whether the movements are accurately mapped in the VR environment. To accurately test the mapping of movements, a separate scene was created in Unity (Figure 7) that simulates a football field.

Figure 7. Simulation of a test with the defence of a flying ball

The player's task is to prevent the ball from flying into the goal by using the app's leg mapping. The user evaluates whether the movements of their lower limbs are accurately reproduced in the VR environment on the avatar. The test

was conducted on a group of 6 people unfamiliar with VR technology, but each of them confirmed that the mapping and movement take place in a realistic way. Users have found that their limb movements are accurately mapped in VR, greatly improving the immersion and quality of the VR experience. However, it has been noted that sometimes the limbs twitch, which may be due to the phones being loosely attached to the legs or the game being too sensitive to the readings obtained.

5. Results

The project showed that the movement of the lower limbs can be accurately mapped using rotational sensors in phones. The approach presented in the paper successfully achieved the intended goals, providing a functional and precise solution for mapping lower limb movements in VR. The identified limitations and difficulties indicate areas that can still be improved, which opens up a wide range of opportunities for further development and optimization of the system.

Several limitations and difficulties were encountered during the implementation of the project. The main challenge was to ensure low latency and minimal loss of data packets when transmitted over UDP. While increasing the allowed latency helped reduce losses, there were still some delays that could affect the smoothness of motion mapping. The need to use multiple smartphones as sensors can be inconvenient for users and needs further optimization. In addition, the complexity of calibration and initial settings could be challenging for users less familiar with VR technology. Limb vibration problems, although sporadic, indicate the need for further research and optimisation of sensor data processing algorithms.

The project offers many opportunities for development and further research. One of the directions of development may be the integration of more advanced motion sensors or the development of dedicated devices that could replace smartphones, offering better precision and convenience of use. It is also possible to further improve data processing algorithms to reduce latency and improve the smoothness of motion mapping. Extending the functionality of the system with additional sensors on other parts of the body could enable full tracking of full-body movements in VR, which would significantly increase the realism and immersion of VR experiences. In addition, research into optimizing network communication could improve data transmission reliability and reduce the number of lost packets, which is crucial for real-time applications.

Acknowledgments

The work was fulfilled within the framework of Erasmus+ project "The transferable training model - the best choice for training IT business leaders" (project no. 2023-2-PL01-KA220-HED-000179445). Namely, the work contributes to the project results concerning studying use cases of AI and IoT good practices (work packages 2 and 3).

Reference

1. Virtual Reality (VR). interaction-design. [Online] 10 06 2024. https://www.interaction design.org/literature/topics/virtual-reality.
2. Pimax What is VR? Virtual Reality Explained. pimax. [Online] 10 06 2024. https://pimax.com/blogs/blogs/what-is-vr-virtual-reality-explained.
3. Lowood Henry E. virtual reality. britannica. [Online] 10 06 2024. https://www.britannica.com/technology/virtual-reality.
4. Studiobinder what is virtual reality. studiobinder. [Online] 10 06 2024. https://www.studiobinder.com/blog/what-is-virtual-reality/.
5. Mysticle The. youtube. [Online] 10 06 2024. https://www.youtube.com/watch?v=6JqEJ17ul9k .
6. Vive tracker3. vive. [Online] 10 06 2024. https://www.vive.com/uk/accessory/tracker3/. 16. Developer.oculus move body tracking. developer.oculus. [Online] 10 06 2024. https://developer.oculus.com/documentation/unity/move-body-tracking/?locale=pl_PL.
7. Calliepepper. Slimevr101. docs.slimevr. [Online] 10 06 2024. https://docs.slimevr.dev/slimevr101.html.
8. Wikipedia User Datagram Protocol . wikipedia. [Online] 10 06 2024. https://en.wikipedia.org/wiki/User_Datagram_Protocol.

9. Principles of Motion Sensing. wikipedia. [Online] 10 06 2024. https://en.wikipedia.org/wiki/Principles_of_Motion_Sensing.
10. Virtual reality motion tracking technology. servreality. [Online] 10 06 2024. https://servreality.com/blog/virtual-reality-motion-tracking-technology/.
11. What sensors are used in ar vr systems faq. sensortips. [Online] 10 06 2024. https://www.sensortips.com/featured/what-sensors-are-used-in-ar-vr-systems-faq/.
12. Developer.android sensors_position. developer.android. [Online] 10 06 2024. https://developer.android.com/develop/sensors-and-location/sensors/sensors_position?hl=pl..
13. User datagram protocol udp. cloudflare. [Online] 10 06 2024. https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/.
14. Eater Grant Sanderson Technology by Ben. eater quaternions. eater. [Online] 10 06 2024. https://eater.net/quaternions.
15. Buyuksalih, Ismail, et al. 3D modelling and visualization based on the unity game engine–advantages and challenges. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 4 (2017): 161-166.
16. Kaja Ines Raszyd, Alicja Wesołowska, Klaudia Tomaszewska. Sztuczna Inteligencja w nauce – jak studenci wykorzystują AI w edukacji wyższej. Akademia Zarządzania, ISSN 2544-512X, 8(3), pp. 373-400, 2024.
17. Fan, Xueman, J. Wu, and L. Tian. "A review of artificial intelligence for games." Artificial Intelligence in China: Proceedings of the International Conference on Artificial Intelligence in China. Singapore: Springer Singapore, 2020.

Appendix 1. Codes to manage your data transfer app

Starting the initialization of the connection related to the SendData button

```
public void ConnectButton_Clicked()
{
if (!isSendingData)
{
try
{
string ipAddress = textAddress.text;
string portText = textPort.text;
int port = int.Parse(portText);
bool isConnected = ConnectToServer(ipAddress, port);
if (isConnected)
{
textStatus.text = "Połączono z serwerem";
isSendingData = true;
StartSendingData();
}
else
{
textStatus.text = "Błąd połączenia z serwerem";
}
}
catch (Exception ex)
{
textStatus.text = $"Błąd: Error 001";
}
}
}
```

Method responsible for creating the connection

```
public bool ConnectToServer(string ipAddress, int port)
{
try
{
udpClient = new UdpClient(ipAddress, port);
return true;
}
```

```
catch (Exception ex)
{
return false;
}
}
```

Sending data

```
private void StartSendingData()
{
isSendingData = true;
sendingThread = new Thread(SendDataLoop);
sendingThread.Start();
}
```

Pętla wysyłająca dane

```
private void SendDataLoop()
{
int freq = int.Parse(textFreqency.text);
while (isSendingData)
{
try
{
SendDataToServer(converter());
Thread.Sleep(freq);
}
catch (Exception ex)
{
textStatus.text += ex.Message;
}
}
}
```

Appendix 2. The most important methods from the implementation of the limb simulation management application.

Creating a listening server

```
public UdpServer(int port)
{
this.port = port;
}
public async Task StartListening()
{
udpListener = new UdpClient(port);
Debug.Log($"Server started on port {port}, waiting for connection...");
try
{
while (true)
{
UdpReceiveResult result = await udpListener.ReceiveAsync();
string receivedData = Encoding.UTF8.GetString(result.Buffer);
DataConnector.Instance.SendData(port, receivedData);
}
}
catch (Exception ex)
{
Debug.Log($"Error receiving UDP data on port {port}: {ex.Message}");
}
}
```

Distribution of received data

```
public void SendData(int port, string data)
{
switch (port)
{ 39
```

```
case 12345:
{
DataSentToPort12345(data);
break;
};
…
default:
Debug.LogError($"Invalid port number: {port}");
break;
}
```

Data subscription

```
private void Subscribe(int port)
{
switch (port)
{
case 12345:
DataConnector.Instance.DataSentToPort12345 += ProcessData;
sensorName = "LeftTop ";
break;
…
default:
Debug.LogError($"Invalid port number: {port}");
break;
}
}
```

Data deserialization

```
Regex regex = new Regex(@"X:\s*(-?\d+[\.,]?\d*(?:E[-+]?\d+)?),\s*Y:\s*(-?\d+[\.,]?\d*(?:E[-
+]?\d+)?),\s*Z:\s*(-?\d+[\.,]?\d*(?:E[-+]?\d+)?);\s*X:\s*(-?\d+[\.,]?\d*(?:E[-
+]?\d+)?),\s*Y:\s*(-?\d+[\.,]?\d*(?:E[-+]?\d+)?),\s*Z:\s*(-?\d+[\.,]?\d*(?:E[-
+]?\d+)?);\s*X:\s*(-?\d+[\.,]?\d*(?:E[-+]?\d+)?),\s*Y:\s*(-?\d+[\.,]?\d*(?:E[-
+]?\d+)?),\s*Z:\s*(-?\d+[\.,]?\d*(?:E[-+]?\d+)?),\s*W:\s*(-?\d+[\.,]?\d*(?:E[-
+]?\d+)?);\s*(\d+:\d+:\d+\.\d+)");
if (match.Success)
{
var cultureInfo = new CultureInfo("en-US");
cultureInfo.NumberFormat.NumberDecimalSeparator = ".";
accelerometer = new Vector3(
float.Parse(match.Groups[1].Value, NumberStyles.Float, cultureInfo),
float.Parse(match.Groups[2].Value, NumberStyles.Float, cultureInfo),
float.Parse(match.Groups[3].Value, NumberStyles.Float, cultureInfo)
);
…
attitude = new Quaternion(
float.Parse(match.Groups[7].Value, NumberStyles.Float, cultureInfo),
float.Parse(match.Groups[8].Value, NumberStyles.Float, cultureInfo),
float.Parse(match.Groups[9].Value, NumberStyles.Float, cultureInfo),
float.Parse(match.Groups[10].Value, NumberStyles.Float, cultureInfo)
);
timestamp = DateTime.Parse(match.Groups[11].Value);
return (accelerometer, gyroscope, attitude, timestamp);
```

Data processing

```
private Quaternion rightCoordToUnityCord(Quaternion q)
{
Quaternion originalRotation = new Quaternion(q.x, q.y, q.z, q.w);
// Quaternion reprezentujący rotację o 180 stopni wokół osi Y
Quaternion yRotation180 = new Quaternion(0, 0, 1, 0);
// Mnożenie oryginalnego quaternionu przez rotację wokół osi Y
return originalRotation * yRotation180;
}

attitude = rightCoordToUnityCord(attitude);
```

DOI: https://doi.org/10.53052/9788367652292.17

```
if (first)
{
startingPhoneRotation = attitude;
muscleObject.transform.rotation = startingRotation;
startingPlayerRotation = muscleObject.transform.rotation;
first = false;
}
```

Avatar Update

```
Quaternion relativeRotation = Quaternion.Inverse(startingPhoneRotation) * attitude;
muscleObject.transform.rotation = startingPlayerRotation * relativeRotation;
```

DOI: https://doi.org/10.53052/9788367652292.17