Hand Gesture Recognition using Glove Mounted Sensor Data

Szabolcs Hanzel*, Tamás Kari*, Dávid Varga*, Katalin Péter†, Ferenc Füstös†, Csaba Sulyok*
*Faculty of Mathematics and Computer Science, Babeș-Bolyai University
RO-400084 Cluj-Napoca, Romania
E-mail: {szabolcs.hanzel, tamas.kari, david.varga1}@stud.ubbcluj.ro, csaba.sulyok@ubbcluj.ro
†Codespring
RO-400458 Cluj-Napoca, Romania
E-mail: {peter.katalin, fustos.ferenc}@codespring.ro

***Abstract*—This paper presents a method to recognise fine motor hand movements and visualise them in virtual space by using a smart glove. Sensors mounted on the glove provide data to determine the position of the hands. The presented system is capable of real-time motion recognition by processing continuously incoming sensor data, making it suitable for solving complex problems such as interaction with the VR/AR world, sign language translation or even as a training tool for physiotherapists.**

A microcontroller-based glove is built for the project, which monitors head and hand position as well as finger joint curvature, drawing inspiration from open source projects such as OpenGloves and LucidVR. Sensor data from the Oculus Rift VR glasses and the glove is extracted and displayed using the SteamVR extension in the Unity video game engine. The data is streamed to the gesture recognition server, which classifies the motion into one of the predefined hand motion classes.

The paper demonstrates the practical utility of the smart glove through a simulated traffic management system. The user can control traffic using the glove by predefined hand movements. The virtual space is based on the Unity3D game engine and Unity VR technology.

***Index Terms*—gesture recognition; smart glove; joint movement; virtual reality**

I. Introduction

Gesture recognition [1] refers to the identification of meaningful expressions of motion by a human, involving the hands, arms, face, head and/or body. Hand gestures are researched in the field of computer science as they encapsulate essential information for use cases ranging from sign language recognition through medical rehabilitation to interaction with virtual reality. The main approaches to hand gesture research can be classified based on the nature of their input data: some deploy wearables such as gloves equipped with sensors, while others use cameras and analyze visual information [2].

This study focuses on the glove-based sensor approach of hand gesture recognition. The main motivation lies in the recognition of fine motor hand movements in real time, and their representation in virtual space. Both problems require precise determination of the position of the hands. Data with sufficient accuracy is collected using a glove, which is created during the project. The purchase of such a glove proves to be expensive[1], thus it is made in-house using affordable components.

The literature shows several solutions for detecting hand movements and gestures. Chen et al. [3] extracts the data describing hand movements from images captured by a monochrome 2D camera and uses a hidden Markov model (HMM) for recognition. Their recognition system consists of three main parts: real-time hand tracking, feature extraction, HMM training and HMM-based gesture recognition. Munir Oudah et al. [4] discusses the advantages and disadvantages of different hand gesture recognition systems. Systems that use a camera require fewer tools and are easier to use, but have the disadvantage of separating the hand from the background of the image. In contrast, gloves using sensors can extract more accurate data but limit the user's freedom of movement. This is due to the connection to a computer. Connely et al. [5] presents a similar glove with sensors that acts as a monitoring device for patients with arthritis. The main use of the glove described is to monitor the degree of flexion of the fingers, thereby assessing the patient's condition and accelerating the healing process. Their glove focuses on the detection of finger flexion, in contrast to the glove presented in this paper, where, in addition to finger curvatures, additional data describing hand movements is taken into account to detect movements.

The proposed system is capable of recognizing the continuously incoming hand motion data in real-time, thus it is suitable for solving various complex problems. In the VR/AR world, it serves as an additional tool for making interactions more realistic. It can also be used as a translation tool for sign language or a tool to assess the patients in physiotherapy.

For the recognition the data is collected from the separate components of the glove. The data consists of several parts: head position and rotation provided by the Oculus Rift[2] VR headset. The position and rotation of the hands are also parts of the data, they are calculated based on values from the Oculus Rift controllers and from the finger curves provided by the microcontrollers. A neural network is responsible for motion recognition, which is accessed through a server. To ensure

[1]XSens metagloves buyable example: http://bit.ly/4by7zqn
[2]Oculus Rift: https://www.oculus.com/rift-s/features/

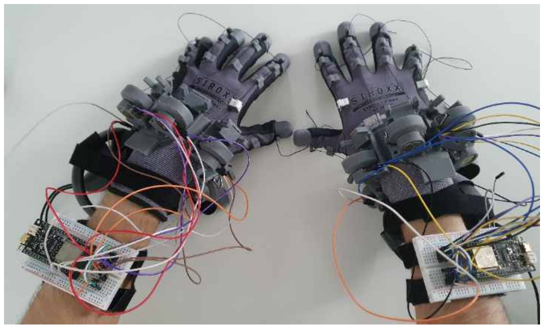

Fig. 1. The glove with the sensors used for the hand movement recognition

real-time information exchange, communication is done via a WebSocket connection [6].

Another element of the project is the deployment of a motion recognition server on a Kubernetes cluster [7]. In order to optimize the deployment, a multi-phase GitLab pipeline is used to provide access to the neural network used by the server and to containerize the server itself.

The remainder of the paper delves into the three main topics mentioned: data processing within the virtual space, the server responsible for motion recognition, and the deployment of the server. Section II describes the process of building a glove for motion detection, as well as the required components and their operation. Section III details the architecture of the virtual space, the communication with the server, and describes a possible application and its components. In addition, Section IV introduces the server, provides a description of its deployment and explains the operation of the artificial intelligence responsible for motion recognition, its architecture and the prediction process. Finally, Section V draws conclusions from the thesis and identifies some opportunities for improvement.

II. The Microcontroller-based Glove

The current section describes the construction of a glove (see Fig. 1) meant to accurately identify hand movements both static (position) and dynamic (gestures). The fundamental signal to be monitored is the curvature and joint position of the fingers. Hence, the main components of the glove are sensors measuring finger curvatures, and controllers measuring global hand positions provided by the Oculus Rift [8]. The headset can approximate the curvature of the fingers according to the positions of the fingers on the buttons of the controllers, but this data is not accurate. The glove extension is introduction to increase measurement accuracy. The sensor data consists of several parts: the position and rotation rate of the headset, which determine the position of the head, and the data provided by the controllers. These describe the position and rotation of the hands. The data is complemented by finger curvature values from additional sensors (see Section II-A), which help to detect accurate hand movements.

Therefore the glove gives the user the possibility to influence the virtual world through natural movements with their own hands, presenting potential usefulness in various environments.

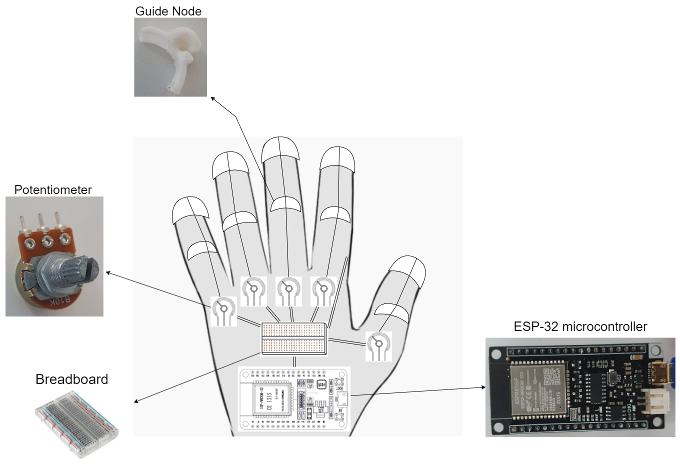

Fig. 2. The glove components

A. Build

The physical architecture of the glove follows the LucidVR[3] project. Fig. 2 depicts its components: an ESP32 microcontroller [9], potentiometers for each finger, a spring structure from a moveable card holder, and 3D printed parts to hold them together. The ESP32 microcontroller is mounted on a breadboard (electronic test board), which facilitates the wiring of the cables and their placement. The potentiometers are placed in separate plastic printed parts that track the operation of the movable card holders. The plastic parts printed for the potentiometers contain a spring mechanism to hold the thread taut for proper operation of the moving contact. When the finger is not locked, the potentiometer is returned to the “zero” position. The thread that connects the potentiometer to the finger is passed through a plastic support. This is necessary for the tension of the thread.

B. Functioning

Each glove is equipped with an ESP32 microcontroller - paired with five potentiometers positioned to fit human fingers. The positioning of the potentiometers allows the curvature of the fingers to be determined by means of threads and springs (see Fig. 2). The potentiometer is connected to an input point of the microcontroller. These devices act as sensors by measuring the bending of the user’s fingers. The twisting causes the voltage to change. This way, the data is accurate, as the slightest movement leads to a voltage change.

C. Calibration

To collect finger curve values, the user has to put on the gloves, attach the Oculus Rift controllers and then connect the ESP32 microcontroller on the gloves to the computer via a USB cable. The calibration requires a fist clench for a few seconds, as this is how the system remembers the limits of

[3]https://github.com/LucidVR

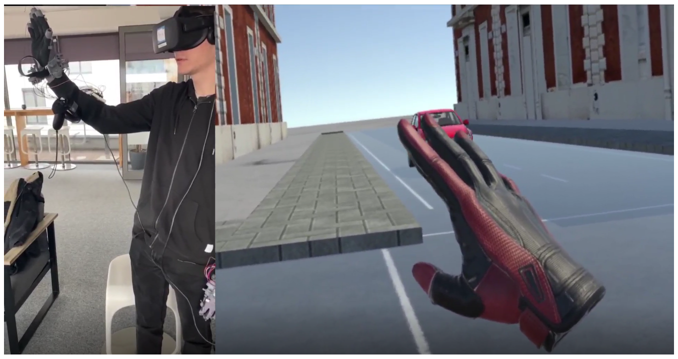

Fig. 3. Traffic management inside the VR application

the physical glove's range of motion. The user can adjust the rotation of the two arms via the OpenGloves[4] software. For this step, another person is needed to press the "Start Calibration" button, which will stop the movement of one virtual arm. The next step in the calibration process is for the user to rotate his/her own hand in the same direction as the virtual hand. This operation must be repeated for both hands separately. The result of the calibration is that the position and rotation of the two physical hands are identical to the virtual hands.

III. Virtual space

The current section proposes a visualization mechanism for the presented hardware, moving it into the virtual space and maximizing interactivity.

There are several ways to implement visualisations and simulations: for example, using Python's PyTorch package. Unity[5] as a video game engine helps to achieve visualisation by providing built-in solutions for graphical rendering and physical simulation, thus facilitating the development process. It also provides an easy to use development solution for VR systems, as well as the possibility of free licensing. For the VR implementation, the SteamVR[6] [10] package is used.

A. Traffic management system

A traffic management system implemented in a virtual space illustrates the practical benefits of the hand motion detection gloves.

The user acts as a traffic police officer. His task is to control cars by hand movements within a virtual space (see Fig. 3). At the same time, the technologies and methods used here can be used in any other related field. The user can give different commands to the cars by imitating the police officer's work. He can stop them or let them pass after the intersection is cleared. He can also speed up or slow down the cars.

[4]https://github.com/LucidVR/opengloves-driver
[5]https://www.unity.com/
[6]https://github.com/ValveSoftware/steamvr_unity_plugin

1) City: The virtual space consists of dynamically generated elements. These elements are buildings, roads and intersections represented in the virtual space. The location of each element is defined by a map in the form of a matrix. It has externally overridable values for dynamic generation. In the case of roads, the type of road, the location and type of intersections between roads can be specified. These are modelled from the Free Road Unity package. Furthermore, roads are composed of several smaller mesh details that specify the shape of the road. In addition, the package natively includes the road search algorithm A*, specialized for roads. This is used in traffic by the drivers.

The FreeRoad package includes road-related visual elements: road models with pavements and different types of road intersections. In addition to the visual elements, it provides a graphical interface for building paths.

2) Car mechanics: In order to achieve a good user experience, it is important to have a realistic approach to the car movement in the traffic management simulation. The movement of the cars are influenced by many simulated forces, such as gravity, acceleration, drag, friction, elasticity and even the force of constraint in case of a collision.

During the project, a realistic engine, brake, steering, wheel friction and suspension mechanics are implemented. In addition, a simple car crash mechanism, mainly for visual purposes, is also implemented. The simulation offers two different car models.

The Unity Wheel Collider system is responsible for most of the cars operation. The Wheel Collider is a collider designed for land vehicles, with built-in collision detection and a friction physics model based on sliding. The Wheel collider has many variables that can be manipulated to model different types of land vehicles. It is possible to set the wheel attributes, shock absorbing features and the springs behaviour.

3) Drivers: In addition to the operation of the cars, it is important to control them, also in a way that is close to reality. An algorithm is responsible for the drivers decision making and its intensity. Drivers can adjust their speed to the car in front of them. They also brake before turns and, in case of other drivers.

In order not to make the traffic simulation too monotonous and predictable, each driver has five so-called personality variables, which are randomly assigned from a well-defined interval. The personality variables determine the driving style of the driver.

Each drivers behaviour can be overriden by the traffic officers commands. If the police officer turns towards the car and shows a predetermined hand movement with the glove, the driver closest to the officer will respond to the hand signal and starts following the new instruction. In the absence of an instruction, drivers will return to their original driving pace and style.

B. Communication with the server

The virtual reality management system acts as a client side that communicates with the server side (see Section IV). The data packet that the client sends to the server consists of the values needed to recognize hand movements. These values are various sensor data: the position and rotation of the head, as determined by the Oculus headset; the positions and rotations of the hands, which come from the Oculus controllers; and the finger curvature values from the glove potentiometers. The data is packaged in JSON format.

The client side sends the collected data to the server for recognition, continuously asynchronously, annotated over time, one image at a time.

The server processes and evaluates the data and responds in text form. The server's response is the name of the identified hand movement class, or the neutral class if it could not be recognized.

IV. Gesture Recognition

Gesture recognition is performed by a server implemented in Python, using the FastAPI[7] web framework, and run in the Uvicorn asynchronous server gateway interface (ASGI). The server exposes a single, WebSocket-based [11] endpoint for bidirectional communication, which provides an inconspicuous delay in the response.

The server is deployed on a Kubernetes [7] cluster. Kubernetes is an open source platform for managing containerized workloads and services. The purpose of deployment is to make the predictions provided by the server continuously accessible.

The project is built around a *continuous deployment* [12] concept. Continuous integration pipelines automate building, Docker containerization, which allows the server to run in an isolated environment, and also the deployment processes.

The Kubernetes topology of the project includes an *ingress*, ensuring external access to the server through preset access rules. It also provides load balancing, HTTPS access and name-based virtual hosting.

Gesture recognition is performed using artificial intelligence methods. The first step in gesture recognition is to be able to recognise static hand movements. These movements consist of only a single snapshot. The trained AI, a multilayer perceptron (MLP) [13] neural network (see Section IV-B), is able to classify a total of 15 classes, which are studied as a guide rather than as an upper bound. On the other hand, continuous gesture recognition required the neural network to not only recognize snapshots, but also take historical data into consideration. To this end, the MLP model is replaced by a recurrent neural network based one.

Recognition begins with preprocessing, molding the raw data into a format that can be interpreted by the neural network. The latter then performs the classification.

A. Preprocessing

The data received by the server is in JSON format, including the positions of the head, hands and the curvatures of the fingers. The preprocessing transforms the data into an array. This includes a frequency function enriched version of the head direction vector and the hand positions, rotation and finger curvatures. The method of storing the points of the space is based on the preprocessing of the NeRF [14] model.

[7]https://fastapi.tiangolo.com/

A frequency function defines the position of a point, the result of which is an array, and its elements give accurate information about the spatial position over a smaller interval (in this case $[0, 1]$). Otherwise, the problem may arise that the neural network places too much emphasis on some properties of a point. For example, the data may be in different quantities or distributed over different intervals.

The direction vector of the coordinates defining the head is as follows:

$$f(x) = \begin{cases} \cos(x) * \cos(z) \\ \sin(x) \\ \cos(x) * \sin(z) \end{cases}$$

Here the variables x and z define the horizontal and vertical forward directions of the head. These coordinates are obtained from the frequency function. Furthermore, the position of the hands, rotation and curvature of the fingers are enriched using the frequency function.

The formula for the frequency function:

$$f(x) = \begin{cases} \cos(2^L * \pi * x) \\ \sin(2^L * \pi * x) \end{cases} ,$$

where x is a real number, or

$$f(x) = \begin{cases} \cos(2^L * x) \\ \sin(2^L * x) \end{cases} ,$$

where x is an angle, and $L = \overline{(1, n)}, n \in N$ refers to an element of the array.

The final step of pre-processing is to convert the data into a tensor, to be used as an input to the neural network. A tensor is a multidimensional array with elements of the same type.

B. Neural network

The classification of hand movements is handled by an artificial neural network [15] consisting of three layers: a linear layer, a long-short term memory layer (LSTM), and finally another linear layer. The output of the latter is passed to the Softmax function, which returns an array. The class predicted by the neural network is the significantly largest element of the array.

Recurrent neural networks (RNN) [16] are suitable for processing sequential data models, which is a problem for other models that can only work with fixed-size data models, such as MLP (Multilayer-Perceptron) and CNN (Convolutional Neural Network). *Long short-term memory (LSTM)* [17] is an improved version of RNN. Among other things, LSTM solves the vanishing gradient problem of the RNN by manipulating the flow of information through three different gates: forget, input and output. The forget gate selects the information to be erased; the input gate decides the new data to be added; and the output gate determines the information to be sent from the cell state to the output. It also introduces the concept of cell state and has its own equations and operations. With all these modifications, it ensures long term data retention against the RNN.

With *linear layer* it is possible to achieve higher performance and accuracy with the network. The layer itself is, in most cases, a simple matrix multiplication, to which a so-called bias vector is added to ensure the layer's accountability. The layer is usually followed by a non-linear activation function, which prepares the output of the layer for the next step of the mesh.

Softmax [15] is an activation function that is predominantly used to solve classification problems. The result is a probability vector showing the probability distribution between classes. The sum of the elements of the output vector must return 1, or a value within the rounding error bounds. The activation function can be written as follows:

$$softmax(z_i) = \frac{e^{z_i}}{sum_{j=1}^{N} e^{z_j}}$$

The number of snapshots used for prediction is a system preference. 10 snapshots are taken every second. In the course of the project, 25 snapshots are chosen, due to the nature of the data used for testing. The model classifies the data into three classes, these are "Stop", "Accelerate", and "Slow down". All of the mentioned classes consists of repetitive hand movements. The model's F1-score averages around 82.5%. Data that cannot be classified into one of these classes is assigned to the class defined as "None". Finally, the model returns the predicted class as a response to the client.

V. Conclusions and further development

This paper presents the development of a system with two main aspects: artificial intelligence-based recognition of hand movements and visualisation of hand gestures in a virtual space powered by a video game engine.

For the recognition, the data received from VR sensors is supplemented by data received from the smart glove built during the project. The VR devices used are the Oculus Rift glasses and its controllers, providing data about the head and hand positions, respectively. The glove provides more detailed information about the position of the hands, namely about the curvature of each finger.

Behind the artificial intelligence is a neural network, whose training data is fed by the glove. The virtual space provides a suitable environment for testing the neural network.

To achieve more accurate results, modifications to the construction of the glove are necessary. Resizing and redesigning the plastic printed parts that hold the glove components together would provide a better experience for users. In order to match the real finger movements as closely as possible to the recorded finger curves, it is necessary that the glove can operate within the appropriate parameters without problems.

For a better user experience an improved calibration process could be developed which doesn't require the help of another person.

Since the ESP32 microcontroller has built-in Bluetooth support, the Smart Glove is able to offer the user a greater range of motion by replacing the USB cable connection to wireless connection. To implement a cable-free connection, external batteries are required to act as power source.

The range of use can be increased by adding servo motors. By incorporating servo motors, a more realistic simulation can be achieved, as they can be used to provide feedback to the fingers when grasping objects in virtual space, thus making the user experience more realistic.

References

[1] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 311–324, 2007.

[2] M. Oudah, A. Al-Naji, and J. Chahl, "Hand gesture recognition based on computer vision: A review of techniques," *Journal of Imaging*, vol. 6, no. 8, 2020. [Online]. Available: https://www.mdpi.com/2313-433X/6/8/73

[3] F.-S. Chen, C.-M. Fu, and C.-L. Huang, "Hand gesture recognition using a real-time tracking method and hidden markov models," *Image and Vision Computing*, vol. 21, no. 8, pp. 745–758, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885603000702

[4] M. Oudah, A. Al-Naji, and J. Chahl, "Hand gesture recognition based on computer vision: A review of techniques," *Journal of Imaging*, vol. 6, no. 8, 2020. [Online]. Available: https://www.mdpi.com/2313-433X/6/8/73

[5] J. Connolly, J. Condell, B. O'Flynn, J. T. Sanchez, and P. Gardiner, "IMU Sensor-Based Electronic Goniometric Glove for Clinical Finger Movement Analysis," *IEEE Sensors Journal*, vol. 18, no. 3, pp. 1273–1281, 2018.

[6] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with websocket," *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, 2012.

[7] G. Sayfan, *Mastering Kubernetes: Master the art of container management by using the power of Kubernetes, 2nd Edition*. Packt Publishing, 2018.

[8] B. A. Davis, *Oculus Rift in Action*. Manning, 2015.

[9] A. Kurniawan, *Internet of Things Projects with ESP32: Build exciting and powerful IoT projects using the all-new Espressif ESP32*. Packt Publishing, 2019.

[10] J. W. Murray, *Building Virtual Reality with Unity and SteamVR*. A K Peters/CRC Press, 2020.

[11] A. Lombardi, *WebSocket: Lightweight Client-Server Communications*. O'Reilly Media, 2015.

[12] S. Rossel, *Continuous Integration, Delivery, and Deployment: Reliable and faster software releases with automating builds, tests, and deployment*. Packt Publishing, 2017.

[13] S. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.

[14] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: representing scenes as neural radiance fields for view synthesis," *Commun. ACM*, vol. 65, no. 1, p. 99–106, dec 2021. [Online]. Available: https://doi.org/10.1145/3503250

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2016.

[16] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," *Andrej Karpathy blog*, 2015. [Online]. Available: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

[17] A. Graves, *Long Short-Term Memory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 37–45. [Online]. Available: https://doi.org/10.1007/978-3-642-24797-2_4