Control Interface for Multi-User Video Games with Hand or Head Gestures in Directional Key-Based Games

Oscar Ramirez-Valdez, César Baluarte-Araya, Rodrigo Castillo-Lazo, Italo Ccoscco-Alvis,
Alexander Valdiviezo-Tovar, Alexander Villafuerte-Quispe, Dylan Zuñiga-Huraca
Universidad Nacional de San Agustín de Arequipa, Arequipa, Perú

***Abstract*—This paper describes the development and implementation of a hand or head gesture-based control interface for video games, enhanced for games that use directional keys. The objective is to develop an adaptive control system for a multiplayer video game that allows users to choose between the use of traditional directional keys or a gesture-based interface. The methodology used follows the Cross-Industry Standard Process for Data Mining (CRISP-DM) development model, which allows a structured integration of analysis, design, implementation and evaluation steps. Technologies such as OpenCV, MediaPipe and deep learning algorithms are used, translating hand movements into directional commands in real time. In addition, the system integrates a client-server architecture based on Node.js that supports multiple users, enabling an immersive gaming experience on PC and mobile platforms. The results highlight the accuracy of the system and its potential to improve accessibility, especially for users with motor disabilities by using their hands or head movements to control the directional keys. Concluding that the control interface for multi-user video games provides the necessary support to gamers in performing the task, promoting accessibility in the entertainment environment.**

Keywords*—Control interface; video games; artificial vision; gesture-based interface; directional commands; human-computer interaction; deep learning algorithms; accessibility; real-time; pattern recognition

I. Introduction

Gesture interaction has revolutionised the gaming experience [13], eliminating the dependence on traditional keyboards and controls. This paper presents an interface that allows video games [36] to be controlled by hand movements, using computer vision [3] and real-time pattern recognition. The solution aims to improve immersion and accessibility, especially for users with motor disabilities, promoting their integration in recreational and therapeutic activities.

The interface employs gesture detection algorithms and a robust client-server system, compatible with PC and mobile devices, providing an inclusive experience. As a result, a gesture recognition system was designed and developed to replace the directional keys, allowing precise and fluid control in real time. In addition, it supports multiple users with individual network configurations and in-game character selection. Developed in Unity, the game integrates traditional and gesture controls, adapting to both PC and mobile.

It is concluded that this interface is viable, offering an accessible alternative for controlling video games through hand gestures or head movements, broadening access to entertainment and promoting inclusion.

II. Theoretical Framework

A. Gesture-based Control Interfaces and Games

Gesture-based control interfaces have revolutionised the way users interact with devices and computer systems [9], opening the door to immersive and intuitive experiences. In the context of video games, these interfaces allow the user to control game elements through body gestures, specifically hand movements. The advantages of this approach include greater immersion and accessibility [23], as it allows play without the need for traditional controls such as keyboards or controllers. To achieve this, advanced gesture recognition [1] and computer vision technologies are used to interpret the user's movements and translate them into real-time actions.

B. Artificial Vision and Gesture Recognition

Artificial vision is a discipline that allows machines to process and interpret the visual world. This technology uses image processing algorithms and automatic learning techniques to identify objects, gestures and patterns in real time [10] [15]. In hand gesture recognition, a sub-area of machine vision, it allows the detection and analysis of specific hand movements to control interactive applications [12]. In the context of this project, libraries such as OpenCV and MediaPipe are fundamental to capture images of hands, identify key points (such as articulations and fingers) and translate this data into control actions for games based on directional keys.

C. Real Time Pattern Recognition

Real-time pattern recognition is key to achieving smooth [2] [11] and accurate interaction in gesture-based interfaces. Real-time recognition systems allow capturing and processing images in a fraction of a second, detecting movements instantaneously. The ability to process gestures in real time is especially relevant for video game applications [25], where any delay can affect the user experience [6] and decrease the effectiveness of the control. To implement this functionality, fast image processing techniques and movement detection and analysis algorithms, optimised to operate on common devices such as webcams, are employed.

D. Image Processing Technologies and Benchmark points Models

Reference point models are essential to accurately recognise the position of fingers and hands. These models identify key points on the hands and, using deep learning techniques, detect specific gestures, such as left, right, up or down movements. These points help determine the orientation and height of the hands, which are essential for translating gestures into commands in the videogame. The precision of these models depends on the quality of the camera and the capacity of the algorithms to quickly process the images.

E. Gesture Control of Video Games: Benefits and Challenges

Gesture control of video games [7] has significant benefits, including greater immersion and accessibility for users with motor limitations. It also allows for a more natural and direct gaming experience, eliminating the need for additional control devices. However, there are also challenges, such as the need for recognition algorithms that work accurately in varied environments and lighting conditions. Also, minimising the delay [18] between capturing the gesture and executing the command in the game is critical to ensure a satisfactory experience.

F. Implementation of the Control Interface

The implementation of hand gesture control interface for video games [7] based on directional keys requires effective integration [19] of various software and hardware elements. In this project, a combination of Python libraries is used for the creation of the graphical user interface [17], camera control and gesture detection. Tkinter is used to develop the user interface, allowing custom settings for sensitivity and direction control. In addition, OpenCV and MediaPipe are employed for processing the captured images and detecting gestures in real time, as shown in Fig. 1 and Fig. 2.

Fig. 1. Control interface.

Fig. 2. Control of racing game in unity.

G. Server Implementation

The development of an efficient server is essential to guarantee real-time communication between players and to maintain synchronisation during the multiplayer gaming experience, see Fig. 3. The server implementation was carried out using modern technologies such as Node.js, which offers a lightweight and scalable environment, together with libraries such as Express for HTTP route management and Socket.io for real-time communication [20].

Fig. 3. Multi-user racing game in Unity.

III. Related Work

The use of computer vision and hand gesture recognition techniques in video games has been an active area of research. The research in [31] presented GestureFlow, a novel hand gesture control system for interactive games that leverages advanced tools such as OpenCV, Mediapipe and Numpy. The study in [32] presented a methodology for gesture-based contactless operations, combining his algorithm with Mediapipe and OpenCV.

Various studies have focused on the development of game applications based on hand gestures. Thus, the study in [33] employed object detection and an artificial neural network for hand gesture recognition in games, using Python and OpenCV; also the study in [34] developed a game that interacts with users through hand gesture movements, using Mediapipe and Pygame; and the study in [35] explored the use of Mediapipe for real-time online games, creating a gesture recognition-based control system using OpenCV and Python.

The integration of hand gestures and voice commands for immersive gaming has also been studied. In study [36] they developed a game control system based on hand gesture recognition using Mediapipe, OpenCV and Python; also study in [37] reviewed the opportunities of using hand gestures to play video games, highlighting the use of Mediapipe and OpenCV for hand tracking and gesture recognition.

In addition, some studies have explored the application of gesture recognition in rehabilitation and quality of life improvement. Thus, the study in [38] presented a human body gesture-controlled gaming application using OpenCV and Mediapipe; on the other hand, the study in [39] developed a camera-based real-time motion detection gaming tool for cervical rehabilitation, employing a convolutional neural network for hand gesture recognition; also the study [40] used OpenCV functions and Mediapipe modelling technology for real-time human movement recognition and interaction in virtual fitness applications.

IV. METHODOLOGY

The methodology used in this work is based on the Cross-Industry Standard Process for Data Mining (CRISP-DM) development model, adapted for the context of gesture recognition and real-time video game control; it includes the following phases:

Phase 1. Understanding the Business and Defining the Objective

The principal objective of the project is to design and implement a control interface for video games that allows users to control the game using hand gestures or head movements, as an alternative to traditional directional keys. It is not only to improve immersion in the game, but also to promote accessibility for players with motor disabilities.

Phase 2. Data Collection and Preparation

Advanced artificial vision technologies, such as OpenCV and MediaPipe, are used to capture real-time images of the user's hand or head gestures via a webcam. This data is then processed and labelled for gesture detection. The model captures key points, such as finger articulations, and this data is translated into commands for directional key control.

Phase 3. Development of the Gesture Recognition Model

Deep learning algorithms are developed and trained to detect and recognise gestures. Key point reference models are implemented to map hand movements and translate them into directions. The algorithms were optimised to achieve a high degree of accuracy and low latency in real time.

Phase 4. Control Interface System Development and Implementation

Gesture detection is integrated with a user interface developed using Tkinter to configure the sensitivity and controls of the system; the interface also allows for actors selection and network settings. A client-server system is implemented using Node.js and Socket.io to ensure real-time communication between users, and enable a fluently gaming experience.

Phase 5. Integration and Evaluation of the System in Multiplayer Games

The system is integrated into a multiplayer video game developed in Unity, which supports the interaction of multiple players simultaneously. The gesture-based control is evaluated in terms of accuracy, latency and user experience compared to traditional control. Tests are conducted in different lighting conditions and types of movement to ensure the robustness of the system.

Phase 6. Optimisation and Results

Once the basic system is implemented, the algorithms are optimised to improve accuracy and minimise latency. The results of the system are analysed using metrics such as response time and gesture accuracy, and compared to traditional control to determine the effectiveness of the interface in multiplayer games.

This approach ensures a robust solution that not only responds to user requirements, but also contributes to achieving accessibility and usability across PC and mobile platforms.

A. Explanation of Control Interface Code

1) Interface configuration and resources: The graphical user interface (GUI) is created using Python's Tkinter for interface creation; the necessary libraries are imported to manage the interface, threads of execution, running local files and external URLs for Unity-based games [8].

2) Camera and game control: The functions implemented as start_camera_thread and start_camera_tk_thread are in charge of starting the camera capture using different control methods, such as run_virtual_steering and run_virtual_tk, which are executed in separate threads so as not to block the principal interface. Both functions take as arguments the control methods for the directional keys (such as hand or head) defined in get_control_methods; the stop_camera_thread and stop_camera_tk_thread functions stop these threads of execution using stop events (stop_event and stop_event_tk).

3) Interface design: For the principal interface, multiple frames are created representing sections such as the camera control and the Unity game, configured by grid to be arranged according to the selected layout.

The functions show_columns, show_quadrants and show_rows allow different GUI layouts to be changed according to user preference, see Fig. 4 make_draggable is used to make each frame draggable, offering flexibility in the layout of the interface.

Fig. 4. Design of the interface control.

4) Personalisation of configurations: The configurations frame includes sensitivity and distance threshold setting controls, implemented as sliders (ttk.Scale) that allow the user to customise the threshold and sensitivity of the gesture detection system. Additionally, the user can choose between

direction controls with different methods (hand or head gestures), defined in the settings_frame Radiobuttons.

5) Menu bar: The menu bar provides easy access to camera and configuration options. The Checkbuttons allow you to activate or deactivate the main sections of the interface, such as camera control or the Unity game, the views submenu offers different settings for the layout of the frames (show_columns, show_quadrants and show_rows), shown in Fig. 5.

Fig. 5. Menu options.

6) Execution of the main window: The main window is started with root.mainloop(), a loop that keeps the interface active until the user decides to close it, by calling the on_closing function, which stops all active camera threads.

7) Camera processing and visualisation with OpenCV: With and Without Graphical User Interface (GUI)

The run_virtual_steering function starts initialising the modules needed for gesture control [28].

- Image processing without GUI

The mediapipe libraries are used for recognition of hands [16] [26] and face, and OpenCV's cv2 is used for processing the captured image. The webcam is initialised with cv2.VideoCapture(0), capturing the video in real time; a virtual keyboard controller is configured using pynput.keyboard.Controller, used to send simulated commands to the game in Unity, see Fig. 6.

Fig. 6. Camera processing without GUI.

- Image processing with GUI

The real-time processed video is integrated into a graphical interface using Tkinter. A subwindow displaying the processed frames in a continuously updated Label component is shown in Fig. 7.

Fig. 7. Camera processing without GUI.

8) Image processing and hand detection: The software processes each camera frame in real time [24] [29], the captured image is flipped horizontally with cv2.flip for a more intuitive orientation, then converted to RGB before being passed to the mediapipe model to process the hand_results and face_results [22] [27]. These data allow the position and movement of the hands and face to be determined.

9) Direction control based on vertical position of the hands: Are used the reference points obtained to detect the position of the hands on the Y-axis, allowing to differentiate whether the right hand is more up than the left hand or vice versa. When it detects that the right hand is more raised, the program simulates a movement to the right by pressing the Key.right key. Similarly, if the left hand is higher, Key.left is pressed; shown in Fig. 8.

Fig. 8. Ownership of frames.

10)Face distance based control: The program calculates the distance of the face from the area of the face detection box; it is used to detect approaching or moving away movements, activating the Key.up key to accelerate as the face approaches and Key.down to slow down as it moves away. This control allows to adapt the speed of the vehicle within the game, simulating acceleration and deceleration depending on the proximity of the face, as shown in Fig. 9.

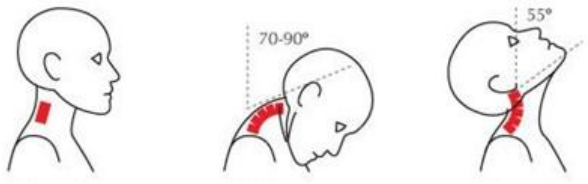

Fig. 9. Head movement conditionals to accelerate or decelerate the vehicle.

11)Additional hand gesture control: The program also includes an additional control [14] based on specific hand gestures [3]. If the index finger is fully extended downwards, it is interpreted as an acceleration gesture, activating the Key.up key, when the index finger is raised, it is assumed as a braking gesture, activating Key.down. This logic, according to the initial configuration in the interface, allows using both face movements and hand gestures [3] [30] to provide a more intuitive control experience [13].

12)Error handling and camera shutdown: If the OpenCV view window is closed or ESC is pressed, the program stops the image processing loop [5]. The close_camera function ensures that all OpenCV camera and window resources are properly released.

13)Initialisation of interface styles: Using the apply_styles function you can customise the visual styles of widgets; thus: a) the TFrame style sets a light blue background, a 5 pixel border and a ridge relief that gives depth to the frame, b) the TLabel style sets a 12 point Helvetica font with the same background, the text uses a darker blue to stand out against the background, c) the TButton style uses a 10 point Helvetica font in bold, with a dark blue background and white text to ensure contrast, d) with style map, the button is adjusted so that, on hover, the background changes to an even darker blue, while keeping the text white to ensure legibility.

14)Initialisation of the camera module with tkinter: OpenCV libraries are imported for video handling, MediaPipe for gesture and face detection, and pynput for keyboard input simulation. Global variables, camera_running (check if camera is active), last_action (store the last action performed) and press_duration (measure how long a key is pressed) are initialized to ensure continuous tracking of the system state during execution.

15)Video capture and processing: The video_stream function uses OpenCV to capture video in real time. Each frame is processed with MediaPipe to detect hand gestures and faces. The key points of the hands are used to calculate their position in space, allowing to determine actions such as moving left or right. In addition, the points and connections of the hands are visualized in the video to facilitate the interpretation of the system.

16)Hand gesture detection: Within video_stream, hand gestures are analyzed using the positions of specific points. This analysis includes logic to avoid simultaneous keystrokes and ensure smooth transitions between actions.

17)Face distance based control: The face distance is implemented by comparing the relative size of the detected bounding box around the face [21]. If it increases or decreases beyond a configured threshold, the "up" or "down" keys are triggered, simulating actions such as accelerating or braking.

B. Explanation of the Server Code

1) Server initialisation and dependency configuration: The express, http, and socket.io libraries are used to create a server in Node.js; an HTTP server is configured with http.createServer and its functionality is extended with socket.io to handle real-time connections.

2) HTTP path to check server status: The path app.get('/') returns a simple message to confirm that the server is running. The /getPlayers path uses a JSON format returning information about the connected players.

3) HTTP path to check server status: The io.on('connection', callback) function handles each client connection. Each connected player is assigned a unique identifier (PlayerID) and is stored in the players object.

4) Player synchronisation and start of the race: When the number of connected players reaches the maximum allowed (maxPlayers), the server sends a startRace event to all clients, indicating the start of the game.

5) Real time position update: The updatePosition event receives data from the clients, such as position and rotation in the X, Y and Z axes; it is updated in real time in the players object; the server emits the updated list of players through io.emit('updatePlayers').

6) Handling disconnections: When a player disconnects, the server deletes his information from the players object and issues an event to update the list of players in the clients.

7) Server and listening port configuration: The server starts on port 3020 with address 0.0.0.0, which allows accepting connections from any IP address, ideal for multiplayer environments.

C. Explanation of the Multi-User Game Code in Unity

1) Automatic object rotation (AutoRotation.cs): The AutoRotation script implements a simple functionality to make an object in Unity rotate continuously around its Y-axis; it is controlled by a public variable rotationSpeed.

2) Dynamic player tracking (FollowPlayer.cs): The FollowPlayer script implements functionality for a camera to follow the player in Unity, dynamically adjusting to the player's position and rotation; it includes support for virtual reality (VR) scenarios, see Fig. 10.

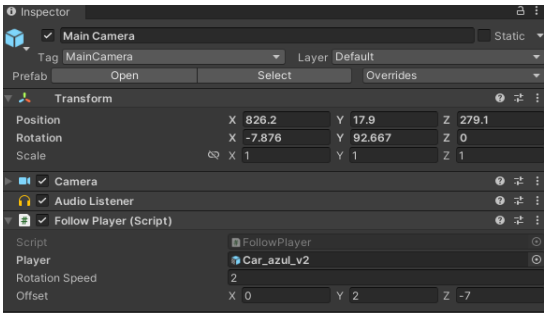

Fig. 10. Player camera configuration.

3) Interactive In-game console (InGameConsole.cs): The InGameConsole script implements an in-game console in Unity, useful for real-time debugging; it allows to display system and user messages in a panel, see Fig. 11.

Fig. 11. View of the start of the game.

4) Panel management and connection in the main menu (Panel.cs): The Panel script manages user interaction with a main menu, providing options to configure a network connection, select colors for a car and manage other related panels.

5) Player behaviour (player.cs): The player.cs script controls the player's behavior in the game, including movement, interaction with the environment, updating the user interface and communication with the server; it is essential to manage the game logic.

6) Player movement management (player.cs): The Update method is called once per frame and handles the main logic of the player's movement. Depending on the platform and the input mode (keyboard or gestures), the input values for horizontal and vertical movement are obtained.

7) Straighten the overturned car (player.cs): The RightCar method is responsible for straightening the player's car by applying an upward force and continuing the game.

8) User interface update (player.cs): The UpdateUI method updates the points and lives texts in the user interface; it is called whenever the player's points or lives change.

9) Collision management (player.cs): The OnCollisionEnter method is called when the player's car collides with another object, if it has the ObjectCollision tag, the player's points are reduced, if the points reach zero, a life is reduced and the points are reset. If the lives reach zero, a "You lost" message is displayed.

10)Restart car position (player.cs): The ResetCarPosition method resets the position and rotation of the player's car to its initial state. This is used when the car falls off the stage or when resetting the player's points and lives.

11)Management and connection to the server (SocketManager.cs): The SocketManager.cs script in Unity handles the connection and communication with a server via WebSockets; it is crucial for the multiplayer functionality of the game, allowing data synchronization between players and the server.

12)Connection to server (SocketManager.cs): The ConnectToServer method establishes the connection to the server using the IP address and the color of the car. It configures connection, disconnection and error, and defines handlers for various game events.

13)Player ID assignment (SocketManager.cs): The OnAssignPlayerID method handles the player ID assignment event; it gets the ID from the server's response and stores it in a local variable.

14)Initialisation and player update (SocketManager.cs): The OnInitializePlayers method creates or updates players at the start of the game, and OnUpdatePlayers updates player positions and rotations during the game.

15)Position and rotation sending (SocketManager.cs): The UpdatePosition method sends the player's position and rotation to the server. It converts the data to a JSON object and outputs it to the server via the socket.

16)Point and life adjustment in the interface (size,cs): The script tamano.cs in Unity adjusts the position and size of dot and life texts in the user interface.

D. Explanation of the Multi-User Game Code in Unity

The main figures of the video game scenario are shown below.

- Principal Camera, shown in Fig. 12.

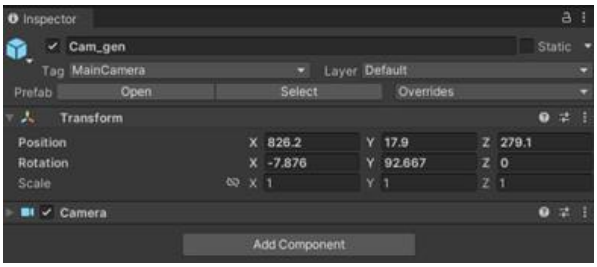

Fig. 12. Game camera position configuration.

- Vehicle models, we organized the designs of the vehicles to compete in the prefabs folder; this is shown in Fig. 13.

Fig. 13. 3D model of vehicles.

- The design of the race track, as would be the scenario, is shown in Fig. 14.

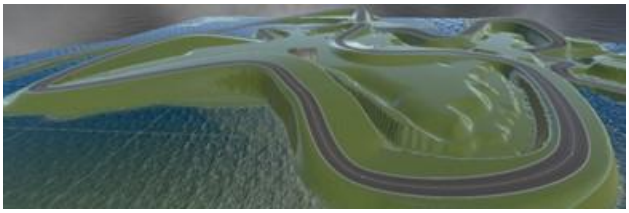

Fig. 14. Running track along the terrain.

- The configuration of the race track is shown in Fig. 15.

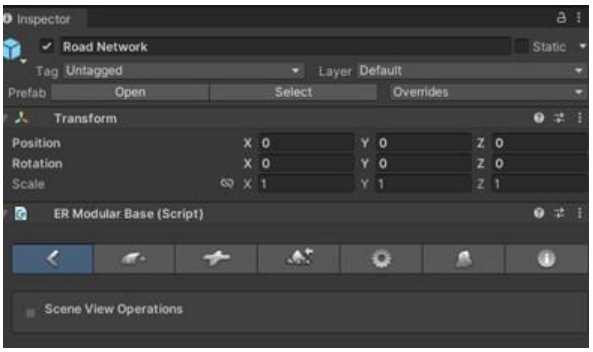

Fig. 15. Race track configuration.

- Principal player design, as shown in Fig. 16.

Fig. 16. Local player configuration.

- Principal menu configuration, three sub-panels are organized within a canvas where the user can configure the ip and connection port, as well as choose the color of the vehicle. If the entered data is validated, the system displays a confirmation message to start the multiuser racing game, as shown in Fig. 17.

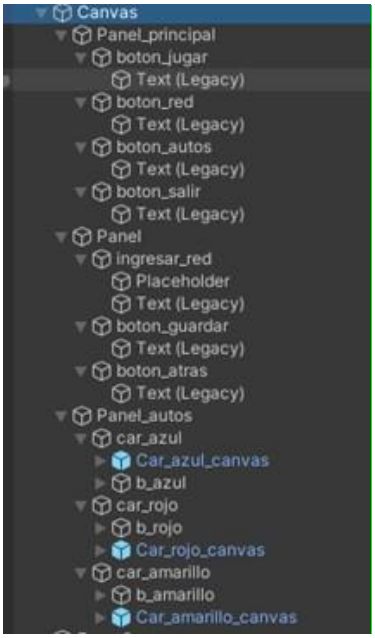

Fig. 17. Components view in unity.

- The principal menu of the game is displayed in the interface shown in Fig. 18.

Fig. 18. Main menu in the game.

- Eleccion of the player's cart, the options referred to the colours, is shown in Fig. 19.

Fig. 19. Selection of the player's car.

- Successful connection view of the player, ready to complete the players, is shown in Fig. 20.

Fig. 20. View of successful player connection.

V. Results and Discussion

A robust client-server system was built using Node.js, capable of supporting a configurable number of players. Each client, developed in Unity, incorporates a script called SocketManager that facilitates bidirectional communication with the server. This allows to synchronise the start of the game when all players have connected and configured.

In the client, a start panel was designed that offers each player the possibility of configuring the network parameters, selecting a character and waiting for the minimum number of participants configured in the server to be reached before starting the game.

The developed videogame was adapted to run on both PC and mobile devices Tablet, Smartphone, see Fig. 21; providing a multiplatform experience. In addition, two control options were incorporated for players using PCs:

- Traditional Control: Use of directional keys to move left, right, forward or backward.
- Gesture Control Interface: A system that uses the device's camera to detect hand and face gestures, which are mapped to the game's directional key actions.

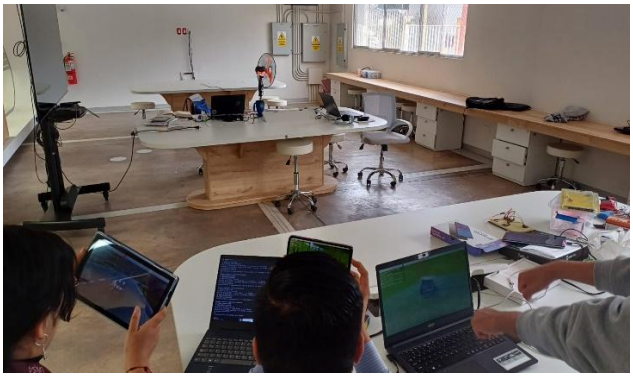

Fig. 21. Video game running on Multiplatform, PC, Tablet, smartphone.

In the following figures the execution of the multiplatform videogame is shown, in Fig. 22 the cars in full race can be seen on two platforms, in Fig. 23 the control of the red car with hand gestures, in Fig. 24 the car at a different speed leaves the circuit, in Fig. 25 with hand gestures the car returns to the circuit and continues the race.

Fig. 22. Cars in full race on two platforms, PC and tablet.

Fig. 23. Controlling the red car with hand gestures.

Fig. 24. Car at a different speed wanders off track.

Fig. 25. With a hands gesture the car is steered back to the circuit and the race continues.

The system executes key presses using two types of control: hand gestures and facial movements, both detected by MediaPipe [14]. For hand control, the relative position of the hands is evaluated: if the right hand is higher than the left hand, the system simulates the action of pressing the right arrow key; if the left hand is higher, it simulates the action of pressing the left arrow key. The green node, which marks the centre between the two hands, indicates that both hands are centred. In addition, the facial movement adjusts the distance control, triggering

zoom in or zoom out actions, depending on the position of the face in front of the camera. For acceleration and deceleration control [31], the system uses the position of the index finger relative to the wrist: if the index finger is higher than the wrist, it simulates the action of pressing the 'up' key to accelerate; if it is lower, it simulates the action of pressing the 'down' key to decelerate, thus allowing the speed to be dynamically adjusted by these movements.

As seen in Fig. 26, Fig. 27, Fig. 28, the system provides visual messages on screen, such as 'Move left (Right hand higher)', which orients the user on the movements required to interact as discussed by [11] with the virtual steering wheel for the game developed in Unity. This information is useful, that in some cases the message shows unrecognisable (''higher'') characters, thus improving the clarity of the instructions.

Fig. 26. Hand movement to the left (a).

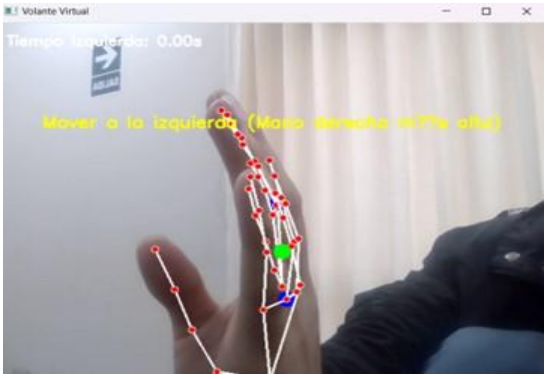

Fig. 27. Hand movement to the left (b).

Fig. 28. Hand movement to the left (c).

The system was able to detect the hand movements [20] in real time [25] and quickly reflect the changes in position by the variations in the positions of the points between the two images, where the hand is seen to rise and change orientation, resulting in an immediate adjustment in the structure detected by the system.

In the case of the multi-user server developed with Node, the tests showed that for a range of 2 to 4 players connected from PC or mobile it is feasible to have a competition with few moments of instability in the connection, which suggests that the optimisation of the secondary threads of each client should be deepened so that the system supports a greater number of users. Table I shows the technical aspects of the multi-user server.

TABLE I. TECHNICAL ASPECTS OF THE MULTI-USER SERVER

Aspect	Technical Details	Benefits	Limitations	Performance Metrics
Server Hardware	Lenovo with Core i5/i7 processor, 12GB RAM, integrated card or NVIDIA graphics.	Good performance on standard test hardware.	Performance may not be representative for lower hardware.	50-60% CPU usage during 4-player testing.
Server Operating System	Windows 11 in both cases (Core i5/i7).	Compatibility with modern systems.	Linux servers may offer better performance.	Response time: ~80-120 ms under ideal conditions.
Player Devices	Android devices (version 12 or higher) and PCs (Linux or Windows).	Stable connections on Android 12 or higher devices.	Variability in connection quality depending on device.	Connection success rate: 98% on mobile devices.
Control interface (PC)	Python control interface for PC connection.	Efficient connection from Linux or Windows PCs.	Interface could be more complex for non-technical users.	Synchronisation time: 100-150 ms
Gesture recognition	Gesture recognition performed with specific cameras and algorithms.	High accuracy in Gesture Recognition with suitable lighting conditions.	Success rate decreases with poor lighting or fast movement.	Success rate: 85-90% with ideal conditions. Failure rate: 15% in low light.
Latency and Response	Tests conducted in controlled environment with low latency local network.	Low latency under controlled conditions.	Latency increases with more players connected simultaneously.	Average latency: ~120 ms in local network. Average latency with 4 players: ~200 ms.

Finally, [7] uses gestural interaction techniques to control a video, as from [4] in hand gesture recognition, in the present work the developed gesture control interface has proved to be an inclusive tool, allowing players with disabilities to use their hands or head movements to control the directional keys in any multiplayer game in Unity. This solution facilitates the participation of all players, regardless of their physical abilities, promoting accessibility in digital entertainment. A sample of the code worked on the system can be seen in Fig. 29.

```
if face_results.detections:
    for detection in face_results.detections:
        bbox = detection.location_data.relative_bounding_box
        left = int(bbox.xmin * width)
        top = int(bbox.ymin * height)
        right = left + int(bbox.width * width)
        bottom = top + int(bbox.height * height)
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

        current_distance = bbox.width * bbox.height
        if last_distance is not None:
            if current_distance > last_distance * (1 + distance_threshold):
                if up_method == "Cabeza":
                    keyboard.press(Key.up)
                    keyboard.release(Key.down)
                    last_action = "Acercando"
                    action_text = "Acercando (Presionando tecla arriba)"
            elif current_distance < last_distance * (1 - distance_threshold):
                if down_method == "Cabeza":
                    keyboard.press(Key.down)
                    keyboard.release(Key.up)
                    last_action = "Alejando"
                    action_text = "Alejando (Presionando tecla abajo)"
            else:
                keyboard.release(Key.up)
                keyboard.release(Key.down)
        last_distance = current_distance
```

Fig. 29. Face detection.

VI. Conclusions

A hand gesture recognition system based on artificial vision [40] and real-time pattern detection was designed and implemented, achieving an innovative alternative to the use of directional keys in video games. This approach improves player immersion and makes the gaming experience more accessible, especially for users with motor disabilities.

The results demonstrated a high accuracy and speed of response to gestures ensuring a smooth and ideal interaction. Furthermore, customisation options were implemented in the interface, such as sensitivity, distance threshold and widget layout, contributing to its usability and versatility.

An efficient client-server architecture was developed that supports multiple simultaneous users, ensuring real-time communication and offering flexibility for individual network configurations and character selection within the game. The design uses threads and resources such as the camera responsibly, improving stability and minimising conflicts during interface use, contributing to increased performance and synchronisation.

The video game developed in Unity was adapted to run on PC platforms and mobile devices, extending its reach and allowing players to enjoy a consistent and immersive experience regardless of the device used.

The integration of traditional controls together with the gesture-based interface ensures greater inclusion of different play styles and user preferences.

In the comparative evaluation of the gesture control system against traditional methods, advantages in innovation and immersive experience were evident. While traditional controls maintain an advantage in accuracy and reliability under less optimal technical conditions, the gesture interface proved to be an intuitive and accessible solution, capable of improving user satisfaction by adapting to their preferences and needs.

Future Works

As a result of the present work, a prospective vision for future work can be gained:

Optimisation of gesture recognition algorithms in variable lighting environments: It is essential to improve the accuracy of gesture recognition systems in changing lighting conditions, ensuring a consistent and reliable user experience.

Application of self-supervised learning in human-computer interaction systems: Implementing self-supervised learning techniques can improve the adaptability and efficiency of gesture interfaces, allowing systems to learn and adjust to individual user preferences and behaviours.

Integration of gesture recognition into augmented and virtual reality interfaces: Combining gesture recognition technologies with augmented and virtual reality environments can offer more immersive and natural gaming experiences, improving user interaction with digital content.

Development of deep learning models for the identification of complex gestures: The use of deep neural networks can facilitate the detection and classification of more sophisticated gestures, expanding the repertoire of available commands and improving interaction in multiplayer games.

Implementing gesture recognition using radar technology for mobile applications: The use of radar sensors in mobile devices can enable accurate gesture recognition without relying on cameras, offering an efficient and less invasive alternative for video game control.

Acknowledgment

Thanks to the Universidad Nacional de San Agustín de Arequipa for the support it provides for the development and implementation of proposals that benefit the continuous improvement of student performance through proposals that help to solve society's problems.

References

[1] M. Kassim, Y. San and R. Norlis, "Hand Gesture Recognition System using Image Processing," 2021 IEEE 17th International Colloquium on Signal Processing & Its Applications (CSPA), Selangor, Malaysia, 2021, pp. 57-62, doi: 10.1109/CSPA52141.2021.9377292.

[2] T. Häckel, C. Eppner, and R. Stolkin, "Self-Supervised Learning of Hand-Eye Coordination for Robotic Grasping," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 2648-2655, April 2021, doi: 10.1109/LRA.2021.3055843.

[3] L. H. Chen, J. H. Wang, and M. T. Hsieh, "Real-Time Hand Gesture Recognition for Human-Computer Interaction," IEEE Transactions on Multimedia, vol. 23, pp. 226-235, Jan. 2021, doi: 10.1109/TMM.2020.2994535.

[4] C. Li and M. Fu, "Real-Time Hand Gesture Detection and Recognition Based on Deep Learning," 2020 IEEE International Conference on Visual

Communications and Image Processing (VCIP), Macau, China, 2020, pp. 1-4, doi: 10.1109/VCIP49819.2020.9301843.

[5] H. R. Lee, J. Park, and Y.-J. Suh, "Improving Classification Accuracy of Hand Gesture Recognition Based on 60 GHz FMCW Radar with Deep Learning Domain Adaptation," Electronics, vol. 9, no. 12, p. 2140, Dec. 2020. DOI: 10.3390/electronics9122140.

[6] B. P. S. Ahluwalia y R. Wason, "Gestural Interface Interaction: A Methodical Review," International Journal of Computer Applications, vol. 60, no. 1, pp. 21, Dec. 2012.

[7] C. Peng, L. Cao, J. T. Hansberger, and V. A. Shanthakumar, "Hand gesture controls for image categorization in immersive virtual environments," 2017 IEEE Virtual Reality (VR), Los Angeles, CA, USA, 2017, pp. 18-22, doi: 10.1109/VR.2017.7892237.

[8] F. W. Simor, M. R. Brum, J. D. E. Schmidt, R. Rieder, and A. C. B. De Marchi, "Usability evaluation methods for gesture-based games: A systematic review," JMIR Serious Games, vol. 4, no. 2, p. e17, 2016, doi: 10.2196/games.5860.

[9] L. Chen, F. Wang, H. Deng and K. Ji, "A Survey on Hand Gesture Recognition," 2013 International Conference on Computer Sciences and Applications, Wuhan, China, 2013, pp. 313-316, doi: 10.1109/CSA.2013.79. keywords: {Gesture recognition;Computers;Thumb;Cameras;Human computer interaction;Robots;Human-Computer Interaction (HCI);Hand Gesture Recognition;Kinect},

[10] A. S. Mohamed, N. F. Hassan, and A. S. Jamil, "Real-Time Hand Gesture Recognition: A Comprehensive Review of Techniques, Applications, and Challenges," Cybern. Inf. Technol., vol. 24, no. 3, pp. 163–181, Sep. 2024, doi: 10.2478/cait-2024-0031

[11] B. J. Jo, S.-K. Kim, and S. Kim, "Enhancing Virtual and Augmented Reality Interactions with a MediaPipe-Based Hand Gesture Recognition User Interface," Ingénierie des Systèmes d'Information, vol. 28, no. 3, pp. 311–318, 2023, doi: 10.18280/isi.280311.

[12] C. Li, Q. Wu, and S. Gao, "Deep learning models for real-time gesture recognition in interactive applications," Pattern Recognition, vol. 131, pp. 108–114, 2023

[13] J. Pirker, M. Pojer, A. Holzinger, and C. Gütl, "Gesture-Based Interactions in Video Games with the Leap Motion Controller," in Human-Computer Interaction. User Interface Design, Development and Multimodality (HCI 2017), Lecture Notes in Computer Science, vol. 10271, Springer, pp. 620–633, May 2017.

[14] M. L. Amit, A. C. Fajardo and R. P. Medina, "Recognition of Real-Time Hand Gestures using Mediapipe Holistic Model and LSTM with MLP Architecture," 2022 IEEE 10th Conference on Systems, Process & Control (ICSPC), Malacca, Malaysia, 2022, pp. 292-295, doi: 10.1109/ICSPC55597.2022.10001800.

[15] Y. Zhu and B. Yuan, "Real-time hand gesture recognition with Kinect for playing racing video games," 2014 International Joint Conference on Neural Networks (IJCNN), Beijing, China, 2014, pp. 3240-3246, doi: 10.1109/IJCNN.2014.6889481.

[16] J. Xu, H. Wang, J. Zhang, and L. Cai, "Robust Hand Gesture Recognition Based on RGB-D Data for Natural Human–Computer Interaction," IEEE Access, vol. 10, pp. 46123-46133, 2022, doi: 10.1109/ACCESS.2022.3176717.

[17] S. S. Rautaray and A. Agrawal, "Real-Time Hand Gesture Recognition System for Dynamic Applications," International Journal of UbiComp (IJU), vol. 3, no. 1, pp. 21–31, Jan. 2012. doi: 10.5121/iju.2012.3103.

[18] A. S. Khalaf, S. A. Alharthi, I. Dolgov, and P. O. Toups Dugas, "A Comparative Study of Hand Gesture Recognition Devices in the Context of Game Design," Proceedings of the 2019 ACM International Conference on Interactive Surfaces and Spaces, Daejeon, Republic of Korea, 2019, pp. 397-402, doi: 10.1145/3343055.3360758.

[19] J. Liu and M. Kavakli, "A survey of speech-hand gesture recognition for the development of multimodal interfaces in computer games," 2010 IEEE International Conference on Multimedia and Expo, Singapore, 2010, pp. 1564-1569, doi: 10.1109/ICME.2010.5583252.

[20] J. Warchocki, M. Vlasenko, and Y. B. Eisma, "GRLib: An Open-Source Hand Gesture Detection and Recognition Python Library," arXiv preprint arXiv:2310.12476, 2023, [Online]. Available: https://arxiv.org/abs/2310.12476.

[21] Y. Li, J. Huang, F. Tian, H.-A. Wang, and G.-Z. Dai, "Gesture interaction in virtual reality," Virtual Reality & Intelligent Hardware, vol. 1, no. 1, pp. 84-112, 2019, doi: 10.3724/SP.J.2096-5796.2018.0006.

[22] K. Kondo, G. Mizuno, and Y. Nakamura, "Feedback Control Model of a Gesture-Based Pointing Interface for a Large Display," IEICE Transactions on Information and Systems, vol. E101-D, no. 7, pp. 1894-1905, Jul. 2018, doi: 10.1587/transinf.2017EDP7298.

[23] S. Spanogianopoulos, K. Sirlantzis, M. Mentzelopoulos and A. Protopsaltis, "Human computer interaction using gestures for mobile devices and serious games: A review," 2014 International Conference on Interactive Mobile Communication Technologies and Learning (IMCL2014), Thessaloniki, Greece, 2014, pp. 310-314, doi: 10.1109/IMCTL.2014.7011154.

[24] D. Avola, L. Cinque, A. Fagioli, G. L. Foresti, A. Fragomeni, and D. Pannone, "3D hand pose and shape estimation from RGB images for keypoint-based hand gesture recognition," Pattern Recognition, vol. 129, p. 108762, 2022, doi: 10.1016/j.patcog.2022.108762.

[25] O. Köpüklü, A. Gunduz, N. Kose and G. Rigoll, "Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks," 2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019), Lille, France, 2019, pp. 1-8, doi: 10.1109/FG.2019.8756576.

[26] Y. Yaseen, O.-J. Kwon, J. Kim, F. Ullah, J. Lee, and S. Jamil, "Comparative Analysis of Hand Gesture Datasets for Drone Control Using Mediapipe," SSRN Electronic Journal, pp. 1–24, Jun. 2024. DOI: 10.2139/ssrn.12345678.

[27] J.-O. Kim, M. Kim, and K.-H. Yoo, "Real-Time Hand Gesture-Based Interaction with Objects in 3D Virtual Environments," in Proceedings of the Digital Informatics and Convergence Symposium, Chungbuk National University, South Korea, pp. 1–7, 2024.

[28] D. Bachmann, F. Weichert, and G. Rinkenauer, "Review of Three-Dimensional Human-Computer Interaction with Focus on the Leap Motion Controller," Sensors, vol. 18, no. 7, p. 2194, Jul. 2018. DOI: 10.3390/s18072194.

[29] S. S. Rautaray and A. Agrawal, "Interaction with virtual game through hand gesture recognition," 2011 International Conference on Multimedia, Signal Processing and Communication Technologies, Aligarh, India, 2011, pp. 244-247, doi: 10.1109/MSPCT.2011.6150485.

[30] A. Safa et al., "Improving the Accuracy of Spiking Neural Networks for Radar Gesture Recognition Through Preprocessing," in IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 6, pp. 2869-2881, June 2023, doi: 10.1109/TNNLS.2021.3109958.

[31] S. D. Bharatula, U. R. Vadhegar and M. Maiti, "GestureFlow: A Novel Hand Gesture Control System for Interactive Gaming," 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1-6, doi: 10.1109/ICCCNT61001.2024.10724912.

[32] A. Gupta, N. Chawla, R. Jain, N. Thakur, and A. Devi, "Gesture-Based Touchless Operations: Leveraging MediaPipe and OpenCV," NEU Journal for Artificial Intelligence and Internet of Things, vol. 1, no. 2, pp. 1-10, Oct. 2023.

[33] P. S, G. Deena, H. D, A. K B, and H. S, "Gaming using different hand gestures using artificial neural network", EAI Endorsed Trans IoT, vol. 10, Feb. 2024.

[34] M. R. Islam, R. Rahman, A. Ahmed, and R. Jany, "NFS: A Hand Gesture Recognition Based Game Using MediaPipe and PyGame," Islamic University of Technology, Gazipur, Dhaka, Bangladesh, 2022.

[35] U. Patel, S. Rupani, V. Saini and X. Tan, "Gesture Recognition Using MediaPipe for Online Realtime Gameplay," 2022 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), Niagara Falls, ON, Canada, 2022, pp. 223-229, doi: 10.1109/WI-IAT55865.2022.00039.

[36] A. Sharma, Simran, L. Verma, H. Kaur, A. Modgil and A. Soni, "Hand Gesture Recognition Gaming Control System: Harnessing Hand Gestures and Voice Commands for Immersive Gameplay," 2024 International Conference on Emerging Innovations and Advanced Computing (INNOCOMP), Sonipat, India, 2024, pp. 101-107, doi: 10.1109/INNOCOMP63224.2024.00026.

[37] E. Sophiya and S. S. Reddy, "Hand Gesture-Driven Gaming for Effective Rehabilitation and Improved Quality of Life - A Review," 2024 5th International Conference on Innovative Trends in Information Technology (ICITIIT), Kottayam, India, 2024, pp. 1-6, doi: 10.1109/ICITIIT61487.2024.10580667.

[38] S. Metkar, J. Mahajan, J. Adsul and B. Chavan, "Human body gesture-controlled gaming application," 2022 Second International Conference on Next Generation Intelligent Systems (ICNGIS), Kottayam, India, 2022, pp. 1-6, doi: 10.1109/ICNGIS54955.2022.10079850.

[39] D. Jatain, S. Singh, N. Jatana, G. Sharma, V. Garg and M. Niranjanamurthy, "A Real-Time Camera-based Motion Sensing Game Tool for Cervical Rehabilitation," 2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS), Chikkaballapur, India, 2024, pp. 1-8, doi: 10.1109/ICKECS61492.2024.10617271.

[40] C. Yeh, W. -C. Shen, C. -W. Ma, Q. -T. Yeh, C. -W. Kuo and J. -S. Chen, "Real-time Human Movement Recognition and Interaction in Virtual Fitness using Image Recognition and Motion Analysis," 2023 12th International Conference on Awareness Science and Technology (iCAST), Taichung, Taiwan, 2023, pp. 242-246, doi: 10.1109/iCAST57874.2023.10359266.