iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** XI **Month of publication:** November 2026

DOI: https://doi.org/10.22214/ijraset.2026.85917

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

Gesture Flow: Vision-Based Game Control

Mr. Nayan Gour[1], Dr. Sarvesh Warjurkar[2]

[1] *Student, Department of Computer Engineering, G H Raisoni College of Engineering &* Management, Nagpur, *Maharashtra, India*

[2]*Assistant Professor, Department of Computer Engineering, G H Raisoni College of Engineering &* Management, Nagpur, *Maharashtra, India*

Abstract: ***This paper presents Gesture Flow, a webcam-based hand gesture recognition system developed to control a computer game without using a traditional physical controller. The system explores an alternative method of human–computer interaction by translating natural hand movements into game control commands. It uses a webcam to capture the user's hand movements, while OpenCV handles video acquisition and image processing. MediaPipe Hands is used to detect the hand and extract 21 key landmarks, which are then examined to identify the performed gesture. Based on the recognized hand position, the application generates keyboard commands through PyAutoGUI to interact with the game. The prototype is implemented using Hill Climb Racing, where an open palm is mapped to the acceleration command and a closed fist is mapped to the braking command. The system provides visual feedback to help the user understand the detected gesture and its corresponding action. This approach demonstrates how commonly available computer vision tools can be combined to develop a gesture-based gaming interface with minimal additional hardware requirements. The project also provides a foundation for exploring more natural and accessible interaction methods in computer applications. Further improvements can focus on recognizing additional gestures, handling variations in lighting and background conditions, and evaluating recognition performance through systematic testing.***

Index Terms: Hand Gesture Recognition, Human–Computer Interaction, Computer Vision, OpenCV, MediaPipe, Hand Landmark Detection, Gesture-Based Gaming, Python.

I. INTRODUCTION

Human–Computer Interaction (HCI) has evolved considerably with the development of computer vision, artificial intelligence, and interactive computing technologies. Traditional interaction methods, such as keyboards, mice, and game controllers, remain widely used because they provide reliable and familiar ways to communicate with computer systems. However, these devices require direct physical input and may not always offer the most natural form of interaction. As computing applications continue to expand into areas such as gaming, robotics, education, and assistive technologies, there is growing interest in developing alternative interaction techniques that allow users to communicate with computers through more natural movements. Hand gesture recognition is one such approach, in which the movements or positions of the human hand are interpreted as commands by a computer system [1].

Hand gestures provide a visual and intuitive way of interacting with digital applications. Unlike conventional input devices, gesture-based interfaces can interpret hand movements captured through a camera and convert them into predefined actions. Computer vision has made this approach more practical by allowing systems to identify and analyse hand positions from image and video input. Previous research has explored different techniques for gesture recognition, including sensor-based methods and camera-based approaches. These techniques have been investigated for applications such as sign language interpretation, robotic control, home automation, and human–computer interaction. However, recognition performance can be influenced by factors such as lighting conditions, background complexity, hand orientation, occlusion, and variations in gesture execution [1], [2].

The gaming industry has also explored different forms of interaction beyond conventional keyboards and controllers. Motion-based interfaces have demonstrated how physical movements can be used to control actions within a virtual environment. Hand gesture recognition offers another possibility for developing interactive gaming applications, particularly where a camera is already available on the computer. A webcam-based approach can reduce the dependence on specialised motion-sensing hardware and provide an accessible way to experiment with gesture-controlled gameplay. However, developing such an interface requires more than simply detecting a hand in a video frame. The system must identify relevant hand features, distinguish between predefined gestures, and translate the recognised gestures into commands that the target application can understand.

Recent developments in real-time hand tracking have made camera-based gesture interfaces easier to implement. MediaPipe Hands provides a hand-tracking pipeline that uses a single RGB camera to detect a hand and estimate its landmark positions. The framework represents the hand using 21 landmarks, covering important points such as the wrist, finger joints, and fingertips [3].

©IJRASET: All Rights are Reserved

These landmarks provide useful spatial information for interpreting hand configurations without requiring a separate physical sensor or a manually designed image-processing pipeline for every finger position. When combined with OpenCV for video acquisition and image processing, this approach can support the development of responsive gesture-based applications using commonly available computing hardware.

This paper presents Gesture Flow, a webcam-based hand gesture recognition system developed to control a computer game through predefined hand gestures. The proposed prototype uses a webcam to capture the user's hand movements and applies OpenCV and MediaPipe Hands to detect the hand and obtain its 21 landmark coordinates. The detected hand configuration is then interpreted to identify the intended control action. In the current implementation, an open palm is assigned to the acceleration function, while a closed fist is assigned to the braking function. Python and PyAutoGUI are used to translate these recognised gestures into keyboard events corresponding to the game's default controls. Hill Climb Racing is used as the demonstration application to examine the interaction between the gesture recognition system and the game environment.

The main objective of Gesture Flow is to demonstrate a practical method of controlling a computer game through hand gestures using a standard webcam and software-based computer vision techniques. The system is organised around video capture, hand detection, landmark extraction, gesture interpretation, command generation, and visual feedback. By connecting recognised hand configurations with existing keyboard controls, the prototype provides an alternative input method without requiring the user to operate the keyboard directly during gameplay. The implementation focuses on two clearly defined gestures rather than attempting to recognise a large vocabulary of hand movements.

Although the prototype demonstrates the feasibility of gesture-based game control, practical challenges remain. Changes in illumination, background conditions, hand visibility, and the consistency of hand positioning may affect detection and interaction. In addition, the current implementation is limited to acceleration and braking and does not represent a complete gesture vocabulary for all possible game operations. The work therefore provides a foundation for further development, including additional gesture commands, improved gesture stability, testing across different environments, and more systematic evaluation of recognition accuracy and response time. Through this implementation, the study explores how readily available computer vision tools can be used to develop a simple and accessible human–computer interaction interface for gaming.

A. Motivation

The increasing use of computer applications and digital games has created a need for interaction methods that are simple, convenient, and adaptable to different users. Although keyboards and gaming controllers are commonly used to operate computer games, they require users to interact with physical input devices throughout gameplay. This dependence on conventional controls motivated the exploration of an alternative approach in which basic game operations could be performed through natural hand movements.

One of the main motivations behind this work was to investigate whether an ordinary webcam could be used as an input device for controlling a computer game. Most computers already support camera-based video input, making webcam-based interaction a practical area to explore without depending on additional motion-sensing equipment. By combining computer vision techniques with hand landmark detection, it becomes possible to examine hand positions and associate specific gestures with predefined computer commands.

Another motivation was to make game interaction more intuitive by connecting familiar hand configurations with simple gameplay actions. Instead of continuously pressing keyboard keys, a user could perform an open-palm gesture to accelerate and a closed-fist gesture to apply the brake. This approach formed the basis of Gesture Flow, with Hill Climb Racing selected as the application for demonstrating the proposed interaction method.

The project was also motivated by the learning and implementation opportunities offered by real-time computer vision. Integrating OpenCV, MediaPipe Hands, Python, and PyAutoGUI provided an opportunity to understand how video acquisition, hand tracking, landmark extraction, gesture interpretation, and command execution can work together in a single application. It also helped explore practical challenges such as maintaining hand visibility, recognising gestures consistently, and providing feedback during interaction.

Furthermore, gesture-based control may offer an alternative interaction option for users who find conventional input devices inconvenient in certain situations. However, the suitability of the system for users with physical limitations would require separate accessibility testing and evaluation.

Therefore, the central motivation of this project was to develop and demonstrate a simple webcam-based gaming interface that converts predefined hand gestures into real-time game commands. The work serves as a practical exploration of gesture-based human–computer interaction and provides scope for future improvements in gesture recognition, interaction stability, and support for additional commands.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

II. LITERATURE REVIEW

Hand gesture recognition has been studied as an alternative method of human–computer interaction, allowing users to communicate with computer systems through physical hand movements. Existing research has explored different approaches for detecting and interpreting gestures, broadly including sensor-based techniques and vision-based techniques. Sensor-based systems commonly use devices such as data gloves equipped with sensors to measure finger bending, hand orientation, or movement. Although these systems can provide direct information about hand position and finger movement, they require additional hardware and may restrict the user's natural movement. In contrast, vision-based systems use cameras to capture hand movements and process the resulting images to identify gestures. This approach offers the possibility of contact-free interaction using commonly available cameras.

Oudah, Al-Naji, and Chahl [1] presented a comprehensive review of computer vision techniques used for hand gesture recognition. Their work examined several approaches, including skin-colour-based detection, appearance-based methods, motion analysis, skeleton-based techniques, depth information, three-dimensional modelling, and deep-learning-based recognition. The study discussed the advantages and limitations of these techniques in different application environments. It highlighted that camera-based recognition can reduce dependence on wearable sensors, while also identifying challenges related to illumination, background complexity, hand orientation, and camera distance. This review provides a foundation for selecting vision-based techniques for applications such as Gesture Flow, where hand configurations are interpreted as computer input.

Pisharady and Saerbeck [2] reviewed methods and databases used in vision-based hand gesture recognition. Their study considered recognition techniques involving conventional RGB cameras and RGB-D sensors, with particular attention to static hand postures and dynamic gestures. Static gestures are identified from a particular hand configuration, whereas dynamic gestures are interpreted through movement or changes across consecutive frames. The distinction is important when designing gesture-controlled applications because the type of gesture influences the recognition method and processing requirements. The authors also discussed the role of datasets in developing and evaluating recognition systems. Their findings indicate that the selection of a recognition approach should depend on gesture complexity, available hardware, and the processing requirements of the intended application.

Zhang et al. [3] introduced MediaPipe Hands, a real-time hand-tracking framework designed to estimate hand landmarks from camera images. The approach uses a hand detection stage followed by landmark estimation to identify important points on the hand, including the wrist, finger joints, and fingertips. The framework provides 21 hand landmarks, allowing applications to analyse finger positions and hand configurations. Its focus on efficient tracking makes it useful for interactive systems that require continuous visual input. An important aspect of this work is that hand tracking and gesture interpretation are separate tasks: the framework provides landmark information, while the application developer defines the rules for identifying specific gestures and connecting them to actions. This distinction is directly relevant to Gesture Flow, which uses MediaPipe Hands for landmark extraction and application-level logic for identifying the required game controls.

Research on vision-based gesture interpretation has also highlighted the difficulties of maintaining reliable recognition under different operating conditions. Factors such as partial hand visibility, overlapping fingers, complex backgrounds, variations in illumination, and changes in hand orientation can affect the information available to a recognition system. In addition, a camera captures two-dimensional image information, which can make the interpretation of certain three-dimensional hand configurations more challenging. These issues are important in real-time applications because an incorrect gesture classification or delayed command may affect the user's interaction with the application. Therefore, recognition performance must be considered alongside tracking stability and processing requirements. The reviewed literature establishes that camera-based hand tracking provides a suitable foundation for developing gesture-driven interfaces. However, a hand-tracking framework alone does not determine how a particular application should respond to a detected hand configuration. Additional application-specific logic is required to interpret the extracted landmarks, define gesture conditions, and generate the corresponding control commands. In gaming applications, this mapping must also be sufficiently responsive to support interaction during gameplay. Based on these observations, Gesture Flow applies vision-based hand tracking to a practical computer gaming environment. The proposed implementation combines OpenCV for webcam acquisition and image processing with MediaPipe Hands for hand detection and landmark extraction. The extracted landmark information is used to identify two predefined hand configurations: an open palm for acceleration and a closed fist for braking. These gestures are then converted into keyboard commands using PyAutoGUI to control Hill Climb Racing.

The main research gap addressed by this work is the application-level integration of real-time hand tracking, predefined gesture interpretation, and game command execution using a standard webcam. Rather than proposing a new hand-tracking model, the study demonstrates how existing computer vision tools can be combined to create a functional gesture-based gaming interface. The work also identifies the need for further evaluation under different lighting conditions, camera positions, and hand orientations to establish the reliability and responsiveness of the implemented system.

©IJRASET: All Rights are Reserved

III. SYSTEM ANALYSIS

The development of Gesture Flow began with the idea of making computer-based gaming more interactive by allowing users to control a game through simple hand movements. Traditional gaming systems generally depend on physical input devices such as keyboards, mice, or dedicated controllers. Although these devices provide reliable control, they require direct physical interaction. This project explores a different approach in which hand gestures captured through a webcam are interpreted as commands and used to control a computer game.

The main focus of the system is to recognize specific hand gestures and translate them into corresponding game actions with minimal delay. To achieve this, the application uses computer vision techniques to process live video captured by a webcam. Computer vision enables a system to obtain useful information from images and video frames. In Gesture Flow, this capability is applied to hand detection and landmark extraction rather than general object classification.

The system uses OpenCV to access the webcam and process incoming video frames. MediaPipe Hands is integrated to detect the user's hand and identify its 21 landmark points. These landmarks provide information about the position and arrangement of different parts of the hand, making it possible to distinguish between the gestures supported by the application. The detected hand configuration is then examined by the gesture-processing logic to determine which action should be performed.

For the current implementation, two hand gestures are used to control the game. An open palm is interpreted as an acceleration command, while a closed fist is interpreted as a braking command. Once a gesture is identified, PyAutoGUI generates the corresponding keyboard input, allowing the user to interact with Hill Climb Racing without continuously using a physical keyboard. Visual feedback is also provided so that the user can understand which gesture has been detected.

Human–Computer Interaction (HCI) plays an important role in this project because it focuses on how people communicate with and operate computer systems. Gesture Flow applies this concept by using familiar hand movements as an alternative input method. Instead of requiring the user to press specific keys, the application attempts to make game control more direct and interactive.

The overall objective is to demonstrate how commonly available technologies, including a webcam, OpenCV, MediaPipe, and Python, can be combined to develop a practical gesture-controlled gaming application. The system is designed as a working prototype and is limited to the two implemented gestures. Its performance can be influenced by factors such as lighting, camera quality, hand positioning, and background conditions. These factors also provide opportunities for further improvement, including support for additional gestures, better handling of environmental variations, and more detailed performance evaluation.

A. Problem Description

Most computer games rely on traditional input devices such as keyboards, mice, and gaming controllers for user interaction. Although these devices provide a familiar way to control games, they require physical contact and predefined input actions. This creates an opportunity to explore alternative interaction methods that allow users to communicate with a computer through more natural movements. The main problem addressed in this project is the development of a system that can recognize hand gestures through a standard webcam and convert them into corresponding game-control commands. The challenge is not limited to detecting a hand in a video frame; the system must also identify its position, interpret the gesture correctly, and generate the appropriate input without introducing noticeable delays or frequent incorrect commands. Another important challenge is maintaining consistent gesture recognition under different environmental conditions. Changes in lighting, background appearance, camera position, and hand orientation may affect the quality of captured frames and the accuracy of hand detection. Therefore, a suitable approach is required to process webcam input and identify the intended gestures reliably during gameplay. Gesture Flow addresses this problem by combining OpenCV for video acquisition and image processing with MediaPipe Hands for hand detection and landmark extraction. The recognized hand configurations are used to distinguish between two predefined gestures: an open palm for acceleration and a closed fist for braking. These gestures are then translated into keyboard commands using PyAutoGUI to control Hill Climb Racing.

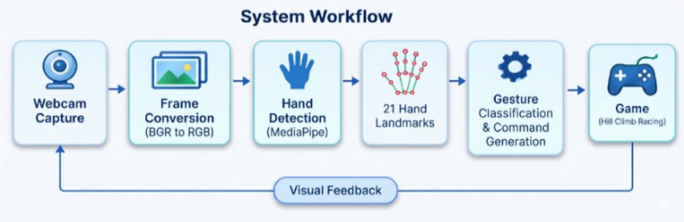


Fig -1: System workflow

A. Issues & Challenges

Developing a reliable webcam-based gesture control system involves several practical difficulties. First, environmental factors can significantly affect hand detection and tracking. The system's accuracy depends on the quality of the video input, which can be degraded by poor lighting or visual clutter. Inconsistent or harsh lighting (e.g. strong backlight or low-light conditions) can change the appearance of the hand or create shadows that confuse the detector. Likewise, a busy background can introduce colors and shapes that distract the vision model. In MediaPipe, for example, rapid changes in illumination or odd viewing angles can cause fingers to be missed or the hand to be undetected. In practice, if part of the hand moves outside the camera view or overlaps with another object, MediaPipe's palm detector may fail completely, causing the landmark estimation to break down. These issues mean that maintaining adequate lighting and clear visibility of the hand is crucial for consistent operation.

Second, interpreting the hand configuration robustly is challenging even with 21 hand landmarks available. Gesture Flow currently distinguishes only two static gestures (an open palm vs. a closed fist), but transitions between these gestures can be ambiguous. For example, a partially closed fist or a hand that is not fully visible may not clearly satisfy either gesture criterion. The system's gesture-recognition logic must therefore use thresholds on the landmark positions to decide when to issue a command. No matter how well-tuned, some natural hand motions might hover between the defined states, causing misclassifications or jitter. This emphasizes the need for designing clear decision rules and possibly smoothing or timing thresholds to prevent false triggers during ambiguous hand poses.

Another key challenge is meeting real-time performance requirements. Gesture Flow must capture webcam frames, convert color format (BGR to RGB), detect the hand, extract landmarks, interpret the gesture, and send a keyboard event all within a fraction of a second for each frame. If the processing pipeline is too slow, the interface will feel laggy or unresponsive. For instance, running MediaPipe's two-stage detector on high-resolution frames can introduce latency. As the MediaPipe developers note, there is an inherent trade-off between processing speed and tracking fidelity: reducing frame resolution or simplifying detection can improve speed at the cost of robustness. In our system, any delay between the user's motion and the game response would degrade the gameplay experience. Thus, efficient coding and possibly lower-resolution frames are used to keep the command latency low, but the exact performance depends heavily on hardware (CPU/GPU) capability and available memory.

Generating reliable input to the game also has its own concerns. The application uses Python's PyAutoGUI library to simulate keyboard presses (e.g. pressing the "gas" or "brake" key). However, this method injects synthetic events into the Windows input queue, and many game engines – especially full-screen or DirectX-based games – may ignore such injected inputs as a security or design measure. In Windows, the Send Input function marks events with a special "LLMHF_INJECTED" flag, which can be recognized by some applications as non-human input. If the game window is not correctly focused or does not accept injected events, the simulated keystrokes might not register. In practice, ensuring that the game receives the keystrokes requires that it be running in a mode compatible with PyAutoGUI, or alternatively using a different input method (for example, the Py DirectInput library is often recommended for games to avoid this issue). In any case, this introduces a reliability concern: the system must guarantee that each recognized gesture results in a corresponding game action, which may require additional handling to maintain the correct window focus.

Finally, there are compatibility and system-level issues. The implementation depends on specific software libraries (Python, OpenCV, Media Pipe, NumPy, PyAutoGUI) and expects them to work together on the chosen platform (e.g. Windows 10/11). Differences in operating systems or library versions could affect everything from camera access to gesture detection accuracy. Likewise, the performance varies with hardware: a low-end CPU or an old integrated GPU may struggle with real-time video processing. The system must also manage resources properly (e.g. releasing the webcam on exit) to avoid crashes. An additional limitation is the small gesture vocabulary: only two gestures are currently mapped. This suffices to demonstrate the concept, but in a real application more commands would be needed. Extending the gesture set would introduce new classification challenges and risks of confusion between gestures.

Overall, these issues highlight that creating a robust gesture-controlled interface involves careful engineering of the recognition pipeline, environment considerations, and integration with the game. While the prototype works under controlled conditions, achieving consistent accuracy and low latency in everyday use remains a challenge that requires further testing and refinement.

B. Proposed System

The proposed system, Gesture Flow, is designed to control a computer game through hand gestures captured using a standard webcam. It provides an alternative way of interacting with a game by interpreting hand movements and converting them into keyboard commands. The system is developed using Python and combines OpenCV, MediaPipe Hands and PyAutoGUI to perform

video capture, hand detection, gesture recognition and game control. The overall working process is divided into five main components: webcam input and frame acquisition, hand detection and landmark extraction, hand gesture recognition, gesture-to-command mapping, and game control with visual feedback.

C. *Webcam Input and Frame Acquisition*

The webcam acts as the primary input device of the proposed system. It continuously captures video frames of the user's hand while the application is running. OpenCV is used to access the webcam and process the incoming video stream. Each captured frame is initially received in BGR colour format and converted into RGB format before being passed to MediaPipe Hands for further processing. This step prepares the image data for hand detection and allows the system to examine the user's hand movements continuously.

D. *Hand Detection and Landmark Extraction*

After frame acquisition, the processed image is passed to the MediaPipe Hands framework to detect the user's hand. Once a hand is detected, the framework identifies 21 landmark points representing important positions on the hand, including the wrist, finger joints and fingertips. These landmarks provide information about the hand's structure and the relative positions of its fingers. The extracted landmark information is used by the gesture recognition component to examine the hand configuration in each frame. This approach allows the system to use the detected hand points for gesture interpretation without requiring a separately trained hand recognition model.

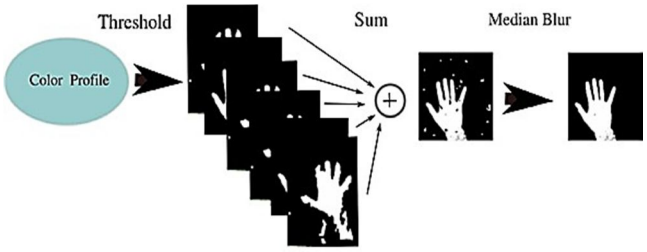


Fig -2: Hand Tracking

E. *Hand Gesture Recognition*

Hand gesture recognition is the main decision-making stage of the proposed system. In this stage, the positions of the detected landmarks are examined to identify the hand gesture performed by the user. The implemented prototype recognises two basic hand configurations. An open palm is interpreted as the acceleration gesture, whereas a closed fist is interpreted as the braking gesture. The recognised hand configuration is evaluated continuously so that the application can respond to changes in the user's gesture. If the detected hand configuration does not match the supported gestures, it can be treated as an unknown gesture rather than being assigned an incorrect control command.

F. *Gesture-to-Command Mapping*

Once a gesture has been recognised, it must be converted into an appropriate command for the game. The gesture-to-command mapping component connects each supported hand configuration with its corresponding keyboard input. In the proposed implementation, the open palm gesture is mapped to the GAS command, while the closed fist gesture is mapped to the BRAKE command. PyAutoGUI is used to generate the corresponding keyboard inputs. In this way, the user's hand gestures are translated into commands that can be understood by the game.

G. *Game Control and Visual Feedback*

The final component is responsible for applying the generated commands to the game. The proposed system uses Hill Climb Racing as the application for demonstrating gesture-based control. When the user performs an open palm gesture, the system sends the acceleration input, and when a closed fist is detected, it sends the braking input. Along with game control, the application displays visual feedback by showing the detected hand landmarks and the recognised gesture on the webcam feed. This feedback helps the user understand how the system is interpreting the hand configuration. The processing continues while the application is active and can be stopped by pressing the designated exit key.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Through the integration of these components, Gesture Flow demonstrates how webcam-based hand detection and gesture recognition can be used to provide an alternative input method for computer gaming. The proposed approach relies on commonly available hardware and existing computer vision libraries, making the prototype practical to implement without specialised gesture-sensing equipment.

IV. SYSTEM DESIGN

The system design of Gesture Flow describes how the different software components work together to convert hand gestures into commands for controlling a computer game. The proposed system follows a modular design in which each component performs a specific task, beginning with video capture and ending with the execution of the corresponding game action. The design uses a standard webcam as the input device and combines OpenCV, MediaPipe Hands and PyAutoGUI to establish communication between the user's hand movements and the gaming application.

The overall design focuses on keeping the processing sequence simple and suitable for continuous operation. Instead of using separate hardware sensors or wearable devices, the system obtains visual information directly from the webcam. MediaPipe Hands is used to detect the hand and estimate its 21 landmark points, which provide the information required for interpreting the hand configuration [1]. The recognised gestures are then associated with predefined keyboard commands and passed to Hill Climb Racing.

A. Overall System Architecture

The architecture of Gesture Flow is organised as a sequence of connected processing modules. The process begins when the webcam captures the user's hand movements. OpenCV receives the video frames and prepares them for analysis. The processed frames are then supplied to MediaPipe Hands, which identifies the hand and returns its landmark information. Based on the detected hand configuration, the gesture recognition logic determines whether the user is showing an open palm or a closed fist.

After recognition, the command mapping component associates the detected gesture with its corresponding game action. PyAutoGUI generates the required keyboard input, allowing the game to respond to the user's hand gesture. The system also displays the camera feed with hand landmarks and the recognised gesture to provide visual feedback during operation.

The complete processing sequence can be represented as:

Webcam Input → Frame Acquisition and Preprocessing → Hand Detection and Landmark Extraction → Gesture Recognition → Command Mapping → Game Control

B. Webcam Input and Image Processing

The webcam is the primary source of visual input for the system. It captures the user's hand movements continuously and supplies the resulting frames to the application. OpenCV is responsible for accessing the camera, reading the video frames and handling the basic image-processing operations. Before hand detection takes place, the captured frame is converted from BGR colour format to RGB format. This conversion is required to prepare the image for processing by MediaPipe Hands. The prepared frame is then passed to the hand detection module. Since the application processes incoming frames repeatedly, this stage forms the input foundation for the complete gesture recognition process.

C. Hand Detection and Landmark Extraction

The next stage is responsible for locating the user's hand in the captured image. Gesture Flow uses MediaPipe Hands to perform hand detection and landmark estimation. The framework identifies 21 key points on the hand, covering the wrist, palm region, finger joints and fingertips. These points provide a structured representation of the hand and help describe the arrangement of the fingers. The landmark coordinates obtained from the detected hand are supplied to the gesture recognition logic. Rather than analysing the entire image manually, the system uses these landmark positions to examine the hand configuration. This makes it possible to distinguish the supported gestures using the relative positions of the detected hand points. The use of MediaPipe Hands also avoids the need to develop and train a separate hand detection model for this project.

D. Gesture Recognition and Decision Logic

Gesture recognition is the central processing stage of the proposed design. In this stage, the extracted hand landmarks are examined to determine which supported gesture is being performed. The current implementation is based on two hand configurations: an open palm and a closed fist.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

When the system identifies an open palm, the gesture is classified as an acceleration command. When the hand is recognised as a closed fist, it is classified as a braking command. The recognition process is repeated for successive video frames so that changes in the user's hand configuration can be detected during operation. If the detected configuration does not correspond to either supported gesture, it is not assigned an acceleration or braking command.

E. Gesture-to-Command Mapping

After a gesture has been identified, the system converts it into a command that can be used by the target application. This is achieved through a predefined mapping between the recognised hand configurations and the game's keyboard controls. The open palm gesture is associated with the GAS input, while the closed fist gesture is associated with the BRAKE input.

PyAutoGUI is used to generate the corresponding keyboard events. This component acts as the connection between the gesture recognition process and the gaming application. As a result, the user can perform the supported game actions through hand gestures rather than pressing the corresponding physical keys.

F. Game Interaction and Visual Feedback

The final stage of the system design involves interaction with Hill Climb Racing. The keyboard commands generated by PyAutoGUI are delivered to the game, where they trigger the corresponding acceleration or braking action. The game therefore receives inputs generated from the recognised hand gestures.

Visual feedback is also provided through the application window. The webcam feed displays the detected hand landmarks along with the recognised gesture label. This allows the user to observe whether the system has correctly identified the current hand configuration. The feedback is particularly useful during testing because it helps distinguish between hand detection and gesture interpretation.

G. Continuous Processing and System Termination

Gesture Flow operates through a continuous processing loop while the application is running. During each iteration, the system captures a new webcam frame, prepares the image, detects the hand, extracts the landmarks and evaluates the hand configuration. If a supported gesture is recognised, the corresponding keyboard command is generated and the visual feedback is updated.

This sequence continues until the user presses the designated exit key, 'q'. Once the application receives this input, the processing loop is stopped, the webcam resources are released and the display windows are closed. This termination procedure ensures that the camera is no longer occupied after the application has finished.

Overall, the system design establishes a clear connection between visual hand input and game control. By separating image acquisition, landmark extraction, gesture recognition and command generation into individual modules, Gesture Flow provides a structured approach to webcam-based interaction. The design also allows the gesture recognition logic to be extended in future work without requiring major changes to the other components.

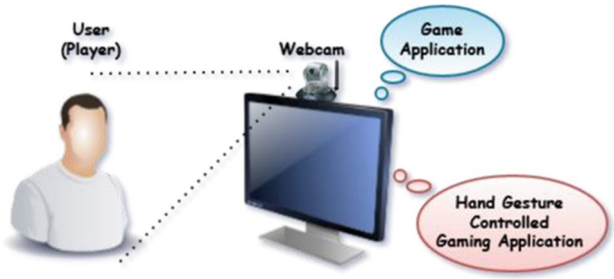


Fig -3: Input Environment

The input environment consists of only the User PC System, a properly installed Web Camera system & the Player (user) himself. Except of all this requirement, the user must have a game application (that supports keyboard game input) running alongside of the Hand Gesture Controlled Gaming application. The input to the system comprises of only the user’s hand pivot positions & then the hand movement gestures. The rest of the entire system works in the background silently providing the user an endless enjoyable & addictive gaming experience.

The input to the system is the HCI Input i.e. the user's hand movement gesture. The input to the system is provided through a Webcam that captures continuous frames, creating a video, that tracks the center of the hand which moves away making a gesture. This same video frame is also displayed on the application UI. The webcam continuously captures frames, through which the motion of the user's hand gesture is tracked, & processed to map it to the keyboard cursor input for the alongside running game application.

V. CORE TASK

The main task of Gesture Flow is to recognise hand gestures through a webcam and use them to control a computer game. Instead of relying on a keyboard for every action, the system allows the user to perform selected actions by changing the position of their hand. The current implementation works with two gestures: an open palm and a closed fist. These are assigned to the acceleration and braking functions in the selected game.

The operation begins when the webcam captures the user's hand movements. OpenCV is used to access the camera and read the incoming video frames. Before further processing, each frame is converted from BGR colour format to RGB format so that it can be passed to the hand detection module. The frames are processed continuously, allowing the application to observe changes in the user's hand position while the program is running.

After receiving the image, MediaPipe Hands detects the visible hand and identifies 21 landmark points. These points represent different parts of the hand, including the wrist, finger joints and fingertips. Their positions provide information about the arrangement of the fingers and the overall hand shape. The system uses this information to examine the hand configuration rather than relying only on the presence of a hand in the camera frame.

The extracted landmark information is then used by the gesture recognition logic developed in Python. The application checks the hand configuration against the two gestures supported by the prototype. An open palm is recognised as the GAS command, whereas a closed fist is identified as the BRAKE command. If the hand is not detected or its configuration does not match either of these gestures, the system does not assign an acceleration or braking action. This prevents unsupported hand positions from being treated as valid control commands.

Once a gesture has been identified, the system converts it into the corresponding keyboard input. PyAutoGUI is used for this purpose, allowing the recognised gesture to interact with Hill Climb Racing through its existing keyboard controls. When the user shows an open palm, the application generates the acceleration input. Similarly, closing the hand into a fist generates the braking input. In this way, the movement performed in front of the webcam is translated into an action within the game.

The recognition and control process continues as new frames arrive from the webcam. Along with processing the hand movements, the application displays the detected landmarks and the recognised gesture on the screen. This visual information helps the user observe how the program interprets the hand configuration and makes it easier to identify recognition errors during testing.

The application remains active until the user presses the 'q' key. Once this key is detected, the program exits the processing loop, releases the webcam and closes the display window. The complete process therefore connects three main activities: capturing the hand movement, identifying the intended gesture and generating the corresponding game input. Through this implementation, Gesture Flow demonstrates a simple webcam-based method of controlling a computer game using two predefined hand gestures without requiring any additional gesture-sensing hardware.

VI. RESULTS & DISCUSSIONS

The developed Gesture Flow system was tested to examine whether hand gestures could be used to control a computer game through a standard webcam. The prototype successfully demonstrated the recognition of two predefined hand gestures: an open palm for acceleration and a closed fist for braking. The detected hand landmarks and recognised gesture were displayed on the screen, allowing the user to observe the system's response during operation.

The system connected hand gesture recognition with game control using Python and PyAutoGUI. The recognised gestures were converted into the corresponding keyboard inputs and used to perform the required actions in Hill Climb Racing. This demonstrated that hand movements captured through a webcam could be translated into practical game commands without using a separate physical controller.

The results also showed that the overall working of the system depends on the quality of the webcam input and the visibility of the hand. Changes in lighting, hand position and background conditions can affect detection and recognition. Although the prototype demonstrated the intended control functions, its performance may vary under different operating conditions. Further testing with a larger number of trials and more varied environments would be required to measure recognition accuracy and response time more reliably.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

VII. CONCLUSION

Gesture Flow demonstrates a simple approach to controlling a computer game through hand gestures using a standard webcam. The system combines OpenCV for video capture, MediaPipe Hands for hand detection and landmark extraction, and PyAutoGUI for generating keyboard inputs. It recognises two predefined gestures, an open palm for acceleration and a closed fist for braking, and uses them to control Hill Climb Racing. The developed prototype shows that hand movements can be converted into game commands without requiring additional gesture-sensing hardware. During implementation, the system also provided visual feedback through displayed hand landmarks and gesture labels, making its operation easier to observe. However, the current version supports only two gestures, and its performance can be affected by lighting, hand visibility and background conditions. Further development could introduce additional gestures, improve recognition under different environments and reduce delays in command execution. More extensive testing would also help evaluate its accuracy and reliability. Overall, the project provides a practical starting point for exploring gesture-based human–computer interaction.

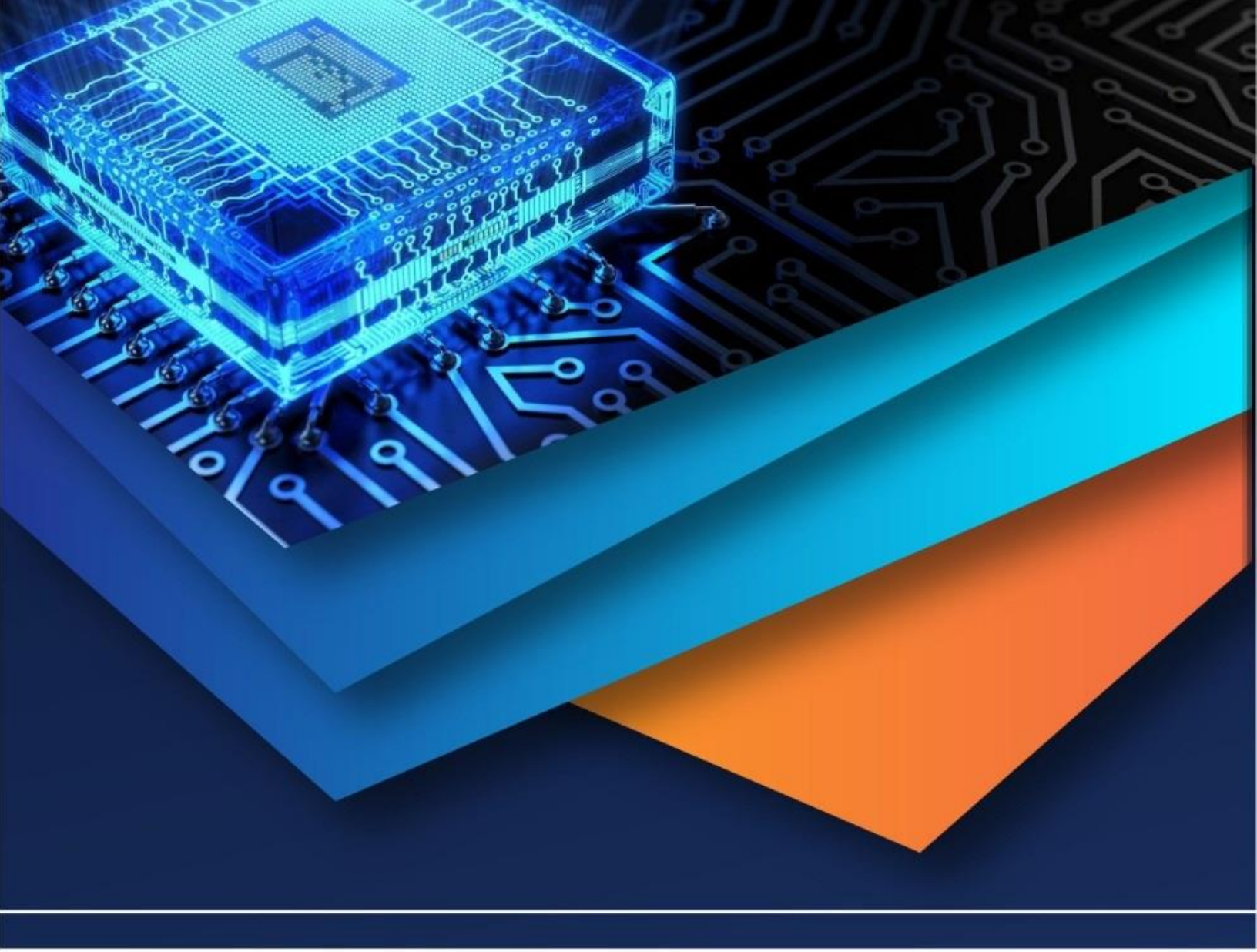
VIII. FUTURE SCOPE

Gesture Flow can be improved in several ways to make gesture-based game control more flexible and reliable. The current version recognises only an open palm and a closed fist, so additional hand gestures could be introduced to support more game actions, such as steering, pausing and menu navigation. The recognition process could also be improved to work more consistently under different lighting conditions, hand orientations and background environments. Another area for improvement is reducing the delay between performing a gesture and the corresponding game response, which would make the interaction feel more natural during gameplay. The system could further be adapted for other computer applications, including presentation control, media playback and basic desktop operations. Support for two-hand gestures could also be explored to provide a wider range of controls. In addition, testing with different users, webcams and computer configurations would help identify practical limitations and improve overall reliability. These developments could extend Gesture Flow beyond a basic gaming prototype into a more flexible gesture-based interaction system.

REFERENCES

[1] Munir Oudah, Ali Al-Naji and Javaan Chahl, “Hand Gesture Recognition Based on Computer Vision: A Review of Techniques,” Journal of Imaging, Vol. 6, Issue 8, Article 73, 2020. https://doi.org/10.3390/jimaging6080073
[2] Pramod Kumar Pisharady and Martin Saerbeck, “Recent Methods and Databases in Vision-Based Hand Gesture Recognition: A Review,” Computer Vision and Image Understanding, Vol. 141, pp. 152–165, 2015. https://doi.org/10.1016/j.cviu.2015.08.004
[3] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang and Matthias Grundmann, “MediaPipe Hands: On-Device Real-Time Hand Tracking,” 2020. https://arxiv.org/abs/2006.10214
[4] OpenCV, “Color Space Conversions – OpenCV Documentation.” https://docs.opencv.org/4.x/d8/d01/group__imgproc__color__conversions.html
[5] PyAutoGUI, “Keyboard Control Functions – PyAutoGUI Documentation.” https://pyautogui.readthedocs.io/en/latest/keyboard.html
[6] Debajit Sarma and M. K. Bhuyan, “Methods, Databases and Recent Advancement of Vision-Based Hand Gesture Recognition for HCI Systems,” SN Computer Science, Vol. 2, Article 436, 2021.
[7] Fahmid Al Farid, Noramiza Hashim, Junaidi Abdullah, Md. Roman Bhuiyan, Wan Noor Shahida Mohd Isa, Jia Uddin, Mohammad Ahsanul Haque and Mohd Nizam Husen, “A Structured and Methodological Review on Vision-Based Hand Gesture Recognition System,” Journal of Imaging, Vol. 8, Issue 6, Article 153, 2022.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)