Touchless and Go: A Feasibility Study of Implementing Gesture Control at The Operating System Level

Triet Minh Ngo
Northeastern University - Khoury College of Computer Sciences
San Jose, California, USA
ngo.tri@northeastern.edu

A Final Report Submitted in Fulfillment of the Course Requirements for
CS 8674: Master's Project supervised by Dr. Scott A. Valcourt
Khoury College of Computer Sciences
Northeastern University

***Abstract*—**Human-computer interaction (HCI) is an active research area that explores new and improved ways people can interact with computers and similar technologies. As technology steadily permeates human societies, with the latest trends and innovations incorporating various forms of artificial intelligence (AI) into existing software and hardware systems in order to better assist people as they interact with technology. Indeed, as technology becomes more sophisticated with advanced features, making technology easier to use therefore has positive accessibility implications. As part of this effort, the Master's project in this report attempts to explore the implementation of hand gesture control as part of the operating system (OS), and not as a separate application. Throughout the project, it is clear that the implementation of OS-level gesture control is theoretically possible, but due to the proprietary nature of commercial OSs combined with the complexity and inconsistencies of open-source OS, the project only succeeded in simulating gesture-based control as a C++ program on Microsoft Windows. Nonetheless, discoveries made within the duration of the project provide an opportunity to explore how computer vision can be incorporated as part of the OS and how it can elevate HCI accessibility research.

***Index Terms*—**human-computer interaction, gesture, machine learning, operating system, open-source, proprietary system, computer vision

I. Introduction

Human-computer interaction, as a discipline, focuses on how people interact with electronic computing devices and how such an interaction can be refined and/or improved. And as devices become more complex, feature-rich, and simultaneously more integral to human society, making sure technology is accessible and usable for the masses is increasingly important. Indeed, more accessibility, researchers argue, means potential increase in capacity and expertise in industry [1].

Accessible technology, in addition to assisting able-bodied people, also enables those with certain impairments, either from physical deformities or circumstances, such as those in the middle of driving, to interact productively with electronic devices. One modern solution to this accessibility issue is touchless technology which exists in many forms with examples including Microsoft Kinect, perhaps the most well-known line of motion sensing input devices, or Amazon Alexa, an extremely popular voice-controlled virtual assistant. These implementations, however, mainly exist as first- or third-party add-ons and applications, some of which require additional hardware on top of the main systems.

Given these premises, it is clear that as the electronic devices become increasingly inseparable from everyday life, the integration of accessible features, namely touchless interaction, into the existing OSs should theoretically open up new ways people can control devices without much tinkering. As such, the following paper will attempt to explore the feasibility of implementing vision-based gesture control as part of the operating system (OS), and introduce a working prototype on Microsoft Windows as a proof of concept.

II. Literature Review

A. *Gesture Control as a Concept*

Touchless interaction, or touchless user interface (TUI), refers to the act of controlling an electronic device exclusively via body movements and gestures without additional accessories such as a keyboard, mouse, or screen [2]. Representations of TUI have long been considered futuristic, and a thing of science fiction: popular sci-fi media, including Blade Runner and the Iron Man series, depicts various characters using their hands and fingers to manipulate data displayed either on a screen or through a holographic projection. In order to avoid confusion, and to abide by the popular interpretation of gesture control, it is worth noting that any mention of "gesture control" in this writing will henceforth be referred exclusively to the human-computer interaction involving vision-based hand gestures.

As computing power has skyrocketed, alongside the proliferation of artificial intelligence and machine learning (ML), attempts have been made at revisiting and implementing a reliable system for vision-based gesture control. Perhaps the most recent and notable example of this effort is the introduction of Apple Vision Pro, a mixed-reality headset that allows users to navigate using hand gestures, mainly through pinching and dragging in front of the built-in camera [3]. On the development side, Microsoft corporation offers the "Project Gesture" software development kit that allows developers to define a certain gesture and trigger predefined actions if the user's gesture is detected to be a match. In addition, computer vision libraries, most notably the Intel-made Open Source Computer Vision Library, also known as OpenCV, have also enabled independent developers and hobbyists to efficiently create gesture recognition and control applications using built in computer vision algorithms [4].

With novel ideas come the question of utility. It is trivial to observe that vision-based gesture control has usability and accessibility merits: being able to use natural hand movements to interact with the user interface can allow for a greater degree of precision and control in terms of organizing and consuming information. In addition, for people whose motor functions are not adequate, gesture control may enable these users to control devices using fewer and/or less sophisticated movements [5]. Public health concerns can be alleviated with gesture-based interaction: for example, a touchless interface can significantly reduce the number of times a screen is touched, and thus potentially limit the transmission of a particular disease [6]. Furthermore, beneficiaries of a good gesture control system are not limited to the less able-bodied, as people impaired by circumstances can use gestures to interact with information not fully within their focus. Drivers of automobiles with little to no physical buttons, as seen on newer car models that put all interactions behind a flat touchscreen, can temporarily use one hand to change the temperature, switch radio channels, or skip to the next song, etc., without ever taking their eyes off the road, and thus ensuring safety [7].

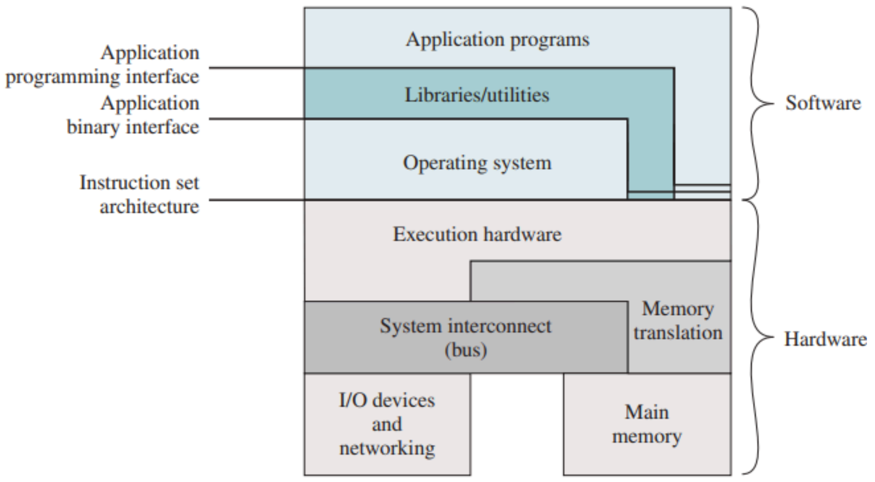
Despite these benefits, there are several limitations hindering the development of contemporary implementations of gesture control. The most prominent concern is accuracy: as gesture control is mainly vision-based, image noise plays a significant part in whether or not a recognition software can detect and interpret a gesture. Poor lighting conditions and messy backgrounds can add noise, while subpar sensors and cameras, or inadequate gesture training models may reduce the detection accuracy [8]. These factors can altogether cause detection algorithms to fail or misclassify a gesture, leading to unpredictable, and in the worst case, fatal results. Inaccuracy in the detection process can often lead to repeated gestures which, depending on the complexity of the movements, may cause fatigue and pain to the user [9]. Aside from the accuracy concerns, gesture control is susceptible to security vulnerabilities, mainly in the form of unauthorized users performing the valid gestures to potentially gain access to a system. Just as the Apple Face ID system can theoretically be bypassed by using an identical facial structure, a gesture control system, especially one where authentication is weak or non-existent, can be bypassed by another identical set of hands [10]. Since the silhouette of a hand is of a much more universal shape, barring fingerprints, this vulnerability is particularly serious.

B. *Gesture Control as Part of the Operating System*

As mentioned, most, if not all, past and current implementations of gesture control are separate applications, which can be added or removed without significantly impacting the underlying OS. Thus, in order to achieve the stated goal of making gesture control universal and reap the associated benefits, an integration of touchless interfaces into the OS layer, which all devices have, is a logical suggestion. Nonetheless, as commercial

OSs become more equipped with features, and in many cases integrating previously third-party applications—for instance, using the iPhone's camera flash as a flashlight has only been part of iOS since 2013 [11]—it is worth revisiting what it means to be at the OS layer.

As a user-computer interface, the OS consists of important system programs, which include utilities or library programs, meant to assist in program development, file management, and control of I/O devices. Essentially, as seen in Fig. 1, the OS is itself a program acting as a translation layer between the computer hardware and applications and thus providing an interface for programming and users to interact with the hardware components. An OS can also be capable of being expanded with new features and services to accommodate ever-evolving requirements [12].



Application programming interface
Application binary interface
Instruction set architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
System interconnect (bus)
Memory translation
I/O devices and networking
Main memory
Software
Hardware

Figure 1. Layers of a computer system. Retrieved from [12].

With this context, to become part of the OS layer, a gesture control program therefore needs to be an integral utility program. With the ability to leverage the hardware components, namely cameras or other sensors, at a lower level, users should be able to use this program out of the box. In simpler terms, an OS-level gesture control, for instance on Microsoft Windows, should be considered a default mode of input alongside keyboard and mouse, trackpad, and touchscreen, and included as part of the OS installation.

III. Gesture Control Project

A. What an OS-level Gesture Control Program Should Be

Having explored the current implementations of gesture control as well as what the OS layer entails, the overarching goal of the Gesture Control project is to construct a gesture recognition program that can be implemented as close to this layer as possible. In essence, the principal investigator (PI) determines that the program should be written in a lower-level language and should incorporate as few external libraries and modules as it can be feasible. With considerations for the following discoveries and time constraints, in terms of functionality, the program should allow users to interact with the computer, including selecting and accessing apps and folders, using hand gestures only.

B. Discoveries and Caveats

Nevertheless, one major component of vision-based gesture control is pattern recognition, which is not trivial to implement. While it is possible to write recognition algorithms without any abstraction, it is a massive undertaking which is not feasible given the project's timeframe, and the process is almost certainly prone to errors. Therefore, the PI made the decision to rely on the latest available version of the aforementioned OpenCV library which contains optimized computer vision functions in order to streamline the development processes related to machine perception.

Although the use of OpenCV partially defeats the purpose of an OS-layer program, its increasing prevalence in projects, companies, and government bodies around the world effectively solidifies its status as the default computer vision library [13]. Furthermore, as commercial solutions such as Microsoft Windows and Apple macOS are proprietary by nature, an OS-level program will inherently introduce low-level changes to the OS whose source code is not available publicly and whose modification privileges are not accessible to the regular user. While a case can be made for developing the program on an open-source OS family such as Linux, the PI was unable to install OpenCV due to unexpected and unexplained errors, which hindered development and, coupled with a narrow timetable, limited the development to commercial OSs instead.

With the library chosen, the next step was determining the programming language. As of this writing, major OpenCV releases contain C++, Python, Java and MATLAB interfaces. However, since the library itself is currently written natively in C++ and that C++ is among the least abstracted programming languages, the PI determined that C++ is the most appropriate language as the documentation is the most up-to-date. The language, as well as its corresponding OpenCV library, have consequently influenced the final selection of the target platform.

As part of the program involves taking control of a device's input, some OS-level functions are needed. Initially developed on macOS using Apple Xcode, the PI discovered that macOS system input functions are accessible only in Objective-C or Swift [14]. And while OpenCV does have an Objective-C release, the library is reserved for Apple iOS only [13]. Limited by time and being unfamiliar with Objective-C and iOS development, the PI decided to regrettably scrap the macOS version of the program.

C. Methodologies

Since the project is part of an exploratory exercise, the pre-prototyping stage involved writing the program across multiple systems. While the process is virtually identical regardless of platform, aforementioned OS-specific restrictions prevent generalizations. In order to provide a complete, self-contained process that may be replicated by future researchers, the following methodology for implementing a gesture control program as close to the OS as possible will be dedicated to the Microsoft Windows platform, with the development environment and hardware considerations elaborated in Appendix A. And since the program is entirely CPU-bound and relatively straightforward, it should be able to be tested on less powerful systems.

1) Hand Gesture Detection

For the program to be able to recognize hand gestures, the PI opted to utilize the computer's built-in camera, or in the case of desktop computers, a third-party webcam. Using cameras with non-specialized sensors limits the detection algorithm to the appearance-based model, which relies on the image or video feed itself to make the recognition decision [15]. With this model, the PI identified two classes of detection algorithms: color-based and machine-learning-assisted detection.

Of these algorithms, color-based detection is more straightforward: the algorithm makes detection decisions based on the pixels in an image that match a predetermined color range [16]. This method is also convenient since most human hands have uniform color, the color processing can leverage already existing RGB sensors in most cameras, and OpenCV already supports optimized color detection methods. However, as mentioned before, color-based detection is much more sensitive to lighting conditions: image noise and/or slight changes in lighting can change the color values and throw off detection [8].

On the other hand, through testing, the PI has determined that detection using a ML model is much more reliable in less-than-ideal conditions, as the computer learns from a multitude of existing images of hand gestures in various lighting conditions to make a decision. While there are many ML-assisted detection models, one method supported by OpenCV and selected by the PI is Haar-cascade detection, which works by rapidly scanning an image using a moving window of predetermined size, which is less than that of the image, to identify and aggregate positive regions while rejecting irrelevant regions that inherently make up most of any single image. The result is a relatively fast detection method with acceptable accuracy [17].

While users can use their own datasets to train a Haar-cascade classifier model using OpenCV, this functionality is disabled in the current version of OpenCV [18]. As such, for convenience and to maintain the development consistency, the PI relied on publicly available Haar-cascade classifiers from other developers, with appropriate attributions. Namely, the program utilizes the right-hand palm and fist classifiers, deemed by the PI to be adequate, from the hand gesture detection project developed by Sandeep Sthapit, as seen in Appendix B and [19].

2) *Algorithms*

The overall flow of the program is as follows: the computer, through a video feed from the webcam, detects and tracks the user's gestures using the aforementioned Haar-cascade models as shown in Fig. 2. While the tracking is active, with each frame is an interaction of the overall while-loop, a right-hand open palm is detected, the position of the detection is recorded and passed onto the helper function $cursorControl()$ which determines the direction—up, down, left, right—the selection highlight should go. Once the direction is confirmed, it is passed as a string into another helper function $pressKey()$ to execute the appropriate directional key press, leveraging functions in the Windows' $Winuser.h$ header [20]. On the other hand, if the camera detects a fist, $pressKey()$ will execute the RETURN/ESC key and run the current selection. With these combined functionalities, the user can theoretically navigate a screen using their open palm, and then close their fist to select and run the highlighted item on the desktop screen.

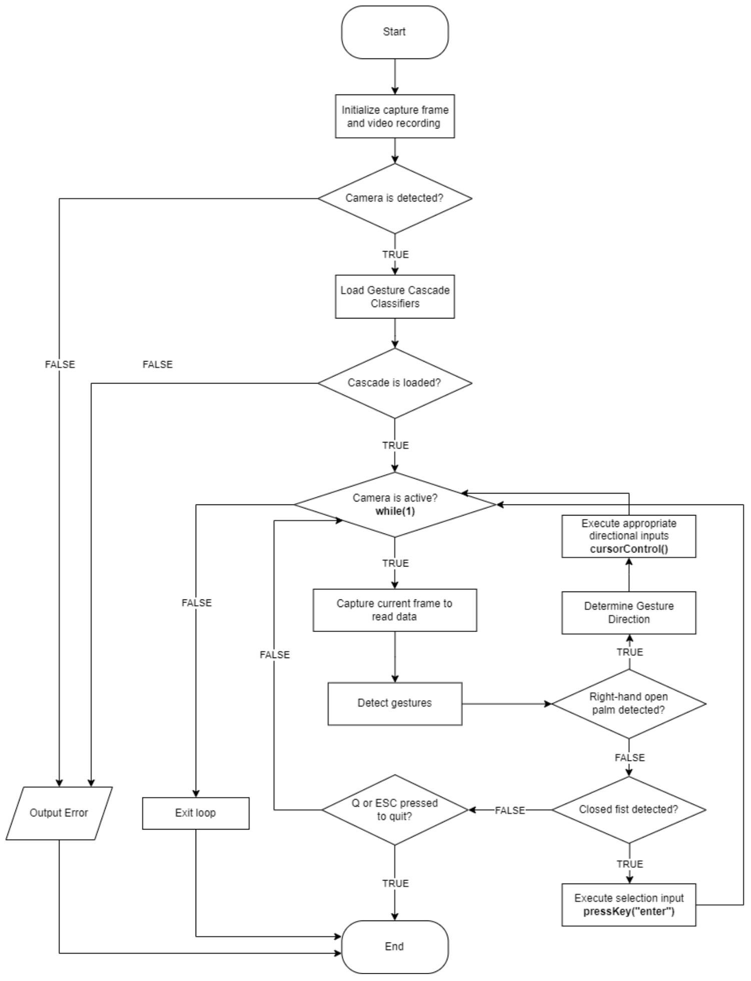

Start
Initialize capture frame and video recording
Camera is detected?
TRUE
FALSE
Load Gesture Cascade Classifiers
FALSE
Cascade is loaded?
TRUE
Camera is active? while(1)
TRUE
FALSE
Capture current frame to read data
FALSE
Detect gestures
Right-hand open palm detected?
TRUE
Determine Gesture Direction
Execute appropriate directional inputs cursorControl()
FALSE
Closed fist detected?
FALSE
Q or ESC pressed to quit?
TRUE
Execute selection input pressKey("enter")
Output Error
Exit loop
TRUE
End

Figure 2. Gesture Control Program Flowchart

a) *cursorControl():* This function, as the name suggests, controls the position of the mouse cursor based exclusively on the detection of the open right-hand palm. More specifically, at each frame, the function takes in two 2-dimensional coordinates as four parameters. The first coordinate, that of the current on-screen cursor, represents the position of the open palm at the end of the previous frame. The second coordinate is derived from the detected open palm in the current frame.

Once these coordinates are known, the function then calculates vertical and horizontal changes, and whichever is greater takes precedence in the decision-making regarding the direction of which to move the selection. Then, the cursor is temporarily reset to the origin point using $cursorRefreshTemp()$, and depending on the determined direction, the appropriate command is passed onto $pressKey()$ to execute the associated virtual key press, as seen in Fig. 3.

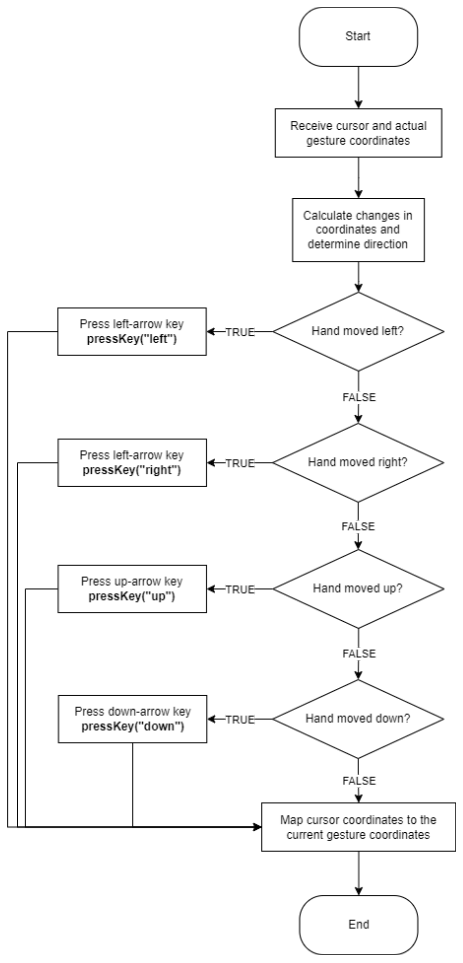


Figure 3. Gesture Control Program Flowchart

b) *pressKey():* This function, taking a string as a command from the $cursorControl()$ function, perform a specific virtual $Winuser$ keyboard press an input event depending on what the command is. For example, if $pressKey()$ receives a string "$left$", the input event will be assigned the virtual key VK_LEFT, which corresponds to a typical keyboard's left-arrow button [19]. After the appropriate virtual key is set, the input event will then be passed into the $SendInput()$ function to simulate pressing and releasing the button. As of this writing, this function supports the all arrow keys, and the RETURN/ENTER key.

c) *cursorRefreshTemp():* This function sets the cursor position using the Windows function $SetCursorPos()$ to position (0, 0), or the top-left corner of the screen, and execute a left-mouse click event consisting of pressing and releasing actions using the $mouse\_event()$ utility function.

3) Results

In its current state, the gesture control program allows users to comfortably navigate Microsoft Windows desktop and open highlighted items using hand gestures. To a lesser extent, users will be able to navigate within an active window on the screen as long as the window covers the origin point (0, 0) and will not be relegated to the background when a click is registered.

IV. Conclusions and Recommendations

The resurgence of touchless interaction, propelled by a marked improvement in processing power and advancement in artificial intelligence, represents the latest ongoing effort in human-computer interaction research aimed at making technologies more accessible to all. With tangible and demonstrable benefits to users, especially those with limited mobility, integrating touchless user interface into current OSs is no doubt a worthwhile endeavor.

Building upon the work of many developers in the past in this paper, the PI has been able to successfully implement a vision-based gesture control prototype on Microsoft Windows as a proof of concept of pseudo low-level implementation. Specifically, this prototype is developed using as few external libraries as possible, namely the ubiquitous computer vision library OpenCV, while leveraging built-in hardware, such as the camera, and existing Windows system-level utilities to achieve hands-off control. Users can use this prototype to navigate and open applications on a desktop screen. However, limited access to truly low-level relegates this program to the application level, which is nonetheless hoped to open up curiosity and interest in further research into OS-level development by independent parties.

V. Limitations and Suggestions for Future Research

Being a prototype and a proof of concept, this implementation of gesture control is not without limitations. Perhaps the most glaring shortcoming of is the fact that the program is not truly at the OS level, as it requires an external computer vision library to compile and run. In addition, the program's gesture detection is limited to two gestures, open palm and closed fist, respectively mapped to navigation and selection. As such, users cannot reliably navigate windows or apps other than the desktop. This ML-based method also requires having a sizable dataset for each gesture in order to train a good recognition model, which can potentially have negative performance implications for older, less powerful systems. And since detection is also highly dependent on the field of view of the camera, where a narrow field of view requires repeated gesturing, which can lead to arm strain and fatigue. Furthermore, due to a short timeframe and various restrictions placed upon proprietary OSs, combined with unexplained package installation peculiarities, cross-platform compatibility is not yet explored. And finally, in terms of security, being a vision-based gesture recognition program, security is not guaranteed: another set of hands can easily take over and perform unauthorized tasks.

These limitations, however, may highlight ways the program and the concept of integrating features into the OS layer can be improved upon. While the PI has insufficient access privileges to make changes to commercially available systems, those who are part of the owning companies can leverage their positions and expertise to integrate vision-based controls, similar to how Apple Inc. has accomplished with VisionOS and the Apple Vision Pro headset [3]. Meanwhile, independent developers could continue exploring implementations on open-source platforms such as Linux and develop distributions with gesture control out of the box. And in addition to desktop computers and laptops, future research can delve into how feasible and safe gesture control can be used as part of operating heavy machinery, such as driving.

Ultimately, the PI's foray into computer vision and low-level software development, limited as it may be, is hoped to provide some guidance for future developers who wish to elevate human-computer interaction and make technologies more accessible and easy-to-use. As ventures into technological accessibility have the potential to prove themselves worthwhile as an investment, business leaders and policymakers, with billions already invested in advanced technologies such as generative artificial intelligence [21], can expand and open up proprietary systems to truly make powerful technologies work for the masses. Until then, however, these assumptions, as hopeful as they are, remain speculations. Nonetheless, as technology continues to solidify its grip on human societies, enabling more people to use products of progress is not only a technological milestone, it is a moral imperative.

References

[1] Z. Sarsenbayeva *et al.*, "Mapping 20 years of accessibility research in HCI: A co-word analysis," *International Journal of Human-Computer Studies*, vol. 175, Jul. 2023, doi: https://doi.org/10.1016/j.ijhcs.2023.103018.

[2] “Definition of touchless user interface,” *PCMAG*.
https://www.pcmag.com/encyclopedia/term/touchless-user-interface (accessed Apr. 25, 2024).

[3] “Apple Vision Pro,” *Apple Inc*, 2024. https://www.apple.com/apple-vision-pro/ (accessed Apr. 25, 2024)

[4] B. J. Doddegowda, C. L. Monika, S. Manoj, K. Manoj, and Lakshmikanta, “Gesture Based OS Navigation and Control,” *International Journal of Advance Research and Innovative Ideas in Education*, vol. 9, no. 3, pp. 1159–1165, 2023.

[5] V. Chang, Rahman Olamide Eniola, L. Golightly, and Q. Xu, “An Exploration into Human–Computer Interaction: Hand Gesture Recognition Management in a Challenging Environment,” *SN computer science*, vol. 4, no. 5, Jun. 2023, doi: https://doi.org/10.1007/s42979-023-01751-y.

[6] M. Salunke, N. Kulkarni, and H. Yadav, “Gesture Control System: Using CNN based Hand Gesture Recognition for Touch-less Operation of Kiosk Machine,” Aug. 2022, doi: https://doi.org/10.1109/iconsip49665.2022.10007509.

[7] “Car gesture control: bringing sci-fi technology to your vehicle,” *Hyundai News*.
https://www.hyundai.news/eu/articles/stories/car-gesture-control-bringing-sci-fi-technology-to-your-vehicle.html

[8] B. K. Chakraborty, D. Sarma, M. K. Bhuyan, and K. F. MacDorman, “Review of constraints on vision-based gesture recognition for human–computer interaction,” *IET Computer Vision*, vol. 12, no. 1, pp. 3–15, Feb. 2018, doi: https://doi.org/10.1049/iet-cvi.2017.0052.

[9] J. Ao, S. Liang, T. Yan, R. Hou, Z. Zheng, and J. Ryu, “Overcoming the effect of muscle fatigue on gesture recognition based on sEMG via generative adversarial networks,” *Expert systems with applications*, vol. 238, pp. 122304–122304, Mar. 2024, doi: https://doi.org/10.1016/j.eswa.2023.122304.

[10] “About Face ID advanced technology,” *Apple Inc.*, Aug. 22, 2023. https://support.apple.com/en-us/102381 (accessed Apr. 25, 2024)

[11] W. Gallagher, “Flashlight on iPhone - everything you need to know,” *AppleInsider*, Jan. 17, 2021.
https://appleinsider.com/articles/21/01/15/flashlight-on-iphone---everything-you-need-to-know (accessed Apr. 25, 2024).

[12] W. Stallings, *Operating Systems: Internals and Design Principles*, 9th ed. Essex: Pearson Education Limited, 2018. pp. 70-71. Accessed: Apr. 25, 2024. [Online]. Available:
https://ia802302.us.archive.org/4/items/c-64_20211011/C64.pdf

[13] “About OpenCV,” *OpenCV*, 2018. https://opencv.org/about. (accessed Apr. 25, 2024)

[14] “Quartz Event Services,” *Apple Inc*.
https://developer.apple.com/documentation/coregraphics/quartz_event_services (accessed Apr. 25, 2024).

[15] R. Gross, I. Matthews, and S. Baker, “Appearance-based face recognition and light-fields,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 4, pp. 449–465, Apr. 2004, doi: https://doi.org/10.1109/tpami.2004.1265861.

[16] “Color Detection,” *MathWorks*. 2024.
https://www.mathworks.com/help/simulink/supportpkg/android_ref/color-detection.html (accessed Apr. 25, 2024)

[17] P. Viola and M. Jones, “Rapid Object Detection Using a Boosted Cascade of Simple Features,” *Mitsubishi Electric Research Laboratories.* 2004. pp. 2-7. Accessed: Apr. 25, 2024. [Online]. Available:
https://www.merl.com/publications/docs/TR2004-043.pdf

[18] “OpenCV: Cascade Classifier Training,” *OpenCV*. 2024.
https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html (accessed Apr. 25, 2024)

[19] S. Sthapit, “Sandeep-Sthapit/HandGestureDetection,” *GitHub*.
https://github.com/Sandeep-Sthapit/HandGestureDetection (accessed Apr. 25, 2024).

[20] Karl-Bridge-Microsoft, DJm00n, masaru-iritani, drewbatgit, DCtheGeek, embender, and msatranjr. “Virtual-Key Codes,” *Microsoft Corporation*, Sep. 22, 2023.
https://learn.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes (accessed Apr. 25, 2024).

[21] G. D. Vynck and N. Nix, “Big Tech keeps spending billions on AI. There’s no end in sight.,” *Washington Post*, Apr. 26, 2024. Available:
https://www.washingtonpost.com/technology/2024/04/25/microsoft-google-ai-investment-profit-facebook-meta/ (accessed Apr. 25, 2024)

Appendix A
Development Environment and Hardware

Environment: Microsoft Visual Studio 2022 Community Edition
OS: Windows 11
CPU: Intel Core i9-13950HX
RAM: 96 gigabytes of DDR5 RAM
Discrete GPU: NVIDIA RTX 4090 Laptop GPU

Appendix B
Gesture Detection

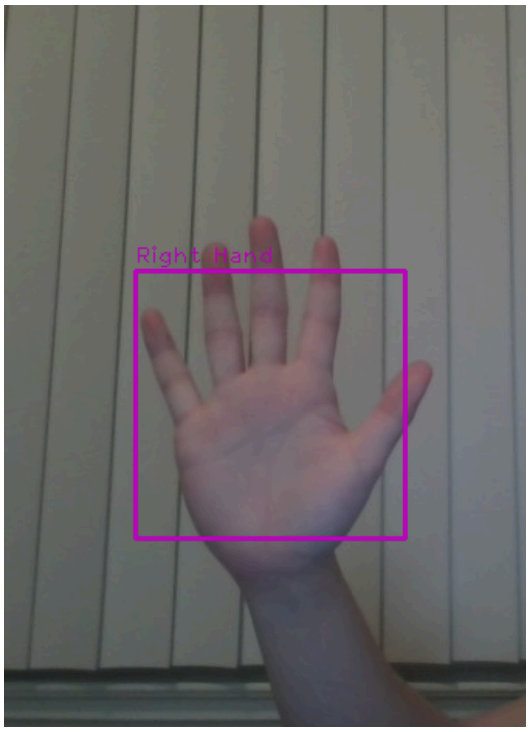


Figure 4. Right-hand Open Palm Detection

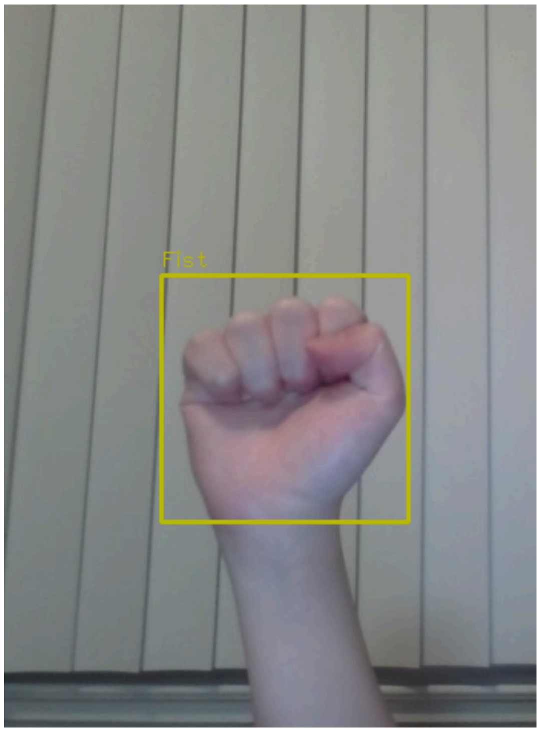


Figure 5. Closed Fist Detection

Acknowledgement: Gesture model trained by Sandeep Sthapit.