Project Report for Bachelor's Degree

Implementation of Touchless Touchscreen Technology in the Form of an Android Application

By

Erfan Sadigh Nejati

Supervised by

Dr. Jafar Razzm Ara

Computer Science Department

University of Tabriz

Tabriz, Iran

Winter - Spring

2023

Abstract

The Touchless technology provides the ability to interact with a system without any physical contact. However, since this technology is not easily accessible to everyone and can be costly if available, in this project, we aim to implement this technology using software on an Android-based device. This implementation will allow users to interact with various types of displays, regardless of whether the display itself supports touch capabilities or not and regardless of its dimensions, by using touchless gestures. This will be achieved without incurring significant expenses.

1. Introduction

When we place our finger on the screen, in order for the touching action to occur, in general, there are two actions of generating and sending the touch event that the operating system must handle. The touch event is generated when we place our finger on the desired screen and its position is calculated based on where we have kept our finger on the screen. In this project, as there is no place for the finger to hit, I have used hand gestures using ML solutions to announce the position of the finger, and the next step is to calculate the position to the system. The agent is informed to perform the necessary reaction to perform the touch action, which in this project, in order not to install interface software for each target system on which the touch action is supposed to occur, I have used the HID protocol provided by the USB company and by implementing it on BLE technology, which at the same time provides the least possible friction for the user and reduces the complexity of the implementation.

1.1. Requirements

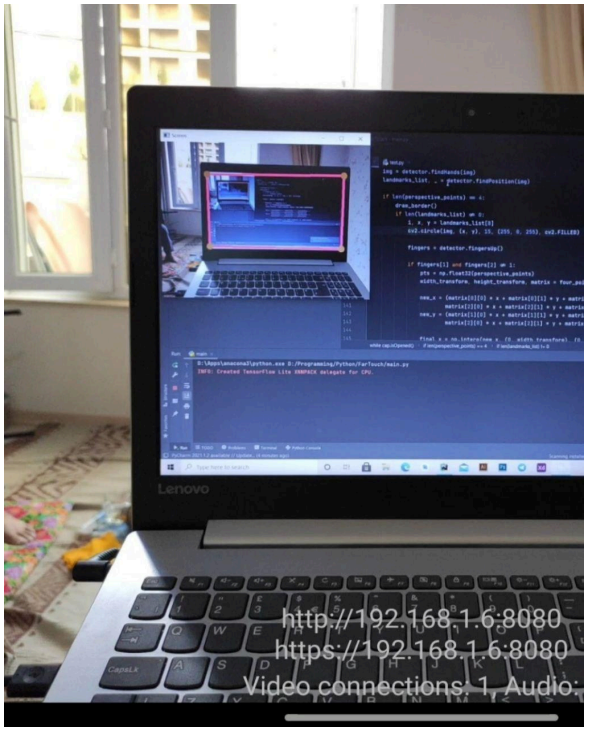
One device with a minimum Android 9 operating system with a mid-range processor.

2. Touch Event Generation

To make the operating system perform the touch action, we first need to deliver the touch event. In this project, I have utilized the device's camera sensor and started with a basic idea, gradually refining it to achieve the desired result.

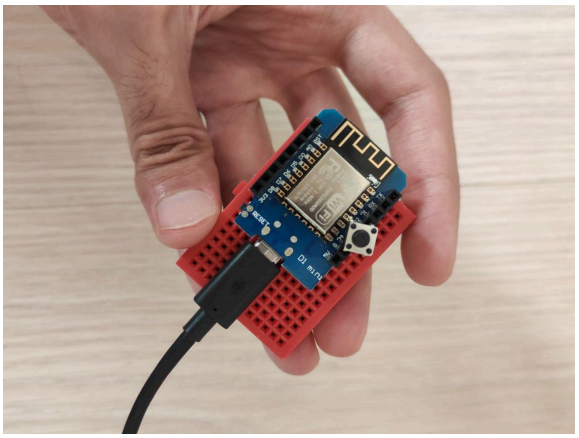
2.1. Static Frame, OpenCV & Wemos Board, External Processing

In this idea, using the image captured by the camera sensor, the corners of the desired display screen were manually identified. Then, with the help of a colored cap on the fingertip, I indicated the relative position on the screen. The image processing, aided by the OpenCV library, was performed in a Python program within the target system. The Android software was responsible for sending camera frames to the software through a socket, while the actual image processing was done within the Python program.



And in order to inform the software on the target system that the target time is the position where we want the touch action to occur, I have taken the help of the Wemos D1 Mini board, which by creating a Socket and releasing a

sequence of characters 0 and 1 that was created after pressing the button connected to it, the sending part of the software, by receiving them, performed the Release and Press actions on the position of the generated event.

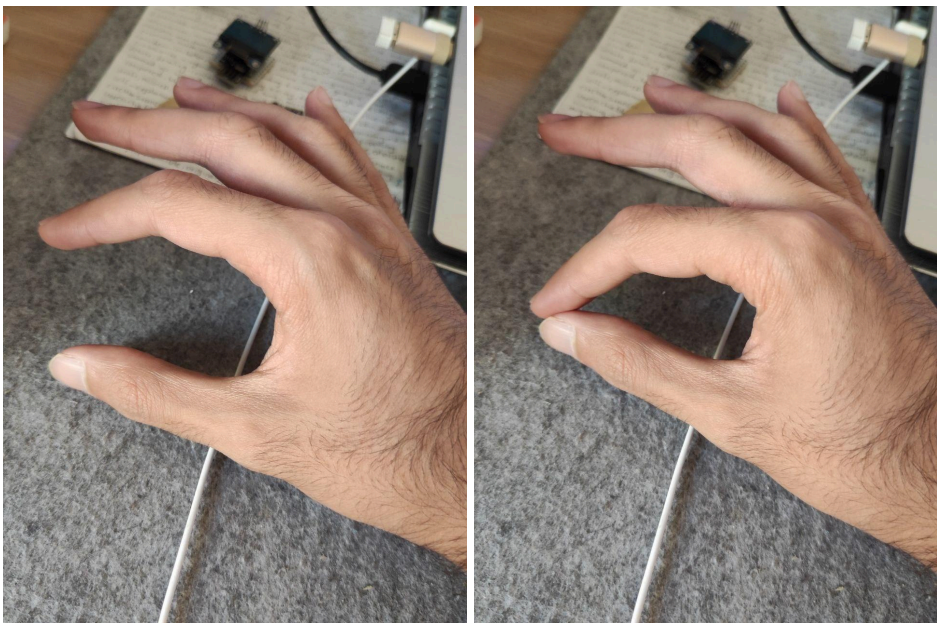


In this idea, there were some fundamental challenges observed, such as inaccurate calculation of the event position in the presence of objects with the same color as the colored cap in the environment, and the inability to detect the colored cap accurately under varying lighting conditions.

2.1. Static Frame, MediaPipe, External Processing

The MediaPipe library, by providing a series of pre-trained machine learning models, allows us to track all the hand landmarks accurately. This proves to be an outstanding solution for the challenges we faced with object detection and noise from objects with similar colors when using the color detection algorithm from the OpenCV library.

As a result, I rewrote the code related to finger-tip detection using the MediaPipe library. Additionally, utilizing this library eliminated the need for the Wemos D1 Mini board to inform the system about "Press" and "Release" actions for touch event positions. Instead, I removed the Wemos board entirely from the project, as I could now measure the distance between the fingertips of the hand and the point of touch. By setting a condition that if the distance between the fingertips is less than a certain threshold value, it signifies "Press," otherwise, it indicates "Release." This simplified approach allowed me to achieve the desired touchless interaction without the need for external hardware like the Wemos board.



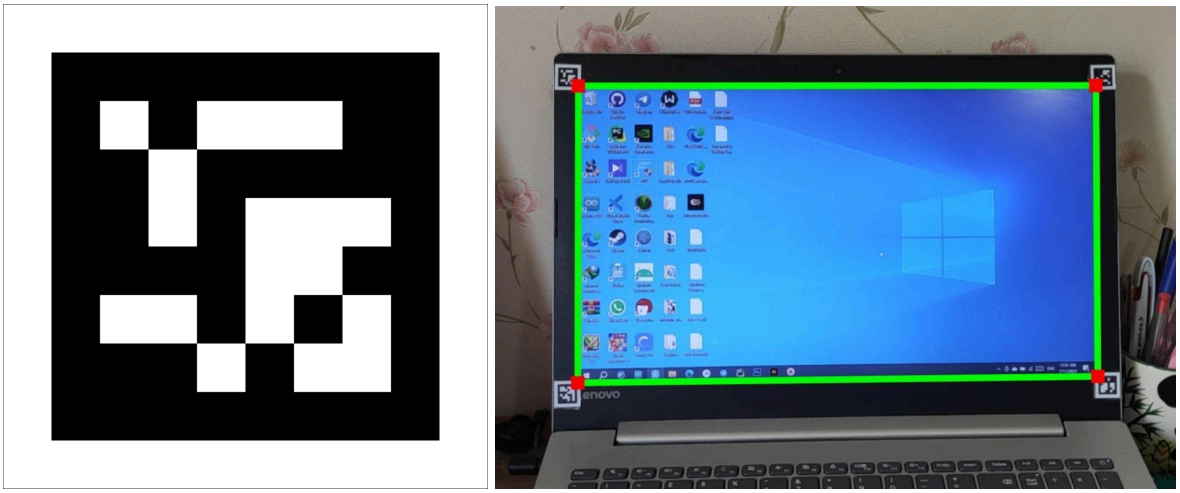
The implementation of the MediaPipe library not only reduces the system resource consumption by the software, consequently eliminating the need for a separate thread to monitor the event status from the received socket data but also simplifies the implementation complexity by eliminating the use of the Wemos D1 Mini board. This is because there is no longer a requirement to delve into hardware programming concepts. As a result, the overall cost of implementing the project is reduced as well.

2.3. Dynamic Frame, MediaPipe, External Processing

Determining the static frame of the display during the program's execution on the target system posed a challenge as it resulted in a poor user experience in various scenarios. For example, if the Android device is shaken or if the user prefers to hold the phone in hand, the process of determining the static frame needs to be repeated. To address these issues, I decided to automate this process.

To achieve this, I started by exploring the idea of detecting a specific shape that is attached to the corners of the desired display screen. This shape would act as a marker to identify the static frame. Since ML-based solutions were proving to be efficient in various aspects, I decided to use the Tensorflow framework to train a machine learning model for object detection. This allowed me to efficiently detect the specific shape and automate the process of determining the static frame, improving the project's overall efficiency.

I was able to train a model with the help of the tutorials available in this regard, but because, firstly, I had no practical experience in this field before, and I had taken help from YouTube videos to build the model, and I was not specialized in the concepts of model optimization and how to do it, and the time frame of the project was limited, I could not learn them. With poor hardware resources, it did not meet my expectations. As a result, I came across ArUco markers as an alternative solution. As their name suggests, these markers are used in AR technology, and the OpenCV library, by providing a series of functions, gave developers the option to generate different examples of it and use it to detect the position.



This solution simultaneously demonstrates excellent performance in detecting while consuming minimal hardware resources, and its implementation is straightforward by writing a few lines of code.

2.4. Dynamic Frame, MediaPipe, Internal Processing

As a result, transferring the frames generated by the camera sensor through a socket using Wi-Fi caused delays and even dropped frames, leading to unintended touch events. Additionally, it introduced extra overhead to the device, as the frame transfer occurred twice, once from GPU to CPU and then from CPU to the socket, resulting in increased energy consumption and elevated device temperature, affecting the overall performance of the frame transfer process.

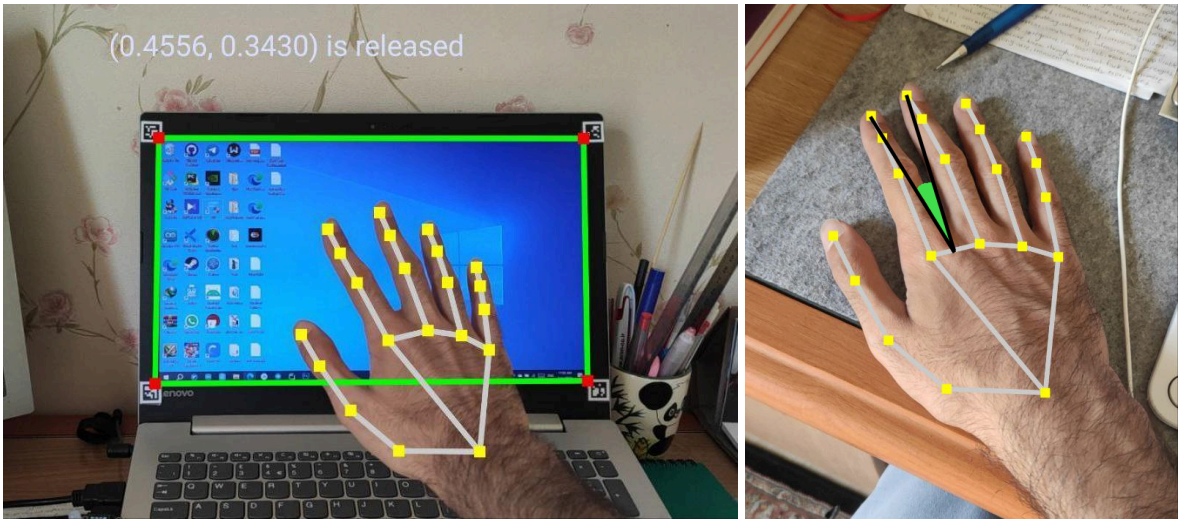
Moreover, the dependency on a router device as an intermediary for data transfer between the Android device and the target system, along with the mandatory installation of intermediary software for image processing, reduced the cross-platform capabilities of the project. To address these issues, I moved the entire image processing process responsible for generating touch events to the Android device.

The use of the MediaPipe library provided an efficient solution as it offered cross-platform support and easy implementation with the Kotlin programming language for Android. By passing the received camera frame

through the available API to the analyzer and receiving the results through a callback, the image processing task was handled within the device. Additionally, I decided to use ArUco markers as an alternative solution. The OpenCV library facilitated this by providing a set of functions for generating and detecting ArUco markers, which are widely used in Augmented Reality (AR) technology.

Throughout multiple trials, I discovered that the position of the hand in relation to the camera often led to obscuring some fingers, particularly when they were close together or separated. This required the model to predict the positions of the obscured fingers, leading to noisy finger positions. To address this, I needed to normalize the finger positions with a higher threshold, which resulted in a delay in finger tracking. To mitigate this issue, I decided to use the middle finger instead of the index finger, as it offered a more complete hand view in the camera's field of view, reducing the need for extensive normalization.

Furthermore, changes in the distance between the hand and the camera resulted in unintended touch events, as the size metric used in a fixed depth from the camera became unreliable. To counteract this, I introduced an additional criterion by measuring the minimum angle between the middle finger and the tip of the thumb's metacarpal joint, which helped minimize the occurrence of unintended touch events.



3. Touch event dispatching.

After the touch event is generated, it is time to send it to the target system. For this, I have followed an evolutionary process, just like the event generation section, to reach the target.

3.1. Using the interface software on the target system

As with the original idea of generating the touch event, I had written special programs to pass the touch event with the functions that the operating system provided to the programmers, so that the touch action is performed on the desired position, because this method created a dependency on the type of operating system, and in addition, some operating systems such as Android itself had a series of restrictions to work with event emitting functions, so I looked for a better solution.

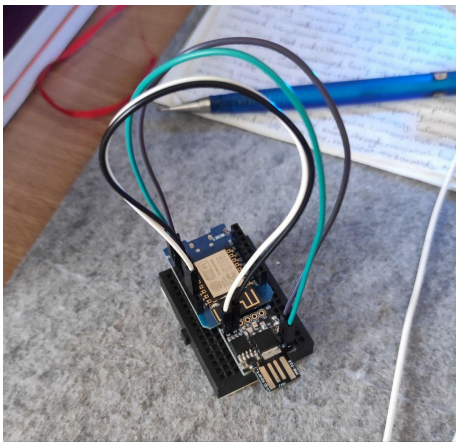
3.2. Use the HID protocol

Although most of us use this protocol on a daily basis to carry out daily tasks while working with electronic tools, we don't even know its name. This protocol is used from mouse and keyboard to TV controls and smart watches, and even to display content and touch screens. This protocol was provided by the USB company to reduce the cost of designing and manufacturing peripherals.

3.2.1. Implementation with a combo of Digispark and Wemos boards

The V-USB library makes it possible to implement the HID protocol on the cheap Digispark ATtiny 85 board more easily. I decided to connect them due to the WiFi module on the Wemos D1 Mini board by establishing serial

communication between these two boards and by sending an event to the socket built in the Wemos D1 Mini board through the Android application, after the event is generated, the responsibility of sending the event to the target system will be assigned to the Digispark ATtiny 85 board. Fortunately, before wasting more time and energy, I understood the type of socket with Wemos D1 Mini board can create not compatible with Android OS.



3.2.2. Implementation of Bluetooth Low Energy Technology

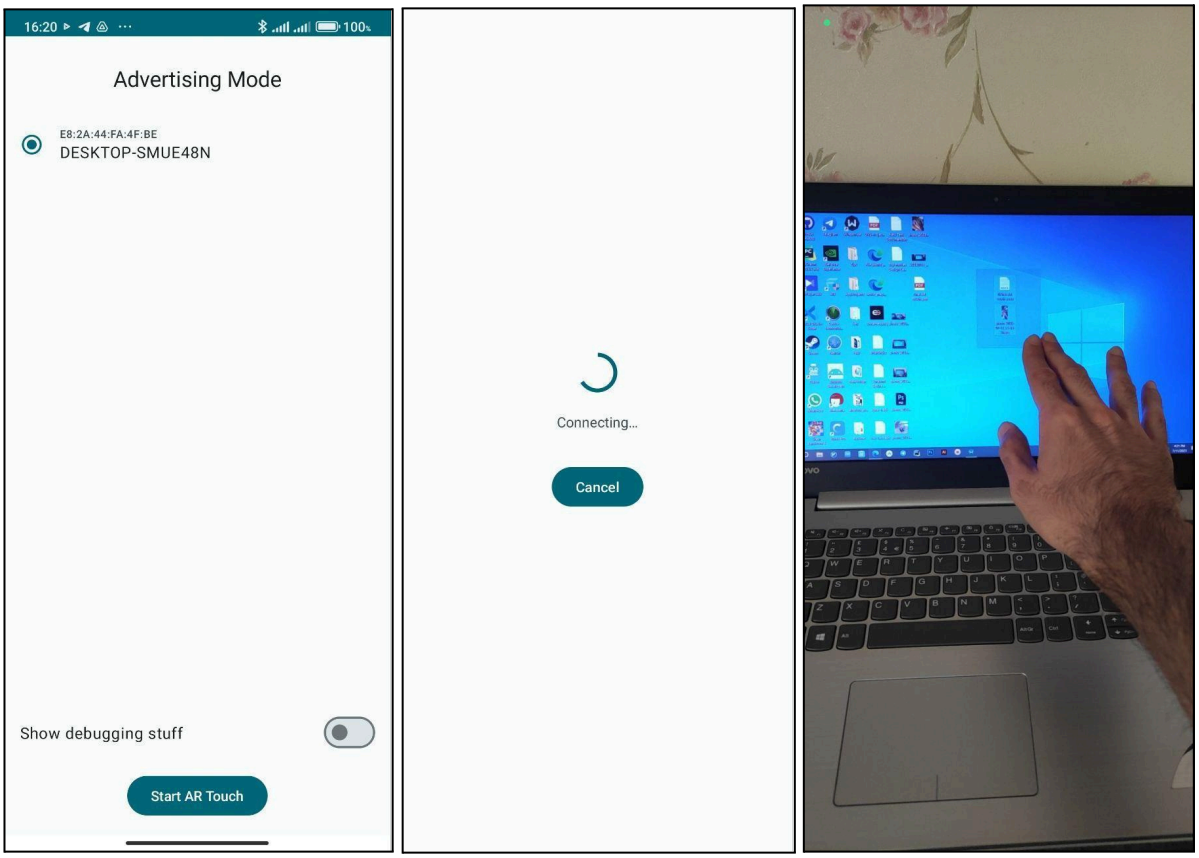
The technology that most wireless peripherals rely on is called BLE, and it is a version of Bluetooth Classic that, as the name suggests, is designed for less consumption and has many uses in IoT. Fortunately, the HID protocol can be implemented on it, and since the Android operating system itself, of course, from version 9 onwards, it is provided to its developers in the form of the BluetoothProfile class, which is implemented. along with the Touch report descriptor passed to it for the touch action to occur.

4. Project planning

After each part reached its initial maturity, I did the planning phase at the same time as the Agile implementation phase using the Github Project tool, which was that I set a milestone with general tasks and advance the project along with it, although sometimes it was necessary to step beyond it.

5. Conclusion

This project has evolved with the initial idea of processing and sending events in the target system to perform all processes on the Android device, and during that, the combined use of new and old technologies related to Electronics and AI has been able to bring Touchless technology to touch any type of screen by relying only on an Android software with the lowest possible cost and an acceptable experience, and the experience of working with large touch screens that involve a lot of money is provided almost free of charge.



6. References

[1] OpenCV and Python Color Detection - PyImageSearch
[2] ESP8266 Arduino: Socket Server - techtutorialsx
[3] MediaPipe | Google for Developers
[4] OpenCV: Detection of ArUco Markers
[5] ArUco - Browse /ArucoNano at SourceForge.net
[6] Human Interface Devices (HID) Specifications and Tools | USB-IF
[7] A USB HID Keyboard, Mouse, Touchscreen emulator with Teensy - CodeProject
[8] USB Complete Fourth Edition : The Developer's Guide (Complete Guides series): Axelson, Jan: 9781931448086: Amazon.com: Books
[9] Bluetooth HID: An Intro to Human Interface Devices with BLE (novelbits.io)
[10] The Ultimate Guide to Android Bluetooth Low Energy | Punch Through