COMP4801 Final Year Project [2023-24]

FinalReport

Virtual Keyboard App Using Computer Vision

Supervisor: Dr Pan Jia (jpan@cs.hku.hk)

Student: Mak Tsz Shing (3035685914) (u3568591@connect.hku.hk)

Data of Submission: 26 April 2024

Abstract

Texting on mobile devices is usually achieved by tapping both thumbs on the built-in on-screen virtual keyboard. However, the small form factor of mobile devices has made typing difficult. Therefore, there is need for intuitive, flexible, and efficient alternative text input method for mobile devices. In this project, a virtual keyboard Android app using computer vision technique will be developed to allow users to type like typing on a physical keyboard. The front facing camera of the mobile device will be utilized to track users' finger gestures of typing on a flat surface in front of the mobile device. The app will be developed in Android Studio using Kotlin as the programming language. The Hand landmark detection solution from MediaPipe will be used as the machine learning solution for hand landmark detection. Algorithm and solutions for detecting finger tapping motion and transferring the projection of tapping position to key input in the keyboard layout is introduced. Several challenges and limitations are identified, such as the difficulties in building relationship between the tapping finger and the tapping surface, the limitation of motion tracking by a single camera and the requirement for high device performance. It is expected that this project will be beneficial to virtualization of other controllers using computer vision techniques.

Acknowledgement

I would like to express a great gratitude to Dr. Pan Jia for supervising this final year project and providing advice.

List of Contents

List of Figures

List of Algorithms

Abbreviations

CV	Computer Vision
UI	User Interface

1. Introduction

1.1. Background and motivation

The emergence of mobile devices has revolutionized the way we communicate, work and access information. As mobile networking and microprocessor technologies progress, mobile devices such as smartphones are becoming powerful enough to replace desktop computers [1]. In 2019, mobile users accounted for 53% of all web traffic [2]. People are more likely to use a smartphone instead of a desktop computer to perform tasks like online shopping, video streaming and browsing social media sites. Moreover, there is an growing trend of people using their smartphones for work-related tasks, such as emailing, calling and appointment scheduling [3].

Despite the convenience offered by mobile devices, their compact form factor makes text typing difficult. For instance, texting on smartphones usually involves tapping on the virtual on-screen keyboard with both thumbs. In contrast to a standard physical keyboard that enables users to type with all ten fingers, a virtual on-screen keyboard has considerably smaller keys that makes it hard for users to use multiple fingers for typing. The small keystroke size on an on-screen keyboard would also reduce the typing speed and increase the chance of making a typo. To improve the typing experience on mobile devices, new features like autocorrect, SwiftKey [4], and speech-to-text input are currently added to mobile virtual keyboard apps. These features allow users to type on mobile devices without tapping on individual keys on a virtual on-screen keyboard and reduces typing errors.

Inspired by the new features on mobile virtual keyboard apps, it is believed that alternative text input methods are needed on mobile devices. The new text input method should overcome the limitation of the small form factor and provide good user experience. With the increase in computation power and resolution on mobile devices, we are motivated to explore an innovative solution by leveraging computer vision and machine learning techniques. We seek to provide users with a more natural and intuitive way to interact with their mobile devices, ultimately enhancing their typing experience and overall satisfaction.

In the Consumer Electronics Show 2020 (CES2020), Samsung introduced a conceptual product called “SelfieType”, which uses the front-facing camera as a keyboard for mobile devices. The proprietary SelfieType AI engine will analyze user’s finger movements and convert them into keyboard inputs on the smart phone. No additional hardware is required. This technology provides a portable and user-friendly typing experience on smart phones. [5]

With this idea, the project aims to build an innovative input method by developing an application with similar idea to “SelfieType” and explores the possibility of building a keyboard using computer vision techniques and finger motion tracking on mobile devices.

1.2. Objectives and deliverables

The objective of the project is to develop a system level virtual keyboard app on the Android platform that provides user with similar typing experiences on a standard keyboard by leveraging Computer Vision and motion tracking techniques. Instead of tapping on the screen of mobile devices, users can type by tapping on any flat surfaces as tapping on a real keyboard in front of the device camera. The virtual keyboard app utilizes the device camera to track users’ finger movements and gestures to determine the text input. No external hardware is required.

1.3. Report content

The remainder of this paper proceeds as follows. Chapter 2 explains the detailed methodologies for developing a vision-based virtual keyboard app. Chapter 3 reports the observations and findings during the app development process. Chapter 4 introduces the app interface. Chapter 5 describes the challenges and limitations in developing the app. Chapter 6 outlies the future plan. Chapter 7 summarizes the report.

2. Methodologies

2.1. IDEs and programming languages

The app is developed using Android Studio. It is the official IDE developed by Google particularly for development of Android applications [6]. Comparing to other IDEs, Android Studio offers Graphical UI for drag-and-drop components on the app interface.

Kotlin is used as the programming language for the app. It is a modern programming language used by over 60% of professional Android developers [7]. In addition to its compatibility with Java, Kotlin has a more concise syntax than Java. With the great community support, developing the app using Kotlin would be quicker and easier.

2.2 Android Input Method Service

To create a system level input method, an input method editor (IME) is created in the app. A service class that extends the InputMethodService class is implemented to handle key input events, send text inputs, and manage the keyboard UI in different states. The states sequence follows the IME lifecycle (see Figure 1). Each state in the lifecycle is responsible for a single task, such as inflating the keyboard layout and creating the candidate view for showing word suggestions.

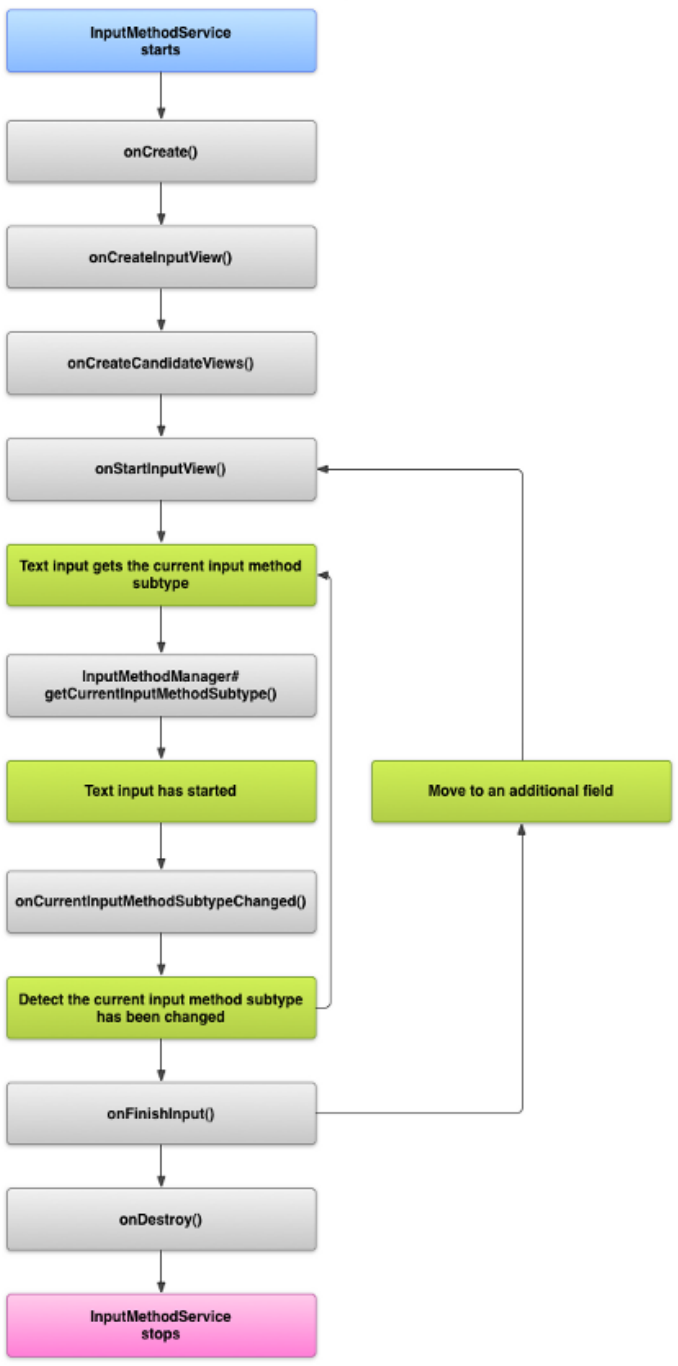

InputMethodService starts
onCreate()
onCreateInputView()
onCreateCandidateViews()
onStartInputView()
Text input gets the current input method subtype
InputMethodManager# getCurrentInputMethodSubtype()
Text input has started
Move to an additional field
onCurrentInputMethodSubtypeChanged()
Detect the current input method subtype has been changed
onFinishInput()
onDestroy()
InputMethodService stops

Figure 1. The lifecycle of an IME [8]

2.3. Image Capture

The typing actions of user fingers are captured using the front camera of users' mobile devices. CameraX, a jetpack library developed by Google, is used to manage the camera usage of the app, including frame capturing, frame previewing and frame processing in background. CameraX also maintains the consistency of camera behavior across various devices with different Android versions (Android 5.0+).

2.4. Hand detection

Hand detection is done using the Hand landmark detection solution provided by MediaPipe. MediaPipe is an open-sourced framework developed by Google Research for building custom on-device ML solutions [9]. It abstracts the complexity of running on-device ML solutions, such as CPU/GPU acceleration and deploying the ML solutions on different platforms. It also supports well-known programming languages such as JavaScript, Python and C++, and running on mobile environment such as iOS and Android [9].

Mediapipe provides 14 pretrained solutions of common ML use cases, such as hand gesture recognition and face detection. The hand landmark detection solution will be used for hand detection in this project. The solution detects the existence of hand in static or continuous stream of image data. If one or more hands exists, the model will return the handedness (left or right hand) and the world coordinates of the 21 hand knuckles (landmarks) on the hand in real time (see Figure 1 for details of the landmark). Figure 2 shows a example of using the land landmark detection solution in actual.

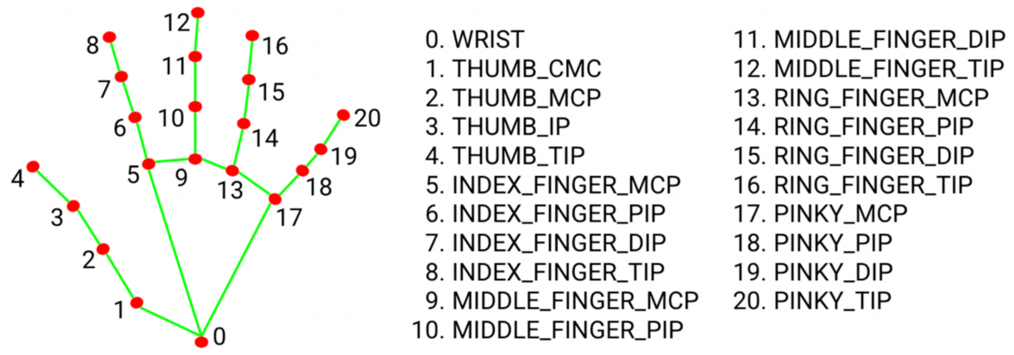

0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 2. 21 hand landmarks detected by MediaPipe hand landmark detection solution [10]. Coordinates of the landmarks of user's hand on the captured image frame can be retrieved.

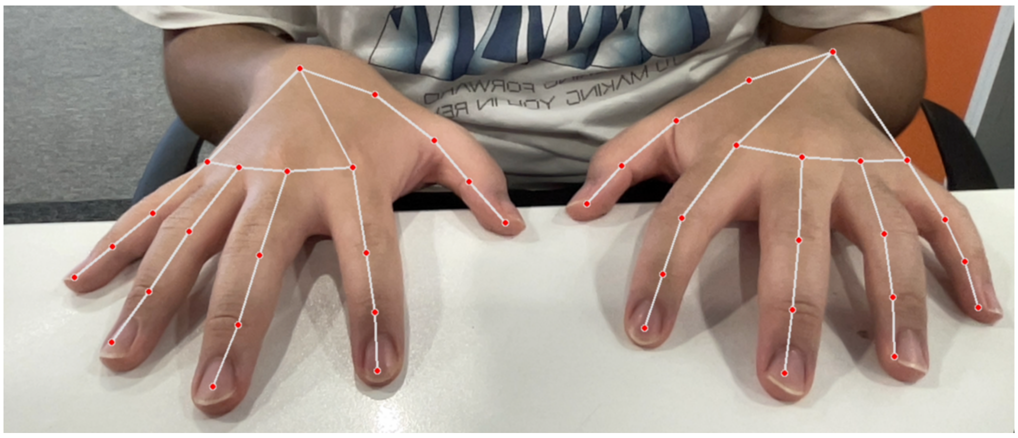


Figure 3. Actual example of using the MediaPipe hand landmark detection model. The key points on both hands are shown in red dots.

There are several advantages for utilizing the hand landmark detection solution by MediaPipe as the hand detection method in this project. First, the model is already well-trained with approximately 30K real-world images and several rendered synthetic hand models imposed over various background [10]. The effort of developing and training a custom ML model for hand recognition can be eliminated. Second, the latency of the model is acceptable for typing. According to the benchmark test done by Google on Pixel 6, the latency of the ML model is 17.12ms and 12.27ms on CPU and GPU acceleration respectively [10].

2.5. Reference Position Construction

Before typing, users need to place their hand in a stable position for a few seconds for the app to record the reference positions after the initialization of the camera and the hand landmark detection model. The reference position of the index finger, the middle finger, the ring finger and the pinky finger will be constructed. The coordinates of the reference positions are used as reference for key input determination on the virtual keyboard.

When the hand landmark detection model first detects the presence of both hands after the user starts the keyboard, the coordinates of fingertip landmarks of each frame will be recorded in a queue. It is assumed that the variation of each recorded coordinates will not be great, as the user’s fingers are staying stable on the tapping surface. Therefore, the reference position of each fingertip is calculated as the mean coordinates of the records in the queue.

To improve the accuracy of the reference position, outliers will be removed from the queue after the queue size exceeds a certain value. It is assumed that the recorded coordinates in the queue of each fingertip are normally distributed. To remove outliers in the queue of a fingertip, two temporary lists are constructed to store the horizontal and vertical projected coordinates in the record queue respectively. The mean and the standard deviation for both lists are then calculated. After that, the z-score of each record in both lists are checked to be within the range of two standard deviations of the corresponding list. If a record is out of the range, the corresponding coordinate record in the queue is removed.

After the queue size for all required fingertips reached a certain value, it is confident to adopt the mean coordinate from the queue of each fingertip as the reference position. The projected coordinates of all reference positions constructed will be shown on the screen.

2.6. Tap detection

Tap detection is to detect whether the user has pressed on a key on the virtual keyboard or not. Since there are no external sensors installed, the fingertip tapping can only be determined based on the finger joint movements when a user press a key.

The current algorithm of tap detection is based on the vertical motion tracking of fingertips. In general, a fingertip is considered as tapping when its fingertip keeps moving down towards the tapping surface across a period, and then moves up from the surface. This V-shaped motion is tracked by recording the coordinates of the fingertips returned by the hand landmark detection model in each frame into a queue. The records in the queue can be then split into two parts. The first part checks if the vertical projected coordinate of each record is greater than that of the previous record. The second part checks if the vertical projected coordinate of each record is smaller than that of the previous record. In the actual implementation, the queue is split in the (n-1) frames, letting n to be the size of the queue. The first part of the queue has size of (n-1) and the second part of the queue has size (n-(n-1)+1) = 2. Note that the (n-1)th frame is presented in both the first and the second part of the split queue.

Algorithm 1 Algorithm for finding candidate tapping fingers

```
 1: procedure GetTappingFinger($LandmarkQueue$)
 2:     $candidateTappingFingers \leftarrow []$
 3:     for all $hand$ in $LandmarkQueue$ do
 4:         $candidateFingertipsForThisHand \leftarrow []$
 5:         for all ($fingertip, coordinateQueue$) in $hand$ do

    ▷ Check fingertip is moving down in first N-1 frames
 6:             for $i$ in range ($coordinateQueue$.size() - 3) do
 7:                 if $coordinateQueue[i+1] < coordianteQueue[i]$ then
 8:                     skip this fingertip
 9:                 end if
10:             end for                              ▷ fingertip coordinate records

    ▷ Check fingertip decrease in y-coordinate exceeds threshold
11:             $j \leftarrow coordinateQueue$.size() - 2
12:             if ($coordinateQueue[j]$ - $coordinateQueue[0]$ < $threshold[fingertip]$) then
13:                 skip this fingertip
14:             end if

    ▷ Check fingertip is moving up in the last frame
15:             $k \leftarrow coordinateQueue$.size() - 1
16:             if $coordinateQueue[k] < coordinateQueue[k-1]$ then
17:                 $candidateFingertipsForThisHand \leftarrow fingertip$
18:             end if
19:         end for                                  ▷ fingertips in this hand
20:         $candidateTappingFingers \leftarrow candidateFingertipsForThisHand$
21:     end for                                      ▷ hand in LandmarkQueue

22:     return $candidateTappingFingers$
23: end procedure
```

Algorithm 1a. Procedure to find candidate tapping fingers

Algorithm 1a shows the actual implementation of finding candidate tapping fingers based on the coordinates record queue. Line 6-10 checks if a fingertip in a hand is moving towards the tapping surface by checking the vertical projected coordinate records in the first part in the queue with (n-1) entries. Line 11-14 checks if the extent of downward movement of a fingertip has exceeded a certain threshold. The threshold value will be discussed in the Chapter 3.2. This check prevents false detection for tiny fingertip movements. Line 15-19 checks if a fingertip in a hand has moved up between the second last frame and the last frame in the record queue. If all three conditions are satisfied, the fingertip is added to the candidate tapping finger list.

There may exist more than one finger detected as tapping at the same time. It is because when a user taps with a finger, the neighboring finger may move together. It is hard for a user to tap with only one finger moving while other fingers staying still. Therefore, if a finger is checked for performing the V-shaped tapping motion, it is added to the candidate tapping finger list of the corresponding hand.

To find the actual finger that is tapping, the candidate fingertips are passed to the second function (Algorithm 1b). The second function outputs the candidate fingertip with the greatest vertical projected coordinate. It is assumed that when a user is typing, only the finger pressing a key is touching the tapping surface, while other fingertips are located above the tapping surface. The output of the second function is the actual tapping finger of both hands.

```
procedure GetRealTappingFinger($candidateList, LandmarkQueue$)
    $realTappingFinger \leftarrow []$
    for ($hand, candidateTappingFingers$) in $candidateList$ do
        $realTappingFinger \leftarrow -1$
        $maxYCoor \leftarrow 0$

        for $candidate$ in $candidateTappingFingers$ do
            $yCoor \leftarrow LandmarkQueue[hand][candidate].y$
            if $yCoor > maxYCoor$ then
                $maxYCoor \leftarrow yCoor$
                $realTappingFinger \leftarrow candidate$
            end if
        end for                                          ▷ $candidate$
    end for                                              ▷ $candidateList$
    return $realtappingFinger$
end procedure
```

Algorithm 1b. Procedure to find the actual tapping finger from candidate tapping fingers

2.7. Key input determination

Key input determination is done by examining the relationship between the projected coordinates of the tapping fingertip (retrieved by the hand detection model) and the reference position of all fingertips. When a finger is detected as tapping a key, the projected coordinates of the tapping fingertip will be compared to the reference positions for determining the key tapped on the keyboard layout.

The relationship between the projected coordinates of the tapping fingertip and the reference position are generalized by observations. The details about the relationship will be described in the Chapter 3.3.

2.8. Key input generation

The determined key input is generated as text input in the text input field with the help of InputMethodService. The determined key input will be passed to the service class which extends the InputMethodService class. The InputConnection instance of the service class, which is the communication channel from the input method to the application that is receiving the text, is responsible to deliver the key input.

In addition, key input is also determined by a fixed finger positioning on the keyboard layout. Each finger of a user is responsible for only a limited set of keys. Figure 4 shows the finger positioning on the keyboard. For the left hand, the pinky finger is responsible for the keys “q”, “a” and “z”. The ring finger is responsible for the keys “w”, “s” and “x”. The middle finger is responsible for the keys “e”, “d” and “c”. The index finger is responsible for the keys “r”, “f”, “v”, “t”, “g”, and “b”. For the right hand, the pinky finger is responsible for the keys “p”, “enter” and “backspace”. The ring finger is responsible for the keys “o”, “l” and “period”. The middle finger is responsible for keys “i”, “k” and “comma”. The index finger is responsible for keys “y”, “h”, “n”, “u”, “j” and “m”.

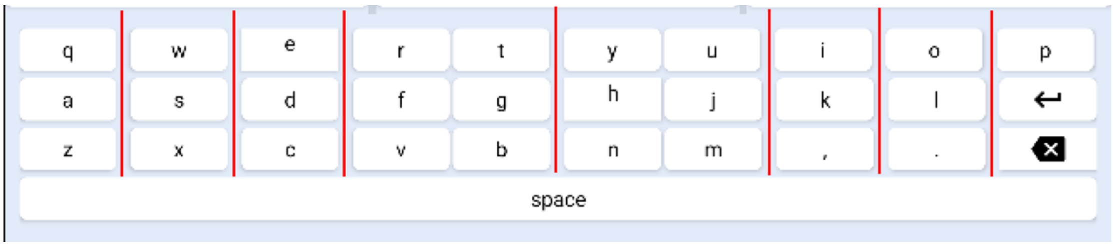


Figure 4. Finger Positioning on the keyboard layout

2.9. Word Suggestions

A Natural Language Processing model is used to provide suggested words while the user is inputting text. The model is adopted from an open-source GitHub repository. When a user has enabled the word suggestion preference and he / she is typing on a general text input field that does not contains sensitive information like passwords and addresses, the characters input by the user are feed into the model, and the suggestions from the model will be shown on the candidate view of the keyboard (see the Chapter 4.3 for detail).

3. Observations and Findings

This chapter reports the observations and findings over the app development process.

3.1. Tap Detection Threshold

Described in line 11 -14 of Algorithm 1a in Chapter 2.6, The downward movement extent of the fingertip of a tapping finger must exceed a certain threshold to prevent false tap detection. This eliminates the noise due to fluctuation of the hand landmark detection model result, and the small movements of fingertips.

The tap detection threshold is the minimum decrease in vertical projected coordinates of tapping fingertip. By observations, the extent of movement of each fingertip is different on a hand. The value represents the length portion of the height of the captured frame and are determined throughout app testing.

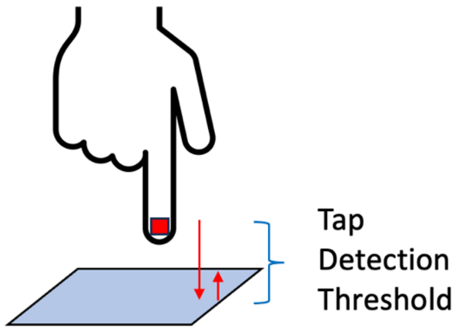


Figure 5. Illustration of tap detection threshold

3.2. Coordinates Queue Size

The projected coordinates of hand landmarks in each frame returned by the hand landmark detection model are stored into queues for motion tracking. The size of the coordinates queue is the number of recent records of projected landmark coordinates stored, which represents the length of the time period the motion of the hand landmarks is tracked.

Before the reference positions of fingertips are constructed, the projected coordinates of each fingertip (except the thumb tip) will be recorded in a set of queues. The size of this set of queues represents the confidence required for constructing the reference positions, as the reference positions are retrieved by the mean of coordinates in the queues. If a smaller queue size is set, the time required for constructing the reference positions will be shorter as a smaller number of frames are required. However, the probability of constructing skewed reference position will be higher due to fluctuation in the hand landmark detection model results and the possible movements of user's hand during the period. If a greater queue size is set, the time taken for constructing the reference positions will be longer. User experience will be affected because of the long initialization time before the user can start typing. To provide a better user experience, a shorter queue size for this set of queues is set in the actual implementation. The method to overcome the shortcoming of skewed reference position is to remove outliers in the queues, as described in Chapter 2.5.

After the reference positions are constructed, all the hand landmarks in the afterward frames will be stored to another set of queues for tracking the tapping motion. The size of this set of queues represents the time period a fingertip needs to move downward continuously to be considered as tapping on a key. If a small key size is set, tapping will be detected easily with the fingertip only moves downward in a short period of time. The probability of false tap detection is higher due to the fluctuation in the hand landmark detection model results. If a larger key size is set, the tapping finger would have to move downward continuously in a longer consecutive period, resulting in omitted detection for fast tapping motion. In the actual implementation, a short queue size is set for this set of queues to detect all possible tapping. To decrease the number of false tapping detected by noise, the tapping finger must also move downward to an extent greater than the tap detection threshold, as described in the previous chapter. Note that a finger is considered tapping on a key if the fingertip moves downwards in a continuous number of frames and the extent of downward displacement must exceed the corresponding tap detection threshold. The queue size for this set of queues and the tap detection threshold are closely related.

Although the two queue sizes represents the time period for hand motion tracking, they are dependent on the frames per second (fps) of the front camera of the user's mobile device. For the same queue sizes set, the time period will be shorter if the fps of the camera is higher, and vice versa.

3.3. Key Input Determination

Since only the front facing camera of a smart phone is used, the camera's depth information cannot be retrieved. The world coordinates of the fingertips cannot be determined by stereo vision or camera calibration. When using the app, the smart phone must be placed perpendicularly on the tapping surface in front of the user in the landscape orientation. The front facing camera will be viewing a plane that is perpendicular to the tapping surface. Although the horizontal position on the keyboard can be obtained from the projected coordinates easily, the depth information of the camera cannot be determined, which represents the row on the keyboard layout. Therefore, an alternative solution is needed to determine the row of the keyboard layout the user's finger is tapping to generate a correct text input.

This chapter explains the alternative methods to determine the correct text input on detecting a tapping finger. Chapter 3.3.1 introduces the initial proposed solutions and the experimental result. Chapter 3.3.2 to Chapter 3.3.6 describes the method adopted in the actual application to determine the key input for each finger based on the relationship between the projected coordinates of the tapping fingertip and the reference positions.

3.3.1. Initial Proposed Solutions

The first proposed solution is to determine the row of the keyboard by the projected size of hands on the captured image. It is suggested that the projected size of the hand will be larger if the user is tapping on the top row of the keyboard, as the top row of the keyboard is closer to the front facing camera. It is assumed that the reference positions are representing the position of keys on the middle row of the keyboard. The reference size of the hands tapping on the middle row of the keyboard row is recorded. Then, on a tap detected, the projected size of the tapping hand is compared against the reference size.

However, in actual testing, it is found that the change in projected size of hand is too small when fingers of the hand is tapping on keys on different rows on the keyboard. It is because the difference in distance between the rows on the keyboard layout is small. In addition, the palms would be staying stable on the tapping surface. Thus, the size of hand does not change significantly throughout the typing process.

An improved method to determine the tapping row on the keyboard is by comparing the projected size of individual fingers instead of comparing the size of the hand. The size of a finger can be retrieved by calculating the distance between the landmarks. Yet, the change in size of fingers tapping on different rows on the keyboard is still too small, because of the small distance between different rows on the keyboard. In addition, the orientation of the fingers are not constant when tapping on different keyboard rows. For instance, the index fingers are more flattened when tapping on the top row and are more upright when tapping on the bottom row. The different orientation of fingers would make distance comparison inconsistent.

By observing the difference in the orientation of fingers when tapping on different keys, another method for determining the tapping row on the keyboard is suggested. This method checks the orientation of fingers (flattened or upright) by comparing the slope between the projected landmarks on the finger. A flattened finger would have a smaller slope and an upright finger would have a greater slope. However, since the fingers are perpendicular to the camera's viewing plane, the actual distance between landmarks in the finger cannot be projected on the captured frame. The connection line between the projected finger landmarks will appear to be straight lines in the captured frame, especially for the fingers near to the center of the frame. The orientation of fingers cannot be distinguished.

3.3.2. Thumb Input Determination

According to the finger positioning, the thumbs on both hands are only responsible for the spacebar on the keyboard. Thus, when the thumb is detected as tapping, the key input will be determined as spacebar directly.

3.3.3. Index Finger Input Determination

The key input determination for the index finger is the most complex, as the index fingers are responsible for two columns of keys on the keyboard layout. The left index finger is responsible for the keys on the fourth and fifth column in the keyboard layout, including the keys "r", "f", and "v" on the forth column, and keys "t", "f" and "b" on the fifth column. The right index finger is responsible for the keys on the sixth and seventh column in the keyboard layout, including the keys "y", "h" and "n" on the sixth column, and keys "u", "j" and "m" on the seventh column. Thus, in addition to determining the keyboard row the index finger is tapping, it is also needed to determine the keyboard column the index finger is tapping.

The first step is to determine the keyboard column the index finger is tapping on. It is done by comparing the horizontal projection of the tapping index finger with a certain value. For the left index finger, if the horizontal projection of the fingertip is less than the value, the tapping keyboard column will be the fourth column. For the right index finger, if the horizontal projection of the fingertip is greater than the value, the tapping keyboard column will be the seventh column. In the actual implementation, the value is set as the horizontal coordinate of the left index finger's reference position plus 3% of the image width for the left index finger, and is the horizontal coordinate of the right index finger's reference position minus 3% of the image width for the right index finger.

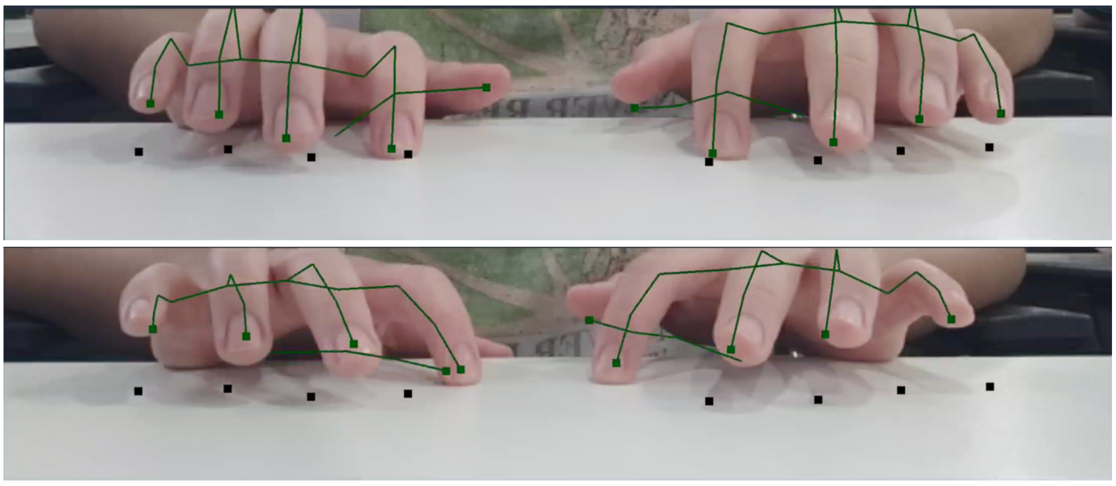


Figure 6. Index finger tapping on different keyboard columns. The upper picture shows the index finger tapping on the fourth and the seventh column. The bottom image shows the index fingers tapping on the fifth and the sixth column

When the index finger is tapping on the fourth or the seventh column (the inner column), the tapping keyboard row is determined by the comparison of the horizontal distance between the projected fingertip of the middle finger and the reference position to the index finger, the middle finger, and the ring finger. It is observed that when the index finger is tapping on the top keyboard row, the fingers will become more spread on the projection. And when the index finger is tapping on the bottom keyboard row, the fingers will become more congregate on the projection. Therefore, comparing the horizontal distance between the projected fingertip of the middle finger and the reference positions for neighboring fingers is a valid measure of the spread of the fingers.

If the horizontal distance between projection of the fingertip of the middle finger to the reference position of the ring finger is the shortest, the index finger is determined as tapping on the top row, which is the “r” key for the left index finger and the “u” key for the right index finger.

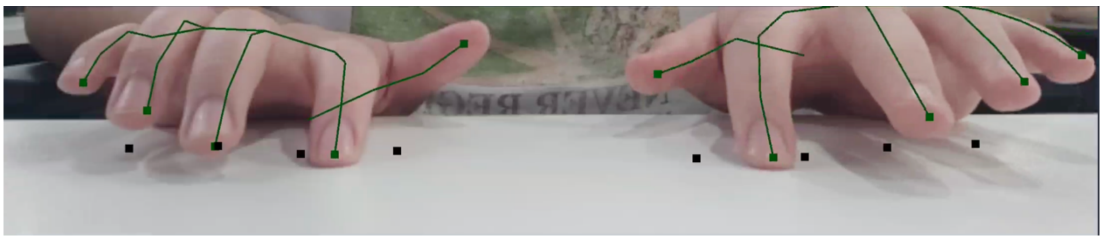


Figure 7. Index finger tapping on the top row of the inner column

If the horizontal distance between projection of the fingertip of the middle finger to the reference position of the index finger is the shortest, the index finger is determined as tapping on the bottom row, which is the “v” key for the left index finger and the “m” key for the right index finger.

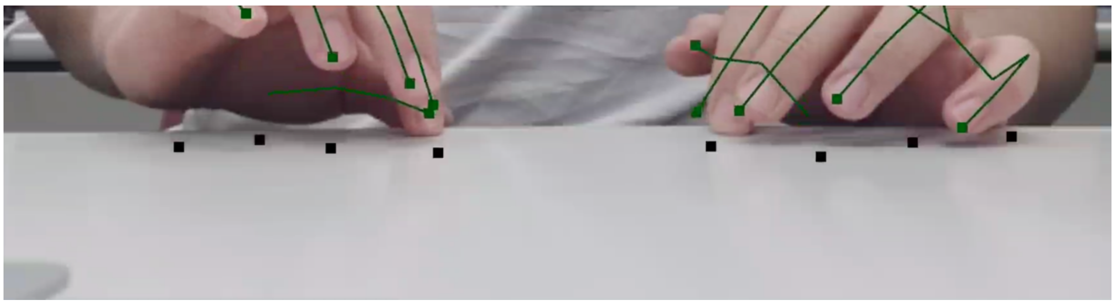


Figure 8. Index fingertip tapping on the bottom keyboard row of the inner column

If both conditions are not satisfied, the index finger is defined as tapping on the keys on the middle keyboard row, which is the “f” key for the left index finger and the “j” key for the right index finger.

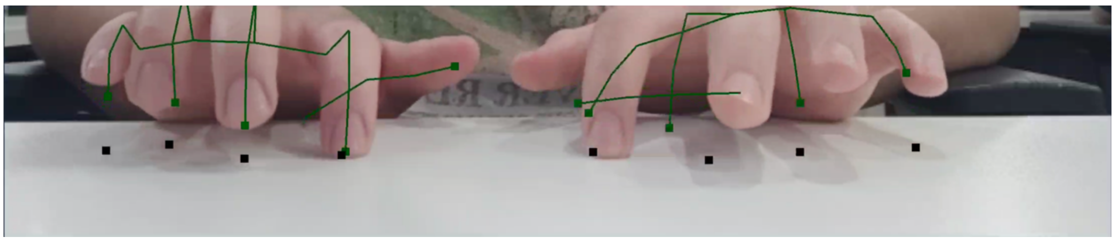


Figure 9. Index finger tapping on the middle row in the inner column

When the index finger is tapping on the fifth or the sixth column (the outer column), the tapping keyboard row is determined by the relationship between the projected fingertip of the pinky finger and the reference position of the pinky finger and the ring finger.

When the horizontal projection of the pinky fingertip is outside the range of all reference coordinates (projected to the left of left pinky’s reference coordinate and projected to the right of the right pinky’s reference coordinate), the index finger is considered tapping on the top keyboard row. That is, the “t” key for the left index finger and the “y” key for the right index finger.

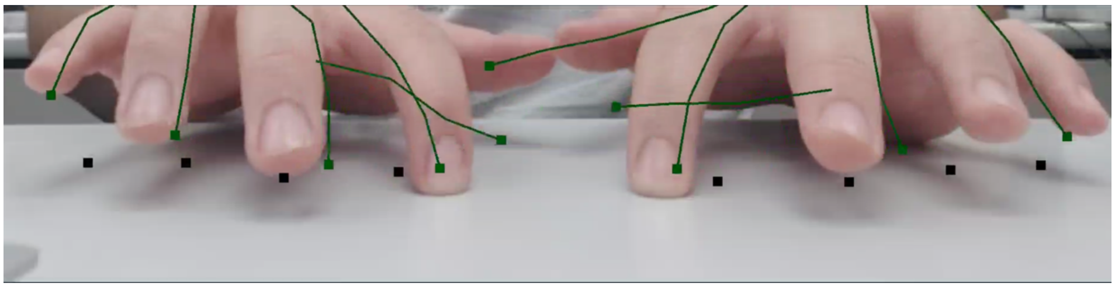


Figure 10. Index finger tapping on the top keyboard row of the outer column

Otherwise, the horizontal distance between the projection of the pinky fingertip and the reference position of the ring finger and the pinky finger are compared. If the horizontal distance between the projection of the pinky fingertip and the reference position of the ring finger is shorter, the index finger is considered as tapping on the bottom row, which is the “b” key of the left index finger and the “n” key of the right index finger.

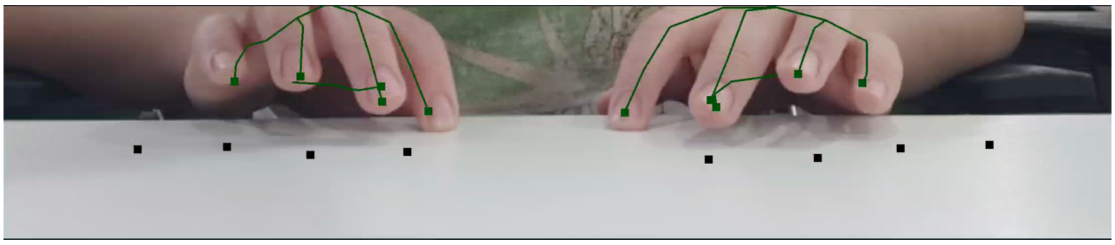


Figure 11. Index fingertip tapping on the bottom keyboard row of the outer column

The decision tree below shows the decision of the tapping keyboard row of the index finger.

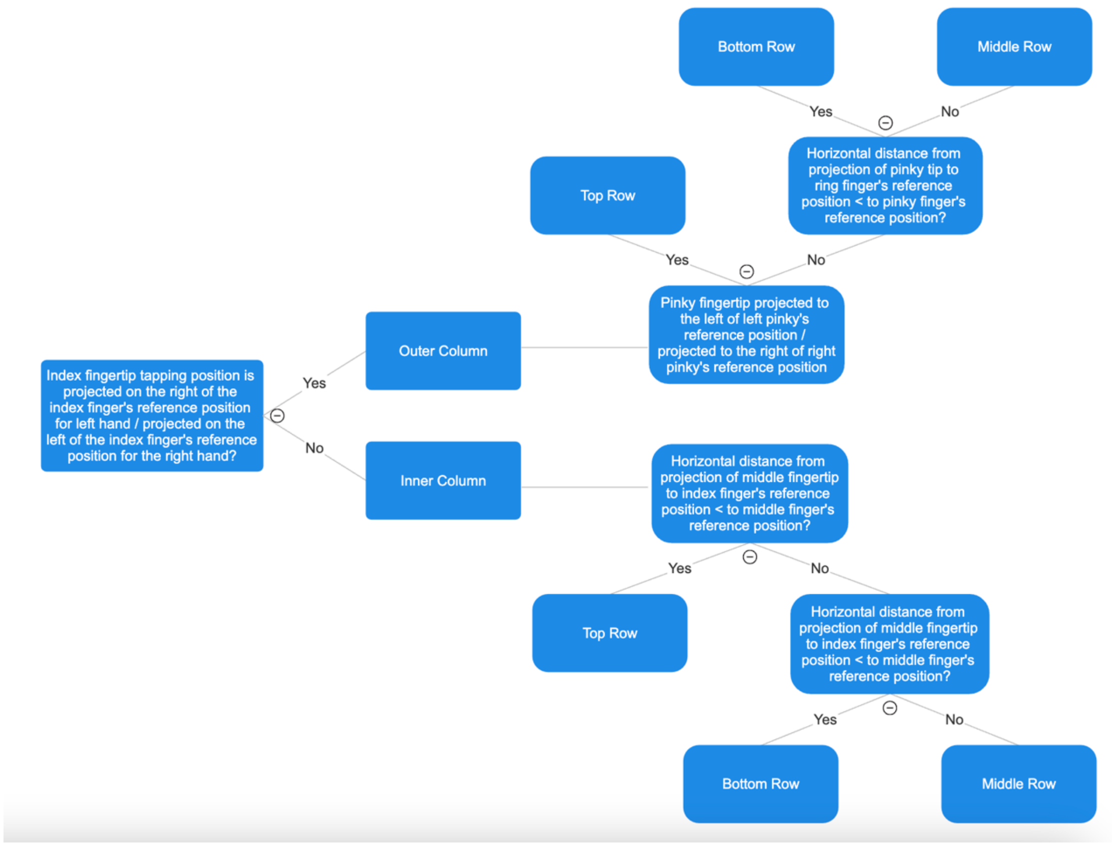

Bottom Row
Middle Row
Yes
No
Horizontal distance from projection of pinky tip to ring finger's reference position < to pinky finger's reference position?
Top Row
Yes
No
Pinky fingertip projected to the left of left pinky's reference position / projected to the right of right pinky's reference position
Outer Column
Index fingertip tapping position is projected on the right of the index finger's reference position for left hand / projected on the left of the index finger's reference position for the right hand?
Yes
No
Inner Column
Horizontal distance from projection of middle fingertip to index finger's reference position < to middle finger's reference position?
Yes
No
Top Row
Horizontal distance from projection of middle fingertip to index finger's reference position < to middle finger's reference position?
Yes
No
Bottom Row
Middle Row

Figure 12. Decision tree to determine the tapping keyboard row of index finger

3.3.4. Middle Finger Input Determination

The middle finger on the left hand is responsible for the keys on the third column in the keyboard layout, including the “e”, “d” and “c” key. The right middle finger is responsible for the keys on the third last column in the keyboard layout, including the “i”, “k” and “comma” key.

By observation, the horizontal projection of the tapping position of the middle finger varies when it taps on different keyboard rows, due to the trapezoid layout of the alphabet keys on a traditional keyboard layout. Thus, the projection of the tapping position of the middle finger can be compared with the reference position of the index finger, the middle finger, and the ring finger.

If the horizontal distance between the projection of the tapping position of middle finger to the reference position of the ring finger is the shortest, the middle finger is determined as tapping on the top row, which is the “e” key for the left middle finger and the “i” key for the right middle finger.

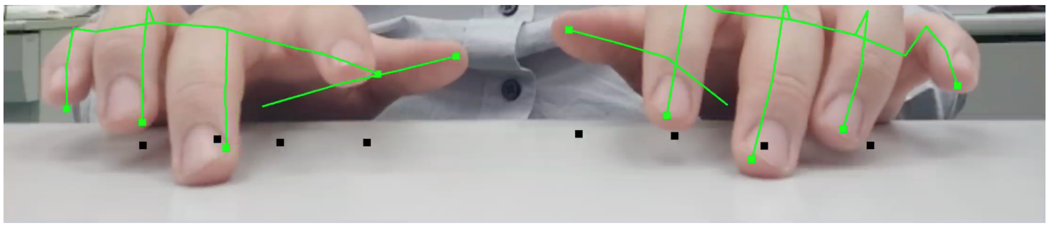


Figure 13. Middle finger tapping on the top keyboard row

If the horizontal distance between the projection of the tapping position of middle finger to the reference position of the index finger is the shortest, the middle finger is determined as tapping on the bottom row, which is the “c” key for the left middle finger and the “comma” key for the right middle finger.

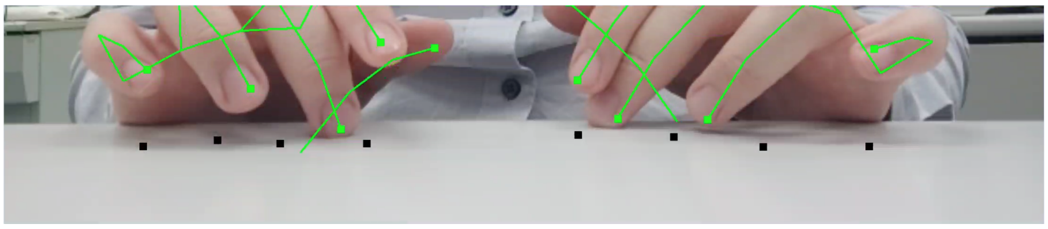


Figure 14. Middle finger tapping on the bottom keyboard row

If both conditions are not satisfied, the middle finger is defined as tapping on the keys on the middle keyboard row, which is the “d” key for the left middle finger and the “k” key for the right middle finger.

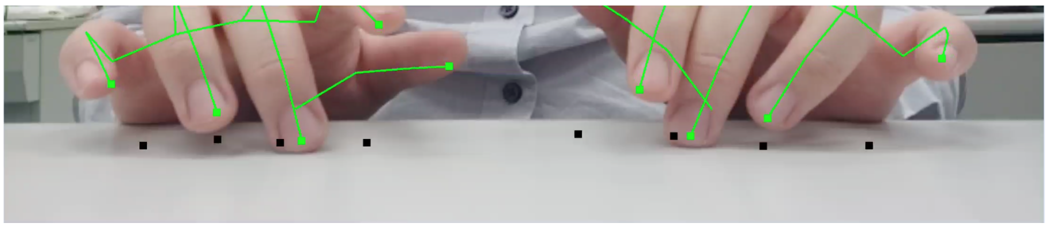


Figure 15. Middle finger tapping on the middle keyboard row

The decision tree below shows the decision of the tapping keyboard row of the middle finger.

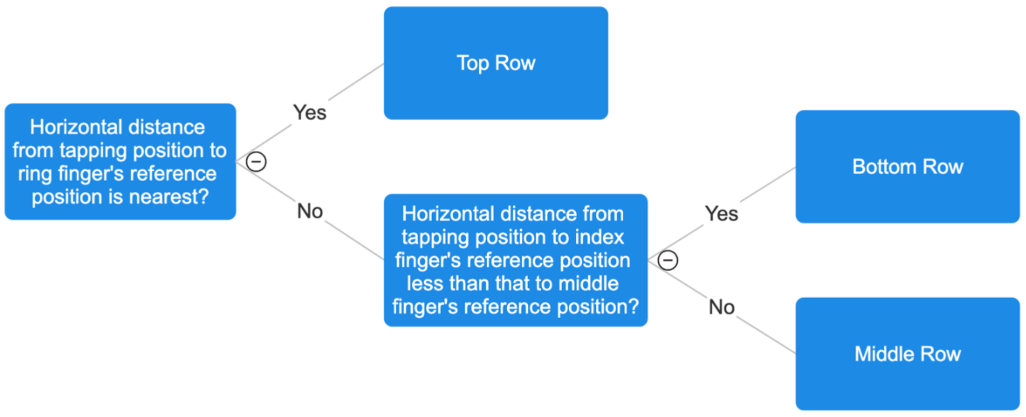

Horizontal distance from tapping position to ring finger's reference position is nearest?
Yes
Top Row
No
Horizontal distance from tapping position to index finger's reference position less than that to middle finger's reference position?
Yes
Bottom Row
No
Middle Row

Figure 16. Decision tree to determine the tapping keyboard row of middle finger

3.3.5. Ring Finger Input Determination

The ring finger on the left hand is responsible for the keys on the second column in the keyboard layout, including the “w”, “s” and “x” key. The right ring finger is responsible for the keys on the second last column in the keyboard layout, including the “o”, “l” and “period” key.

Similar to the middle finger, the determination of the tapping keyboard row for the ring finger depends on the horizontal distance between the projection of tapping position and the reference positions of the middle finger, the ring finger and the pinky finger.

If the horizontal distance between the projection of the tapping position of ring finger to the reference position of the pinky finger is the shortest, the ring finger is defined as tapping on the keys on the top keyboard row, which is the “w” key for the left ring finger and the “o” key for the right ring finger.

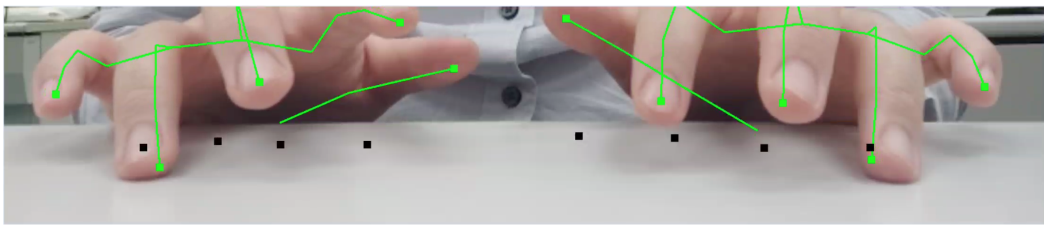


Figure 17. Ring finger tapping on the top keyboard row

If the horizontal distance between the projection of the tapping position of ring finger to the reference position of the middle finger is the shortest, the ring finger is defined as tapping on the keys on the bottom keyboard row, which is the “x” key for the left ring finger and the “period” key for the right ring finger.

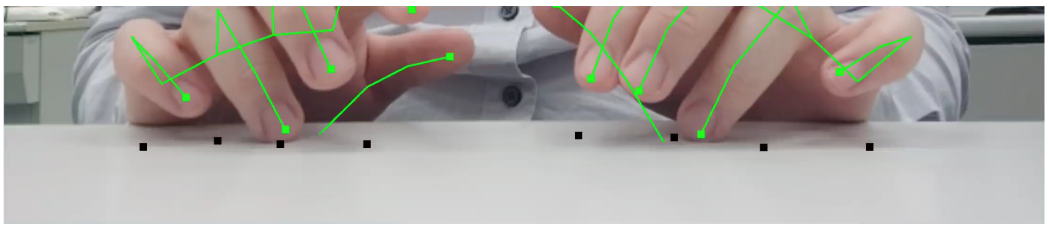


Figure 18. Ring finger tapping on the bottom keyboard row

If both conditions are not satisfied, the ring finger is defined as tapping on the keys on the middle keyboard row, which is the “s” key for the left ring finger and the “l” key for the right ring finger.

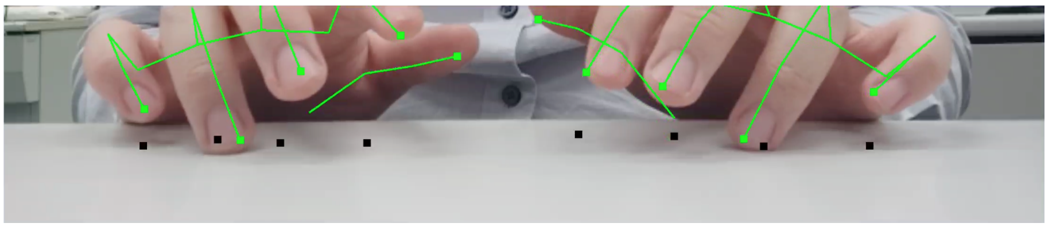


Figure 19. Ring finger tapping on the middle keyboard row

The decision tree below shows the decision of the tapping keyboard row of the ring finger.

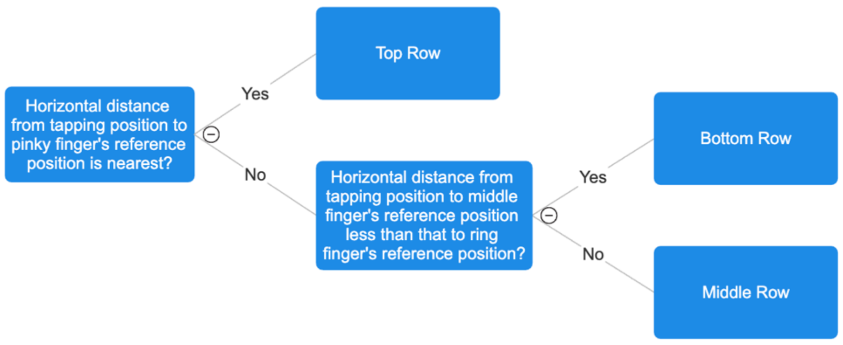

Horizontal distance from tapping position to pinky finger's reference position is nearest?
Yes
Top Row
No
Horizontal distance from tapping position to middle finger's reference position less than that to ring finger's reference position?
Yes
Bottom Row
No
Middle Row

Figure 20. Decision tree to determine the tapping keyboard row for ring finger

3.3.6. Pinky Finger Input Determination

The pinky finger on the left hand is responsible for the keys on the first column in the keyboard layout, including the “q”, “a” and “z” key. The right pinky finger is responsible for the keys on the last column in the keyboard layout, including the “p”, “enter” and “backspace” key.

When the pinky finger is tapping on the top keyboard row (key “q” on the left hand and key “p” on the right hand), it is observed that the pinky tip will exceed the boundary of all the reference positions. Thus, if the projection of left pinky’s tapping position is on the left of the left pinky finger’s reference position, the key input will be determined as the “q” key on the top row. Similarly, if the projection of the right pinky’s tapping position is on the right of the right pinky finger’s reference position, the key input will be determined as the “p” key. In addition, the distance between the projection of the tapping pinky tip and the pinky’s reference position must exceed a certain threshold. The current value of the threshold is 5% of the image width.

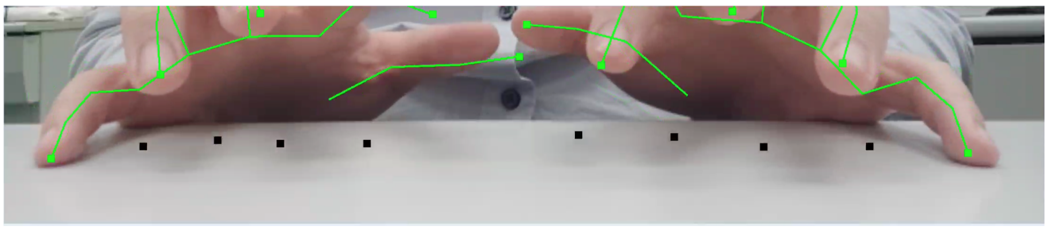


Figure 21. Pinky finger tapping on the top keyboard row

When the pinky finger is tapping on the bottom keyboard row (key “z” on the left hand and key “backspace” on the right hand), it is observed that the nearest reference position to the projected pinky tip is the ring finger’s reference position for both hands. Therefore, if the pinky of either hand taps on a position which its projection is horizontally nearer to the ring finger’s reference position than the pinky finger’s reference position, the row of key input will be determined as the bottom row. The key input will be the “z” key for the left pinky and the “backspace” key for the right pinky finger.

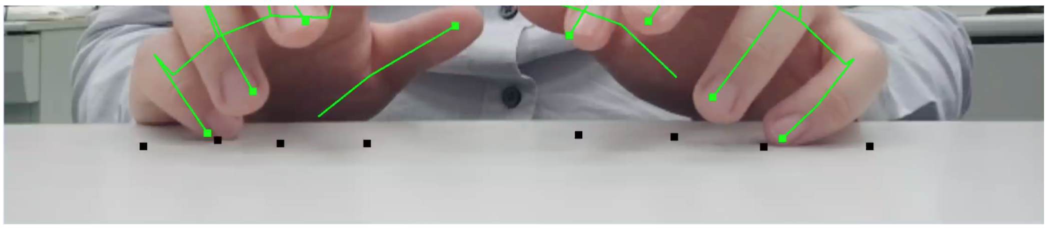


Figure 22. Pinky finger tapping on the bottom keyboard row

If neither of the two conditions are satisfied, the tapping position of the pinky must be nearest to the reference position of the pinky finger, which represents the pinky finger is tapping on the middle row of the keyboard.

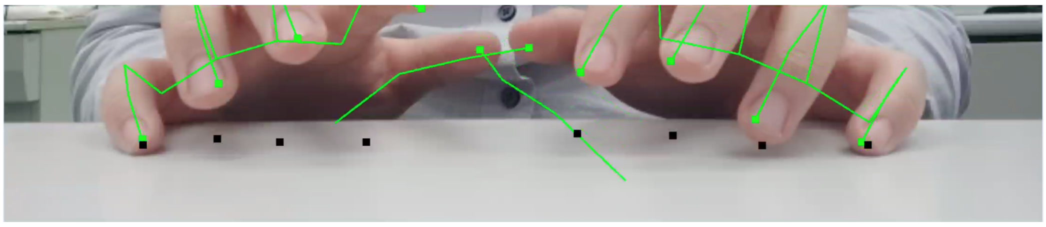


Figure 23. Pinky finger tapping on the middle keyboard row

The decision tree below shows the decision of the tapping keyboard row of the pinky finger.

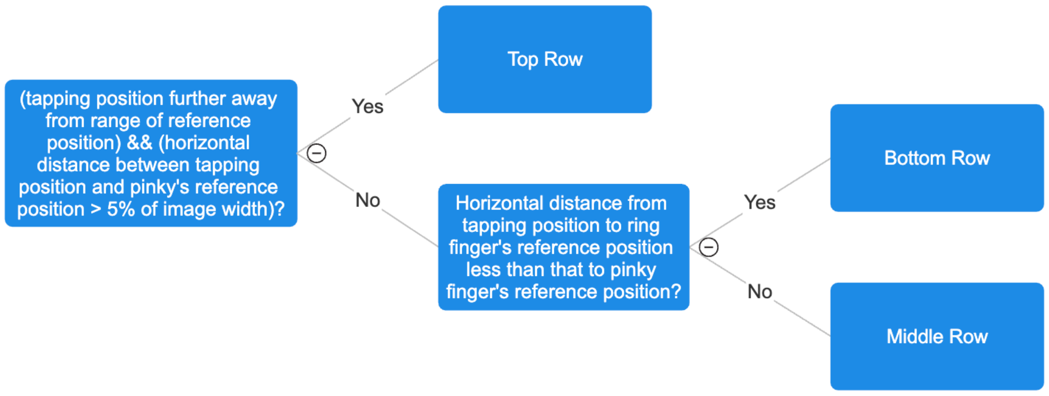
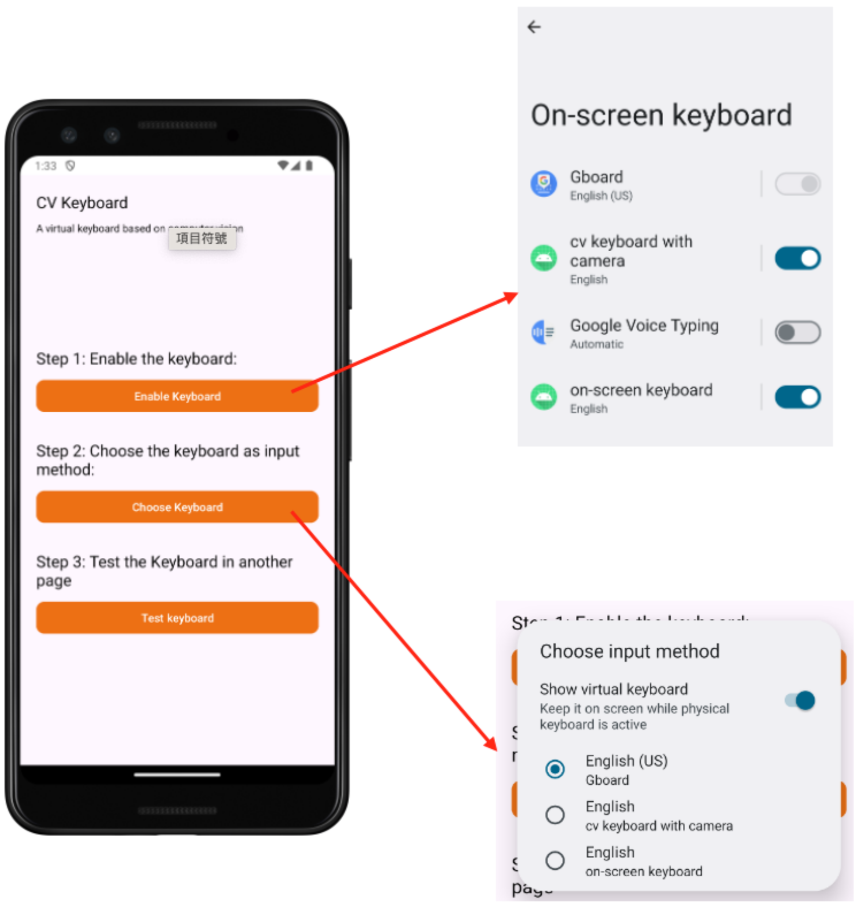

(tapping position further away from range of reference position) && (horizontal distance between tapping position and pinky's reference position > 5% of image width)?
Yes
Top Row
No
Horizontal distance from tapping position to ring finger's reference position less than that to pinky finger's reference position?
Yes
Bottom Row
No
Middle Row

Figure 24. Decision tree to determine the tapping keyboard for pinky finger

4. Results

4.1. Welcome Page


1:33
CV Keyboard
A virtual keyboard based on
項目符號
Step 1: Enable the keyboard:
Enable Keyboard
Step 2: Choose the keyboard as input method:
Choose Keyboard
Step 3: Test the Keyboard in another page
Test keyboard

On-screen keyboard

- Gboard, English (US): off
- cv keyboard with camera, English: on
- Google Voice Typing, Automatic: off
- on-screen keyboard, English: on

Choose input method

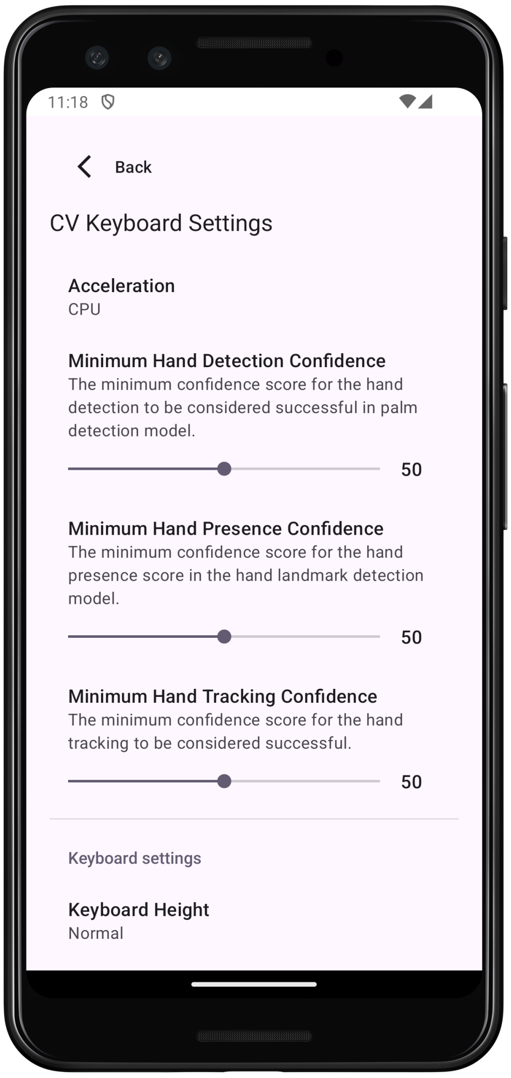
Show virtual keyboard: on
Keep it on screen while physical keyboard is active

- [✓ English (US) Gboard]
- English cv keyboard with camera
- English on-screen keyboard

Figure 25. Welcome page and functions of buttons

After a user has installed and opened the app, a welcome page is shown on the screen. Users can set up the keyboard in their device according to the step-by-step guide. Firstly, users need to enable the cv keyboard service on their device. The button in step 1 will direct user to the settings page of on-screen keyboard. Secondly, users need to set the cv keyboard service as current input method on their device by clicking on the button in step 2. The button on step 3 will direct users to another page with text input fields for testing the keyboard.

4.2. Preference Page


11:18

Back

CV Keyboard Settings

Acceleration
CPU

Minimum Hand Detection Confidence
The minimum confidence score for the hand detection to be considered successful in palm detection model.
50

Minimum Hand Presence Confidence
The minimum confidence score for the hand presence score in the hand landmark detection model.
50

Minimum Hand Tracking Confidence
The minimum confidence score for the hand tracking to be considered successful.
50

Keyboard settings

Keyboard Height
Normal

Figure 26. Preference Page

Users can adjust the configuration of the hand detection model and the keyboard. The configuration options for the hand detection model are the parameters used to setup the model. The configuration options for the keyboard includes the keyboard height, enabling / disabling word suggestions and auto capitalization.

4.3. Keyboard Layout

Before using the keyboard, users need to ensure that the orientation of their device is in landscape. It is because the front facing camera can hardly include both the users' hands and the tapping surface to the scope of vision due to the long distance between the camera and the tapping surface. Because of the limitation on the device orientation, this keyboard app can only be used for input fields that the parent activity supports landscape orientation. Users need to ensure the current app satisfy this requirement.

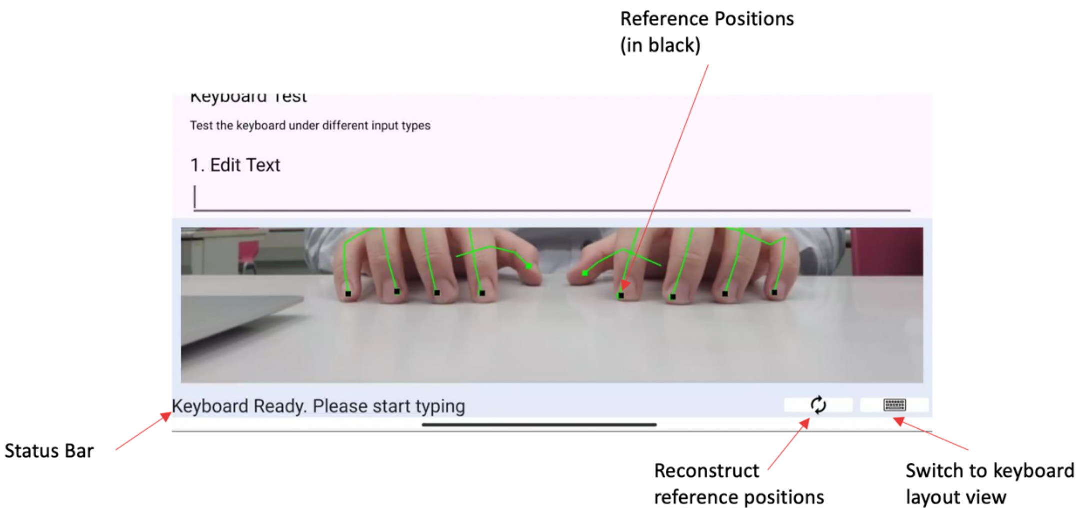

Reference Positions (in black)
Keyboard Test
Test the keyboard under different input types
1. Edit Text
Keyboard Ready. Please start typing
Status Bar
Reconstruct reference positions
Switch to keyboard layout view

Figure 27. Keyboard layout with camera preview

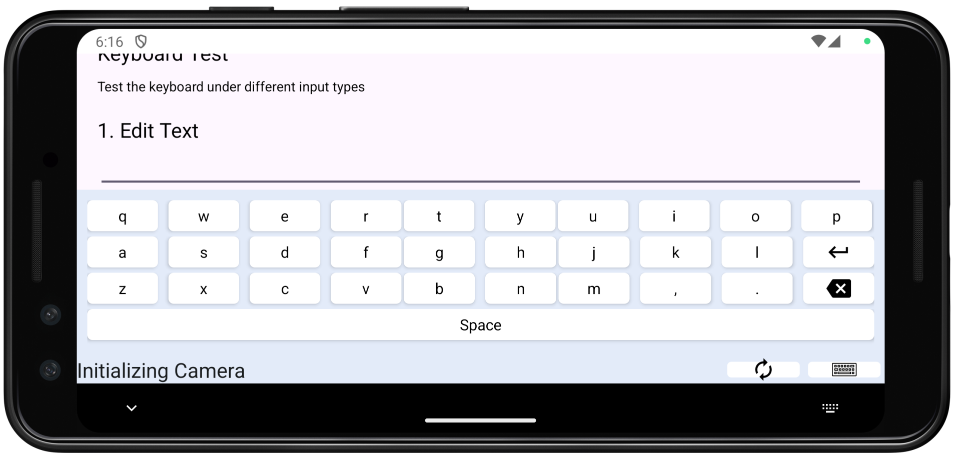


Figure 28. Keyboard layout with keyboard view

There are two layouts for the keyboard. The first one is the default layout with camera preview (Figure 8). The camera preview will be shown once the hand landmark detection model is initialized. The status bar below the camera preview indicates the status of the camera and the status of reference positions construction. Before the camera is ready, the text on the status bar will be “initializing camera”. After the camera is ready and the app requires to construct the reference positions, the text on the status bar will become “Generating Keyboard. Please stay your fingers still”. After the reference positions are constructed, the text on the status bar will become “Keyboard ready. Please start typing”. Users can start typing at this moment.

The second layout comes with the keyboard view instead of the camera preview. It can be shown when the user press the “switch keyboard layout view” button on the right bottom corner (shown in Figure 8). This layout enables users to check the finger positioning and the position of keys on the keyboard. When this layout is enabled, the camera and the hand landmark detection model are still running in the background. The camera preview and the results of the hand landmark detection are set to invisible.

4.4. Candidate View

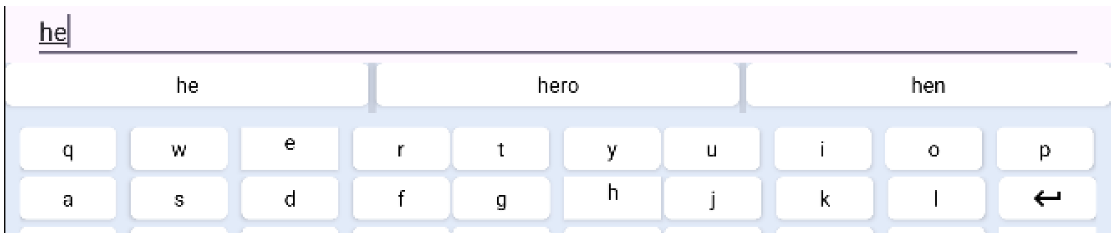


Figure 29. Candidate View

When a user enables the “word suggestions” preference, word suggestions will be provided if the user is inputting text in a general text field. Note that for security reason, word suggestions will not be provided if the input field contains sensitive information like passwords. The characters input by the user will be passed to the natural language processor model. If the model returns any word suggestions, the first three suggestions will appear in the candidate view, which is located above the keyboard layout (see Figure 29). If the user would like to pick the suggested word, they have to touch on the suggested word on the screen. If there is no word suggestion or word suggestion is disabled, the candidate view is invisible.

5. Challenges and Limitations

This section introduces some challenges encountered during the development of the app, and discuss the limitations on using the app.

5.1. Tap Detection Algorithm

As described in Chapter 2.5, the current tap detection algorithm only depends on the vertical motion of fingertips. The short coming of this algorithm is that it is independent to the tapping surface. Current algorithm cannot tell if a moving down finger has touched the tapping surface or not. False tapping are detected when user moves the hands in the air.

There are difficulties in building a relationship between the fingertips and the tapping surface, as the front camera of the user's device is not perfectly aligned with the tapping surface. The front camera is at a certain distance above the tapping surface. Thus, the tapping surface is projected as an area on the captured image. There is no reference level for determining the touch event on the tapping surface.

The current solution is to add an additional check to the distance between the projected coordinates of the tapping fingertip and the average vertical coordinate of the reference positions of the corresponding hand in tap detection. As a result, tapping is only detected when the tapping position of a fingertip is around the level of the hand's reference position. This helps to reduce the false tapping detected for moving the hand in the air. However, this check still cannot prevent the false tap detection at the position which its projection is near the reference positions.

5.2. Key Input Determination Method

As mentioned in Chapter 3.4, the key input cannot be determined directly from the projected coordinates of the tapping fingertip. The current alternative method was deduced by generalizing the observations by the author. It may not work satisfactory for some parties, as the typing behavior is different among individuals. For instance, a strict rule on finger positioning is set for this vision-based keyboard app, but it is believed that not every individual would follow this finger positioning exactly in daily typing tasks.

In addition to the challenge of single camera, an additional challenge is to deduce a generalized key input determination method that is satisfied by most people on the vision-based keyboard. Unlike an on-screen virtual keyboard that contains a definite boundary for keys on the keyboard, an invisible vision-based keyboard requires the user to memorize the relative positions and order of keys, as well as to type according to muscle memories of typing on a traditional physical keyboard. A suggested solution to overcome the challenge is to build a machine learning model which learns the typing behavior of a specific user. As a result, the keyboard would be optimized custom keyboard app for each user. This solution would be one of the future work of this project.

5.3. Device Performance

Due to the variety of Android devices, the hardware performance difference across devices can be large. The vision-based keyboard requires a high-level hardware performance, including the front facing camera and CPU/GPU. Since the keyboard app mainly uses the front-facing camera to capture finger movements, the resolution of the front camera must be high enough for the hand detection model to detect the presence of hands. In addition, the frames per second (fps) of the front facing camera is also important for motion tracking. For the calculation unit, a great number of operations to the CPU/GPU will be required by the hand landmark detection model. Since both the front facing camera and the CPU/GPU are demanding throughout the typing process, the power consumption of the app is expected to be large. Therefore, although there are no restrictions set for running the keyboard app on low-end Android devices, the performance of the keyboard app is expected to be poor on those devices. Therefore, devices with high hardware performances and good quality camera are preferred.

6. Future Work

Most proposed features are implemented in the app. However, there are still possible improvements to improve the user experience of the keyboard app. First, auto correction and spell-checking can be added to the app. This would greatly reduce the compact of inaccurate tap detection and key input determination. Second, a better word suggestion model can be developed. The word suggestion model provides customized word suggestions by building a custom dictionary for each user.

Moreover, the model used for hand detection can be replaced by a better model that is designated for detecting hands in typing gesture, so that the performance of hand detection can be improved. Furthermore, addition models can be added to detect hand gestures like finger tapping, finger swiping and open palm. Not only the accuracy for tap detection can be increased, but the additional supported gestures can be utilized for additional keyboard control, such as choosing the word suggestions presented in the candidate view and clear all text in the text input field.

7. Conclusion

This paper has presented a method to solve the difficulty of texting on small mobile devices by developing a virtual keyboard app using CV on the Android platform. The app generates text input by analyzing users' finger movements captured by the front facing camera on their mobile devices. It provides an intuitive and convenient text input method on mobile devices.

The major challenge of the project is detecting finger tap detection and translating the projection of tap position into correct key input on the keyboard layout. As only the front facing camera of the user's device is used as sensor for finger motion tracking, it is hard to determine the row on the keyboard the user's finger is tapping on. Alternative methods on determining the key input introduced in this paper was deduced from observations by the author. However, there are limitations on the accuracy of the alternative methods. Furthermore, there are limitations on the requirement of hardware performance of the running device.

Despite the strict limitations of the project, it is believed that vision-based controllers still transcend traditional hardware controllers on the extent of customizability, extendibility and portability. Therefore, an important avenue for future work is to rebuild the app with using more cameras. The limitations in this project can be resolved by more advanced computer vision technologies such as stereo vision. It is hoped that there will be more innovative vision-based controllers in the future.

8. References

[1] *"A Computer In Your Pocket: The Rise of Smartphones"*. Science Museum UK. Accessed: Apr 26, 2024 [Online]. Available: https://www.sciencemuseum.org.uk/objects-and-stories/computer-your-pocket-rise-smartphones

[2] I. Bouchrika. *"Mobile vs Desktop Usage Statistics for 2023"*. Research.com. Accessed: Apr 26, 2024 [Online]. Available: https://research.com/software/mobile-vs-desktop-usage

[3] R. Lay and A. Stanford. "*More and more people are using their smartphones for work"*. Deloitte. Accessed: Apr 26, 2024 [Online]. Available: https://www2.deloitte.com/ch/en/pages/technology-media-and-telecommunications/articles/immer-mehr-menschen-arbeiten-auf-dem-smartphone.html

[4] *"Microsoft SwiftKey Keyboard"*. Microsoft. Accessed: Apr 26, 2024 [Online]. Available: https://www.microsoft.com/en-us/swiftkey?activetab=pivot_1:primaryr2

[5] Samsung Newsroom. "How C-Lab is Preparing for a Future Full of Potential – Part 1: C-Lab Inside". Accessed Apr 26, 2024 [Online]. Available: https://news.samsung.com/global/how-c-lab-is-preparing-for-a-future-full-of-potential-part-1-c-lab-inside

[6] *"Download Android Studio & App Tools – Android Developers"*. Google. Accessed: Apr 26, 2024 [Online]. Available: https://developer.android.com/studio

[7] *"Kotlin and Android | Android Developers"*. Google. Accessed: Apr 26, 2024 [Online]. Available: https://developer.android.com/kotlin

[8] Google Developers. "Create an input method". Google. Accessed: Apr 26, 2024

[Online]. Available: https://developer.android.com/develop/ui/views/touch-and-input/creating-input-method#GeneralDesign

[9] *"MediaPipe | Google For Developers"*. Google. Accessed: Apr 26, 2024 [Online]. Available: https://developers.google.com/mediapipe

[10] *"Hand landmarks detection guide"*. Google. Accessed: Apr 26, 2024 [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

[11] mccorby. "*Machine Learning*". Github repository. Accessed Apr 26, 2024 [Online]. Available: https://github.com/mccorby/MachineLearning/tree/master?tab=readme-ov-file