2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT) | 979-8-3503-6944-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/DCOSS-IoT61029.2024.00046

VKM: A Virtual Keyboard and Mouse Solution Towards a Lightweight Computing System

Divyansh Bisht*, Amitangshu Pal*, Shashwati Banerjea†
*Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, UP, India
†Computer Science and Engineering, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, UP, India
Email: *dbisht@cse.iitk.ac.in, amitangshu@cse.iitk.ac.in, shashwati@mnnit.ac.in*

***Abstract*—This paper provides a novel solution for developing a virtual keyboard and mouse (VKM) system that is easily manageable and portable. The traditional keyboards and mouse devices take up valuable desk space and are not easily customizable to different languages. On-screen keyboards and 3-D cameras are alternatives, but they also have drawbacks. Our proposed method makes use of computer vision techniques and calls for a mini-projector and a web camera as necessary hardware. The system tracks hand keypoints to detect real-time touch events and uses the Mediapipe tool to detect hands and keystrokes. The mouse functionality is also implemented by monitoring the finger hovering. Through experimentation, we show that our VKM solution can provide an accuracy of >90% for detecting the correct keystroke, with a typing speed of ∼55 letters/min.**

***Index Terms*—Virtual keyboard, user interface, computer vision, hand gesture, touch detection**

I. Introduction

Electronic devices have undergone a tremendous change over the past several decades, with ever-increasing computing power and decreasing size. In the computing world, machines have become more compact with time, where the efforts were to carry out complicated operations with the smallest equipment possible. Some sophisticated computers can combine the CPU and monitor into a single unit. However, there hasn't been a reliable replacement for the reliance on conventional keyboards and mouse devices which still take up significant desk space. Conventional keyboards also lack versatility and support a limited number of languages. However, we may get around this restriction and enable cross-lingual typing by using "projected keyboards", giving users the comfort of typing in their native tongues. While on-screen keyboards provide a partial answer, they frequently take up a sizable area on the screen.

Existing approaches and their limitation: Numerous studies have been conducted in this field, with the broad goal of developing a virtual keyboard and mouse system that does away with the requirement for physical input devices [1]–[5]. The main technology used in these earlier attempts to accomplish this goal is through the use of laser keyboards, but these devices are rather slow and error-prone [2]. Apart from laser keyboards, many of the other works use expensive devices like depth cameras, touch gloves, or sensors like the Leap Motion sensor. This makes the solutions far more expensive than typical keyboards and mouse devices; for example, the Asus Xtion depth camera used in [1] costs >$400 [6], whereas the Leap Motion sensor used in [3] costs >$100 [7]. Therefore, these solutions necessitate costly initial setups and are impractical for general deployment.

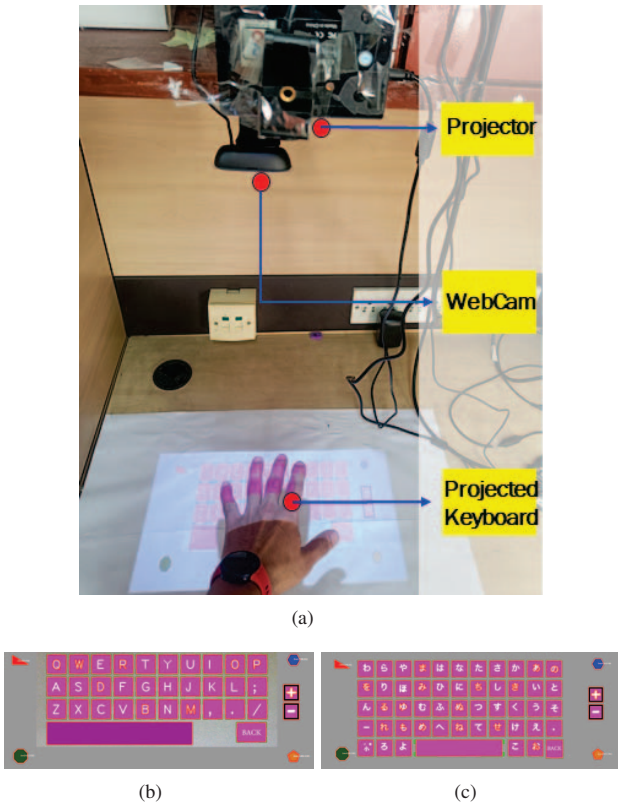


Fig. 1. (a) Our experimental setup for the Virtual keyboard. Illustration of the projected keyboard for (b) English and (c) Japanese languages.

Our contribution: In this context, we develop a *virtual keyboard and mouse (VKM)* solution – the proposed technique uses computer vision concepts and provides a more reasonably priced solution that only needs a mini-projector and a web camera, which incur an additional hardware cost of <$100. The overall model is shown in Fig. 1(a). The mini-projector projects a keyboard onto the table surface, where the user is willing to type, whereas the camera captures the hand movements. The camera inputs are then analyzed to accurately track the coordinates of hand keypoints by using the *Mediapipe* hands algorithm [8]. A *Dynamic Time Warping (DTW)* technique [9] is then used to find real-time touch

2325-2944/24/$31.00 ©2024 IEEE
DOI 10.1109/DCOSS-IoT61029.2024.00046

events and detect keystrokes, by examining the changes in the angles created by these keypoints over time. The corresponding fingertip coordinate is then used to infer the typed key. With extensive experimentation, we demonstrate that the proposed VKM solution achieves an accuracy of >90% with a typing speed of ∼55 letters/min, and even higher at a slower typing speed. VKM is also flexible enough to include various customization, like multiple languages, shapes, etc., as shown in Fig. 1(b)-(c). We also include features like mouse usage and left/right clicks in our VKM solution.

Paper organization: The paper is organized as follows. Different stages of our proposed VKM scheme is discussed in section II. Section III summarizes our detailed experimental evaluations. The paper is concluded in section IV along with some future scope for improvements.

II. Framework for our Virtual Keyboard

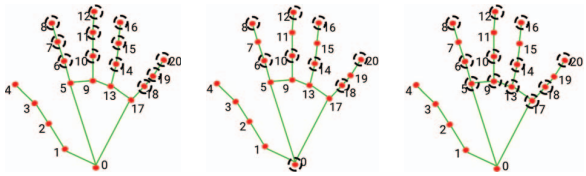
Our hardware setup consists of an external webcam and a mini-projector; we specifically use the Livato T300 Mini (800 lm / Remote Controller) Portable Projector (Black) [10] with the Logitech C270 Digital HD camera [11]. An HDMI cable is used to connect the projector to the nearby laptop or PC, and a USB connector is used to attach the webcam. White sheets are attached to the area below where the keyboard is displayed, to ensure that the user can see the projected keyboard clearly. The projector and camera are attached together and set up at a height of roughly 60 cm above the surface as shown in Fig. 1(a). Below we discuss various steps of VKM system's implementation in detail.

A. Detection of keyboard coordinates

Notice that the distance between the surface and the hardware (consisting of the camera and the projector) may change in a real setting. Therefore, to make the approach versatile, we need to make sure that the scheme works across numerous settings. To automatically find out the projected keyboard size, the four corners of our keyboard are designed with certain shapes: a triangle at top left, a hexagon at top right, an octagon at bottom left, and a pentagon at bottom right as shown in Fig. 1(b)-(c). These shapes are immediately recognizable within the image and can be located using a general shape identification technique [12] that makes use of contours, allowing our solution to acquire the coordinates for each shape. We next determine the relative coordinates of the keys with respect to these discovered shapes. When the keyboard's size changes, the coordinates are scaled accordingly. This strategy guarantees that the requisite relativity in our keyboard coordinates are preserved.

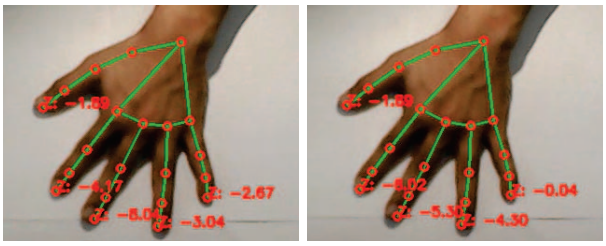
B. Detection of the hands

The user then presents his hand to the camera in the following step, during which we'll figure out how to recognize the hand and pinpoint important features like the tips of the fingers. For hand detection, we use the "MediaPipe Hands" framework [8], a transfer learning model developed on a sizable dataset made up of millions of hand samples. This



(a) (b) (c)

Fig. 2. Joints corresponding to different fingers provided by the Mediapipe tool; the figures are adapted from [8] (licensed under CC BY 4.0 DEED). The red dots show different joints, whereas the black circles around the red dots show the selected joints for (a) model-1, (b) model-2, and (c) model-3.



(a) (b)

Fig. 3. Z-axis coordinates of a hand from the same camera height in different times instances (a)-(b).

structure works well in a variety of lighting conditions and with different hand textures. For this model to work well, we need to adjust two crucial parameters: (a) the *minimum detection confidence* and (b) the *minimum tracking confidence*. Below we summarize these two tuning parameters.

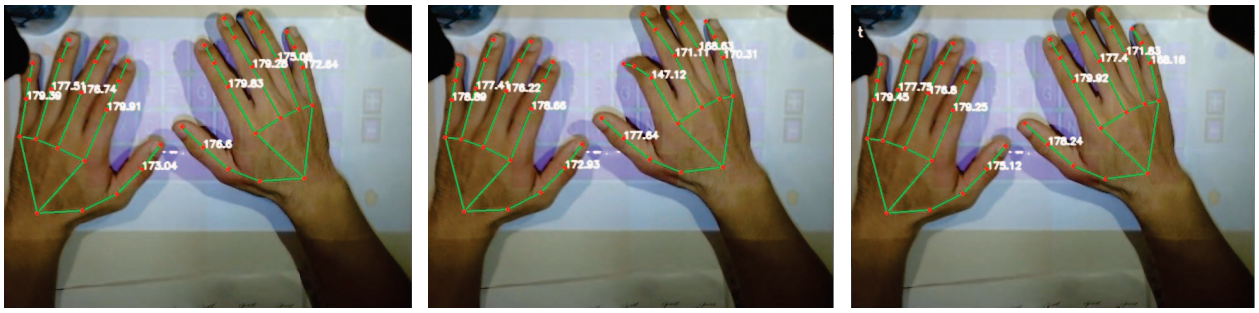
Detection confidence threshold: The minimal confidence threshold is used to identify the detection of a hand. A detected hand will not be regarded as a valid detection if its confidence score is below this limit. Raising this amount boosts the likelihood that hands will be detected correctly, but it also raises the risk that they won't be detected if the confidence ratings fall short of the higher threshold., i.e. the higher values denote a stricter confidence level.

Tracking confidence threshold: Once a hand is identified, hand tracking is used to keep track of the hand's landmarks (such as the locations of the fingertips and palm) over succeeding frames. The minimal confidence threshold is used to successfully perform the hand tracking. Increasing this threshold improves tracking accuracy but may lessen sensitivity to minute hand movements.

Both of these parameter's range vary from 0 to 1, however, can be adjusted depending on the application's needs and current circumstances. In our configuration, the minimum tracking confidence is set to 0.6, and the minimum detection confidence is set to 0.7. Depending on the camera quality, these numbers may need to be adjusted; greater values may be used for the high-quality cameras, while lower values may be suitable for the lesser-quality ones.

C. Touch detection on a surface

We now use the webcam images and perform some intuitive analysis using the MediaPipe tool to register touches. The wrist point, index fingertip, and thumb tip are just a few of



(a) (b) (c)

Fig. 4. Mediapipe finger tracking when the index finger types the letter 't'; the numbers show the angles created by joints 8, 7, and 5 (a) before, (b) during, and (c) after the touch.

the 21 critical points that the MediaPipe hands architecture generates for each hand after successfully detecting the user's hands. We examine how these landmarks enable touch recognition, i.e. when a finger is touching the surface or typing a letter. We begin by obtaining the coordinates of three finger joints because they are essential to the calculations that follow. These three joint coordinates provide angular changes during flexes while performing a touch. These angular changes must be significantly high in order to perform correct predictions. Therefore, we need to target heavy-usage finger joints and not the stationary ones.

As observed from Fig. 2, each of the five fingers has four joint points (including one for the tip), plus a fifth point for the wrist joint. We can record the x-y coordinates for "some" of these joints and can calculate the angles in between them to detect the touches. For example, in Fig. 2(a), joints 6, 7, and 8 of the index finger are chosen and the angle between 6-7 and 7-8 is calculated, whereas in Fig. 2(b), joints 0, 6 and 8 are chosen for angle calculation. Various such choices of these joints are possible; however, based on our thorough evaluations we found that the choices of Fig. 2(a)-(c) demonstrate good performance; we denote these choices as models 1-3 respectively.

Using the MediaPipe library, we determine the angles between finger joints for all three models. Notice that we did not use the z-axis coordinates for the angle calculation, as the Mediapipe z-axis coordinates are inaccurate and unstable as seen from Fig. 3. From this figure, we can observe that, even if the hand position is not changing, the z-axis coordinates are changing drastically. We therefore use the x-y coordinates to find out the angle between different finger joints. For angle calculation, we use the arctangent-based trigonometric calculations with the x-y coordinates of 3 joint points [13]. Fig. 4(a)-(c) shows how the angle between the joints 8, 7, and 5 of the index finger changes while typing 't'. This figure clearly shows that our angle calculation-based touch detection solution indeed provides a cheaper alternative to the depth camera-based approaches [1].

Fig. 5 shows the angular variations of different fingers while typing using model-2. Notice that the angles are almost always close to 180 degrees when the user places their hands

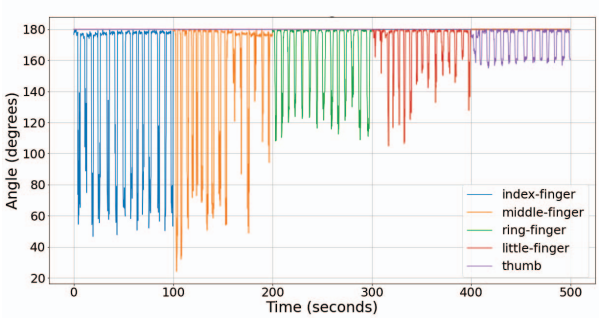


Fig. 5. Angular variation in between different finger joints in model-2.

flat above the keyboard. However, we observe that during typing, these finger angles stray from 180 degrees, drop below some threshold, and then return to 180 degrees after the touch.

Therefore, one simple technique for touch detection is to record if the angle corresponding to any finger is going beyond some threshold. However, this simple technique has its own challenges, because this threshold will greatly depend on (a) an individual's hand sizes and typing patterns, (b) the height between the camera and the surface where the keyboard is projected etc. On the other hand, from Fig. 5 we can observe that, with the same setting, the angular change is different for different fingers. In fact, we also observe that for the same finger, the angle values are different for different keystrokes; therefore, setting a uniform threshold is challenging and perhaps inappropriate. To alleviate this problem, we adopt the Dynamic Time Warping (DTW) technique [9] to register the touch.

The DTW technique is used to compare how close two temporal sequences are even if they are of different lengths or have temporal abnormalities. This method is useful when timing irregularities or fluctuations render conventional measurements like Euclidean distance or correlation inappropriate. We choose DTW especially because of the fact that different users type at different speeds; therefore, a time warping approach is essential instead of directly matching the timing sequences. For our experiments, we use the *fast* DTW approach [14], to ensure real-time performance.

In our solution, we store some known touch patterns in a dictionary for all the fingers. We then use DTW to compare

these stored temporal sequences with the ones produced by each finger at runtime, yielding a DTW score. Through experiments, we determine a DTW threshold for each finger. The two sequences are deemed to be similar if the resultant score is below the threshold, and we record it as a touch.

D. The mapping of touch coordinates to buttons

After a touch is properly detected, the next challenge is to map it to the proper coordinates and display the associated letter on the screen. To implement this, we keep a record of the last 30 touches (i.e. the x-y coordinates for all the finger joints and the angles between them) at any iteration. When a touch is detected for a finger, we extract the time instance when the angle is minimum (as this instance marks the actual touch), and record the x-y coordinates of that fingertip. Once we know the coordinates of the contacted fingertip, we can take the closest key as the key the user wants to type. The solution can also be customized to add some special keys, like the "backspace" feature as shown in Fig. 1(b)-(c).

E. Additional features

As opposed to ordinary keyboards, the VKM solution can also provide an array of extended functionalities and flexibility. For example, users can change the size of the displayed keyboard by using zoom-in (+) and zoom-out (-) buttons, as shown in Fig. 1(b)-(c). This provides some flexibility to the users to switch between larger and smaller keyboard sizes based on their individual needs. Additionally, we develop a mouse system that uses the user's wrist position to track the cursor's movement and uses the index and middle fingers to implement the mouse clicks. Furthermore, our system's adaptability makes it possible for future additions to support keyboards with different languages, forms, and design requirements. A brief demonstration of our proposed solution can be found in https://youtu.be/YPeB7Iuy-CI.

III. Experimental Evaluation

In this section, we provide an in-depth analysis of VKM models shown in Fig. 2. We first conduct a thorough analysis of the system with a single user before broadening our studies to incorporate multiple users with differing hand sizes, complexions, and typing rates. The goal is to make sure that our solution is flexible and adaptable for a wide range of users. To set the best DTW thresholds, we thoroughly experimented with all the fingers and set the best threshold for each finger.

A. Evaluation of the accuracy of each keystroke

We first evaluate the accuracy of VKM for a single user, who is asked to type all the letters on the keyboard multiple times. The user is asked to type the keys at different typing speeds; we record the corresponding hand movements and run different models to compare their accuracies.

Fig. 6 shows the typing accuracy of these models at different typing speeds. From Fig. 6 we can observe that the accuracy of VKM remains above ∼86% for all models up to a typing speed of ∼55 letters/minute, whereas for model-2

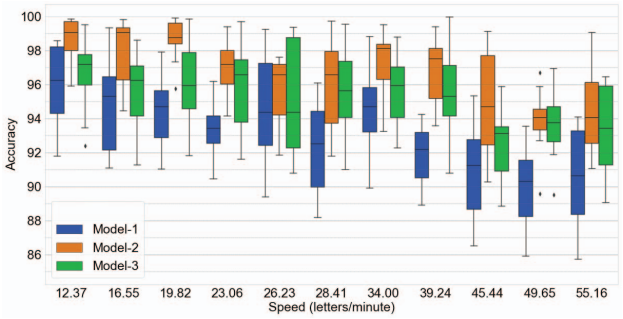


Fig. 6. Accuracy of three models in VKM at different typing speeds.

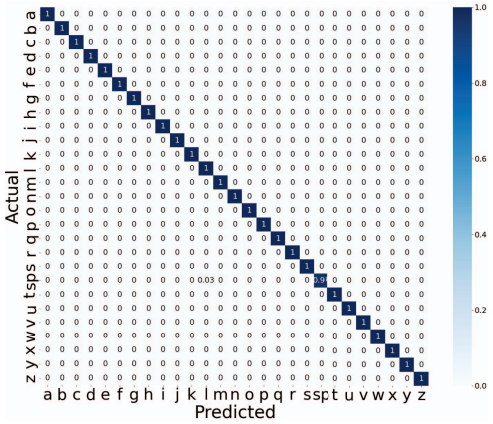


Fig. 7. The overall confusion matrix (Accuracy: ∼0.99, Precision: ∼0.99, Recall: ∼0.99, F1 Score: ∼0.99).

the accuracy remains above ∼90%. We can also observe that the accuracy across all models decreases by ∼5-6% as typing speed increases from 12 letters/minute to 55 letters/minute. Even if the faster typing results in more typos, we can observe from our results that the accuracy drop is rather modest.

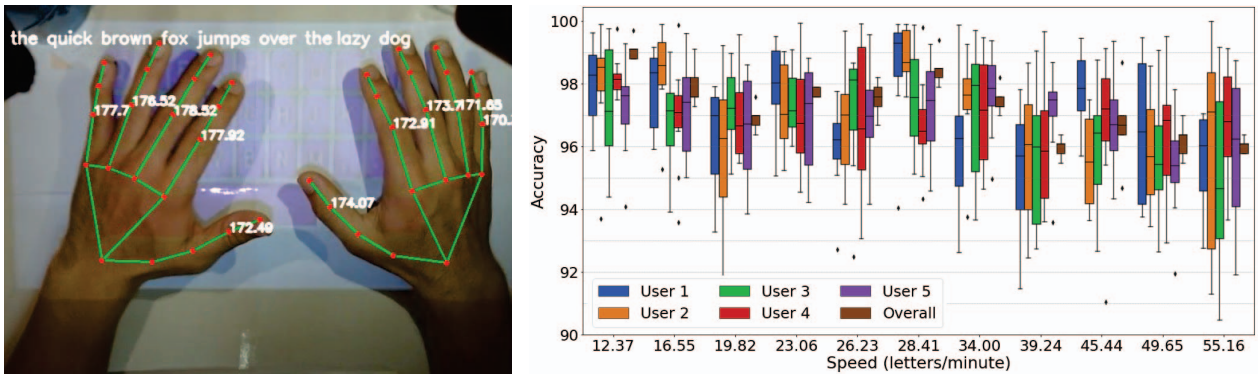
From Fig. 6, we can also observe that model-2 in general performs slightly better than the other two. This performance can be explained from Fig. 2; where we can observe that while typing, the chosen joint points in model-2 (i.e. 0, 6 and 8) demonstrate the maximum angular change as compared to the other two models. Therefore, for all the remaining results, we choose to use model-2 for the VKM performance evaluations.

B. Typing a sentence

We next test the accuracy of VKM from multiple users; we involve 5 volunteers with diverse hand sizes and skin tones. The volunteers are given instructions about the experiments and how to use the VKM solution. They are then asked to type the following sentence, consisting of 43 letters (including spaces).

> *"The quick brown fox jumps over the lazy dog."*

We chose to use this pangram so that all the letters on the keyboard are typed at least once, and the overall accuracy can be measured. Users are instructed to retype their input if the model fails to recognize their touch. Each user is allowed to



(a) (b)

Fig. 8. (a) The outcome of sentence typing. (b) Variation of accuracy corresponding with different typing speeds, while typing a paragraph.

complete his task at his own pace or typing speed. At the end, we compute the total number of additional touches each user is required to make. The accuracy is then determined by contrasting the frequency of correct predictions with the standard 43 touches.

For each user, we create a confusion matrix by comparing the expected and actual letters given by the model. The overall confusion matrix is shown in Fig. 7. From these figure we can observe that the model performs remarkably well, reaching an overall accuracy of 99.53%. Fig. 8(a) provides a snapshot when one of the users complete typing the pangram.

C. Typing a paragraph

We next evaluate the performance of VKM with the same volunteers but at different typing speeds. We ask them to type a short paragraph; the purpose is to assess how well our model is to perform real-world typing tasks. We chose the following paragraph where each letter appears at least once.

> *"On a hot summer day, a hungry lazy lion roamed the forest. He spotted a rabbit but let it go, thinking it would not satisfy his hunger. He then joyfully chased a quick deer but could not keep up. Exhausted and defeated, he returned to find the rabbit, but it was gone. The lion remained hungry and learned that greed is never good."*

Fig. 8(b) shows the accuracy for all the users. From this figure, we can observe that the accuracy drops mildly (up to ∼6%) with the increase in typing speed, however, remains above ∼90% with a typing speed of 55 letters/minute. The performance of individual users also varies up to ∼4%. The overall accuracy of all the users remains above ∼95%. These results clearly demonstrate that our proposed VKM scheme performs extremely well in identifying the keystrokes across different users, at a moderate typing speed.

IV. Conclusions and Discussion

In this paper, we demonstrate VKM for designing a projection-based virtual keyboard and mouse solution that does away with the requirement for actual input devices. The system makes use of computer vision techniques and has hardware needs of a projector and a web camera. The system shows promising adaptability, typing speed (∼55 letters/minute), and accuracy (more than 90%). The system also enables portability, versatility, and cross-lingual typing capabilities that greatly improve the user experience.

In future, we want to improve this solution by taking into account various hand sizes and shapes, increasing typing speed, and incorporating enhanced gesture detection to make the solution more interactive. In addition to that, the current VKM solution does not include detecting multiple simultaneous keystrokes (like Ctrl+Alt+Del or Ctrl+B). These enhancements are part of our future works.

References

[1] R. Xiao *et al.*, "Supporting responsive cohabitation between virtual interfaces and physical objects on everyday surfaces," *Proc. ACM Hum.-Comput. Interact.*, vol. 1, no. EICS, 2017.

[2] A. Fedor *et al.*, "Performance evaluation and efficiency of laser holographic peripherals," in *HCII*, 2021, pp. 3–16.

[3] X. Yi *et al.*, "Atk: Enabling ten-finger freehand typing in air based on 3d hand tracking data," in *ACM UIST*, 2015, pp. 539–548.

[4] P. Tan *et al.*, "Self-powered gesture recognition wristband enabled by machine learning for full keyboard and multicommand input," *Advanced Materials*, vol. 34, 04 2022.

[5] D. Gür *et al.*, "A human–computer interface replacing mouse and keyboard for individuals with limited upper limb mobility," *Multimodal Technologies and Interaction*, vol. 4, p. 84, 11 2020.

[6] "Asus," http://xtionprolive.com/asus-3d-depth-camera.

[7] "Leap motion sensor," https://www.amazon.com/Leap-Motion-Controller-Packaging-Software/dp/B00HVYBWQO/ref=psdc_229575_t1_B00E3CP9UM.

[8] "Hand landmarks detection guide," https://developers.google.com/mediapipe/solutions/vision/hand_landmarker.

[9] "Dynamic time warping," https://paperswithcode.com/method/dtw.

[10] Projector, "Livato t300 mini (800 lm / remote controller) portable projector (black)," https://brother-mart.com/products/t300-projector.

[11] Webcam, "logitech c270 digital hd camera," https://www.logitech.com/en-in/products/webcams/c270-hd-webcam.960-000584.html.

[12] GFG, "Shape detection contour algorithm," https://www.geeksforgeeks.org/how-to-detect-shapes-in-images-in-python-using-opencv/.

[13] V. Mudgal, "Real-time gesture recognition using google's mediapipe hands — add your own gestures," https://mudgalvaibhav.medium.com/real-time-gesture-recognition-using-googles-mediapipe-hands-add-your-own-gestures-tutorial-1-dd7f14169c19, 2021.

[14] W. Choi *et al.*, "Fast constrained dynamic time warping for similarity measure of time series data," *IEEE Access*, vol. 8, pp. 222 841–222 858, 2020.