Gesture Recognition as an Interactive User Interface

Jairaj Pisupati jp6712@rit.edu
Dr. Thomas B. Kinsman thomask@rit.edu

Introduction and Background

- The Gesture User Interface (GUI) proposes a new user interface that relies on gestures as a form of interaction with devices. New User Interfaces are necessary to revolutionize the way we interact with devices. This also helps the specially-abled to communicate and use devices effectively.
- This system primarily uses Deep Learning models to identify the position of different landmarks on the hand. The GUI uses computer vision and calculates gestures from a live video stream of a webcam in real-time without any additional hardware.
- Multiple data sets were used to train the model:
  1) In-the-wild dataset.
  2) In-house collected gesture dataset.
  3) Synthetic dataset.
- Environmental factors like background complexity and lighting affect the accuracy.

Future Work

- Further model development will enable the system to recognize global sign language gestures.
- This improvement will allow users to call people using their name.
- Sign Language can be used to input the person's name. The model will prompt possible names.

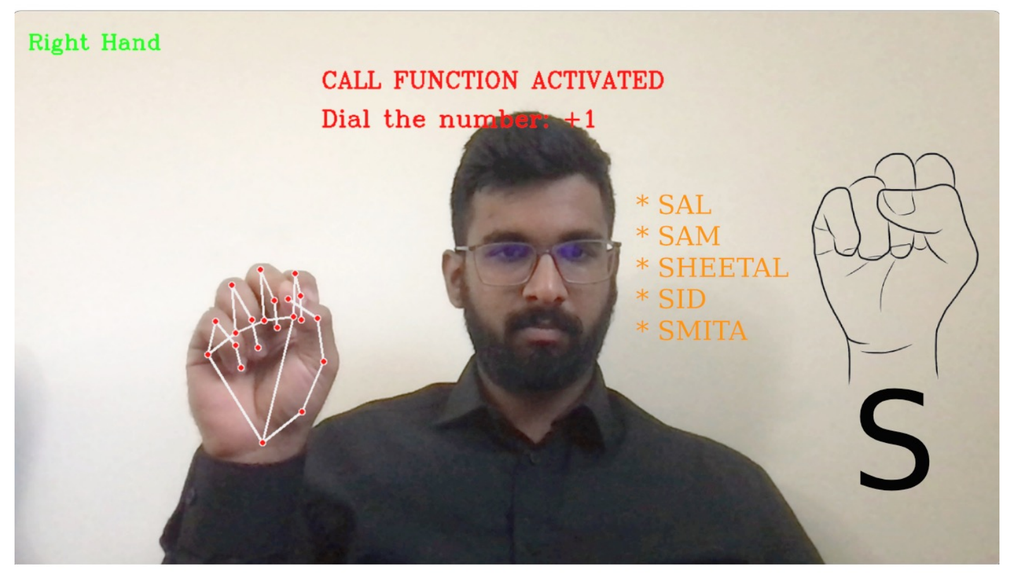


Fig. ASL recognition.

- "S" using sign language will give a prompt on the screen of possible names (Sal, Sam, Sheetal,…). Entering further letters will shorten this list.

User Interface and Implementation

- The steps in creating this new interface are:
  1. Hand Landmark detection.

```
results=hands.process(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
(Source: https://google.github.io/mediapipe/solutions/hands[1])
```

  2. Gesture recognition.

```
tip_0_dist=(math.sqrt((tip_x-Base_x)**2+(tip_y-Base_y)**2))
dip_0_dist=(math.sqrt((dip_x-Base_x)**2+(dip_y-Base_y)**2))
pip_0_dist=(math.sqrt((pip_x-Base_x)**2+(pip_y-Base_y)**2))
if tip_0_dist > dip_0_dist > pip_0_dist:  fingers.append(1)
else:  fingers.append(0)
dist_3_5=math.sqrt((3_x-5_x)**2+(3_y-5_y)**2)
dist_0_3=math.sqrt((Base_x-3_x)**2+(Base_y-3_y)**2)
if dist_0_3 * 0.55 < dist_3_5:   fingers.append(1)
else:   fingers.append(0)
```

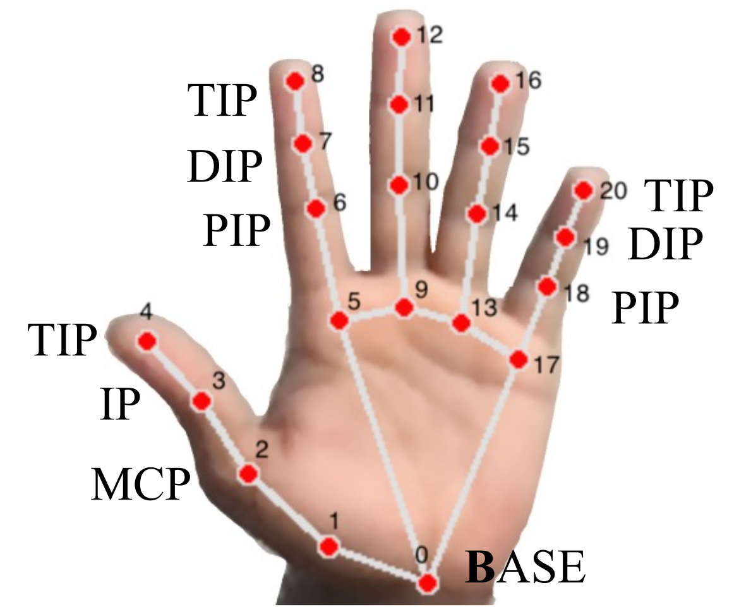


Fig. Hand landmark detection.

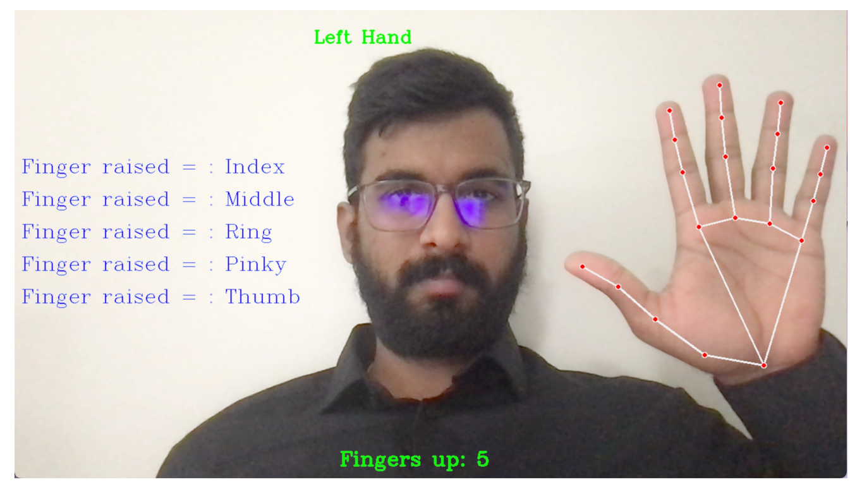


Fig. Calculating raised fingers.

  3. These gestures trigger actions like dialing a number and adjusting the volume or brightness of the device.

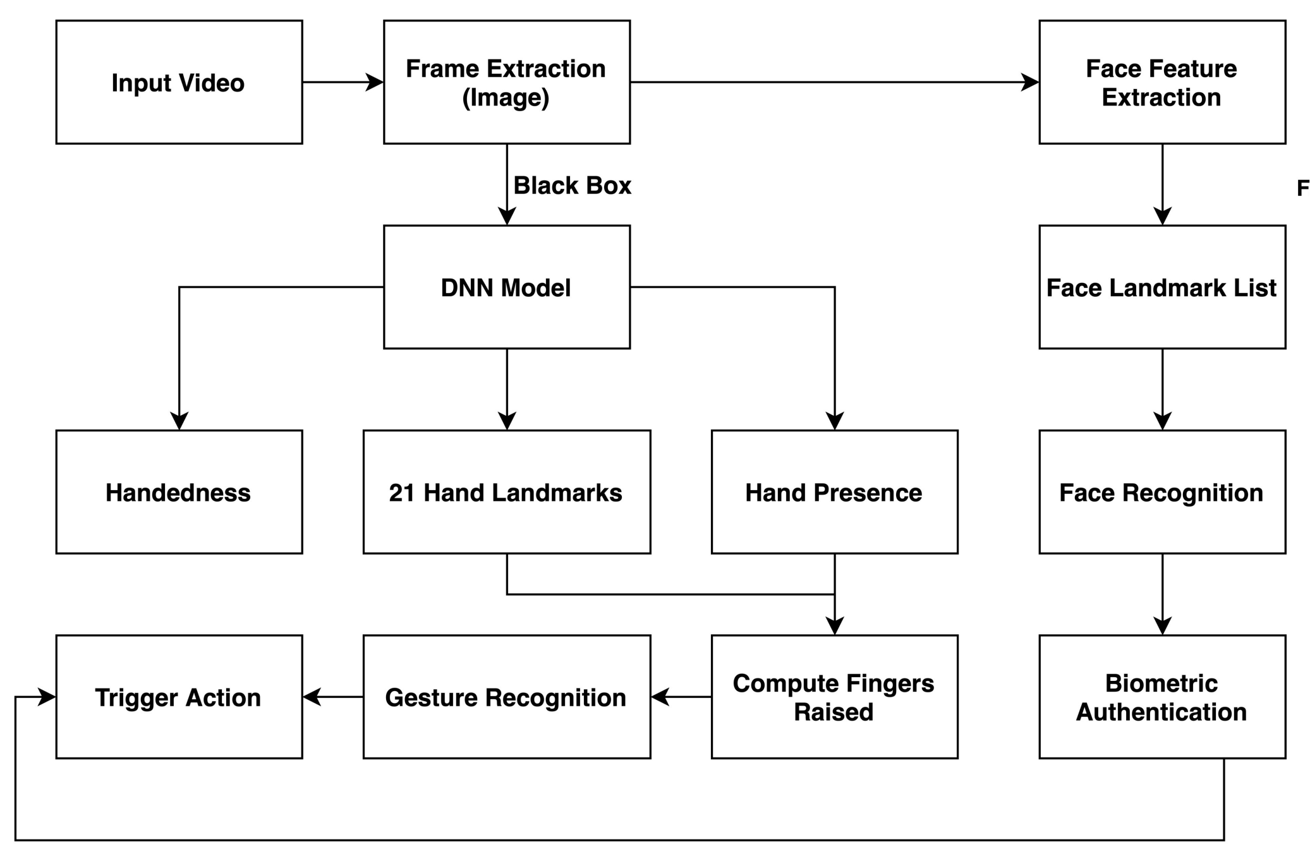
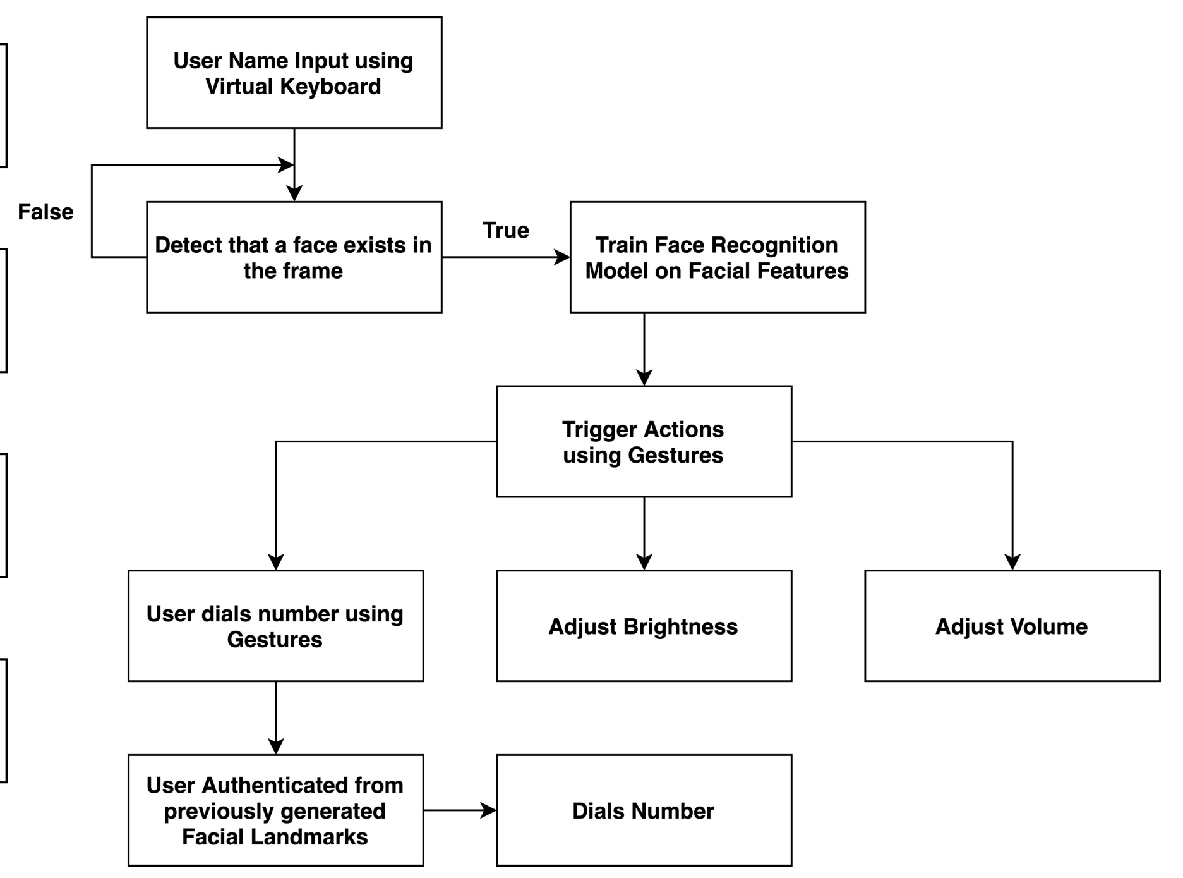


Fig. Implementation flow diagram.


User Name Input using Virtual Keyboard
False
Detect that a face exists in the frame
True
Train Face Recognition Model on Facial Features
Trigger Actions using Gestures
User dials number using Gestures
Adjust Brightness
Adjust Volume
User Authenticated from previously generated Facial Landmarks
Dials Number
Fig. User Interface flow diagram.

- The Palm Detection model and the Hand Landmark Model are used in the MediaPipe (MP) Framework. The MP framework takes frames from the video as input and gives the following output:
  1) Presence of hand (True or False)
  2) Handedness (Left or Right)
  3) 21 Hand Landmarks ((x, y, z) coordinates)
- The palm detection model accurately detects the location of the hand. The hand landmark model accurately detects the 21 hand landmarks.

Results

Iteration	Detection Latency	Tracking Latency	Efficiency
1	29.63 ms	22.36 ms	24.53%
2	30.73 ms	19.18 ms	37.58%
3	27.13 ms	18.91 ms	30.29%
4	37.69 ms	18.87 ms	49.93%

Fig. This shows that tracking is faster than detection.

- The model has been designed for subject 1 and gives 100% accuracy in gesture recognition.
- Hand detection is the most compute-intensive task.
- The static mode detects hand landmarks for every input. Whereas the non-static mode tracks the landmarks after initial detection.
- Tracking is less compute-intensive. Therefore, static mode on average performs 36.62% faster than non-static mode.

Conclusion

- This project demonstrates that Computer Vision based models can be fast enough to be effectively used as an alternate user interface.
- The model is designed specifically for landmark detection in real-time and performs with low latency when the input is a live video stream.
- MP framework utilizes the CPU and GPU resources efficiently for low latency performance.

References

[1] Google Research. Mediapipe. https://google.github.io/mediapipe/solutions/hands, December 2022.
[2] Allinfohere. Finger detection and counting using opencv. https://allinfohere.net/2021/07/ finger-detection-and-counting-using-opencv-allinfohere.html, July 2021.
[3] Gesture volume control. https://www.computervision.zone/courses/gesture- volume- control/, December 2022.