Singidunum University

Faculty of Technical Sciences

Department of Software Data Engineering

Finger Detection in Quiz Application

Subject: Computer Vision

Professor:
prof. dr. Marina Marjanović

Student:
Željko Stojković 2021230319

Belgrade, 2024.

Brief introduction with problem description

mTutor Vision is quiz application that enables users to interact with a quiz through hand gestures. By using computer vision techniques, the application detects the raising of fingers to select answers and a thumbs-up gesture to proceed to the next question. It is an interactive game where the user raises fingers, and the webcam detects how many fingers are raised. Based on this, the corresponding answer is selected. 1 Finger -> Option A, 2 fingers -> Option B, 3 fingers -> Option D, thumbs up -> Next question (marked answer will be selected or null if there is no answer).

This application consist of a frontend developed in React.js and a backend developed in Python. The MediaPipe library is used for hand gesture recognition in backend. MediaPipe is a set of libraries that researchers and developer can use to build world-class machine learning solutions and applications for mobile, web or cloud. For this specific case, the HandTrackingModule library is used.

Communication between the frontend and backend is established through WebSocket. WebSocket is a computer communications protocol, providing a simultaneous two-way communication channel over single Transmission Control Protocol (TCP) connection.. WebSockets were chosen for this project due to real-time communication between client and server which is essential for ensuring that the hand gestures is detected by the webcam.

In quiz applications, users interact with the interface using keyboards and mouse to select their answer and navigate through question. This method can sometimes be less engaging and intuitive. This application improve the interactivity and engagement by offering alternative interaction method. This approach can be more comfortable for older people or those who may not be familiar with using input devices. By using hand gestures for interaction, it provides a more natural way to engage with quizzes.

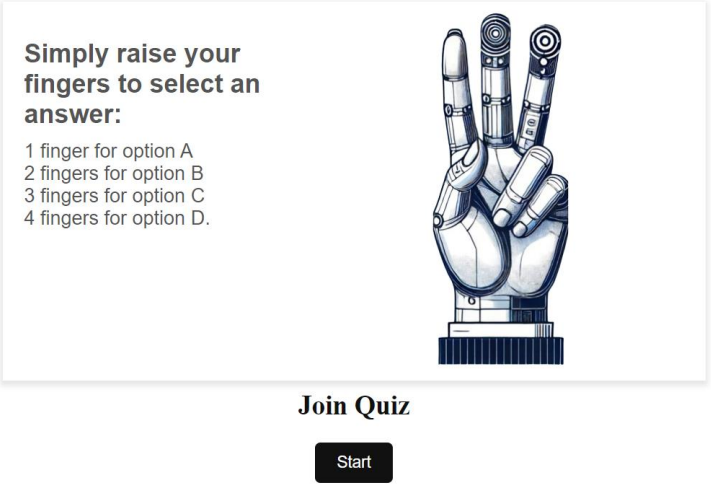


Figure 1. Main Page of mTutor Vision Application

Solution

When I started writing this application I separated the development into units. The first thing I wanted to see was how to connect the frontend and backend. As most of tutorial on MediaPipe primarily explains how to work with a webcam directly within Python. However, sine my webcam is on the frontend side, I cannot simply apply the MediaPipe library directly. To establish real-time communication between the frontend and backend I used WebSocket because they are ideal for application that require real-time updates and low latency, such as quiz application. The trick is that this application is not fully real-time because it is bad for performance. The frontend caputres images from the webcam at regular intervals (every 100 ms).

The react-webcam library is used to integrate the webcam into frontend. This library provides a simple way to access the webcam and capture images.

```
1+ usages   zstojkovic
const captureFrame = () : void  => {
    const imageSrc : string  = webcamRef.current.getScreenshot();
    if (imageSrc) {
        socket.emit( ev: 'process_frame',  args: {image: imageSrc});
    }
};
```

The captureFrame function takes a screenshot from a webcam and convert it into a base640encoded string. The image taken is sent to the backend using WebSocket.

```
@socketio.on('process_frame')
def handle_frame(data):
    global last_thumb_time

    # Decode the base64 image
    image_data = data['image']
    nparr = np.frombuffer(base64.b64decode(image_data.split(',')[1]), np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

    if not os.path.exists('received_images'):
        os.makedirs('received_images')
    cv2.imwrite( filename: 'received_images/received_frame.jpg', img)
```

The backend receives the base64-encoded image through WebSocket, then image is decoded and processed using the MediaPipe library. Saving image is added for debugging purposes. Reason why I didn't use grayscale image is because MediaPipe HandTrackingModule is trained on RGB images and converting them to grayscale won't provide any significant advantages and could potentially degrade the performance or accuracy of the hand detection.

The decoded image is then passed to the MediaPipe HandDetector which is initialized to detect one hand with certain confidence thresholds. HandDetector is initialized with few parameters.
staticMode parameter determines whether the hand detection should be performed in static or dynamic mode. When is set to false it indicates dynamic. Dynamic mode is suitable for real-time video processing where hand is continuously moving, since im sending images it is logical to use static mode.
maxHands parameter is self explanatory, in my case I set it to one hand because I have only 4 options and I don't need all 10 of my fingers for that.
ModelComplexity sets the complexity of the hand landmark model, 1 indicates high-complexity model, which provides more accurate hand detection and landmark placement but requires more CPU power.
detectionCon sets minimum confidence value for the hand detection to be considered successful. I set this to 0.8 because it is a commonly accepted standard.
minTrackCon sets minimum tracking confidence threshold. I set this to 0.5 which means that at least 50% of tracking confidence must be maintained for the hand to continue being tracked. This parameter is less critical for static image since continues tracking is not required.

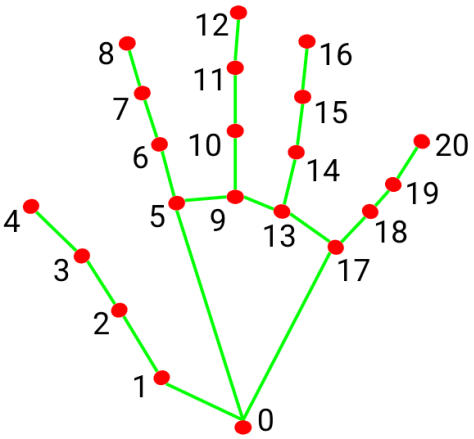
```
42      # Process the image to detect hands
43      hands, img = detector.findHands(img, flipType=True)
44
45      answer = None
46      if hands:
47          hand = hands[0]
48
49          fingers = detector.fingersUp(hand)
50          print('Fingers up:', fingers)
51          num_fingers = fingers.count(1)
52          if 1 <= num_fingers <= 4:
53              answer = num_fingers - 1
54          print('Answer:', answer)
55          emit( event: 'answer',  *args: {'answer': answer})
56      else:
57          print('Fingers up: 0')
```

After processing the image, the next step is to count the number of raised fingers, this is done using detector.fingersUp(hand) method from the MediaPipe library. The detector.findgersUp(hand) return a list where each elements represents if finger is raised (1) or not (0). For example: [1,1,0,0,0] means that thumb and index finger are raised, while other fingers are not.

Fingers.count(1) method counts the number of elements in the list that are equal to 1. This basically gives us number of raised fingers. Since there are four possible answers (0 to 3) we check if the number of raised fingers is between 1 and 4 and if its true, we map numbers of raised fingers to an answer by subtracting 1, because my index of answers start from 0.

The emit function is used to send answer back to frontend, if no fingers are raised we just print message for debugging purpose.

Detecting thumbs up is a little bit harder because it involves examining the positions of specific landmark on the hand. We need to understand that MediaPipe library provides 21 landmarks for each detected hands. These landmarks represents specific points on the hand.



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. Hand landmarks.

Landmark variable is a list of tuples where each tuple contains x,y,z coordinates of a landmark. For example, landmarks[4] gives the coordinates of the THUMB_TIP.

To detect when thumb is raised , we check the y-coordinates of thumb and index finger landmarks. The tip of the thumb (THUMB_TIP) should be higher than the thumb joint (THUMB_MCP), and the thumb joint should be higher than the index finger joint (INDEX_FINGER_MCP)

```
1 usage
def is_thumb_up(landmarks):
    thumb_tip_y = landmarks[4][1]  # THUMB_TIP
    thumb_mcp_y = landmarks[2][1]  # THUMB_MCP
    index_mcp_y = landmarks[5][1]  # INDEX_FINGER_MCP
    return thumb_tip_y < thumb_mcp_y < index_mcp_y
```

Summary

In this project real-time quiz application with computer vision have been implemented. Most difficult challenges were achieving a high-performance game with accurate hand-detection and ensuring communication between the frontend and backend to mark answers correctly on the frontend side.

Future improvements could be implementing multiplayer quiz mode where the user who first raise their finger with correct answers wins and adding more gestures for additional functionalities.