EyeDentify

A low-cost, Raspberry Pi powered cap that assists people with visual impairment through face recognition

By Rhiaan Jhaveri

PROJECT DOCUMENTATION

Introduction

My neighbour, Mr. Didwania, is visually impaired (100% loss of vision). Every morning, I see his wife guiding him on a walk, whispering in his ear when my sister and I were around to say 'hello'. It always nagged me that Mr. Didwania had to be told who he was about to meet.

I was hence spurred into action to brainstorm some sort of device that could assist him, and that could be worn by him on his walks. Upon discussing this with him, I realized such a device would greatly benefit both him and his wife during social outings.

I hence formulated the problem statement for this project:
"To design a device to enable visually impaired people to recognize who is around them without them having to ask many questions or having someone else describe their surroundings every time."

To begin with, I identified some preliminary constraints that the design would require:

- It must work in any language (English, Hindi, Gujarati)
- It cannot be overly conspicuous
- It needs to be portable, since it must be able to work during walks
- It must be usable without external help

Existing solutions

With advancement in wearable electronics and artificial intelligence, there already exist some devices that assist people who are either blind or visually impaired. For instance, '*OrCam MyEye'* (https://www.orcam.com/en/myeye2/) is one such voice activated device that can be attached to any glasses. This device contains features like face recognition, barcode scanning and text reading.

Figure 1: *'OrCam MyEye', a device that could possibly solve similar problems*

However, devices like 'OrCam MyEye' are costly with prices ranging from $2500 to $4500. Furthermore, such devices are not modifiable/improvable according to the needs of a visually impaired person. Other available assisting devices for a blind or visually impaired person are generally not compact in size as a result these devices can be intrusive or annoying to the user. For instance, a part of the assisting device like a battery may be required to be placed in shirt pocket or taped/attached to the user.

Possible solutions considered

1. An easily wearable device that recognizes common faces and basic surrounding objects, and describes these to the wearer through audio. A camera could be used for a video feed, which is run through a machine-learning program in an embedded computer to detect the faces of known people. A headband or cap was chosen since it is an attire suitable for walking. These could also possibly be integrated into glasses that he already wears

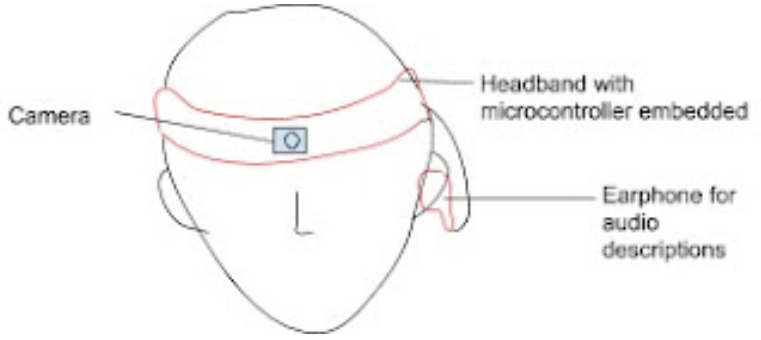

Figure 2: *Concept drawing of first possible solution*

2. A remote-like device with differently-textured buttons that when pressed, have different functions such as detecting the distance to the nearest object (possibly using ultrasonic sensors), recognizing faces when pointed, and alerting for assistance.

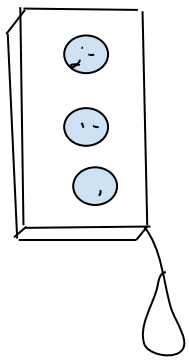

Figure 3: *Concept drawing of second possible solution*

Plan

I decided to stick with the *first solution*, with some modifications. The basic idea is of a cap with a Raspberry Pi (RPI) 3B+ computer attached to it, and a Raspberry Pi Camera at the front. For now, it would have to be powered by a generic 'powerbank' in the user's pocket (this constraint would be overcome later). The RPI would run a code in Python, which after recognizing a face from a database, would play a pre-recorded audio file with the associated name. I also decided to name it '*EyeDentify*'.

Prototype 1

I started with developing the software. I first experimented with the Github 'face-recognition' library on my computer (https://github.com/ageitgey/face_recognition). Combined with OpenCV (https://opencv.org/) for live-video processing, it could detect and distinguish my face in real time, with a name on the textbox, albeit with some lag. The faces that were not recognised were labelled 'Unknown'.

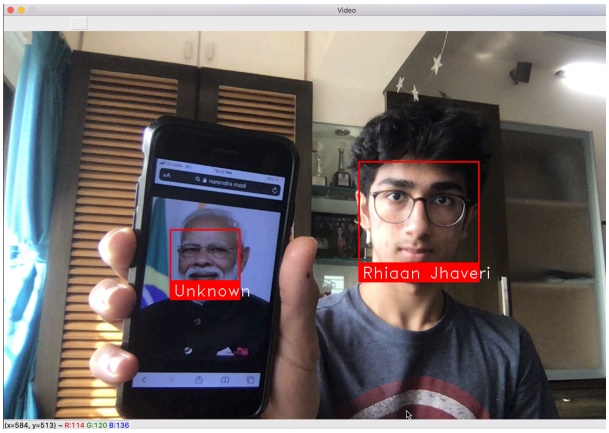

Figure 5: *Face recognition program running on live video from my laptop webcam*

Prototype 2

Next, I tried to prototype the hardware. I found that using screws to attach the RPI board to the cap was effective, held it firmly in place, and used minimal material. However, at this point, I found that the default RPI camera was causing complications in the program due to compatibility with OpenCV. Thus, I decided to opt for a USB camera.

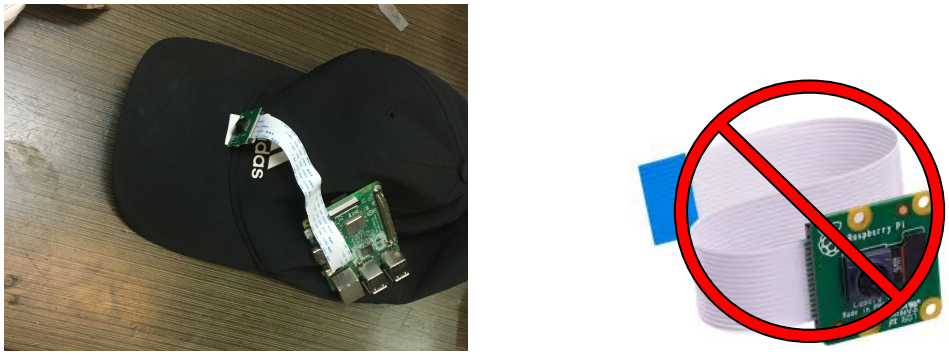

Figure 6: *The second prototype; the Raspberry Pi camera was found to be incompatible*

Prototype 3

I then purchased a generic USB camera like the one seen in Figure 6. Now that I had the camera, with help from my hot glue gun, I stuck the camera in place at the front. Plugging in a powerbank in my pocket with a wire that extended till the RPI, and plugging in generic earphones, I now had a working solution, albeit one that was extremely cluttered with wires

(for the camera, earphones and powerbank), required multiple steps for set-up, and had a glaringly visible RPI on the outside. Still, the fact that this contraption was the first working prototype was, in my opinion, commendable. The working of this prototype can be seen in this video.

Figure 7: *Generic USB Camera*

Prototype 4 (Final)

The next prototype - and one that I felt was really close to the final finished product - involved performing multiple workarounds in order to:

- Reduce the number of loose, dangling wires
- Conceal all the electronics within the cap
- Get rid of the powerbank from the pocket
- Get rid of the earphones
- Improve the camera resolution and integrate it better with the cap

The following are the changes and additions I made, corresponding to the pointers mentioned above:

- Replaced the first cap with a higher quality 'golfer's cap'. This contained two layers of fabric, ideal for concealing all the wires within to make the cap more aesthetically pleasing in the overall design.

- Rather than stitch the electronics within the layers of the cap, I decided to place them in soft pouches that could stick with Velcro to the inside of the cap – this would enable his wife to easily access the power supply to recharge it, and also allow the repair or replacement of parts to occur with ease, in case it was required.
- Replaced the power bank with an Uninterruptible Power Supply (UPS) hat that was about 7x lighter.
- Used a generic mobile phone speaker with an amplifier, connected to the Raspberry Pi and concealed in a pocket at the side of the cap to replace the earphones (to eliminate the need for more wires).
- Used a higher quality camera (Logitech C270). I decided not to conceal the camera, and to leave it placed at the top of the cap. This was because I wanted to give the cap some sort of identity; wearing it would let Mr. Didwania show others that the cap is not ordinary and is a piece of technology he is using to assist himself. It would also absolve any concerns about a hidden camera.

Bearing in mind the above changes to the hardware, I also modified the code slightly. First, I enabled the program to be run immediately upon powering on. I also changed the code to create process the video at a slower rate of frames per second, because this would recognise faces almost just as fine, but would also reduce the lag slightly.

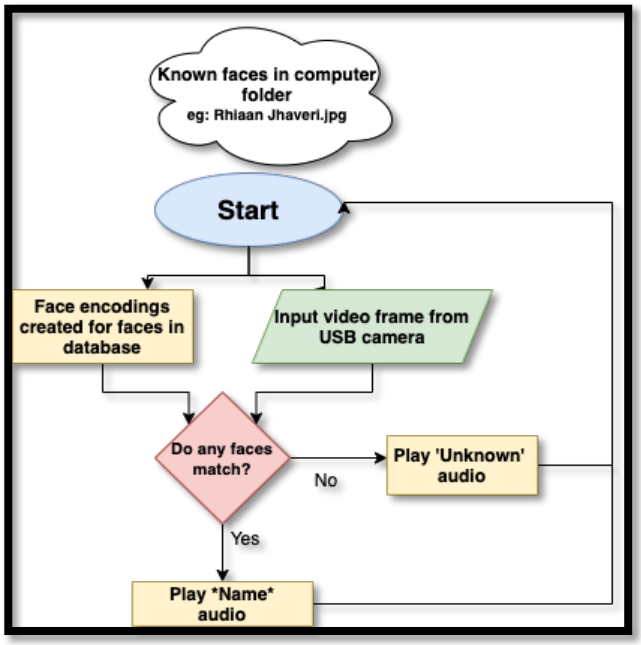

Figure 8: *Flowchart of code*

Construction of the Cap

The final electronic components have been delineated in Figure 8 below:

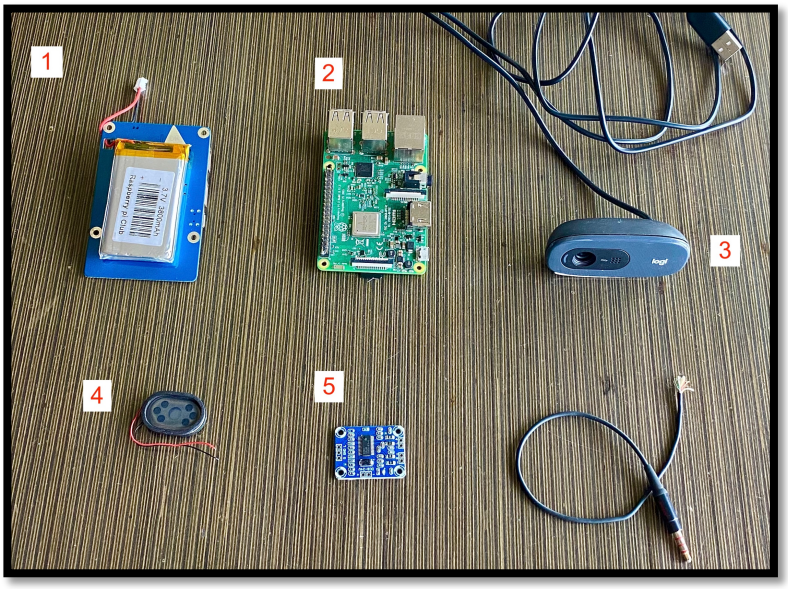

Figure 9: *Electronic components of 'EyeDentify'*

1. A rechargeable 3800 mAh battery (since the Raspberry Pi current requirement is 2A, it would hence last ~1.9 hours).
2. A Raspberry Pi 3B+ - a single board computer with USB ports that would run my Python script
3. An USB camera (with components removed to reduce weight)
4. An ordinary mobile speaker
5. An amplifier module to louden the speaker, and an aux cable to connect it to the Raspberry Pi

To connect electronic components to each other, I used a soldering iron; to hold these components in place I used either a hot glue gun or tape. The electronics were housed in various forms within Velcro pouches and between the layers of fabric of the cap by cutting holes and stitching with help from a tailor.

Figure 10: *Images of a fully constructed and working EyeDentify*

Testing and Evaluation

This final prototype was tested with Mr. Didwania, and can be seen in this video. I also later tested it at the National Association for the Blind (NAB), wherein I was able to conduct demonstrations with multiple candidates and received valuable feedback. From these trials, I was able to evaluate the effectiveness of this project.

There existed some limitations to the final design. These were related, in particular, to its bulkiness and weight. I was able to find some unconventional methods to overcome these to an extent; for example, I tinkered with the camera and removed some of its unnecessary internal components to reduce its weight. Yet, although the cap could be worn in a stable manner, the slight displacement of weight to the back (where the RPI and battery were stored) provided some discomfort to the user. This could be overcome in future iterations by changing the battery to cylindrical Lithium-ion ones and shifting them to the front of the cap so as to balance its weight. The bulkiness of the Raspberry Pi could also be drastically reduced by replacing it with a custom-made printed circuit board (PCB).

Furthermore, once switched on, the program took up to 6 minutes to initialise and start running with live video. This, once again, could be overcome by creating a custom PCB without the unnecessary elements that come with a standard RPI, as the initial creation of

face encodings from the images in the database is a bottleneck in this code and can be overcome with higher processing power.

Nevertheless, the device ran successfully and was able to detect, recognise and inform the wearer of the identity of faces from a distance of up to 3 metres, as was tested. A surprising discovery was when the face-recognition exceeded expectations by recognising faces that were covered by surgical masks, too – this would be an extremely useful feature in the pandemic. Additionally, the cap could be worn and operated with minimal to no external assistance, which was a crucial requirement for the target users who are visually impaired.

Lastly, I received useful suggestions for how I could take this project further. One visually-impaired user at NAB suggested I add the feature of colour-detection for a program which could suggest a colour that complements the colour of one piece of clothing. Another user recommended the inclusion of an option to instant-capture an image from the camera and save it to memory, since visually impaired people in Mumbai are often meddled with and harassed, and this could help them to later catch any perpetrators.

Overall, the problem statement was met satisfactorily, as I had designed a workable solution that could be used by Mr. Didwania in his day-to-day life to identify those around him, and could also be a solution to visually impaired people from across the world, due to its cheap cost, universal design, and high disability-friendliness.

Appendix

Program code:

```
1.  import face_recognition
2.  import cv2
3.  import numpy as np
4.  import simpleaudio as sa
5.
6.  # Connect to default webcam
7.  video_capture = cv2.VideoCapture(0)
8.
9.  # Load a sample picture and learn how to recognize it.
10. rhiaan_image = face_recognition.load_image_file("./known/Rhiaan.jpg")
11. rhiaan_face_encoding = face_recognition.face_encodings(rhiaan_image)[0]
12.
13. # Load a second sample picture and learn how to recognize it.
14. sheetal_image = face_recognition.load_image_file("./known/Sheetal.jpg")
15. sheetal_face_encoding = face_recognition.face_encodings(sheetal_image)[0]
16.
```

```
# Load a third sample picture and learn how to recognize it.
vidya_image = face_recognition.load_image_file("./known/Vidya.jpg")
vidya_face_encoding = face_recognition.face_encodings(vidya_image)[0]

# Create arrays of known face encodings and their names
known_face_encodings = [
    rhiaan_face_encoding,
    sheetal_face_encoding,
    vidya_face_encoding

]
known_face_names = [
    "Rhiaan",
    "Sheetal",
    "Vidya"
]

# Initialize variables
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()
    frame = cv2.flip(frame,1)
    # Resize frame of video to 1/4 size for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

# Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Unknown"

            # If a match was found in known_face_encodings, use the first one.
            # Or instead, use the known face with the smallest distance to the new face
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]

            face_names.append(name)
            if name != 'Unknown':
                filename = name + '.wav'
                print(filename)
                wave_obj = sa.WaveObject.from_wave_file('./known/'+filename)
                play_obj = wave_obj.play()
                play_obj.wait_done()

    process_this_frame = not process_this_frame


    # Display the results
    for (top, right, bottom, left), name in zip(face_locations, face_names):
```

```
        # Scale back up face locations since the frame we detected in was scaled to 1/4 size
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        # Draw a box around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

        # Draw a label with a name below the face
        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

    # Display the resulting image
    cv2.imshow('Video', frame)

    # 'q' on the keyboard to quit
      if cv2.waitKey(1) & 0xFF == ord('q'):
          break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
```