CHAPTER 1

INTRODUCTION

1. 1. Origin of Proposal

According to estimates from the World Health Organization (WHO) Prevention of Blindness, about **285 million** people are visually impaired worldwide: where 39 million are blind and 246 million have low vision (severe or moderate visual impairment).

The onset of vision loss or blindness changes an individual's life dramatically. As the individual deals with the loss - its severity in relationships, identity, and functioning become clear. Blind and impaired people always depend on other people for their locomotion. They become dependent on other people in a way for their day to day routine chores. As a result, previously balanced relationships may take on a new aspect of dependency and can make people feel isolated, immobile and housebound.

Even within the house, colliding with furniture, fear of tripping, falling, dropping things, etc. can lead those with vision impairment to restrict their movements and activities, becoming more sedentary - and thereby putting both mental and physical health at risk.

Mobility for blind people can be described as the flexibility to move with safety and ease in their environment whenever they want, without relying on another person. Most commonly used mobility aids include canes and guide dogs. However, the cane provides a limited preview for the user and as a result, the user has to be more careful while walking and move very slowly. As for the guiding dogs, training and coordinating the dogs with blind people is a difficult task and the results are minimal.

The increment of the blind population to unprecedented levels, demands the understanding and assistance to their needs. According to a study published in Lancet Global Health (2017), the number of people in the world who are blind is set to increase threefold by 2050, from around 36 million (2017) to 115 million. The same research further elucidates the decline in the number of blind people per population group from 0.75% in 1990 to 0.48% in 2015 and the rate of moderate to severe visual impairment from 3.83% to 2.9% over the same time. This decrement can be accredited to improved health interventions.

It is to be noted that the rate of decline of treated vision cases is unmatched to the rapid growth in blindness. Although reasonable work is being put into enhancing the health services, it is fair to say that, the untreated people are leftover without significant heed paid to improve their current quality of life. Equipping the blind with the right technology can empower them as well as gain time for medical professionals to reach a solution/cure.

1. 2. Review of Status of Research & Development

Over the years, the development of assistive techniques to guide the visually impaired is evident. This exploration has empowered the low vision populace towards independence enabling free movement in their environment without guidance from others.

In 2005, D. Yuan et al. have discussed the virtual white cane sensing device based on active triangulation that can measure distances at a rate of 15 m/s. A blind person can use this device for sensing the environment, pointing it as if it were a flashlight. Besides measuring distances, this device can detect surface discontinuities, such as the foot of a wall, a step, or a drop-off. This is obtained by analyzing the range of data collected as the user swings the device around, tracking planar patches and finding discontinuities.

In 2013, a "Smart walking stick" presented by Mohammed H. Rana and Sayemil involved obstacle detection by a ping sensor and the obstacle distance is communicated to the visually impaired by the vibration of the motor. The model is based on a ping sensor (for detecting an obstacle), wet electrode, vibration motor, and the buzzer. Further that year, in a navigation and monitoring system for the blind, an ARM7 controller and ultrasonic technology were used to detect the obstacle and inform the distance of the obstacle to the visually impaired. GSM and GPS technologies for localization of the visually impaired were also included.

Further advancement in the above concept in 2014, the 3D ultrasonic stick for the blind, uses an ultrasonic sensor for detecting the obstacle in three directions (i.e. front, left and the right sides of the visually impaired), and the vibration motor which vibrates with the intensity depending on the obstacle distance. The device has ultrasonic detection, GSM, GPS, voice recognition and voice synthesis technologies. The design process consists of two parts; first is the obstacle detection and voice generation unit design, using ultrasonic detection technology and voice synthesis technology respectively, and the second is the localization and monitoring unit design using GPS technology GSM technology, as well as the voice recognition technology. The two units are then combined to form the complete device.

1. 3. Background Problem

The interest to build a commodity for the sightless is escalating among developers and technology enthusiasts. In most of the methods proposed for the visually impaired people, accuracy and efficiency are mediocre. Also, these approaches are void of a user-friendly and accessible interface.

Despite the global effort to develop a sustainable system, the pragmatism is doubtful. Furthermore, none of the methods delivers a consolidation of various features that can render a persistent guide unnecessary, nor does it present a real-world perception for the visionless.

1. 4. Objective

The fundamental objective is to integrate computer vision, cognitive abilities, real-world perception, and audio representation into a unique wearable prototype with the purpose of assisting visually impaired people indoors as well as outdoors. Furthermore:

- ➔ To design a wearable prototype that is comfortable and easy to access by the visually impaired.
- ➔ To consolidate features that can enhance the virtue of life for the blind.

- ➔ To develop a cheap, affordable and efficient way to help the blind people to navigate with greater comfort, speed, and confidence.
- ➔ To enable the blind to regain the ability to be self-sufficient anywhere.
- ➔ To ensure the blind of safety while moving around and accomplishing daily activities.

1. 5. Proposed System

"*Humans are not disabled. A person can never be broken. Our built environment, our technologies, is what is broken and disabled. We the people need not accept our limitations, but can transfer disability through Technological Innovation*"

Technology has always helped human surpass their limitations and move towards a better life. Even for the blind, the need for an effective solution that can help them perform daily chores is clearly noticeable. In this research titled '**Third Eye**', extensive work has been put in, to overcome the difficulties in the existing methods and, to provide a cost-effective and user-friendly system for the visually-challenged.

The proposed solution involves designing a wearable as an aid for the visually impaired to cope with the day to day challenges they face. Efforts to use the proposed system are minimal- neither does the system require special training nor does it include holding a huge device for a long distance. The intent of the research, further, solicits the need for a light, portable and, most importantly, affordable device. On a larger scale, with improvements in the prototype, the cost will reduce and drastically benefit the society.

This electronic guidance system, which is proposed as a constructive assistant and support, can function accurately, hardly requires any training, is easy to wear and operate. The design is a special wearable device that can be worn like spectacles and is based on a Raspberry Pi. This device is equipped with hardware components like Pi camera, GPS module, and Bluetooth headphones.

Predominantly, this system has six assistive features:

- **Object Detection**: The challenged person will be updated about the surrounding objects. This enables the user to understand his/her environment to take corresponding actions.
- **Know Your Friend**: ‘Third Eye’ is made to be smart enough to detect basic attributes and determine your friends. This allows the challenged person to greet their friends in the usual manner.
- **Text Reader**: This feature is, essentially, an image-to-voice converter that keeps the user informed about elemental activities like reading price tags, reading labels on medicine.
- **Geofence**: It monitors their position and the challenged person will be alerted with buzzer/ vibration in case they go out beyond the desired distance or in case of a detour. Also, the guardians are alerted and can live-track the user.
- **Community for help**: ‘Third eye’ intends to build a community around its users. It helps them connect among themselves and with others.

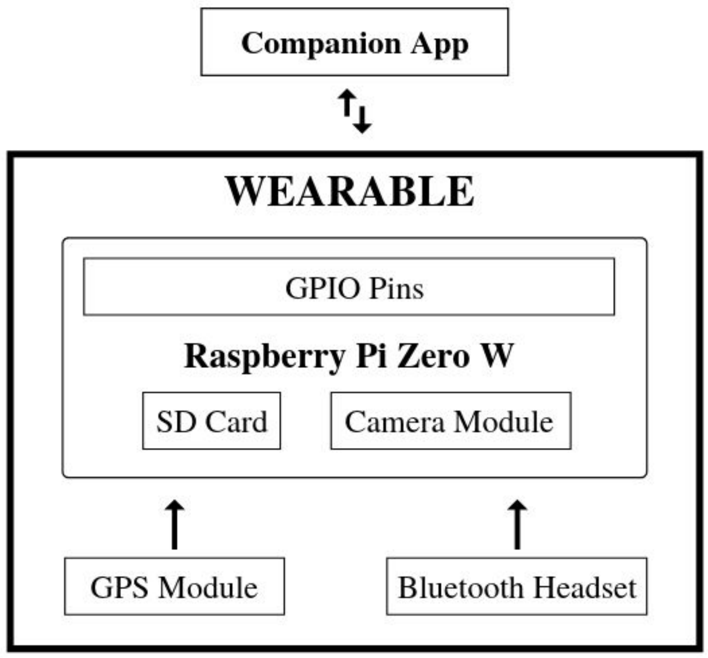


Fig. 1.1. Block Diagram of the Proposed System

A detailed description of each of the features has been provided in the following chapter.

CHAPTER 2

FEATURES

2. 1. Object Detection

The ‘Object Detection’ in ‘Third Eye’ is a lucid feature that updates the challenged person about the surrounding objects. This enables the user to understand his/her environment to take corresponding actions.

The spectacle-shaped device captures an image of the surroundings using an imager. By analyzing the visual content of the image, we can ascertain the objects in the captured image. Further, the images can be tagged into categories to help the user comprehend the atmosphere and situation around him/her.

2. 2. Know Your Friend

‘Third Eye’ is made to be smart enough to detect basic attributes and determine your friends. This allows the challenged person to greet their friends in the usual manner. The ‘Know Your Friend’ feature can alert the user about the following:

- The facial expression of the person
- Age and gender of the person
- Name of the person (If an acquaintance)
- Other physical details

Using a captured image, we can determine the age and gender of a person. If the face is stored in the database, it identifies the person in front of our user. If not, it provides a range of facial and physical attributes. Also, it recognizes the person’s expressions, thus, stating his/her mood to the challenged person. This can help build a genuine interaction between the two.

2. 3. Image To Text Reader

This feature is, essentially, an image-to-voice converter that helps the user perform elemental activities like:

- Reading price tags
- Reading product name or brand of a commodity
- Reading labels on medicine

Using optical character recognition (OCR), we can detect the text in images. Using Image Processing, we can analyze visual data and extract embedded text. The custom-made voice assistant helps us convey the text to the user.

2. 4. Geofence

Geofence, as the name suggests, is a preset virtual boundary or fence based on geopositioning systems. The 'Geofence' feature monitors their position and the challenged person will be alerted with buzzer/ vibration in case they go beyond the desired distance or in case of a detour. Also, the guardians are alerted and can live-track the user. Using the Google Maps platform & U-Blox GPS module, we can record (& live-track to guardians) the location of the challenged person. In the case of a Geofence breach, both, the user and their guardians are alerted.

2. 5. Community

'Third eye' intends to build a community around its users. It helps them connect among themselves and with others. It allows volunteers to be online/ alert to help the visually challenged people. This way, anyone can help the challenged in mere seconds, anytime, anywhere. In return for their service, volunteers are rewarded with Social Credit Points (SCP). The users can identify people (similar to them) around them who may seek assistance & spend time with a like-minded peer. (The users can discover friends through "Third Eye"). Using Flutter framework for the mobile applications, we provide an interface for citizens to volunteer and help the ones in need. The users can send requests when they require assistance. Moreover, the users can connect with others like them and socialize. We use PubNub Cloud services for request transactions and notifications.

All these features are built to collectively render a real-world perception via audio representation.

CHAPTER 3

HARDWARE

3. 1. Raspberry Pi Zero W

3.1.1. Introduction to Raspberry Pi

The Raspberry Pi is an economical, “credit-card sized” computer that plugs into a display device (like monitor or Television) and uses a common keyboard and mouse for control. This adept device enables people of all ages to explore computing and to learn how to program in languages like Scratch and Python. It has been technically equipped to do everything a desktop computer can- from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

The Raspberry Pi is a range of small single-board computers developed by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The Raspberry Pi is being used by people all over the world to learn to program and understand how computers work.

The product was initiated to make some boards and give them out to encourage students to experiment with programming. However, the model became more popular than expected, selling outside of its intended market for uses such as robotics. This small device has helped enthusiasts to make their own games machines, cameras, skateboards, robot arms, walkie-talkies, etc. It has also given small startups a solid technical base to build their own products.

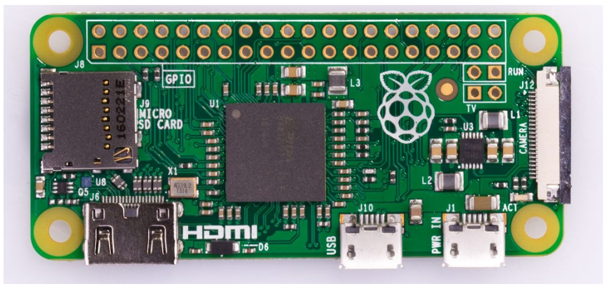


Fig. 3.1. Raspberry Pi Zero W

The Pi Zero model is an ultra-low-cost and incredibly small microcomputer packed onto a single board. It's roughly one third the size of the Raspberry Pi 3. Despite its miniature stature, the Pi Zero is a full-fledged microcomputer with a 1GHz ARM CPU and 512MB RAM. It bundles enough technology to run the full version of Raspbian.

In early 2017, Raspberry Pi community disclosed a new board with wireless extension- The "Pi Zero W" which is identical to the Pi Zero model but includes an onboard 802.11 b/g/n wireless and Bluetooth 4.1 LE. This latest version of the Zero makes the board far more useful due to its nominal, minuscule structure, powerful processor and wireless extension.

3.1.2. Overview

The Pi Zero W has an onboard Wi-Fi 802.11n and Bluetooth. The small form factor, integrated Wi-Fi chip and low price, make Pi Zero W the perfect computer for running low-power and portable jobs, especially where space is an issue. The Pi Zero W's tiny circuit board holds an entire computer, RAM and the wireless chip.

Zero W contains Micro USB port for power, a Micro USB OTG (On the Go) port for peripherals and a micro HDMI output. A mini HDMI-to-HDMI adapter or cable is required to connect the Raspberry Pi to a television or monitor. Alternatively, developers can attach an RCA cable directly to the video headers on the board. RCA cables are white, yellow and red plugs often found for older televisions. This feature makes the Pi Zero W a great choice for retro gaming enthusiasts.

The foundation involves Raspbian, a Linux operating system based on the popular Debian distribution, as well as third-party Ubuntu, Windows 10 IOT Core, RISC OS, and specialized media centre distributions. It features Python and Scratch as the main programming language, with support for many other languages. While the default firmware is closed source, an unofficial open source is available. Customized specifically for the Raspberry Pi hardware, it's a hassle-free experience using a Raspberry Pi with Raspbian (as it is fully compatible).

3.1.3. Technical Specifications

- **SoC**: Broadcom BCM2835

- **CPU**: ARM11 running at 1GHz
- **RAM**: 512MB LPDDR2 SDRAM
- **Wireless**: 2.4GHz 802.11n wireless LAN
- **Bluetooth**: Bluetooth Classic 4.1 and Bluetooth LE
- **Power**: 5V, supplied via a micro USB connector
- **Video & Audio**: 1080P HD video & stereo audio via mini-HDMI connector
- **Storage**: MicroSD card
- **Output**: Micro USB
- **GPIO**: 40-pin GPIO, unpopulated
- **Pins**: Run mode, unpopulated; RCA composite, unpopulated
- **Camera Serial Interface (CSI)**

3.1.4. Design

For the Pi Zero W, the processor is situated at approximately the middle of the board, while the GPIO pins are positioned at the top of the board. The micro SD slot is placed at the left end of the board. Due to the unavailability of built-in flash memory on the Pi Zero W, we need to use a microSD card for the OS and for data storage. The Pi Zero W cannot boot without a microSD card (as it stores the OS). Towards the bottom of the board, there is a mini-HDMI port for a display device as well as two micro-USB ports – one for power and the other for peripherals and data. To use the micro-USB port for peripheral connections (like a keyboard or mouse) a micro-USB male to USB A female adapter (USB OTG) is required. The Pi Zero W is not equipped with an Ethernet port. However, it has built-in Wi-Fi capabilities to facilitate connection to the board. To get the Raspberry Pi Zero W up and running, a micro-USB power supply and a microSD card (at least 4 GB) are essential.

3.1.5. Processor

Raspberry Pi Zero W board is built on interesting System on Chip (SoC) manufactured by Broadcom company. Successive versions of the board using very similar versions of the chip marked as BCM2835, BCM2836, BCM2837. The chips are multimedia processors, i.e. they contain application processors and a lot of hardware that supports multimedia processing (e.g. video de/compression). These Broadcom chips are part of the Video Core IV family. The Raspberry Pi Zero W uses a Broadcom BCM2835 processor.

The ARM11™ **processor family** provides the engine that is widely used in consumer, home, and embedded applications. It delivers extremely low power and a range of performance from 350 MHz in small area designs up to 1 GHz in speed optimized designs in 45 and 65 nm.

3.1.6. RAM

LPDDR2 SDRAM double data rate architecture is a prefetch architecture with an interface designed to transfer two data words per clock cycle at the I/O pins.

3.1.7. Micro SD Slot

We require a micro SD card with at least a 4GB loaded with the OS of your choice. Typically, this means installing the lightweight Linux distribution, Raspbian, which is designed specifically for the Pi.

3.1.8. Mini HDMI

In contrast to the previous models of the Raspberry Pi which use a standard HDMI connector, the Zero W uses a mini HDMI connector to save space. To connect the Zero W to a monitor or television, mini HDMI to HDMI adapter or cable should be used.

3.1.9. USB On-The-Go

The other models of Raspberry Pi traditionally had 2-4 standard size female USB connectors, to facilitate the connection of various devices like mice, keyboards, and WiFi dongles. The Zero W has opted for a USB On-the-Go (OTG) connection to save space. The Pi Zero W uses the same Broadcom IC that powered the original Raspberry Pi A and A+ models. This IC is directly connected to the USB port allowing the OTG

functionality (unlike the Pi B, B+, 2 and 3 models, which use an onboard USB hub to allow for multiple USB connections).
To connect a certain device with a standard male USB connector, we need a USB OTG cable. Plug the micro USB connector-end into the Pi Zero W, and connect your USB device via the standard female USB end.

3.1.10. Power

Like other Pi models, the power supply to the board is provided through a micro USB connector. The voltage supplied to the power USB should be in the range of **5-5.75V**.

3.1.11. Wi-Fi and Bluetooth

The Zero W offers an 802.11n wireless LAN and a Bluetooth 4.1 connectivity. This frees up most of the connections that would have been made over USB, such as a WiFi dongle, USB keyboard, and mouse if substituted with a Bluetooth keyboard/mouse.

3.1.12. Camera Connector

The **Raspberry Pi** consists of a Mobile Industry Processor Interface (MIPI) Camera Serial Interface Type 2 (**CSI**-2), which facilitates the connection of a small camera to the main Broadcom BCM2835 processor. The data communication is one-way; from the camera to the processor. This model has an integrated camera interface (CSI), which was absent on the first released Pi Zero. The Raspberry Pi Zero W has an onboard camera connector. This can be used to connect the Raspberry Pi Camera module. The connector is a 22pin 0.5mm and it is different compared to the connector of the standard Pi.

3.1.13. GPIO (General Purpose Input Output Pins)

- **Voltages:** Two 5V pins, two 3V3 pins and a number of ground pins (0V) which are not configurable are present onboard. The remaining pins are all general purpose 3V3 pins, which means that the outputs are set to 3V3 and inputs are 3V3-tolerant.

- **Outputs:** A GPIO output pin can be set to high (3V3) or low (0V).
- **Inputs:** A GPIO input pin can be read as high (3V3) or low (0V). This is made simple with the use of internal pull-up or pull-down resistors. GPIO2 and GPIO3 pins have fixed pull-up resistors, but for other pins, this can be configured in software.

3.1.14. Operating Systems

The Raspberry Pi Foundation recommends the utilization of Raspbian, a Debian-based Linux operating system. Other third-party operating systems available include Ubuntu MATE, Snappy Ubuntu Core, Windows 10 IoT Core, RISC OS and Kodi (aka OpenElec or OSMC). Several other operating systems can also run on the Raspberry Pi.

3.1.15. Firmware

The official firmware is a freely redistributable binary blob, which is closed source. A fundamental proof-of-concept open source firmware is also accessible, chiefly aimed towards initializing and starting the ARM cores as well as performing minimal startup that is required on the ARM aspect. It is additionally capable of booting a minute Linux kernel, with patches to exclude the dependency on the mailbox interface being responsive. It is best known to work on Raspberry Pi 1, 2 and 3, just as few variations of Raspberry Pi Zero and Zero W.

3.1.16. Connections

- To connect the Pi Zero W to a Monitor or TV that has an HDMI input, attach a mini HDMI to HDMI cable or adapter to the mini HDMI connector on the Pi Zero W. Connect the other end to the HDMI port to your television or monitor.
- Connect the USB OTG cable to the Pi Zero W via the micro USB connector. If you have a keyboard/mouse combination, attach the dongle to the standard female USB end. If you have a separate mouse and keyboard, you will need a USB hub to attach both to the USB OTG cable.
- A valid Raspberry Pi image should be present on your microSD card. Insert the microSD card into the microSD slot.

- Power your Pi Zero W via the micro USB power input.
- The Pi Zero W has built-in WiFi, so external WiFi dongles are not needed.
- The device should automatically boot up. When the Raspberry Pi logo appears, it symbolizes successful bootup. This is the first boot for your device.
- Once the Wifi connection is established, subsequent access to the device can be remote using protocols like SSH.

3. 2. Pi Camera

3.2.1. Introduction

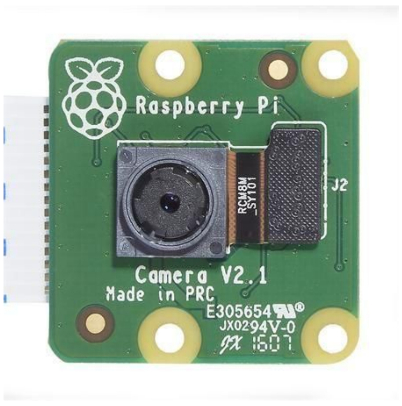


Fig. 3.2. Pi Camera

The Pi camera module is a portable lightweight camera that is supported by Raspberry Pi Zero W. It is a custom designed add-on for Raspberry Pi Zero W. It attaches to Zero W by a flexible 15cm ribbon cable. It communicates with Pi Zero W using the MIPI camera serial interface protocol. This CSI interface, which was designed specifically for camera interfacing is capable of extremely high data rates as it carries the pixel data. The board is tiny, with dimensions 25mm x 20mm x 9mm and weighs over 3g, making it perfect for mobile and other applications where size and weight are important. No adapters are required, this camera directly plugs into the Raspberry Pi Zero W camera port.

The sensor has a native resolution of 5 megapixels and has a fixed focus lens onboard. In terms of still images, the camera is capable of 2592 x 1944 pixel static images and also supports 1080p30, 720p60 and 640x480p60/90 video. It can be accessed through V4L APIs, MMAL and numerous third-party libraries built for it, including the Pi camera Python library. The Raspberry Pi camera module delivers outstanding photos but can also shoot video, ideal for drones or CCTV projects. It is easy to use for beginners but has plenty to offer to advanced users for expanding their knowledge. We can bundle libraries to the camera to create effects like time-lapse, slow motion and other video discernment.

It finds applications in image processing, machine learning and surveillance projects. It is commonly used in surveillance drones where the payload capacity is limited.

3.2.2. Features

- Lightweight and portable
- Fully Compatible with all the Raspberry Pi Models
- MIPI Camera Serial Interface - Plugs Directly into the Raspberry Pi Board
- OmniVision 5647 Camera Module
- Resolution: Wider image, capable of 2592x1944 stills
- Supports: 1080p, 720p and 480p
- Supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 Recording
- JPEG (accelerated), JPEG + RAW, GIF, BMP, PNG, YUV420, RGB888 picture formats supported
- Provides high sensitivity, low crosstalk, and low noise image capture

3.2.3. Technical Specifications

- **Resolution:** 5MP
- **Lens focus:** Fixed focus
- **Image size(pixels) :** 2592*1944
- **Interface type:** CSI(Camera Serial Interface)
- **Sensors:** Omnivision 5647 fixed focus
- **Aperture:** 2.9
- **Focal Length:** 3.29

- **FOV:** 72.4°

3. 3. GPS Module

3.3.1. Introduction

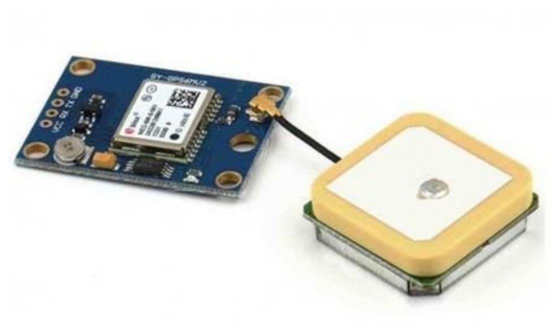


Fig. 3.3. UBlox NEO-6M GPS Module

GPS stands for **Global Positioning System** which is used to detect the Latitude and Longitude of any location on the Earth, with exact UTC time (Universal Time Coordinated). It plays a prominent role in the present day navigation systems, starting with smartphones and automobiles to much more complex missile guidance systems.

GPS module includes a larger built-in 25 x 25mm active GPS antenna with a UART TTL socket. This antenna is connected to the module through a UFL cable which allows for flexibility in mounting the GPS. It has serial TTL output with four pins: TX, RX, VCC, and GND. The GPS Module has a battery for power backup and EEPROM for storing configuration settings. This module gives high precision binary output and has high sensitivity for indoor applications. It is powerful enough to use it with cars and other mobile applications.

GPS module sends the data related to tracking position in real time in the NMEA format. NMEA format consists of several sentences, in which only one sentence is required. This sentence starts with **$GPGGA** and contains the coordinates, time and other useful information. This $**GPGGA** is the **Global Positioning System Fix Data**. We can extract coordinate from $GPGGA string by counting the commas in the string.

The module receives the coordinates from the satellite for each and every second, with time and date.

Raspberry Pi Zero W, interfaced with a GPS module, is used for developing an advanced real-time navigation system. Incorporating the Pi's image processing and web interface capabilities along with the GPS data we can develop advanced navigation schemes for real-time implementation.

3.3.2. Features

- 5Hz position update rate
- Operating temperature range: -40 TO 85°C UART TTL socket
- EEPROM to save configuration settings
- Rechargeable battery for Backup
- The cold start time of 38 s and Hot start time of 1 s
- Supply voltage: 3.3 V
- Configurable from 4800 Baud to 115200 Baud rates. (default 9600)
- SuperSense ® Indoor GPS: -162 dBm tracking sensitivity
- Support SBAS (WAAS, EGNOS, MSAS, GAGAN)
- Separated 18X18mm GPS antenna

3.3.3. Technical Specifications

- **Receiver Type:** 50 Channels,GPS L1 frequency, C/A Code SBAS: WAAS, EGNOS, MSAS
- **Supply Voltage (V):** 2.7~ 6 VDC
- **Sensitivity:** Cold Start: -147 dBm ;Hot Start: -156 dBm; Reacquisition: -160 dBm

  Tracking & Navigation: -161 dBm
- **Navigation Update Rate:** 5Hz
- **Position Accuracy:** 2 M and better with multiple good satellite signals
- **Temperature Range:** -24°C ~ 84°C
- **Tracking Sensitivity:** -161 dBm
- **Cold Start Time:** 27s

- **Warm Start Time:** 7s
- **Maximum Speed:** 500 M/s
- **Dimensions (mm):** GPS Board – 22 x 30 x 4
- **Antenna** – 25 x 25 x 7
- **Weight (gm):** 12gm
- **Cable Length:** 10cm

3. 4. Bluetooth Headphones

Fig. 3.4. Bluetooth Headphones

One of the best features of the Raspberry Pi Zero W is its onboard Bluetooth. Bluetooth is a proprietary open wireless technology used for exchanging data over short distances, creating personal area networks with high levels of security. It was at first perceived as a wireless alternative to RS-232 data cables. It can connect several devices, thereby overcoming problems of synchronization. Learning how to use a Bluetooth headset is easy, as these headsets are designed to be user-friendly. There are different versions of Bluetooth, so users must ensure that the devices they are trying to pair are compatible. In addition, this technology has the power to connect cell phones, computers, speakers, printers and other devices

CHAPTER 4

INTERFACING THE HARDWARE

4. 1. Interfacing Raspberry Pi and Pi Camera

4.1.1. Configure the Raspberry Pi Computer

On first boot up of the Raspberry Pi, you will be presented with the ‘raspi-config’ menu. There are three steps that need to be completed before we start installing the camera software:

1) Update the OS to the latest version
2) Update the Firmware
3) Update Boot Configuration File

4.1.2. Connection

With the Pi Zero W switched off, we connect the Camera Module to the Raspberry Pi’s camera port and then start up the Pi. Locate the port on the Pi Camera and connect the cable by gently pulling up the edges of the plastic clip and inserting the wider end of the adapter cable with the conductors facing in the same direction as the camera’s lens. Then, attach the adapter to the Pi Zero W by gently lifting the collar at the edge of the board and inserting the smaller end of the adapter with the conductors facing the back of the Pi Zero W.

4.1.3. Camera Preview

1) Apply power to your Pi. Once booted, start the Raspberry Pi Configuration utility from the main menu and enable the camera module
2) If it’s not enabled, enable it and reboot the Pi to begin.
3) Once the camera is connected and the software is enabled, get started by trying out the camera preview.
4) After rebooting, start the terminal and use the following command:
   ```
   raspistill -o image.jpg
   ```

5) The camera will start; a preview from the camera appears on the display and, after a 5-second delay it will capture an image (storing it as image.jpg) before shutting down the camera.

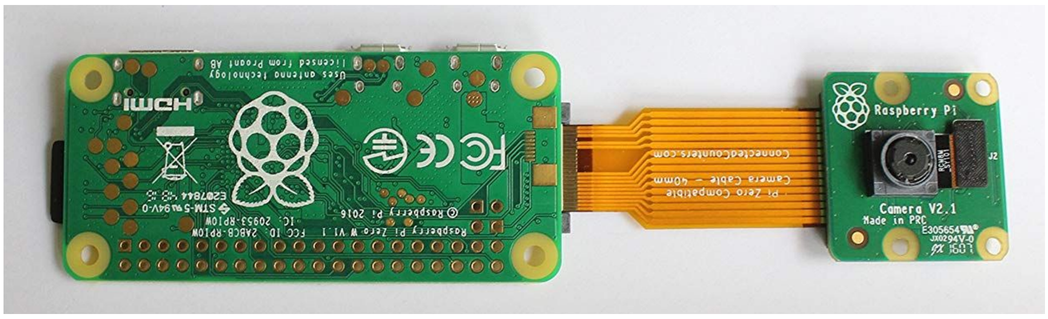


Fig. 4.1. Raspberry Pi Zero W and Pi Camera Interfacing

4. 2. Interfacing Raspberry Pi and GPS Module

4.2.1. Electrical Connection

The 4 pins of the GPS Module have to be connected as described below:

- VCC to Pin 1 (3.3V)
- TX to Pin 10, which is RX (GPIO15)
- RX to Pin 8, Which is TX (GPIO14)
- Gnd to Pin 6, which is Gnd

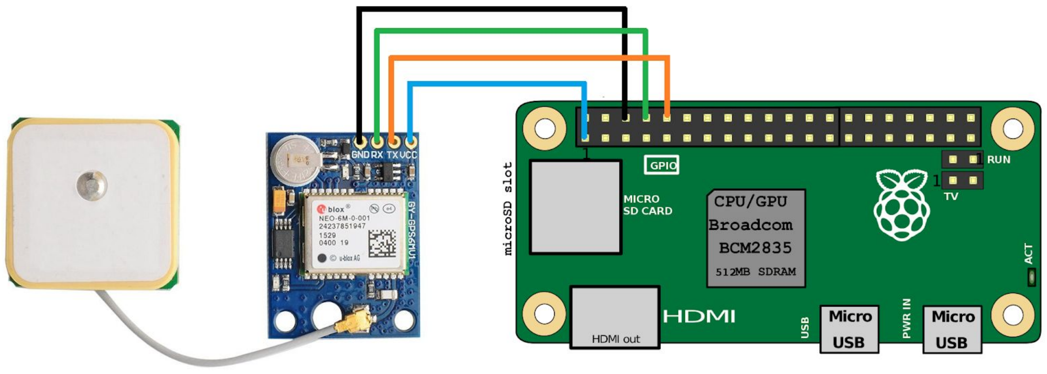


Fig. 4.2. Raspberry Pi Zero W and GPS Module Connection Circuit

4.2.2. Installing GPS Daemon

To install the GPS Daemon, open the terminal in Raspberry Pi and type the following command. This step may take a while.

```
sudo apt-get install gpsd gpsd-clients cmake subversion build-essential espeak freeglut3-dev imagemagick libdbus-1-dev libdbus-glib-1-dev libdevil-dev libfontconfig1-dev libfreetype6-dev libfribidi-dev libgarmin-dev libglc-dev libgps-dev libgtk2.0-dev libimlib2-dev libpq-dev libqt4-dev libqtwebkit-dev librsvg2-bin libsdl-image1.2-dev libspeechd-dev libxml2-dev ttf-liberation
```

4.2.3. After installation

Perform the following steps after the installation:

1) Type the following command in the terminal:

   ```
   sudo nano /boot/config.txt
   ```

   In the file that opens up, add:

   ```
   core_freq=250
   enable_uart=1
   ```

2) Then, type:

   ```
   sudo nano /boot/cmdline.txt
   ```

   Change the file to the following:

   ```
   dwc_otg.lpm_enable=0  console=tty1 root=/dev/mmcblk0p2
   rootfstype=ext4    elevator=deadline  fsck.repair=yes
   rootwait
   ```

3) Run:

   ```
   sudo systemctl stop serial-getty@ttyS0.service
   sudo systemctl disable serial-getty@ttyS0.service
   sudo systemctl stop gpsd.socket
   sudo systemctl disable gpsd.socket
   ```

4) Reboot the device

5) Execute the daemon reset:

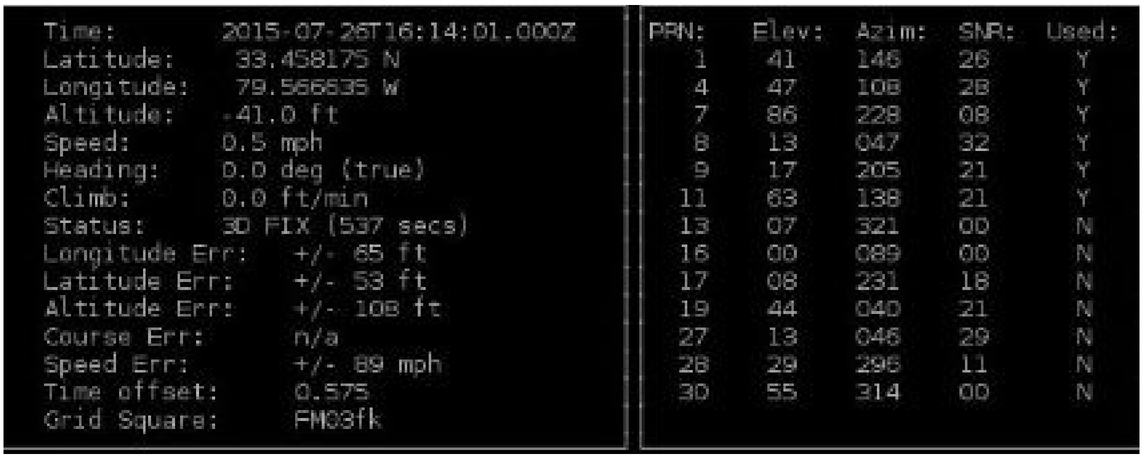
```
sudo killall gpsd
sudo gpsd /dev/ttyS0 -F /var/run/gpsd.sock
```

4.2.4. Testing the GPS

Test GPS NMEA data by typing the following command:

```
sudo cgps -s
```

The output for the command is as follows:



```
Time:        2015-07-26T16:14:01.000Z    PRN:   Elev:  Azim:  SNR:  Used:
Latitude:     33.458175 N                   1    41     146    26      Y
Longitude:    79.566635 W                   4    47     108    28      Y
Altitude:    -41.0 ft                       7    86     228    08      Y
Speed:       0.5 mph                        8    13     047    32      Y
Heading:     0.0 deg (true)                 9    17     205    21      Y
Climb:       0.0 ft/min                    11    63     138    21      Y
Status:      3D FIX (537 secs)             13    07     321    00      N
Longitude Err:   +/- 65 ft                 16    00     089    00      N
Latitude Err:    +/- 53 ft                 17    08     231    18      N
Altitude Err:    +/- 108 ft                19    44     040    21      N
Course Err:      n/a                       27    13     046    29      N
Speed Err:       +/- 89 mph                28    29     296    11      N
Time offset:     0.575                     30    55     314    00      N
Grid Square:     FM03fk
```

Fig. 4.3. GPS Output in the Command Prompt

4. 3. Interfacing Raspberry Pi and Bluetooth Headphones

Steps to interface Pi Zero W and Bluetooth Headphones:

1) Open a terminal and issue the following command to install the Bluetooth support software:

```
sudo apt-get install blueman pulseaudio pavucontrol
pulseaudio-module-bluetooth
```

2) Restart your Pi

3) Click Menu > Preferences > Bluetooth Manager to open the Bluetooth Manager Window.

4) Be sure to turn off Bluetooth on any other device you have previously used the adapter with to ensure it doesn’t automatically connect with that device.
5) In the Bluetooth Devices window that appears, click Search. You should see your headphones under the listed devices.
6) Right-click on your device and select Connect. Your device should connect. Orange, green, and blue symbols should appear at the right.

CHAPTER 5

SOFTWARE

5. 1. Python

5.1.1. Introduction

Python is a high-level programming language. It is often also termed as a scripting language as a result of its easy to use and understandable syntax. Python follows the rules of an Object-Oriented Language unlike procedure-oriented languages such as C. According to many programmers it is a successor of the JAVA programming language and addresses the formerly unmet needs. Python is popular amongst beginners and tech enthusiasts due to its intuitive syntax. Like many other languages, Python has its own Integrated Development Environment called the Python IDLE facilitating convenient and fast implementation of Python programs for beginners.

Raspberry Pi is an Embedded System that leverages the power of Python to perform otherwise computationally expensive tasks. The hardware access libraries of a Raspberry Pi have also been implemented in Python. The RPI.GPIO Library is a great example of the Raspberry Pi’s implementation of hardware procedures. Considering all the above-mentioned reasons it becomes quintessential to understand the basic building blocks of the Python programming language.

5.1.2. Basic Building Blocks in Python

➔ Basic Hello World in Python:

```
print(“Hello, World!”)
```

➔ Variables in Python:

```
# defining a variable: In Python, there is no need to
mention the data type

var1 = 10      # An integer assignment
var2 = 3.146   # A floating point
var3 = "Hello" # A string
```

```
print(var1,' ',var2,' ',var3)
```

➔ Strings:

```
str = 'Hello World!'  # A string
print(str)          # Prints complete string
print(str[0])
print(str[2:5])      # Prints characters starting from
3rd to 5th
print(str[2:])       # Prints string starting from 3rd
character
print(str[:2])
print(str * 2)      # Prints string two times
print(str + "TEST") # Prints concatenated string
```

➔ Lists:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]  # A list
print(list)
print(list[0])
print(list[-1])
#append
list.append(8)
```

➔ Dictionaries:

```
#  Lists  are  ordered  sets  of  objects,  whereas
dictionaries  are  unordered  sets.  But  the  main
difference is that items in dictionaries are accessed
via keys and not via their position.
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
print(tel)
print(tel['jack'])
tel['irv'] = 4127
#Dictionary Operations
print(tel.keys())
print(sorted(tel.keys()))
print(sorted(tel.values()))
print('guido' in tel)
print('jack' not in tel)
```

➔ Conditional Statements:

If-Else:

```
n = 5
if(n%2==0):
    print("The number is even") #mind the indentation
else:
    print("The number is odd") #same here
```
Else-if:

```
if(3<1):
    print("Entered if")
elif(2<3):
    print("Entered elif")
else:
    print("Entered else")
```

➔ Loops:

```
for i in range(0,100):
    print(i)

j = 0
while(j<10):
    print("I love python!!!")
    j+=1
```

➔ Functions in Python:

```
def area(length,width):
    return length*width

are = area(10,20)

print("Area of rectangle:",are)
```

➔ Imports:

```
import math as mt      # importing a package
print("Log value: ",mt.log(10))
```

5. 2. Microsoft Cognitive Services

5.2.1. Introduction

Microsoft Cognitive Services are Application Programmable Interfaces (API's), Software Development Kits(SDK's), and services available to help developers build intelligent applications. They aggrandize on Microsoft's evolving portfolio of machine learning APIs and allow the developers to easily add cognitive features – such as video detection; vision, speech, and facial recognition; and language - speech understanding – into their applications. The goal of Azure Cognitive Services is to assist developers to create applications that can see, hear, speak, understand, and even begin to reason. The catalog of services within Azure Cognitive Services can be classified into five main pillars - Vision, Speech, Language, Search, and Knowledge.

5.2.2. Computer Vision

Azure's Computer Vision service equips the developers with access to advanced algorithms that process images and returning information. To evaluate an image, we can either upload an image or specify an image URL. The images processing algorithms can inspect the content in several different ways, depending on the visual features you're interested in. For example, Computer Vision can detect if an image contains racy or adult content, or it can find all of the human faces in an image. We can use Computer Vision in our application by using a native SDK or invoking the REST API directly.

5.2.3. Object Detection

Object detection identifies various objects in an image and returns the bounding box coordinates (in pixels) for each object found. For example, if an image contains a book, a phone, and a person, this service will list those objects along with their coordinates in the image. Hence, we can use this functionality to process the relationship between various objects in an image. It also determines multiple instances, if any, of the same tag in an image.

Describing images in Human-Readable Language:

The algorithms in this API analyze the content in an image to make a 'description' which can be displayed in a human-readable language. A description that summarizes what is found in the image, is of vital importance for the targeted (visually challenged) users. Thus, each description is examined and a confidence score is generated. A list is these descriptions, in ascending order of their confidence scores, is returned.

Object Detection and Image Description Example

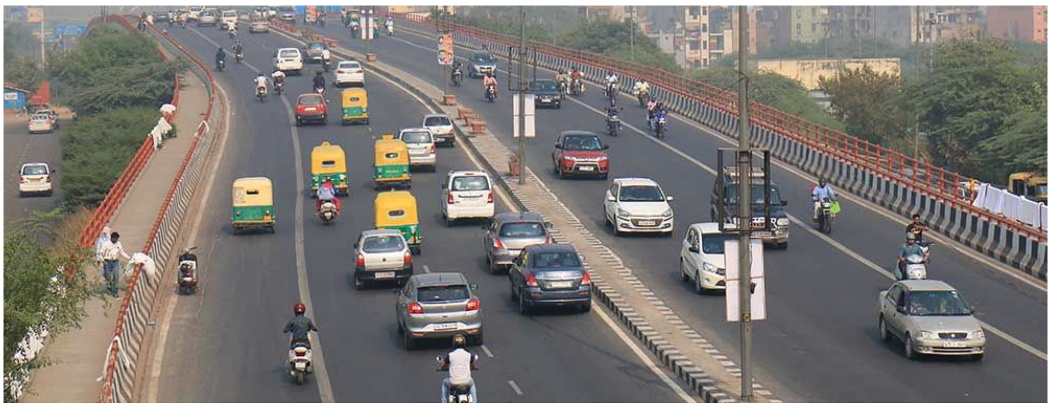


Fig. 5.1. Example Image for Object Detection

```
{'description':

{'tags': ['road', 'scene', 'car', 'outdoor', 'building',
'filled', 'traffic', 'driving', 'city', 'highway',
'street', 'busy', 'bus', 'full', 'view', 'truck', 'riding',
'bridge', 'many', 'traveling', 'group', 'parked',
'mirror'],

'captions': [{'text': 'cars driving down a busy highway',

 'confidence': 0.8150843045020368}]},

'requestId':'e47a251a-1164-4147-87fe-33a5cf15247e',

  'metadata': {'width': 1000, 'height': 430, 'format':
'Jpeg'}}
```

For this project, the user receives a voice directive that describes the surroundings as:

```
'cars driving down a busy highway'
```

5.2.4. Extracting Text

The Optical character recognition (OCR) technology in the Computer Vision API detects text content in an image and extracts the recognized text into a machine-readable character stream. We can input this text to a voice assistant which can read out the information to the user. Also, if required, OCR corrects the rotation of the recognized text, in degrees, around the horizontal image axis.

Computer Vision can detect and extract printed or handwritten text from images of various objects with different surfaces and backgrounds, such as receipts, posters, business cards, letters, and whiteboards. Text recognition saves time and effort. You can be more productive by taking an image of text rather than transcribing it. Text recognition makes it possible to digitize notes. This digitization allows you to implement a quick and easy search. It also reduces paper clutter.

Text Recognition Requirements

Computer Vision can recognize printed and handwritten text in images that meet the following requirements:

- The image must be presented in JPEG, PNG, or BMP format
- The file size of the image must be less than 4 megabytes (MB)
- The dimensions of the image must be between 50 x 50 and 4200 x 4200 pixels

Note: This technology is currently in preview and is only available for English text.

5.2.5. Azure Face API

The Azure Face API is a cognitive service that enables accurate detection, recognition, and analysis of human faces in digital imagery. The service equips developers with the ability to process human face information for numerous applications in management, security etc.

We have utilized the following APIs in our project:

1) **Face detection:** This API is used human face detection in images and specifies their location coordinates. Moreover, it can extract various attributes like gender, age, head pose, facial hair, and glasses. Detection of the face is also available in Computer Vision API, however, this API offers more operations on face data.

2) **Face verification:** In order to authenticate two face images and determine whether they belong to the same person, we use the Verify API. This service is extremely useful for security systems.
3) **Find similar faces:** This feature takes in a desired face as the target and looks for a set of faces that look most similar to the target face. Of the two modes available, matchPerson and matchFace, we will use matchPerson. The **matchPerson** mode will return images of the same person as the target face, however, **matchFace** mode returns people's images which are similar to that of the target face. The matched images need not depict the same person in the desired face.
4) **Person identification:** For applications where it is necessary to identify a person from a database, the Identify API does exactly that. This service requires the creation of a database in prior which can be modified later. Multiple groups of people can be created with person objects (or people fitting into that category). When a new face is detected, we can identify people from various groups. If identified, it will return the person object is returned.

5. 3. Cloud Computing: PubNub

5.3.1. Introduction

PubNub is a programmable network for developing real-time applications built to handle the complexities of data streams. PubNub applies logic to real-time data and minimizes the latency to 2.5 seconds or less- guaranteeing scalability and reliability. The primary product is a publish/subscribe (PubSub) messaging API built on a global DSN which lets us establish and maintain persistent socket connections to any device and push data to the global audience. We can publish messages to any given channel, and only the subscribed clients receive messages associated with that channel. The message payload can be any JSON data including numbers, arrays, strings, and objects.

5.3.2. PubNub Key Concepts

PubNub's globally distributed network infrastructure involves three core services:

1) **Publish/Subscribe:** This is a messaging pattern to stream data in real-time across the PubNub network for real-time updates and Internet of Things.
2) **PubNub Functions:** These functions enable us to run code directly inside the PubNub network, which executes on our data during its streamed from senders to receivers.
3) **ChatEngine:** It is an open and extensible chat framework for building and powering 1:1 and group chat.

In our project, we avail the Publish/Subscribe feature to facilitate communications among various modules.

5.3.3. Publish/Subscribe (Pub/Sub)

PubNub operates on a publish/subscribe (pub/sub) paradigm, which consists of the following components:

- Publisher
- Subscriber
- Message
- Channel
- Channel group

The request to publish data will immediately push out that data to anyone (or thing/device) that was interested in it (subscribers). Subscribers will continue to receive these data streams in real-time as and when the data is published

Messages

Messages are the packets of data that get published to channels. They can contain any serializable data (Objects, Arrays, Numbers, and Strings). The maximum size for any single message is 32KB. Prior to the publishing of a message, PubNub SDKs will automatically stringify JSON objects. Most developers use the JSON format, and the keys and values are completely up to us.

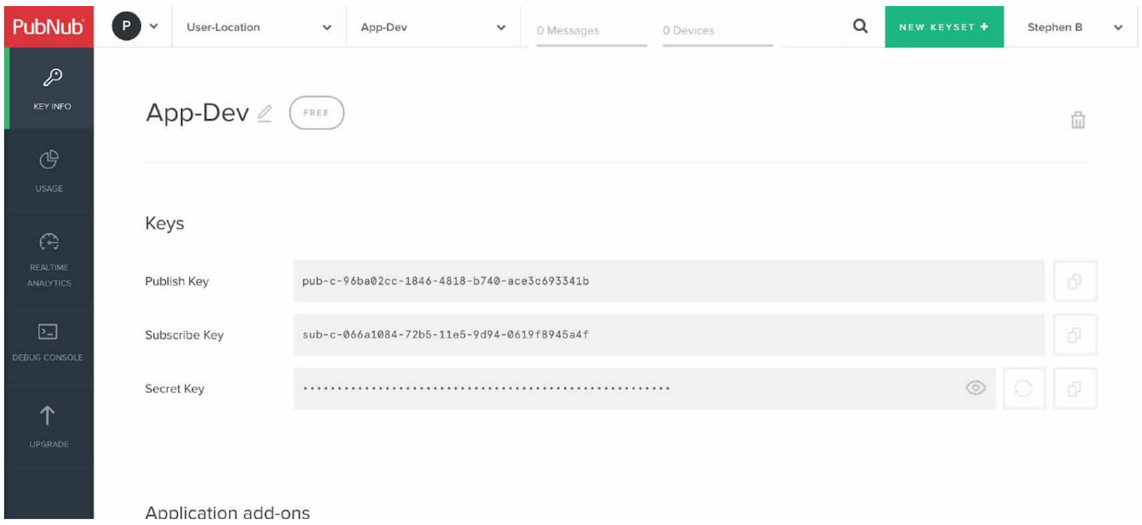
Channels

Messages sent via PubNub are transmitted on a “channel”. Devices can subscribe to thousands of channels simultaneously and each device subscribed to a channel will receive the published messages in under 250ms. PubNub supports unlimited Channels and they are automatically created by publishing a message onto the channel (no need to define them in prior).

Channels are unique for each set of publish and subscribe key. A channel named ‘thirdeye’ a pub/sub key set is not the same as a channel by the same name under a different key set. We cannot publish a message on a channel under one key set and receive that message under another key set. The Channel and Channel Group names are limited to 92 characters in length.

5.3.4. PubNub Admin Dashboard

The PubNub Admin Dashboard allows the users to create pub/sub keysets, track account usage, manage billing information, adjust configuration settings, create blocks, and more.



App-Dev FREE

Keys

Publish Key pub-c-96ba02cc-1846-4818-b740-ace3c693341b

Subscribe Key sub-c-066a1084-72b5-11e5-9d94-0619f8945a4f

Secret Key ••

Application add-ons

Fig. 5.2. PubNub Admin Dashboard

5. 4. Flutter

5.4.1. Introduction

Flutter is a framework created by Google for mobile app development. Although it is the primary approach for creating apps for Fuschia (an Operating System developed by Google), this open source software can be used to develop applications for Android and iOS as well. The apps built in Flutter are written in the Dart language.

5.4.2. Widgets

Flutter's slogan, "everything is a widget," conceptualizes the use of a basic structure called a "widget". These are essentially the building blocks of any UI element in Flutter.

Complex widgets can be constructed by combining simpler ones. Each widget is a subclass of either Stateless Widget or Stateful Widget, depending on whether the widget manages any state (static or dynamic respectively). For the Stateful Widgets, the state change triggers the widget to rebuild its description, which the framework compares against the previous description in order to determine the minimal changes to be made for transition from one state to the next.

Some of the basic, yet powerful, widgets that are commonly used in Flutter are:

- **Text**: The Text widget lets us create styled text within our application.
- **Row, Column**: These flex widgets let us create flexible layouts, in both, the horizontal (Row) and vertical (Column) directions.
- **Stack**: Rather than linear orientation (either horizontally or vertically), a Stack widget lets us stack widgets on top of each other in paint order. Later, we can use the Positioned widget on children of a Stack to position them relative to the top, right, bottom, or left edge of the stack.
- **Container**: The Container widget lets us create a rectangular visual element. A container can be decorated with a `BoxDecoration`, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size.

5.4.3. Material Components

We can easily incorporate Material Components in Flutter as it provides a number of widgets that follow Material Design. Such a Material app starts of with the `MaterialApp` widget, which builds a number of useful widgets at the root of your app. This also includes a Navigator that manages a stack of widgets identified by strings, also known as “routes”. These routes are screen transitions and the Navigator allows smooth transitioning. Using the `MaterialApp` widget is entirely optional but a favourable practice.

5.4.4. Layout Widgets

Standard Layout Widgets:

- **Container**: Adds padding, margins, borders, background color, or other decorations to a widget.
- **GridView**: Lays widgets out as a scrollable grid.
- **ListView**: Lays widgets out as a scrollable list.
- **Stack**: Overlaps a widget on top of another.

Material widgets:

- **Card**: Organizes related info into a box with rounded corners and a drop shadow.
- **ListTile**: Organizes up to 3 lines of text, and optional leading and trailing icons, into a row.

The following illustrations represent the Standard layout widgets:

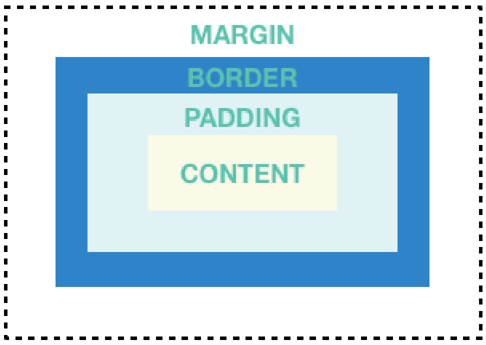
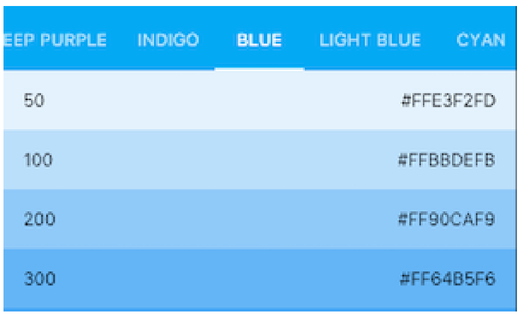


Fig. 5.3. Container Widget

Fig. 5.4. Grid View Widget



50	#FFE3F2FD
100	#FFBBDEFB
200	#FF90CAF9
300	#FF64B5F6

Fig. 5.5. List View Widget

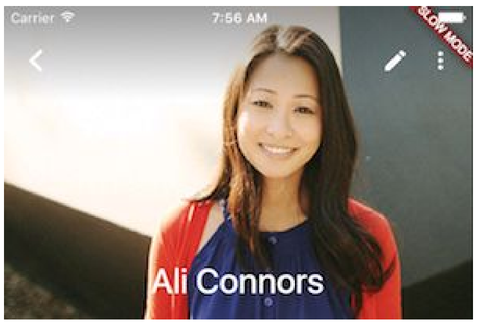
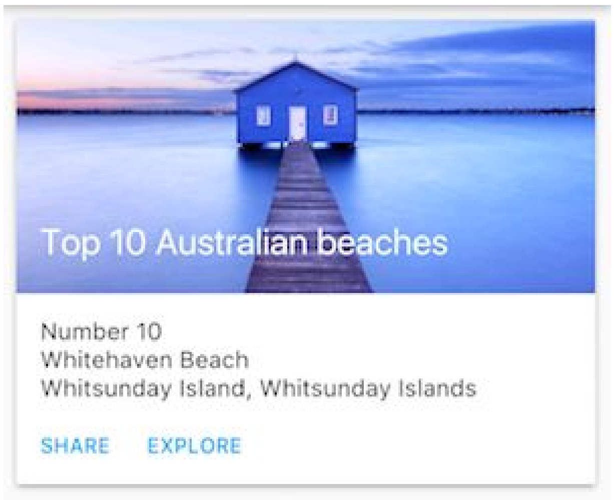


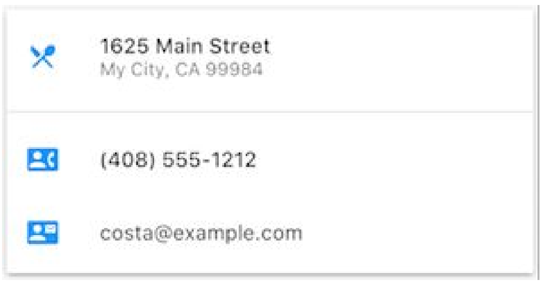
Fig. 5.6. Stack Widget

The following illustrations represent the Material layout widgets:



Number 10
Whitehaven Beach
Whitsunday Island, Whitsunday Islands

Fig. 5.7. Card Widget



1625 Main Street
My City, CA 99984

(408) 555-1212

costa@example.com

Fig. 5.8. List Tile Widget

CHAPTER 6

COMPUTER VISION

6. 1. Technical Overview

The hardware components utilized for the Computer Vision features are:

- Raspberry Pi
- Pi camera
- Button (trigger)

To facilitate the threefold computer vision features: Object Detection, Know Your Friend and Image to Text Reader; we arrange three buttons on the wearable glasses of the user. Each button on the wearable, acts as a trigger to its corresponding feature. In order to initiate any feature we capture the image, which has to be converted into a voice directive, using the Pi camera. The image will then be stored on the Raspberry Pi. The image will be uploaded onto cloud and the resultant URL will be passed for corresponding analysis using the REST API of Microsoft Computer Vision.

After analysis, the objects in the image will be added as tags. Based on these tags, we can generate a 'description' in a human-readable language. This description will be passed to a voice assistant that will inform the user of the objects in his/her surroundings.

To detect the friends of the user, we initially create a database to store the names and their respective images. The analysis for this feature includes individual comparison of the current image with each image in the database. We use various functions from the API to detect as well as verify these facial images.

Using the OCR technology facilitated by the Computer Vision API, text in the image will be converted and displayed in the console window to enhance debuggability. We convert handwritten as well as printed text in the images to voice, empowering the person to apprehend labels/written instructions within his/her surroundings.

6. 2. Object Detection

6.2.1. Feature Description

The ‘Object Detection’ in ‘Third Eye’ is a lucid feature that updates the challenged person about the surrounding objects. This enables the user to understand his/her environment to take corresponding actions. The spectacle-shaped device captures an image of the surroundings using an imager. By analyzing the visual content of the image, we can ascertain the objects in the captured image. Further, the images can be tagged into categories to help the user comprehend the atmosphere and situation around him/her.

6.2.2. Software Code

```
import requests
from time import sleep
import json
import picamera
from cloudinary.uploader import upload
from cloudinary.utils import cloudinary_url
from cloudinary.api import delete_resources_by_tag,
resources_by_tag
# config
os.chdir(os.path.join(os.path.dirname(sys.argv[0]), '.'))
if os.path.exists('settings.py'):
    exec(open('settings.py').read())
```

These are the required library imports for our program. The `requests` import enables HTTP requests. The `os`, `sys` and `subprocess` are for hardware interfacing. `json` enables JSON reading and interpretation. The other included imports are cloudinary libraries followed by the `picamera` import. The last line of code authenticates the cloudinary access.

```
DEFAULT_TAG = "python_sample_basic"

cam = picamera.PiCamera()

def dump_response(response):
    print("Upload response:")
    for key in sorted(response.keys()):
        print("  %s: %s" % (key, response[key]))
```

The `cam` variable is an instantiation of a PiCamera object. The `dump_response()` is a function prerequisite for cloudinary API. It takes the key “URL” to return the URL of the image.

```
def upload_files(id1):
    print("--- Upload a local file")
    response = upload("image.jpg", tags = DEFAULT_TAG)
    dump_response(response)
    url, options = cloudinary_url(response['public_id'],
        format = response['format'],
        width = 200,
        height = 150,
        crop = "fill"
    )
    print(url)
    identify_img(url)
```

The `upload_files()` is a function that takes an argument file `id1` and uploads this image file to cloudinary for a public URL. After fetching the public URL, it prints the URL and passes this as an argument to another function called the `identify_img()` which analyzes the image.

```
def identify_img(url):
    headers = { 'Ocp-Apim-Subscription-Key' : '****' }
    api_url =
'https://westcentralus.api.cognitive.microsoft.com/vision/v
2.0/describe?maxCandidates=1'
    response=
requests.post(api_url,headers=headers,json={'url':img})
    res=response.json()
    print(res)
```

The `identify_img()` function takes an argument of a public URL of an image and identifies the objects in that image. The detected objects are returned in the JSON format which are printed.

```
cam.capture("img.jpg")
upload_files("img.jpg")
```

The camera captures an image and stores it as “img.jpg”. On calling the `upload_files()` the image is uploaded to Cloudinary and calls the `identify_img()`. Once this function recognizes the objects in that image, the JSON response is printed.

6. 3. Know Your Friend

6.3.1. Feature Description

‘Third Eye’ is made to be smart enough to detect basic attributes and determine your friends. This allows the challenged person to greet their friends in the usual manner. The ‘Know Your Friend’ feature can alert the user about the following:

1) The facial expression of the person
2) Age and gender of the person
3) Name of the person (If an acquaintance)
4) The colour of their shirt & other physical details

Using a captured image, we can determine the age and gender of a person. If the face is stored in the database, it identifies the person in front of our user. If not, it provides a range of facial and physical attributes. Also, it recognizes the person’s expressions, thus, stating his/her mood to the challenged person. This can help build a genuine interaction between the two.

6.3.2. Software Code

```
import requests
import os, sys
from time import sleep
import json
import subprocess
from cloudinary.uploader import upload
from cloudinary.utils import cloudinary_url
from cloudinary.api import delete_resources_by_tag,
resources_by_tag
import picamera

os.chdir(os.path.join(os.path.dirname(sys.argv[0]), '.'))
if os.path.exists('settings.py'):
    exec(open('settings.py').read())
```

These are the required library imports for our program. The `requests` import enables HTTP requests. The `os`, `sys` and `subprocess` are for hardware interfacing. `json` enables JSON reading and interpretation. The other included imports are cloudinary libraries followed by the `picamera` import. The last line of code authenticates the cloudinary access.

```
camera= picamera.PiCamera()

def detect_faces(url):

    headers = { 'Ocp-Apim-Subscription-Key' : '****' }

    params = {
            'returnFaceId':'true',
            'returnFaceLandmarks':'false',
            'returnFaceAttributes':'age,gender',
    }
    face_api_url =
'https://westcentralus.api.cognitive.microsoft.com/face/v1.
0/detect'
    image_url = url
    response=
requests.post(face_api_url,params=params,headers=headers,js
on={'url':image_url})
    faces=response.json()

    if(len(faces)>0):
        retlist=[]
        id=faces[0]['faceId']
        att = faces[0]['faceAttributes']
        retlist.append(id)
        retlist.append(att)
        return(retlist)
    else:
        print('Please retake image.')
        return('no id')
```

The `detect_faces(url)` is a user defined python function that takes a URL as an argument. This function looks for faces in the image in the URL and returns a list `retlist` with the detected face count `len(faces)`, the `faceId` and the `faceAttributes` of the detected faces. For images with an undetected face, it requests the user to retake the image and returns “no id”.

```
def verify_faces(id1,id2):
    headers = { 'Ocp-Apim-Subscription-Key' : '****' }
    face_api_url =
'https://westcentralus.api.cognitive.microsoft.com/face/v1.
0/verify'
    response=
requests.post(face_api_url,headers=headers,json={'faceId1':
id1,'faceId2':id2})
    res=response.json()
    if(res['isIdentical']==True):
        return(True)
    else:
        return(False)
```

The `verify_faces(id1,id2)` is also user defined python function that takes 2 IDs- `id1` and `id2` as arguments. Each of these IDs are generated in the `detect_faces(url)` function. This function, essentially, compares two faces and returns whether they are identical. If the face in both the IDs are identical, the function returns `true` else returns `false`.

```
def upload_files(id1):
     print("--- Upload a local file")
     response = upload("img.jpg")
     dump_response(response)
     url, options = cloudinary_url(response['public_id'],
       format = response['format'],
       width = 200,
       height = 150,
       crop = "fill"
   )
     print(url)
     id2 = detect_faces(url)
     verify_faces(id1,id2)

friends= { 'anuroop':
'http://res.cloudinary.com/shannu/image/upload/v1533544913/
WIN_20180806_14_11_01_Pro_gqqeec.jpg', 'pragnya':
'http://res.cloudinary.com/shannu/image/upload/v1533544769/
WIN_20180806_14_08_25_Pro_qvfjmb.jpg'}
camera.capture(“img.jpg”)
upload_files(“img.jpg”)
```

The variable `friends` is a dictionary that stores the user's friend database. Currently, it has the images of two friends of the user, "anuroop" and "pragnya". A new image `img.jpg` is captured using the Pi camera interface. This image will be compared with those in the `friends` dictionary to determine whether the person in front of the user is his/her friend or not. Prior to the comparison of the image, it is first uploaded onto the cloudinary service to fetch a public URL facilitating easy verification by the Microsoft Cognitive Services.

```
new_face=detect_faces(new)
if(len(new_face)!=0):
    id2=new_face[0]
    print("The person is a " + str(new_face[1]['age']) + "
year old " + new_face[1]['gender'])
    c=0
    for key in friends:
        print('Checking your friends...')
        # print(friends[key])
        face=detect_faces(friends[key])
        id1=face[0]
        if(verify_faces(id1,id2) == True):
            print('It is your friend ' + key)
            c+=1
            break

    if(c==0):
        print("Unknown person")
```

In this last block of code, the `faceId` from the `new` image is extracted and their `faceattributes` are informed to the user. A `for` loop in the friends dictionary compares the `new` image with every friend stored and tell the user the name of the friend. If not a friend, it quotes "unknown person".

6. 4. Image-to-text Reader

6.4.1. Feature Description

This feature is, essentially, an image-to-voice converter that helps the user perform elemental activities like:

- Reading price tags

- Reading product name or brand of a commodity
- Reading labels on medicine

Using optical character recognition (OCR), we can detect the text in images. Using Image Processing, we can analyze visual data and extract embedded text. The custom-made voice assistant helps us convey the text to the user.

To convert the image into text we have used Microsoft Cognitive Services' Computer Vision API which extracts text using Optical Character Recognition (OCR) from an image into a human-readable character stream. To analyze the image, we can either upload an image or specify its URL. If required, OCR corrects the inclination of the image text to recognize it and provides the frame coordinates of each word. It enables the recognition of printed and handwritten text in an image. Computer Vision can detect and extract both printed and handwritten text from images of various objects with diverse surfaces and backgrounds. Currently, printed and handwritten text recognition is available only in English.

6.4.2. Software Code

```
import requests
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image
from io import BytesIO
```

The `requests` library, which is an Apache2 Licensed HTTP library written in Python, is imported. This automates the process of conversion of query strings to URLs, or form-encode your POST data. We also imported Matplotlib, Image from PIL, BytesIO from io.

```
subscription_key = "****"
assert subscription_key
vision_base_url =
"https://westcentralus.api.cognitive.microsoft.com/vision/v
1.0/"
ocr_url = vision_base_url + "ocr"
image_url =
"https://westartwithgood.co/wp-content/uploads/2018/03/phot
ograph-handwritten-text-1.jpg"
```

```
headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params  = {'language': 'unk', 'detectOrientation': 'true'}
data    = {'url': image_url}
response = requests.post(ocr_url, headers=headers,
params=params, json=data)
response.raise_for_status()

analysis = response.json()
print(analysis)
```

Subscriber key is useful to connect to the Microsoft API and every user has a unique subscriber key to access his Microsoft Cloud account. As the subscriber key is essential for accessing the API, the key to our Microsoft Azure account is provided in the above code. The image desired to be converted into text should be given into the image URL called `vision_base_url`. By adding `ocr` to the `vision_base_url` we facilitate Optical Character Recognition request to the API. Then we can create the dictionaries to add the headers, params, and data. Next, we send a request for text extraction in the image using Vision API and the corresponding URL. This will return the recognized text as a JSON object which is printed in the console as depicted in the last block of the aforementioned code.

```
# Extract the word bounding boxes and text.
line_infos = [region["lines"] for region in
analysis["regions"]]
word_infos = []
for line in line_infos:
    for word_metadata in line:
        for word_info in word_metadata["words"]:
            word_infos.append(word_info)
word_infos
str=""
for x in word_infos :
     str = str + x["text"] + " "
print(str)

# Display the image and overlay it with the extracted text.
plt.figure(figsize=(5, 5))
image =
Image.open(BytesIO(requests.get(image_url).content))
ax = plt.imshow(image, alpha=0.5)
```

```
for word in word_infos:
    bbox = [int(num) for num in
word["boundingBox"].split(",")]
    text = word["text"]
    origin = (bbox[0], bbox[1])
    patch  = Rectangle(origin, bbox[2], bbox[3],
fill=False, linewidth=2, color='y')
    ax.axes.add_patch(patch)
    plt.text(origin[0], origin[1], text, fontsize=20,
weight="bold", va="top")
plt.axis("off")
```

We create two lists, namely, `line_infos` and `word_infos` to store the identified text. A `line_infos` list is a collection of numerous `word_infos` lists and overall bounding box coordinates. A bounding box denotes the region in which the text is detected in the image. Each `word_infos` list holds word detected and corresponding bounding box details. The coordinates of the rectangular bounding box for each word are stored in the list `word_infos`.

In the JSON response from API, we check if the lines list contains the metadata “words”. If found, they will be added into words list which will be printed.

Let’s consider the following example image with three words.

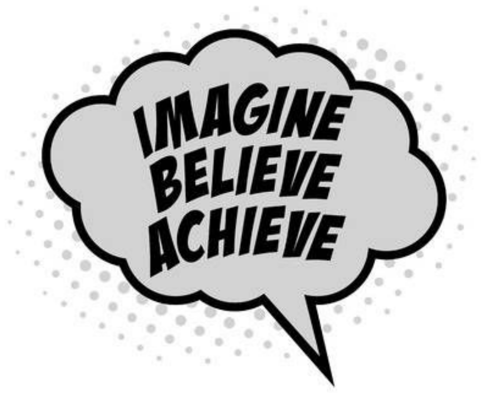


Fig. 6.1. Image to Text Conversion Example

The following is the data stored in `line_infos`:

```
"lines": [
      {
        "boundingBox": [
          131,
    93,
    317,
    82,
    320,
    141,
    134,
    152
  ],
  "text": "IMAGINE",
  "words": [
    {
      "boundingBox": [
        125,
        104,
        319,
        93,
        313,
        143,
        136,
        154
      ],
      "text": "IMAGINE"
    }
  ]
},
{
  "boundingBox": [
    143,
    142,
    306,
    135,
    309,
    188,
    146,
    196
  ],
  "text": "BELIEVE",
  "words": [
    {
      "boundingBox": [
        144,
        147,
        308,
        140,
        306,
        188,
        150,
              197
            ],
            "text": "BELIEVE"
          }
        ]
      },
      {
        "boundingBox": [
          138,
          190,
          308,
          183,
          311,
          242,
          141,
          249
        ],
        "text": "ACHIEVE",
        "words": [
          {
            "boundingBox": [
              139,
              198,
              314,
              190,
              311,
              243,
              146,
              250
            ],
            "text": "ACHIEVE"
          }
        ]
      }
    ]
```

For the word “imagine”, the following is the data stored in `word_infos`:

```
"words": [
          {
            "boundingBox": [
              125,
              104,
              319,
              93,
              313,
                    143,
                    136,
                    154
                ],
                "text": "IMAGINE"
            }
]
```

CHAPTER 7

GEOFENCE

7. 1. Feature Description

This feature envisions the creation of a virtual geographic boundary using GPS. It enables a software trigger response when the user leaves a particular preset boundary.

Establishment of the geofence is done via a web app by providing the location coordinates (latitude and longitude) and the necessary radius of the enclosure. Once the geofence is set up, we can track the person every second using the GPS module present in the wearable glasses of the user. The GPS module constantly sends the location coordinates to the server, where, we verify if the location is within the radius of the geo-fence or not. If the location coordinates are in bounds, there is no problem. However, if the given location is out of bounds, then a notification will be sent to the user and their incharge/caretaker.

7. 2. Technical Overview

The hardware components requisite for Geofence are the Raspberry Pi and GPS module. We have used Google maps API and JavaScript to facilitate user interface and easy access. We use the Google Maps API to mark the location as well as the radius of the virtual boundary in a map. We have designed a Web Page where the user can input the location's latitude and longitude in boxes. The page also consists of a dropdown with the radius (in meters) of the Geofence. In case the input boxes are null, then the page shows a "Required field" popup. For publish/subscribe messaging, we create a PubNub account and use the publisher/subscriber keys for data transactions. We have used the Pushbullet API for sending notifications to the user and guardian/incharge of the user in case of a Geofence breach.

User Instructions:

To create a geofence, the user has to enter the location's latitude and longitude along with a radius for a virtual circular boundary.

7. 3. Geofence Webpage

7.3.1. HTML Code

The HTML code consists of language attribute tag assigned to "en" to display the webpage in English. In the meta tag, we provide metadata about the HTML document. Metadata is not displayed on the webpage but is used by the browsers, search engines, or other web services to understand the content of the page to display relevant information for users in need. HTML5 introduced a method to let web designers take control over the viewport through the <meta> tag. The web page code is given below:

```
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/b
ootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263X
mFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

  <title>My map</title>
  <style>
#map {
    width: 100%;
    height: 400px;
    background-color: grey;
}
</style>
</head>
<body>
  <div id="map"></div>
  <div class="container">
    <br/>
     <h4>Details to be entered for applying geofence</h4>
     <br/>
     <br/>
    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS
-->
     <form id = "myForm">
      <div class="form-group">
  <div class="row">
    <div class="col-md-4">
     <input type="text" class="form-control"
placeholder="Lat" required>
    </div>
    <div class="col-md-2"></div>
    <div class="col-md-4">
     <input type="text" class="form-control"
placeholder="Long" required>
    </div>
  </div>
  </div>
     <div class="form-group">
    <label for="exampleFormControlSelect1">Select range in
meters</label>
    <select class="form-control" id="mySelect" required>
      <option>100</option>
      <option>200</option>
      <option>500</option>
      <option>1000</option>
    </select>
  </div>
  <br/>
  <div>
  <button type="button" onclick="myFunction()"  class="btn
btn-success">Apply Geofence</button>
  </div>
</form>
<p id="demo"></p>
  </div>
```

For the page code, Bootstrap CSS has been used to design the webpage and make it responsive. The title "My map" has been given to the page using the `<title>` tag. In the `<style>` tag, the dimensions and background colour have been mentioned. The `<body>` tag utilizes a container which has the form-group class. In the form group class, we created the rows and columns. Created under the row class, the first row will consist of the latitude input box while the longitude input box will be present in the second row. To provide the options in the dropdown we use the form-control class. The button has been created using the button attribute.

7.3.2. JavaScript Code

```
<script
src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJ
wFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.
9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUib
X39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/boo
tstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWp
i1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>
```

These lines of code have been used to download Bootstrap. This is a compiled code that we can use directly in our project base. The components of Bootstrap require JavaScript (specifically jQuery, Popper.jsto function and other JavaScript plugins) to function. To enable these requirement, we have placed the following scripts right before the closing `</body>` tag. The scripts are in the order of jQuery followed by Popper.js and then the JavaScript plugins.

```
<script
src="http://alexschneider.github.io/pushbullet-js/pushbulle
t.min.js" type="text/javascript"></script>
<script
src="https://cdn.pubnub.com/sdk/javascript/pubnub.4.21.7.js
"></script>
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=****&callb
ack=initMap"></script>
<script src="maps.js"></script>
</body>
</html>
```

This code will add the APIs of dependencies in our project. The first one is the API for Pushbullet notifications, the second one is for Pubnub API and the last one is the Google Maps API. The last script tag adds the map.js file to this HTML file.

7.3.3. Getting a Google maps API key

An API (Application Programming Interface) is a set of functions provided so that you can interact with another computer - in this case, the Google Maps server. Hence, we need to create a Google Maps API key to access the functions of the API. To load the Maps JavaScript API, we use `<script>` tag in the following way:

```
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_K
EY&callback=initMap">
</script>
```

The URL in the `script` tag is the location of a JavaScript file that loads the symbols and definitions we need for using the Maps JavaScript API. The `async` attribute allows the browser to render the rest of your website while the Maps JavaScript API loads. When the API is ready, it will call the function specified using the callback parameter.

7.3.4. Adding the map to our Webpage

In our index.html file we added the following code after `<head>` tag:

```
<style>
#map {
    width: 100%;
    height: 400px;
    background-color: grey;
}
</style>
```

This CSS code determine the map dimensions, i.e. whole width of the screen and 400px in height.

The following code will create the map:

```
<script>
    function initMap() {
        var location = {lat: #, lng: #};
        var map = new
google.maps.Map(document.getElementById('map'), {
             zoom: 10,
             center: Hyderabad
         });
     }
</script>

var geocoder = new google.maps.Geocoder();
var incident_location = "Secunderabad, Hyderabad, IN";
geocoder.geocode( { 'address': incident_location },
function(results) {
    var marker = new google.maps.Marker({
        map: map,
        position: results[0].geometry.location,
    });

});
```

Once the code is saved and the web page is refreshed, the marker will be placed on your required location.

7.3.5. Serverless Architecture

Setting up javascript code :

```
var map;
var home={lat:0, lng:0};
var radius;
function initMap() {

  var Hyderabad = {lat:17.397367 ,lng: 78.490102};
  console.log(Hyderabad.lat);
  map = new google.maps.Map(document.getElementById('map'),
{
  zoom: 10,
  center: Hyderabad,
});
};
```

In the setup, we created the variables map and radius along with a dictionary named home. The function `initMap` will be called when we use the Google API. The `home` dictionary comprises of our main city location, latitude and longitude values in key-value pairs.

The initial resolution to display the map is set by the `zoom` property (specified value is set to an integer), where `zoom 0` corresponds to the map display of the Earth fully zoomed out. At lower zoom levels, a small set of map tiles covers a wide area while the higher zoom levels, cover a smaller area with higher resolution.

```
var pubnub = new PubNub({
    subscribeKey: "****",
    publishKey: "****",
    ssl: true
})

pubnub.addListener({
    message: function(m) {
        // handle message
        var channelName = m.channel; // The channel for
which the message belongs
        var channelGroup = m.subscription; // The channel
group or wildcard subscription match (if exists)
        var pubTT = m.timetoken; // Publish timetoken
        var msg = m.message; // The Payload
        var publisher = m.publisher; //The Publisher
                 console.log(msg);
        console.log(typeof msg);
        eval('var obj='+msg);
        console.log(obj);
        var marker1 = new google.maps.Marker({
            map: map,
            position: obj,
            animation: google.maps.Animation.DROP,
            icon: {
                  path:
google.maps.SymbolPath.BACKWARD_CLOSED_ARROW,
                  scale: 10
                },
        });
        if(home.lat!=0 || home.lng!=0){
        var dist = distcal(home.lat, home.lng, obj.lat,
obj.lng);
        console.log(dist);
        if(dist>radius){
          console.log("Geofence Breach Detected!");
          sendNotification();
        }
      else{
        console.log("All cool here!");
      }
    }

    },
});
```

We create a PubNub object named `pubnub`, this will call the constructor which has the values: subscribe key, publish key and SSL (Secure Sockets Layer). Next, we call the `addListener` function using the `pubnub` object, which will handle the message received by PubNub. We will be assigning the `channelname` which creates a unique identifier for all messages belonging to that particular channel. We finally initialize the subscriber, receive a message and print the received message in the console for debugging.

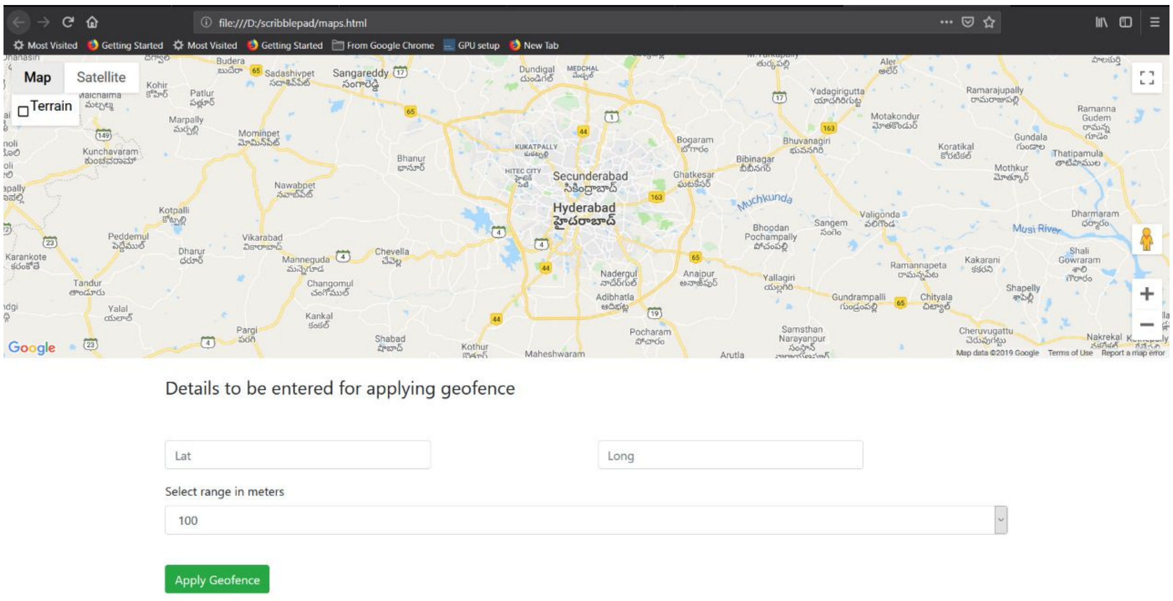
We then, create a marker which helps us mark the specified location on the map. The last received message comprises of the person's location attributes which are plotted on the map using a marker of our choice. If the received `lat` and `long` are not equal to zero then the distance calculate function is called which in turn calculates the distance from the geofence origin and prints the output in the console. If the distance obtained is greater than the radius then a geofence breach is adeptly detected followed by notification alerts.

```
function myFunction() {
var x,y,z;
x = document.getElementById("myForm").elements[0].value;
y = document.getElementById("myForm").elements[1].value;
z = document.getElementById("mySelect").value;
if(x==0 || y==0){
alert("Please enter lat and long values");
}
else{
  home = {lat: parseFloat(x), lng: parseFloat(y)};
  radius = parseFloat(z);
  console.log(home);
  console.log(radius);
  var marker = new google.maps.Marker({
      map: map,
      position: home,
      animation: google.maps.Animation.DROP,
  });
};
}

function degreesToRadians(degrees) {
  return degrees * Math.PI / 180;
}
function distcal(lat1, lon1, lat2, lon2) {
  var earthRadiusKm = 6371;
  var dLat = degreesToRadians(lat2-lat1);
  var dLon = degreesToRadians(lon2-lon1);
  lat1 = degreesToRadians(lat1);
  lat2 = degreesToRadians(lat2);

  var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                   Math.sin(dLon/2)  *  Math.sin(dLon/2)  *
Math.cos(lat1) * Math.cos(lat2);
  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
  return(earthRadiusKm * c * 1000);
}

PushBullet.APIKey= "****";
function sendNotification(){
   var devID = "****";
   var res = PushBullet.push("note", devID, null, {title:
"GeoFence Breach!", body: "We've just detected a GeoFence
Breach! "});
}

console.log("Subscribing..");
    pubnub.subscribe({
        channels: ['geofence']
    });
```

Here we use the Pushbullet API to send the notifications. The `sendNotification` function is called to deploy a notification. We pass the corresponding `devId` and the message (that needs to be sent to the user and incharge) as arguments to the procedure.

In the procedure call `myFunction` we check the values from the `myform` input handler i.e, the latitude and longitude values required to establish a geofence origin point. In case the values are not provided then an alert window is raised reading

"Please enter the values of lat and lng" thereby preventing the user from submitting the form.



Details to be entered for applying geofence

Lat

Long

Select range in meters

100

Apply Geofence

Fig. 7.1. Geofence User Interface

7. 4. Overall Data Flow

Initial setup requires the user to set the latitude, longitude, and radius of the geofence. In the back-end, three variables- map, radius and a home dictionary (comprising of latitude and longitude keys) are created. The function `initMap` sets the location, zoom percentage, and centre of the map. Then, the PubNub constructor with the subscriber and publish key is called to initiate message reception. The `addListener` function handles the assignment of the subscriber, channel names and channel group to receive any message.

The location marker constructor from Google Maps is used to mark the location. The distance from the provided location and the current location of the person is calculated using the `distcal` function. This distance is compared to the radius provided by the user while setting the geofence. If the distance is within the radius, then we print "All cool here!" in the console. If not, then "Geofence Breach Detected" is printed. In the latter case, the `sendNotification` function is called to send a PushBullet notification to the user and their guardian.

CHAPTER 8

BUILDING A COMMUNITY

8. 1. Feature Description

‘Third eye’ intends to build a community around its users. It helps them connect among themselves and with others. It allows volunteers to be online/ alert to help visually challenged people. This way, anyone can help the challenged in mere seconds, anytime, anywhere. In return for their service, volunteers are rewarded with Social Credit Points (SCP). The users can identify people (similar to them) around them who may seek assistance & spend time with a like-minded peer. (The users can discover friends through “Third Eye”)

Using Flutter framework for the mobile applications, we provide an interface for citizens to volunteer and help the ones in need. The users can send requests when they require assistance. Moreover, users can connect with others like them and socialize. We use PubNub Cloud services for request transactions and notifications.

8. 2. Third Eye App Code

```
import 'package:flutter/material.dart';
import 'package:http/src/response.dart';
import
'package:speech_recognition/speech_recognition.dart';
import 'package:thirdeye/services/mlab.dart';
import 'package:tts/tts.dart';
import 'dart:async';
```

The above code block represents the imports required. We import Material components, HTTP requests, Speech Recognition, MLab REST API, Text to Speech and Asynchronous programming modules respectively.

```
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
    theme: ThemeData(
      primaryColor: Colors.white,
      textTheme: TextTheme(
        body1: TextStyle(fontSize: 20, color: Colors.green,
fontWeight: FontWeight.bold),
      )
    ),

    home:new _homeWgt(),
   );
 }
}
```

We define a class called `MyApp` which is a stateless widget. This class holds the theme data corresponding to colors and text styles. The primary color is set to white. We define a text style called `body1` which is green in color and bold. We pass a stateful widget called `_homeWgt()` which is defined in the following code.

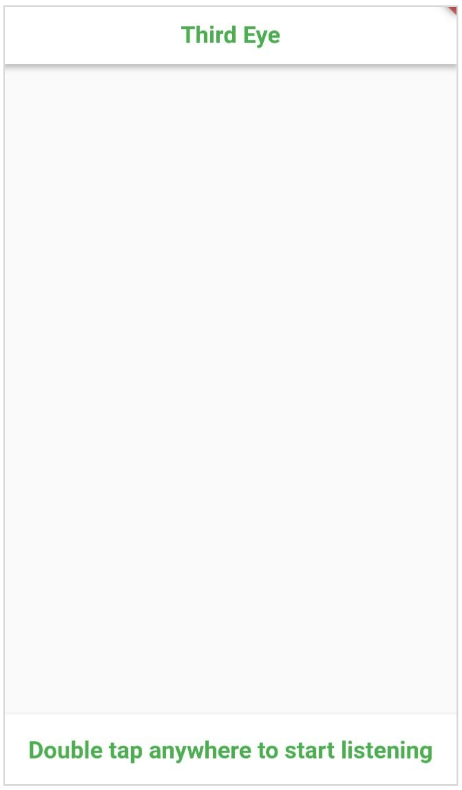
```
class _homeWgt extends StatefulWidget{
 @override
 _homeWgtState createState() => _homeWgtState();
 }
  class _homeWgtState extends State <_homeWgt> {

 SpeechRecognition _speechRecognition;
 bool _isAvailable = false;
 bool _isListening = false;

 String resultText = "Double tap anywhere to start
listening";
@override
 void initState() {
   super.initState();
   initSpeechRecognizer();
 }

 void initSpeechRecognizer(){
   _speechRecognition = SpeechRecognition();
   List<String> queries = [];

   void myLogic() async{
     if(resultText=="new request"){
         await speak("Sure! where do you need to go?");
         Future.delayed(Duration(seconds:
2),()=>mainLogic());
       }
       else if(resultText.contains("please")){
         await speak("Thanks! I'll notify once a volunteer
has been assigned.");
         Mlab data = Mlab(userName: "Anuroop", purpose:
resultText.toString());
         createPost(data);
       }
       }
   _speechRecognition.setAvailabilityHandler(
     (bool result) => setState(() => _isAvailable =
result),
   );
   _speechRecognition.setRecognitionStartedHandler(
     () => setState(() => _isListening = true),
   );
   _speechRecognition.setRecognitionResultHandler(
     (String speech) => setState((){
       debugPrint("$queries");
       resultText=speech;
     })
   );
   _speechRecognition.setRecognitionCompleteHandler(
     (){
       setState(() {_isListening = false; myLogic();});
       },
   );
   _speechRecognition.activate().then(
         (result) => setState(() => _isAvailable = result),
       );
 }
 void mainLogic(){
 if (_isAvailable && !_isListening){
   _speechRecognition.listen(locale: "en_US").then((result)
=> debugPrint(result));
   }
 }

 speak(String txt) async {
 var setLanguage = await Tts.setLanguage("en-US");
 Tts.speak(txt);
}

 @override
 Widget build(BuildContext context) {
   return new Scaffold(
     appBar: new AppBar(title: Text("Third Eye", style: 
Theme.of(context).textTheme.body1, textAlign: 
TextAlign.center,), backgroundColor: 
Theme.of(context).primaryColor,centerTitle: true,),
     body: new GestureDetector(
       onDoubleTap: ()=>mainLogic(),
     ),
     bottomNavigationBar: new BottomAppBar( child: new 
SizedBox(height: 70,
           child: new Center(child: new Text(resultText, 
style: Theme.of(context).textTheme.body1),)
         ),
         color: Theme.of(context).primaryColor,
   ),

   );
  }
}
```

The `_homeWgt` class is a stateful widget that depends on the `_homeWgtState()`. This class sets the text on the screen to "Double tap anywhere to start listening". Next, it initializes the Speech Recognizer plugin using the function `initSpeechRecognizer()`.

The `Widget build` method builds the `_homeWgt`. The User Interface of the app consists of a "Third Eye" title `AppBar` and a `BottomNavigationBar` which displays the recognized speech. A `GestureDetector` is defined to call the `mainLogic()` when the user double taps anywhere on the body of the app.

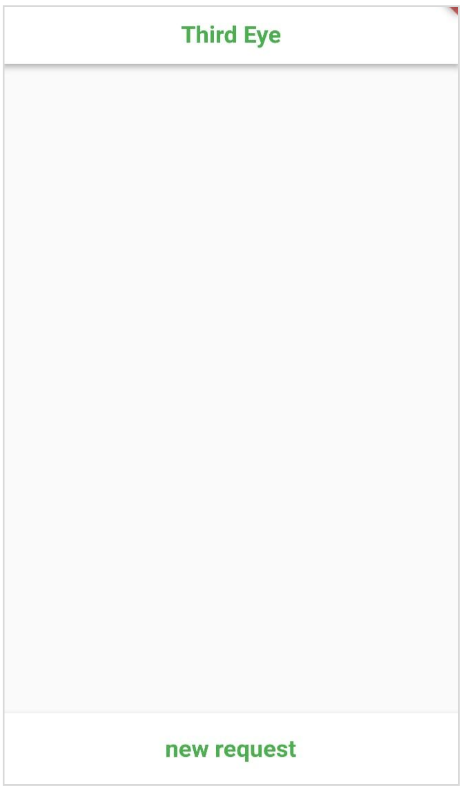
The `mainLogic()` turns on the microphone thereby enabling the user to make a request. On detecting the words "new request", a voice directive asks "Sure! where do you need to go?". The user can then provide the necessary details for his/her request after which the voice replies "Thanks! I'll notify once a volunteer has been assigned". The username, as well as the request details, are then stored in the MLab database using REST API calls.

The screenshots of the mobile app at various stages of making a request are shown below:


Third Eye

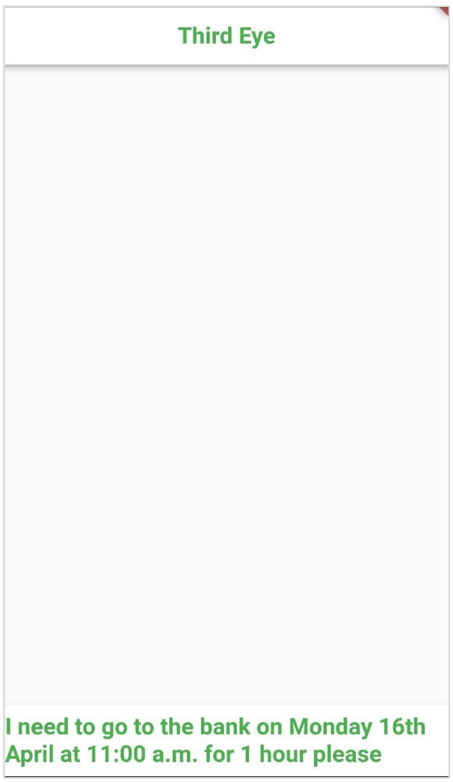
Double tap anywhere to start listening

Fig. 8.1. Initial Screen


Third Eye

new request

Fig. 8.2. User makes new request


Third Eye

I need to go to the bank on Monday 16th April at 11:00 a.m. for 1 hour please

Fig. 8.3. After request completion

8. 3. MLab Code

```
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'dart:async';
import 'dart:io';
```

The above-mentioned lines of code are used for the sole purpose of importing various libraries. The `convert` module converts Strings to JSON. The `http.dart` library facilitates HTTP transactions back and forth the app. The `async` and `io` libraries enable asynchronous programming and user input-output functions respectively.

```
String url =
"https://api.mlab.com/api/1/databases/thirdeye/collections/
user_data?apiKey=****";
```

This is the URL used to perform REST API calls to the MLab database. It essentially consists of details such as the name of collection and API keys for authentication.

```
Future<http.Response> createPost(Mlab post) async{
 final response = await http.post('$url',
     headers: {
       HttpHeaders.contentTypeHeader: 'application/json',
       HttpHeaders.authorizationHeader : ''
     },
     body: mlabToJson(post)
 );
 return response;
}
```

The most important part of the `Mlab.dart` code is the above method. It performs an HTTP POST call to the MLab server by providing the details of a new request in JSON format. When a new request is made by the user, the database is immediately updated so that volunteers can be notified. The method then returns a transaction status. For example, a 404 is returned for a failed transaction while a 200 OK is returned after a successful database updation.

```
Mlab mlabFromJson(String str) {
   final jsonData = json.decode(str);
   return Mlab.fromJson(jsonData);
}
```

```
String mlabToJson(Mlab data) {
   final dyn = data.toJson();
   return json.encode(dyn);
}

class Mlab {
   String userName;
   String purpose;

   Mlab({
       this.userName,
       this.purpose,
   });

   factory Mlab.fromJson(Map<String, dynamic> json) => new
Mlab(
       userName: json["user_name"],
       purpose: json["purpose"],
   );

   Map<String, dynamic> toJson() => {
       "user_name": userName,
       "purpose": purpose,
   };
}
```

In the above code, we declare and define a class (user-defined data type) called Mlab. In frameworks like Flutter, it is common to define classes to handle data transactions between the app and the backend. Such classes are popularly known as models. In the above code, an Mlab model is created by clearly specifying the various attributes exchanged between the frontend and the backend. The attributes exchanged are `userName` and `purpose`. While the name of the visually challenged person is contained in the `userName` variable, his purpose to request for assistance is contained in the variable `purpose`. The methods `MlabToJson` and `MlabFromJson`, as discussed above, facilitate data handling. `MlabToJson` converts String data types to JSON before performing a POST operation for database side data handling. The `MlabFromJson` method converts received JSON data to Mlab elements for application end data handling.

8. 4. Community App Code

```
import 'package:community/services/mlab.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
          home: new homePage(),
   );
 }
}
```

The first three lines of the above code import various libraries necessary for the smooth functioning of the app. The `mlab.dart` library is a data handling model. The `material.dart` library facilitates fast and easy UI building with the help of predefined widgets or components. The `http.dart` library is used to perform REST API calls to fetch data from the database.

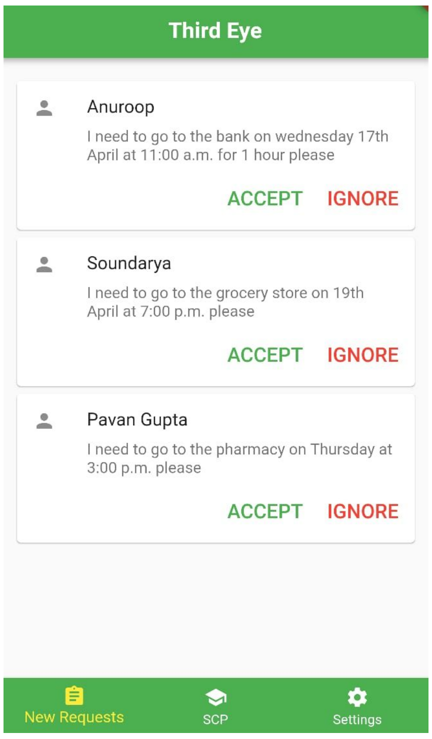
The `void main` method returns the `runApp` process which in turn returns the `MyApp` stateless widget. The `MyApp` stateless widget is declared as a material app to facilitate the usage of material widgets. The home attribute of the material app is the first screen visible to the user. Here, in this case, the `homePage` widget is first rendered as the home screen.

```
class homePage extends StatefulWidget {
 @override
 homeState createState() => new homeState();
}

class homeState extends State<homePage> {
int currentTab = 0;
PageOne pageOne = new PageOne();
PageTwo pageTwo = new PageTwo();
PageThree pageThree = new PageThree();
List<Widget> pages;
Widget currentPage;
```

The `homePage` widget is a stateful widget with a `homeState`. It is essentially a widget with a bottom navigation bar and three tab views to facilitate navigation between the three pages. Page one consists of a list view of requests with "ACCEPT" and "IGNORE" buttons for the volunteer. Page two reveals the Social Credit Points of a volunteer and page three is the Settings page with volunteer details and a toggle switch to turn on and turn off notifications for new requests.

```
  Widget   createList(BuildContext   context,   AsyncSnapshot
snapshot) {
   List<Mlab> values = snapshot.data;
   return new ListView.builder(
     itemCount: values.length,
     itemBuilder: (BuildContext context, int index) {
       return new myCard(
         title: values[index].userName,
         desc: values[index].purpose,
       );
     },
   );
 }
}
```

The createList widget takes context and snapshot of the returned REST API call as arguments to build a list view of all the received requests. For each item in the list, the list view builder initializes a card with title corresponding to the username of the requesting user and description corresponding to the purpose for the request.


Third Eye

Anuroop

I need to go to the bank on wednesday 17th April at 11:00 a.m. for 1 hour please

ACCEPT IGNORE

Soundarya

I need to go to the grocery store on 19th April at 7:00 p.m. please

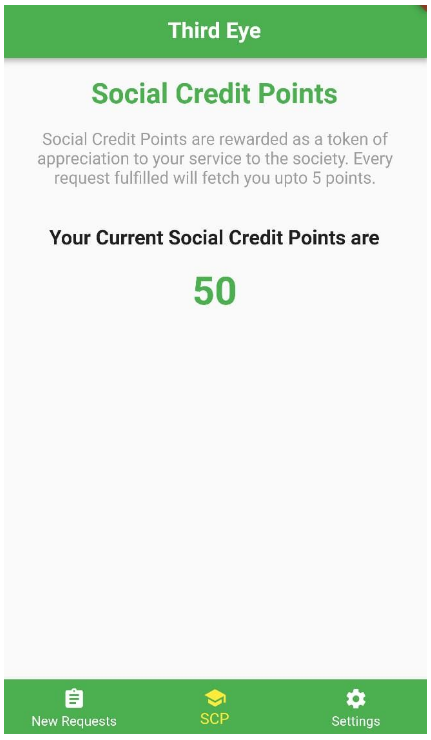
ACCEPT IGNORE

Pavan Gupta

I need to go to the pharmacy on Thursday at 3:00 p.m. please

ACCEPT IGNORE

Fig. 8.4. Home Screen of Community App


Third Eye

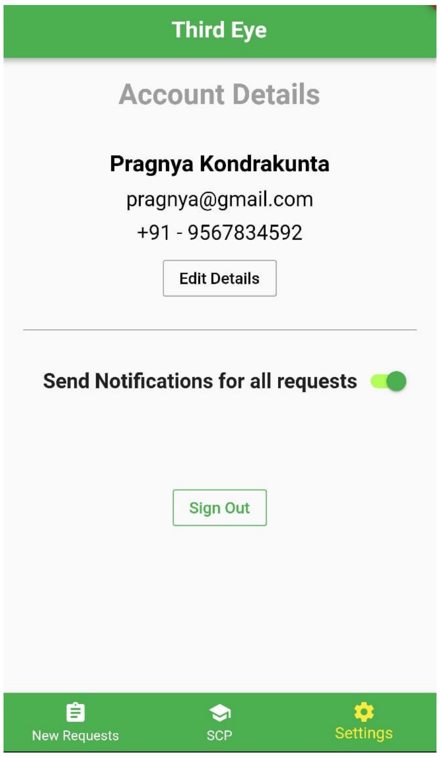
Social Credit Points

Social Credit Points are rewarded as a token of appreciation to your service to the society. Every request fulfilled will fetch you upto 5 points.

Your Current Social Credit Points are

50

Fig. 8.5. Social Credit Points Screen


Third Eye

Account Details

Pragnya Kondrakunta

pragnya@gmail.com

+91 - 9567834592

Edit Details

Send Notifications for all requests

Sign Out

Fig. 8.6. Account Details and Settings Screen

CHAPTER 9

DESIGNING THE WEARABLE

9. 1. Choosing the Design Structure

A vital element in the conceptualization of this project is accessibility. The irrefutable need for a convenient device compelled us to design a wearable that is comfortable and can be easily operated. The overall structure of the wearable proves to be particularly crucial. The abstraction of a wrist band or a waist belt were dismissed due to their inflexibility and indefinite point of view of the camera. A prospective proposal included a belt-like structure around the chest of the user. However, the most commendable design was that of a spectacle. The spectacles (or glasses) enable flexibility due to the head rotation and easy change of point of view. Also, since glasses are a typical accessory for the targeted users, the familiarization and adjustment to this new smart accessory will be easier.

9. 2. 3D Printing the Design

The wearable device has been designed in SketchUp as a 3D model and extracted as a stereolithography file (.stl). This model was printed in a FlashForge 3D printer "Dreamer". The Layer resolution is 100~500 microns, while the positioning Precision in XY plane is 11 microns and 2.5 microns in Z plane. The nozzle diameter was 0.4mm. The material used for the wearable is a white color PLA (Polylactic Acid).

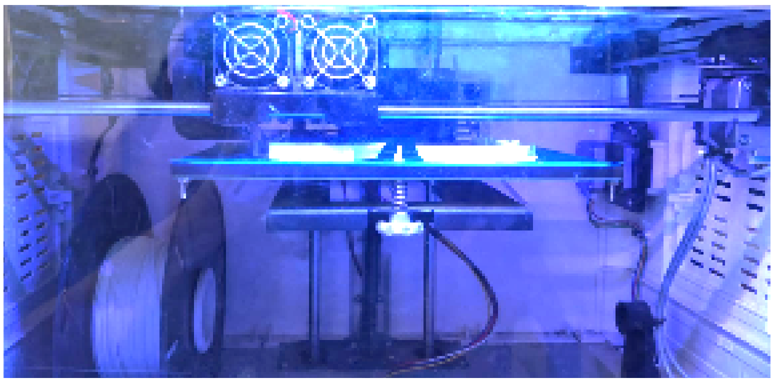


Fig. 9.1. 3D Printing the Wearable

9. 3. Version 1 - The First Design

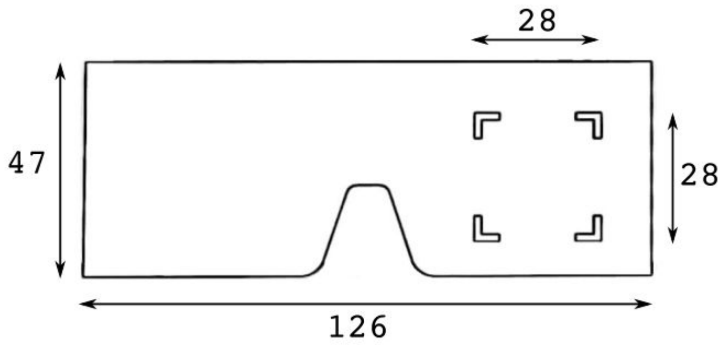


Fig. 9.2. Version 1 - Front View

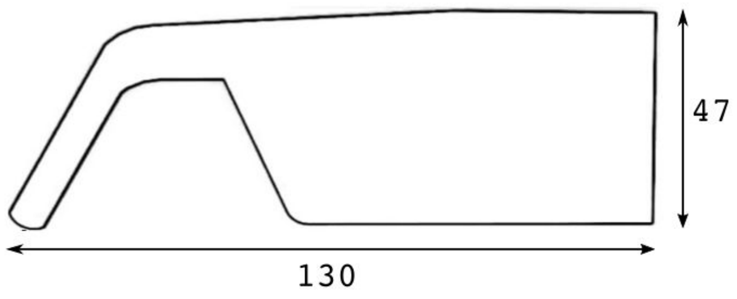


Fig. 9.3. Version 1 - Right View

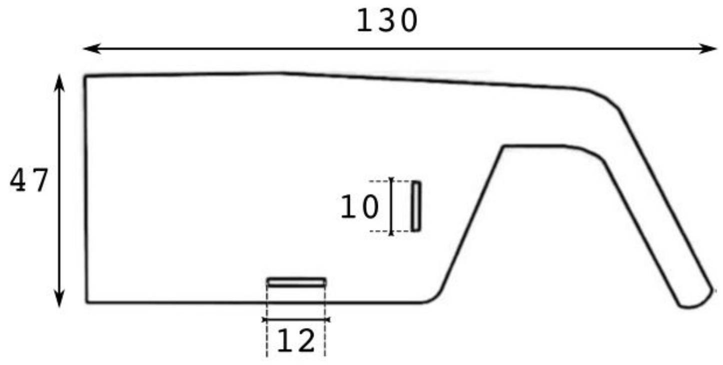


Fig. 9.4. Version 1 - Left View

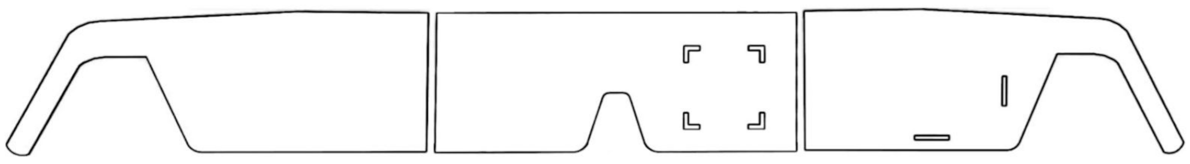


Fig. 9.5. Version 1 - Panorama View

Modifications for the next Design:

While the quality and strength of the initial design were satisfactory, we noticed that the frontal frame was a little narrow. To ensure that the wearable fits everyone, we can widen the frame in the next design. Also, in the next model, we aim to introduce a sliding mechanism as a separate enclosure for each of the various components (Raspberry Pi Zero W, Pi camera and GPS Module). Moreover, we can reduce the frame thickness (currently 3mm) to reduce the weight of the wearable.

9. 4. Version 2 - Enhanced Comfort

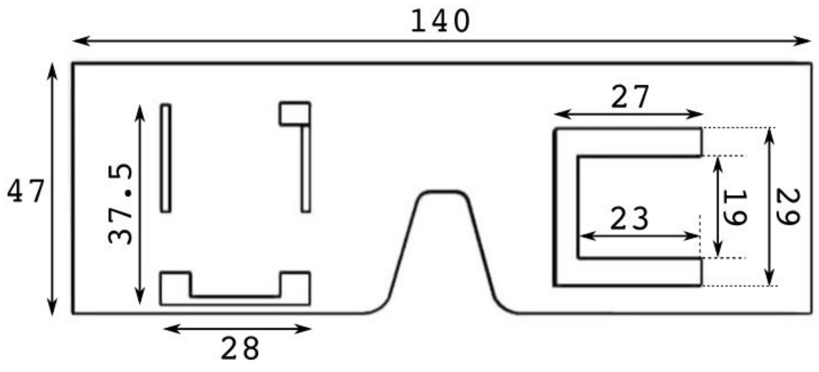


Fig. 9.6. Version 2 - Front View

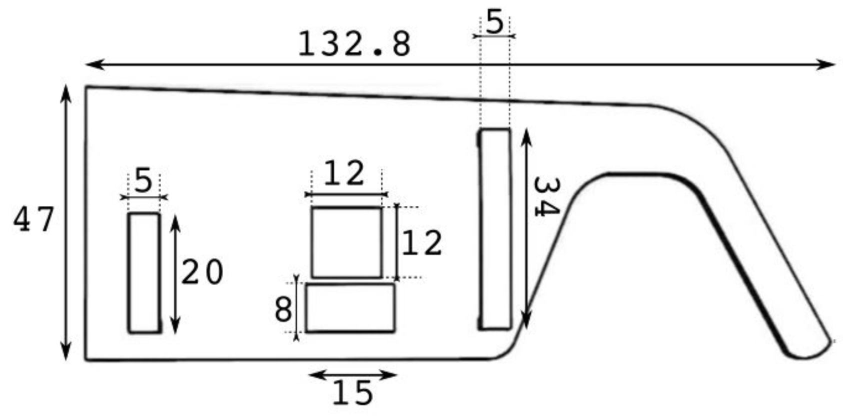


Fig. 9.7. Version 2 - Left View

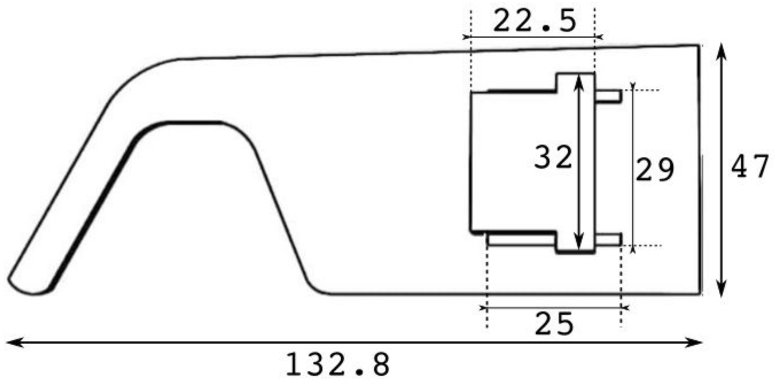


Fig. 9.8. Version 2 - Right View

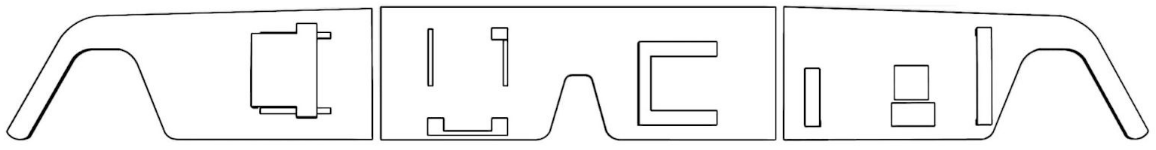


Fig. 9.9. Version 2 - Panorama View

Modifications for the next Design:

The thickness has been reduced by 0.5mm (i.e. it is 2.5mm now) and can be further reduced by another 0.5mm. The wearable was tested by numerous volunteers and was found to be comfortable, there is no need to further widen the frontal frame. A specific enclosure was designed to fit each component accurately. However, the lack of precision during printing made the closure unfit for the Raspberry Pi. To allow room for error and free movement, the closure for every component should be redesigned slightly for the next model.

9. 5. Version 3 - Final design

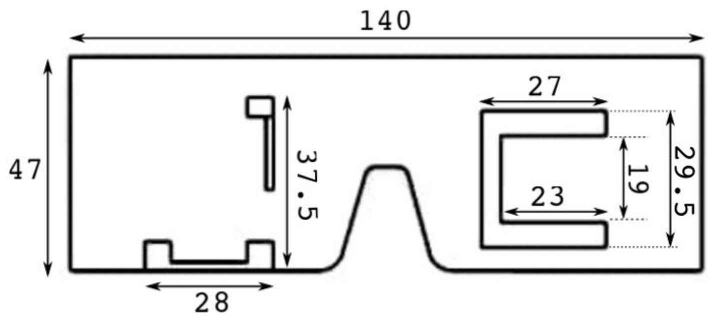


Fig. 9.10. Version 3 - Front View

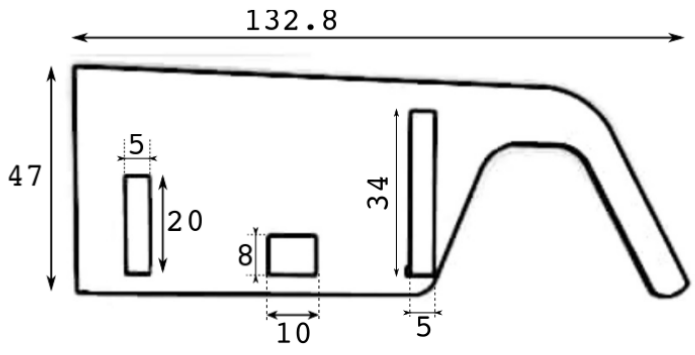


Fig. 9.11. Version 3 - Left View

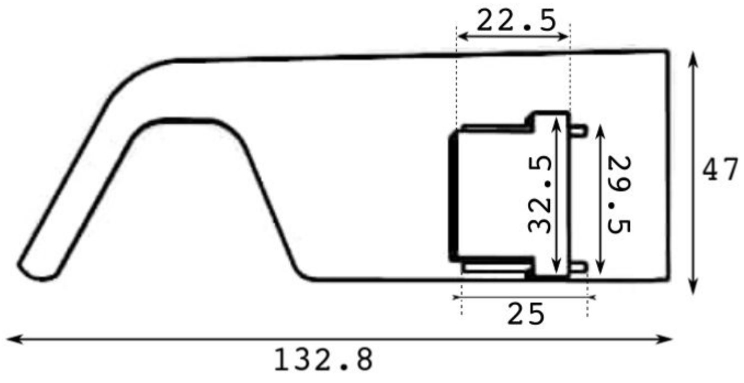


Fig. 9.12. Version 3 - Right View

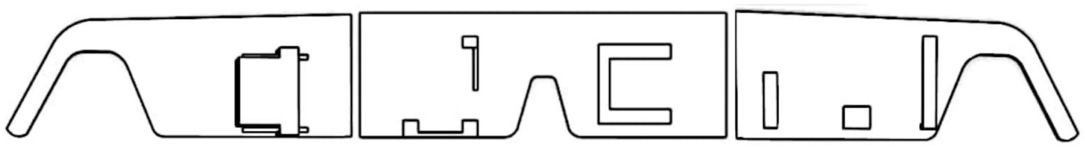


Fig. 9.13. Version 3 - Panorama View

CHAPTER 10

PUTTING IT ALL TOGETHER

10. 1. End to End Code

```
import RPi.GPIO as GPIO
from time import sleep
import requests
import json
import picamera
from cloudinary import config
from cloudinary.uploader import upload
from gtts import gTTS
import subprocess
```

Imports are a very important aspect of a project's code. In this project, we utilize various libraries such as the `RPi.GPIO` library for accessing the hardware GPIO pins of a Raspberry Pi. The `gTTS` library is the Google Text to Speech library used to convey voice directives to the visually challenged.

```
config(
 cloud_name = '****',
 api_key = '****',
 api_secret = '****'
)
```

The above code configures the Cloudinary library and authenticates the wearable to the Cloudinary platform with the `api_key` and `api_secret` parameters along with the `cloud_name` parameter.

```
GPIO.setmode(GPIO.BCM)

GPIO.setup(23, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(24, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(25, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```

The above code depicts the General Purpose Input Output pin programming in our Raspberry Pi. Three push buttons are connected to GPIO pins 23, 24 and 25 to activate various features available onboard the wearable. Button 1 connected to pin 23 activates the Object Detection feature of the wearable. Button 2 connected to pin 24 activates the Know your Friend feature of the wearable and Button 3 connected to pin 25 activates the Image to Text feature of the wearable. GPIO pin numbers 23, 24 and 25 are thus declared as input pins.

```
def upload_files():
   print("--- Upload a local file")
   response = upload("image.jpg")
   url = response['url']
   print(url)
   return(url)

def speak(txt):
   tts = gTTS(txt)
   tts.save("hello.mp3")

subprocess.Popen(['omxplayer','-o','alsa','hello.mp3']).wai
t()
```

The speak() function takes a String argument and generates voice output for the given text. The procedure for converting a string to voice with the help of Google text to speech service can be evidently observed in the above code.

```
def obj_det():
   speak("Object Detection Activated")
   print("Object Detection Activated")
   cam = picamera.PiCamera()
   cam.capture('image.jpg')
   cam.close()
   img = upload_files()
   img =
"http://res.cloudinary.com/shannu/image/upload/v1555585593/
bsyzetny3jogxocidp9p.png"
   headers = { 'Ocp-Apim-Subscription-Key' : '****' }
   api_url =
'https://westcentralus.api.cognitive.microsoft.com/vision/v
2.0/describe?maxCandidates=1'
   response=
requests.post(api_url,headers=headers,json={'url': img})
   res=response.json()
   print(res['description']['captions'][0]['text'])
   speak(res['description']['captions'][0]['text'])

def detect_faces(url):
   headers = { 'Ocp-Apim-Subscription-Key' : '****' }
   params = {
           'returnFaceId':'true',
           'returnFaceLandmarks':'false',
           'returnFaceAttributes':'age,gender',
   }
   face_api_url =
'https://westcentralus.api.cognitive.microsoft.com/face/v1.
0/detect'
   image_url = url
   response=
requests.post(face_api_url,params=params,headers=headers,js
on={'url':image_url})
   faces=response.json()
   if(len(faces)>0):
       retlist=[]
       id=faces[0]['faceId']
       att = faces[0]['faceAttributes']
       retlist.append(id)
       retlist.append(att)
       return(retlist)
   else:
       print('Please retake image.')
       speak('plese retake image')
       return('no id')

def verify_faces(id1,id2):
   headers = { 'Ocp-Apim-Subscription-Key' : '****' }
   face_api_url =
'https://westcentralus.api.cognitive.microsoft.com/face/v1.
0/verify'
   response=
requests.post(face_api_url,headers=headers,json={'faceId1':
id1,'faceId2':id2})
   res=response.json()
   if(res['isIdentical']==True):
       return(True)
   else:
       return(False)

def kyf():
   speak("Know Your Friend Activated")
   print("Know Your Friend Activated")
   cam = picamera.PiCamera()
   cam.capture("image.jpg")
   cam.close()
   img = upload_files()
   friends= { 'anuroop':
'http://res.cloudinary.com/shannu/image/upload/v1533544913/
WIN_20180806_14_11_01_Pro_gqqeec.jpg', 'pragnya':
'http://res.cloudinary.com/shannu/image/upload/v1533544769/
WIN_20180806_14_08_25_Pro_qvfjmb.jpg'}
   img =
'http://res.cloudinary.com/shannu/image/upload/v1533624870/
WIN_20180807_12_24_06_Pro_j1trjd.jpg'
   #img =
'http://res.cloudinary.com/shannu/image/upload/v1533624859/
WIN_20180807_12_23_25_Pro_pabetq.jpg'
   new_face=detect_faces(img)
   if(len(new_face)!=0):
       id2=new_face[0]
       print("The person is a " + str(new_face[1]['age']) +
" year old " + new_face[1]['gender'])
       speak("The person is a " + str(new_face[1]['age']) +
" year old " + new_face[1]['gender'])
       c=0
       for key in friends:
           print('Checking your friends...')
           speak('Checking your friends database for
resembelance')
           face=detect_faces(friends[key])
           id1=face[0]
           if(verify_faces(id1,id2) == True):
               print('It is your friend ' + key)
               speak('It is your friend ' + key)
               c+=1
               break

       if(c==0):
           print("Unknown person")
           speak("Unknown person")

def img2txt():
   speak("Image to Text Activated")
   print("Image to Text Activated")
   cam = picamera.PiCamera()
   cam.capture("image.jpg")
   cam.close()
   img = upload_files()
   img =
"http://res.cloudinary.com/shannu/image/upload/v1533627380/
WIN_20180807_13_05_44_Pro_wyxa09.jpg"
   headers = {'Ocp-Apim-Subscription-Key': "****"}
   params  = {'mode': 'Handwritten'}
   data    = {'url': img}
   response =
requests.post("https://westcentralus.api.cognitive.microsof
t.com/vision/v2.0/recognizeText", headers=headers,
params=params, json=data)
   response.raise_for_status()
   operation_url = response.headers["Operation-Location"]
   analysis = {}
   while "recognitionResult" not in analysis:
       response_final = requests.get(
           response.headers["Operation-Location"],
headers=headers)
       analysis = response_final.json()

print(analysis['recognitionResult']['lines'][0]['text'])

speak(analysis['recognitionResult']['lines'][0]['text'])
       sleep(1)

while(True):
   btn1 = GPIO.input(23)
   btn2 = GPIO.input(24)
   btn3 = GPIO.input(25)
   if(btn1==False):
       obj_det()
   elif(btn2==False):
       kyf()
   elif(btn3==False):
       img2txt()
   else:
       print("Device Idle")
       sleep(5)
```

The actual program execution starts at the aforementioned While loop. It is an infinite loop asynchronously listening to push button triggers. Depending on the button pressed, a corresponding function is executed. The working of each feature has been delineated in prior chapters.

CHAPTER 11

RESULTS AND DISCUSSION

11. 1. Glimpse of the Wearable

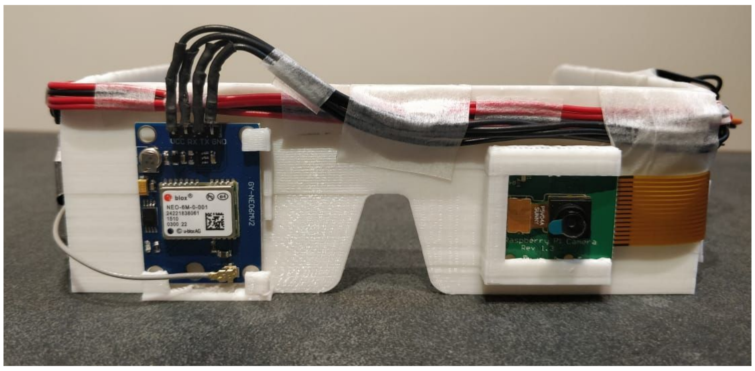


Fig. 11.1. Front View

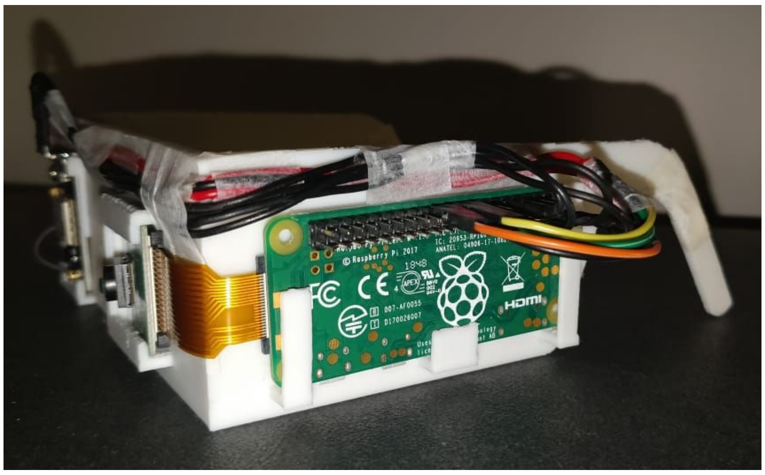


Fig. 11.2. Left View

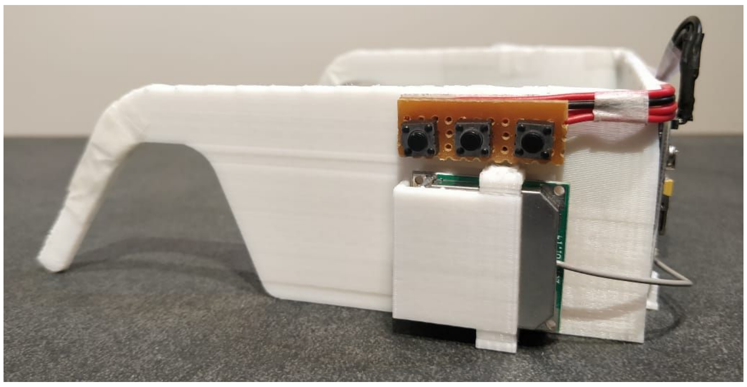


Fig. 11.3. Right View

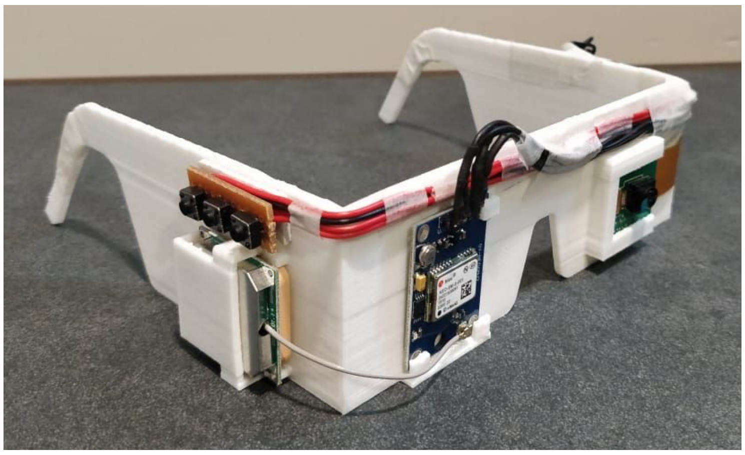


Fig. 11.4. Isometric View

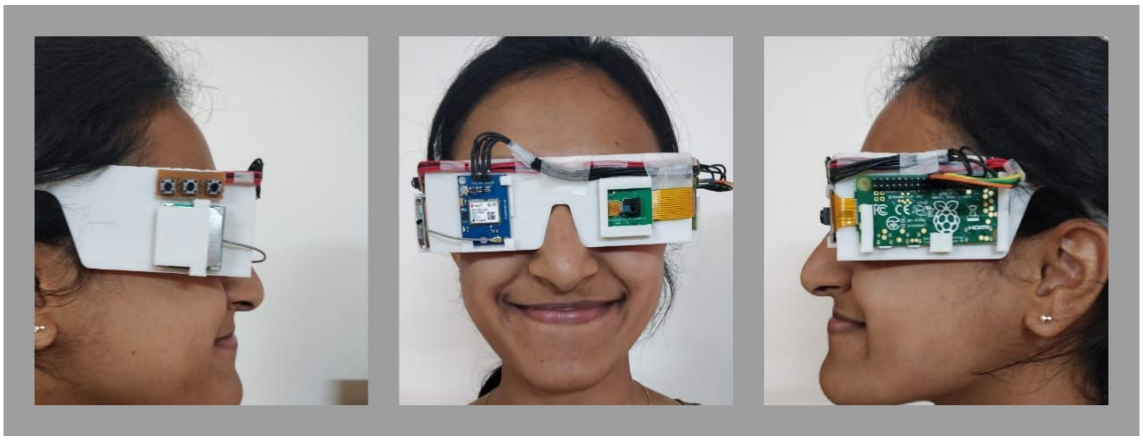
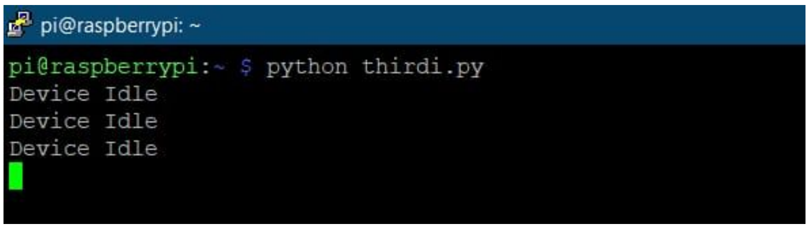


Fig. 11.5. Person wearing the device



```
pi@raspberrypi:~ $ python thirdi.py
Device Idle
Device Idle
Device Idle
```

Fig. 11.6. Device Idle State

The device is initially in an idle state when no button is pressed and is asynchronously listening for a button press. Fig. 11.6. represents this state.

11. 2. Object Detection Results

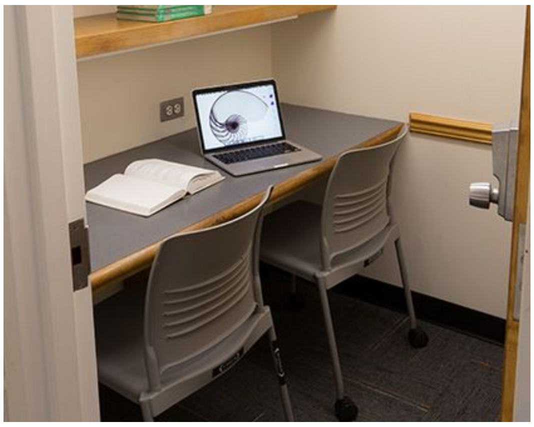
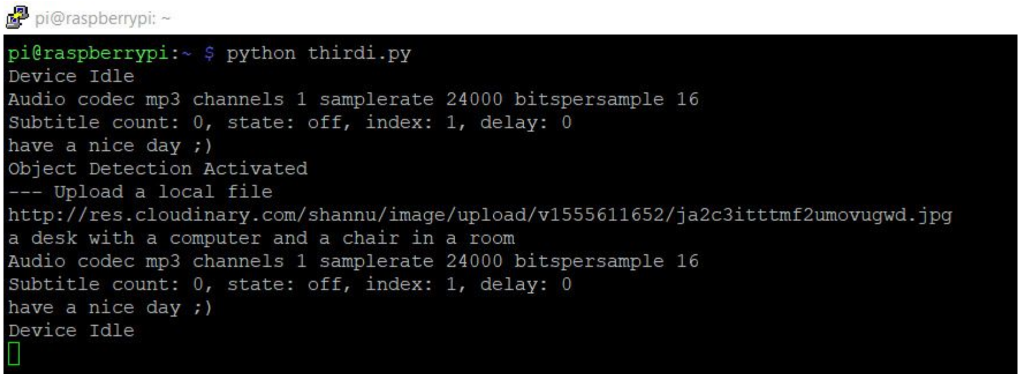


Fig. 11.7. Input Image clicked for Object Detection



```
pi@raspberrypi:~ $ python thirdi.py
Device Idle
Audio codec mp3 channels 1 samplerate 24000 bitspersample 16
Subtitle count: 0, state: off, index: 1, delay: 0
have a nice day ;)
Object Detection Activated
--- Upload a local file
http://res.cloudinary.com/shannu/image/upload/v1555611652/ja2c3itttmf2umovugwd.jpg
a desk with a computer and a chair in a room
Audio codec mp3 channels 1 samplerate 24000 bitspersample 16
Subtitle count: 0, state: off, index: 1, delay: 0
have a nice day ;)
Device Idle
```

Fig. 11.8. Result for Object Detection

When the first button is pressed, Object Detection is activated and a voice directive is given to the user. The device then captures an image, as shown in Fig. 11.7., for analysis. The user receives an audio output "**a desk with a computer and a chair in a room**" as shown in the Fig. 11.8.

11. 3. Know Your Friend Results

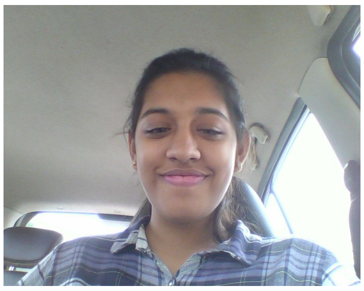
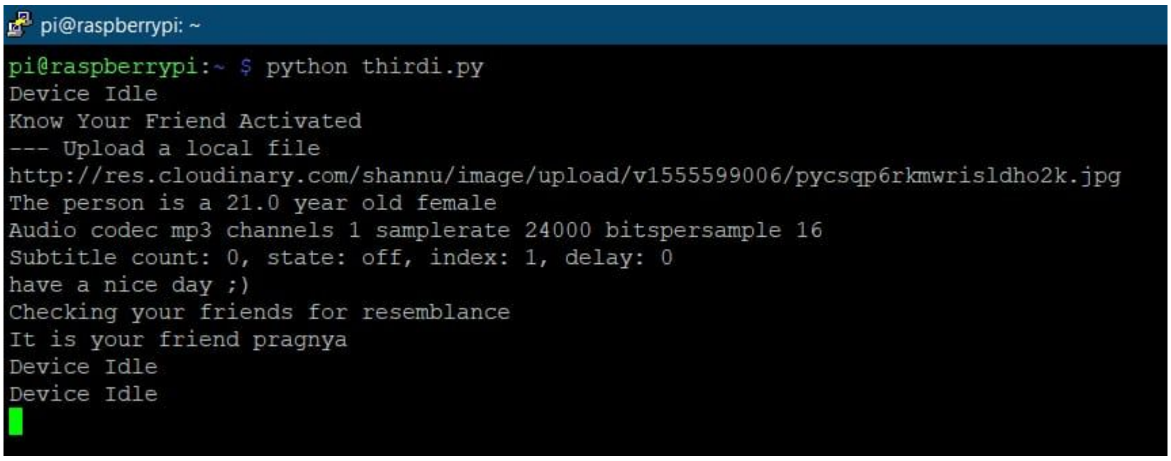


Fig. 11.9. Input Image clicked for Know Your Friend



```
pi@raspberrypi:~ $ python thirdi.py
Device Idle
Know Your Friend Activated
--- Upload a local file
http://res.cloudinary.com/shannu/image/upload/v1555599006/pycsqp6rkmwrisldho2k.jpg
The person is a 21.0 year old female
Audio codec mp3 channels 1 samplerate 24000 bitspersample 16
Subtitle count: 0, state: off, index: 1, delay: 0
have a nice day ;)
Checking your friends for resemblance
It is your friend pragnya
Device Idle
Device Idle
```

Fig. 11.10. Result for Know Your Friend

When the second button is pressed, the Know Your Friend feature is activated and a voice directive is given to the user. The device then captures an image, as shown in Fig. 11.9., for analysis. An initial audio output of "**The person is a 21.0 year old female**" is received. Furthermore, it checks if the person is the user's friend and outputs "**It is your friend pragnya**" via a voice directive as shown in Fig. 11.10.

11. 4. Image to Text Results

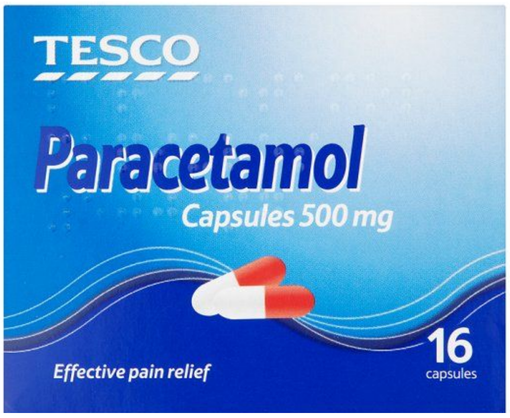
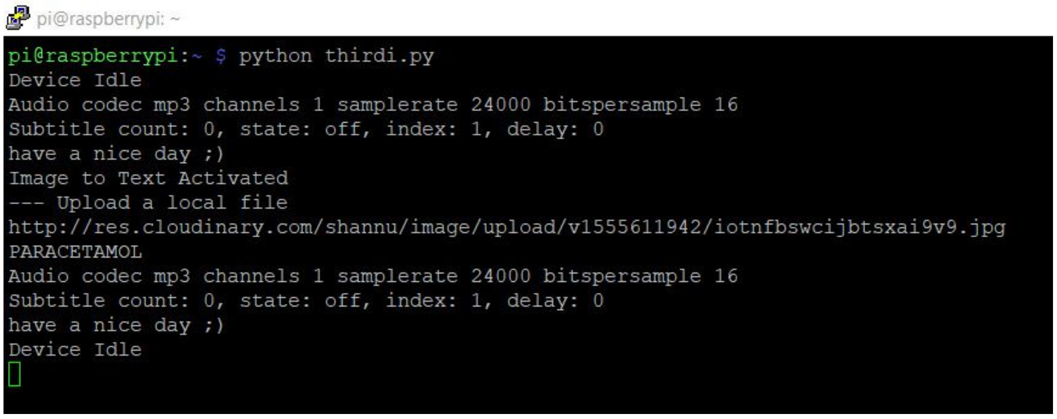


Fig. 11.11. Input Image clicked for Image to Text

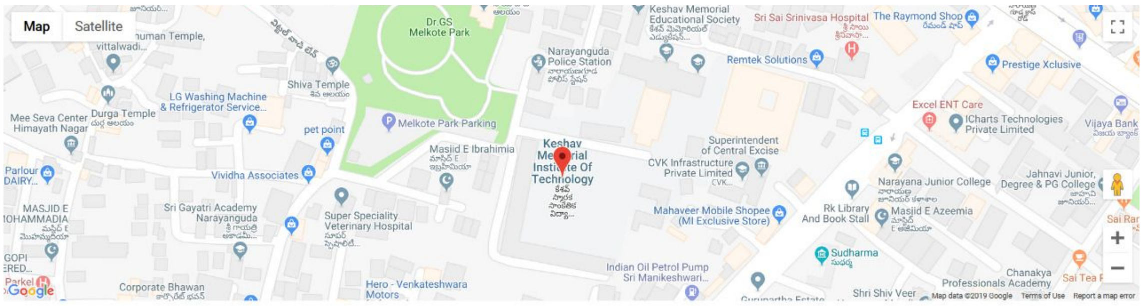


```
pi@raspberrypi: ~
pi@raspberrypi:~ $ python thirdi.py
Device Idle
Audio codec mp3 channels 1 samplerate 24000 bitspersample 16
Subtitle count: 0, state: off, index: 1, delay: 0
have a nice day ;)
Image to Text Activated
--- Upload a local file
http://res.cloudinary.com/shannu/image/upload/v1555611942/iotnfbswcijbtsxai9v9.jpg
PARACETAMOL
Audio codec mp3 channels 1 samplerate 24000 bitspersample 16
Subtitle count: 0, state: off, index: 1, delay: 0
have a nice day ;)
Device Idle
```

Fig. 11.12. Result for Image to Text

When the third button is pressed, Image to Text is activated and a voice directive is given to the user. The wearable then captures an image, as shown in Fig. 11.11., for analysis. The output received is "**PARACETAMOL**" as shown in Fig. 11.12.

11. 5. Geofence Results



Details to be entered for applying geofence

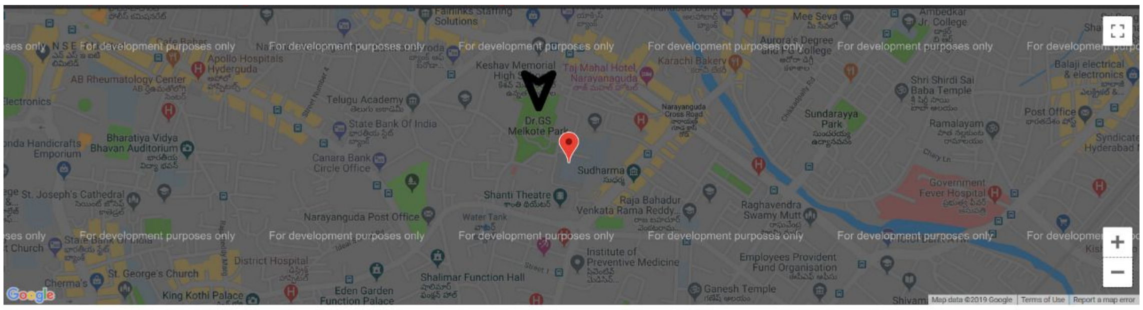
17.3971932

78.490038

Select range in meters

1000

Fig. 11.13. Applying Geofence Location and Radius



Details to be entered for applying geofence

17.3971932

78.490038

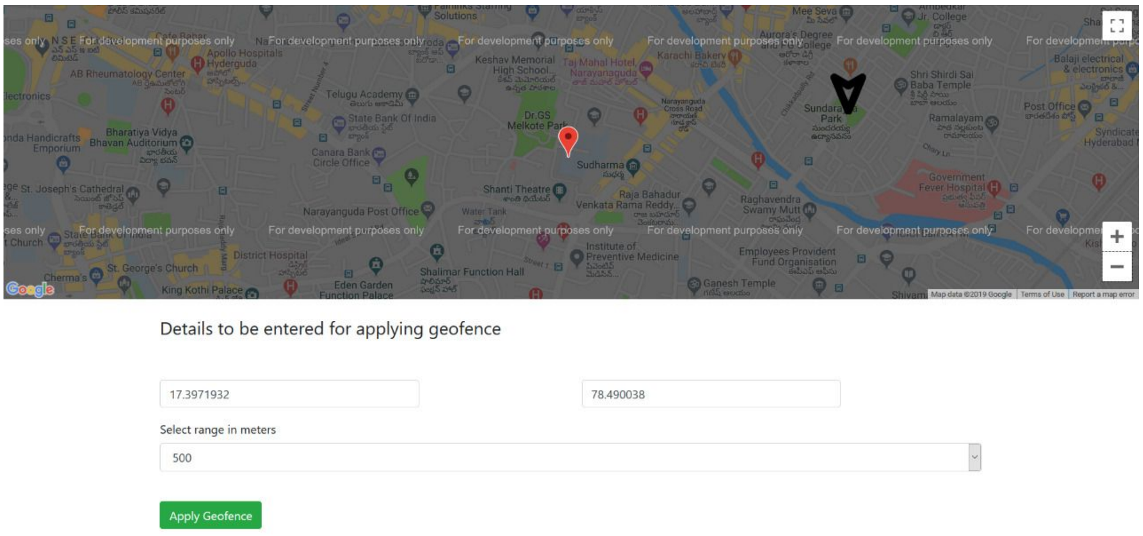
Select range in meters

1000

Fig. 11.14. Location fetched within Geofence Boundary

To create a geofence, the guardian has to enter the location's latitude and longitude along with a radius for a virtual circular boundary.

Once the geofence has been set, the GPS location of the visually challenged is continuously monitored. As long as the location falls within the radius of the geofence, no action is taken as shown in Fig. 11.14.


Details to be entered for applying geofence

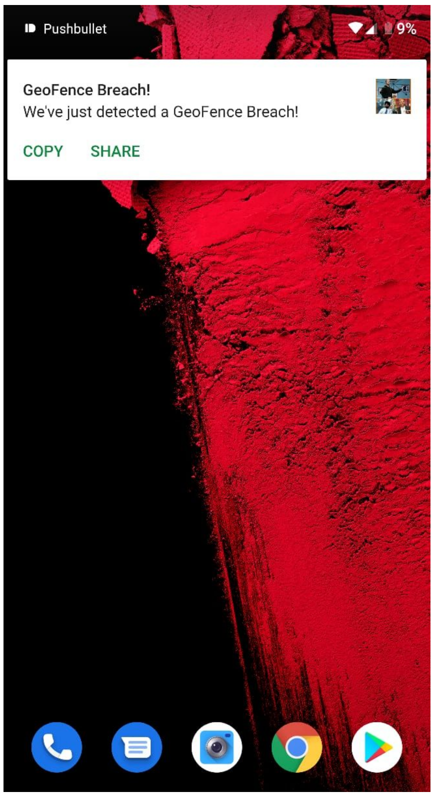
17.3971932

78.490038

Select range in meters

500

Fig. 11.15. Location fetched outside Geofence Boundary



GeoFence Breach!
We've just detected a GeoFence Breach!

Fig. 11.16. Geofence Breach Notification sent to Guardian

However, once the user crosses the boundary, a geofence breach notification is sent to the guardian as shown in Fig. 11.15. And Fig. 11.16.

CHAPTER 12

CONCLUSION & FUTURE SCOPE

12. 1. Conclusion

In today‘s world, disability of any kind can be hard and it is the same in the case of blindness. Blind people are generally left underprivileged and offering vision to them is very difficult. This project mainly focused on building an innovative-assistive technology that promotes the visually impaired to live with greater comfort and confidence.

A new Raspberry Pi Zero W based system called - **Third Eye** as an assistance for the needy has been proposed and developed. In this research, we successfully accomplished building a wearable device that is light in weight, compact and portable. This system offered a simple electronic guidance embedded vision system which is configurable and efficient. This technology described several features in-depth that mainly included Object Detection, Identifying a Friend, an Image-to-Text Reader, Geofence and building a volunteer Community for the blind. This wearable was proven to be comfortable and easily accessible. It helps blind and visually impaired people to be highly self-dependent and helped in enhancing the virtue of life by assisting their mobility regardless of where they are: indoors or outdoors.

This wearable resolves most of the problems of existing technologies. In this era of modern technology, there are so many instruments and smart devices for visually impaired people but most of them have a common drawback which is the need for excessive training to use. This prototype brings intelligence to a common accessory sported by the blind- the spectacles. With only 3 hardware buttons and voice commands, Third Eye presents a set-up which requires almost nil training for usage. With minimal improvements in the prototype, the proposed architecture will significantly benefit the community. The concept of "quality of life" for the visually challenged, through technological means, is explored and their relevance as practical implementation is assessed.

12. 2. Future Scope and Overcoming the Limitations

1. Future work will be focused on enhancing the performance of the system and reducing the load on the user.
2. Minimal changes, such as replacing the opaque covering with a transparent lens, can broaden the user-base for this wearable. Circumstantial inhibitions like foggy mornings will encourage the use among individuals having low visibility. This system also finds indisputable (short-term) utilization by patients suffering from various eye ailments like cataract, xerophthalmia, post eye operative situations.
3. This system, after modification into a more sophisticated version of itself, can be used as a navigation system by including GPS navigation feature on the wearable for geological explorations.
4. We can integrate a common platform for all the data acquired from the environment from different sources
5. Industrial applications can be devised and enhanced like robots and machinery.
6. Algorithms can be further highly optimized to run on smartphones, hence, providing high accuracy and low latency.
7. Collaboration with third-party applications like Uber / Ola / Swiggy/ Zomato can prove to be handy and establish active participation of the blind with emerging technologies.
8. Image processing can be included for calculating the physical dimensions of obstacles and object patterns, thereby, keeping the user informed about his trajectory.
9. The system's ability to correctly identify or recognize user phrases can be improved using Artificial Intelligence, also furthering apt and adept responses from the database.
10. The principles of monopulse radar can be utilized for determining long range target objects.

11. Object detection algorithm utilizes 100% CPU which overloads the Raspberry Pi Zero W and, thus, in the future a more efficient single board computer is recommended to be used.
12. The concept of a battery management system or a power management system for the entire wearable can be introduced in the future.
13. The ergonomics of the wearable can be further improved towards a sleeker design and finish.
14. Subjects like the centre of mass and the centre of gravity will be taken into consideration to provide efficient weight management of the components placed on the wearable.
15. Owing to the modifications in design and close inspection of user feedback the overall weight and experience of the wearable can be tremendously improved.

REFERENCES

[1] Andreas Ess1, Bastian Leibe1, and Konrad Schindler, “SURF: A Mobile Vision System for Robust Multi-Person Tracking”, IEEE Conference on Computer Vision, 2016.

[2] Ajeet Ram Pathak, Manjusha Pandey, and Siddharth Rautaray, “Application of Deep Learning for Object Detection”, International Conference on Computer and Information Science 2018.

[3] Shaun K. Kane, Meredith Ringel Morris, “Combining Image Recognition and Eye Gaze to Support Conversation for People with ALS”, Conference on Designing Interactive Systems 2018.

[4] Afshin Dehghan, Enrique G. Ortiz, and Guang Shu, “Deep Age, Gender and Emotion Recognition Using Convolutional neural network”, International Conference on Computer and Information Science 2017.

[5] Dong Chen, Xudong Cao, Feng Wen, Jian Sun, “Blessing of Dimensionality: High Dimensional Feature and Its Efficient Compression for Face Verification”, IEEE Conference on Computer Vision and Pattern Recognition 2013.

[6] Luis Antonio Beltrán Prieto, Zuzana Komínková Oplatková, “Comparing the Performance of Emotion-Recognition Implementations in OpenCV, Cognitive Services, and Google Vision API”, WSEAS Transactions on Information Science and Application 2017.

[7] Saeed Turabzadeh, Hongying Meng, Rafiq M. Swash, Matus Pleva, “Facial Expression Emotion Detection for Real-Time Embedded Systems”. MDPI, January 2018.

[8] Rosch Goes, Dr. Suchithra Nair, “Secured access using Cognitive Services on Raspberry Pi”, International Journal of Engineering and Computer Science, May 2017.

[9] Daniel Hubbell, “New Technology Research to Support the Blind and Visually Impaired Community by microsoft cognitive services”, Microsoft Build Conference, April 2016.

[10] Tingting Zhao, Patty Ryan, "Making Sense of Handwritten text in Scanned Documents using Azure cognitive services", Microsoft Build Conference, May 2018.

[11] Julian Brinkley, David Hoffman, Nasseh Tabrizi, "A Social Networking Site Profile System for Blind and Visually Impaired users", IEEE, June 2017.

[12] Arsénio Monteiro Reis, Dennis Paulino, Joao Barroso, "Using Artificial Vision Services to Assist the Blind" , Researchgate, March 2018.

[13] Dirk Sauer, "Secured access using Cognitive Services on Raspberry Pi", November 2017.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "Image classification and conversion of image into text", Cornell University, February 2015.

[15] Dong Chen, Xudong Cao, Feng Wen, Jian Sun. "High Dimensional Feature and its compression for face verification", IEEE Conference on Computer Vision and Pattern Recognition, 2013.

[16] Xuedong Huang, James Baker, Raj Reddy. "A Historical Perspective of Speech Recognition", Association for Computing Machinery, January 2014.

[17] Zhiding Yu, Cha Zhang. "Image based static facial expression recognition using cognitive services", IEEE, November 2015.

[18] Sachin W. Rahate, Dr. M.Z. Shaikh. "Geo fencing Infrastructure location based service", International Research Journal of Engineering and Technology, November 2016.

[19] Prashant Rai, Roma Lakhmani, Yadvendra Singh, Puneet Varshney. "Security Based Alert System Using Geofence", International Journal of Advance Research and Innovative Ideas in Education, May 2018.

[20] Puspa Miladin Nuraida Safitri A Basid, Herman Tolle. "Designing Complaint System on geotagging and geofencing". IEEE, November 2017.