

Pranati Trivedi

Design Project on:

Smart Device for blind people

Abstract:

As we know there are many physically challenged people who cannot see from their own eyes so as we are blessed to have that isn't it our responsibility to help them with our ability. Although there are many other alternatives to deal with their daily routines, they face difficulty in certain navigation for unknown place and also social awkwardness. With this problem there is also an issue of safety. So for that they require a person or sometime no one knows what happened to them so here using different technologies I am trying to solve the problem. Fall needs to be attentively considered due to its highly frequent occurrence especially with old people - up to one third of 65 and above year-old people around the world are risk of being injured due to falling. Furthermore, fall is a direct or indirect factor causing severe traumas such as brain injuries or bone fractures. I am applying Image Processing and object detection represent by their names and then using ttsx3's text to speech API to listen that word. Here object detection will be done using python and open CV. As we know there are more chances of them to fall down due to vision problem so using accelerometer there will be a system through which relative can detect if they are fall down. And if they fall down then their GPS data will be sent to server using GPS tracking and python programming so that near ones/police can be informed. So it will be a really helpful device for the old and blind people.

Description:

This Smart Device for Blind People uses Raspberry 3 B+ as a main processor. Sever is used to send the data to the laptop. It uses the Sense Hat, Adafruit Ultimate GPS and USB webcam for the sensor. The procedure is in two types. First is about Visual recognition and text to speech conversion and second is fall detection with GPS tracking.

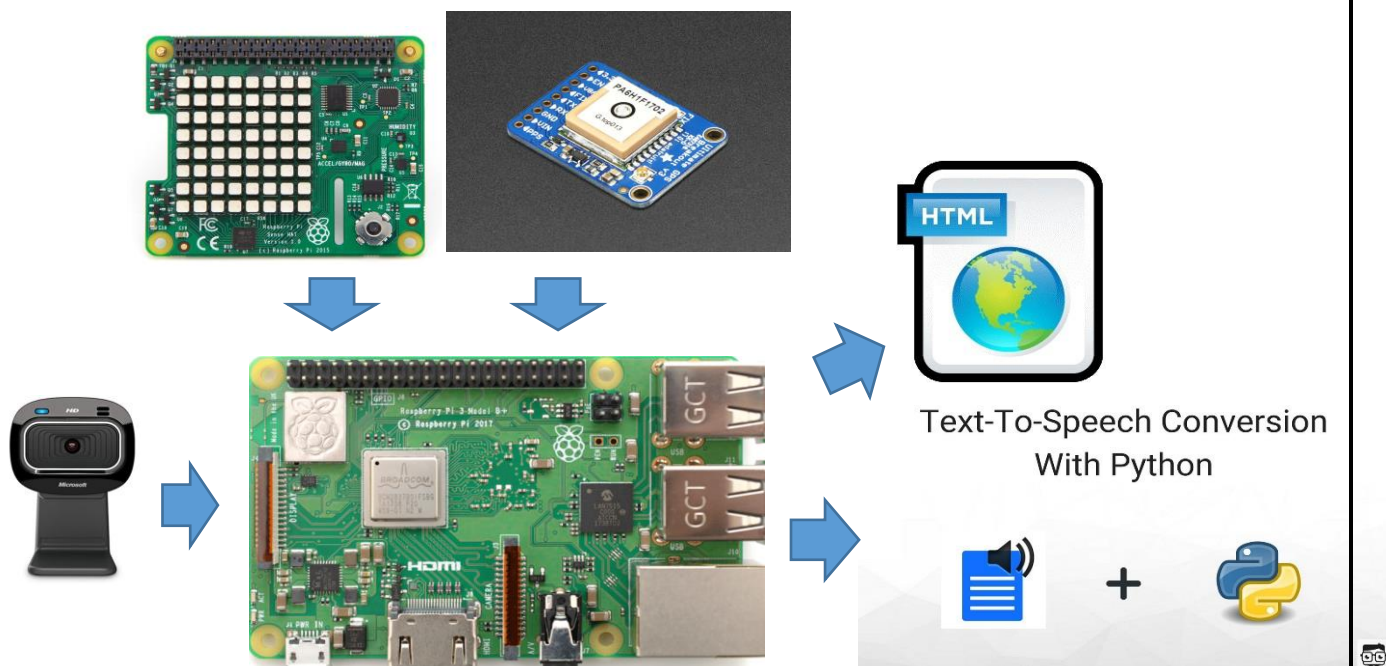
First Part:

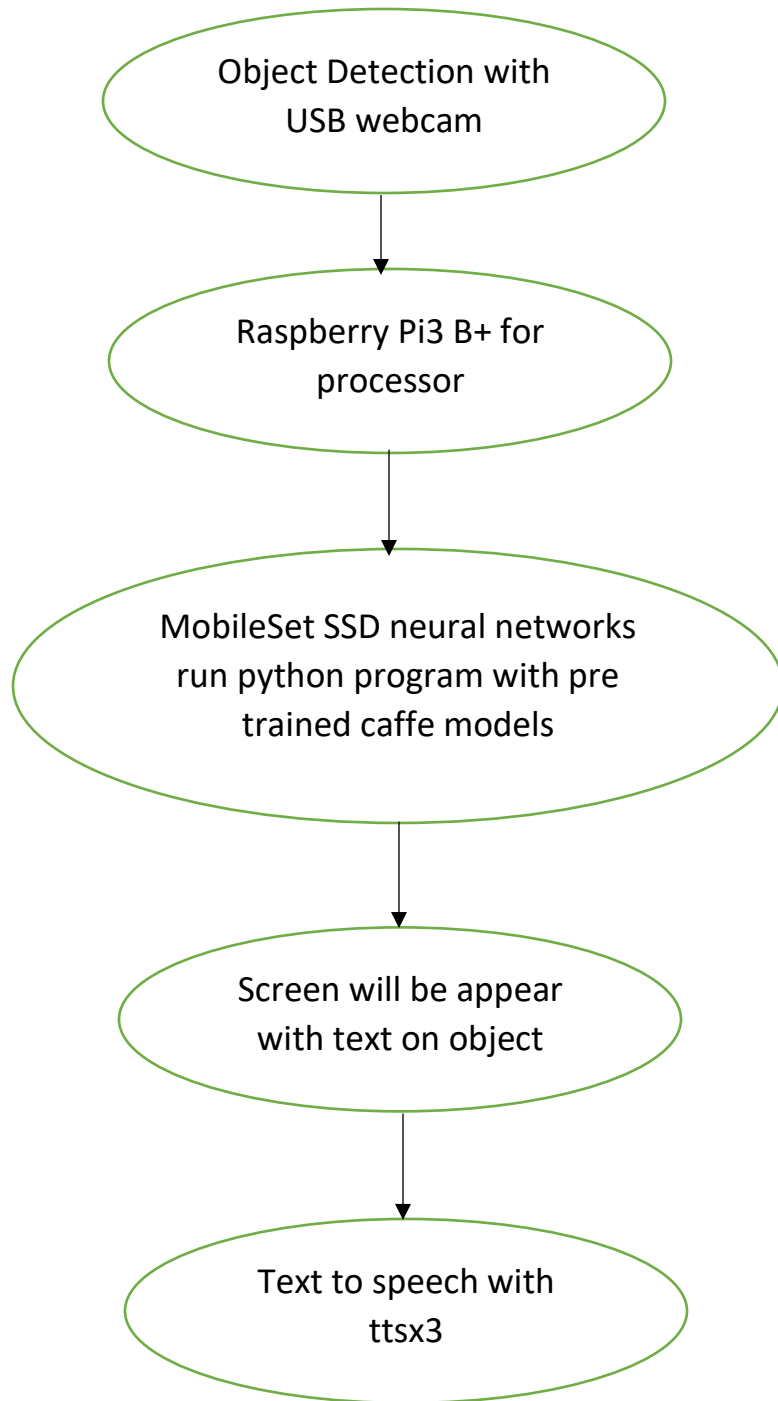
Visual recognition is done using RPi3 B+ and USB webcam using OpenCV libraries and python programming. Even when applying our optimized OpenCV + Raspberry Pi install the Pi is only capable of getting up to ~0.9 frames per second when applying deep learning for object detection with Python and OpenCV. I downloaded and install opencv and opencv-contrib releases for OpenCV 3.3. This will ensure that the deep neural network (dnn) module is installed. You must have OpenCV 3.3 (or newer) to run. The path to the Caffe prototxt file. The path to the pre-trained model. The minimum probability threshold to filter weak detections. The default is 20%.Then initialize CLASS labels and corresponding random COLORS. We set the blob as the input to our neural network and feed the input through the net which gives us our detections. We start by looping over our detections, keeping in mind that multiple objects can be detected in a single image. We also apply a check to the confidence (i.e., probability) associated with each detection. If the confidence is high enough (i.e. above the threshold), then we'll display the prediction in the terminal as well as draw the prediction on the image with text and a colored bounding box. First we display the frame. Then we capture a key press while checking if the 'q' key (for "quit") is pressed, at which point we break out of the frame capture loop. Finally we update our fps counter. Then after printed text I used ttsx3 for text to speech. So I can listen that word on my earphone.

Second Part:

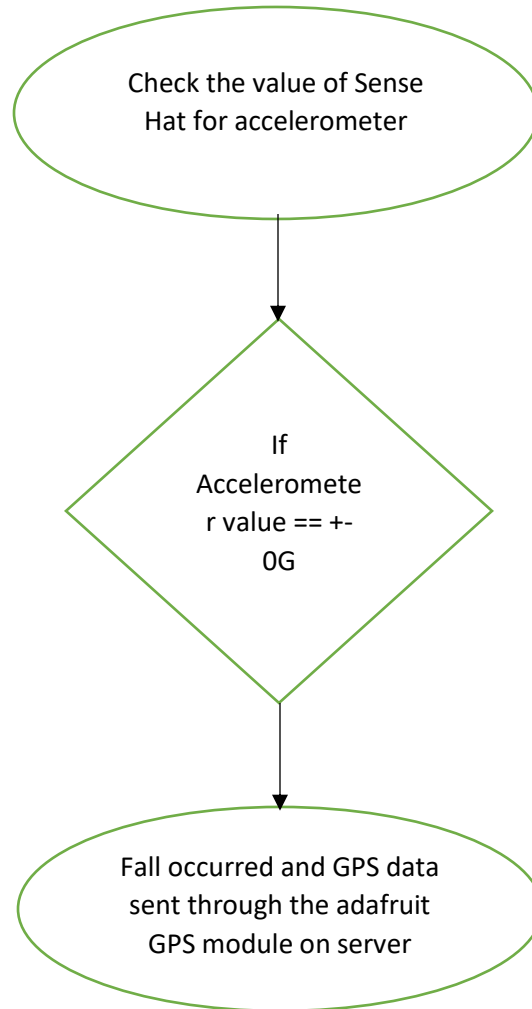
For security reason, I made fall detection for blind people. For that I used Sense Hat and GPS sensor. Sense Hat is a sensor which can measure accelerometer value, gyroscope and other facilities. For fall detection as we know that fall observed when there is a zero gravitational force. So I observed that before the fall, the Sense Hat accelerometer measures approximately $\pm 1G$, then falls/rises to 0 while it is falling because we know falling object will experience zero G. And as fall detected, location of that person will be sent to the server.

Flow Chart:





Flow chart of fall detection



And the server system work as shown in flow chart, the fall detected data and GPS location data is captured and stored in the database. This data is sent to the server using socket method. For the server part, I tried to use IBM Watson, Azura, ThingSpeak ,Apache2, http request. But due to time constraint, I was not able to successfully implement any fancy method. I concluded in using socket method for transmission on server. The server stores this information locally. Also, a simple html website is implemented by me which works on localhost to display the authenticated data of the user.

Hardware:

1. Raspberry Pi3 B+:

1.4GHz 64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and Power-over-Ethernet support (with separate PoE HAT).

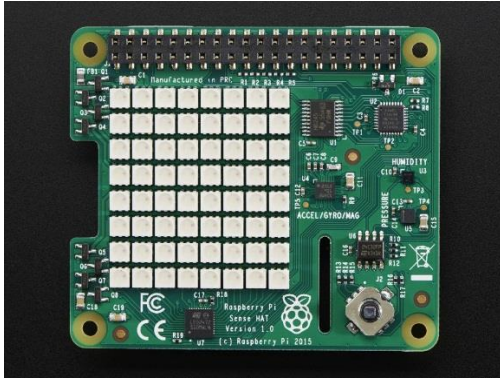
Specification

- SoC: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- GPU: Broadcom Videocore-IV
- RAM: 1GB LPDDR2 SDRAM
- Networking: Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
- Bluetooth: Bluetooth 4.2, Bluetooth Low Energy (BLE)
- Storage: Micro-SD
- GPIO: 40-pin GPIO header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- Dimensions: 82mm x 56mm x 19.5mm, 50g

2. Sense Hat

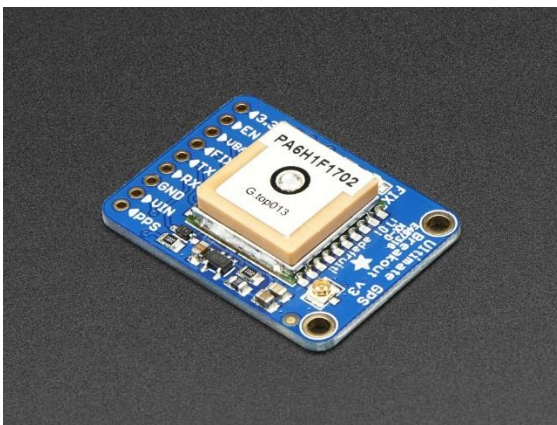
The Raspberry Pi Sense HAT is attached on top of the Raspberry Pi via the 40 GPIO pins to create an 'Astro Pi'. The sensors enable you to read:

- Orientation (yaw, pitch & roll) via an accelerometer, 3D gyroscope, and magnetometer
- Pressure
- Humidity
- Temperature



3. Adafruit Ultimate GPS:

- -165 dBm sensitivity, 10 Hz updates, 66 channels
- 5V friendly design and only 20mA current draw
- Breadboard friendly + two mounting holes
- RTC battery-compatible
- Built-in datalogging
- PPS output on fix
- Internal patch antenna + u.FL connector for external active antenna
- Fix status LED
- The breakout is built around the MTK3339 chipset, a no-nonsense, high-quality GPS module that can track up to 22 satellites on 66 channels, has an excellent high-sensitivity receiver (-165 dBm tracking!), and a built in antenna. It can do up to 10 location.



4. USB Webcam

It supports 720p with 1280x720 resolution and 30 frames per second. It has an in-built microphone. It works on USB interface connection.



Link for datasheet of Adafruit ultimate GPS:

<https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf>

Link for datasheet of Sense Hat:

<https://www.farnell.com/datasheets/1958037.pdf>

Link for datasheet of USB Webcam:

http://download.microsoft.com/download/0/9/5/0952776D-7A26-40E1-80C4-76D73FC729DF/TDS_LifeCamHD-3000.pdf

Software:

Raspbian OS:

It is the most suitable operating system for the raspberry pi and convenient for interfacing sensors.

Apache Server – to host the website on the localhost domain using a server.

VNC Viewer – to view the graphical format/desktop of the raspberry pi.

Programming languages involved are-

- Python – the coding for most of the part like processing sensor data and sending data to the server is done using python. Python was used as it has many in-built libraries and also the coding is easy and in a natural type of language for the pi.
- Html – it was used to write the html webpage for the web server.
- Shell Scripting – basic shell scripting was used on the pi to create directories and check, change USB modes and interfaces.
- Php/Java – it was used for hosting the website on the webserver and also to execute html tags within the php script. This was even used to run python script from the web server itself.

The system completely works real time. As it gets the data from the sensor and USB webcam and then real time spoken words on earphone and the real data of fall detection and GPS location on the server without delay. I used OpenCV libraries for the object detection and MobileNet SSD pre trained caffe models and for text to speech I used tts3 python library.

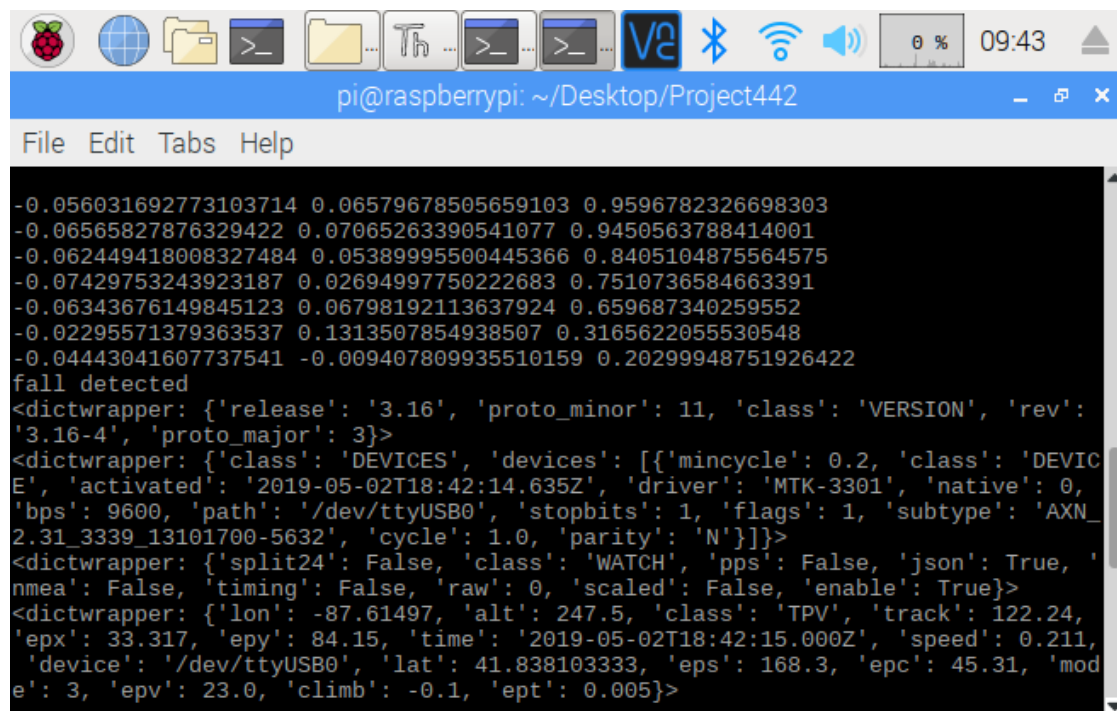
The biggest security issue is the transmission of the data on the sever. The data is sent locally on to the server and it does not have any strong password protection. So, any device that is functioning in the same network might get access to the pi's data.

The code developed in this project is mostly written by me. The object detection code which requires to call openCV and some prerained models and classes of objects so it was hard for me to write from so I took already available code. While text to speech and print text on monitor was done by me. While the fall detection logic and store data to the file of GPS location was done by me but to get raw data of sensors I took help from already available code. The code on the server part involves socket communication is done by me which includes html programming.

Related Work:

I did not find same idea as mine so some changes in code can be made to make it more professional. As of now, this idea is unique. But still there is many changes to made. I have seen object detection which was done separately but not the same what I did the whole.

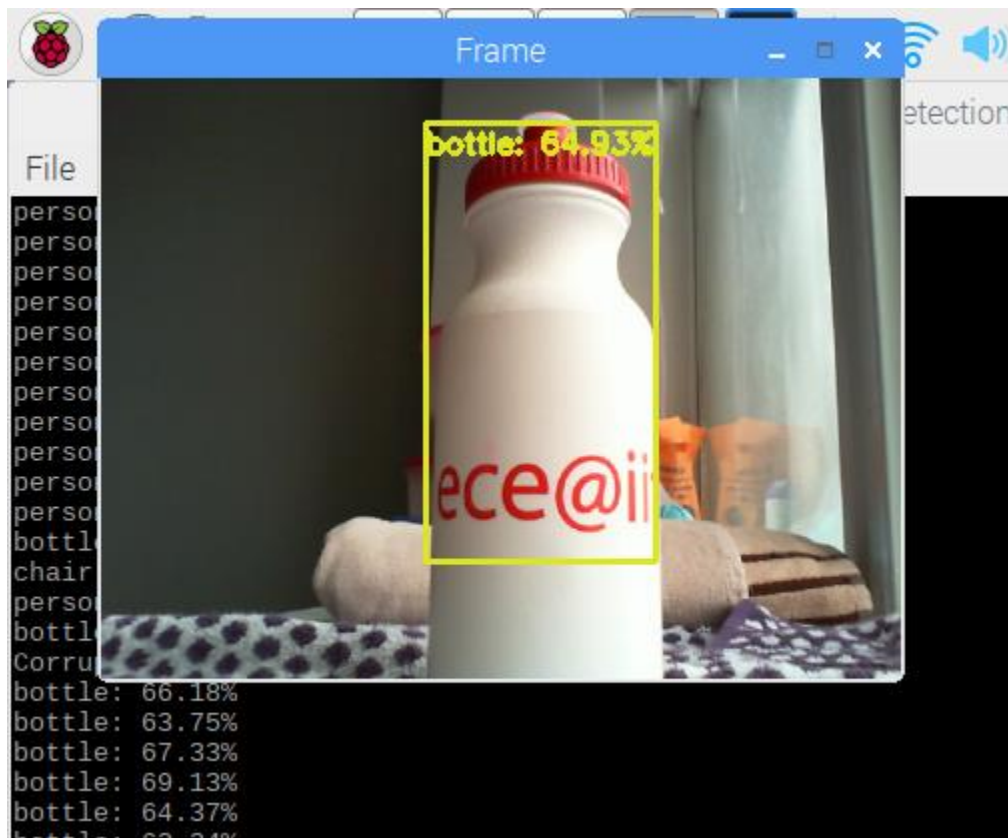
Results and Discussion:



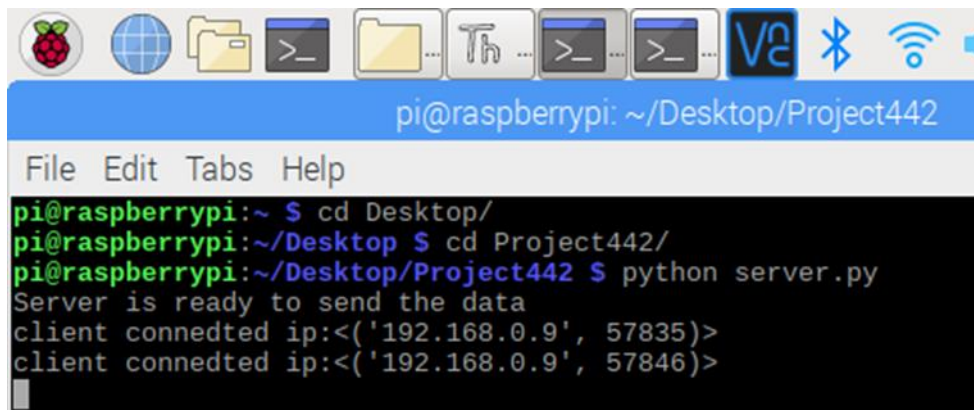
The screenshot shows a terminal window titled 'pi@raspberrypi: ~/Desktop/Project442'. The terminal displays several lines of numerical data, followed by the text 'fall detected'. Below this, there are three JSON objects wrapped in <dictwrapper> tags. The first JSON object contains 'release', 'proto_minor', 'class', and 'rev'. The second JSON object contains 'class', 'devices' (an array of device information), and 'activated'. The third JSON object contains 'lon', 'alt', 'class', 'track', 'epx', 'epy', 'time', 'speed', 'device', 'lat', 'eps', 'epc', 'mode', 'epv', 'climb', and 'ept'.

```
-0.056031692773103714 0.06579678505659103 0.9596782326698303
-0.06565827876329422 0.07065263390541077 0.9450563788414001
-0.062449418008327484 0.05389995500445366 0.8405104875564575
-0.07429753243923187 0.02694997750222683 0.7510736584663391
-0.06343676149845123 0.06798192113637924 0.659687340259552
-0.02295571379363537 0.1313507854938507 0.3165622055530548
-0.04443041607737541 -0.009407809935510159 0.20299948751926422
fall detected
<dictwrapper: {'release': '3.16', 'proto_minor': 11, 'class': 'VERSION', 'rev':
'3.16-4', 'proto_major': 3}>
<dictwrapper: {'class': 'DEVICES', 'devices': [{'mincycle': 0.2, 'class': 'DEVIC
E', 'activated': '2019-05-02T18:42:14.635Z', 'driver': 'MTK-3301', 'native': 0,
'bps': 9600, 'path': '/dev/ttyUSB0', 'stopbits': 1, 'flags': 1, 'subtype': 'AXN_
2.31_3339_13101700-5632', 'cycle': 1.0, 'parity': 'N'}]}>
<dictwrapper: {'split24': False, 'class': 'WATCH', 'pps': False, 'json': True, '
nmea': False, 'timing': False, 'raw': 0, 'scaled': False, 'enable': True}>
<dictwrapper: {'lon': -87.61497, 'alt': 247.5, 'class': 'TPV', 'track': 122.24,
'epx': 33.317, 'epy': 84.15, 'time': '2019-05-02T18:42:15.000Z', 'speed': 0.211,
'device': '/dev/ttyUSB0', 'lat': 41.838103333, 'eps': 168.3, 'epc': 45.31, 'mod
e': 3, 'epv': 23.0, 'climb': -0.1, 'ept': 0.005}>
```

Value of Fall detection and Sense Hat with GPS location



Result of Object detection with precision



Server part to send the data of GPS location and fall detection

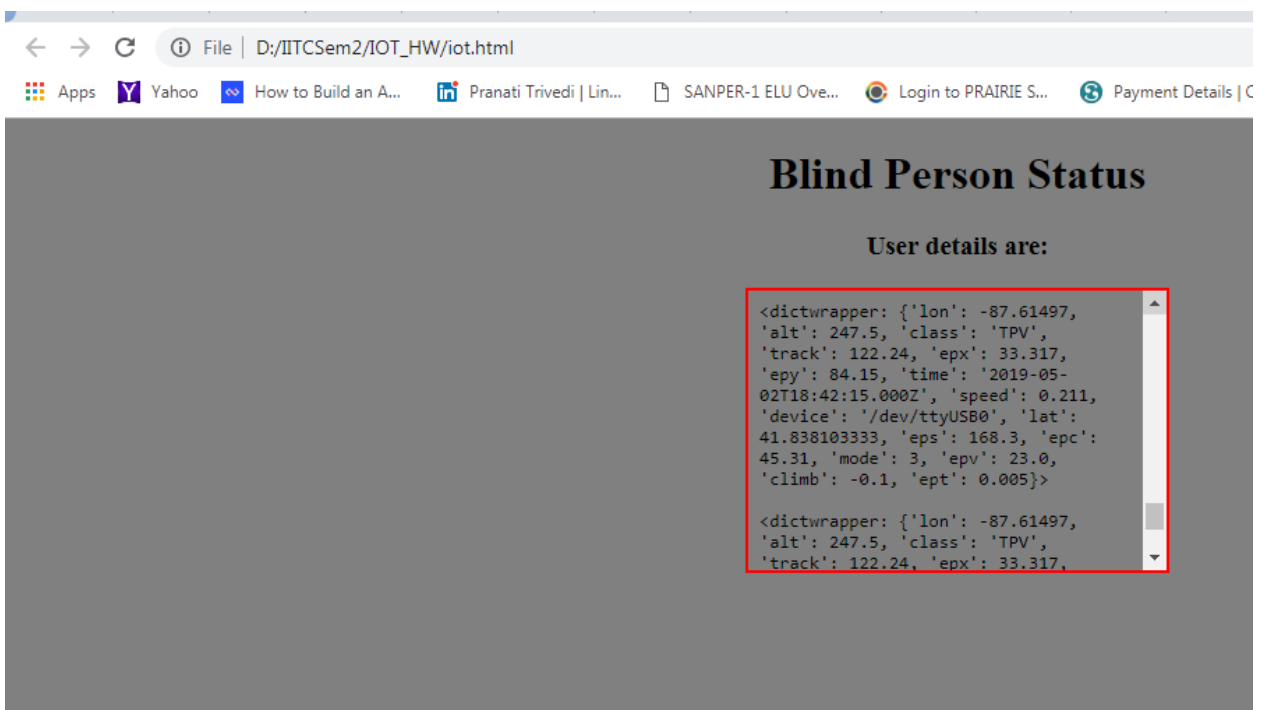
```
C:\Users\admin>cd Documents

C:\Users\admin\Documents>python client_text.py
95.70% Done
100.00% Done
Data is received to see

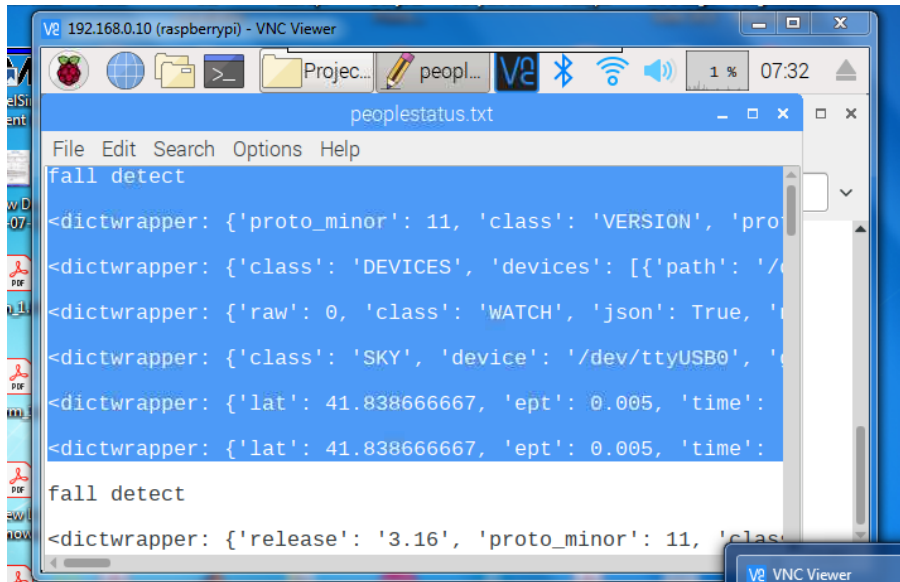
C:\Users\admin\Documents>python client_text.py
62.06% Done
93.09% Done
100.00% Done
Data is received to see

C:\Users\admin\Documents>_
```

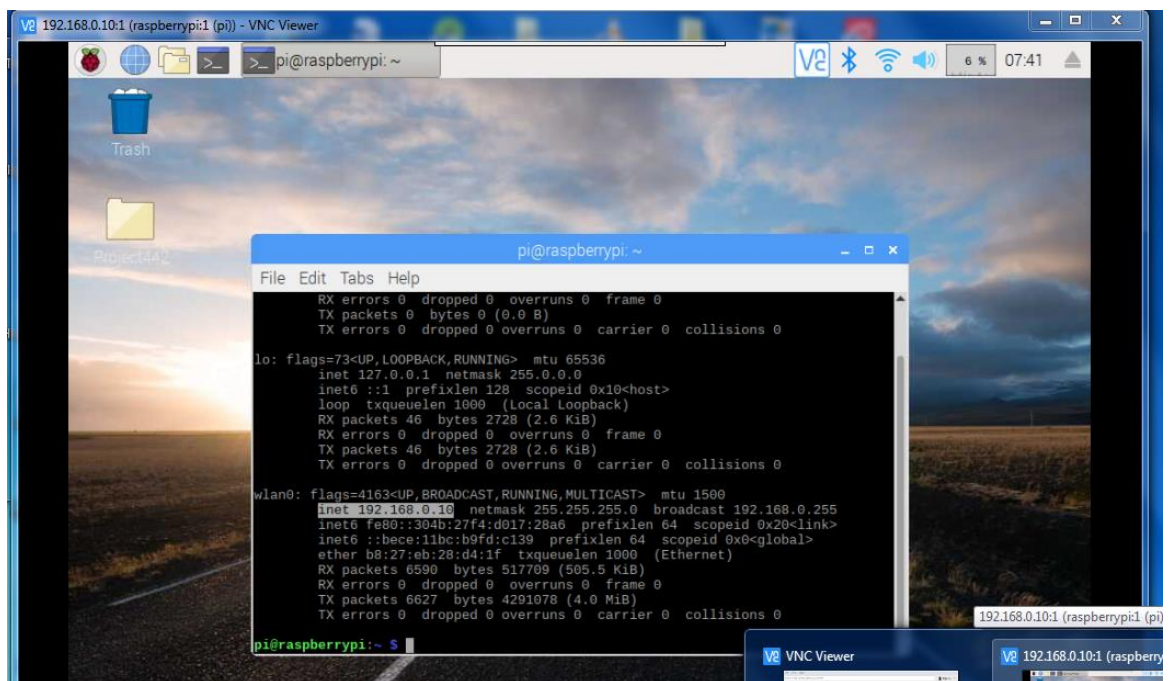
Client part to fetch the data from the server



Local Web Page result



Stored data at peoplestatus.txt file



Get IP address of the PI

Server is created using this IP address of the Pi

Discussion:

The GPS module and OpenCV was completely new for me so it took so much time to understand and make it work according to my application.

Server part, I tried many IoT platforms like IBM Watson, Microsoft Azura, ThingSpeak and Apache Spark. But due to IP address and server connection for these platform using node-red. My node-red environment crashed so unfortunately, I could not make it fancy and more user friendly.

I displayed with local server and normal html webpage which I have attached screenshot in this report. I could do better and also do live video streaming with different Rpi or using android application and make it more feasible for security. Like every application could work together with text to speech and fall detection could be better.

To get all the codes together with the logic and in client server data to store data with text file. And GPS module to make it fix in the sky and to understand its parameters was difficult task for me to learn and make it work.

Conclusion:

This project can successfully detect the object and can listen in the earphone with safety feature of fall detection for blind people. It can be improved by good indoor GPS module and with good object detection recognition with good frames per second. And also by good user interface and using compact devices. Although it is a very good for blind people to detect object and can listen. And we can see the location in the local web page of fall detection. And we can put that with dear ones so they could have update of the blind person.

List of References:

[1] <https://www.pyimagesearch.com/2018/09/19/pip-install-opencv/>

This is a link where I got the guidance to install image processing. How can I do and which libraries and dependences are needed.

[2] <https://www.pyimagesearch.com/2017/10/09/optimizing-opencv-on-the-raspberry-pi/>

This is also the same installation process with more detailed explanation and which libraries to install and how can I make it more space for OpenCV in Rpi

[3] <https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/>

Here I could get guidance to make object detection and where to call pre trained models for object detection. How can make box around object in image put text on it etc.

[4] <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>

To understand installation process of RPI 3 and its configuration.

[5] <https://www.farnell.com/datasheets/1958037.pdf>

This is the link where I got the information about Sense Hat with pin diagram and specification. Interface SenseHat and get the data for accelerometer for fall detection.

[6] <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf>

This is the link for GPS module of detailed information with configuration, installation and connection. Interface GPS module with Rpi and make it fix to get data from satellites.

[7] <https://developer.ibm.com/recipes/tutorials/raspberry-pi-4/>

For IBM Watson to work as a dashboard

[8] <https://nodered.org/docs/hardware/raspberrypi>

To work and installation of node-red

[9] <https://websockets.readthedocs.io/en/stable/intro.html>

Web socket programming for client server part

[10] <https://www.devdungeon.com/content/text-speech-python-pyttsx3>

This is a link for text to speech conversion in python which I used and took guidance for my project

Future Work:

Good GPS module which can work indoor or outdoor or use Google maps API and I can use NodeJS or JavaScript Language. Good processor to increase the rate FPS and if I can do R language to train models. To make my own predefined model and make object detection more precise and can add more objects. To make dynamic server or make mobile App for more user friendly.

Appendix:

Code for object detection with text to speech :

```
# USAGE
# python real_time_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt --
model MobileNetSSD_deploy.caffemodel

# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import cv2
import os
import pyttsx3

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()

# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
            "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable",
            "dog", "horse", "motorbike", "person", "pottedplant",
            "sheep",
            "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and
    # resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
```



```

frame = imutils.resize(frame, width=400)

# grab the frame dimensions and convert it to a blob
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300,
300)),
                                0.007843, (300, 300), 127.5)

# pass the blob through the network and obtain the
detections and

# predictions
net.setInput(blob)
detections = net.forward()

# loop over the detections
for i in np.arange(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated
    with

    # the prediction
    confidence = detections[0, 0, i, 2]

    # filter out weak detections by ensuring the
    `confidence` is

    # greater than the minimum confidence
    if confidence > args["confidence"]:
        # extract the index of the class label from
        the

        # draw the prediction on the frame
        label = "{}:
{: .2f}%".format(CLASSES[idx],
                    confidence * 100)
        cv2.rectangle(frame, (startX, startY), (endX,
endY),
                    COLORS[idx], 2)
        y = startY - 15 if startY - 15 > 15 else startY
        + 15
        cv2.putText(frame, label, (startX, y),
                    cv2.FONT_HERSHEY_SIMPLEX,
0.5, COLORS[idx], 2)

        print(label)
        #os.system("espeak label")
        engine = pyttsx3.init()

```



```

engine.say(label)
engine.runAndWait()
# show the output frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop
if key == ord("q"):
    break

# update the FPS counter
fps.update()

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()

```

Fall detection code :

```

from sense_hat import SenseHat
from datetime import datetime
import sys
from time import sleep
from evdev import InputDevice, ecodes, list_devices
from select import select
import logging
import gps
count=0
# Listen on port 2947 (gpsd) of localhost
session = gps.gps("localhost", "2947")
session.stream(gps.WATCH_ENABLE | gps.WATCH_NEWSTYLE)

# Use the logging library to handle all our data recording
logfile = "gravity-"+str(datetime.now().strftime("%Y%m%d-%H%M"))+".csv"
# Set the format for the timestamp to be used in the log filename
logging.basicConfig(filename=logfile, level=logging.DEBUG,
    format='%(asctime)s %(message)s')

def gentle_close(): # A function to end the program gracefully
    sh.clear(255,0,0) # Turn on the LEDs red

```

```

        sleep(0.5) # Wait half a second
        sh.clear(0,0,0) # Turn all the LEDs off
        sys.exit() # Quit the program

sh = SenseHat() # Connect to SenseHAT
sh.clear() # Turn all the LEDs off
# Find all the input devices connect to the Pi
devices=[InputDevice(fn) for fn in list_devices()]
for dev in devices:
    # Look for the SenseHAT Joystick
    if dev.name=="Raspberry Pi Sense HAT Joystick":
        js=dev
# Create a variable to store whether or not we're logging data
running = True # No data being recorded (yet)
try:
    print('Press the Joystick button to start recording.')
    print('Press it again to stop.')
    while True:
        # capture all the relevant events from joystick
        #r,w,x=select([js.fd],[],[],0.01)
        #for fd in r:
            #    for event in js.read():
                # If the event is a key press...
                #    if event.type==ecodes.EV_KEY and event.value==1:
                    # If we're not logging data, start now
                    #if event.code==ecodes.KEY_ENTER and not running:
                        #running = True # Start recording data
                        #sh.clear(0,255,0) # Light up LEDs green
                        # If we were already logging data, stop now
                        #elif event.code==ecodes.KEY_ENTER and running:
                            #running = False
                            #gentle_close()
                # If we're logging data...
                if running:
                    # Read from accelerometer
                    acc_x,acc_y,acc_z = [sh.get_accelerometer_raw()[key] for
key in ['x','y','z']]
                    # Format the results and log to file
                    logging.info('{:12.10f}, {:12.10f},
{:12.10f}'.format(acc_x,acc_y,acc_z))
                    print(acc_x,acc_y,acc_z) # Also write to screen
                    if acc_x<0.05 and acc_x>-0.05 and acc_y<0.05 and acc_y>-
0.05 and acc_z<0.3 and acc_z>-0.0:
                        print('fall detected')
                        #datafile=acc_x,acc_y,acc_z,report
                        filee=str('fall detect')

f=open('/home/pi/Desktop/Project442/peoplestatus.txt','a')
    f.write(filee+'\n\n')
    count = count + 1

    while count==1:

```

```

        try:
            report = session.next()
            # Wait for a 'TPV' report and display the
current time
            # To see all report data, uncomment the
line below
            print(report)
            fileeee=str(report)

f=open('/home/pi/Desktop/Project442/peoplestatus.txt','a')
            f.write(fileeee+'\n\n')

            if report['class'] == 'TPV':
                if hasattr(report, 'time'):
                    print(report.time)
                    count = count + 1
#                    datafile=acc_x,acc_y,acc_z,report
#                    filee=str(datafile)
#                    f=open('/home/pi/data.txt','a')
#                    f.write(filee+'\n\n')

        #gentle_close()
            if count==2:
                #quit()

#datafile=acc_x,acc_y,acc_z,report
            fileeeee=str(report)

f=open('/home/pi/Desktop/Project442/peoplestatus.txt','a')
            f.write(fileeeee+'\n\n')

            count = count + 1
            if count==3:
                quit()

        except KeyError:
            pass
        except KeyboardInterrupt:
            quit()
        except StopIteration:
            session = None

        #print("GPSD has terminated")

        #datafile=acc_x,acc_y,acc_z,report
        #filee=str(datafile)
        #f=open('/home/pi/data.txt','a')
        #f.write(filee+'\n\n')

        #filee=str('fall detected',report)
        #f=open('/home/pi/data.txt','a')
        #f.write(filee+'\n\n')

```

```
except: # If something goes wrong, quit
    gentle_close()
```

Server code to send data:

```
import socket
import threading
import os

def RetrFile(name, sock):
    filename = sock.recv(1024)
    if os.path.isfile(filename):
        sock.send("EXISTS " + str(os.path.getsize(filename)))
        userResponse = sock.recv(1024)
        if userResponse[:2] == 'OK':
            with open(filename, 'rb') as f:
                bytesToSend = f.read(1024)
                sock.send(bytesToSend)
                while bytesToSend != "":
                    bytesToSend = f.read(1024)
                    sock.send(bytesToSend)
        else:
            sock.send("ERR ")

    sock.close()

def Main():
    host = '192.168.43.21'
    port = 5000

    s = socket.socket()
    s.bind((host,port))

    s.listen(5)

    print("Server is ready to send the data")
    while True:
        c, addr = s.accept()
        print("client conncted ip:<" + str(addr) + ">")
        t = threading.Thread(target=RetrFile, args=("RetrThread", c))
        t.start()

    s.close()

if __name__ == '__main__':
    Main()
```

Client code to receive data:

```
import socket                                     # import socket for server client
communication
import re

def main():
    host = '192.168.43.21'                        # defines the IP address of the
    pi on which the server is set up
    port = 5000                                    # defines port that is used
    for communication

    s = socket.socket()                            # open socket for
    communication
    s.connect((host, port))

    filename = "peoplestatus.txt"                 # defines file that is to
    be received
    if filename != 'q':                             # requests server to check
    for the given filename
        s.sendall(filename)
        data = s.recv(1024)
        if data[:6] == 'EXISTS':
            filesize = long(data[6:])              # creates a long type
            variable to store the incoming file
            message = "y"
            if message == 'y':
                s.send("OK")
                f = open('new_'+filename, 'wb')    #store the new file
                with the preceeding word 'new_'. w stands for write parameter.
                data = s.recv(1024)
                totalRecv = len(data)
                f.write(data)

                while totalRecv < filesize:
                    data = s.recv(1024)
                    totalRecv += len(data)
                    f.write(data)                  # write the data

    to the new file

    print("{0:.2f}".format((totalRecv/float(filesize))*100)+ "% Done")
    # print the status of the file received

    print("Data is received to see")

    f.close()
    #print(data)

    s.close()                                     # close socket

if __name__ == '__main__':
```

```
main()
```

HTML Code to make webpage:

```
<!DOCTYPE html>
<html>
<head>
<title>Blind Person Status</title>
</head>
<body bgcolor="Gray">
<center>
<h1>Blind Person Status</h1>
</center>
<center>
<h3> User details are: <p><iframe
src="C:\Users\admin\Documents\new_peoplestatus.txt"
style="height:200px;width:300px;border:2px solid
red"></iframe></p></h3>
</center>
</body>
</html>
```

Cost of the Project:

Raspberry Pi	35\$
USB Webcam	20\$
Sense Hat	39\$
Adafruit Ultimate GPS	39\$
USB to serial converter	8\$
16 GB Memory card	9\$
Cables, wires	10\$
Total	160\$

Power Usage:

5V DC power is used for the whole project.