### **OBJECT DETECTION FOR**

### **VISUALLY IMAPAIRED PERSON**

### **USING RASPBERRY PI**

### **CODE:**

### from flask import Flask, Response, render\_template

### import cv2

### import numpy as np

### app = Flask(\_name\_)

### # Load MobileNet SSD model

### prototxt\_path = "models/MobileNetSSD\_deploy.prototxt.txt"

### model\_path = "models/MobileNetSSD\_deploy.caffemodel"

### net = cv2.dnn.readNetFromCaffe(prototxt\_path, model\_path)

### # Define class labels

### CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",

### "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",

### "dog", "horse", "motorbike", "person", "pottedplant",

### "sheep", "sofa", "train", "tvmonitor"]

### # Initialize Video Capture

### cap = cv2.VideoCapture(0)

### def generate\_frames(detect=False):

### while True:

### success, frame = cap.read()

### if not success:

### break

### else:

### if detect:

### # Prepare frame for detection

### (h, w) = frame.shape[:2]

### blob = cv2.dnn.blobFromImage(frame, 0.007843, (300, 300), 127.5)

### net.setInput(blob)

### detections = net.forward()

### for i in range(detections.shape[2]):

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

### if confidence > 0.5: # Detection threshold

### idx = int(detections[0, 0, i, 1])

### box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

### (startX, startY, endX, endY) = box.astype("int")

### label = f"{CLASSES[idx]}: {confidence\*100:.2f}%"

### cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 255, 0), 2)

### cv2.putText(frame, label, (startX, startY - 10),

### cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

### # Encode frame as JPEG

### \_, buffer = cv2.imencode('.jpg', frame)

### frame = buffer.tobytes()

### 

### # Yield frame in multipart format

### yield (b'--frame\r\n'

### b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

### @app.route('/')

### def index():

### """ Main page with links to video streams """

### return render\_template('index.html')

### @app.route('/video\_feed')

### def video\_feed():

### """ Normal video streaming route """

### return Response(generate\_frames(detect=False), mimetype='multipart/x-mixed-replace; boundary=frame')

### @app.route('/detect\_objects')

### def detect\_objects():

### """ Object detection video streaming route """

### return Response(generate\_frames(detect=True), mimetype='multipart/x-mixed-replace; boundary=frame')

### if \_name\_ == "\_main\_":

### app.run(host="0.0.0.0", port=5000, debug=False)

import cv2

cap = cv2.VideoCapture(0)

if not cap.isOpened():

print("Cannot open camera")

exit()

while True:

ret, frame = cap.read()

if not ret:

print("Can't receive frame. Exiting...")

break

cv2.imshow("Frame", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

import RPi.GPIO as GPIO

import time

class UltrasonicHandler:

def \_init\_(self, trigger\_pin, echo\_pin):

self.trigger\_pin = trigger\_pin

self.echo\_pin = echo\_pin

GPIO.setmode(GPIO.BCM)

GPIO.setup(self.trigger\_pin, GPIO.OUT)

GPIO.setup(self.echo\_pin, GPIO.IN)

GPIO.output(self.trigger\_pin, False)

time.sleep(2) # Allow sensor to settle

def get\_distance(self):

# Send trigger pulse

GPIO.output(self.trigger\_pin, True)

time.sleep(0.00001)

GPIO.output(self.trigger\_pin, False)

# Wait for echo response

pulse\_start, pulse\_end = 0, 0

while GPIO.input(self.echo\_pin) == 0:

pulse\_start = time.time()

while GPIO.input(self.echo\_pin) == 1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = (pulse\_duration \* 34300) / 2 # Convert time to distance in cm

return round(distance, 2)

def cleanup(self):

GPIO.cleanup()

# To import this module in another file:

# from ultrasonic\_handler import UltrasonicHandler

# sensor = UltrasonicHandler(trigger\_pin=23, echo\_pin=24)

# distance = sensor.get\_distance()

# print(f"Measured Distance: {distance} cm")

# sensor.cleanup()

# Example usage

if \_name\_ == "\_main\_":

sensor = UltrasonicHandler(trigger\_pin=20, echo\_pin=21)

try:

while True:

distance = sensor.get\_distance()

print(f"Distance: {distance} cm")

time.sleep(1)

except KeyboardInterrupt:

sensor.cleanup()