

Assistance for Visually Challenged Individuals

Soham Gupta , Kaustav Sharma

Guided by: Maddhuja Sen (Core Member of ML domain in KIIT Robotics Society)

School of Computer Science, Kalinga Institute of Industrial Technology,

Email of students : 20051039@kiit.ac.in, 2005731@kiit.ac.in

Supervisor Email ID : 2028021@kiit.ac.in

Abstract- Blind individuals face numerous challenges in their daily life, particularly when navigating their environment, which often involves busy roads and various obstacles. This problem is a critical concern that affects their safety, independence, and quality of life. To tackle this challenge, we have proposed an innovative solution that uses computer vision and artificial intelligence. The proposed solution works by using a camera that captures a live video stream of the environment and feeds it to a deep learning model. The model uses computer vision algorithms to identify and locate objects in the video feed, including obstacles, other people, and any other potential hazards. The algorithm then determines the direction and distance of these objects relative to the camera and translates the data into speech, providing real-time audio feedback to the blind individual. With this real-time audio information, the blind person can navigate their surroundings with confidence, knowing exactly what obstacles are in their path and how to avoid them. The technology has the potential to improve their quality of life and independence, and make their daily activities easier and safer. This project represents a significant step forward in the development of technologies that can help blind individuals to live more independent and fulfilling lives. By leveraging the power of deep learning and computer vision, it has the potential to transform the way that blind individuals experience and interact with their surroundings, providing them with a new level of freedom and independence.

Index Terms- Computer Vision, YOLO, object detection, blind people, deep learning

I. Introduction

This project aims to deliver a product to help those with vision impairment. Vision impairment persists in all walks of the society and there is not much done in order to help them with their day to day needs; hence, with the help of machine learning , we have trained a model that help a person with vision impairment dodge obstacles while moving.

We have implemented this model with the help of computer vision, text to speech, distance of an object as well as it's direction from the camera that the person will carry to scan it's surroundings [7]. This project is at it's starting phase and is just a prototype of what it is capable of doing. Further implementation of hardware components are required in order to fully benefit from it.

In this prototype, we have simply used the web cam of our computer (laptop) to view the surroundings and detect objects, further hardware enhancements will take place in the next stage of this project.

II. Study of similar projects or technology\ literature review

There is a vast body of research on object detection using OpenCV, and many different approaches have been proposed and evaluated. Some of the most popular object detection algorithms in use today include:

Haar cascades: Haar cascades are a simple and efficient way to detect objects in images, and they have been widely used for face detection and other applications [8].

HOG (Histograms of Oriented Gradients): HOG is a feature extraction method that has been used in many object detection algorithms, including the popular Dlib library.

YOLO (You Only Look Once): YOLO is a fast and accurate object detection system that has been used for a wide range of applications, including self-driving cars, surveillance, and robotics [1].

Faster R-CNN: Faster R-CNN is a two-stage object detection algorithm that is widely used for its accuracy and speed.

There are many other object detection algorithms that have been proposed and evaluated in the literature, and the best choice will depend on your specific needs and the constraints of your application.

For this project, we have chosen YOLO version 4 to detect objects.

III. Basic concepts/ Technology used

YOLO V4 (You Only Look Once version 4): YOLO (You Only Look Once) is an object detection algorithm that has been developed through several versions, including YOLO v4. YOLO v4 is the latest version of the YOLO algorithm and is known for its high accuracy and speed compared to other object detection algorithms [2]. It uses a single neural network to predict bounding boxes and class probabilities for objects in an image, instead of using multiple stages like in other algorithms. YOLO v4 also uses a variety of techniques to improve its accuracy, including improvements in the network architecture, data augmentation, and object scale normalization.

Architecture of YOLO V4:

YOLO v4 architecture is based on a hybrid approach, combining features from both the ResNet and Darknet architectures. The architecture of YOLO v4 consists of three main components: a neck module, a head module, and a detector module [3].

1. Neck Module: This module is responsible for feature extraction and is based on ResNet architecture. The neck module uses residual connections and shortcuts to help the network learn more effectively.
2. Head Module: This module takes the features from the neck module and predicts the object detections. The head module consists of several Convolutional Layers and a series of upsampling and concatenation layers.
3. Detector Module: This module is responsible for generating the final detections from the predictions of the head module. The detector module uses anchor boxes and non-maximal suppression to produce the final detections.

ResNet architecture used in the neck module : ResNet (Residual Network) is a deep neural network architecture that was introduced in 2015. The key idea behind ResNet is the use of residual connections, which allow the network to bypass some of the layers and learn more effectively. The residual connections help the network to avoid the vanishing gradient problem, which can occur in very deep networks, and make it easier for the network to learn.

IV. PROJECT COMPONENTS

1. Jupyter Notebook [Editor]
2. Tkinter
3. Set of Images and their classes (tags)

V. Proposed Model / Architecture / Methodology / Model Tool

In this innovative project, we have combined the power of computer vision and artificial intelligence to create an object detection algorithm that can assist blind individuals in navigating their surroundings [4]. The algorithm is implemented using OpenCV, a powerful open-source computer vision library, and Tkinter, a Python library for building interactive User interface applications [9].

The system works by capturing a live video stream of the environment and feeding it to a deep learning model. This model uses computer vision algorithms to identify and locate objects in the video feed, including obstacles, other people, and any other potential hazards. The algorithm then determines the direction and distance of these objects relative to the camera, allowing it to provide the blind individual with detailed information about their surroundings.

To make this information accessible to the blind individual, the algorithm translates the data into speech, providing real-time audio feedback about the environment [5]. This enables the individual to move with confidence, knowing exactly what obstacles are in their path and how to avoid them.

A model was trained to identify those set of objects with images. The algorithm used to train the model is YOLO V4.

```
! pip install opencv-contrib-python==4.5.3.56

! pip install pyttsx3

import cv2 as cv # computer vision library for image processing
import pyttsx3 # for text to speech conversion
import numpy as np
```

Importing the necessary libraries for object detection and text to speech translation

```
n_rows = 1

n_images_per_row = 3

cap = cv.VideoCapture(0)
ret, frame = cap.read()

height,width,ch = frame.shape

roi_height = height // n_rows

roi_width = width // n_images_per_row

print("Width =", width)

print("Width/3 =", roi_width)
```

This code captures video from the default camera (camera index 0) using OpenCV's VideoCapture class and stores the captured frame in the "frame" variable. It then calculates the height and width of the frame

using the "shape" attribute and stores them in the "height" and "width" variables, respectively. The number of channels (ch) in the frame is also extracted and stored in the "ch" variable. The code then calculates the height and width of the region of interest (ROI) by dividing the height and width of the frame by the number of rows (n_rows) and the number of images per row (n_images_per_row), respectively. The values are stored in the "roi_height" and "roi_width" variables. Finally, the code prints the calculated width of the frame and the width of the ROI.

```
def text_to_speech(string):  
  
    text_speech = pyttsx3.init()  
  
    text_speech.say(string)  
  
    text_speech.runAndWait()
```

The above code snippet is a user defined function which will convert text to speech. It is using the pyttsx3 library to do the conversion [6].

```
# object detector function /method  
def object_detector(image):  
  
    classes, scores, boxes = model.detect(image, CONFIDENCE_THRESHOLD, NMS_THRESHOLD)  
  
    # creating empty list to add objects data  
  
    data_list = []  
  
    for (classid, score, box) in zip(classes, scores, boxes):  
  
        # define color of each, object based on its class id  
        """  
  
        80 object names are stored in classes.txt and 6 colours are stored in var COLORS  
  
        For 1st object name (person) -> 0 % 6 = 0. So colour[0] = RED will be chosen  
  
        For 2nd object name (bicycle) -> 1 % 6 = 1. So colour[1] = PURPLE will be chosen  
  
        And so on...  
        """  
  
        color= COLORS[int(classid) % len(COLORS)]  
  
        # label -> stores class id and confidence score  
  
        label = "%s : %f" % (class_names[classid[0]], score)
```

```

# draw rectangle on and label on object

cv.rectangle(image, box, color, 2)

cv.putText(image, label, (box[0], box[1]-14), FONTS, 0.5, color, 2)


# width of bounding box

bb_width = box[1] - box[0]

bb_height = box[2] - box[3]


# finding the central coordinates of bounding box

mid_x = box[0]

# mid_y = box[1]


# plotting a point at the centre of the bounding box

# cv.circle(frame, (mid_x, mid_y), 5, (255, 0, 255), cv.FILLED)


# finding the x coordinates of the frames

right_end_of_frame1 = roi_width

right_end_of_frame2 = 2*roi_width

# right_end_of_frame3 = right_end_of_frame2 + roi_width


# if the central coordinates lie in the the left/right/middle frame

if(mid_x < right_end_of_frame1):

    text_to_speech('SLIGHTLY ON YOUR LEFT')

elif(mid_x > right_end_of_frame1 and mid_x < right_end_of_frame2):

    text_to_speech('IN FRONT OF YOU')

elif(mid_x > right_end_of_frame2):

    text_to_speech('SLIGHTLY ON YOUR RIGHT')


# getting the data

# 1: class name 2: object width in pixels, 3: position where have to draw text(distance)

if classid == 0: # person class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 67: # cell phone class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

```

```
elif classid == 1: # bicycle class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 2: # car class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 3: # motorbike class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 5: # bus class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 6: # train class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 7: # truck class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 13: # bench class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 15: # cat class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 16: # dog class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 19: # cow class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 39: # bottle class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 56: # chair class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 57: # sofa class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 59: # bed class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 60: # dining table class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 61: # toilet class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 62: # tvmonitor class id
```

```

data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

elif classid == 63: # laptop class id

    data_list.append([class_names[classid[0]], box[2], (box[0], box[1]-2)])

# if you want include more classes then you have to simply add more [elif] statements here

# returning list containing the object data.

return data_list

```

The snippet of code is an example of a user-defined function that will identify the items in the image. In order to distinguish the items accurately, different coloured bounding boxes are first assigned to each one. The class id and confidence score of various items are then stored in the "label" variable. Discover the object's centre to determine its direction, then use class ids to identify and detect the objects in the image.

```

def focal_length_finder (measured_distance, real_width, width_in_rf):

    focal_length = (width_in_rf * measured_distance) / real_width

    print("FOCAL LENGTH: ", focal_length)

    # focal_length = 560

    return focal_length

```

This function will calculate the focal length of the lens.

```

def distance_finder(focal_length, real_object_width, width_in_frame):

    distance = (real_object_width * focal_length) / width_in_frame

    return distance

```

The above function will return the distance between the object and the camera.

```

# Distance constants

KNOWN_DISTANCE = 114.3 #CENTIMETERS

PERSON_WIDTH = 37 #CENTIMETERS

MOBILE_WIDTH = 7 #CENTIMETERS

BOTTLE_WIDTH = 8

BICYCLE_WIDTH = 67.5

CAR_WIDTH = 177

```



```

MOTORBIKE_WIDTH = 82.5

BUS_WIDTH = 90

BENCH_WIDTH = 129.5

CAT_WIDTH = 46

DOG_WIDTH = 51

COW_WIDTH = 183

CHAIR_WIDTH = 43

SOFA_WIDTH = 150

BED_WIDTH = 190

DINING_TABLE_WIDTH = 96.5

TOILET_WIDTH = 51

TV_WIDTH = 100

LAPTOP_WIDTH = 14


# Object detector constant

CONFIDENCE_THRESHOLD = 0.5

NMS_THRESHOLD = 0.3

# colors for object detected

COLORS = [(255,0,0),(255,0,255),(0, 255, 255), (255, 255, 0), (0, 255, 0), (255, 0, 0)]

# COLORS = [red, purple, cyan, brown, green, red]

GREEN =(0,255,0) # To show the distance text above the object in GREEN

BLACK =(0,0,0) # To show the bounding box (rectangle) in BLACK

# defining fonts

FONTs = cv.FONT_HERSHEY_COMPLEX

# getting class names from classes.txt file

class_names = []

with open("classes.txt", "r") as f:

    class_names = [cname.strip() for cname in f.readlines()]

# setting up opencv net

yoloNet = cv.dnn.readNet('yolov4-tiny.weights', 'yolov4-tiny.cfg')

# Reads deep learning network represented in one of the supported formats.

# For more info, check the links below:

# https://docs.opencv.org/3.4/d6/d0f/group\_\_dnn.html#ga3b34fe7a29494a6a4295c169a7d32422

# https://stackoverflow.com/questions/50390836/understanding-darknets-yolo-cfg-config-files

```

```

yoloNet.setPreferableBackend(cv.dnn.DNN_BACKEND_CUDA)

yoloNet.setPreferableTarget(cv.dnn.DNN_TARGET_CUDA_FP16)

model = cv.dnn_DetectionModel(yoloNet)

'''
scale -> scale factor (1/255 to scale the pixel values to [0..1])

size -> 416x416 square image

(swapBR=True) -> since OpenCV uses BGR.

    OpenCV assumes images are in BGR channel order;

    however, the `mean` value assumes we are using RGB order.

    To resolve this discrepancy we can swap the R and B channels in image

    by setting this value to `True`.

    By default OpenCV performs this channel swapping for us.
'''

model.setInputParams(size=(416, 416), scale=1/255, swapRB=True)

```

In the above snippet some distance variables are initialized and the font is set. The model is being created and stored in the 'model' variable.

```

def calc_announce_store_dist(obj_info, width):

    # typecasting to int and removing decimals

    dist_cm = distance_finder(focal_length, width, obj_info[1])

    dist_m = dist_cm / 100

    # use to limit no. of digits after decimal

    approx_dist = round(dist_m, 2)

    dist_text = f'{obj_info[0]} detected {approx_dist} meters'

    text_to_speech(dist_text)

    return approx_dist

```

The above code snippet is used to draw bounding box later.

```

# ACTIVATING AND IMPLEMENTING IN CAMERA

cap = cv.VideoCapture(0) # capturing using cam of pc

```

```

n_rows = 1

n_images_per_row = 3

while True:

    ret, frame = cap.read()

    height,width,ch = frame.shape


    roi_height = height // n_rows

    roi_width = width // n_images_per_row


    '''

    if a person comes in front of the camera, variable data stores

    [['person', 532, (54, 124)]]

    Same for other objects

    '''

    data = object_detector(frame)

    #print(data)


    for d in data:


        if d[0] == 'person':

            x, y = d[2]

            distance = calc_announce_store_dist(d, PERSON_WIDTH)


        elif d[0] == 'cell phone':

            x, y = d[2]

            distance = calc_announce_store_dist(d, MOBILE_WIDTH)


        elif d[0] == 'bottle':

            x, y = d[2]

            distance = calc_announce_store_dist(d, BOTTLE_WIDTH)


        elif d[0] == 'bicycle':

            x, y = d[2]

            distance = calc_announce_store_dist(d, BICYCLE_WIDTH)

```

```
elif d[0] == 'car':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, CAR_WIDTH)
```

```
elif d[0] == 'motorbike':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, MOTORBIKE_WIDTH)
```

```
elif d[0] == 'bus':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, BUS_WIDTH)
```

```
elif d[0] == 'train':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, TRAIN_WIDTH)
```

```
elif d[0] == 'truck':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, TRUCK_WIDTH)
```

```
elif d[0] == 'bench':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, BENCH_WIDTH)
```

```
elif d[0] == 'cat':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, CAT_WIDTH)
```

```
elif d[0] == 'dog':
```

```
    x, y = d[2]
```

```
    distance = calc_announce_store_dist(d, DOG_WIDTH)
```

```
elif d[0] == 'cow':
```

```
x, y = d[2]

distance = calc_announce_store_dist(d, COW_WIDTHH)

elif d[0] == 'chair':

    x, y = d[2]

    distance = calc_announce_store_dist(d, CHAIR_WIDTHH)

elif d[0] == 'sofa':

    x, y = d[2]

    distance = calc_announce_store_dist(d, SOFA_WIDTHH)

elif d[0] == 'bed':

    x, y = d[2]

    distance = calc_announce_store_dist(d, BED_WIDTHH)

elif d[0] == 'diningtable':

    x, y = d[2]

    distance = calc_announce_store_dist(d, DINING_TABLE_WIDTHH)

elif d[0] == 'toilet':

    x, y = d[2]

    distance = calc_announce_store_dist(d, TOILET_WIDTHH)

elif d[0] == 'tvmonitor':

    x, y = d[2]

    distance = calc_announce_store_dist(d, TV_WIDTHH)

elif d[0] == 'laptop':

    x, y = d[2]

    distance = calc_announce_store_dist(d, LAPTOP_WIDTHH)

cv.rectangle(frame, (x, y-3), (x+150, y+23), BLACK, -1)

# green text inside the rectangle

cv.putText(frame, f'DIST: {round(distance, 0)} M', (x+5, y+13), FONTS, 0.48, GREEN, 2)
```

```
cv.imshow('frame', frame)

key = cv.waitKey(1)

if key == ord('q'):
    break

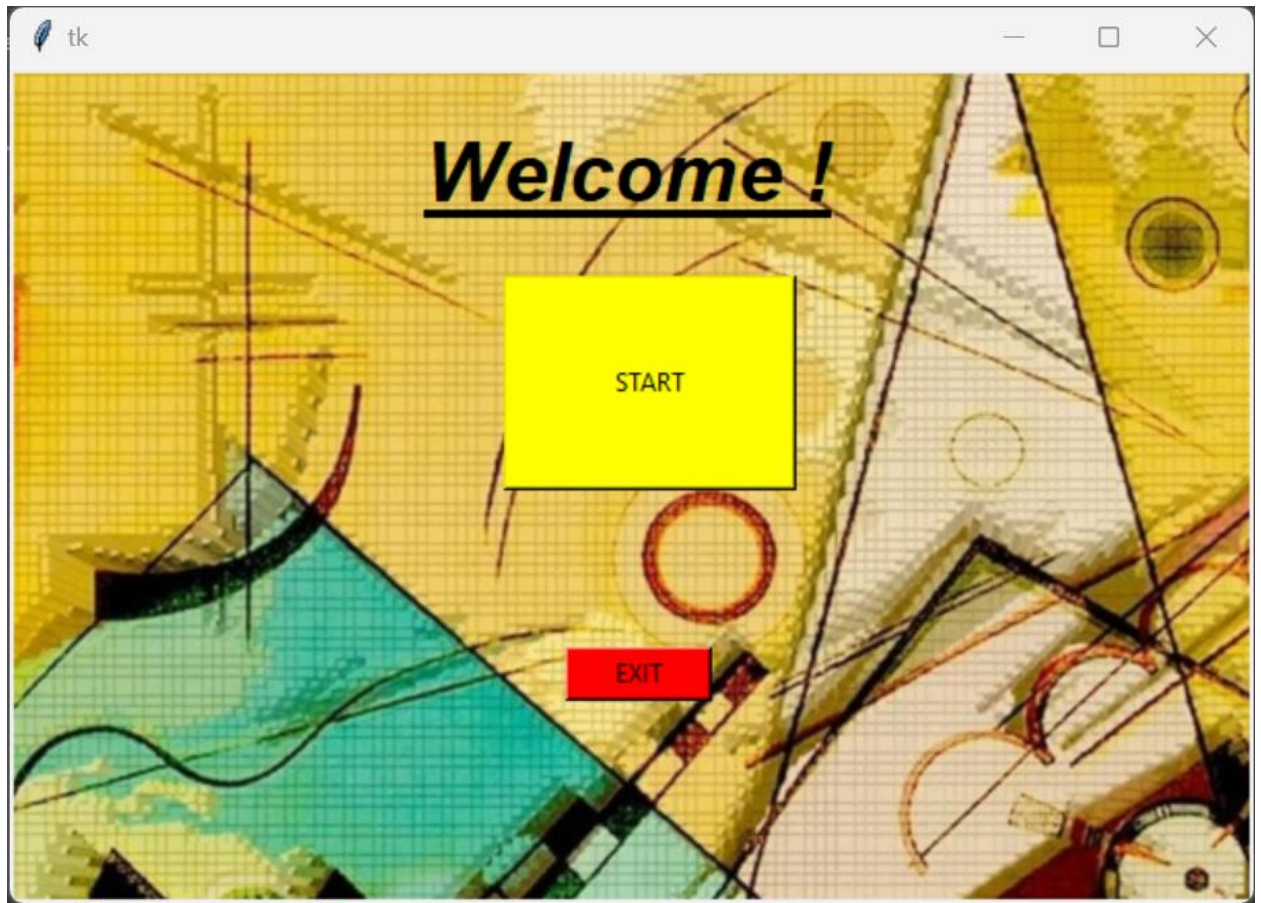
cv.destroyAllWindows()

cap.release()
```

The camera is activated, objects are recognized, bounding boxes are drawn and distance and direction of the objects are calculated with respect to the camera. And the identity of the object, the distance and the direction are converted to speech.

VI. Implementation and results

In this section, describe how the project is developed and implemented. Mention about the case studies done using the proposed technique and findings of the case studies. (**Working model and operation**)



VII PERFORMANCE ANALYSIS



Image 2.1

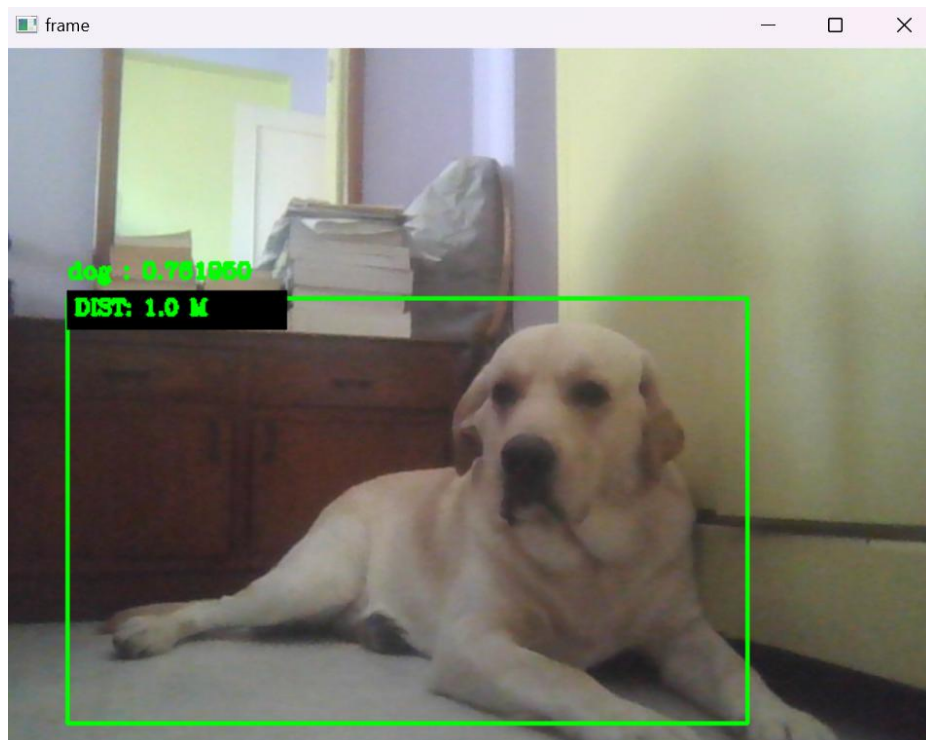
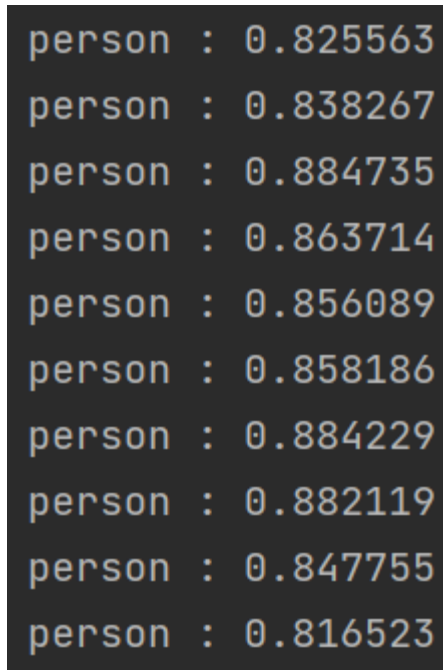


Image 2.2

Images 2.1, 2.2 - Detecting objects / living beings



person	: 0.825563
person	: 0.838267
person	: 0.884735
person	: 0.863714
person	: 0.856089
person	: 0.858186
person	: 0.884229
person	: 0.882119
person	: 0.847755
person	: 0.816523

Image 3 - Testing accuracy of the trained model

Average accuracy = 85.0718%

VIII. SOCIETAL IMPACT AND FUTURE SCOPE

The project combines the latest advancements in computer vision and artificial intelligence to tackle a real-world challenge faced by blind individuals [10]. By using deep learning and computer vision algorithms, the object detection system provides blind individuals with a new level of information about their surroundings. With real-time audio feedback about obstacles, other people, and potential hazards, the blind can confidently navigate their environment, reducing the risk of accidents and increasing their independence.

The impact of this project extends beyond just blind individuals, as it highlights the positive potential of technology to solve societal problems. The application of computer vision and artificial intelligence to this area opens up new possibilities for further development and improvement, potentially leading to a more accessible world for all. The successful implementation of this project also serves as an inspiration to other developers and researchers to develop technology solutions for similar challenges.

The potential for future advancements in this field is immense. The object detection algorithm for blind individuals could be integrated into wearable devices, providing a more hands-free experience for the users. This technology could also be extended to cover a wider range of objects and environments, including indoor spaces, busy streets, and public transportation. The algorithms used in the project could

also be adapted to provide additional information to the users, such as the type of object, its height, and other relevant details. Additionally, the technology could be combined with other assistive technologies, such as voice commands and haptic feedback, to create an even more comprehensive and intuitive navigation system.

Moreover, the development of this project could also stimulate further research into computer vision and artificial intelligence, with potential applications in other areas such as accessibility, healthcare, and robotics. The object detection algorithm for blind individuals is not only a practical solution to a real-world challenge but also a shining example of the positive impact technology can have on society. With continued advancements and improvement, the future scope of this project holds the potential to revolutionize the lives of visually impaired individuals and lead to a more inclusive and accessible world.

IX. CONCLUSION

In conclusion, our proposed solution has the potential to significantly improve the lives of blind individuals, making it safer and easier for them to navigate their environment. The development of wearable technology that integrates this technology can make it more accessible and convenient for the blind to use, providing them with an additional layer of safety and independence.

References

- [1] Diwan, T., Anirudh, G. & Tembhurne, J.V. Object detection using YOLO: challenges, architectural successors, datasets and applications. *Multimed Tools Appl* (2022). <https://doi.org/10.1007/s11042-022-13644-y>
- [2] A. Ćorović, V. Ilić, S. Đurić, M. Marijan and B. Pavković, "The Real-Time Detection of Traffic Participants Using YOLO Algorithm," 2018 26th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2018, pp. 1-4, doi: 10.1109/TELFOR.2018.8611986.
- [3] Papageorgiou, C., Poggio, T. A Trainable System for Object Detection. *International Journal of Computer Vision* 38, 15–33 (2000). <https://doi.org/10.1023/A:1008162616689>
- [4] Amit, Y., Felzenszwalb, P., Girshick, R. (2020). Object Detection. In: Computer Vision. Springer, Cham. https://doi.org/10.1007/978-3-030-03243-2_660-1
- [5] H. Rithika and B. N. Santhoshi, "Image text to speech conversion in the desired language by translating with Raspberry Pi," 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Chennai, India, 2016, pp. 1-4, doi: 10.1109/ICCIC.2016.7919526.

- [6] P. Manage, V. Ambe, P. Gokhale, V. Patil, R. M. Kulkarni and P. R. Kalburgimath, "An Intelligent Text Reader based on Python," 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, 2020, pp. 1-5, doi: 10.1109/ICISS49785.2020.9315996.
- [7] Serrão, M., Shahrabadi, S., Moreno, M. *et al.* Computer vision and GIS for the navigation of blind persons in buildings. *Univ Access Inf Soc* 14, 67–80 (2015). <https://doi.org/10.1007/s10209-013-0338-8>
- [8] R. Tapu, B. Mocanu and T. Zaharia, "A computer vision system that ensure the autonomous navigation of blind people," 2013 E-Health and Bioengineering Conference (EHB), Iasi, Romania, 2013, pp. 1-4, doi: 10.1109/EHB.2013.6707267.
- [9] J. R. Terven, J. Salas and B. Raducanu, "New Opportunities for Computer Vision-Based Assistive Technology Systems for the Visually Impaired," in *Computer*, vol. 47, no. 4, pp. 52-58, Apr. 2014, doi: 10.1109/MC.2013.265.
- [10] Tian, Y., Yang, X., Yi, C. *et al.* Toward a computer vision-based wayfinding aid for blind persons to access unfamiliar indoor environments. *Machine Vision and Applications* 24, 521–535 (2013). <https://doi.org/10.1007/s00138-012-0431-7>

Appendix A: Project Code (if any)