OBJECT DETECTION WITH VOICE ALERT FOR BLIND

A Report submitted to the Rajiv Gandhi University of Knowledge and Technologies in partial fulfilment of the degree of

Bachelor Of Technology

in

Computer Science and Engineering

Submitted by

P.Prameela (S180048)

J.Bhuvanasree (S180594)

T.Jayasree (S180334)

U.sirisha (S180777)

B.Byularani (S180684)

4th year BTech 2st Semester

Under the supervision of

Mrs. S. Lakshmi sri M.Tech

Asst. Professor-Department of CSE

RGUKT Srikakulam

Department of Computer Science and Engineering

Rajiv Gandhi University of Knowledge and Technologies, Srikakulam

S.M. Puram (V), Etcherla (M), Srikakulam (Dt) – 532410

CERTIFICATE

This is to certify that the report entitled "**Object Detection With Voice Alert For BLIND**" was submitted by P.Prameela, bearing ID. No. S180048, J.Bhuvanasree, bearing ID. No. S180073,T.Jayasri bearing ID. No. S180334,U. Sirisha bearing ID No. S180777, B.Byularani bearing ID No. S180684 in partial fulfilment of the requirements for the award of Bachelor of Technology in Computer Science is a Bonafede work carried out by them under my supervision and guidance.

The report has not been submitted previously in part or in full to this or any other University or Institution to award any degree or diploma.

Mrs. S. Lakshmi sri,	**Mrs. CH.Lakshmibala,**
Project Guide,	**Head of the Department,**
Department of CSE,	**Department of CSE,**
RGUKT, SRIKAKULAM	**RGUKT, SRIKAKULAM**

DECLARATION

We declared that this thesis work entitled "**Object Detection With Voice Alert For BLIND**" is carried out by us during the year 2023-24 in partial fulfilment of the requirements for the Major Project in Computer Science and Engineering.

We further declare that, the work contained in this report is original and has been done by us under the guidance of my supervisor(s). The work has not been submitted to any other Institute for any degree or diploma. We have followed the guidelines provided by the University in preparing the report. We have confirmed the norms and guidelines given in the Ethical Code of Conduct of the University. Furthermore, the technical details furnished in various chapters of this thesis are purely relevant to the above project and there is no deviation from the theoretical point of view for design, development and implementation.

Date:

Place: Srikakulam

P.Prameela(S180048)

J.Bhuvanasree(S180073)

T.Jayasri(S180334)

U.Sirisha(S180777)

B.Byularani(S180684)

ACKNOWLEDGEMENTS

We would like to express my sincere gratitude to, my project Guide **S. Lakshmi sri**, for valuable suggestions and keen interest throughout the progress of my course of research.

We are grateful to **Ch. Lakshhmibala** , HOD CSE, for providing excellent computing facilities and a congenial atmosphere for progressing with my project.

At the outset, We would like to thank **Rajiv Gandhi University of Knowledge and Technologies**, Srikakulam for providing all the necessary resources for the successful completion of our course work. At last, but not least we thank our classmates and other students for their physical and moral support.

ABSTRACT

As object recognition technology has developed recently, various technologies have been applied to autonomous vehicles, robots, and industrial facilities. However, the benefits of these technologies are not reaching the visually impaired, who need it the most. This paper proposed an object detection system for the blind using deep learning technologies. This system provides voice alerts to the blind, conveying the distance of objects detected in front of them. The object recognition deep learning model utilizes the You Only Look Once(YOLO) algorithm and a voice announcement is synthesized using text-tospeech (TTS) to make it easier for the blind to get information about objects. Asa result, it implements an efficient object-detection system that helps the blind find objects in a specific space without help from others, and the system is analyzed through experiments to verify performance.

Keywords: You Look Only Once (YOLO), Text-to-speech (TTS)

CONTENTS

CHAPTER-1 1

INTRODUCTION 1

1.1 Introduction 1

1.2 Motivation 1

1.3 Problem Statement 1

1.4 Objectives 2

1.5 Goal 2

1.6 Scope 3

1.7 Applications 3

1.8 Limitations 4

CHAPTER-2 5

LITERATURE SURVEY 5

2.1 Collect Information 5

2.2 Study 6

2.3 Summary 6

CHAPTER-3 7

EXISTING SYSTEM 7

3.1 Existing System 7

3.2 Disadvantages 8

CHAPTER-4 9

PROPOSED SYSTEM 9

4.2 Advantages 10

4.3 System Requirements 10

CHAPTER-5 11

METHODS AND ALGORITHMS 11

5.1 Algorithms 11

CHAPTER-6 ..... **Error! Bookmark not defined.**

SYSTEM DESIGN 16

6.1 UML : 16

6.2 Characteristics of UML : 17

6.3 Goals of UML : 18

6.4 Class Diagram : 19

6.5 Use case Diagram : 21

6.6 DFD Diagrams : 23

CHAPTER-7 27

SYSTEM IMPLEMENTATION 27

7.1 Implementation Tools 27

7.2 Frameworks 28

7.3Source code : 32

CHAPTER-8 56

SYSTEM TESTING..... 56

8.1 Introduction 56

8.2 Types of testing 57

8.2.1 Unit testing :..... 57

8.2.2 Integration testing :..... 58

8.2.3 Functional Testing :..... 58

8.3 System testing 59

8.4 White Box Testing..... 60

8.5 Black Box Testing :..... 61

8.3 Levels of Testing: 62

8.4 Integration Testing Stratagy: 62

CHAPTER-9 64

CONCLUSION 64

9.1 Conclusion..... 64

CHAPTER-10 REFERENCES

..... 65

CHAPTER-1

INTRODUCTION

1.1 Introduction

In today's advanced hi-tech environment, the need for self-sufficiency is recognised in the situation of visually impaired people who are socially restricted. Visually impaired people encounter challenges and are at a disadvantage as a result of a lack of critical information in the surrounding environment, as visual information is what they lack the most. The visually handicapped can be helped with the use of innovative technologies. The system can recognise items in the environment using voice commands.It may be an effective approach for blind persons to interact with others and may aid with their independence. Those who are wholly or partially blind are considered visually impaired. According to the World Health Organization (WHO), 285 million people worldwide suffer from vision impairment, 39 people are blind, and around 3% of the population of all ages is visually impaired. Visually impaired people go through a lot and encounter a lot of difficulties in their daily lives, such as finding their way and directions, as well as going to places they don't go very often.

1.2 Motivation

To make it possible for those who are blind to independently travel like everyone else. The objective of this project is to develop a visual aid system that will allow users to travel independently by becoming familiar with their surroundings, avoiding potential hazards, and identifying objects.

1.3 Problem Statement

the blind by a software-based system that helps them with voice commands using text-to-speech conversion The confidence to move independently without the help of another person would be given to techniques and helps the visually impaired by detecting the objects using the camera in front of them

1.4 Objectives

The objectives of object detection involves :

1) Object Detection

Our project's first module, object detection, acts as its cornerstone. In essence, it involves employing the YOLOv4 Algorithm to find nearby and distant objects. Because people with visual impairments are the primary audience for our application, detection is essential. The item is detected and a rectangle box is made around it.

2) Distance Estimation

Following the goal of detecting the objects comes the goal of distance estimation. A crucial factor that affects the precision and range of distance measurements is the distance estimate module's focal length. The focal length describes the separation between the subject of the lens and the viewer.

3) Text to speech conversion

One of the essential elements of the project that enables the system to transform text into audible voice is the text-to-speech conversion utilising the Pyttsx3 module.

4) Deploying the Project

It is simple to deploy the project using Netlify, which enables the system to be hosted and used online. The cloud-based platform Netlify is perfect for launching web-based apps like the one we are creating because it offers hosting and continuous deployment for web projects.

1.5 Goal

1. Low cost and effective

A low-cost and efficient option for the project that enables the system to detect and gauge the distance of things in the surroundings is object detection and distance estimation utilising

focal length. The system can take pictures of the surroundings and precisely estimate the distance of objects in the range of view by employing a camera-based system.

One of the main advantages of this approach is that it is a low cost solution that can be implemented using affordable off-the-shelf components. This makes it an attractive option for projects with budget constraints or limited resources.

2. Simple fast

The cutting-edge object detection method YOLOv4 makes use of deep neural networks to identify things in real-time. It is the perfect option for the project's object detection needs because it is extremely precise and effective. The system can quickly and efficiently determine the distance to objects in the environment by integrating YOLOv4 with a camera-based distance estimate algorithm that uses focus length.

3. User friendly

One of the main advantages of this technology is the easy and straightforward user interface that enables people to interact with the system and perform object recognition and distance estimation tasks. The combination of YOLOv4 and focal length-based distance estimation, which yields precise and reliable results, significantly enhances the user experience.

1.6 Scope

In order to achieve high accuracy and real-time performance, modern object recognition techniques will be implemented in the project. In this project, we use Python and a Tensor Flow-based framework to tackle object detection from beginning to end. The system that forms is swift and accurate. The device was developed using a Tensor Flow-based web application that makes use of its built-in camera to recognise objects.

1.7 Applications

- Used to detect the objects.
- Calculating the distance of object from camera
- Converting the text into speech
- Integrating the python files with HTML file using the flask app

1.8 Limitations

1.Works only with certain things

This method's potential inability to detect and measure all kinds of environmental objects is one of its key drawbacks. Since YOLOv4 was trained on a particular set of objects, it might not be able to detect objects that are not part of this set with accuracy. Similar to this, an item with an unusual shape or size may cause the distance calculation algorithm based on focal length to be inaccurate.

2. It won't recognize side things that appear unexpectedly

The system might not be able to accurately detect items that are positioned outside the camera's field of vision, which is one of this method's shortcomings. This means that the system might not be able to detect an object accurately if it quickly appears from the side or is partially blocked.

3. It won't recognize when the image is unclear

One of the major downsides of this approach is that the system relies on accurate visual signals to consistently detect and measure objects in the surroundings. As a result, if the help being provided or the visual signals are vague, the system might not be able to accurately detect and measure the objects of interest.

4. Assisted only in the English language

A self-walking aid system for the blind aims to detect impediments in the user's path using machine learning algorithms and to offer auditory or haptic feedback to assist the user in navigating past them. The system may also have capabilities like voice recognition and GPS navigation for hands-free use, but it only provides assistance in English.

CHAPTER-2

LITERATURE SURVEY

2.1 Collect Information

1. This study included the standard datasets as well as an explanation of the current detection model methodologies. This work discussed many detectors, including one-stage and two-stage detectors, which aided in the analysis of various object detection strategies and also gathered some conventional as well as novel applications. Additionally, it listed a few branches connected to object recognition. Additionally, some development trends were highlighted in order to better follow the set of art algorithms and subsequent processes.
2. You only look once: Unified real-time object detection" is a landmark study that was published in the IEEE conference proceedings on computer vision and pattern recognition in 2016. The YOLO (You Only Look Once) technique is described in the paper. It is a unified real-time object detection system that can identify many items in an image within a single neural network forward pass. By dividing the image into a grid of cells, the YOLO algorithm determines the bounding boxes and class probabilities for each cell. The technique also makes use of a distinctive loss function that balances classification and localization mistakes. The study demonstrates that YOLO surpasses cutting-edge object detection techniques in terms of accuracy and speed, making it a workable option for real-time applications.
3. After reviewing numerous studies in the area of pattern recognition, it provides an effective demonstration of the technique for identifying an item and analysing its gesture using machine learning and computer vision for decision-making.
4. The author suggested a well-known computer technique related to computer vision and image processing that focuses on identifying items or instances of those objects in computerised images and videos (such as people, flowers, and other creatures). Numerous applications of object detection, such as face detection, character recognition, and vehicle calculator, have been thoroughly researched. Using object detection can help with recovery and monitoring, among other things. The effectiveness and accuracy of object detection are improved in this study by utilising

various key ideas in object detection while utilising the OpenCV package of Python 2.7.

5. According to the author, everyone has the right to live independently, especially those who are impaired. Over the past few decades, technology has made it possible for people with disabilities to live more independently. In this study, an assistive system for the blind is proposed that makes use of OpenCV running on a Raspberry Pi 3 and Python-based deep neural network-based YOLO for object recognition in photos and video streams. The results show that the suggested strategy is effective at giving blind users the freedom to roam around in unfamiliar indoor-outdoor environments, thanks to a user-friendly gadget that uses a person- and object-identification model.

2.2 Study

Key features are

1.Detecting the objects in front of him using the camera

2.Calculate the distance from object to camera using the focal length

3.After detecting the objects then it will identify what it is and using the text to speech conversion technique the system will assist using the voice commands

2.3 Summary

To enable the blind to move around autonomously without assistance from others, we created a system with capabilities including object detection, distance calculation, and voice instructions.

CHAPTER-3

EXISTING SYSTEM

3.1 Existing System

SVM Algorithm:

The SVM method is used to characterise and identify objects automatically. In particular, the SVM is used as a binary classifier, categorising two classes, namely, object and background. The algorithm's objective is to accurately detect an object against its background while using the fewest training samples possible.

Limitations:

SVM algorithm is not suitable for large data sets

SVM struggles when the dataset contains more noise.

CNN Algorithm:

CNN is used to identify objects in images. The internal operations of CNN were succinctly described.

Steps in CNN object detection Using a picture as input

1. Separate the picture into several areas 3. Each section is viewed as a distinct picture.
4. Transmit all of these pictures to CNN, who will categories them into different groups.
5. Merge all of these zones to get a unique picture that includes the detected object

Limitations:

1. This algorithm runs slowly.
2. An image's object detection process takes roughly 47 seconds.
3. Training takes more than one step to complete.

4. The training procedure takes time since there are several models for doing various portions.

RCNN Algorithm:

The RCNN is the selective search algorithm works is that, it applies a segmentation algorithm to find blobs in an image to figure what could be an object.

Limitations:

1. It is an independent component at each level of a multi-stage model.
2. RCNN depends on the selective search algorithm for generating region proposals which takes a lot of time.

TensorFlow:

A foundation for building deep learning networks that address the object detection issue is provided via the TensorFlow object detection API. The model zoo already has trained models. We have the ability to create our own models.

A still image or video can have an object that is found, tracked, and recognised using a computer vision technique called TensorFlow object detection. By identifying objects, the technique helps us understand the image or video more thoroughly and helps us understand how the models work.

Limitations:

1. It takes more time for training
2. Need powerful GPU for computation

3.2 Disadvantages

1. More expensive
2. Consumes more power
3. Need external hardware

CHAPTER-4

PROPOSED SYSTEM

4.1 Proposed System

We developed a software-based system that uses the YOLOv4 algorithm for object detection and distance estimation. These algorithms make use of deep convolutional neural networks to identify objects using cameras and to determine their names. The system also uses text-to-speech conversion to help the blind respond to voice commands.

Working procedure of YOLOv4 Algorithm:

- **Input Preparation:** YoloV4 prepares input by resizing a picture to a fixed size that is defined in the model setup. After normalising the input image to make sure the pixel values are within a predetermined range, any required preprocessing is then applied.
- **Feature Extraction:** A convolutional neural network (CNN) is used to extract features from the input image. YoloV4's network is based on a fast and effective form of the DarkNet architecture called the DarkNet architecture.
- **Object Detection:** Several detection layers are used to identify objects in the image using the output of the feature extraction network. For objects in the image at various scales and locations, each detection layer forecasts a collection of bounding boxes and related confidence scores.
- **Non-Maximum Suppression:** A Non-Maximum Suppression (NMS) technique is used to filter out overlapping detections and choose the most certain bounding boxes for each object once the detection layers output their predictions.
- **Output Post-Processing:** A set of bounding boxes with corresponding class names and confidence ratings for the items found in the input image make up YoloV4's final output. The application of these detections for further tasks like object tracking or image segmentation follows.

4.2 Advantages

1. YOLOv4 algorithm uses single shot detection (SSD), so it is very fast
2. No need of any hardware devices
3. Less expensive
4. Low consumption of power

4.3 System Requirements

Software requirements:

1. Python IDE
2. ML modules like OpenCV,pyttsx3,Speechrecognition and DCNN
3. Windows 10
4. VScode

Hardware requirements

1. RAM:4gb or above
2. Harddisk

CHAPTER-5

METHODS AND ALGORITHMS

5.1 Algorithms

YOLOv4:

YOLOv4, or you only look once, is an acronym. Several objects can be detected in a single frame by this real-time object detection system. Compared to previous recognition systems, YOLO recognises objects with greater accuracy and speed. Up to 9000 classes, as well as unforeseen classes, may be predicted.

The real-time recognition system will extract many things from an image and build a boundary box around each object. It is straightforward to utilise in a production system and to learn.

Yolo is a furistic recognizer that performs better in terms of accuracy and FPS than current detectors. A standard GPU may be used to run the detector and train it, making it widely adoptable. New features in YOLOv4 have increased the accuracy of the classifier and detector and can be used in a variety of research projects.

Following the division of the input image into a grid of cells, the system forecasts bounding boxes and class probabilities for each cell.

Here is a step-by-step algorithm for YOLOv4:

1. Enter the image to be processed and specify the grid size for the image's cell division.
2. To extract feature maps, apply a number of convolutional layers to the input image.
3. Predict the objectness score and bounding box coordinates for each grid cell using the feature maps.
4. This is achieved by estimating the size and aspect ratio of the objects using a series of anchor boxes, which are predetermined bounding boxes.

5. Determine each anticipated bounding box's confidence score, which indicates the likelihood that it will include an object. The objectness score and the class probability score are combined to create the confidence score.
6. Use non-maximum suppression to get rid of redundant detections and choose the most.
7. Continue with step 6 for each image in the input dataset.
8. Using a loss function that balances localization and classification errors, train the model using a collection of annotated images.
9. If necessary, fine-tune the model using a smaller dataset or particular task.
10. Evaluate the model's precision and speed using a fresh dataset.
11. Use the model in a real-time application, such as autonomous driving or video analysis.

Overall, the YOLOv4 algorithm expands on the strengths of earlier iterations of the YOLO algorithm by including a number of novel features, such as a larger network design, a more potent backbone, and advanced training methodologies, to attain cutting-edge performance in terms of accuracy and speed.

Speech Recognition:

Speech recognition is the ability of a machine or program to understand words spoken aloud and translate them into readable text. Basic speech recognition software has a limited vocabulary and can only recognise words and phrases that are spoken clearly. Software that is more sophisticated can handle different accents, languages, and natural speech. Speech recognition makes use of research from computer science, linguistics, and computer engineering.

Here is a step-by-step algorithm for speech recognition:

1. the speech-containing audio signal.
2. To improve the quality of the voice signal, preprocess the audio stream by applying noise reduction, filtering, and normalisation.

3. Use a method known as windowing to divide the audio signal into brief time intervals, usually around 10 milliseconds. As a result, a string of frames is produced, each of which contains a piece of the audio stream.
4. Using the feature extraction method, separate each audio frame into a group of feature vectors that characterise the spectral and temporal characteristics of the speech signal. Typical techniques for feature extraction include spectrograms, linear predictive coding (LPC), and Mel Frequency Cepstral Coefficients (MFCCs).
5. Calculate the likelihood of each word or group of words that corresponds to the extracted feature vectors using a language model. The vocabulary.
6. Use a decoding method to identify the word order that most closely matches the feature vectors. The Viterbi algorithm and beam search are typical decoding algorithms.
7. To increase the recognition's precision, post-process the words using techniques like language modelling, grammar checking, and error repair.
8. Display the commands or recognised text to the user or a subsequent application.
9. Train and perfect the speech recognition system utilising a sizable dataset of text transcriptions and annotated audio signals.
10. To assess the speech recognition system's performance and accuracy, subject it to various audio signals.

Feature extraction, language modelling, decoding, and post-processing are all parts of the lengthy process of voice recognition. Frequently, modern speech recognition systems.

DCNN:

Computer vision tasks like image categorization and object detection frequently employ Deep Convolutional Neural Networks (DCNNs), a type of artificial neural network. They are made to automatically learn visual feature hierarchies from the underlying raw image data.

Convolutional, pooling, and fully linked layers are among the layers that make up DCNNs. Convolutional layers apply filters to the input image to convolve over it and extract features like edges and corners. The feature maps' dimensionality is decreased by pooling layers, which also increases the network's resistance to slight input changes. Finally, the image is classified based on the learnt characteristics using fully connected layers.

Hierarchically, where lower-level features are joined to create higher-level features, DCNNs are capable of learning complicated features. As a result, they are able to perform at the cutting edge on a variety of computer vision tasks, such as picture classification, object identification, and image segmentation.

1. Enter a frame from an image or video that includes one or more items.
2. Use an object detection model that has already been trained to recognise and localise the objects in the image. This model is often based on a DCNN architecture. A backbone DCNN that pulls features from the image is often used as the foundation of an object detection model, which is then followed by one or more detection heads that forecast the type and location of each object.
3. Using the focus length and object size, calculate the distance of each object from the camera. The camera calibration settings can be used to estimate the focal length, and the bounding box dimensions or other indicators, such as the object's pixel count, can be used to determine the object's size.
4. Output any additional information, such as confidence scores or tracking IDs, along with the class, location, and distance of each object that was detected.
5. You can also use post-processing techniques, like non-maximum suppression, object tracking, or multi-object tracking, to improve the results of object detection and distance estimates.
6. Using a sizable dataset of annotated photos and distance measurements, train and fine-tune the object detection and distance estimation model.
7. Check the model's performance and accuracy using a range of photos and videos, and make any necessary adjustments.

In general, DCNN-based object detection and distance measuring systems are effective tools for a variety of applications, including as autonomous driving, robotics, and

surveillance. Through careful DCNN model training and tuning, as well as the application of cutting-edge post-processing techniques, the performance of these systems can be significantly enhanced.

Text to Speech Conversion:

A process known as text-to-speech (TTS) conversion converts written material into spoken words. A machine can now read text aloud in a natural voice thanks to this technology.

TTS conversion normally takes a number of steps. The words, punctuation, and structure of the text are first identified and analysed. The next step is to break the words down into phonemes, which are the fundamental building blocks of language. The desired speech output is then produced by combining the phonemes.

Concatenative synthesis, which includes pre-recording and sewing together small units of speech, and statistical parametric synthesis, which entails training a machine learning model on a sizable corpus of speech data, are two of the approaches used in TTS conversion.

interfaces TTS conversion has a wide range of uses, including giving visually impaired persons auditory feedback, producing audio versions of digital content like e-books and news stories, and making digital more accessible to those who struggle with reading. TTS is a valuable tool for a variety of applications as it advances in sophistication, with more natural sounding voices and better language comprehension.

1. Enter the text that will be spoken.
2. Transform the text into phonemes, which are a language's smallest units of sound.
3. Use a language model to generate the right intonation and pronunciation for the text by applying it to the phonemes.
4. Using a speech synthesis algorithm like concatenative synthesis or parametric synthesis, create the speech waveform using the phonemes and language model output.
5. Produce an audio signal from the synthesised speech waveform.
6. You might also use post-processing methods to enhance the output speech quality, such as noise reduction or equalisation.

CHAPTER-6

SYSTEM DESIGN

6.1 UML :

- UML is a standardized modeling language used in software engineering to visually represent system designs.
- It provides a set of graphical notations for creating visual models of software systems.

- UML helps in visualizing, specifying, constructing, and documenting the artifacts of a software system.
- It serves as a common language for developers, analysts, and stakeholders to communicate and understand the system's architecture and behavior.

6.2 Characteristics of UML :

- **Generalized Modeling Language**:
  - UML serves as a generalized modeling language applicable across various domains and industries, including software engineering, business processes, and system design.
  - Its flexibility allows it to be adapted to different modeling needs, ranging from high-level conceptual modeling to detailed implementation modeling.

- **Distinct from Programming Languages**:
  - Unlike programming languages such as C++, Python, Java, etc., UML focuses on modeling the structure and behavior of systems rather than implementing them.
  - While programming languages are used to write code for executing tasks, UML is used to create visual representations that aid in understanding and communicating system designs.

- **Interrelated to Object-Oriented Analysis and Design (OOAD)**:
  - UML is closely tied to the principles of object-oriented analysis and design (OOAD), providing graphical notations to represent concepts like classes, objects, inheritance, polymorphism, etc.
  - It facilitates the application of object-oriented methodologies in system development by enabling visual modeling of system components and their interactions.

- **Visualization of Workflow**:
  - UML allows for the visualization of the workflow of a system through various types of diagrams, each focusing on different aspects of the system's functionality and behavior.
  - Diagrams like use case diagrams, activity diagrams, sequence diagrams, and state diagrams help depict the flow of activities, interactions between system components, and the overall behavior of the system.
- **Pictorial Language for Modeling**:
  - UML employs a pictorial language consisting of standardized symbols, notations, and diagrams to represent system elements and relationships.
  - It enables stakeholders to create visual models that convey complex information in a clear and concise manner, fostering better understanding and communication among team members and stakeholders.
- **Standardization and Adoption**:
  - UML is an industry-standard modeling language endorsed by the Object Management Group (OMG), ensuring consistency and interoperability across different tools and methodologies.
  - Its widespread adoption in the software industry makes it a valuable skill for professionals involved in system design, development, and maintenance.

6.3 Goals of UML :

- **General-purpose Modeling Language**:
  - Due to its general-purpose nature, UML can be effectively utilized by modelers across various industries and domains.

- It provides a common platform for representing system designs, regardless of the specific application or industry, making it accessible to a wide range of users with diverse modeling needs.

- **Origins in Object-Oriented Development**:
  - UML emerged as a response to the need for standardized methods to systemize and consolidate the growing popularity of object-oriented development.
  - With the proliferation of object-oriented concepts in software engineering, there was a lack of standardized modeling techniques, prompting the development of UML as a unified language for modeling object-oriented systems.

- **Versatility in Audience**:
  - UML diagrams cater to a diverse audience, including business users, developers, and individuals with varying levels of technical expertise.
  - Its visual representations are designed to be understandable by anyone seeking to comprehend the system, whether they are involved in software development or not.
  - UML can be applied to model both software and non-software systems, accommodating a wide range of applications and domains.

- **Simple Modeling Approach**:
  - UML offers a straightforward and intuitive modeling approach, allowing users to create representations of practical systems in a systematic manner.
  - Its standardized notation and diagramming techniques simplify the modeling process, enabling modelers to capture and communicate system designs effectively.
  - By providing a common language for modeling, UML facilitates collaboration and understanding among stakeholders involved in system development and analysis.

6.4 Class Diagram :

- A Class Diagram, a fundamental element of UML, is a static structure diagram used to visualize the structure of a system.

- It illustrates the composition of a system by depicting classes, their attributes, and the relationships between them.
- The primary objective of a Class Diagram is to provide a comprehensive overview of the classes within a model, facilitating a deeper understanding of the system's structure.

➢ **Components of a Class Diagram:**

- A Class Diagram typically consists of classes, represented by rectangles, which encapsulate data and behavior.
- Within each class, attributes (member variables) and operations (member capabilities or methods) are specified to define its properties and behaviors.
- Relationships between classes, such as associations, generalizations, aggregations, and compositions, are depicted using various types of connectors to illustrate how classes interact with each other.

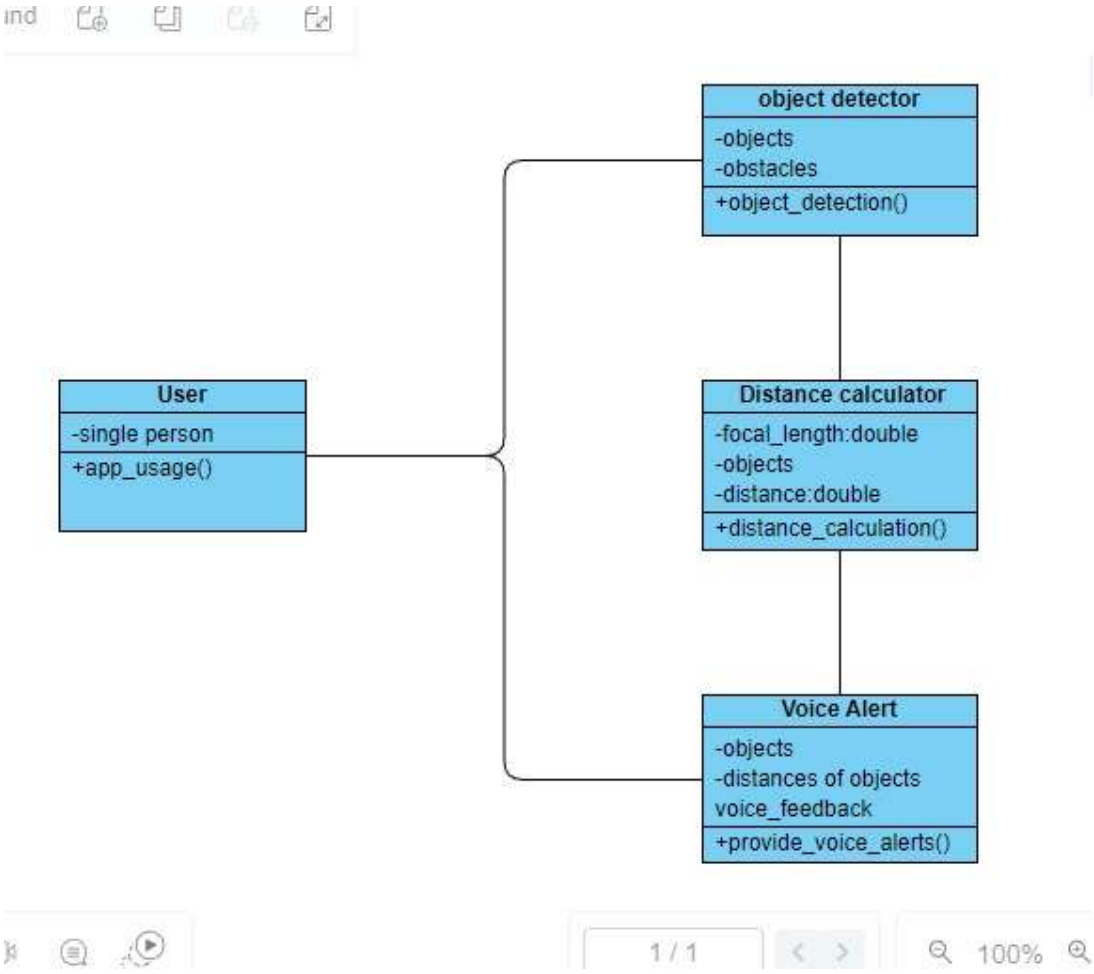
➢ **Attributes and Operations:**

- Classes in an object-oriented software system possess attributes, which represent the data associated with the class.
- Attributes are depicted within classes and represent the member variables that store data specific to instances of the class.
- Operations, also known as methods, define the behaviors or actions that objects of the class can perform.
- Together, attributes and operations define the state and behavior of objects within the system, encapsulating its functionality.

➢ **Representation of Relationships:**

- Class Diagrams visualize the relationships between classes, providing insights into how different classes are related and interact within the system.

- Associations represent connections between classes, indicating how instances of one class are related to instances of another class.
- Generalization (inheritance), aggregation, and composition relationships depict hierarchical structures and dependencies between classes, defining the inheritance and part-whole relationships.



6.5 Use case Diagram :

- **Visual Representation of Interactions**:
  - A Use Case Diagram in UML visually represents how external actors interact with the system to achieve specific goals or tasks.

- It provides a high-level overview of the functional aspects of the system, illustrating the system's behavior from the perspective of its users or external entities.

➢ **Capture of Functional Aspects**:

- Use Case Diagrams capture the functional requirements of a system by depicting the various use cases, or functionalities, that the system provides to its users.
- Each use case represents a distinct action or task that a user can perform within the system to accomplish a specific goal or objective.

➢ **Elements of a Use Case Diagram**:

- The system itself is represented as a rectangle with its name or identifier written inside, encapsulating all the functionalities provided by the system.
- External actors, such as users, other systems, or entities interacting with the system, are depicted as stick figures outside the system boundary.
- Use cases, representing the specific functionalities or tasks offered by the system, are depicted as solid-bordered ovals labeled with descriptive names.

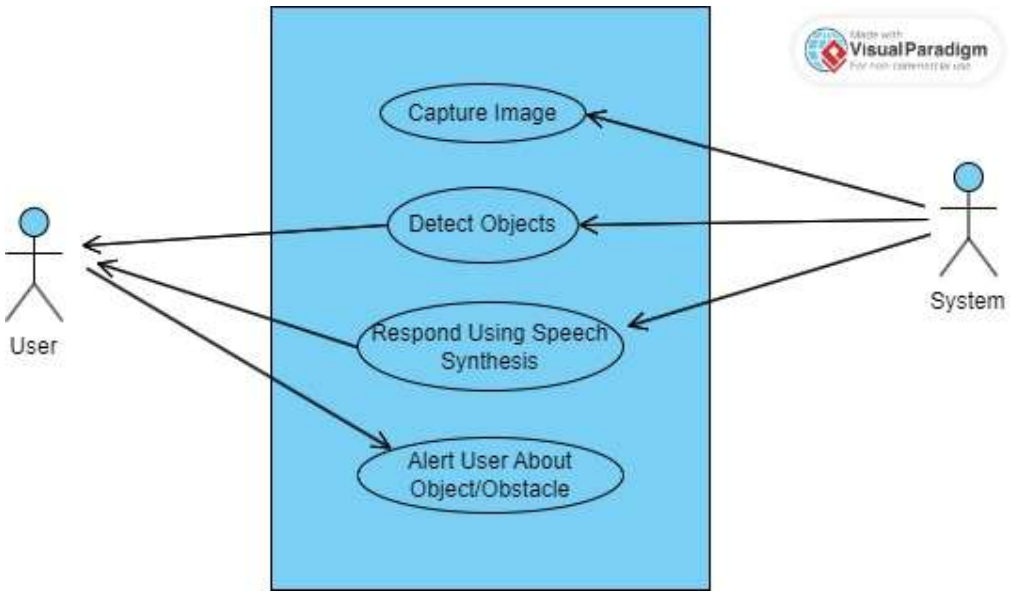
➢ **Representation of Relationships**:

- Relationships between actors and use cases are illustrated using lines or arrows connecting them on the diagram.
- An arrow pointing from an actor to a use case indicates that the actor initiates or interacts with that particular use case.
- These relationships depict the interactions between users and the system, outlining the roles and responsibilities of each actor in relation to the system's functionalities.

➢ **Symbols Used in Use Case Diagrams**:

- Use Case Diagrams utilize specific symbols to represent different elements:
- Rectangles represent the system boundary, encapsulating all functionalities.
- Stick figures depict external actors interacting with the system.

- Ovals represent use cases, outlining specific functionalities or tasks.
- Lines or arrows symbolize relationships between actors and use cases, indicating interactions and dependencies.



6.6 DFD Diagrams :

- A Data Flow Diagram (DFD), also known as a bubble chart, is a graphical representation used to illustrate the flow of data within an information system.
- It models the process aspects of the system, providing an overview of how data moves through the system's processes.

➢ **Preliminary Step for System Overview**:

- DFDs are often used as a preliminary step to create an overview of the system's data flow, which can later be elaborated upon.
- They help stakeholders visualize the flow of data through the system, understand its structure, and identify potential areas for improvement or optimization.

➢ **Visualization of Data Processing**:

- DFDs can be used to visualize data processing within the system, facilitating structured design and analysis.
- They depict how data is input into the system, processed, and outputted, as well as where the data originates from, where it goes, and where it is stored.

➢ **Representation of Information Flow**:

- DFDs illustrate the types of information that are input to and output from the system, as well as the sources and destinations of the data.
- They show how data flows between various processes, data stores, and external entities, providing a clear understanding of the system's data flow dynamics.

➢ **Exclusion of Timing Information**:

- Unlike flowcharts, DFDs do not depict information about the timing of processes or whether processes operate in sequence or in parallel.
- They focus solely on the flow of data through the system, without detailing the temporal aspects of process execution.

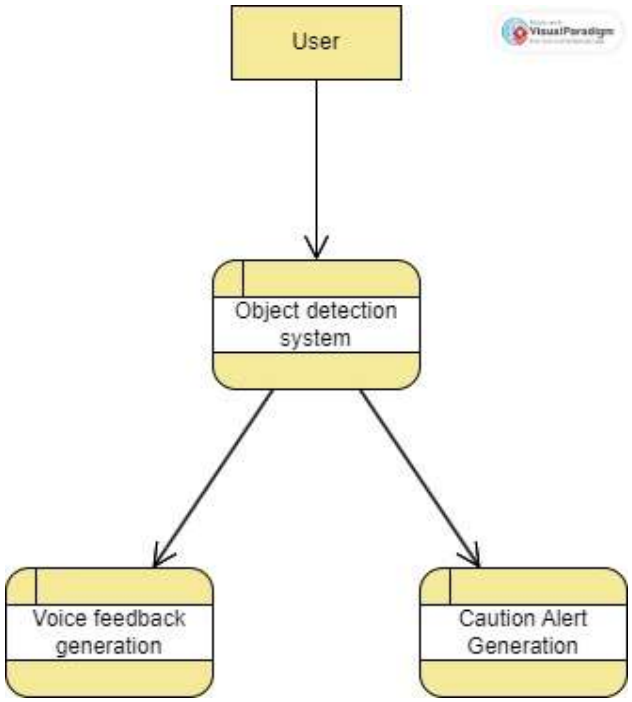
➢ **Primitive Symbols for Construction**:

- The primitive symbols used for constructing DFDs include:
- Circles or bubbles represent processes or activities within the system that manipulate data.
- Arrows represent the flow of data between processes, data stores, and external entities.
- Squares represent data stores where data is stored within the system.
- External entities, such as users or other systems, are represented by rectangles.

6.6.1 Level 0 -- DFD :

1. **Blind User**: Represents the user of the system who is visually impaired and interacts with the system.
2. **Object Detection**: This component is responsible for detecting objects using the YOLO algorithm. It takes input from the environment through sensors or cameras.

3. **Voice Alert**: This component synthesizes voice alerts using the text converted from the object detection system.
4. **Output** (Voice Alerts): Provides the voice alerts to the blind user conveying the distance of objects detected in front of them.

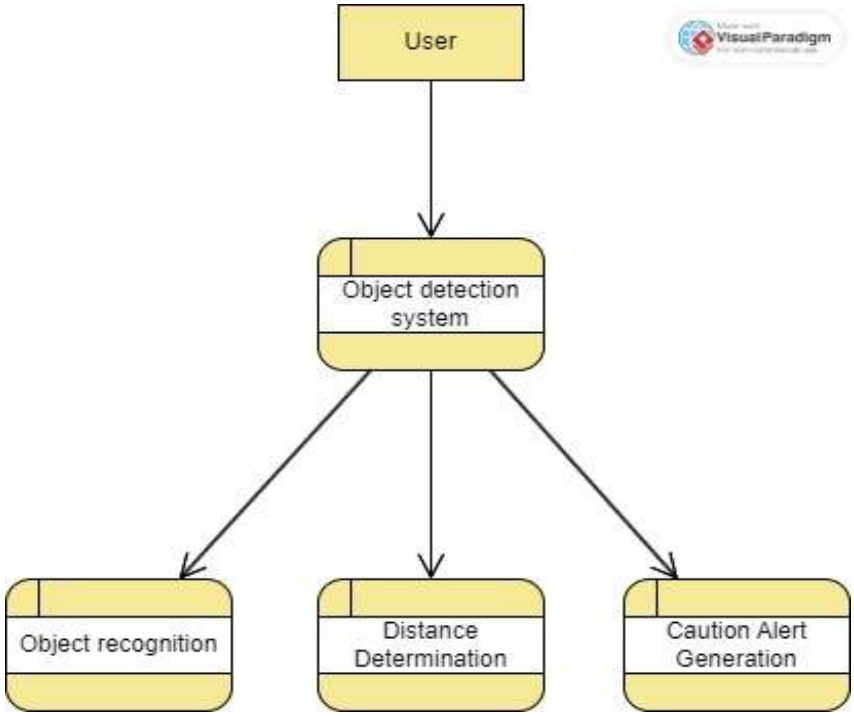


6.6.2 LEVEL1 -- DFD :

1. **Blind User**: Represents the user of the system who is visually impaired and interacts with the system.
2. **Object Detection**: This component is responsible for detecting objects using the YOLO algorithm. It takes input from the environment through sensors or cameras.
3. **Distance Calculation:** Utilizes the focal length and known object dimensions. Estimates the distance of detected objects from the camera.

4. **Voice Alert**: This component synthesizes voice alerts using the text converted from the object detection system.
5. **Output (Voice Alerts):** Provides the voice alerts to the blind user conveying the distance

6.6.3LEVEL2 -- DFD :



1. **Blind User**: Represents the user of the system who is visually impaired and interacts with the system.
2. **Object Detection**: This component is responsible for detecting objects using the YOLO algorithm. It takes input from the environment through sensors or cameras.
3. **Focal Length calculation:** Uses the known width of a reference object and its actual dimensions. Calculates the focal length of the camera lens using the width of the detected object and its known dimensions.
4. **Distance Calculation:** Utilizes the focal length and known object dimensions. Estimates the distance of detected objects from the camera.

5.**Voice Alert**: This component synthesizes voice alerts using the text converted from the object detection system.

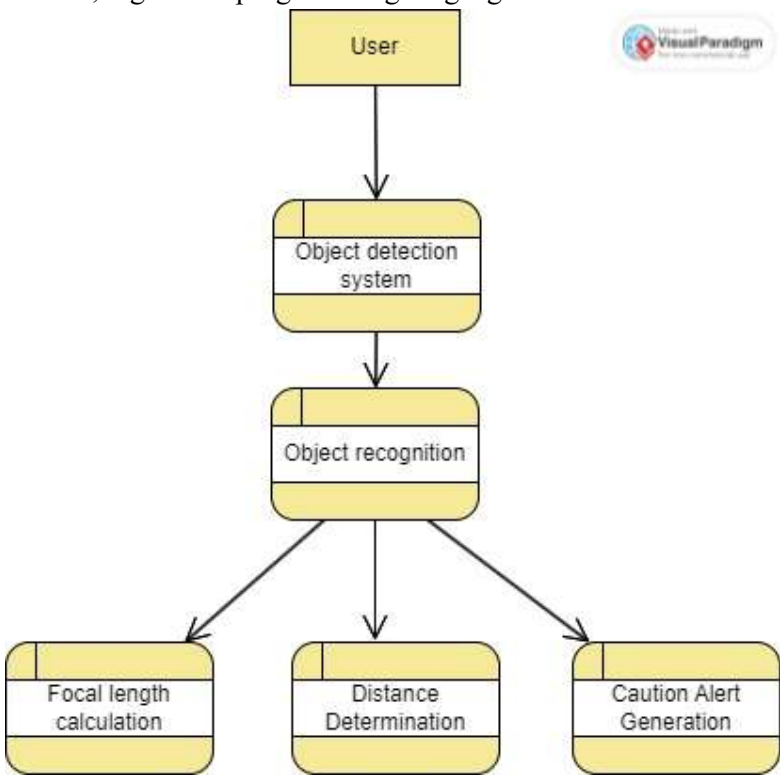
CHAPTER-7

SYSTEM IMPLEMENTATION

7.1 Implementation Tools

Python:

Python is a powerful, high-level programming language that is renowned for being easy to learn and understand. With libraries and frameworks for numerous applications, including web development, scientific computing, and artificial intelligence, it has a sizable and vibrant community.



Visual Studio Code:

Visual studio code is the platform for wring the source code

7.2 Frameworks

OpenCV:

OpenCV (Open Source Computer Vision) is a machine learning and computer vision library that was first made available in 2000. It is intended to make it simpler to create real-time vision applications and to offer a common architecture for computer vision applications.

Over 2,500 highly optimised algorithms for tasks like object detection, image processing, and feature extraction are included in OpenCV, which is written in C++ and Python. Additionally, it works with many different operating systems, including Windows, Linux, macOS, and Android

Facial recognition, motion detection, object tracking, and augmented reality are some of OpenCV's most used capabilities. For more complex image processing and machine learning tasks, OpenCV can also be used in conjunction with other libraries, such as NumPy and SciPy.

In general, OpenCV is a robust and adaptable library for computer vision applications that has grown to be a well-liked tool in the machine learning and artificial intelligence fields.

Flask:

The initial version of the well-known Python web framework Flask appeared in 2010. Its lightweight and modular design makes it simple to construct web apps fast and effectively.

Flask offers a straightforward and adaptable method of handling online requests and responses because it is built on the Werkzeug toolkit and the Jinja2 template engine. Additionally, it supports a wide variety of extensions for jobs like testing, database integration, and user authentication.

Built-in development server, support for secure cookies and sessions, and support for RESTful request handling are some of the features of Flask. Additionally, Flask has a sizable and vibrant developer community, which has sparked the development of numerous tools and extensions.

In general, Flask is a well-liked and adaptable Python web framework that works well for creating small to medium-sized online applications. It is simple to understand and use, which makes it a wonderful option for new users, and it is a favourite among seasoned developers thanks to its versatility and adaptability.

Trained Objects List:

We have trained only few objects and objects are listed below. This will be extended to N number of objects.

```
person
bicycle
car
motorbike
aeroplane
bus
train
truck
boat
traffic light
fire hydrant
stop sign
parking meter
bench
bird
cat
dog
horse
sheep
cow
elephant
bear
zebra
giraffe
backpack
umbrella
handbag
tie
suitcase
frisbee
skis
snowboard
sports ball
kite
baseball bat
baseball glove
skateboard
surfboard
tennis racket
bottle
wine glass
cup
fork
knife
spoon
bowl
banana
```

```
    apple
    sandwich
    orange
    broccoli
    carrot
    hot dog
    pizza
    donut
    cake
    chair
    sofa
    pottedplant
    bed
    diningtable
    toilet
    tvmonitor
    laptop
    mouse
    remote
    keyboard
    cell phone
    microwave
    oven
    toaster
    sink
    refrigerator
    book
    clock
    vase
    scissors
    teddy bear
    hair drier
    toothbrush
```

7.3Source code :

Distance_estimation.py:

```
def SWAB():
    import cv2 as cv
    import numpy as np
    # Distance constants
    KNOWN_DISTANCE = 45 #INCHES
    PERSON_WIDTH = 16 #INCHES
    MOBILE_WIDTH = 2.5 #INCHES
    BOTTLE_WIDTH=3.0
    BOOK_WIDTH=6.5
    CHAIR_WIDTH=17
    LAPTOP_WIDTH=14
    APPLE_WIDTH=2
    MOTOR_BIKE_WIDTH=60
    CUP_WIDTH=4
    VASE_WIDTH=6
    POTTEDPLANT_WIDTH=12
    SPORTS_BALL_WIDTH=9
    CAR_WIDTH=196
    DOG_WIDTH=13
    STOPSIGN_WIDTH=30
    FIREHYDRANT_WIDTH=4.5
```

TRAFFICLIGHT_WIDTH=9.5

KEYBOARD_WIDTH=66

REMOTE_WIDTH=2

MOUSE_WIDTH=5

BED_WIDTH=59

BICYCLE_WIDTH=43

BUS_WIDTH=90

TRAIN_WIDTH=60

TRUCK_WIDTH=80.4

CAT_WIDTH=5.9

CAR_WIDTH=70

BUS_WIDTH=96

BENCH_WIDTH=45

BOAT_WIDTH=216

PARKING_METER_WIDTH=20

HORSE_WIDTH=56

SHEEP_WIDTH=17

COW_WIDTH=23

ELEPHANT_WIDTH=82

BEAR_WIDTH=52

ZEBRA_WIDTH=64

GIRAFFE_WIDTH=36

Object detector constant

```
CONFIDENCE_THRESHOLD = 0.4

NMS_THRESHOLD = 0.3

# colours for object detected

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

GREEN = (0,255,0)

BLACK = (0,0,0)

 # 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()]

#  setttng up opencv net

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

yoloNet.setPreferableBackend(cv.dnn.DNN_BACKEND_CUDA)

yoloNet.setPreferableTarget(cv.dnn.DNN_TARGET_CUDA_FP16)

model = cv.dnn_DetectionModel(yoloNet)

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

# object detector funciton /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
    color= COLORS[int(classid) % len(COLORS)]
    label = "%s : %f" % (class_names[classid], 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)
    # 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], box[2], (box[0], box[1]-2)]
 elif classid ==67:#mobile
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==39:#bottle
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==73:#book
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==56:#chair
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==63:#laptop\
```

```
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==47:#apple
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==3:#BIKE
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==41:#cup
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==75:#vase
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==58:#pottedplant
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==32:#sports ball
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==16:#DOG
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==11:#stopsign
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==10:#firehydrant
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==9:#trafficlight
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
 elif classid ==66:#keyboard
    data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
```

```
elif classid ==65:#remote
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==64:#mouse
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==1:#bicycle
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==5:#bus
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==7:#truck
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==15:#cat
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==2:#car
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==5:#bus
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==13:#bench
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==8:#boat
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
elif classid ==12:#parking meter
   data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
```

```
,          elif classid ==18:#sheep
              data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
           elif classid ==19:#caw
              data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
           elif classid ==20:#elephant
              data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
           elif classid ==21:#bear
              data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
           elif classid ==22:#zebra
              data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
           elif classid ==23:#giraffe
              data_list.append([class_names[classid], box[2], (box[0], box[1]-2)])
        return data_list
    def focal_length_finder (measured_distance, real_width, width_in_rf):
      focal_length = (width_in_rf * measured_distance) / real_width
      return focal_length
    # distance finder function
    def distance_finder(focal_length, real_object_width, width_in_frmae):
      distance = (real_object_width * focal_length) / width_in_frmae
      return distance
   # reading the reference image from dir
   ref_person = cv.imread('ReferenceImages/persons.png')
   ref_mobile = cv.imread('ReferenceImages/image4.png')
```

```
ref_bottle = cv.imread('ReferenceImages/bottle2.png')

ref_book = cv.imread('ReferenceImages/books.png')

ref_chair= cv.imread('ReferenceImages/chairs.png')

ref_laptop= cv.imread('ReferenceImages/laptops.png')

ref_apple= cv.imread('ReferenceImages/apples.png')

ref_motorbike= cv.imread('ReferenceImages/bike.png')

ref_cup= cv.imread('ReferenceImages/cups.png')

ref_vase= cv.imread('ReferenceImages/vase.png')

ref_pottedplant= cv.imread('ReferenceImages/pottedplant.png')

ref_sports_ball= cv.imread('ReferenceImages/sports ball.png')

ref_car= cv.imread('ReferenceImages/car.png')

ref_dog= cv.imread('ReferenceImages/dogs.png')

ref_stopsign= cv.imread('ReferenceImages/stopsign.png')

ref_firehydrant= cv.imread('ReferenceImages/fire-hydrant.png')

ref_trafficlight= cv.imread('ReferenceImages/trafficlight.png')

ref_keyboard= cv.imread('ReferenceImages/keyboard.png')

ref_remote= cv.imread('ReferenceImages/remote.png')

ref_bicycle= cv.imread('ReferenceImages/bicycle.png')

ref_bus= cv.imread('ReferenceImages/bus.png')

ref_truck= cv.imread('ReferenceImages/truck.png')

ref_cat= cv.imread('ReferenceImages/cat.png')

ref_bus= cv.imread('ReferenceImages/buses.png')

ref_bench= cv.imread('ReferenceImages/bench.png')
```

```
ref_boat= cv.imread('ReferenceImages/boat.png')

ref_parking_meter= cv.imread('ReferenceImages/parking_meter.png')

ref_sheep= cv.imread('ReferenceImages/sheep.png')

ref_cow= cv.imread('ReferenceImages/cow.png')

ref_elephant= cv.imread('ReferenceImages/elephant.png')

ref_bear= cv.imread('ReferenceImages/bear.png')

ref_zebra= cv.imread('ReferenceImages/zebra.png')

ref_giraffe= cv.imread('ReferenceImages/giraffe.png')

person_data = object_detector(ref_person)

person_width_in_rf = person_data[0][1]


mobile_data = object_detector(ref_mobile)

mobile_width_in_rf = mobile_data[0][1]


bottle_data = object_detector(ref_bottle)

bottle_width_in_rf = bottle_data[0][1]


book_data = object_detector(ref_book)

book_width_in_rf = book_data[0][1]


chair_data = object_detector(ref_chair)

chair_width_in_rf = chair_data[0][1]
```

```
laptop_data = object_detector(ref_laptop)

laptop_width_in_rf = laptop_data[0][1]


apple_data = object_detector(ref_apple)

apple_width_in_rf = apple_data[0][1]


teddy_data = object_detector(ref_teddy)

teddy_width_in_rf = teddy_data[0][1]


motorbike_data = object_detector(ref_motorbike)

motorbike_width_in_rf = motorbike_data[0][1]


cup_data = object_detector(ref_cup)

cup_width_in_rf = cup_data[0][1]


vase_data = object_detector(ref_vase)

vase_width_in_rf = vase_data[0][1]


pottedplant_data = object_detector(ref_pottedplant)

pottedplant_width_in_rf = pottedplant_data[0][1]


sports_ball_data = object_detector(ref_sports_ball)

sports_ball_width_in_rf = sports_ball_data[0][1]
```

```
car_data = object_detector(ref_car)

car_width_in_rf = car_data[0][1]


dog_data = object_detector(ref_dog)

dog_width_in_rf = dog_data[0][1]


stopsign_data = object_detector(ref_stopsign)

stopsign_width_in_rf = stopsign_data[0][1]


firehydrant_data = object_detector(ref_firehydrant)

firehydrant_width_in_rf = firehydrant_data[0][1]


trafficlight_data = object_detector(ref_trafficlight)

trafficlight_width_in_rf = trafficlight_data[0][1]


keyboard_data = object_detector(ref_keyboard)

keyboard_width_in_rf = keyboard_data[0][1]


remote_data = object_detector(ref_remote)

remote_width_in_rf = remote_data[0][1]


bed_data = object_detector(ref_bed)
```

```
bed_width_in_rf = bed_data[0][1]


bicycle_data = object_detector(ref_bicycle)

bicycle_width_in_rf = bicycle_data[0][1]


bus_data = object_detector(ref_bus)

bus_width_in_rf = bus_data[0][1]


truck_data = object_detector(ref_truck)

truck_width_in_rf = truck_data[0][1]


cat_data = object_detector(ref_cat)

cat_width_in_rf = cat_data[0][1]


car_data = object_detector(ref_car)

car_width_in_rf = car_data[0][1]


parking_meter_data = object_detector(ref_parking_meter)

parking_meter_width_in_rf = parking_meter_data[0][1]


sheep_data = object_detector(ref_sheep)

sheep_width_in_rf = sheep_data[0][1]
```

```
cow_data = object_detector(ref_cow)

cow_width_in_rf = cow_data[0][1]


elephant_data = object_detector(ref_elephant)

elephant_width_in_rf = elephant_data[0][1]


bear_data = object_detector(ref_bear)

bear_width_in_rf = bear_data[0][1]


zebra_data = object_detector(ref_zebra)

zebra_width_in_rf = zebra_data[0][1]


giraffe_data = object_detector(ref_giraffe)

print(f"Person width in pixels : {person_width_in_rf} mobile width in pixel: {mobile_width_in_rf}")


# finding focal length

focal_person = focal_length_finder(KNOWN_DISTANCE, PERSON_WIDTH, person_width_in_rf)


focal_mobile = focal_length_finder(KNOWN_DISTANCE, MOBILE_WIDTH, mobile_width_in_rf)

focal_bottle = focal_length_finder(KNOWN_DISTANCE, BOTTLE_WIDTH, bottle_width_in_rf)

focal_book = focal_length_finder(KNOWN_DISTANCE, BOOK_WIDTH, book_width_in_rf)

focal_chair = focal_length_finder(KNOWN_DISTANCE, CHAIR_WIDTH, chair_width_in_rf)

focal_laptop = focal_length_finder(KNOWN_DISTANCE, LAPTOP_WIDTH, laptop_width_in_rf)
```

```
 focal_apple = focal_length_finder(KNOWN_DISTANCE, APPLE_WIDTH, apple_width_in_rf)
 focal_bike=focal_length_finder(KNOWN_DISTANCE,MOTOR_BIKE_WIDTH,motorbike_width_in_rf)
 focal_cup = focal_length_finder(KNOWN_DISTANCE, CUP_WIDTH, cup_width_in_rf)
 focal_vase = focal_length_finder(KNOWN_DISTANCE, VASE_WIDTH, vase_width_in_rf)
 focal_pottedplant = focal_length_finder(KNOWN_DISTANCE, _WIDTH, pottedplant_width_in_rf)
 focal_sports_ball = focal_length_finder(KNOWN_DISTANCE,SPORTS_BALL_WIDTH,
                                                                    sports_ball_width_in_rf)
 focal_car = focal_length_finder(KNOWN_DISTANCE, CAR_WIDTH, car_width_in_rf)
 focal_dog = focal_length_finder(KNOWN_DISTANCE, DOG_WIDTH, dog_width_in_rf)
 focal_stopsign = focal_length_finder(KNOWN_DISTANCE,STOPSIGN_WIDTH, stopsign_width_in_rf)
focal_firehydrant = focal_length_finder(KNOWN_DISTANCE,FIREHYDRANT_WIDTH,
                                                                    firehydrant_width_in_rf)
focal_trafficlight=focal_length_finder(KNOWN_DISTANCE,TRAFFICLIGHT_WIDTH,
                                                                    trafficlight_width_in_rf)
focal_keyboar =  focal_length_finder(KNOWN_DISTANCE,KEYBOARD_WIDTH,
                                                                    keyboard_width_in_rf)
  focal_remote= focal_length_finder(KNOWN_DISTANCE, REMOTE_WIDTH, remote_width_in_rf)
  focal_bed= focal_length_finder(KNOWN_DISTANCE, BED_WIDTH, bed_width_in_rf)
  focal_bicycle= focal_length_finder(KNOWN_DISTANCE, BICYCLE_WIDTH, bicycle_width_in_rf)
  focal_bus= focal_length_finder(KNOWN_DISTANCE, BUS_WIDTH, bus_width_in_rf)
  focal_sheep=focal_length_finder(KNOWN_DISTANCE,SHEEP_WIDTH, sheep_width_in_rf)
  focal_cow= focal_length_finder(KNOWN_DISTANCE, COW_WIDTH, cow_width_in_rf)
  focal_elephant=focal_length_finder(KNOWN_DISTANCE,ELEPHANT_WIDTH,elephant_width_in_rf)
```

```
focal_bear= focal_length_finder(KNOWN_DISTANCE, BEAR_WIDTH, bear_width_in_rf)

focal_zebra=focal_length_finder(KNOWN_DISTANCE, ZEBRA_WIDTH, zebra_width_in_rf)

focal_giraffe=focal_length_finder(KNOWN_DISTANCE, GIRAFFE_WIDTH, giraffe_width_in_rf)

cap = cv.VideoCapture(0)

while True:

    ret, frame = cap.read()

    data = object_detector(frame)

    for d in data:

        if d[0] =='person':

            p="person"

            distance = distance_finder(focal_person, PERSON_WIDTH, d[1])

            x, y = d[2]

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

            p="cell phone"

            distance = distance_finder (focal_mobile, MOBILE_WIDTH, d[1])

            x, y = d[2]

        elif d[0] =='bottle':

            p="bottle"

            distance = distance_finder (focal_bottle, BOTTLE_WIDTH, d[1])

            x, y = d[2]

        elif d[0] =='book':

            p="book"

            distance = distance_finder (focal_book, BOOK_WIDTH, d[1])
```

```
    x, y = d[2]
  elif d[0] =='chair':
    p="chair"
    distance = distance_finder (focal_chair, CHAIR_WIDTH, d[1])
    x, y = d[2]
  elif d[0] =='laptop':
    p="laptop"
    distance = distance_finder (focal_laptop, LAPTOP_WIDTH, d[1])
    x, y = d[2]
  elif d[0] =='apple':
    p="apple"
    distance = distance_finder (focal_apple, APPLE_WIDTH, d[1])
    x, y = d[2]
  elif d[0] =='motorbike':
    p="motorbike"
distance = distance_finder (focal_bike, MOTOR_BIKE_WIDTH, d[1])
    x, y = d[2]
  elif d[0] =='cup':
    p="cup"
    distance = distance_finder (focal_cup, CUP_WIDTH, d[1])
    x, y = d[2]
```

```
elif d[0] =='vase':
            p="vase"
            distance = distance_finder (focal_vase, VASE_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='pottedplant':
            p="potted plant"
            distance = distance_finder (focal_pottedplant, POTTEDPLANT_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='car':
            p='car'
            distance = distance_finder (focal_car, CAR_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='dog':
            p="dog"
            distance = distance_finder (focal_dog, DOG_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='stopsign':
            p="stopsign"
            distance = distance_finder (focal_stopsign, STOPSIGN_WIDTH, d[1])
            x, y = d[2]
        elif d[0] =='firehydrant':
            p="firehydrant"
            distance = distance_finder (focal_firehydrant, FIREHYDRANT_WIDTH, d[1])
```

```
        x, y = d[2]
    elif d[0] =='trafficlight':
        p="trafficlight"
        distance = distance_finder (focal_trafficlight, TRAFFICLIGHT_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='keyboard':
        p="keyboard"
        distance = distance_finder (focal_keyboard, KEYBOARD_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='remote':
        p="remote"
        distance = distance_finder (focal_remote, REMOTE_WIDTH, d[1])
    elif d[0] =='bicycle':
        p="bicycle"
        distance = distance_finder (focal_bicycle, BICYCLE_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='bus':
        p="bus"
        distance = distance_finder (focal_bus, BUS_WIDTH, d[1])
        x, y = d[2]
# elif d[0] =='train':
    #    p="train"
#     distance = distance_finder (focal_train, TRAIN_WIDTH, d[1])
```

```
#     x, y = d[2]
    elif d[0] =='truck':
        p="truck"
        distance = distance_finder (focal_truck, TRUCK_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='cat':
        p="cat"
        distance = distance_finder (focal_cat, CAT_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='parking meter':
        p="parking meter"
        distance = distance_finder (focal_parking_meter, PARKING_METER_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='sheep':
        p="sheep"
        distance = distance_finder (focal_sheep, SHEEP_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='cow':
        p="cow"
        distance = distance_finder (focal_cow, COW_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='elephant':
        p="elephant"
```

```
            distance=distance_finder(focal_elephant,ELEPHANT_WIDTH,d[1])
        x, y = d[2]
    elif d[0] =='bear':
        p="bear"
        distance = distance_finder (focal_bear, BEAR_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='zebra':
        p="zebra"
        distance = distance_finder (focal_zebra, ZEBRA_WIDTH, d[1])
        x, y = d[2]
    elif d[0] =='giraffe':
        distance = distance_finder (focal_giraffe, GIRAFFE_WIDTH,d[1])
        x, y = d[2]
    cv.rectangle(frame, (x, y-3), (x+150, y+23),BLACK,-1 )
    cv.putText(frame, f'Dis: {round(distance,2)} inch', (x+5,y+13), FONTS, 0.48, GREEN, 2)
  cv.imshow('frame',frame)
  key = cv.waitKey(1)
  if key ==ord('q'):
    break
  output= p + "infront of you in" + str(round(distance)) +"inches"
# import speech_recognition as sr
  import pyttsx3
  abc=pyttsx3.init()
```

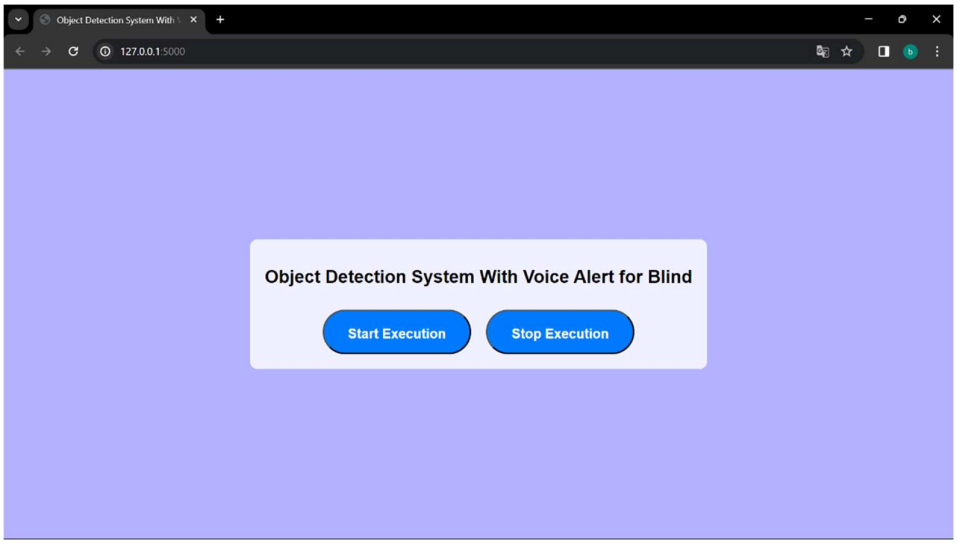
```
        speech_converting_sentence=output
        voice=abc.getProperty('voices')
        abc.setProperty('rate',120)
        abc.setProperty('volume',2.0)
        abc.setProperty('voice',voice[1].id)
        abc.say("there is a "+speech_converting_sentence)
        abc.runAndWait()
    cv.destroyAllWindows()
    cap.release()
```

Code for Flask App

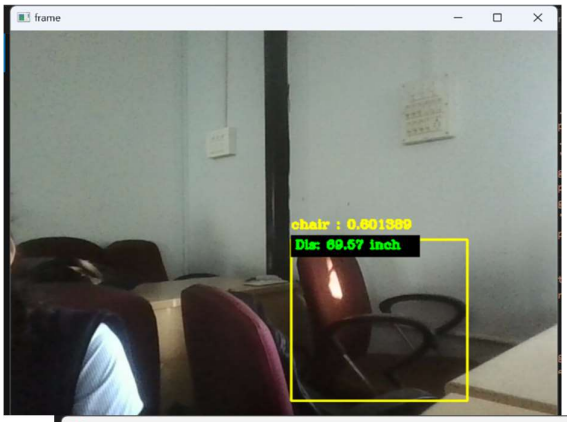
```
from flask import Flask, render_template
import DistanceEstimation
import threading
import os
app = Flask(__name__)
t1 = None  # Thread for DistanceEstimation.SWAB()
running = False  # Variable to track if DistanceEstimation.SWAB() is running
def run_swab():
    global running
    DistanceEstimation.SWAB()
```

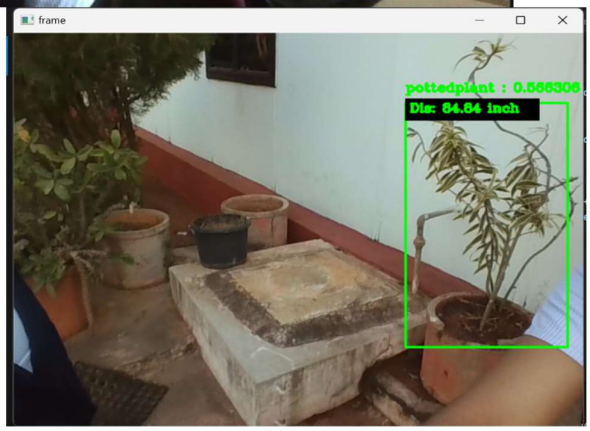
```
@app.route("/")
def home():
    return render_template("Main.html")
@app.route("/start_execution")
def start_execution():
    global t1, running
    if not running:
        t1 = threading.Thread(target=run_swab)
        t1.start()
        running = True
        print("Execution started successfully")
    return "Execution started successfully"
@app.route("/stop_execution")
def stop_execution():
    global running
    if running:
        running = False
        os._exit(0)  # Terminate the script
    return "Execution stopped successfully"
if __name__ == "__main__":
    app.run(debug=True)
```

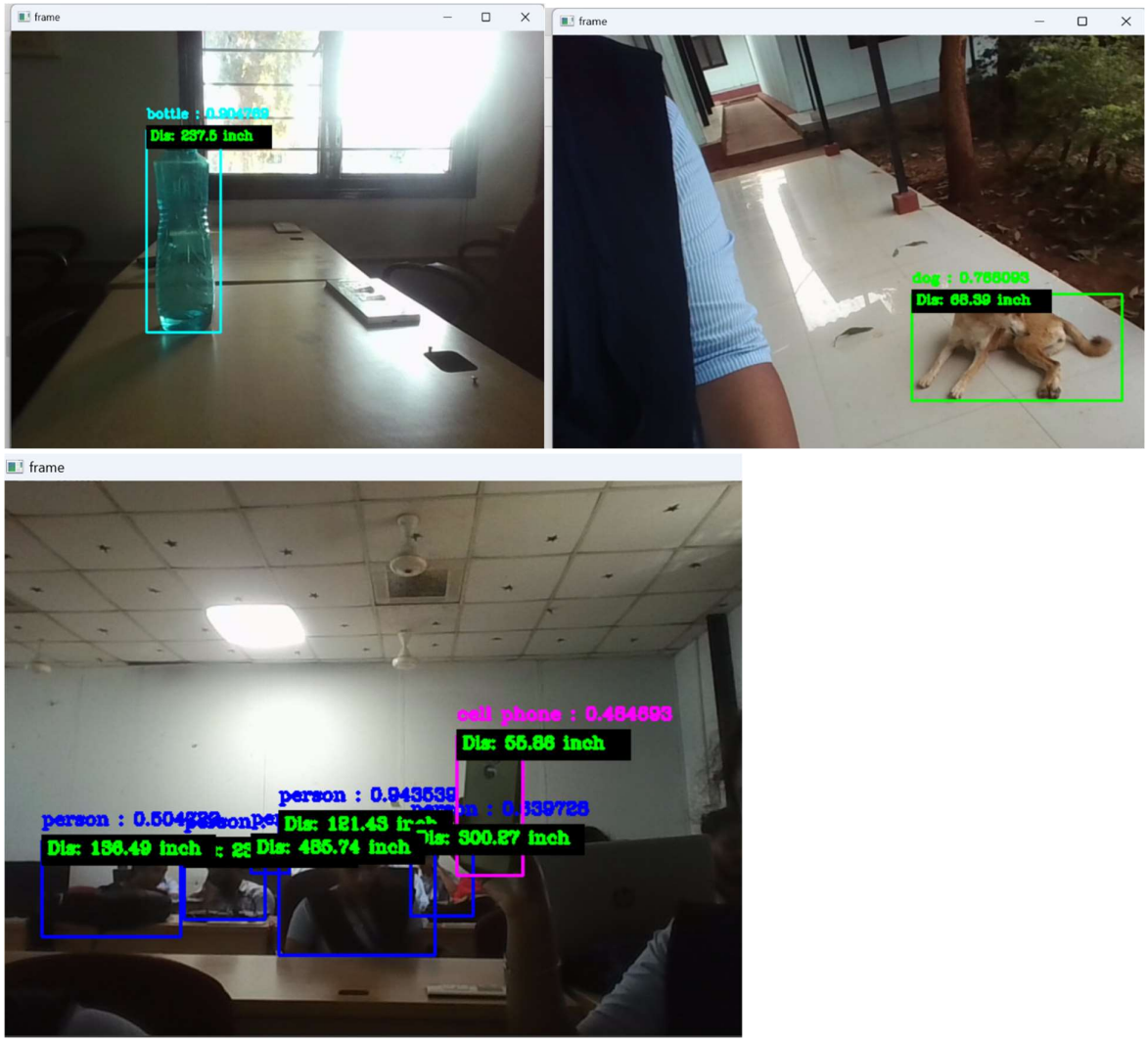
Interface:

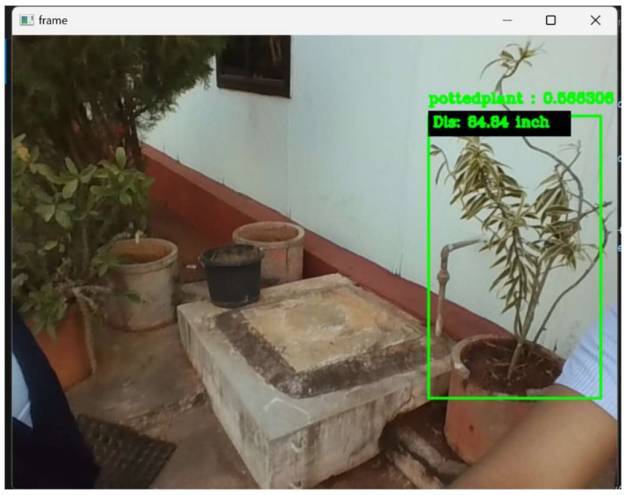


Output:






frame
bottle :
Dis: 237.5 inch
frame
dog : 0.788093
Dis: 66.39 inch
frame
cell phone : 0.484693
Dis: 55.86 inch
person : 0.943535
Dis: 121.43 inch
Dis: 300.27 inch
person : 0.504732
Dis: 136.49 inch
Dis: 485.74 inch



CHAPTER-8

SYSTEM TESTING

8.1 Introduction

- **Purpose of Testing**: The primary goal of testing is to uncover mistakes or errors within a product.

- **Systematic Examination**: Testing involves a methodical approach to search for faults or weaknesses across all aspects of a product, including components, assemblies, and the final product.
- **Performance Evaluation**: It provides a means to assess the functionality and effectiveness of various parts of the product, ensuring they meet both requirements and customer expectations.
- **Preventing Failure**: The aim of testing is to exercise the program rigorously to ensure that it operates as intended and does not fail in an unacceptable manner.
- **Types of Testing**: There are different types of testing methods, each designed to address specific testing requirements and objectives.

8.2 Types of testing

8.2.1 Unit testing :

Unit testing focuses on validating the internal logic of individual components within

the application to ensure they function correctly.

- **Validation of Internal Logic**: It verifies that the internal business logic of the application operates safely and effectively, ensuring that inputs result in valid outputs.
- **Coverage of Decision Branches and Code Flow**: Unit testing involves validating all decision branches and internal code flow within the individual units of the application.
- **Post-Completion Testing**: Unit testing occurs after the completion of each individual unit before integration into larger modules or the application as a whole.
- **Structural Testing Approach**: Unit testing is considered a structural testing approach, relying on knowledge of the internal structure of the units being tested and involving invasive techniques.

- **Integration into Common Tests**: Unit tests are integrated into overall testing processes at the component level and focus on specific business processes, applications, or system configurations.
- **Assurance of Process Compliance**: Unit tests ensure that each specified process within a business function adheres to documented requirements, with clearly defined inputs and expected outputs.

8.2.2 Integration testing :

Integration testing aims to assess the functionality of integrated software components,

ensuring they operate cohesively as a unified application.

- **Event-Driven Testing**: Integration testing is event-driven, focusing more on the overall outcome of screens or fields rather than individual components.
- **Verification of Integration**: Its primary purpose is to verify that although components may have been individually tested successfully, when integrated, they function correctly and consistently.
- **Identification of Component Issues**: Integration testing is primarily concerned with identifying issues that arise from the interaction and integration of different software components.
- **Ensuring Cohesive Operation**: The main goal of integration testing is to ensure that when components are combined, they work together seamlessly to achieve the desired functionality.

8.2.3 Functional Testing :

Functional testing ensures that the functionalities developed within the software meet the

requirements specified by business and technical documentation, as well as user manuals.

- **Testing Criteria**: Functional testing operates based on several criteria:
  - **Valid Input**: Acceptance of known categories of valid input.
  - **Invalid Input**: Rejection of known categories of invalid input.
  - **Exercise of Functions**: Testing exercises known functionalities of the software.

- **Output Validation**: Testing exercises known classes of software outputs.
- **System/Procedure Performance**: Evaluation of the system's performance when invoked.

❖ **Individual and Team Involvement**: Functional testing involves both individual and team efforts, focusing on specifications, key features, and specific test cases.

❖ **Systematic Coverage**: There is a systematic approach to covering established business process flows, data fields, predefined processes, and sequential procedures during functional testing.

❖ **Test Expansion and Validation**: Before concluding functional testing, additional tests may be identified, and the effectiveness of existing tests is determined to ensure thorough coverage and validation of the software's functionalities.

8.3 System testing

❖ **Objective:**

The objective of system testing is to ensure that the entire integrated object detection system for the blind meets predefined standards. It aims to verify that the system produces expected outcomes reliably, particularly in providing accurate voice alerts about detected objects.

❖ **Testing Approach:**

System testing involves examining the entire system as a cohesive unit, focusing on identified and predictable results. It verifies that the system accurately detects objects and provides voice alerts to the blind.

❖ **Methodology:**

System testing for the object detection system relies on descriptions and flows of the system's processes. It emphasizes predefined system interactions and integration aspects to ensure seamless operation.

❖ **Integration Testing:**

Integration testing ensures that various components of the system are properly configured and integrated to perform object detection and voice alerting accurately. This includes testing the

integration between the YOLO object recognition algorithm and the text-to-speech (TTS) synthesis for voice alerts.

❖ **Emphasis on System Links:**

System testing places emphasis on testing pre-driven system links, ensuring that different components interact correctly and contribute to the overall functionality of the system. This includes testing the communication between the object detection module, voice alert generation, and user interface components.

❖ **Testing Scenarios:**

Verify that the object detection system accurately detects objects in the environment.

Ensure that the distance estimation of detected objects is reliable and consistent.

Test the voice alert generation to confirm that it provides clear and understandable alerts to the blind user.

Validate the integration between the object detection module and the voice alerting system.

Assess the system's performance in different environmental conditions (e.g., lighting variations, object distances).

Test the system's response time to detect and alert about new objects.

Evaluate the system's robustness against false positives and false negatives in object detection.

❖ **Testing Tools and Techniques:**

Use sample input scenarios with known objects and distances for validation.

Employ testing frameworks for automated testing of system components.

Conduct manual testing by simulating real-world scenarios to evaluate user experience and system responsiveness.

8.4 White Box Testing

❖ **Unit Testing:**

Test individual components or units of the system, such as:

- Object detection algorithm (e.g., YOLO algorithm)
- Text-to-speech (TTS) synthesis module
- Integration between object detection and TTS modules
- Design test cases to validate the correctness of functions, procedures, and methods within each component.
- Ensure that each unit operates as expected and produces the correct output.

❖ **Code Coverage Analysis:**

- Measure code coverage to determine how much of the source code is executed during testing.
- Use code coverage tools to identify areas of the code that have not been adequately tested.
- Aim for high code coverage to ensure thorough testing of all functionalities.
- Analyze coverage reports to identify any untested code branches or paths.

8.5 Black Box Testing :

❖ **Functional Testing :**

- Input various scenarios of object detection into the system, simulating different environments and object configurations.
- Verify that the system accurately detects objects and provides voice alerts to the blind user.
- Test the system's ability to recognize various types of objects, distances, and orientations.
- Validate that the voice alerts convey the necessary information about detected objects effectively.

❖ **Usability Testing:**

Evaluate the usability of the system from the perspective of a blind user.

Assess factors such as:

- Ease of use: Determine how easy it is for a blind user to interact with the system.

- Navigation: Test the user interface to ensure intuitive navigation and accessibility.
- Intuitiveness: Evaluate whether the voice alerts are clear, understandable, and provide meaningful information to the user.

❖ **Boundary Testing:**

- Explore the boundaries of the system's input parameters to ensure it behaves as expected under various conditions.
- Test extreme values, such as objects located very close or very far from the user.
- Validate the system's behavior when detecting objects of different sizes, shapes, and colors.
- Verify that the system handles boundary conditions, such as crowded environments or occluded objects, appropriately.

8.3 Levels of Testing:

Unit testing can be performed manually, and comprehensive test cases shall be documented in detail

Test objectives include:

Functionality Testing:

- Verify that each function within the object detection system works correctly.
- Ensure that each page or feature of the system is activated through the specified link or user interaction.

Data Validation Testing:

- Validate that inputs from object detection are in the correct format and meet expected criteria.
- Ensure that duplicate entries or false positives in object detection are appropriately handled and not allowed to affect the system's performance.

8.4 Integration Testing Stratagy:

- The objective of integration testing is to ensure that various components, modules, or subsystems of the object detection system interact seamlessly without encountering interface issues. This process verifies that the speech analysis algorithms, evaluation algorithms, user interface components, and other system elements integrate effectively to produce accurate and reliable object detection and voice alerting results.

- Incremental Integration Approach: Adopt an incremental integration approach where individual components or modules are integrated and tested incrementally. Start with integrating and testing smaller units of the system and gradually move towards integrating larger components. This approach helps identify and address interface defects early in the development process.

Test Results:

All of the scan circumstances recounted above passed efficiently. No defects encountered.

Acceptance Testing :

User Acceptance Testing (UAT) is a critical phase of any project, involving significant participation by end users. Its primary goal is to ensure that the system meets the functional specifications and fulfills the requirements.

Test results :

The entire test cases conducted during the User Acceptance Testing phase passed successfully. No defects were encountered during the testing process.

CHAPTER-9

CONCLUSION

9.1 Conclusion

Several technologies have been created to aid visually impaired persons. One such attempt is that we would wish to make an Integrated Machine Learning System that allows the blind victims to identify and classify real-time objects generating voice feedback and distance. Which also produces warnings whether they are very close or far away from the thing. For visually blind folks, this technology gives voice direction. This technique has been introduced specifically to assist blind individuals. The precision, on the other hand, can be improved. Furthermore, the current system is based on the Machine learning, it can be altered to work with any device that is convenient.

CHAPTER-10
REFERENCES

1] Techniques based on Deep Learning “Electronics and Telecommunications Trends, Vol, 33 No. 4 pp. 23-32, Aug.2018

[2] Joseph redmon, “You Only Look Once: Unified, Real-Time Object Detection,” CVPR 2016.IEEE Computer Society Conference on, pp. 27-30 June 2016.

[3] Aditya Raj, "Model for Object Detection using Computer Vision and Machine Learning for Decision Making," International Journal of Computer Applications, 2019.

[4] Bhumika Gupta, "Study on Object Detection using Open CV Python," International Journal of Computer Applications Foundation of Computer Science, vol. 162, 2017.