OBJECT DETECTION FOR BLIND PEOPLE WITH SPEECH AS OUTPUT

A Project report submitted in partial fulfillment of requirements for the award of degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

by

B.BHOOMIKA REDDY (199X1A0508)

K.KISHORE KUMAR RAJU (199X1A0561)

B.PRIYANKA (199X1A0507)

Under the esteemed guidance of

Sri P. Praveen Yadav
Assistant Professor
Department of C.S.E.

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

2022 – 2023

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

CERTIFICATE

This is to certify that the Project Work entitled 'Object detection for blind people for blind people with speech as output' is a bonafide record of work carried out by

B.BHOOMIKA REDDY (199X1A0508)

K.KISHORE KUMAR RAJU (199X1A0561)

B.PRIYANKA (199X1A0507)

Under my guidance and supervision in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Sri P. Praveen Yadav
Assistant Professor,
Department of C.S.E.,
G. Pulla Reddy Engineering College,
Kurnool.

Dr. N. Kasiviswanath
Professor & Head of the Department,
Department of C.S.E.,
G.Pulla Reddy Engineering College,
Kurnool.

DECLARATION

I hereby declare that the project titled "**OBJECT DETECTION FOR BLIND PEOPLE WITH SPEECH AS OUTPUT**" is an authentic work carried out byus as the student of **G. PULLA REDDY ENGINEERING COLLEGE(Autonomous) Kurnool,** during 2022-23 and has not been submitted elsewhere for the award of any degree or diploma in part or in full to any institute.

B. BHOOMIKA
(199X1A0508)

K. KISHORE KUMAR RAJU
(199X1A0561)

B. PRIYANKA
(199X1A0507)

ACKNOWLEDGEMENT

We wish to express our deep sense of gratitude to our project guide **Sri. P. Praveen Yadav**, Head & Professor of CSE Department, G. Pulla Reddy Engineering College, for his immaculate guidance, constant encouragement and cooperation which have made possible to bring out this project work.

We are grateful to our project incharge **Sri. K. Sudhakar**, Assistant Professor of CSE Department, G.Pulla Reddy Engineering College, for helping us and giving us the required information needed for our project work.

We are thankful to our Head of the Department **Dr. N. Kasiviswanath**, for his whole hearted support and encouragement during the project sessions.

We are grateful to our respected Principal **Dr. B. Sreenivasa Reddy** for providing requisite facilities and helping us in providing such a good environment.

We wish to convey our acknowledgements to all the staff members of the Computer Science and Engineering Department for giving the required information needed for our project work.

Finally, we wish to thank all our friends and wellwishers who have helped us directly or indirectly during the course of this project work.

ABSTRACT

One of the most important organs in human body are eyes. We enjoy the beauty of nature, various types of books, and many other aspects of our lives. The visually impaired people face many difficulties in their day-to-day life. They face many difficulties in object detection and analyzing of their surroundings. While walking on the streets, they face many difficulties in identification and recognizing the objects. These cause them many injuries and accidents etc. So, to put an end to these difficulties we came with an idea of recognizing the objects around them by using object detection and converting them in to voice messages to identify and understand the situations around the person.

This project helps the blind "see" better using Image-to-Text and Text-to-Voice, without any complex hardware. Features are implemented using the model You Only Look Once (YOLO) algorithm that runs through a variation of an extremely complex Convolutional Neural Network architecture. This system tries to transform the visual world into the audio world with the potential to inform blind people objects as well as their spatial locations. Objects detected from the scene are represented by their names and converted to speech. Their spatial locations are encoded into the 2-channel audio with the help of sound simulation.

CONTENTS

1.INTRODUCTION

1.1 INTRODUCTION

Vision is one of the very important senses on which every person in this world depends to interact with the different objects and people in the real world. Normal people views into the surrounding and immediately know which objects are there in the surrounding, how far they are, and how to interact with them. It is not difficult for vision enabled people to do their everyday tasks since they are able to see all the surrounding objects, any other people they come across, any obstacles in the path and hence is easy to interact with surrounding objects. At the same time, visually challenged people have to work hard to interact with real world due to the chore in their daily life. Millions of people live in this world with incapacities of understanding the environment due to visual impairment. Although they can develop alternative approaches to deal with daily routines, they suffer from certain navigation difficulties as well as social awkwardness.

According to a report by the World Health Organization, there are currently 284 million people in the world who are visually impaired, and 39 million people are blind by 2022.According to these statistics, the blind and visually impaired community is increasing yearly. Engaging in day-to-day activities without hazel is an extremely difficult task for a visually impaired/ blind person. It becomes more difficult when it requires traveling through unfamiliar locations without a close companion to assist them along the way. Guide dogs are used in assisting visually impaired persons, but it is not easy to get a trained animal due to the high cost. Furthermore, traveling in familiar environments without help could also be challenging since the dynamic situations along the way cannot be predicted earlier, and responding to those situations in real-time is not possible for a blind per Blind people navigate without a clear visual map about the obstacles in their path. Therefore, it is not possible to take precautions to avoid such obstacles similar to a normal person with good vision. So, to put an end to these difficulties we came with an idea of recognizing the objects around them by using object detection and converting them in to voice messages to identify and understand the situations around the person. The use of technology in finding and recognizing objects is huge because of these rapid increase in the technology like AI, ML and Deep Learning facilitate many tools and libraries for the development new ideas which are useful in contemporary society like smart sticks, navigators etc.

The purpose of this project is to develop a system that assists blind individuals in detecting and recognizing objects in their surroundings. The system will utilize image processing algorithms to identify objects in real-time and provide audio-based output to convey the information to the user. By providing a speech-based output, the system aims to enhance the independence and safety of visually impaired individuals in their daily lives.

1.2 MOTIVATION

Many people suffer from temporary or permanent disabilities.

Here are some scenarios related to our project. As we evolved from one generation to generation our needs and facilities are also evolved with us. The main motivation behind this project is to enhance the independence and safety of blind individuals in their daily activities. By equipping them with a reliable and efficient object detection system, we aim to enable them to navigate their surroundings more confidently and make informed decisions.

1.3 PROBLEM DEFINITION

The problem addressed by this project is the limited ability of visually impaired individuals to detect and recognize objects in their surroundings. Visual impairment poses significant challenges in perceiving and interacting with the environment, leading to difficulties in navigation, object identification, and hazard avoidance. Traditional visual cues are not available to visually impaired individuals, requiring alternative methods to provide them with vital information about their surroundings.

Existing solutions, such as guide dogs and white canes, offer partial assistance but rely heavily on tactile feedback and physical interaction. These methods may not provide comprehensive and real-time object detection capabilities, leaving blind individuals vulnerable to obstacles, hazards, and limitations in their daily activities. Therefore, there is a need for an intelligent system that can accurately detect and recognize objects in the environment and convey this information to visually impaired individuals in a convenient and accessible manner.

The primary challenge lies in developing a reliable and efficient object detection system that can analyze real-time images and identify objects without the need for visual feedback. This requires overcoming various obstacles, such as occlusions, varying lighting conditions, and different object orientations. Additionally, the system must generate audio-based output that is natural, intelligible, and informative to ensure effective communication with visually impaired users.

To address these challenges, the project will focus on designing and implementing a computer vision-based system that can accurately detect and recognize objects in real-time. The system will leverage advanced image processing algorithms and machine learning techniques to analyze captured images and identify objects of interest. The speech synthesis module will convert the detected objects into audio-based output, enabling visually impaired individuals to perceive and understand their environment without relying on visual cues.

By tackling the problem of object detection for blind people, this project aims to empower visually impaired individuals, enhance their independence, and improve their safety and quality of life.

1.4 OBJECTIVE OF THE PROJECT

The objective of the project is to create a system that empowers visually impaired individuals by enabling them to identify and recognize objects in their environment. By leveraging computer vision techniques and advanced image processing algorithms, the system will analyze real-time images captured by a camera and provide audio-based feedback to convey the detected objects to the user.

One of the key features of this system is the use of speech as the output modality. By converting the detected objects into audio information, the system ensures that visually impaired

users can conveniently perceive and comprehend the object's presence and attributes without the need for visual cues. This speech-based output will be synthesized using text-to-speech technologies, which will convert the textual representation of the objects into natural and intelligible speech.

By providing real-time object detection and speech-based output, the system aims to improve the mobility, autonomy, and overall quality of life for visually impaired individuals. It has the potential to assist them in various scenarios, such as navigating unfamiliar environments, identifying and locating objects of interest, and avoiding potential hazards or obstacles. The resulting system will be a user-friendly and robust solution that can be easily deployed on a camera-equipped device, such as a smartphone or a dedicated hardware setup.

1.5 LIMITATIONS OF THE PROJECT

The project aims to address the challenges faced by visually impaired individuals in object detection, so there is the chance that the limitations may arise during the course of the project.

The limitations include

1. Hardware Limitations

The performance of the system may be affected by the hardware used, such as the camera's resolution and quality. Limitations in hardware capabilities may impact the accuracy and speed of object detection, potentially limiting the system's overall effectiveness.

2. Software Limitations

The accuracy and efficiency of the object detection algorithm heavily rely on the software implementation. The limitations of the chosen algorithm, such as its ability to handle the complex scenes, occlusions, or certain object types, may affect the system's detection capabilities.

3. Environmental Limitations

The system's performance may be influenced by environmental factors, including varying lighting conditions, reflections, and occlusions. These factors may impact the accuracy of object detection and the reliability of the speech-based output.

4. User Limitations

The effectiveness of the system may vary depending on individual user factors, such as hearing abilities and language preferences. User-specific characteristics, including the ability to understand synthesized speech, may influence the system's overall usability and accessibility.

5. Generalization to Unseen Objects

The system's ability to detect and recognize objects is limited to the objects present in the training dataset. It may face challenges in accurately detecting objects that are rare, novel, or not included in the training data.

1.6 ORGANIZATION OF THE PROJECT

The report begins with an abstract that provides a brief summary of the project, followed by a table of contents listing the sections and subsections with their corresponding page numbers. The first chapter deals with the introduction of the project, motivation for developing the project, problem definition, objective of the project and limitations of the project. The second chapter deals with the system specifications. The third chapter gives us the literature review which provides an overview of existing research and technologies in the field of object detection and speech synthesis. The fourth chapter gives us the Design and implementations which includes introduction, source code and description of languages used for the project, methods of implementation. The fifth chapter deals with the conclusion which summarizes the key findings, achievements, and contributions of the project. It offers insights into the future prospects of the developed system.

2. SYSTEM SPECIFICATIONS

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements.

NON-FUNCTIONAL REQUIREMENTS

Nonfunctional requirements are the functions offered by the system. It includes time constraints and constraints on the development process and standards. The non-functional requirements are as follows:

- **Speed:** The system should process the given input into output within appropriate time.
- **Ease of use:** The software should be user friendly. Then the customers can use easily, so it doesn't require much training time.
- **Reliability:** The rate of failures should be less then only the system is more reliable
- **Portability**: It should be easy to implement in any system.

2.1 SOFTWARE SPECIFICATIONS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed. The system should be able to interface with the existing system

- The system should be accurate
- The system should be better than the existing system

2.2 HARDWARE SPECIFICATIONS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list, especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture.

The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored.

HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

- **Camera:** A high-resolution camera is required to capture real-time images or video feed of the user's surroundings.
- **Processor:** A modern and powerful processor, such as Intel Core i5 or higher, is recommend.
- **RAM:** Sufficient RAM, typically a minimum of 8 GB, is necessary to ensure smooth processing and execution of the system.
- **Storage:** Adequate storage space is required to store the system's software, libraries, and datasets.

- **Audio Output:** The system should have audio output capabilities, such as built-in speakers or support for headphones, to deliver the synthesized speech output to the visually impaired user.

SOFTWARE REQUIREMENTS

- **Operating System:** The system should be compatible with popular operating systems such as Windows, macOS, or Linux distributions.
- **Development Environment:** Software development tools and frameworks, such as Python, OpenCV, and a text-to-speech library, will be utilized.
- **Object Detection Algorithm:** A suitable Object Detection Algorithm, YOLO, You Only Look Once, algorithm is used.
- **Speech Synthesis Software:** A text-to-speech software or library, such as Google Text-to-Speech API, will be integrated into the system to generate audio-based output.
- **Integrated Development Environment (IDE):** An IDE, such as PyCharm, Visual Studio Code, or Jupiter Notebook, can be used for coding, debugging, and testing.

3. LITERATURE SURVEY

3.1 You Only Look Once: Unified, Real-Time Object Detection

Authors: Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

This paper "You Only Look Once: Unified, Real-Time Object Detection" presents a novel approach to object detection using a unified framework called YOLO (You Only Look Once). Traditional object detection systems process images by dividing them into a grid and applying a separate classifier to each grid cell. In contrast, YOLO performs object detection in a single pass by treating it as a regression problem.

The YOLO algorithm divides the input image into a grid of cells and predicts bounding boxes and class probabilities directly from the image pixels. The network architecture of YOLO consists of a series of convolutional layers followed by fully connected layers. The output of the network is a tensor containing the bounding box coordinates and class probabilities for each grid cell.

Unlike traditional methods that rely on separate components for object localization and classification, YOLO performs both tasks simultaneously. Each grid cell predicts multiple bounding boxes, each associated with a confidence score. The confidence score reflects the accuracy of the bounding box prediction and the likelihood of containing an object. Non-maximum suppression is then applied to remove redundant bounding boxes based on their confidence scores.

Training YOLO involves optimizing a loss function that combines localization and classification errors. The loss function penalizes incorrect predictions of bounding box coordinates and class probabilities. YOLO utilizes a predefined set of anchor boxes, which are representative bounding boxes of different sizes and aspect ratios. The anchor boxes help the model make accurate predictions for objects of various shapes and sizes.

One of the major advantages of YOLO is its ability to achieve real-time object detection. By treating object detection as a regression problem, YOLO is highly efficient and can process images at a high frame rate. The unified architecture enables YOLO to make predictions in a single pass, eliminating the need for region proposal methods or additional post-processing steps.

The paper evaluates the performance of YOLO on various benchmark datasets, including COCO. The results demonstrate that YOLO achieves competitive accuracy while operating in real-time. YOLO outperforms existing state-of-the-art methods in terms of both accuracy and speed.

Although YOLO provides impressive real-time object detection capabilities, it has certain limitations. YOLO struggles with detecting small objects due to the coarse grid cell size. Additionally, YOLO may produce false positives in cluttered scenes. The authors suggest that future work could focus on improving these limitations and exploring variants of the YOLO architecture.

The paper presents YOLO, a unified framework for real-time object detection. By treating object detection as a regression problem, YOLO achieves impressive results in terms of both accuracy and speed. YOLO's unified architecture eliminates the need for complex pipelines and post-processing steps, making it an efficient solution for real-world applications requiring rapid object detection. The paper's findings pave the way for further advancements in real-time computer vision systems.

3.2 Real-Time Object Detection Using YOLO Algorithm for Blind People

Authors: Prof. Pradnya Kasture, Akshay Tangade, Aditya Pole, Aishwarya Kumkar, Yash Jagtap

The paper "Real-Time Object Detection Using YOLO Algorithm for Blind People" focuses on developing a system that utilizes the YOLO (You Only Look Once) algorithm to perform real-time object detection specifically designed to assist blind individuals. The objective is to provide visually impaired users with an efficient and reliable tool for object recognition, aiding their navigation and independence.

Blind individuals encounter significant challenges in perceiving and identifying objects in their surroundings. Traditional methods of object recognition may rely on touch or assistance from others, which can be limiting or impractical in certain situations. Hence, an automated system that performs real-time object detection can significantly improve the daily lives of visually impaired individuals.

The paper focuses on utilizing the YOLO algorithm for real-time object detection. YOLO treats object detection as a regression problem, enabling it to process images in a single pass. The

algorithm divides the input image into a grid, and each grid cell predicts bounding boxes and class probabilities directly. This unified approach allows for efficient and accurate object detection.

The authors discuss the process of training the YOLO algorithm for object detection. This involves collecting and annotating a dataset of images with bounding box annotations and class labels. The YOLO architecture is then trained on this dataset using optimization techniques, such as backpropagation and gradient descent. The training process aims to optimize the network's parameters to accurately detect objects in real-time. The authors evaluate the performance of the real-time object detection system through various metrics. This includes measuring the accuracy of object detection, the system's ability to handle different object categories, and its real-time capabilities. The evaluation may involve testing the system with blind individuals in real-world scenarios to assess its effectiveness and usability.

The paper concludes by emphasizing the significance of real-time object detection using the YOLO algorithm in assisting blind individuals. The proposed system offers an efficient and practical solution to address the challenges faced by visually impaired users in object recognition. By leveraging the YOLO algorithm and providing user-friendly interfaces, the system has the potential to significantly enhance the independence and quality of life for blind individuals.

3.3 EXISTING SYSTEM

Other than navigating sticks, guide dogs currently there are no any software systems that help blind people identifying the objects present in their surroundings.

3.4 PROPOSED SYSTEM

Our proposed system helps the blind “see” better using Image-to-Text and Text-to-Voice, without any complex hardware. Features are implemented using the model You Only Look Once (YOLO) algorithm that runs through a variation of an extremely complex Convolutional Neural Network architecture. This project tries to transform the visual world into the audio world with the potential to inform blind people objects as well as their spatial locations. Objects detected from the scene are represented by their names and converted to speech. Their spatial locations are encoded into the 2-channel audio with the help of sound simulation.

4.DESIGN AND IMPLEMENTATION

4.1 INTRODUCTION

ANACONDA PROMPT

Anaconda Prompt is a command-line interface (CLI) tool that comes bundled with Anaconda, a popular distribution of the Python programming language. It provides a convenient way to interact with Anaconda and its associated package management system, conda. Anaconda Prompt is available on Windows, macOS, and Linux operating systems and offers a set of commands that allow users to manage Python environments, install and update packages, and run Python scripts.

Here are some key features and functionalities of Anaconda Prompt:

- **Python Environment Management:** Anaconda Prompt allows users to create and manage isolated Python environments. These environments provide a controlled and separate space for installing packages and running Python code, ensuring that different projects can have their own set of dependencies without conflicts.
- **Package Management with conda:** conda is a powerful package manager that is integrated with Anaconda Prompt. It enables users to easily install, update, and remove packages from their Python environments. Conda manages dependencies automatically, making it easier to handle complex software stacks and ensuring compatibility between different packages.
- **Installing Packages:** Installing packages using Anaconda Prompt is straightforward. You can use the "conda install" command to install packages from the Anaconda distribution or the "pip install" command to install packages from the Python Package Index (PyPI). Anaconda Prompt automatically resolves dependencies and installs the necessary packages.
- **Running Python Scripts:** Anaconda Prompt allows you to execute Python scripts from the command line. You can navigate to the directory where your script is located and run it using the "python" command followed by the script's filename. Anaconda Prompt

provides a convenient environment for running and testing Python code without the need for a graphical interface.

Overall, Anaconda Prompt is a powerful tool for managing Python environments, installing packages, and running Python scripts. It simplifies the process of setting up and maintaining Python development environments, making it an excellent choice for data scientists, developers, and anyone working with Python.

NUMPY

NumPy is a python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other

words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

TIME

The time module in Python provides a range of functions and classes for working with time-related operations. It is a part of Python's standard library, so it is readily available without any additional installations.

The key features offered by the time module are

- **Time Tracking:** The time module allows you to track time using the time() function, which returns the current time in seconds since the Unix epoch. This functionality is commonly used for measuring time intervals, calculating execution durations, or scheduling tasks.
- **Time Delay:** The sleep (seconds) function suspends the program's execution for the specified number of seconds. This feature is useful when you want to introduce delays or pauses in your code, such as waiting for a specific event or controlling the timing of your program's execution.
- **Time Components**: The localtime([seconds]) function converts a specified time in seconds (or the current time if no argument is provided) to a time struct. The time struct contains several attributes, including year, month, day, hour, minute, second, and more, enabling you to work with different time components individually.
- **Time Formatting:** The time module provides functions like asctime([time]) and strftime(format, [time]) to format time values into human-readable representations. These functions allow you to customize the output format by using special placeholders like %Y for the year, %m for the month, %d for the day, and others.
- **Performance Measurement:** The perf_counter() function returns a high-resolution, system-wide timer that is useful for measuring short durations accurately. It is often employed for profiling code performance or benchmarking different implementations.
- **Time Zones:** The datetime module allows you to handle time zone conversions, perform arithmetic operations with time objects, and work with different time zone databases.

IMUTILS

The imutils library is a popular third-party utility library for image processing tasks in Python. It provides a collection of convenience functions and helper classes that simplify common operations performed on images using OpenCV, another widely-used computer vision library in Python. The imutils library is designed to make the development of computer vision applications more efficient and concise.

The imutils library in Python provides convenient functions and utilities for common image processing tasks. The functions include

- **Image Resizing and Rescaling:** The imutils library includes functions for resizing and rescaling images. The resize() function allows you to easily adjust the dimensions of an image while maintaining the aspect ratio or applying different interpolation methods. The resize() function can be helpful when preparing images for input to machine learning models or adjusting the display size of images.
- **Image Rotation and Flipping:** With imutils, you can easily rotate and flip images using the rotate() and flip() functions, respectively. These functions enable you to perform common geometric transformations on images, such as rotating an image by a specific angle or flipping it horizontally or vertically.
- **Image Translation:** The imutils library provides the translate() function for translating an image by a specified number of pixels in the x and y directions. This function is useful when you need to shift the position of an object within an image or adjust the image's alignment.
- **Image Processing Utilities:** The imutils library offers additional utility functions for image processing tasks. These include functions for automatically detecting edges (auto_canny()), finding the dominant colors in an image (find_function()), computing the center of an object (centroid()), and more. These utilities can assist in various computer vision tasks and simplify common image processing operations.
- **Video Processing Utilities:** In addition to image processing, imutils provides utilities for video processing. It includes functions for resizing video frames (resize()), rotating video frames (rotate()) and performing skimming or skipping frames in a video (skip_frames()). These utilities are helpful when working with video streams or processing video files.

GTTS

The gtts (Google Text-to-Speech) library in Python is a simple and user-friendly interface for using Google's Text-to-Speech API. It allows you to convert text into speech by utilizing Google's powerful and natural language processing capabilities.

- The gtts library enables you to convert text strings into spoken audio files. You can pass a text string to the gTTS() function and generate an audio file that contains the synthesized speech. This is useful for applications such as creating voice-overs, generating audio for presentations, or building voice-enabled applications.
- The gtts library supports a wide range of languages. You can specify the language of the input text by passing the appropriate language code as an argument to the gTTS() function. Google's Text-to-Speech API provides high-quality speech synthesis for many languages, allowing you to generate speech in different languages based on your requirements.
- Once you have created a gTTS object, you can save the generated speech as an audio file. The library provides a save() method that allows you to specify the output file path and filename for the audio file. The audio file is typically saved in MP3 format, but you can also save it in WAV format by providing the tld parameter with the value "com".
- After generating the audio file, you can play it back using any media player that supports the chosen audio format. You can also integrate the gtts library into your Python applications to provide text-to-speech functionality

YOLO (You Only Look Once):

The YOLO algorithm predicts from which class object is from and also helps to give the location of object Following components of bounding boxes are used to get the location of object center, width, height, value c defines which class object is from and in addition pc value is used to get the possibility of object present in the bounding box.

In this algorithm only region containing object is not taken into account but the whole image is divided into grids. This is an algorithm used for object recognition Other algorithms such as variants of cnn does not predict the location of object. But in case of YOLO it useful to predict the location using bounding boxes. Also YOLO can detect multiple objects in a single image unlike

other algorithms which classifies objects into categories only. YOLO uses single neural network on the image and not uses multiple neural network.

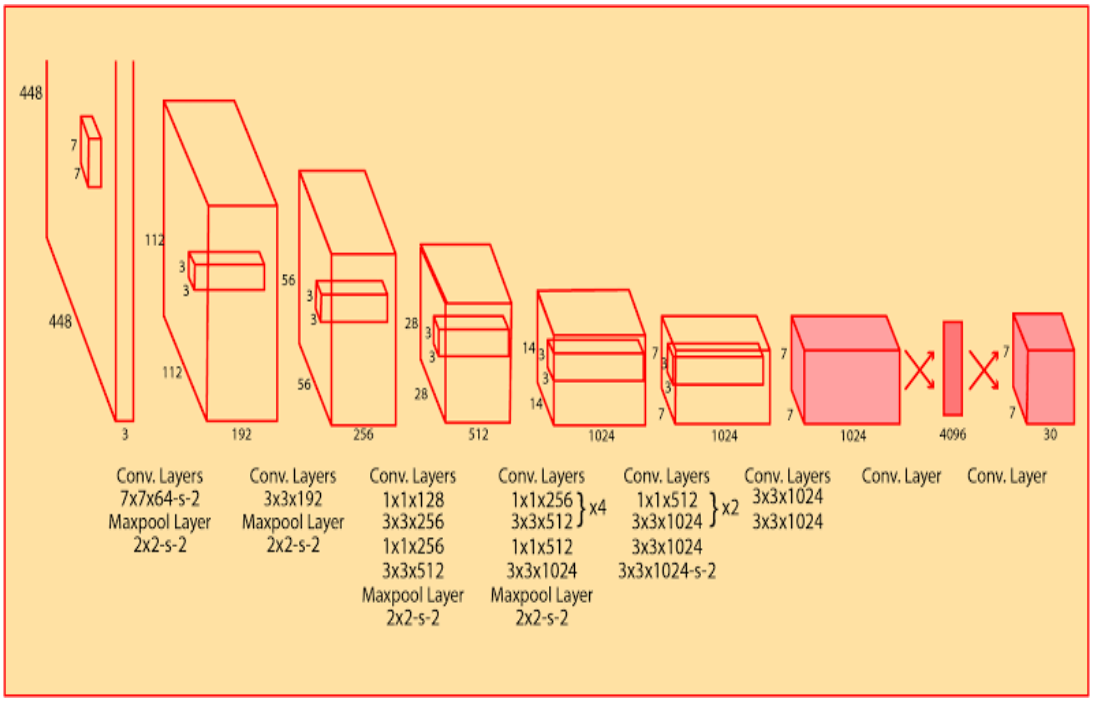

Fig 4.2.1: YOLO Architecture

There are 2 FC layers with 24 convolutional layers in detection that is shown in the fig 4.2.2. That makes alternating convolutional layer. These convolutional layers decrease the feature space of previous layers. In this algorithm, network divides the images into m x m grids which is also called as regions. Further this algorithm decides the bounding boxes and also gives the class probability map which predicts the possibility of object present in certain grid. These bounding boxes contains weight depending on how much object is present in the region. YOLO estimates the type and location of objects using regression inference on the problem of area selection and classification. YOLO processes 45 frames per second so it is incredibly fast and also efficient as it predicts more than one objects from one image.

Bounding Box is used to high- light the object Here, Pc: Probability of object present in region bx, by, bh, bw: Dimension of bounding box c1, c2, c3: Classes of predicted object.

	pc
	bx
	by
	bh
Y=	bw
	c1
	c2
	c3

Fig 4.2.2: Bounding Box

YOLO uses above dimensions to predict the label of object.

- Intersection Over Union (IOU): It determines how predicted box overlaps on determined box. YOLO ensures that these two boxes are overlapped equally.
- Combination of three techniques: Given image shows how three techniques are combined by YOLO.

YOLO will divide the input image in S x S grid, as shown in the fig 4.2.4, then on each grid edge detection filters (Canny Filter) will be applied. With Edges and their connectedness bounding boxes (The rectangular box in which object is enclosed) are generated. Features of each grid in particular bounding box is concatenated and object is predicted using trained model. In YOLO we will train our object detector model on COCO dataset.

The **COCO** dataset consists of 80 labels and 330kimages. Dataset consists labels like Animals, Flowers, People, Traffic signals, buildings, devices like mobile phone, computer, laptop, food items etc.

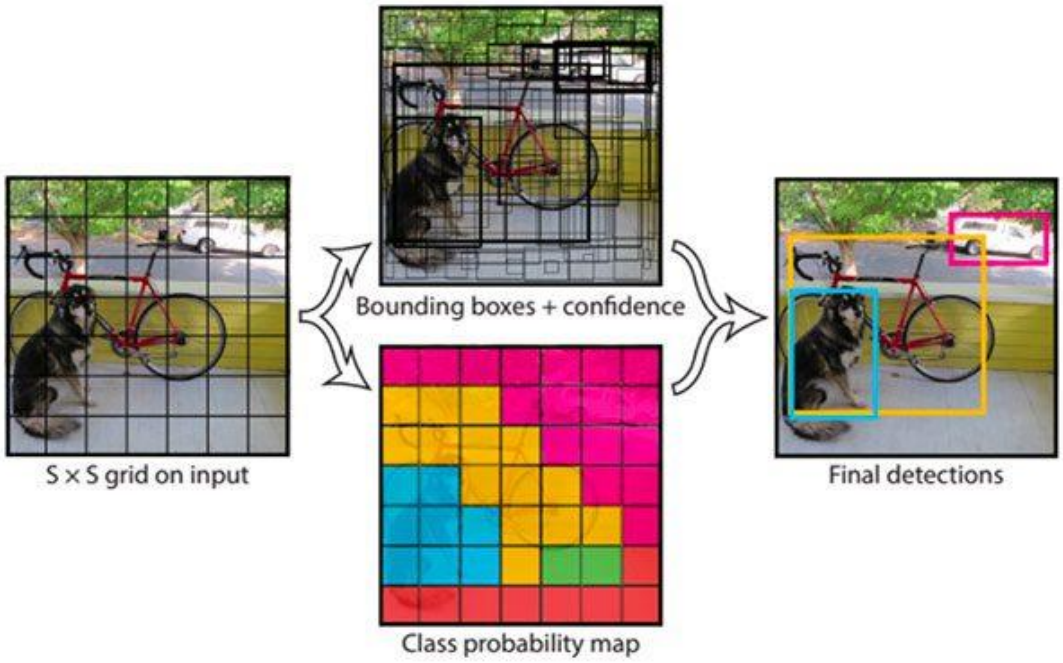

Fig.4.2.3: YOLO Algorithm

ESTIMATING THE POSITION OF OBJECTS:

To estimate the position of image, we need to create the bounding box for every detected object. Then with the height width of particular bounding box, with respect to image frame. 5 values are used to estimate the position of object in bounding box. The first 4 values bx, by, bw, bh shows the position of object. The 5th value is BC: defines how much box contains an object BC is calculated as multiplication of object exist in box and IOU.

As labelled data should be passed forward for training 'y' is considered as label and it can be defined as, Triangle Similarity as object gets closer to the camera its width increases and angle at the camera. Triangle similarity has formula as

$$F= (PxD)/W$$

As continue to move the camera both closer and farther away from the object/marker, applied the triangle similarity to determine the distance of the object to the camera

$$D' = (W \times F) / P$$

GENERATING AUDIO FOR IDENTIFIED OBJECTS

Voice guidance is a function that conveniently provides information to specific users, such as people who are visually impaired, when the system finds the location of the object they want to find the identified class label of detected object and the estimated portion of object is concatenated as text and for voice feedback Google Text-to-Speech API can be used.

It is also available in multiple languages. It is an easy to use as developer can interact with user with multiple languages, applications and devices.

4.2 METHODOLOGY

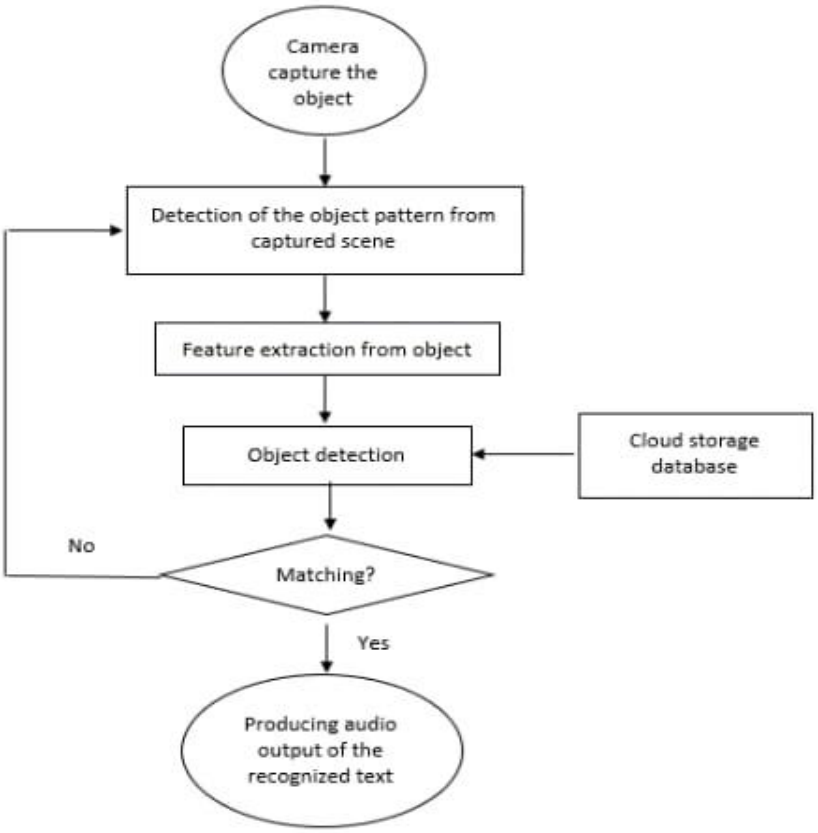

Fig 4.1: Process Flow of the Project

STEPS INVOLVED

1. Image Capture

2. Feature Extraction

3. Object Classification

4. Speech Synthesis

IMAGE CAPTURE

The first step in the working of the Blind-Sight application is Image Capturing. Image Capturing is the process of obtaining images from a video which have to be converted into frames. Live stream is captured with a camera.

FEATURE EXTRACTION

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing.

OBJECT CLASSIFICATION

Object Classification is a classification problem which tends to classify different objects which could flowers, faces, fruits or any object we could imagine. The object classification process includes the following:

- Finding Relative Position
- Class Identification with Confidence
- Text Description

SPEECH SYNTHESIS

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech computer or speech synthesizer, and can be implemented in software or hardware products. It is the process of generating spoken language by machine on the basis of written input. Speech synthesis involves the process of text analysis and letter to sound conversion.

4.3 SOURCE CODE

```
import numpy as np

import time

import cv2

import os

import imutils

import subprocess

from gtts import gTTS

from pydub import AudioSegment

AudioSegment.converter = r"C:\Users\LENOVO\Documents\object-detection-with-voice-feedback\ffmpeg.exe"

AudioSegment.ffprobe = r"C:\Users\LENOVO\Documents\object-detection-with-voice-feedback\ffprobe.exe"

# load the COCO class labels our YOLO model was trained on

LABELS = open("coco.names").read().strip().split("\n")

with open("coco.names", "r") as f:

    classes = [line.strip() for line in f.readlines()]

colors = np.random.uniform(0, 255, size=(len(classes), 3))
```

```
# load our YOLO object detector trained on COCO dataset (80 classes)

print("[INFO] loading YOLO from disk...")

net = cv2.dnn.readNetFromDarknet("yolov3.cfg", "yolov3.weights")

font = cv2.FONT_HERSHEY_PLAIN

# determine only the *output* layer names that we need from YOLO

ln = net.getLayerNames()

ln = [ln[i-1] for i in net.getUnconnectedOutLayers()]

# initialize

cap = cv2.VideoCapture(0)

frame_count = 0

start = time.time()

first = True

frames = []

flag=1

while True:

        frame_count += 1

   # Capture frame-by-frame

        ret, frame = cap.read()

        cv2.imshow("aj", frame)
```

```
frames.append(frame)
if cv2.waitKey(25) & 0xFF == ord('q'):
        break
if ret:
        key = cv2.waitKey(1)
        if frame_count % 60 == 0:
                end = time.time()
                # grab the frame dimensions and convert it to a blob
                (H, W) = frame.shape[:2]
                # construct a blob from the input image and then perform a forward
                # pass of the YOLO object detector, giving us our bounding boxes and
                # associated probabilities
                blob = cv2.dnn.blobFromImage(frame, 1/ 255.0, (416, 416),
                        swapRB=True, crop=False)
                net.setInput(blob)
                layerOutputs = net.forward(ln)
                # initialize our lists of detected bounding boxes, confidences, and
                # class IDs, respectively
                boxes = []
```

```
confidences = []
classIDs = []
centers = []
# loop over each of the layer outputs
for output in layerOutputs:
    # loop over each of the detections
    for detection in output:
        # extract the class ID and confidence (i.e., probability) of
        # the current object detection
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
        # filter out weak predictions by ensuring the detected
        # probability is greater than the minimum probability
        if confidence > 0.5:
        # scale the bounding box coordinates back relative to the
        # size of the image, keeping in mind that YOLO actually
        # returns the center (x, y)-coordinates of the bounding
        # box followed by the boxes' width and height
```

```
                box = detection[0:4] * np.array([W, H, W, H])(centerX, centerY, width, height) = box.astype("int")

                # use the center (x, y)-coordinates to derive the top and

                # and left corner of the bounding box

                x = int(centerX - (width / 2))

                y = int(centerY - (height / 2))

               # update our list of bounding box coordinates, confidences,

                # and class IDs

                boxes.append([x, y, int(width), int(height)])

                confidences.append(float(confidence))

                classIDs.append(classID)

                centers.append((centerX, centerY))

        # apply non-maxima suppression to suppress weak, overlapping bounding

        # boxes

        idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.3)

        for i in range(len(boxes)):

            if i in idxs:

                x, y, w, h = boxes[i]

                label = str(classes[classIDs[i]])
```

```
                    confidence = confidences[i]

                    color = colors[classIDs[i]]

                    cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

                    cv2.putText(frame, label + " " + str(round(confidence, 2)),
(x, y + 30), font, 3, color, 3)

            texts = ["The environment has following objects"]

            # ensure at least one detection exists

            if len(idxs) > 0:

                # loop over the indexes we are keeping

                for i in idxs.flatten():

                    # find positions

                    centerX, centerY = centers[i][0], centers[i][1]

                    if centerX <= W/3:

                        W_pos = "left "

                    elif centerX <= (W/3 * 2):

                        W_pos = "center "

                    else:

                        W_pos = "right "

                    if centerY <= H/3:
```

```
                                H_pos = "top "
                            elif centerY <= (H/3 * 2):
                                H_pos = "mid "
                            else:
                                H_pos = "bottom "
                            texts.append(H_pos + W_pos + LABELS[classIDs[i]])
                            flag=0
                print(texts)
                if (flag==0):
                        description = ', '.join(texts)
                        tts = gTTS(description, lang='en')
                        tts.save(r"C:\Users\LENOVO\Documents\object-detection-with-
voice-feedback\tts.mp3")
                    nts.AudioSegment.from_mp3(r"C:\Users\LENOVO\Documents\object-
detection-with-voice-feedback\tts.mp3")
                        subprocess.call(["ffplay", "-nodisp", "-autoexit", "tts.mp3"])
cap.release()
cv2.destroyAllWindows()
os.remove("tts.mp3")
```

4.3.1 OUTPUT SCREENS

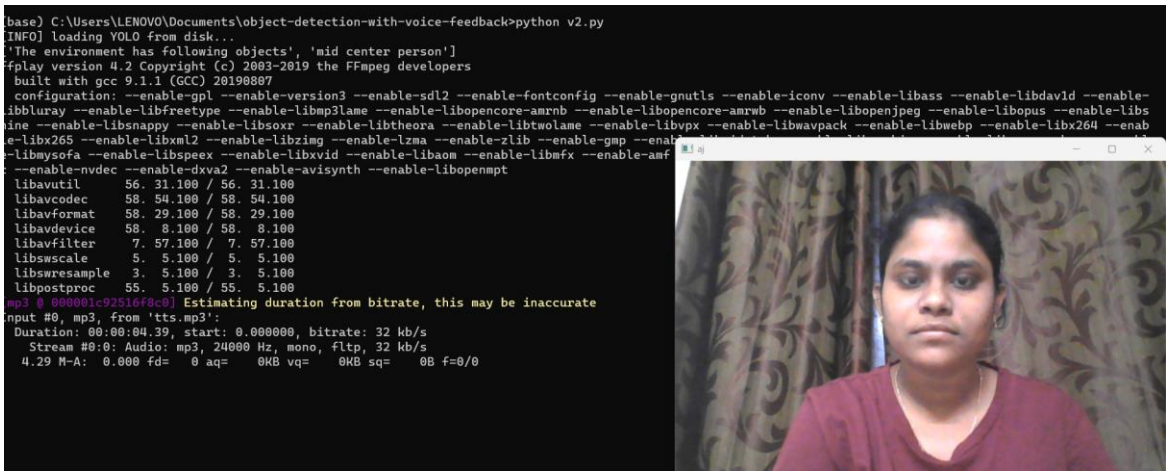

Fig:4.3.1.1 Application identifying that environment has mid center person

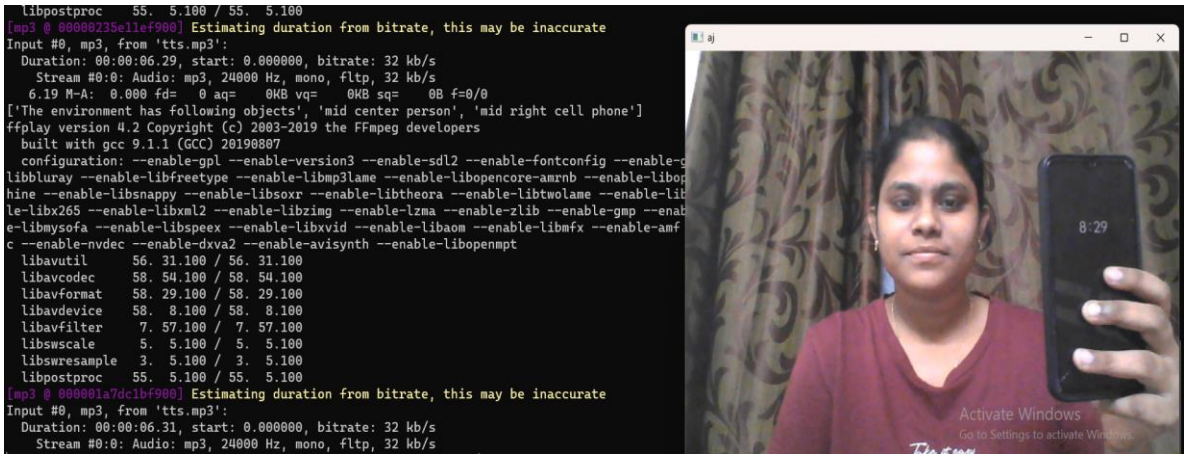

Fig:4.3.1.2 Application identifying that environment has mid center person and mid right cell phone

4.4 TESTING AND VALIDATION

TESTING

The purpose of testing can be quality assurance, verification and validation, or reliability estimation. Testing can be used as a generic metric as well. Correctness testing and reliability testing are two major areas of testing. Software testing is a trade-off between budget, time and quality. The main course of testing is to check for the existence of defects or errors in a program or project or product, based up on some predefined instructions or conditions.

Verification : Checks consistency with the i/p's

The process of determining whether of not the products of a given phase of the software development cycle meets the implementation steps and can be traced to the incoming objectives established during the previous phase. The techniques for verification are testing, inspection.

Validation : Checks Consistency with the User Requirements.

The process of evaluating software at the end of the software development process to ensure compliance with software requirements. The techniques for validation are testing, inspection and reviewing.

Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding software bugs.

Testing can never completely establish the correctness of computer software. Instead, it furnishes a *criticism* or *comparison* that compares the state and behaviour of the product against oracles—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

Over its existence, computer software has continued to grow in complexity and size. Every software product has a target audience. For example, the audience for video game software is

completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it presumably must assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. Software testing is the process of attempting to make this assessment.

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirements gaps, e.g., unrecognized requirements, that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Software faults occur through the following process. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

A frequent cause of software failure is compatibility with another application, a new operating system, or, increasingly, web browser version. In the case of lack of backward compatibility, this can occur (for example...) because the programmers have only considered coding their programs for, or testing the software upon, "the *latest* version of" this-or-that operating

system. The unintended consequence of this fact is that: their latest work might not be fully compatible with earlier mixtures of software/hardware, or it might not be fully compatible with *another* important operating system. In any case, these differences, whatever they might be, may have resulted in (unintended...) software failures, as witnessed by some significant population of computer users.

UNIT TESTING

In computer programming, **unit testing** is a software design and development method where the programmer gains confidence that individual units of source code are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class.

Unit testing can be done by something as simple as stepping through code in a debugger; modern applications include the use of a test framework such as xUnit.

Ideally, each test case is independent from the others; Double objects like stubs, mock or fake objects as well as test harnesses can be used to assist testing a module in isolation. Unit testing is typically done by software developers to ensure that the code other developers have written meets software requirements and behaves as the developer intended.

The first test in the development process is the unit test. The source code is normally divided into modules, which in turn are divided into smaller units called units. These units have specific behavior. The test done on these units of code is called unit test. Unit test depends upon the language on which the project is developed. Unit tests ensure that each unique path of the project performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.

Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

On the other hand, ordinary narrative documentation is more susceptible to drifting from the implementation of the program and will thus become outdated (e.g. design changes, feature creep, relaxed practices in keeping documents up-to-date).

When software is developed using a test-driven approach, the Unit-Test may take the place of formal design. Each unit test can be seen as a design element specifying classes, methods, and observable behaviour. The following Java example will help illustrate this point.

Here is a test class that specifies a number of elements of the implementation. First, that there must be an interface called Adder, and an implementing class with a zero-argument constructor called AdderImpl. It goes on to assert that the Adder interface should have a method called add, with two integer parameters, which returns another integer. It also specifies the behaviour of this method for a small range of values.

LIMITATIONS

Testing cannot be expected to catch every error in the program - it is impossible to evaluate all execution paths for all but the most trivial programs. The same is true for unit testing. Additionally, by definition unit testing only tests the functionality of the units themselves. Therefore, it will not catch integration errors, or broader system level errors (such as functions performed across multiple units, or non-functional test areas such as performance). Unit testing is more effective if it is used in conjunction with other software testing activities. Like all forms of software testing, unit tests can only show the presence of errors; it cannot show the absence of errors.

Software testing is a combinatorial problem. For example, every boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

To obtain the intended benefits from unit testing, a rigorous sense of discipline is needed throughout the software development process. It is essential to keep careful records not only of the tests that have been performed, but also of all changes that have been made to the source code of this or any other unit in the software. Use of a version control system is essential. If a later version of the unit fails a particular test that it had previously passed, the version-control software can provide a list of the source code changes (if any) that have been applied to the unit since that time.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily and addressed immediately. If such a process is not implemented and ingrained into the team's workflow, the application will evolve out of sync with the unit test suite — increasing false positives and reducing the effectiveness of the test suite.

ACCEPTANCE TESTING

Acceptance testing is formal testing conducted to determine whether a system satisfies its acceptance criteria and thus whether the customer should accept the system.

The main types of software testing are:

- Component.
- Interface.
- System.
- Acceptance.
- Release.

Acceptance Testing checks the system against the "Requirements". It is similar to systems testing in that the whole system is checked but the important difference is the change in focus:

Systems Testing checks that the system that was specified has been delivered.

Acceptance Testing checks that the system delivers what was requested. The customer, and not the developer should always do acceptance testing. The customer knows what is required

from the system to achieve value in the business and is the only person qualified to make that judgment. The User Acceptance Test Plan will vary from system to system but, in general, the testing should be planned in order to provide a realistic and adequate exposure of the system to all reasonably expected events. The testing can be based upon the User Requirements Specification to which the system should conform.

In engineering and its various sub disciplines, **acceptance testing** is black-box testing performed on a system (e.g. software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery. It is also known as functional testing, black-box testing, release acceptance, QA testing, application testing, confidence testing, final testing, validation testing, or factory acceptance testing.

In software development, **acceptance testing** by the system provider is often distinguished from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership. In such environments, acceptance testing performed by the customer is known as user acceptance testing (UAT). This is also known as end-user testing, site (acceptance) testing, or field (acceptance) testing.

A smoke test is used as an acceptance test prior to introducing a build to the main testing process.

Acceptance testing generally involves running a suite of tests on the completed system. Each individual test, known as a case, exercises a particular operating condition of the user's environment or feature of the system, and will result in a pass or fail boolean outcome. There is generally no degree of success or failure. The test environment is usually designed to be identical, or as close as possible, to the anticipated user's environment, including extremes of such. These test cases must each be accompanied by test case input data or a formal description of the operational activities (or both) to be performed—intended to thoroughly exercise the specific case—and a formal description of the expected results.

Acceptance Tests/Criterion (in Agile Software Development) are usually created by business customers and expressed in a business domain language. These are high level tests to test the completeness of a user story or stories 'played' during any sprint/iteration. These tests are

created ideally through collaboration between business customers, business analysts, testers and developers, however the business customers (product owners) are the primary owners of these tests. As the user stories pass their acceptance criteria, the business owners can be sure of the fact that the developers are progressing in the right direction about how the application was envisaged to work and so it's essential that these tests include both business logic tests as well as UI validation elements (if need be).

Acceptance test cards are ideally created during sprint planning or iteration planning meeting, before development begins so that the developers have a clear idea of what to develop. Sometimes (due to bad planning!) acceptance tests may span multiple stories (that are not implemented in the same sprint) and there are different ways to test them out during actual sprints. One popular technique is to mock external interfaces or data to mimic other stories which might not be played out during an iteration (as those stories may have been relatively lower business priority). A user story is not considered complete until the acceptance tests have passed.

The acceptance test suite is run against the supplied input data or using an acceptance test script to direct the testers. Then the results obtained are compared with the expected results. If there is a correct match for every case, the test suite is said to pass. If not, the system may either be rejected or accepted on conditions previously agreed between the sponsor and the manufacturer.

The objective is to provide confidence that the delivered system meets the business requirements of both sponsors and users. The acceptance phase may also act as the final quality gateway, where any quality defects not previously detected may be uncovered.

A principal purpose of acceptance testing is that, once completed successfully, and provided certain additional (contractually agreed) acceptance criteria are met, the sponsors will then sign off on the system as satisfying the contract (previously agreed between sponsor and manufacturer), and deliver final payment.

User Acceptance Testing (UAT) is a process to obtain confirmation by a Subject Matter Expert (SME), preferably the owner or client of the object under test, through trial or review, that the modification or addition meets mutually agreed-upon requirements. In software development,

UAT is one of the final stages of a project and often occurs before a client or customer accepts the new system.

Users of the system perform these tests which developers derive from the client's contract or the user requirements specification.

Test designers draw up formal tests and device a range of severity levels. It is preferable that the designer of the user acceptance tests not be the creator of the formal integration and system test cases for the same system, however there are some situations where this may not be avoided. The UAT acts as a final verification of the required business function and proper functioning of the system, emulating real-world usage conditions on behalf of the paying client or a specific large customer. If the software works as intended and without issues during normal use, one can reasonably infer the same level of stability in production. These tests, which are usually performed by clients or end-users, are not usually focused on identifying simple problems such as spelling errors and cosmetic problems, nor show stopper bugs, such as software crashes; testers and developers previously identify and fix these issues during earlier unit testing, integration testing, and system testing phases.

The results of these tests give confidence to the clients as to how the system will perform in production. There may also be legal or contractual requirement for acceptance of the system.

TYPES OF ACCEPTANCE TESTING

User acceptance testing

This may include factory acceptance testing, i.e. the testing done by factory users before the factory is moved to its own site, after which site acceptance testing may be performed by the users at the site.

Operational acceptance testing

Also known as operational readiness testing, this refers to the checking done to a system to ensure that processes and procedures are in place to allow the system to be used and maintained. This may include checks done to back-up facilities, procedures for disaster recovery, training for end users, maintenance procedures, and security procedures.

Contract and regulation acceptance testing

In contract acceptance testing, a system is tested against acceptance criteria as documented in a contract, before the system is accepted. In regulation acceptance testing, a system is tested to ensure it meets governmental, legal and safety standards.

Alpha and beta testing

Alpha testing takes place at developers' sites, and involves testing of the operational system by internal staff, before it is released to external customers. Beta testing takes place at customers' sites, and involves testing by a group of customers who use the system at their own locations and provide feedback, before the system is released to other customers. The latter is often called “field testing”.

INTEGRATION TESTING

Testing in which software components, hardware components, or both together are combined and tested to evaluate interactions between them. Integration testing takes as its input modules that have been checked out by unit testing, groups them in larger aggregates, applies tests defined in an Integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

'Integration testing' (sometimes called Integration and Testing, abbreviated **I&T**) is the phase of software testing in which individual software modules are combined and tested as a group. It follows unit testing and precedes system testing.

Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using Black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input

interface. Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing.

The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.Some different types of integration testing are big bang, top-down, and bottom-up.

There are two types in Integration Testing.

1. Bottom Up

2. Top Down.

Bottom-UP testing: An approach to integration testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage. The Top Down is the procedure where the top integrated modules are tested and the branch of the module is testing step by step till end of the related module. The main advantage of the Bottom up is easy to find bugs. In Top down it is easy to find the missing branch link.

Integration testing is a logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested. A component, in this sense, refers to an integrated aggregate of more than one unit. In a realistic scenario, many units are combined into components, which are in turn aggregated into even larger parts of the program. The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process

are tested together. Beyond that, if the program is composed of more than one process, they should be tested in pairs rather than all at once.

Integration testing identifies problems that occur when units are combined. By using a test plan that requires you to test each unit and ensure the viability of each before combining units, you know that any errors discovered when combining units are likely related to the interface between units. This method reduces the number of possibilities to a far simpler level of analysis.

You can do integration testing in a variety of ways but the following are three common strategies:

- The top-down approach to integration testing requires the highest-level modules be test and integrated first. This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers. However, the need for stubs complicates test management and low-level utilities are tested relatively late in the development cycle. Another disadvantage of top-down integration testing is its poor support for early release of limited functionality.
- The bottom-up approach requires the lowest-level units be tested and integrated first. These units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process and the need for stubs is minimized. The downside, however, is that the need for drivers complicates test management and high-level logic and data flow are tested late. Like the top-down approach, the bottom-up approach also provides poor support for early release of limited functionality.
- The third approach, sometimes referred to as the umbrella approach, requires testing along functional data and control-flow paths. First, the inputs for functions are integrated in the bottom-up pattern discussed above. The outputs for each function are then integrated in the top-down manner. The primary advantage of this approach is the degree of support for early release of limited functionality. It also helps minimize the need for stubs and drivers. The potential weaknesses of this approach are significant, however, in that it can be less systematic than the other two approaches, leading to the need for more regression testing.

REGRESSION TESTING

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process. Test department coders develop code test scenarios and exercises that will test new units of code after they have been written. Before a new version of a software product is released, the old test cases are run against the new version to make sure that all the old capabilities still work. The reason they might not work because changing or adding new code to a program can easily introduce errors into code that is not intended to be changed. It is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity.

Regression testing is any type of software testing which seeks to uncover software regressions. Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically, regressions occur as an unintended consequence of program changes. *Common methods* of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged.

Effective regression tests generate sufficient code execution coverage to exercise all meaningful code branches. Therefore, software testing is a combinatorial problem. However, in practice many combinations are unreachable so the problem size is greatly reduced. Every boolean decision statement requires at least two tests: one with an outcome of "true" and one with an outcome of "false". As a result, for every line of code written, programmers often need 3 to 5 lines of test code.

Traditionally, in the corporate world, regression testing has been performed by a software quality assurance team after the development team has completed work. However, defects found at this stage are most costly to fix. This problem is being addressed by the rise of developer testing. Although developers have always written test cases as part of the development cycle, these test cases have generally been either functional tests or unit tests that verify only intended outcomes. Developer testing compels a developer to focus on unit testing and to include both positive and negative test cases.

Any time you modify an implementation within a program, you should also do regression testing. You can do so by rerunning existing tests against the modified code to determine whether the changes break anything that worked prior to the change and by writing new tests where necessary. Adequate coverage without wasting time should be a primary consideration when conducting regression tests. Try to spend as little time as possible doing regression testing without reducing the probability that you will detect new failures in old, already tested code.

Some strategies and factors to consider during this process include the following:

- Test fixed bugs promptly. The programmer might have handled the symptoms but not have gotten to the underlying cause.
- Watch for side effects of fixes. The bug itself might be fixed but the fix might create other bugs.
- Write a regression test for each bug fixed.
- If two or more tests are similar, determine which is less effective and get rid of it.
- Identify tests that the program consistently passes and archive them.
- Focus on functional issues, not those related to design.
- Make changes (small and large) to data and find any resulting corruption.
- Trace the effects of the changes on program memory.

The selective retesting of a software system that has been modified to ensure that any bugs have been fixed and that no other previously working functions have failed as a result of the reparations and that newly added features have not created problems with previous versions of the software. Also referred to as *verification testing*, regression testing is initiated after a programmer has attempted to fix a recognized problem or has added source code to a program that may have inadvertently introduced errors. It is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity.

Regression testing means rerunning test cases from existing test suites to build confidence that software changes have no unintended side-effects. The “ideal” process would be to create an

extensive test suite and run it after each and every change. Unfortunately, for many projects this is just impossible because test suites are too large, because changes come in too fast, because humans are in the testing loop, because scarce, highly in-demand simulation laboratories are needed, or because testing must be done on many different hardware.

Researchers have tried to make regression testing more effective and efficient by developing regression test selection (RTS) techniques, but many problems remain, such as:

- **Unpredictable performance.** RTS techniques sometimes save time and money, but they sometimes select most or all of the original test cases. Thus, developers using RTS techniques can find themselves worse off for having done so.
- **Incompatible process assumptions.** Testing time is often limited (e.g., must be done overnight). RTS techniques do not consider such constraints and, therefore, can and do select more test cases than can be run.
- **Inappropriate evaluation models.** RTS techniques try to maximize average regression testing performance rather than optimize aggregate performance over many testing sessions. However, companies that test frequently might accept less effective, but cheaper individual testing sessions if the system would, nonetheless, be well-tested over some short period of time.

These and other issues have not been adequately considered in current research, yet they strongly affect the applicability of proposed regression testing processes. Moreover, we believe that solutions to these problems can be exploited, singly and in combination, to dramatically improve the costs and benefits of the regression testing process.

SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any

applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limiting type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

5. CONCLUSION

5.1 CONCLUSION

This project has successfully addressed a crucial challenge faced by visually impaired individuals by leveraging cutting-edge technology. By employing the YOLO (You Only Look Once) algorithm, the project has enabled real-time object detection in images and videos, providing blind people with important visual information about their surroundings.

Through the implementation of this project, blind individuals can now utilize a device or application that captures images or videos using a camera and processes them using the YOLO algorithm. The algorithm accurately identifies and classifies objects within the captured frames, such as pedestrians, vehicles, traffic signs, or obstacles, among others. The results are then converted into speech or audio output, allowing blind users to receive immediate verbal feedback about the objects detected.

The integration of speech as the output modality is a significant achievement, as it allows blind individuals to receive real-time information about their environment without relying on the traditional visual cues they lack. This technology has the potential to greatly enhance the independence, safety, and overall quality of life for blind people, empowering them to navigate their surroundings more confidently and efficiently. By providing real-time object detection and converting the results into speech output, this project offers a practical and accessible solution for blind individuals to perceive their surroundings more effectively.

REFERENCES

1. Object Detection with Voice Feedback — YOLO v3 + gTTS | by Jason Yip | Towards Data Science

2. https://www.w3schools.com/python/.

3. https://www.baeldung.com/cs/yolo-algorithm

4. https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection/