
# ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

## CT-361

## COMPLEX COMPUTING PROBLEM

## PROJECT REPORT


## GROUP MEMBERS:

1. HASNAIN RAZA (CR-034)
2. SHAYAAN MALIK (CR-036)
3. ARHAM NAEEM (CR-045)

# *PROJECT TITLE:*

Assistive Object Detection System for the Visually Impaired (Real-Time)

## Project Description

This project implements a real-time object detection and distance estimation system designed to assist **visually impaired individuals** by providing awareness of nearby obstacles and people in a video stream.

Using a YOLO (You Only Look Once) object detection model, it identifies objects in a video feed, estimates their distance from the camera, and determines their relative position (left, center, or right).

The system is intended as a prototype for assistive devices that can provide **audio or haptic feedback** to visually impaired users, alerting them to the presence and proximity of nearby objects for improved navigation and safety.

## Terms and Technologies Used

- YOLO (You Only Look Once)
- OpenCV (cv2)
- pyttsx3 (For Text-To-Speech conversion)
- Threading
- Queue
- calculate_distance function
- get_position function
- blur_person function
- class_avg_sizes dictionary
- Video Capture
- Output Video Writer
- Pause Functionality
- Detection Loop
- Nearest Object Queueing
- Speech Queue (threaded)
- Motion Direction Detection
- Cooldown System
- FOV Assumption

# How the Project Works

This project functions by capturing a continuous video stream, detecting objects within each frame in real-time, estimating their distance from the camera, and determining their relative position (left, center, right). It integrates an audio feedback system to provide situational awareness to visually impaired users.

## 1. Video Capture and Object Detection

- The system reads frames from a video source using **OpenCV**.
- Each frame is passed through the **YOLO object detection model** to identify objects like people, cars, and other predefined classes.

## 2. Distance Estimation

- For each detected object, the system estimates its distance based on the width of its bounding box and known average size ratios for different classes.
- A horizontal Field of View (FOV) of 70° is assumed to help with these calculations.

## 3. Position Detection

- The object's horizontal location in the frame is used to classify its position as **left**, **center**, or **right** relative to the camera.

## 4. Audio Announcements

- Detected objects are announced via **text-to-speech (pyttsx3)** if they fall within a predefined distance threshold (e.g., 12.5 meters).
- A **threaded speech queue system** ensures that audio announcements do not block ongoing object detection.

## 5. Motion Direction Alerts

- The system continuously compares the newly estimated distance of an object to its previously recorded distance.
- If the distance is **decreasing**, it announces "*Approaching*."
- If the distance is **increasing**, it announces "*Going Away*."
- If the distance stays roughly the same, no motion-related update is given.

## 6. Cooldown Mechanism

- To avoid overwhelming the user with repetitive alerts, a **cooldown timer** limits how frequently the same object can trigger an audio announcement.

# Description of Terms, Code, and Justifications

| Term | Description | Code / Function | Justification |
|---|---|---|---|
| **YOLO** | A deep learning object detection model capable of real-time detection. | model = YOLO(MODEL_PATH) | Chosen for its speed and accuracy for multi-class object detection. |
| **Text-to-Speech (TTS)** | Converts detected object information into spoken audio output. | pyttsx3.init(), engine.say(), engine.runAndWait() | Provides an audio interface for visually impaired users. |
| **calculate_distance** | Computes estimated distance of an object from the camera based on its detected width and known ratios. | def calculate_distance(box, frame_width, label): | Required to give approximate physical distance to detected objects. |
| **get_position** | Determines whether a detected object lies in left, center, or right section of video frame. | def get_position(frame_width, box): | Essential for spatial awareness and directional feedback. |
| **blur_person** | Blurs the upper region of detected person objects to protect identity. | def blur_person(image, box): | Ensures privacy while processing video footage. |
| **class_avg_sizes** | Dictionary storing average width ratios of object classes relative to frame width. | class_avg_sizes = {...} | Used for accurate distance estimation since object size varies by category. |
| **Video Capture** | Opens the video stream for frame-by-frame reading. | cap = cv2.VideoCapture(VIDEO_PATH) | Handles real-time or pre-recorded video input. |
| **Output Video Writer** | Configures and saves processed video with overlays to a file. | cv2.VideoWriter() | Useful for saving annotated video for review or documentation. |
| **Pause Functionality** | Allows pausing/resuming video processing on keypress. | if key == ord('p'): | Aids in interactive debugging or frame-by-frame analysis. |
| **Detection Loop** | Core operational loop: reads frames, applies detection, calculates | while cap.isOpened(): | Main logic for real-time video processing and analysis. |

| | | | |
|---|---|---|---|
| | distance, and draws annotations. | | |
| **Nearest Object Queueing** | Detects nearest object within a threshold and adds it to a queue for TTS feedback. | if nearest_object and nearest_object[1] <= 12.5: | Provides situational awareness by alerting the user about nearby objects. |
| **Speech Queue (threaded)** | Threaded queue system for asynchronous TTS announcements to avoid blocking detection loop. | Queue(), Thread(target=speak, args=(queue,), daemon=True) | Ensures continuous detection without delays from TTS operations. |
| **Motion Direction Detection** | Determines whether an object is approaching, going away, or static based on distance changes. | Inside speak(q) function comparing prev_distance and distance | Adds context to object alerts to inform the user of object movement relative to them. |
| **Cooldown System** | Prevents repeated speech alerts for the same object within a set time interval. | last_spoken dictionary and speech_cooldown variable | Avoids flooding the user with repetitive notifications for the same object within short periods. |
| **FOV Assumption** | Assumes a camera horizontal field of view (FOV) of 70° for distance calculations. | np.tan(np.radians(70 / 2)) in calculate_distance | Necessary for estimating distance based on object width in pixels. Can be calibrated per device. |

# CODE AND OUTPUT:

```
import pyttsx3
from threading import Thread
from queue import Queue
from ultralytics import YOLO
import cv2
import numpy as np
import time
import os

# Define absolute paths
MODEL_PATH = r"C:\Users\pc\Desktop\Object-Detection-Project\gpModel.pt"
VIDEO_PATH = r"C:\Users\pc\Desktop\Object-Detection-Project\test_video1.mp4"

# Initialize TTS
engine = pyttsx3.init()
engine.setProperty('rate', 235)
engine.setProperty('volume', 1.0)
engine.say("System activated")
engine.runAndWait()

# Cooldown and distance trackingh
last_spoken = {}
last_distances = {}
speech_cooldown = 5  # seconds

# Queue for speech
queue = Queue()

def speak(q):
    while True:
        if not q.empty():
            label, distance, position = q.get()
            current_time = time.time()
            rounded_distance = round(distance * 2) / 2
            distance_str = str(int(rounded_distance)) if rounded_distance.is_integer() else str(rounded_distance)

            # Cooldown to prevent flooding
            if label in last_spoken and current_time - last_spoken[label] < speech_cooldown:
                continue
```

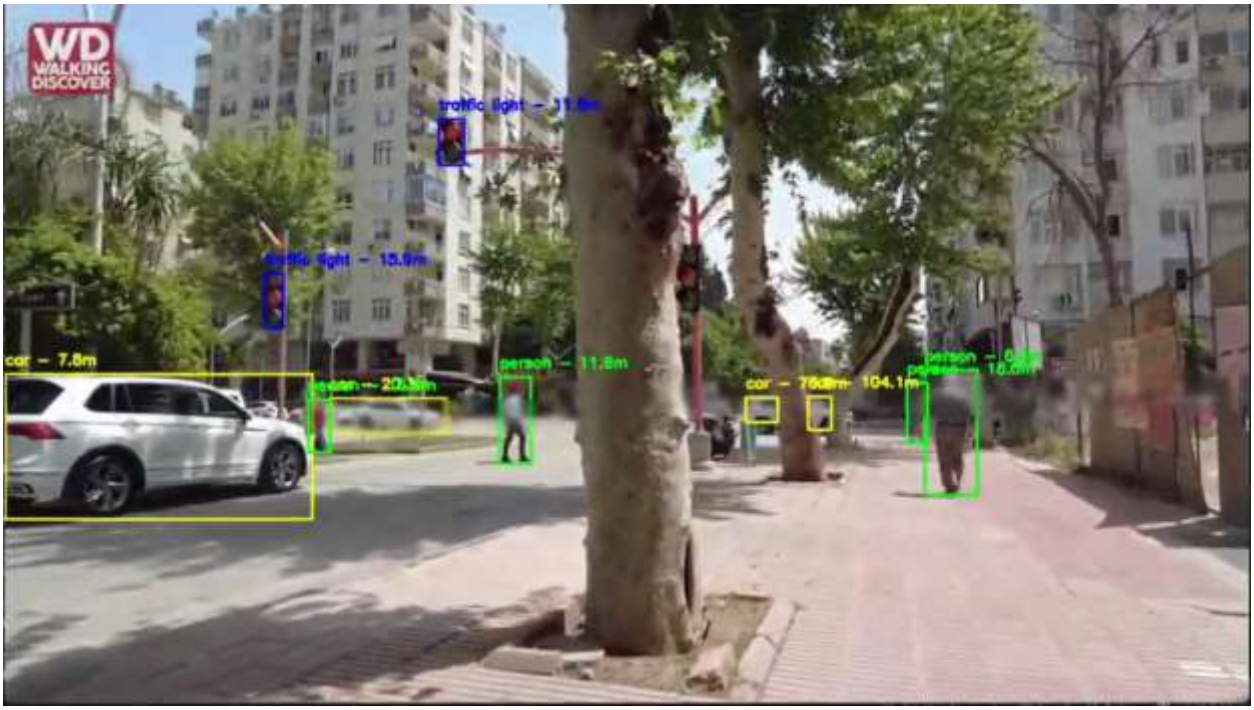
```
            # Determine motion direction
            prev_distance = last_distances.get(label, None)
            if prev_distance is not None:
                if distance < prev_distance - 0.3:
                    motion = "approaching"
                elif distance > prev_distance + 0.3:
                    motion = "going away"
                else:
                    motion = "ahead"
            else:
                motion = "ahead"

            if distance <= 2:
                motion = "very close"

            last_distances[label] = distance

            engine.say(f"{label} is {distance_str} meters on your {position}, {motion}")
            engine.runAndWait()

            last_spoken[label] = current_time

            with queue.mutex:
                queue.queue.clear()
        else:
            time.sleep(0.1)
```

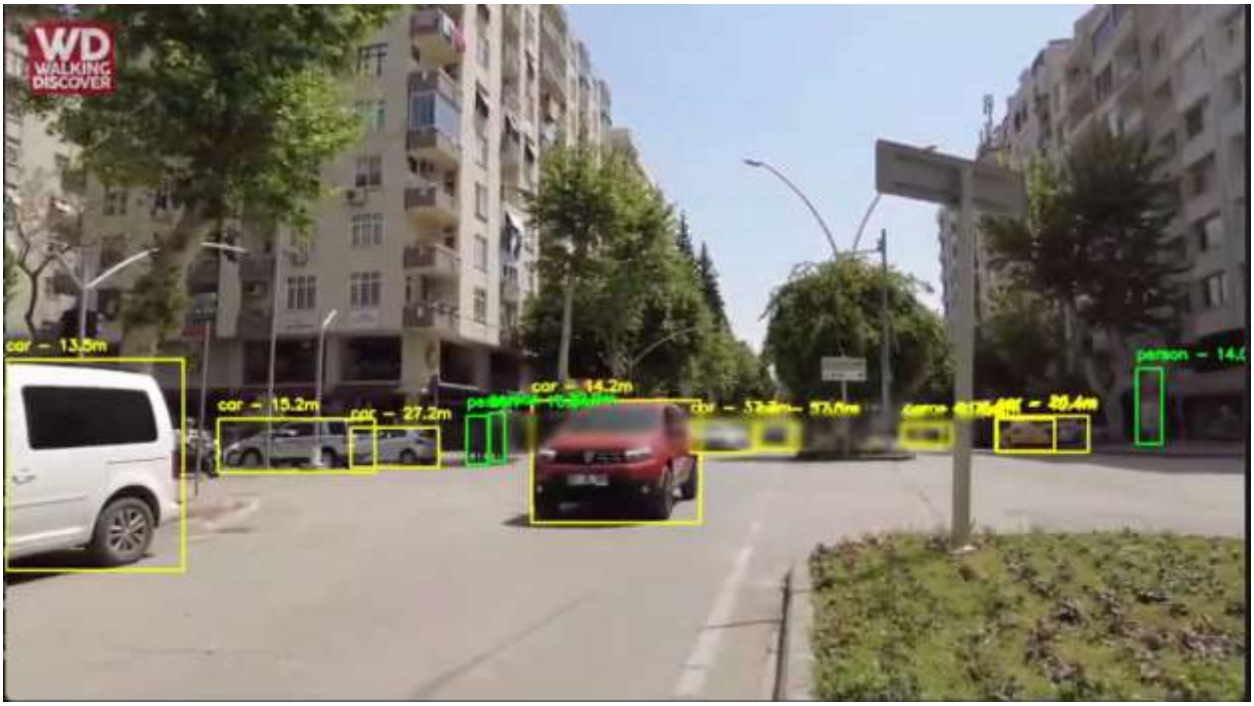
```
# Start TTS thread
Thread(target=speak, args=(queue,), daemon=True).start()

# Calculate distance
def calculate_distance(box, frame_width, label):
    object_width = box.xyxy[0, 2].item() - box.xyxy[0, 0].item()
    if label in class_avg_sizes:
        object_width *= class_avg_sizes[label]["width_ratio"]
    distance = (frame_width * 0.5) / np.tan(np.radians(70 / 2)) / (object_width + 1e-6)
    return round(distance, 2)

# Get object position
def get_position(frame_width, box):
    if box[0] < frame_width // 3:
        return "left"
    elif box[0] < 2 * (frame_width // 3):
        return "center"
    else:
        return "right"

# Blur region
def blur_person(image, box):
    x, y, w, h = box.xyxy[0].cpu().numpy().astype(int)
    top_region = image[y:y+int(0.08 * h), x:x+w]
    blurred_top_region = cv2.GaussianBlur(top_region, (15, 15), 0)
    image[y:y+int(0.08 * h), x:x+w] = blurred_top_region
    return image

# Load model and video
model = YOLO(MODEL_PATH)
cap = cv2.VideoCapture(VIDEO_PATH)

# Output video setup
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output_with_boxes.avi', fourcc, 20.0, (int(cap.get(3)), int(cap.get(4))))
```

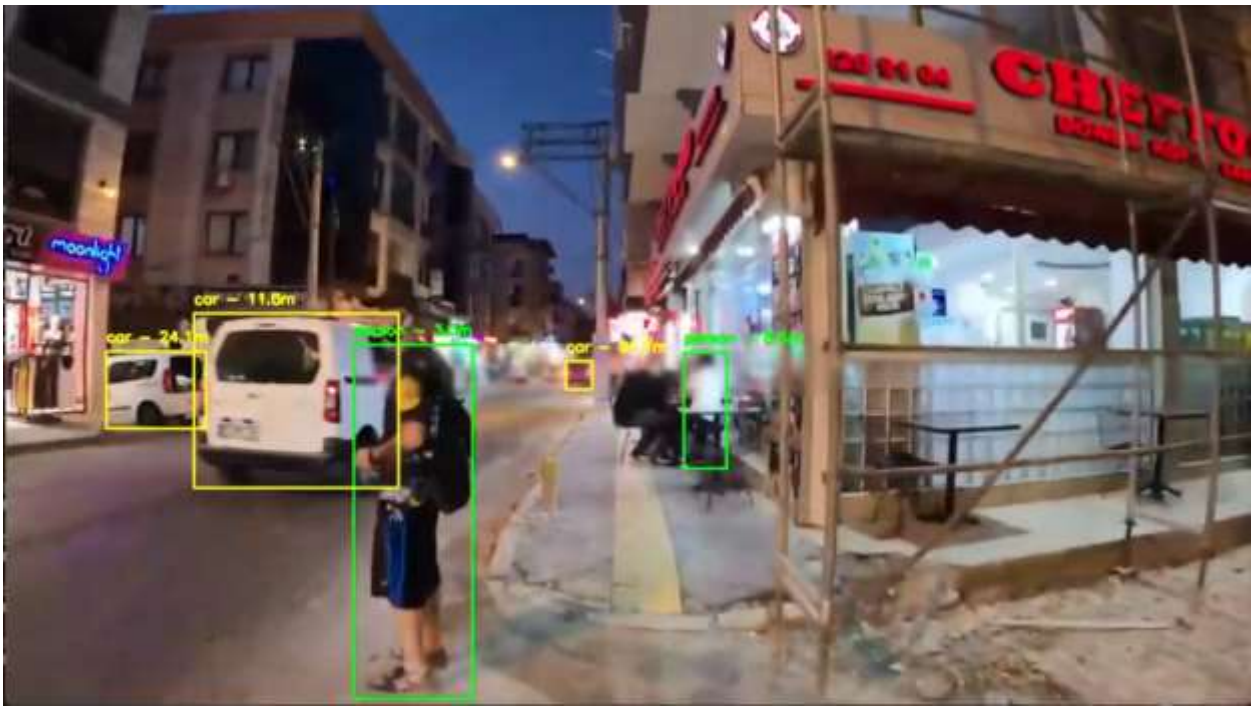
```
# Object widths
class_avg_sizes = {
    "person": {"width_ratio": 2.5},
    "car": {"width_ratio": 0.37},
    "bicycle": {"width_ratio": 2.3},
    "motorcycle": {"width_ratio": 2.4},
    "bus": {"width_ratio": 0.3},
    "traffic light": {"width_ratio": 2.95},
    "stop sign": {"width_ratio": 2.55},
    "bench": {"width_ratio": 1.6},
    "cat": {"width_ratio": 1.9},
    "dog": {"width_ratio": 1.5},
}

pause = False
while cap.isOpened():
    if not pause:
        ret, frame = cap.read()
        if not ret:
            break

        results = model.predict(frame)
        result = results[0]
        nearest_object = None
        min_distance = float('inf')

        for box in result.boxes:
            label = result.names[box.cls[0].item()]
            cords = [round(x) for x in box.xyxy[0].tolist()]
            distance = calculate_distance(box, frame.shape[1], label)
```

```
            if distance < min_distance:
                min_distance = distance
                nearest_object = (label, round(distance, 1), cords)

            if label == "person":
                frame = blur_person(frame, box)
                color = (0, 255, 0)
            elif label == "car":
                color = (0, 255, 255)
            elif label in class_avg_sizes:
                color = (255, 0, 0)
            else:
                continue

            cv2.rectangle(frame, (cords[0], cords[1]), (cords[2], cords[3]), color, 2)
            cv2.putText(frame, f"{label} - {distance:.1f}m", (cords[0], cords[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

        if nearest_object and nearest_object[1] <= 12.5:
            position = get_position(frame.shape[1], nearest_object[2])
            queue.put((nearest_object[0], nearest_object[1], position))

        cv2.imshow('Audio World', frame)
        out.write(frame)  # Save video

    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
    elif key == ord('p'):
        pause = not pause

cap.release()
out.release()
cv2.destroyAllWindows()
```

# OUTPUT:




WD
WALKING
DISCOVER
car – 13.0m
car – 15.2m
car – 27.2m
car – 14.2m
car – 20.4m
person – 14.


moonlight
car – 11.8m
car – 24.1m

# Drafted Assumptions

- The camera field of view (FOV) is assumed to be 70 degrees.
- Average object width ratios are predefined and constant for distance estimation.
- The video stream is assumed to be clear and with a consistent frame rate.
- Distance estimation is approximate and should not be used for precision-critical applications.
- The 'person' object top region blur is limited to 8% of their detected height.

# Conclusion

This project successfully demonstrates a real-time object detection and distance estimation system tailored to assist visually impaired individuals in navigating their surroundings more safely. By integrating the YOLO object detection model with a distance estimation mechanism, the system identifies nearby obstacles such as people, vehicles, and traffic signs within a video stream and calculates their approximate distance and position relative to the user.

Through this prototype, we highlighted the potential of computer vision and AI-powered solutions in the field of assistive technology. The system can be further enhanced to provide audio cues or haptic feedback based on object proximity and position, helping visually impaired individuals avoid collisions and move confidently in both indoor and outdoor environments.

Moreover, this project reinforces how modern AI tools can be repurposed not just for surveillance or analytics, but for inclusive, life-improving applications that address real-world challenges faced by underrepresented communities.

In future iterations, improvements like integrating obstacle classification confidence scores**,** object tracking**,** real-time audio announcements, and wearable camera integration can transform this concept into a practical, real-world assistive device.