EE24BTECH11002 Agamjot Singh
IIT Hyderabad

May 17, 2025

Task 2 - Vision-Based Robot Navigation System Using Real-Time Object Detection

Problem Statement

Develop a Python-based intelligent system that processes live video input to detect objects, analyze spatial positions, and generate navigational commands for autonomous robot movement.

Note: The system must be capable of real-time object detection (≥15 FPS) using lightweight models such as YOLOv5n, YOLOv8n, or MobileNet-SSD. It should perform position analysis via bounding box centroids and generate navigation commands based on object location (Left/Center/Right).

Introduction

The program uses a webcam feed or a video (if provided), identifies objects, determines their positions within the frame, and decides whether to move left, right, forward, stop (to follow the object only if there is one present), or perform an emergency stop if it sees a person in sight.
The Object Detection is done using the pretrained YOLO v8 Nano Model.

Code Explanation

YOLO Model and Webcam Setup

We begin by loading the YOLOv8 nano model and opening the webcam or the video (if provided),

```
from ultralytics import YOLO
import cv2

model = YOLO("yolov8n.pt")  # Load the lightweight nano model

# Open webcam or the video (if provided is sysargs)
if(len(sys.argv) == 1): cap = cv2.VideoCapture(0)
else: cap = cv2.VideoCapture(sys.argv[1])
```

We use ultralytics to use the pretrained YOLO v8 Nano model. Ultralytics uses pytorch under the hood to process real-time object detection.

Zone Classification Function

The frame is divided into three equal horizontal zones: Left, Center, and Right. We define a helper function to classify the zone of an object's box centroid,

```
def zone(centroid, width):
    x = centroid[0]
    if x < width/3:
        return "Left"
    elif width/3 <= x < 2*width/3:
        return "Center"
    else:
        return "Right"
```

It takes in centroid of the box and the width of the image and divides it into three equal zones based on it.

Frame Processing Loop

```
while True:
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame, conf=0.6, device="cpu", verbose=False)
```

```
    height, width = frame.shape[:2]
```

Vertical Green lines are drawn to indicate zone boundaries, for ease of use and testing,

```
x1 = width // 3
x2 = 2 * width // 3
cv2.line(frame, (x1, 0), (x1, height), (0, 255, 0), 2)
cv2.line(frame, (x2, 0), (x2, height), (0, 255, 0), 2)
```

Object Detection and Decision Logic

We access detected bounding boxes which are identified by the YOLO model. (x1, y1) and (x2, y2) are two ends of the diagonal of the rectangle detected by the model (left to right orientation, like you would draw a rectangle in a computer software) and the centroid is given by ((x1 + x2)/2, (y1 + y2)/2).

```
boxes = results[0].boxes

if len(boxes) != 1:
    print("Stop")
else:
    x1, y1, x2, y2 = boxes.xyxy[0]
    class_id = int(boxes[0].cls[0])
    class_name = model.names[class_id]

    if class_name.lower() == "person":
        print("EMERGENCY Stop")
    else:
        centroid = ((x1 + x2) / 2, (y1 + y2) / 2)
        object_zone = zone(centroid, width)
        cv2.circle(frame, (int(centroid[0]), int(centroid[1])), 5,
           (0, 0, 255), -1)
        print(object_zone)
```

First, we get all the boxes (name self explanatory) and we see if there are multiple objects. If there are multiple, we just stop there. If there is only one object we print the zone the bot should go in, or if there is a person then we emergency stop.

Note that the bot will follow the object based on the instructions given, not avoid it.

Display and Exit Conditions

The annotated frame is displayed with the detected bounding boxes, and the loop exits when 'q' is pressed.

```
annotated_frame = results[0].plot()
cv2.imshow("YOLOv8 Detection", annotated_frame)

if cv2.waitKey(10) & 0xFF == ord('q'):
    break

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