Real-Time Object Detection for Inventory Monitoring

By: Kavin Arasan A/L Mudiarasan

Role Applied For: IT Internship

1 Introduction

This project demonstrates the use of a real-time object detection system powered by YOLOv8 and optimized to run on GPU (CUDA) for high performance. The model detects and counts selected everyday objects. The solution is designed with potential applications in inventory management, automated quality checks, workplace monitoring, and smart warehouses. It achieves smooth real-time detection with stable counting and class filtering.

2 Objective

1. Build a real-time object detection model using OpenCV.
2. Achieve almost perfect detection and counting under given timeline.
3. Provide a framework that can be extended to warehouse inventory automation or security monitoring.

3 System Architecture

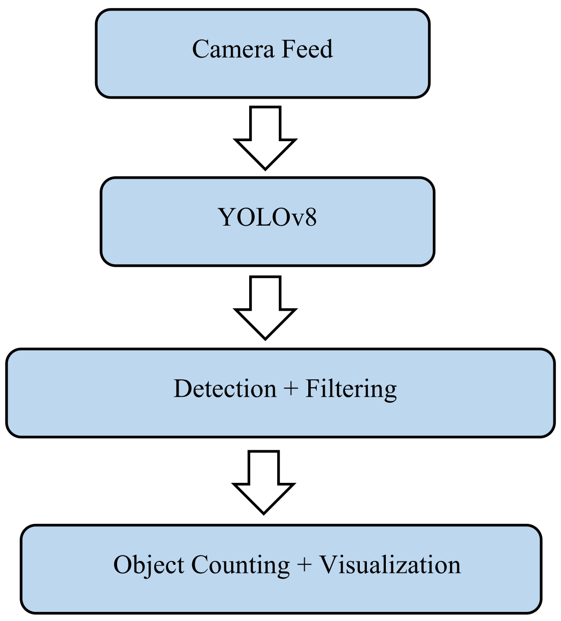
3.1 Hardware Setup

1. Laptop with NVIDIA GPU (CUDA enabled)
2. Laptop webcam as input source.

3.2 Software & Libraries

1. Python 3.11
2. Open CV for video processing and visualization.
3. Ultralytics YOLOv8 for object detection.
4. PyTorch for GPU acceleration.
5. LAP for multi-object tracking support which is installed as dependency.

3.3 Workflow Diagram



4 Model Architecture

4.1 YOLOv8s

The YOLO model chosen for this project is YOLOv8 small version which has :

1. Real-time speed
2. High accuracy across common object categories
3. Optimal for detecting small objects
4. Support for GPU acceleration

4.2 Classes of Interest

Selected 10 classes relevant to real-world usage: person, cup, chair, cell phone, book, scissors, keyboard, toothbrush, and bottle. The class filtering technique is used to ensure that irrelevant classes for this project demonstration, such as "dog" or "car" don’t clutter the results.

4.3 Key Features

4.3.1 Custom Class Filtering

Instead of detecting all COCO classes, the system explicitly filters and restricts detection to 10 chosen classes (cup, chair, cell phone, book, person, scissors, keyboard, toothbrush, bottle). This step reduces noise, improves accuracy for relevant objects, and makes the system domain-specific. By narrowing focus, the detector becomes faster, less error-prone, and better aligned with real-world use case.

4.3.2 Confidence and IoU Threshold Tuning

The confidence threshold (CONF = 0.55) and Intersection-over-Union threshold (IOU = 0.40) were carefully tuned to balance precision and recall. A slightly lower IoU improves the system's ability to separate overlapping objects, while a moderate confidence level reduces false positives. These thresholds make the detector more robust in complex environments where multiple objects may overlap or appear in partial views.

4.3.3 Temporal Smoothing with Track Persistence (TTL)

A Time-To-Live mechanism (TTL = 5) is implemented to handle short-term disappearances when objects are momentarily occluded or detection flickers. This avoids object blinking and ensures stable tracking which is crucial for video analytic where objects may pass behind obstacles. Temporal smoothing gives the perception of continuous tracking instead of disjointed bounding boxes.

4.3.4 ByteTrack Integration for Stable Tracking

The model uses ByteTrack which is a multi-object tracking algorithm to assign unique IDs to each detected object across frames. This enables the system to distinguish between multiple instances of the same class and ensures accurate per-class counting. Without tracking, objects would be re-counted every frame, leading to inflated statistics.

4.3.5 FPS Monitoring for Performance Evaluation

The system measures and overlays the frames-per-second (FPS) rate to ensure real-time performance is maintained. This feature is included because it is crucial when deploying on different hardware setups as FPS directly impacts usability.

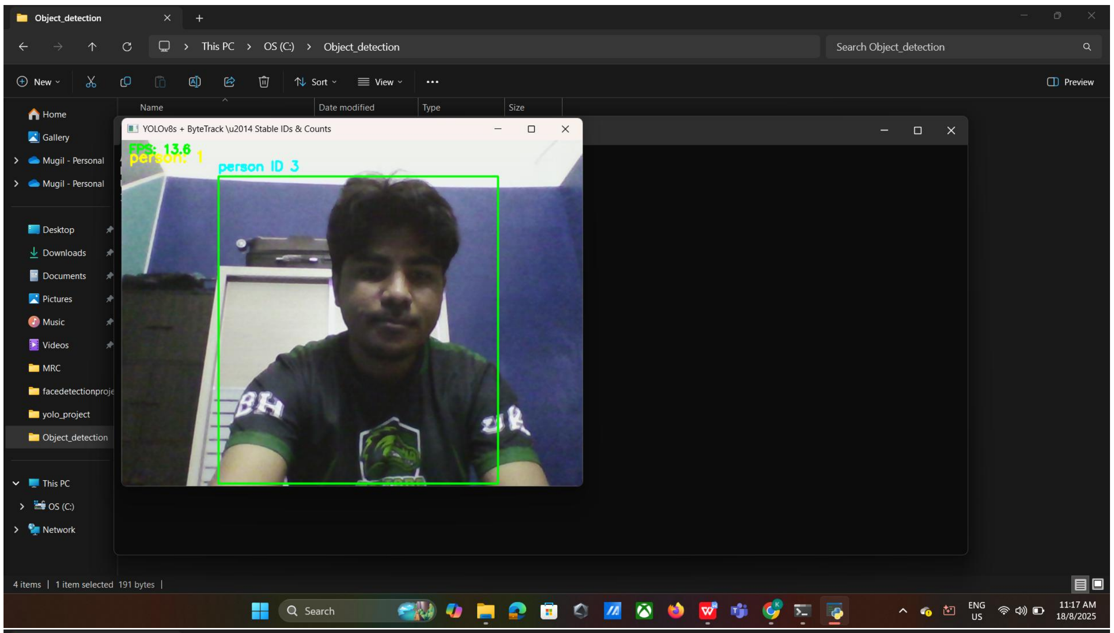
4.3.6 Object Center Point Visualization

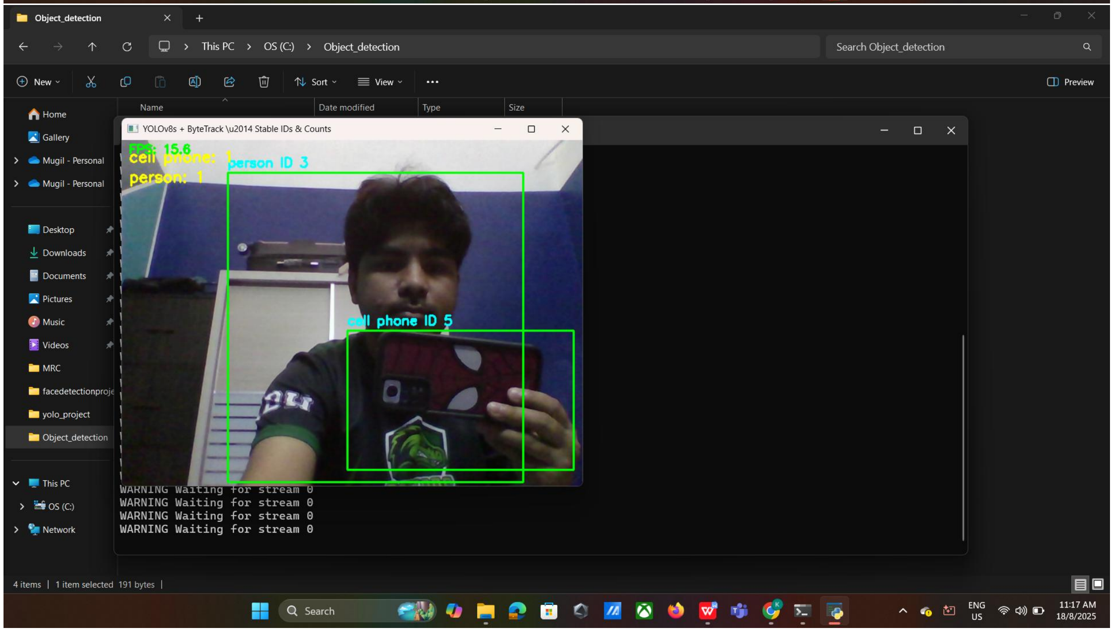
For every detected and tracked object, the system computes and marks the center point of its bounding box. The center is calculated as the midpoint between the top-left and bottom-right coordinates of the detection box. Bounding boxes show the object's outline, but the center point gives an exact reference for the object's position in the scene which actually sets the stage for expansion into advanced applications like robotics navigation, automated surveillance.This feature is included to be useful during the integration of robotics/automation. The robot can grip/hold the object by implementing this model where it can show the centre point of the object for precise position tracking.

5 Results

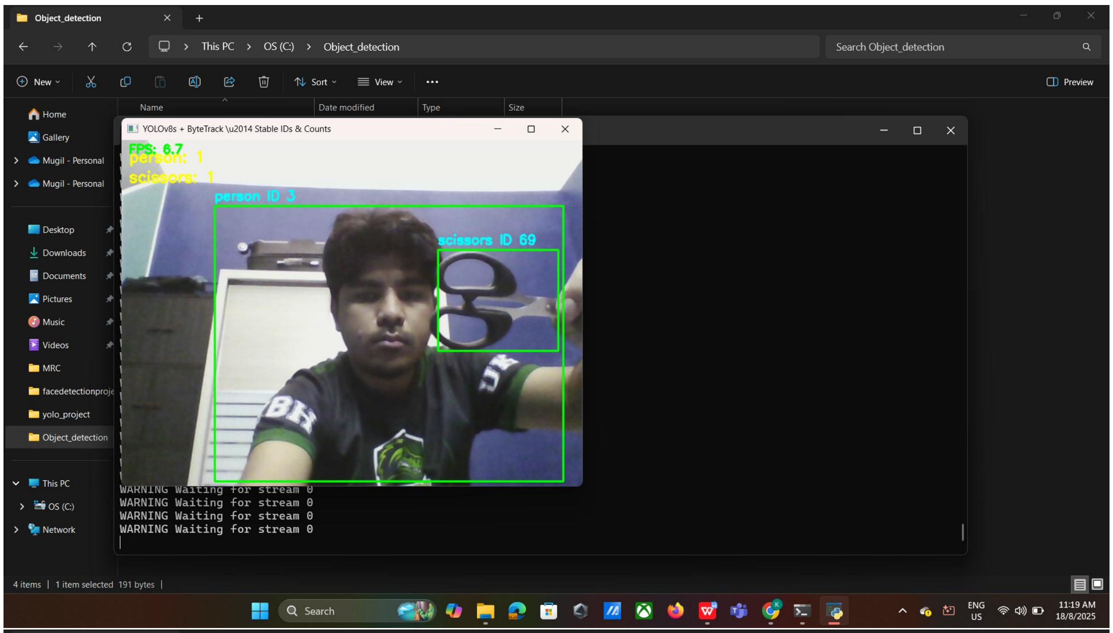
5.1 Performance

Detection quality is high and good accuracy across all objects in the tested 10 classes. The counting accuracy is stable but requires improved tracking and higher resolution camera for overlapping small objects. The speed of the real-time running model is 30 FPS on CUDA.

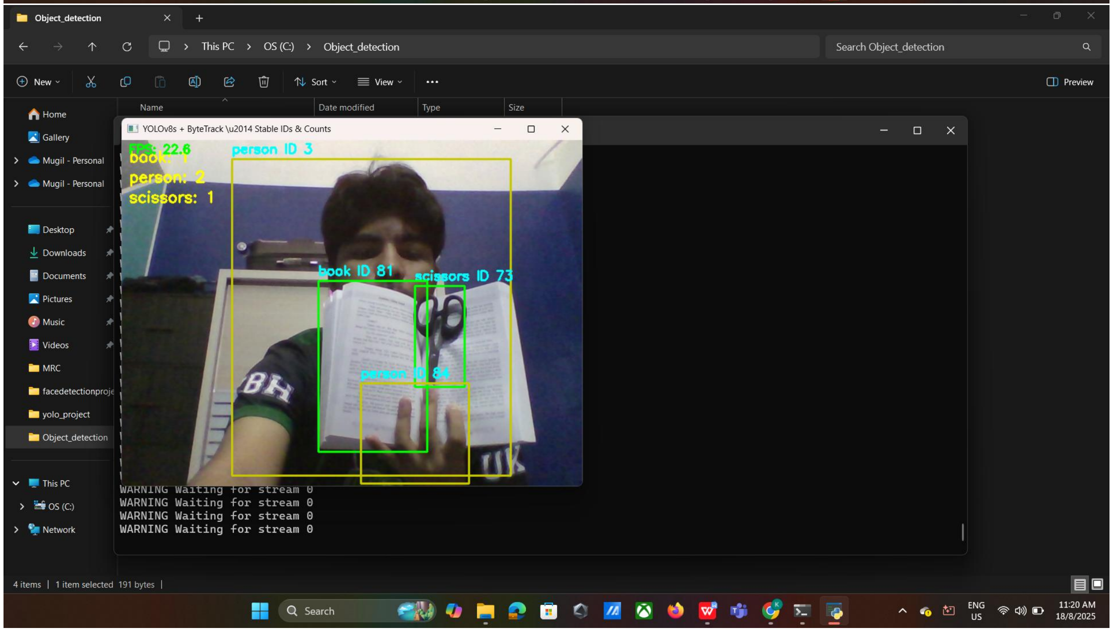

Object_detection
This PC > OS (C:) > Object_detection
Search Object_detection
YOLOv8s + ByteTrack \u2014 Stable IDs & Counts
FPS: 13.6
person: 1
person ID 3
4 items | 1 item selected 191 bytes
11:17 AM
18/8/2025


Object_detection
This PC > OS (C:) > Object_detection
Search Object_detection
YOLOv8s + ByteTrack \u2014 Stable IDs & Counts
FPS: 15.6
cell phone: 1
person: 1
person ID 3
cell phone ID 5
4 items | 1 item selected 191 bytes
11:17 AM
18/8/2025

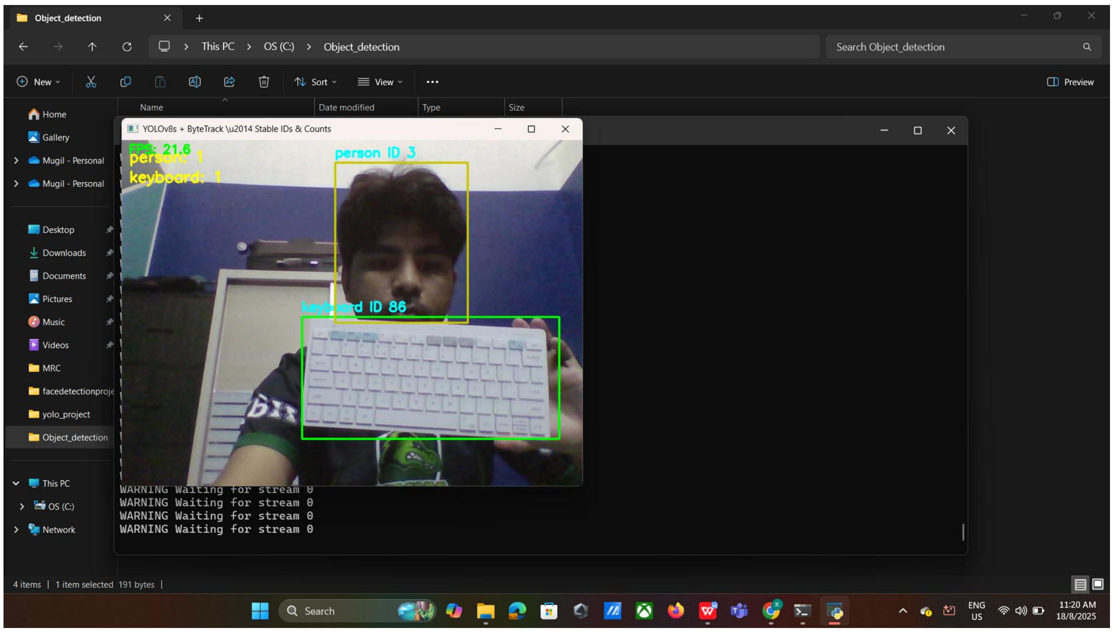
```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```


Object_detection
This PC > OS (C:) > Object_detection
Search Object_detection
YOLOv8s + ByteTrack \u2014 Stable IDs & Counts
FPS: 6.7
person: 1
scissors: 1
person ID 3
scissors ID 69
4 items | 1 item selected 191 bytes
11:19 AM
18/8/2025

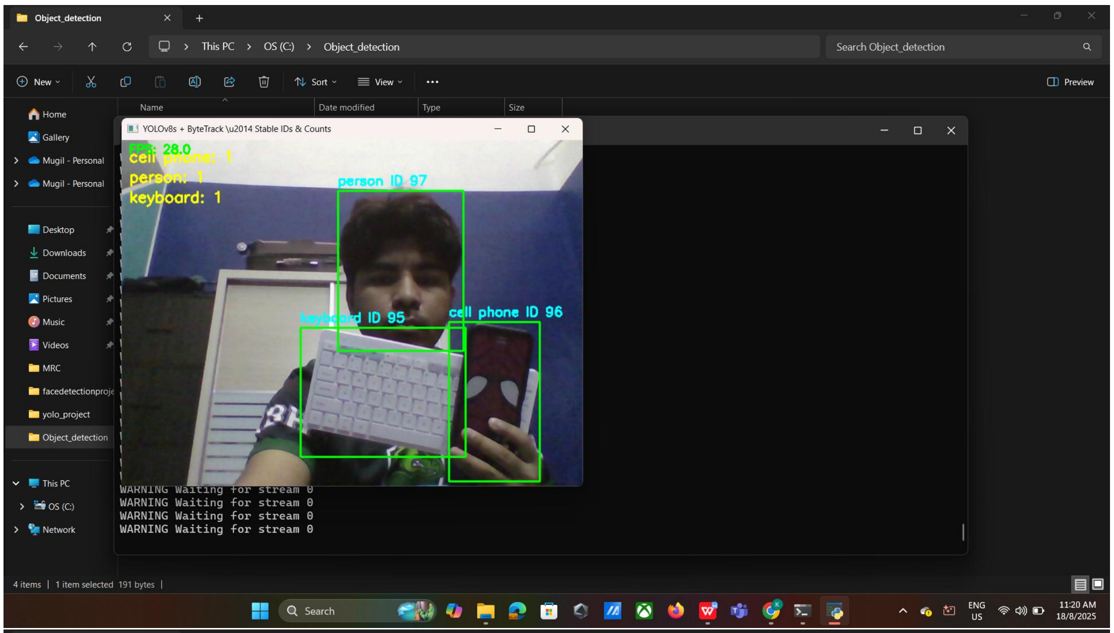
```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```



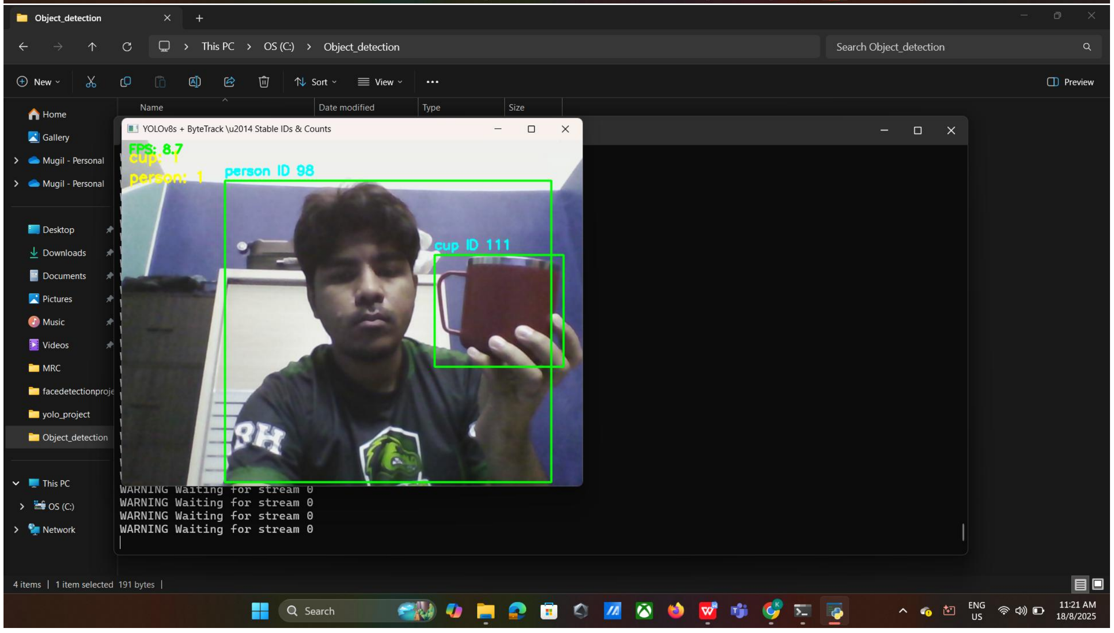
```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```



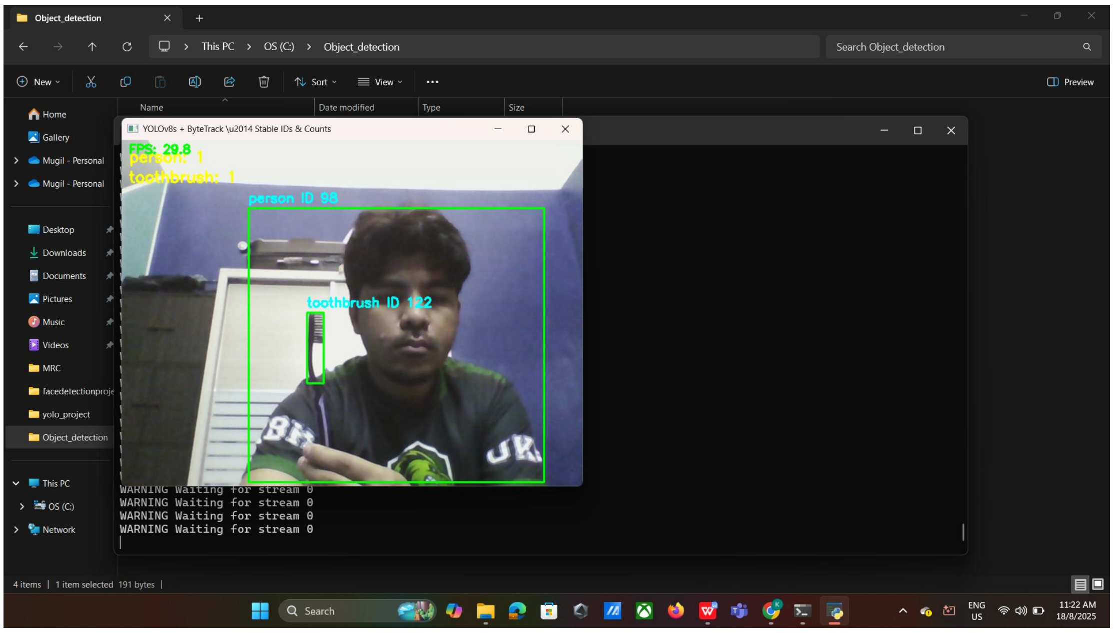
```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```



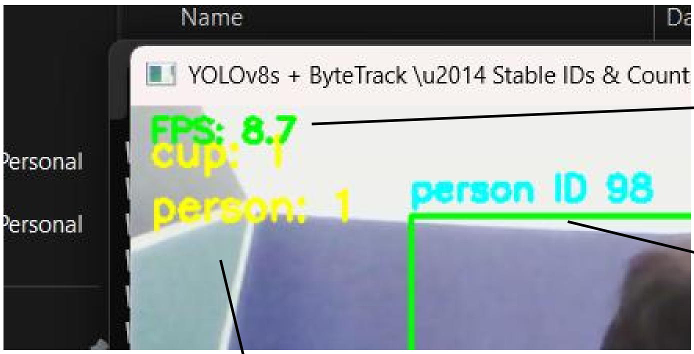
```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```



```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```


Object_detection
This PC > OS (C:) > Object_detection
Search Object_detection
YOLOv8s + ByteTrack \u2014 Stable IDs & Counts
FPS: 29.8
person: 1
toothbrush: 1
person ID 98
toothbrush ID 122
4 items | 1 item selected 191 bytes
11:22 AM
18/8/2025

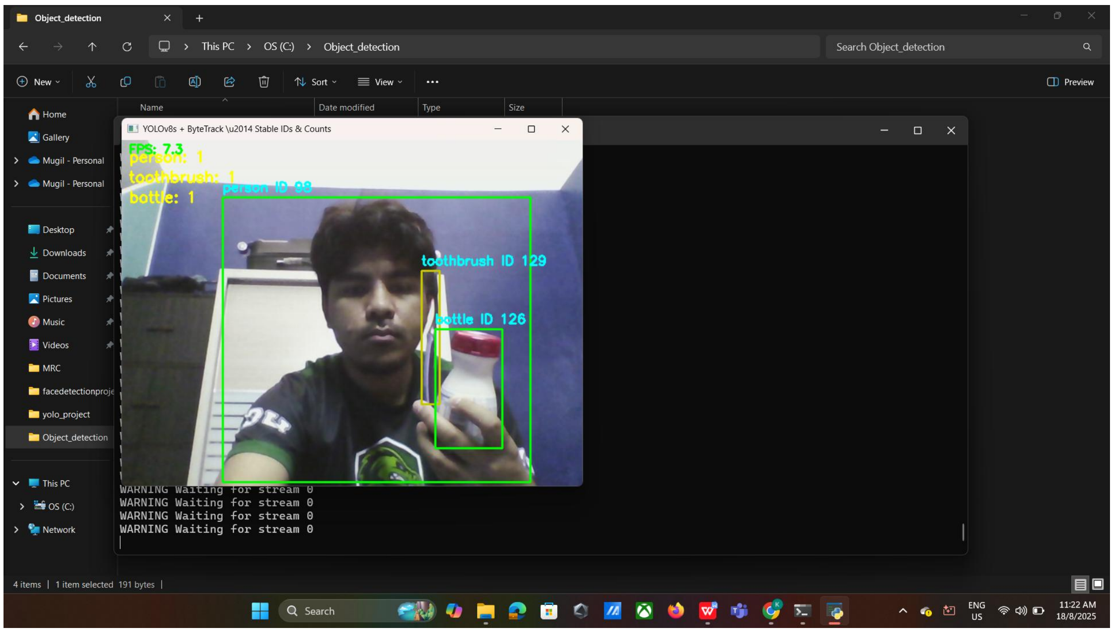
```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```



REAL-TIME FRAME RATE PER SECOND DISPLAY

UNIQUE ID FOR EACH OBJECT DETECTED

REAL-TIME INVENTORY COUNT DISPLAY



```
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
WARNING Waiting for stream 0
```

5.2 Observations

1. Overlapping objects like two small items close together is high but not perfect due to low resolution of webcam camera.
2. Larger objects such person, chair detected reliably.
3. System performs best when objects are well-lit and partially visible.

6 Limitations & Improvements

1. Limitation

Multiple or more than 3 overlapping small objects reduce detection accuracy.

2. Improvements

1. Train YOLO on custom dataset and fine tune the model for higher accuracy in detecting specific objects that user wants.
2. Expand deployment to an Mobile App, Robotic Arm and IoT device.

7 Conclusion

This project successfully delivers a real-time, GPU-accelerated object detection system with stable detection and counting for selected classes. It demonstrates technical knowledge, practical problem-solving, and awareness of industry applications making it directly relevant for automation, AI and computer vision.

Appendix

Full Python script

```
from ultralytics import YOLO
import cv2
import torch
import time
from collections import defaultdict

model = YOLO("yolov8s.pt")
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
print("Using device:", device)

ALLOWED_CLASSES = ["cup", "chair", "cell phone", "book", "person",
               "scissors", "keyboard", "toothbrush", "bottle", "fork"]

CONF = 0.55
IOU  = 0.40
IMGSZ = 960
TTL = 5

name_map = model.names  # id -> label
allowed_ids = [i for i, n in name_map.items() if n in ALLOWED_CLASSES]
allowed_set = set(allowed_ids)

active_tracks = {}

prev_time = time.time()

results_stream = model.track(
    source=0,
    stream=True,
    persist=True,
    tracker="bytetrack.yaml",
    imgsz=IMGSZ,
    conf=CONF,
    iou=IOU,
    classes=allowed_ids,
    verbose=False,
    half=True if device == "cuda" else False
)

for r in results_stream:
    frame = r.orig_img.copy()
    boxes = r.boxes

    present_keys = set()
```

```
if boxes is not None and len(boxes) > 0:
    xyxy = boxes.xyxy
    clss = boxes.cls
    confs = boxes.conf
    ids   = boxes.id

    n = len(xyxy)
    for i in range(n):
        cls_id = int(clss[i].item())
        if cls_id not in allowed_set:
            continue

        if ids is None or ids[i] is None:
            x1, y1, x2, y2 = map(int, xyxy[i].tolist())
            label = name_map[cls_id]
            cv2.rectangle(frame, (x1, y1), (x2, y2), (128, 128, 128), 1)
            cv2.putText(frame, f"{label} ...", (x1, y1 - 8),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.55, (200, 200, 200), 1)
            continue

        tid = int(ids[i].item())
        x1, y1, x2, y2 = map(int, xyxy[i].tolist())
        conf = float(confs[i].item())

        key = (cls_id, tid)
        present_keys.add(key)
        active_tracks[key] = {"bbox": (x1, y1, x2, y2), "conf": conf, "miss": 0}

to_drop = []
for key, info in active_tracks.items():
    if key not in present_keys:
        info["miss"] += 1
        if info["miss"] > TTL:
            to_drop.append(key)
for key in to_drop:
    del active_tracks[key]

per_class_counts = defaultdict(int)
for (cls_id, tid), info in active_tracks.items():
    x1, y1, x2, y2 = info["bbox"]
    label = name_map[cls_id]
    per_class_counts[label] += 1

    color = (0, 255, 0) if info["miss"] == 0 else (0, 200, 200)
    cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
    cv2.putText(frame, f"{label} ID {tid}", (x1, y1 - 8),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 0), 2)

y = 30
for obj in ALLOWED_CLASSES:
    c = per_class_counts.get(obj, 0)
    if c > 0:
        cv2.putText(frame, f"{obj}: {c}", (10, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)
        y += 28
    fps = 1.0 / max(1e-6, (time.time() - prev_time))
    prev_time = time.time()
    cv2.putText(frame, f"FPS: {fps:.1f}", (10, 18),
            cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 255, 0), 2)

    cv2.imshow("YOLOv8s + ByteTrack — Stable IDs & Counts", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```

How To Best Run the Model

Run the model through command prompt by going to the directory using cd prompt.

Create virtual python environment to enable GPU usage.This ensures the model runs smoothly and the FPS and detection has clarity.

https://drive.google.com/drive/folders/1Oo5Fj62aIdpOjfurDKGXCRGpg9KW6gLL?usp=sharing

You can access the project folder by clicking the google drive link above.