2.2 Object Tracking Acroos Frames

- Apply object detection to a short **5-10 second video** containing moving objects
- Track the movement of at least **two objects**acroos frames
- log their position changes and generate a simple visualization.

1. importing and finding libraries and packages

In [7]:
```
import cv2
import requests
import json
# to connect to the training resource
from azure.cognitiveservices.vision.customvision.training import CustomVisionTrainingClient

# to connect to the prediction resource
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient

# to send the input files in batch; and to identify the object regions; for the training of the model.
from azure.cognitiveservices.vision.customvision.training.models import ImageFileCreateBatch, ImageFileCreateEntry, Region

# To authenticate the client
from msrest.authentication import ApiKeyCredentials
import os, uuid

# To read the local environment variables and secret keys
import os, dotenv
from dotenv import load_dotenv, find_dotenv

# To read the dataset
import pandas as pd

# to view images
from IPython.display import Image as img

# to process data in batches
```

```
import itertools

# to draw bounding boxes in local output.
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
import numpy as np

print(np.__version__)
print(cv2.__version__)
```

```
1.23.5
4.5.5
```

In [5]:
```
%pip show python-dotenv
```

```
Name: python-dotenvNote: you may need to restart the kernel to use updated packages.

Version: 0.21.0
Summary: Read key-value pairs from a .env file and set them as environment variables
Home-page: https://github.com/theskumar/python-dotenv
Author: Saurabh Kumar
Author-email: me+github@saurabh-kumar.com
License: BSD-3-Clause
Location: C:\Users\Chirag Pc\anaconda3\Lib\site-packages
Requires:
Required-by: anaconda-cloud-auth, webdriver-manager
```

In [93]:
```
%pip show azure-cognitiveservices-vision-customvision
```

```
Name: azure-cognitiveservices-vision-customvision
Version: 3.1.1
Summary: Microsoft Azure Custom Vision Client Library for Python
Home-page: https://github.com/Azure/azure-sdk-for-python
Author: Microsoft Corporation
Author-email: azpysdkhelp@microsoft.com
License: MIT License
Location: C:\Users\Chirag Pc\anaconda3\Lib\site-packages
Requires: azure-common, azure-mgmt-core, msrest
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

may require to install certain packages

- !pip install --user --upgrade opencv-contrib-python
- !pip install opencv-contrib-python
- !pip install opencv-contrib-python==4.5.5.64 --force-reinstall

2. Applying object detection with custom model train on MS azure cloud based platform

- first test our model on testin g image then we apply on object tracking for further testing

In [8]:
```
# 🔷 Load environment variables from .env file
load_dotenv(r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\.env.txt")


# 🔷 Retrieve API credentials from environment variables
ENDPOINT = os.environ.get("VISION_PREDICTION_ENDPOINT")
PREDICTION_KEY = os.environ.get("VISION_PREDICTION_KEY")
PROJECT_ID = os.environ.get("PROJECT_ID")
MODEL_NAME = os.environ.get("MODEL_NAME")


# 🔷 Authenticate with Azure SDK
credentials = ApiKeyCredentials(in_headers={"Prediction-Key": PREDICTION_KEY})
predictor = CustomVisionPredictionClient(ENDPOINT, credentials)

# 🔷 Load Image
image_path = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\down_loadded_image\test\2149891413.jpg"
image = cv2.imread(image_path)
h, w = image.shape[:2]  # Image height and width

# 🔷 Read image as binary
with open(image_path, "rb") as image_file:
    image_data = image_file.read()
```

```python
# 🔷 Get prediction from Azure Custom Vision
response = predictor.detect_image(PROJECT_ID, MODEL_NAME, image_data)

# 🔷 Print full response for debugging
print(json.dumps(response.as_dict(), indent=4))

# 🔷 Store bounding boxes and confidence scores
boxes = []
confidences = []
labels = []

for pred in response.predictions:
    probability = pred.probability  # Confidence score
    if probability < 0.5:  # Filter out low-confidence predictions
        continue

    tag = pred.tag_name  # Object name (Car, Pedestrian, etc.)
    bbox = pred.bounding_box

    # Convert bbox to image coordinates
    x_min = int(bbox.left * w)
    y_min = int(bbox.top * h)
    width = int(bbox.width * w)
    height = int(bbox.height * h)

    boxes.append([x_min, y_min, width, height])
    confidences.append(float(probability))
    labels.append(tag)

# 🔷 Apply Non-Maximum Suppression (NMS)
indices = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)

# 🔷 Draw only the selected bounding boxes
if len(indices) > 0:
    for i in indices.flatten():
        x_min, y_min, width, height = boxes[i]
        x_max, y_max = x_min + width, y_min + height

        # Draw rectangle
        cv2.rectangle(image, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)
```

```
        # Display label with confidence score
        label = f"{labels[i]}: {confidences[i]:.2f}"
        cv2.putText(image, label, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)


# # 🔹 Save or Show image with detections
cv2.imshow("Detected Objects", image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# cv2.imwrite("output.jpg", image)  # Save the image with detections

# 🔹 Convert BGR to RGB (for correct display in matplotlib)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


# 🔹 Display image in Jupyter Notebook cell
plt.figure(figsize=(10, 6))
plt.imshow(image_rgb)
plt.axis("off")
plt.show()

# 🔹 Save image with detections
cv2.imwrite("output.jpg", image)
```

```
{
    "id": "41d980d6-fa8b-4fcf-a83d-8668d7077849",
    "project": "051890d8-5945-465a-90b6-303bb68a9d33",
    "iteration": "310ba808-aa22-4cfc-9386-a1a8c9404769",
    "created": "2025-04-05T13:02:11.102Z",
    "predictions": [
        {
            "probability": 0.99908113,
            "tag_id": "ed6115d1-7578-47de-bfab-b8a3dc4d3a7c",
            "tag_name": "car",
            "bounding_box": {
                "left": 0.4847986,
                "top": 0.25417635,
                "width": 0.39697093,
                "height": 0.48056576
            },
            "tag_type": "Regular"
        },
        {
            "probability": 0.08099399,
            "tag_id": "ed6115d1-7578-47de-bfab-b8a3dc4d3a7c",
            "tag_name": "car",
            "bounding_box": {
                "left": 0.004586862,
                "top": 0.633463,
                "width": 0.04299519,
                "height": 0.1537326
            },
            "tag_type": "Regular"
        },
        {
            "probability": 0.03141625,
            "tag_id": "ed6115d1-7578-47de-bfab-b8a3dc4d3a7c",
            "tag_name": "car",
            "bounding_box": {
                "left": 0.42769995,
                "top": 0.32693008,
                "width": 0.5723001,
                "height": 0.42163613
            },
            "tag_type": "Regular"
```

```
},
{
    "probability": 0.029826589,
    "tag_id": "ed6115d1-7578-47de-bfab-b8a3dc4d3a7c",
    "tag_name": "car",
    "bounding_box": {
        "left": 0.7629865,
        "top": 0.942111,
        "width": 0.23701352,
        "height": 0.057888985
    },
    "tag_type": "Regular"
},
{
    "probability": 0.02420242,
    "tag_id": "ed6115d1-7578-47de-bfab-b8a3dc4d3a7c",
    "tag_name": "car",
    "bounding_box": {
        "left": 0.46805406,
        "top": 0.33221227,
        "width": 0.02935353,
        "height": 0.06395891
    },
    "tag_type": "Regular"
},
{
    "probability": 0.0111910775,
    "tag_id": "ed6115d1-7578-47de-bfab-b8a3dc4d3a7c",
    "tag_name": "car",
    "bounding_box": {
        "left": 0.34210014,
        "top": 0.5838128,
        "width": 0.051045805,
        "height": 0.13166702
    },
    "tag_type": "Regular"
},
{
    "probability": 0.999689,
    "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
    "tag_name": "pedestrian",
```

```
        "bounding_box": {
            "left": 0.16982625,
            "top": 0.03122391,
            "width": 0.29842728,
            "height": 0.8265566
        },
        "tag_type": "Regular"
    },
    {
        "probability": 0.03389998,
        "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
        "tag_name": "pedestrian",
        "bounding_box": {
            "left": 0.778572,
            "top": 0.93728226,
            "width": 0.22142798,
            "height": 0.062717736
        },
        "tag_type": "Regular"
    },
    {
        "probability": 0.022100307,
        "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
        "tag_name": "pedestrian",
        "bounding_box": {
            "left": 0.20344655,
            "top": 0.18876801,
            "width": 0.587801,
            "height": 0.5733116
        },
        "tag_type": "Regular"
    },
    {
        "probability": 0.02056137,
        "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
        "tag_name": "pedestrian",
        "bounding_box": {
            "left": 0.45883098,
            "top": 0.26316962,
            "width": 0.4878985,
            "height": 0.49057654
```

```
    },
    "tag_type": "Regular"
},
{
    "probability": 0.017078834,
    "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
    "tag_name": "pedestrian",
    "bounding_box": {
        "left": 0.0037801384,
        "top": 0.6243588,
        "width": 0.040072948,
        "height": 0.1665873
    },
    "tag_type": "Regular"
},
{
    "probability": 0.016537195,
    "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
    "tag_name": "pedestrian",
    "bounding_box": {
        "left": 0.199446,
        "top": 0.0,
        "width": 0.06993765,
        "height": 0.9339751
    },
    "tag_type": "Regular"
},
{
    "probability": 0.01628908,
    "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
    "tag_name": "pedestrian",
    "bounding_box": {
        "left": 0.34222835,
        "top": 0.5782825,
        "width": 0.05019492,
        "height": 0.13009697
    },
    "tag_type": "Regular"
},
{
    "probability": 0.015923169,
```

```
            "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
            "tag_name": "pedestrian",
            "bounding_box": {
                "left": 0.46807957,
                "top": 0.3307303,
                "width": 0.029191315,
                "height": 0.06629291
            },
            "tag_type": "Regular"
        },
        {
            "probability": 0.013475627,
            "tag_id": "415fbffc-4c72-4392-9d36-bfbfb1efaa21",
            "tag_name": "pedestrian",
            "bounding_box": {
                "left": 0.84998304,
                "top": 0.8602411,
                "width": 0.15001696,
                "height": 0.13975888
            },
            "tag_type": "Regular"
        }
    ]
}
```

Out[8]: True

3. applying prediction on video and storing tracking log on mp4 file of 14 second

Track the movement of at least two objectsacroos frames

In [ ]:

```python
import csv
# Load Video
video_path = r"C:\Users\Chirag Pc\Downloads\1900-151662242_tiny.mp4"
cap = cv2.VideoCapture(video_path)

# Get video properties
fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Resize settings
max_dim = 1200 # Resize to match COCO preprocessing
if frame_width == 0 or frame_height == 0:
    print("Error: Invalid video dimensions")
    cap.release()
    exit()
scale = max_dim / max(frame_width, frame_height)
new_width = int(frame_width * scale)
new_height = int(frame_height * scale)

# Frame skipping factor
SKIP_FRAMES = 2  # Process every 2nd frame

# Open CSV file for logging
csv_filename = "tracking_log.csv"
with open(csv_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Frame", "Label", "X_min", "Y_min", "Width", "Height"])

    # Read first frame
    ret, frame = cap.read()
    if not ret:
        print("Error: Could not read video")
        cap.release()
        exit()

    # Resize frame
    frame = cv2.resize(frame, (new_width, new_height))
    h, w = frame.shape[:2]
```

```python
#  Send first frame to Azure for object detection
_, img_encoded = cv2.imencode('.jpg', frame)
image_data = img_encoded.tobytes()
results = predictor.detect_image(PROJECT_ID, MODEL_NAME, image_data)

# Initialize OpenCV MultiTracker
tracker = cv2.legacy.MultiTracker_create()
labels = []

for pred in results.predictions:
    if pred.probability > 0.3:
        bbox = pred.bounding_box
        x_min = int(bbox.left * w)
        y_min = int(bbox.top * h)
        x_max = int((bbox.left + bbox.width) * w)
        y_max = int((bbox.top + bbox.height) * h)
        box = (x_min, y_min, x_max - x_min, y_max - y_min)

        tracker.add(cv2.legacy.TrackerCSRT_create(), frame, box)
        labels.append(pred.tag_name)

frame_count = 0

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

    frame_count += 1
    if frame_count % SKIP_FRAMES != 0:
        continue

    frame = cv2.resize(frame, (new_width, new_height))
    success, boxes = tracker.update(frame)

    for i, new_box in enumerate(boxes):
        x, y, w, h = map(int, new_box)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, labels[i], (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

        # Write tracking data to CSV
```

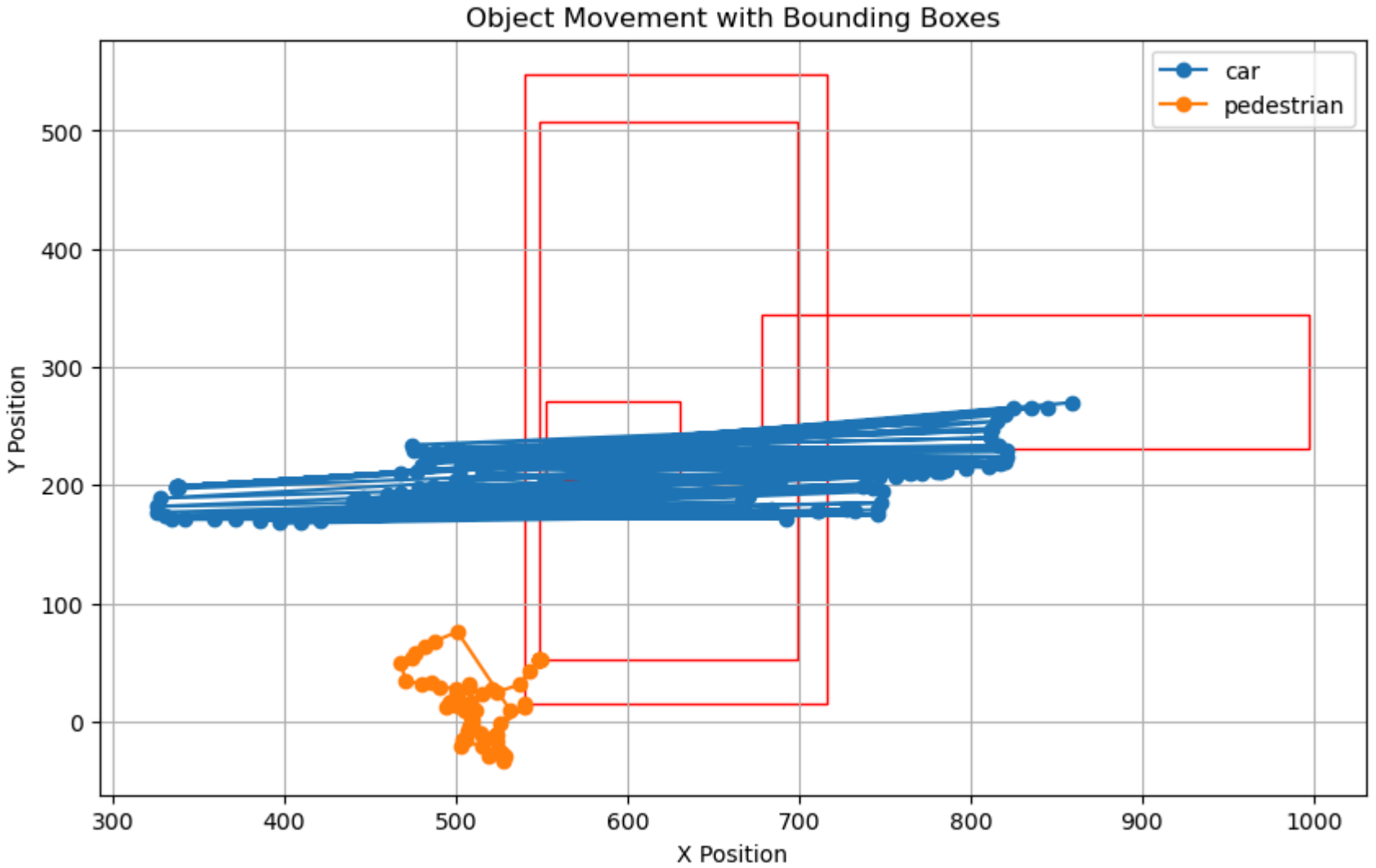
```
            writer.writerow([frame_count, labels[i], x, y, w, h])

        cv2.imshow("Object Tracking", frame)
        if cv2.waitKey(int(1000 / fps)) & 0xFF == ord("q"):
            break

cap.release()
cv2.destroyAllWindows()
print(f"Tracking data saved in {csv_filename}")
```

visualizing their location acrross frame

In [11]:

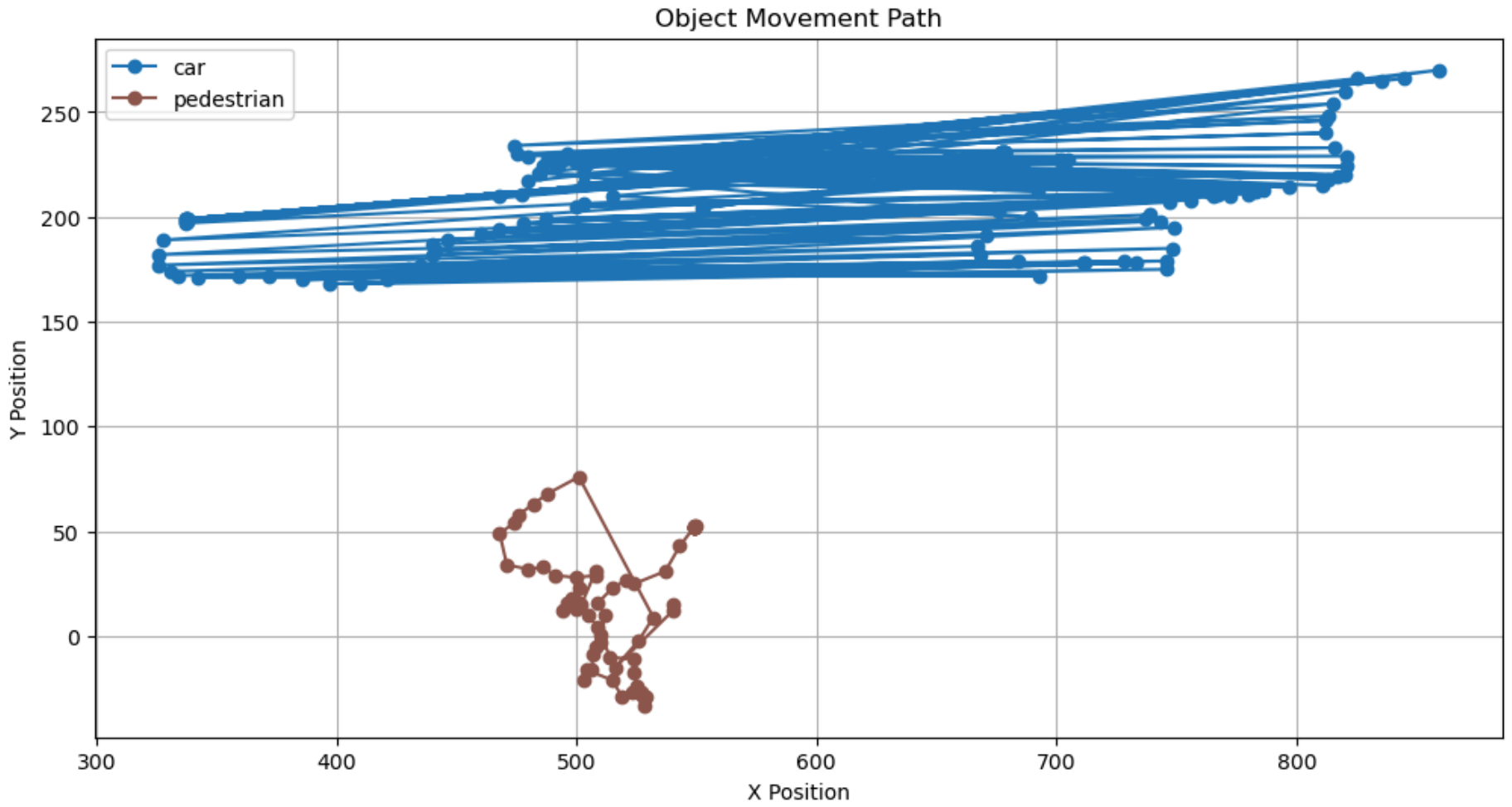
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Load the tracking log
df = pd.read_csv(r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\tracking_log.csv")

# Create a figure for visualization
fig, ax = plt.subplots(figsize=(10, 6))

# Plot each object's movement and bounding boxes
for label in df["Label"].unique():
    obj_data = df[df["Label"] == label]

    # Plot movement path
    ax.plot(obj_data["X_min"], obj_data["Y_min"], marker="o", linestyle="-", label=label)

    # Draw bounding boxes for the first and last frames
    for _, row in obj_data.iloc[[0, -1]].iterrows():
        rect = patches.Rectangle(
            (row["X_min"], row["Y_min"]), row["Width"], row["Height"],
            linewidth=1, edgecolor="r", facecolor="none"
        )
        ax.add_patch(rect)

# Labels and display settings
plt.xlabel("X Position")
```

```
plt.ylabel("Y Position")
plt.title("Object Movement with Bounding Boxes")
plt.legend()
plt.grid(True)
plt.show()
```



In [12]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

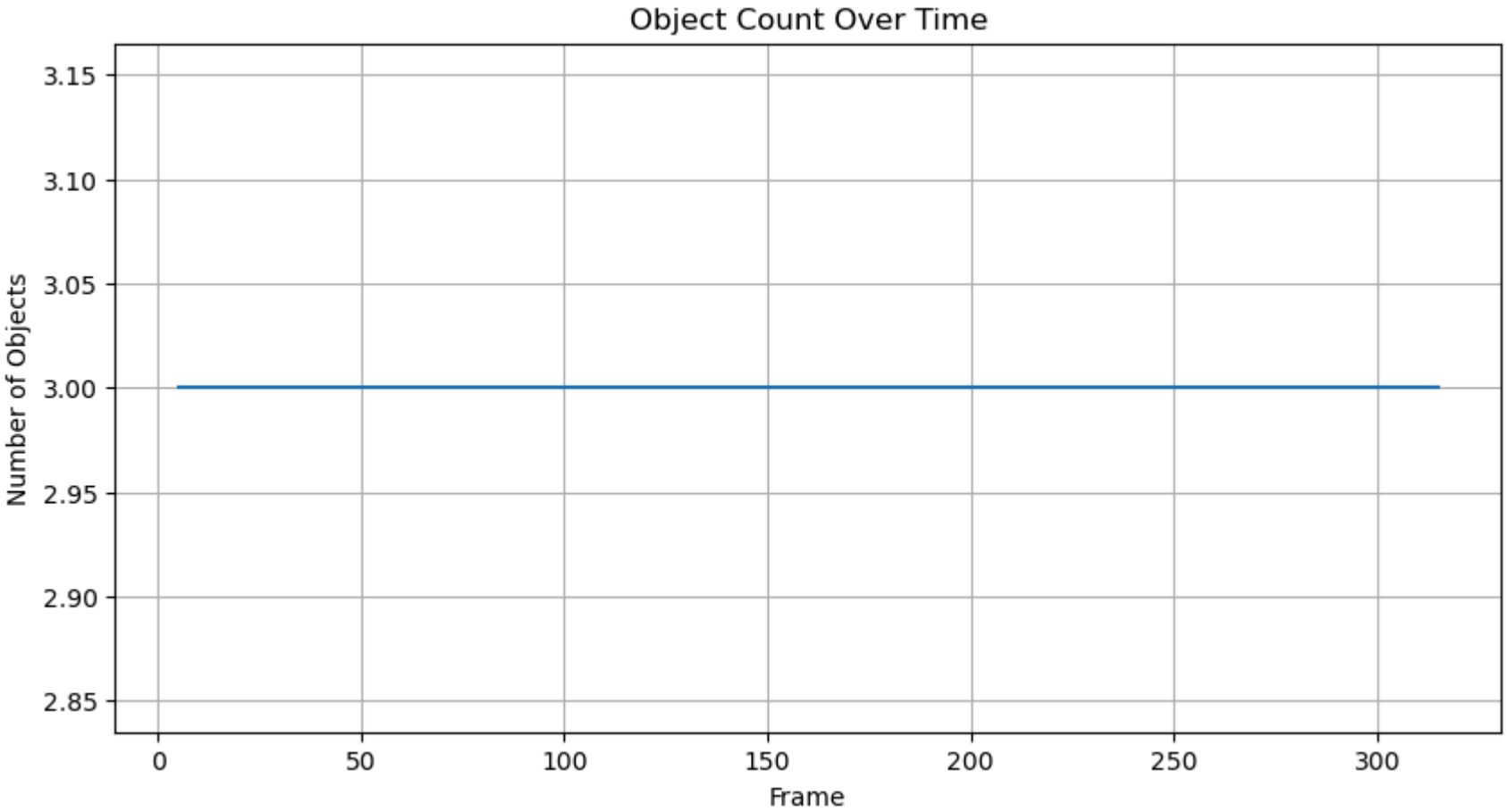
```python
# Load tracking data
df = pd.read_csv(r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\tracking_log.csv")

# Rename columns to match your dataset
df.rename(columns={"X": "X_min", "Y": "Y_min", "width": "Width", "height": "Height"}, inplace=True)

# Get unique labels (objects)
labels = df["Label"].unique()
colors = plt.colormaps["tab10"]  # Get colormap

# Assign unique colors to each object
color_map = {label: colors(i / len(labels)) for i, label in enumerate(labels)}

# Initialize plot
plt.figure(figsize=(12, 6))

# Plot object movement
for label in labels:
    obj_data = df[df["Label"] == label]
    plt.plot(obj_data["X_min"], obj_data["Y_min"], marker="o", linestyle="-", color=color_map[label], label=label)

plt.xlabel("X Position")
plt.ylabel("Y Position")
plt.title("Object Movement Path")
plt.legend()
plt.grid(True)
plt.show()
```

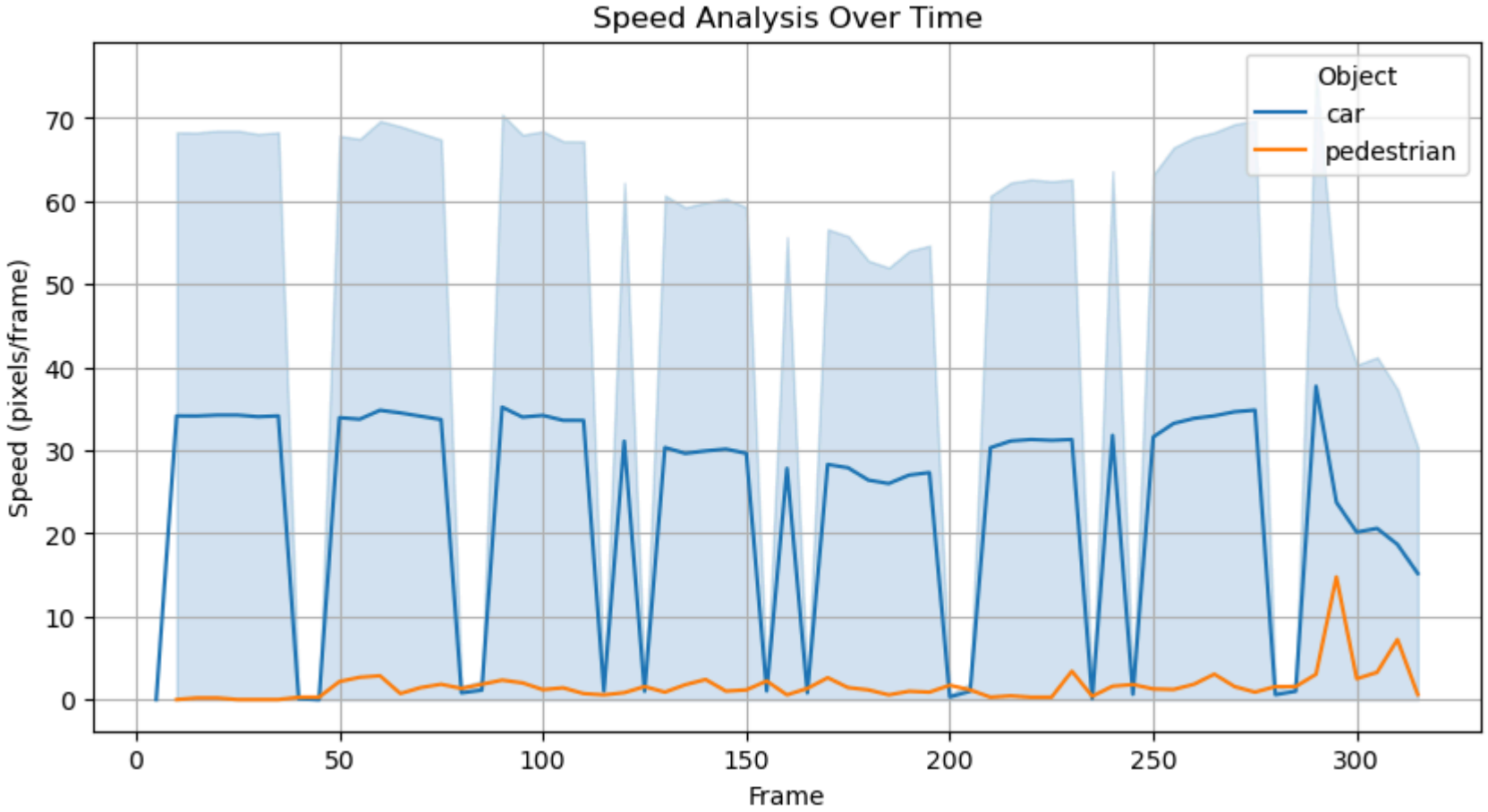


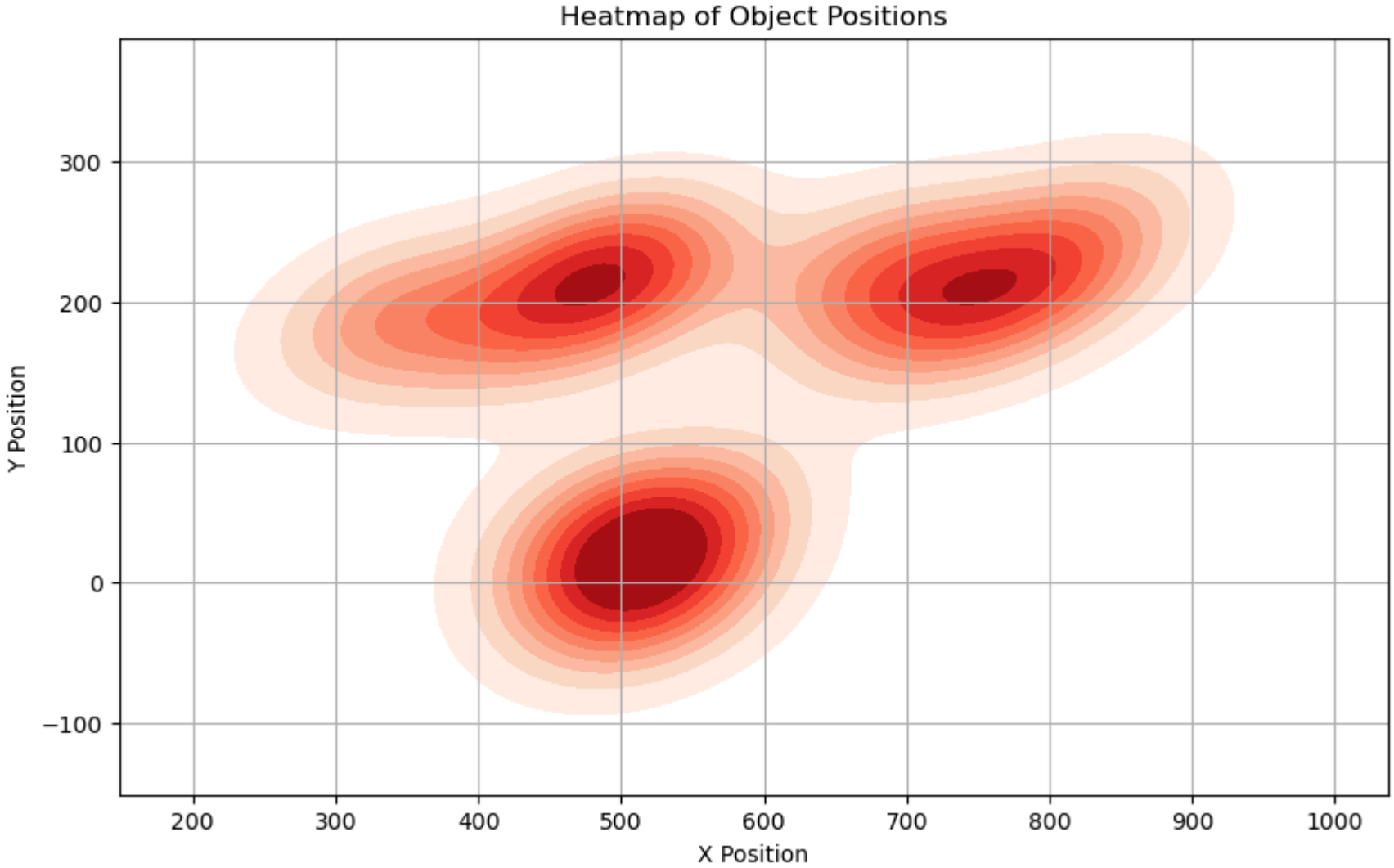
In [13]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Load tracking log
file_path = "object_tracking_log.csv"  # Update with actual path
df = pd.read_csv(r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\tracking_log.csv")

# 1 Object Count Over Time
```

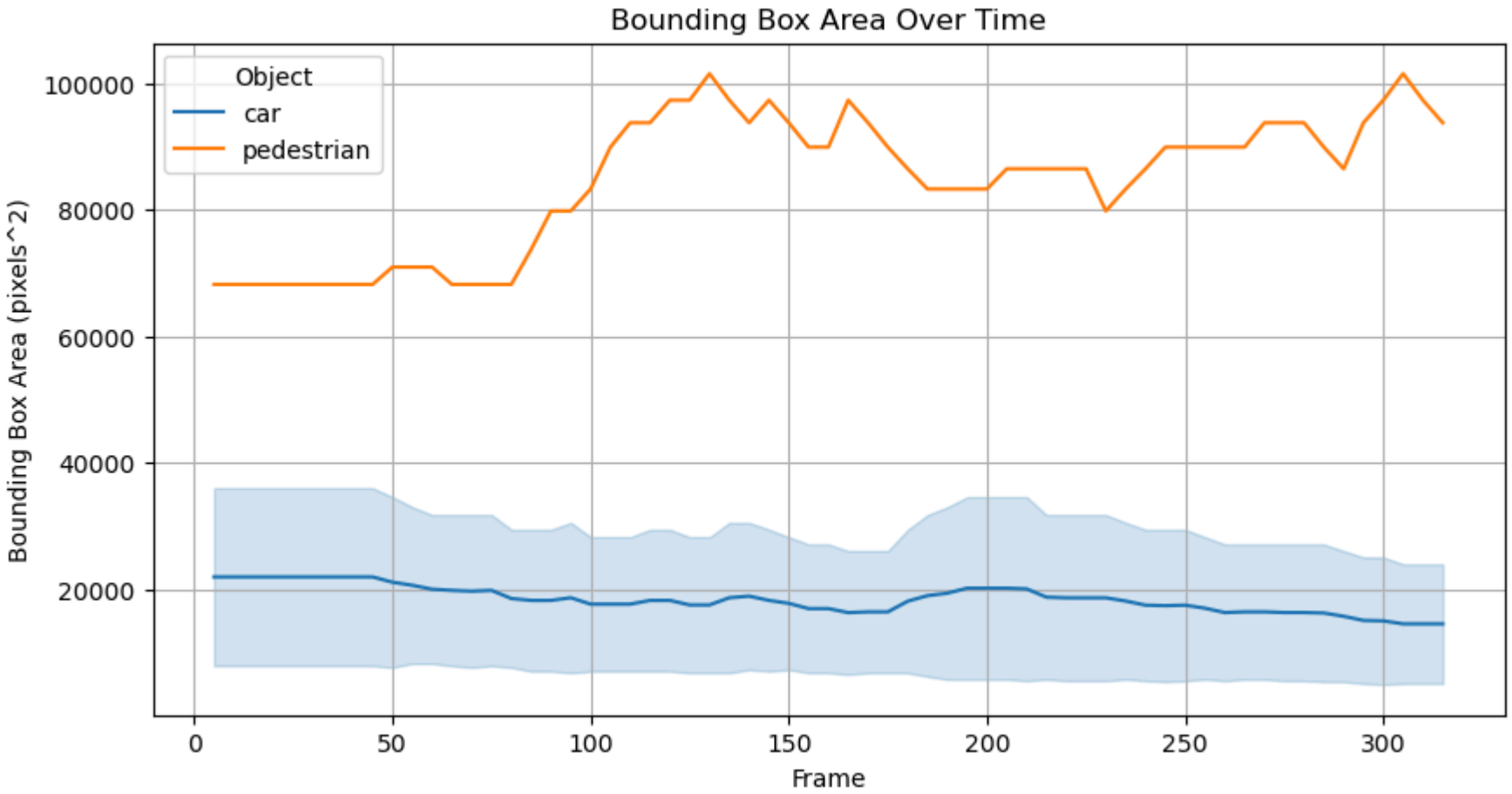
```python
plt.figure(figsize=(10, 5))
df.groupby("Frame")["Label"].count().plot()
plt.xlabel("Frame")
plt.ylabel("Number of Objects")
plt.title("Object Count Over Time")
plt.grid()
plt.show()

# 2️⃣ Speed Analysis (Distance vs. Time)
def calculate_speed(df):
    speeds = []
    for label in df["Label"].unique():
        obj_data = df[df["Label"] == label].sort_values("Frame")
        prev_x, prev_y, prev_frame = None, None, None
        for _, row in obj_data.iterrows():
            if prev_x is not None:
                distance = np.sqrt((row["X_min"] - prev_x) ** 2 + (row["Y_min"] - prev_y) ** 2)
                frame_diff = row["Frame"] - prev_frame
                speed = distance / frame_diff if frame_diff > 0 else 0
                speeds.append([row["Frame"], label, speed])
            prev_x, prev_y, prev_frame = row["X_min"], row["Y_min"], row["Frame"]
    return pd.DataFrame(speeds, columns=["Frame", "Label", "Speed"])

speed_df = calculate_speed(df)
plt.figure(figsize=(10, 5))
sns.lineplot(data=speed_df, x="Frame", y="Speed", hue="Label")
plt.xlabel("Frame")
plt.ylabel("Speed (pixels/frame)")
plt.title("Speed Analysis Over Time")
plt.legend(title="Object")
plt.grid()
plt.show()

# 3️⃣ Heatmap of Object Positions
plt.figure(figsize=(10, 6))
sns.kdeplot(x=df["X_min"], y=df["Y_min"], cmap="Reds", fill=True)
plt.xlabel("X Position")
plt.ylabel("Y Position")
plt.title("Heatmap of Object Positions")
plt.grid()
plt.show()
```

```python
# 4️⃣ Bounding Box Area Over Time
df["Area"] = df["Width"] * df["Height"]
plt.figure(figsize=(10, 5))
sns.lineplot(data=df, x="Frame", y="Area", hue="Label")
plt.xlabel("Frame")
plt.ylabel("Bounding Box Area (pixels^2)")
plt.title("Bounding Box Area Over Time")
plt.legend(title="Object")
plt.grid()
plt.show()
```




Speed Analysis Over Time
Object
car
pedestrian
Speed (pixels/frame)
Frame
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300


Heatmap of Object Positions
Y Position
X Position
300
200
100
0
−100
200
300
400
500
600
700
800
900
1000



2.3 Performance Evaluation

- Evaluate how well the Azure model detects and tracks objects.
- Discuss limitations (e.g., accuracy, speed, false positives).
- Suggest potential improvements (e.g., fine-tuning or using a different model)

creating prediction on each frame of video and store its boundung box prediction

In [22]:

```python
# 🔷 Image folder path
image_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\down_loadded_image\ground truth\train\coco_image"
output_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\predictions"  # Folder to save output images
output_csv = os.path.join(output_folder, "azure_predictions.csv")

# Create output folder if not exists
os.makedirs(output_folder, exist_ok=True)

# 🔷 Process each image in the folder
results = []
image_files = [f for f in os.listdir(image_folder) if f.lower().endswith((".jpg", ".jpeg", ".png"))]

for idx, file_name in enumerate(image_files):
    image_path = os.path.join(image_folder, file_name)
    output_image_path = os.path.join(output_folder, file_name)

    # Load image for dimensions
    image = cv2.imread(image_path)
    if image is None:
        print(f"❌ Error loading {file_name}")
        continue
    h, w = image.shape[:2]  # Image dimensions

    # Read image as binary
    with open(image_path, "rb") as image_file:
        image_data = image_file.read()

    # 🔷 Send request to Azure Custom Vision API
    try:
        response = predictor.detect_image(PROJECT_ID, MODEL_NAME, image_data)
    except Exception as e:
        print(f"❌ Error processing {file_name}: {e}")
        continue

    # 🔷 Process predictions
    for pred in response.predictions:
        tag = pred.tag_name  # Object class
        probability = pred.probability  # Confidence score
        bbox = pred.bounding_box
```

```
        # Convert normalized bbox to image coordinates
        x_min = int(bbox.left * w)
        y_min = int(bbox.top * h)
        width = int(bbox.width * w)
        height = int(bbox.height * h)
        area = width * height

        # Store results
        results.append([idx, file_name, tag, x_min, y_min, width, height, area, probability])

        # Draw bounding box
        cv2.rectangle(image, (x_min, y_min), (x_min + width, y_min + height), (0, 255, 0), 2)  # Green Box
        label = f"{tag} ({probability:.2f})"
        cv2.putText(image, label, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Save output image with bounding boxes
    cv2.imwrite(output_image_path, image)

# 🔷 Save results to CSV
df = pd.DataFrame(results, columns=["img_id", "file_name", "Label", "X_min", "Y_min", "Width", "Height", "Area", "Confidence"]
df.to_csv(output_csv, index=False)
print(f"✅ Predictions saved to {output_csv}")
print(f"✅ Processed images saved in {output_folder}")
```

```
✅ Predictions saved to E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\predictions\azure_predictions.csv
✅ Processed images saved in E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\predictions
```

similarly creatin fcsv file of coco out put for calculaing IOU and evaluation

In [13]:

```
# 🔷 Paths
coco_json_path = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\down_loadded_image\ground truth\train\_annotations.coc
image_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\down_loadded_image\ground truth\train\coco_image"
output_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\coco_predictions"
output_csv = os.path.join(output_folder, "coco_annotations.csv")

# Create output folder if not exists
os.makedirs(output_folder, exist_ok=True)

# 🔷 Load COCO JSON
with open(coco_json_path, "r") as f:
```

```
    coco_data = json.load(f)

# 🔹 Extract relevant data
images = {img["id"]: img["file_name"] for img in coco_data["images"]}
categories = {cat["id"]: cat["name"] for cat in coco_data["categories"]}

# 🔹 Dictionary to store images before writing them
image_results = {}
results = []

for ann in coco_data["annotations"]:
    img_id = ann["image_id"]
    label = categories.get(ann["category_id"], "Unknown")  # Handle missing categories
    x_min, y_min, width, height = map(int, ann["bbox"])
    confidence = ann.get("score", 1.0)  # Some COCO files may not have confidence scores

    # Get file name
    file_name = images.get(img_id)
    if not file_name:
        print(f"❌ Image ID {img_id} not found in COCO dataset.")
        continue

    image_path = os.path.join(image_folder, file_name)
    if not os.path.exists(image_path):
        print(f"❌ Missing image: {file_name}")
        continue

    # Load image if not already loaded
    if file_name not in image_results:
        image = cv2.imread(image_path)
        if image is None:
            print(f"❌ Error loading {file_name}")
            continue
        image_results[file_name] = image  # Store image

    # Draw bounding box
    cv2.rectangle(image_results[file_name], (x_min, y_min), (x_min + width, y_min + height), (0, 0, 255), 2)
    label_text = f"{label} ({confidence:.2f})"
    cv2.putText(image_results[file_name], label_text, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    # Store annotation results
```

```
    results.append([img_id, file_name, label, x_min, y_min, width, height, confidence])

# 🔷 Save processed images
for file_name, img in image_results.items():
    output_image_path = os.path.join(output_folder, file_name)
    cv2.imwrite(output_image_path, img)

# 🔷 Save results to CSV
df = pd.DataFrame(results, columns=["img_id", "file_name", "Label", "X_min", "Y_min", "Width", "Height", "Confidence"])
df.to_csv(output_csv, index=False)

print(f"✅ COCO annotations saved to {output_csv}")
print(f"✅ Processed images saved in {output_folder}")
```

```
✅ COCO annotations saved to E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\coco_predictions\coco_annotations.csv
✅ Processed images saved in E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\coco_predictions
```

Evaluating model

In [15]:

```
import os
import cv2
import pandas as pd
import numpy as np

# 🔷 Function to Compute IoU
def calculate_iou(box1, box2):
    x1_min, y1_min, w1, h1 = box1
    x2_min, y2_min, w2, h2 = box2

    x1_max, y1_max = x1_min + w1, y1_min + h1
    x2_max, y2_max = x2_min + w2, y2_min + h2

    inter_x_min = max(x1_min, x2_min)
    inter_y_min = max(y1_min, y2_min)
    inter_x_max = min(x1_max, x2_max)
    inter_y_max = min(y1_max, y2_max)

    inter_area = max(0, inter_x_max - inter_x_min) * max(0, inter_y_max - inter_y_min)
    area1 = w1 * h1
```

```
    area2 = w2 * h2
    union_area = area1 + area2 - inter_area

    return inter_area / union_area if union_area > 0 else 0.0

# 🔹 Paths
pred_csv_path = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\predictions\azure_predictions.csv"  # Azure Predictions
coco_csv_path = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\coco_predictions\coco_annotations.csv"  # Ground Truth
image_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\down_loadded_image\ground truth\train\coco_image"  # Ima
output_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\evaluated_images"  # Save Results
output_iou_csv = os.path.join(output_folder, "iou_evaluation.csv")

# Create output folder if not exists
os.makedirs(output_folder, exist_ok=True)

# 🔹 Load Data
df_pred = pd.read_csv(pred_csv_path)
df_coco = pd.read_csv(coco_csv_path)

# Ensure file names are strings
df_pred["file_name"] = df_pred["file_name"].astype(str)
df_coco["file_name"] = df_coco["file_name"].astype(str)

# 🔹 Match Predictions to Ground Truth (Selecting Best IoU)
final_matches = []
iou_scores = []

# Group by image to handle multiple objects per image
grouped_coco = df_coco.groupby("file_name")
grouped_pred = df_pred.groupby("file_name")

for file_name, coco_group in grouped_coco:
    preds = grouped_pred.get_group(file_name) if file_name in grouped_pred.groups else pd.DataFrame()

    for _, coco_row in coco_group.iterrows():
        img_id = coco_row["img_id"]
        gt_box = [coco_row["X_min"], coco_row["Y_min"], coco_row["Width"], coco_row["Height"]]
        best_iou = 0
        best_pred_row = None

        for _, pred_row in preds.iterrows():
```

```
            pred_box = [pred_row["X_min"], pred_row["Y_min"], pred_row["Width"], pred_row["Height"]]
            iou = calculate_iou(gt_box, pred_box)

            if iou > best_iou:
                best_iou = iou
                best_pred_row = pred_row

        if best_pred_row is not None:
            final_matches.append([
                img_id, file_name, coco_row["Label"],
                gt_box[0], gt_box[1], gt_box[2], gt_box[3],  # GT BBox
                best_pred_row["X_min"], best_pred_row["Y_min"],
                best_pred_row["Width"], best_pred_row["Height"], best_pred_row["Confidence"], best_iou
            ])
            iou_scores.append(best_iou)
        else:
            # If no matching prediction, record ground truth with NaN values
            final_matches.append([
                img_id, file_name, coco_row["Label"],
                gt_box[0], gt_box[1], gt_box[2], gt_box[3],  # GT BBox
                np.nan, np.nan, np.nan, np.nan, np.nan, 0  # No prediction found
            ])
            iou_scores.append(0)

# 🔷 Save Matched Predictions
df_matched = pd.DataFrame(final_matches, columns=[
    "img_id", "file_name", "Label", "GT_X_min", "GT_Y_min", "GT_Width", "GT_Height",
    "Pred_X_min", "Pred_Y_min", "Pred_Width", "Pred_Height", "Confidence", "IoU"
])
df_matched.to_csv(output_iou_csv, index=False)

# 🔷 Compute Evaluation Metrics
mean_iou = np.mean(iou_scores)
precision = len(df_matched[df_matched["IoU"] > 0]) / len(df_pred)  # TP / (TP + FP)
recall = len(df_matched[df_matched["IoU"] > 0]) / len(df_coco)  # TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

# 🔷 Draw Bounding Boxes on Images
for file_name, group in df_matched.groupby("file_name"):
    img_path = os.path.join(image_folder, file_name)
    if not os.path.exists(img_path):
```

```
        continue

    image = cv2.imread(img_path)

    for _, row in group.iterrows():
        if pd.isna(row["Pred_X_min"]):
            continue  # Skip if no prediction

        # Ground Truth (Red)
        cv2.rectangle(image, (int(row["GT_X_min"]), int(row["GT_Y_min"])),
                      (int(row["GT_X_min"] + row["GT_Width"]), int(row["GT_Y_min"] + row["GT_Height"])),
                      (0, 0, 255), 2)
        cv2.putText(image, "GT: " + row["Label"], (int(row["GT_X_min"]), int(row["GT_Y_min"]) - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

        # Prediction (Green)
        cv2.rectangle(image, (int(row["Pred_X_min"]), int(row["Pred_Y_min"])),
                      (int(row["Pred_X_min"] + row["Pred_Width"]), int(row["Pred_Y_min"] + row["Pred_Height"])),
                      (0, 255, 0), 2)
        cv2.putText(image, f"Pred ({row['Confidence']:.2f})", (int(row["Pred_X_min"]), int(row["Pred_Y_min"]) - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Save Image
    output_img_path = os.path.join(output_folder, file_name)
    cv2.imwrite(output_img_path, image)

print(f"✅ IoU Calculation Completed!")
print(f"📊 Mean IoU: {mean_iou:.4f}")
print(f"🎯 Precision: {precision:.4f}")
print(f"🔍 Recall: {recall:.4f}")
print(f"⭐ F1 Score: {f1_score:.4f}")
print(f"✅ Evaluated images saved in: {output_folder}")
print(f"✅ IoU evaluation saved at: {output_iou_csv}")
```

```
✅ IoU Calculation Completed!
📊 Mean IoU: 0.6854
🎯 Precision: 0.2029
🔍 Recall: 0.8750
⭐ F1 Score: 0.3294
✅ Evaluated images saved in: E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\evaluated_images
✅ IoU evaluation saved at: E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\evaluated_images\iou_evaluation.csv
```

In [18]:

```python
import os
import cv2
import pandas as pd
import numpy as np

# 🔷 Function to Compute IoU
def calculate_iou(box1, box2):
    x1_min, y1_min, w1, h1 = box1
    x2_min, y2_min, w2, h2 = box2

    x1_max, y1_max = x1_min + w1, y1_min + h1
    x2_max, y2_max = x2_min + w2, y2_min + h2

    inter_x_min = max(x1_min, x2_min)
    inter_y_min = max(y1_min, y2_min)
    inter_x_max = min(x1_max, x2_max)
    inter_y_max = min(y1_max, y2_max)

    inter_area = max(0, inter_x_max - inter_x_min) * max(0, inter_y_max - inter_y_min)
    area1 = w1 * h1
    area2 = w2 * h2
    union_area = area1 + area2 - inter_area

    return inter_area / union_area if union_area > 0 else 0.0

# 🔷 Paths
pred_csv_path = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\predictions\azure_predictions.csv"
coco_csv_path = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\coco_predictions\coco_annotations.csv"
image_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\down_loadded_image\ground truth\train\coco_image"
output_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\evaluated_images"
output_iou_csv = os.path.join(output_folder, "iou_evaluation.csv")

# Create output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)

# 🔷 Load Data
df_pred = pd.read_csv(pred_csv_path)
df_coco = pd.read_csv(coco_csv_path)

# Ensure correct types
```

```python
df_pred["file_name"] = df_pred["file_name"].astype(str)
df_coco["file_name"] = df_coco["file_name"].astype(str)

# 🔷 IoU Matching
matches = []
used_pred_indices = set()

for _, gt in df_coco.iterrows():
    gt_box = [gt["X_min"], gt["Y_min"], gt["Width"], gt["Height"]]
    file_preds = df_pred[df_pred["file_name"] == gt["file_name"]]

    best_iou = 0
    best_pred = None
    best_idx = None

    for idx, pred in file_preds.iterrows():
        if idx in used_pred_indices:
            continue

        pred_box = [pred["X_min"], pred["Y_min"], pred["Width"], pred["Height"]]
        iou = calculate_iou(gt_box, pred_box)

        if iou > best_iou:
            best_iou = iou
            best_pred = pred
            best_idx = idx

    if best_pred is not None:
        used_pred_indices.add(best_idx)
        matches.append({
            "img_id": gt["img_id"],
            "file_name": gt["file_name"],
            "Label": gt["Label"],
            "GT_X_min": gt["X_min"], "GT_Y_min": gt["Y_min"],
            "GT_Width": gt["Width"], "GT_Height": gt["Height"],
            "Pred_X_min": best_pred["X_min"], "Pred_Y_min": best_pred["Y_min"],
            "Pred_Width": best_pred["Width"], "Pred_Height": best_pred["Height"],
            "Confidence": best_pred["Confidence"],
            "IoU": best_iou
        })
    else:
```

```
        matches.append({
            "img_id": gt["img_id"],
            "file_name": gt["file_name"],
            "Label": gt["Label"],
            "GT_X_min": gt["X_min"], "GT_Y_min": gt["Y_min"],
            "GT_Width": gt["Width"], "GT_Height": gt["Height"],
            "Pred_X_min": np.nan, "Pred_Y_min": np.nan,
            "Pred_Width": np.nan, "Pred_Height": np.nan,
            "Confidence": np.nan, "IoU": 0.0
        })

# 🔷 Create Evaluation DataFrame
df_eval = pd.DataFrame(matches)
df_eval.to_csv(output_iou_csv, index=False)

# 🔷 Compute Confusion Matrix Metrics
iou_threshold = 0.5
TP = np.sum(df_eval["IoU"] >= iou_threshold)
FN = np.sum(df_eval["IoU"] < iou_threshold)
FP = len(df_pred) - TP

precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0
f1_score = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0
mean_iou = df_eval["IoU"].mean()

# 🔷 Draw Bounding Boxes
for file_name, group in df_eval.groupby("file_name"):
    image_path = os.path.join(image_folder, file_name)
    if not os.path.exists(image_path):
        continue
    image = cv2.imread(image_path)

    for _, row in group.iterrows():
        # Ground Truth
        gt_pt1 = (int(row["GT_X_min"]), int(row["GT_Y_min"]))
        gt_pt2 = (int(row["GT_X_min"] + row["GT_Width"]), int(row["GT_Y_min"] + row["GT_Height"]))
        cv2.rectangle(image, gt_pt1, gt_pt2, (0, 0, 255), 2)
        cv2.putText(image, "GT: " + row["Label"], (gt_pt1[0], gt_pt1[1] - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
```

```
        # Prediction
        if not pd.isna(row["Pred_X_min"]):
            pred_pt1 = (int(row["Pred_X_min"]), int(row["Pred_Y_min"]))
            pred_pt2 = (int(row["Pred_X_min"] + row["Pred_Width"]), int(row["Pred_Y_min"] + row["Pred_Height"]))
            cv2.rectangle(image, pred_pt1, pred_pt2, (0, 255, 0), 2)
            cv2.putText(image, f"Pred ({row['Confidence']:.2f})", (pred_pt1[0], pred_pt1[1] - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)

    # Save evaluated image
    output_img_path = os.path.join(output_folder, file_name)
    cv2.imwrite(output_img_path, image)

# 🔷 Final Output
print(f"✅ IoU Evaluation Complete")
print(f"📊 Mean IoU: {mean_iou:.4f}")
print(f"🎯 Precision: {precision:.4f}")
print(f"🔍 Recall: {recall:.4f}")
print(f"⭐ F1 Score: {f1_score:.4f}")
print(f"📂 Results saved to: {output_iou_csv}")
```

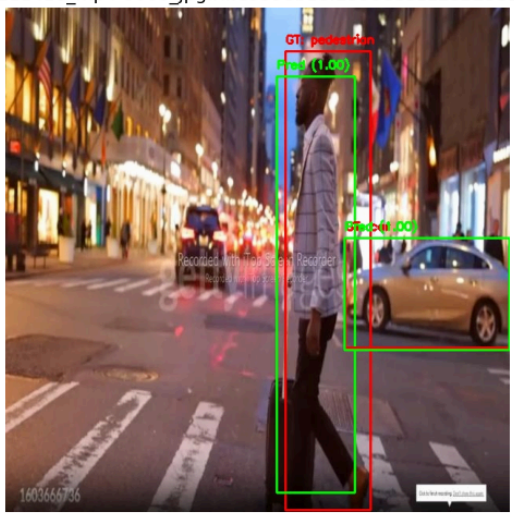
```
✅ IoU Evaluation Complete
📊 Mean IoU: 0.7016
🎯 Precision: 0.1916
🔍 Recall: 0.8542
⭐ F1 Score: 0.3130
📂 Results saved to: E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\evaluated_images\iou_evaluation.csv
```

In [17]:

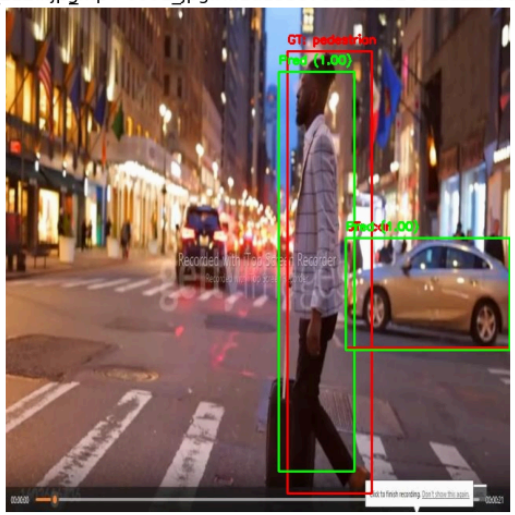
```
import math
# 🔷 Path to evaluated images folder
evaluated_images_folder = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\evaluated_images"

# 🔷 List all image files (ignore CSV files)
image_files = [f for f in os.listdir(evaluated_images_folder) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]

# 🔷 Set number of images to display
num_images = min(len(image_files), 22)  # Display max 12 images
images_per_row = 3  # Number of images per row
num_rows = math.ceil(num_images / images_per_row)  # Calculate required rows

plt.figure(figsize=(15, num_rows * 5))  # Adjust figure size
```

```
for i, img_name in enumerate(image_files[:num_images]):
    img_path = os.path.join(evaluated_images_folder, img_name)
    image = cv2.imread(img_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB for correct display

    plt.subplot(num_rows, images_per_row, i + 1)
    plt.imshow(image)
    plt.axis("off")
    plt.title(img_name)

plt.tight_layout()
plt.show()
```

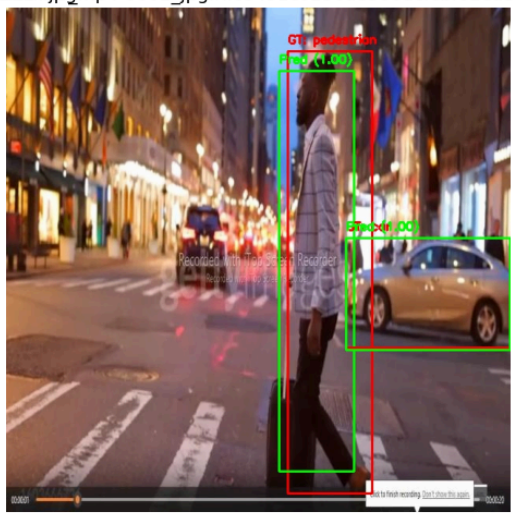
20250402_211416_mp4-0000_jpg.rf.fe9598b18f2598eee12[illegible].jpg



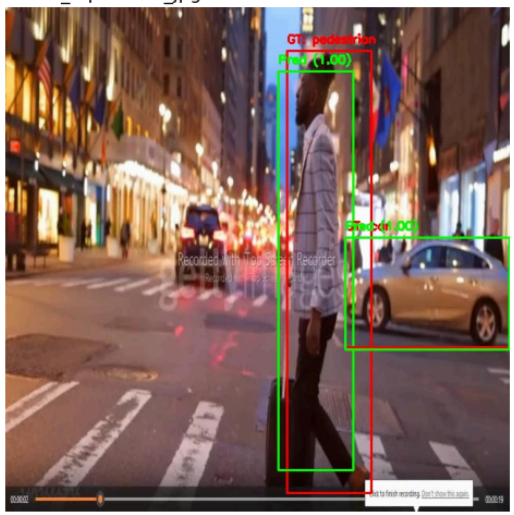
20250402_211416_mp4-0001_jpg.rf.f2a19387030af9324ce[illegible].jpg



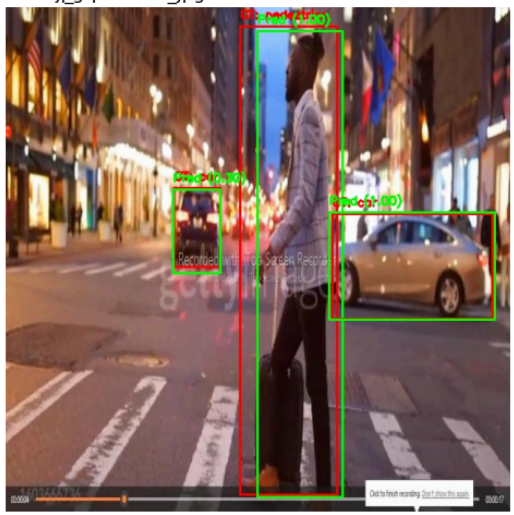
20250402_211416_mp4-0002_jpg.rf.c8491c0230ae306dcfcbf62ea3116d8a.jpg



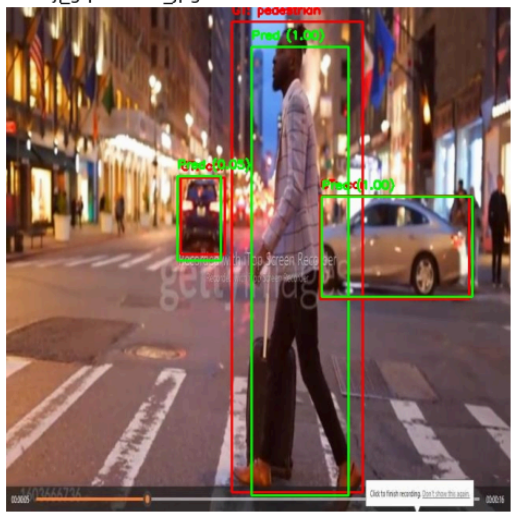
20250402_211416_mp4-0003_jpg.rf.71d477ddcba5d616dc[illegible].jpg



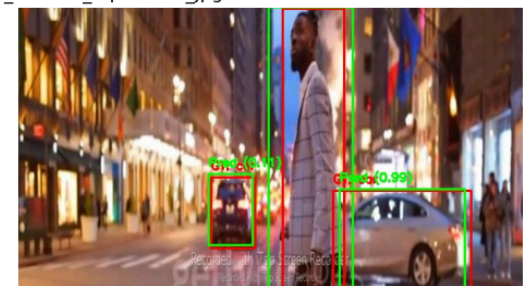
20250402_211416_mp4-0004_jpg.rf.eabcc9d7681731367f[illegible].jpg



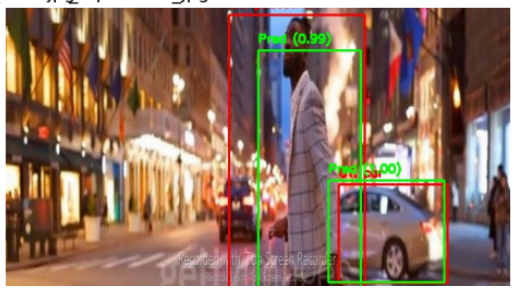
20250402_211416_mp4-0005_jpg.rf.e386798d693a3285784782c0d6e02a8e.jpg



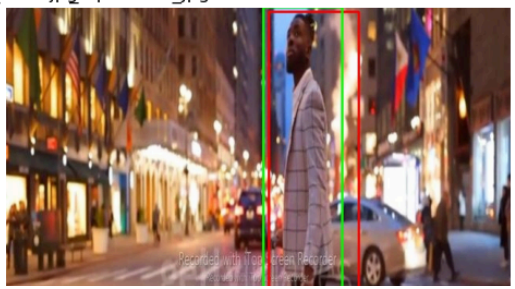
20250402_211416_mp4-0006_jpg.rf.048fbf39ba36fa101290[illegible].jpg



20250402_211416_mp4-0007_jpg.rf.a0b737a35dfe7d8e08f[illegible].jpg



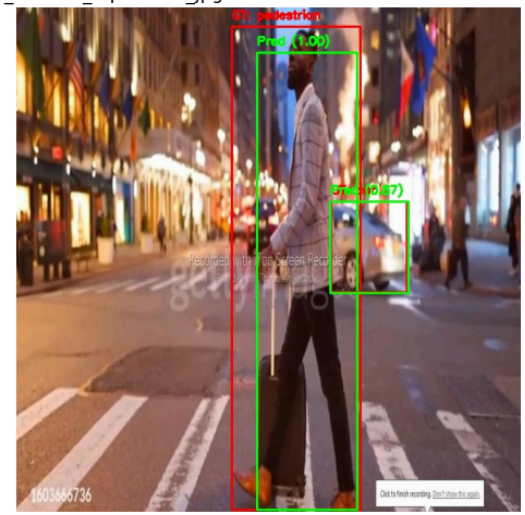
20250402_211416_mp4-0008_jpg.rf.251142676e43eaf6fb3a197cab59262f.jpg



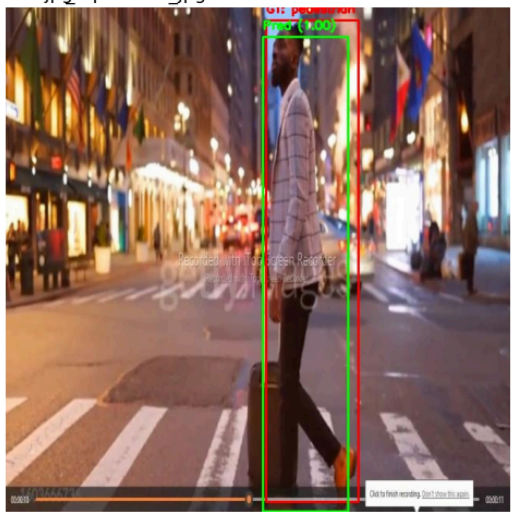
20250402_211416_mp4-0009_jpg.rf.23f6c5be25acdef9f55[illegible]

[illegible]mp4-0010_jpg.rf.9993af059559684c70e[illegible]

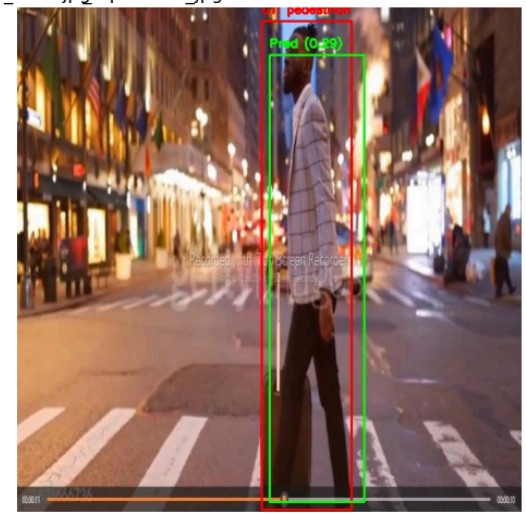
[illegible]mp4-0011_jpg.rf.f36cd3759c3ee6a0af1ece8d14356afc.jpg



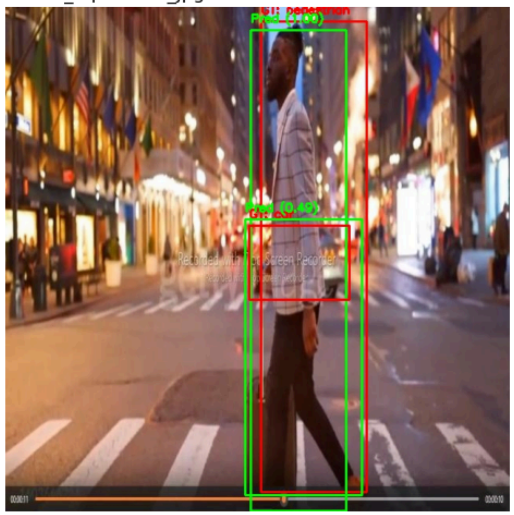
20250402_211416_mp4-0012_jpg.rf.c000c627fb3b8c4575ba[illegible]



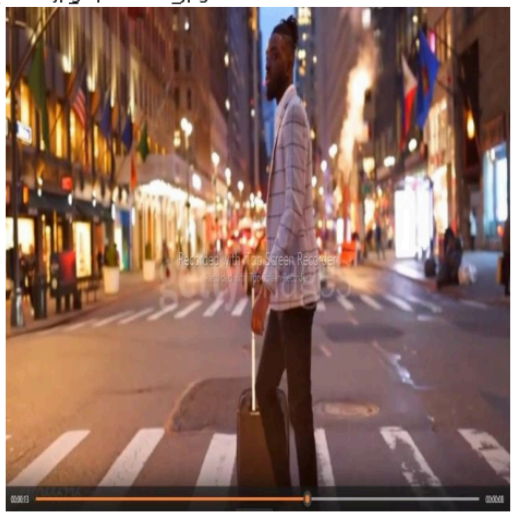
[illegible]mp4-0013_jpg.rf.552f0388feb78b5beb[illegible]



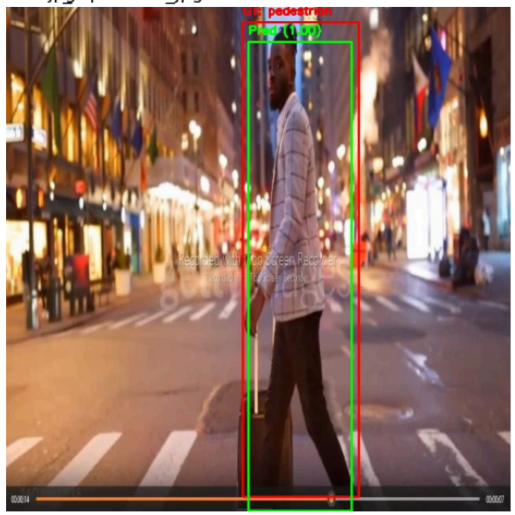
[illegible]mp4-0014_jpg.rf.da98da65935b339b61703ef87489c131.jpg



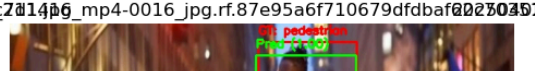
20250402_211416_mp4-0015_jpg.rf.4729f48ffff50527c4ba[illegible]

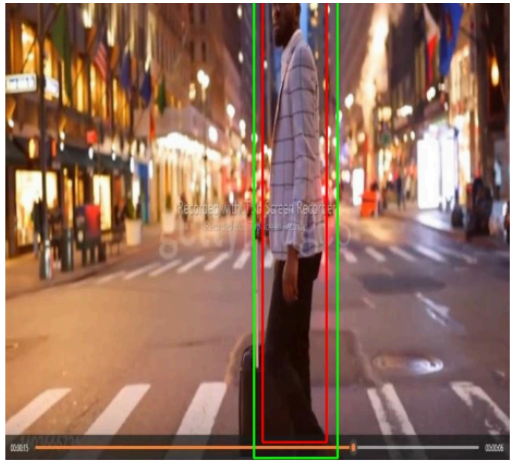


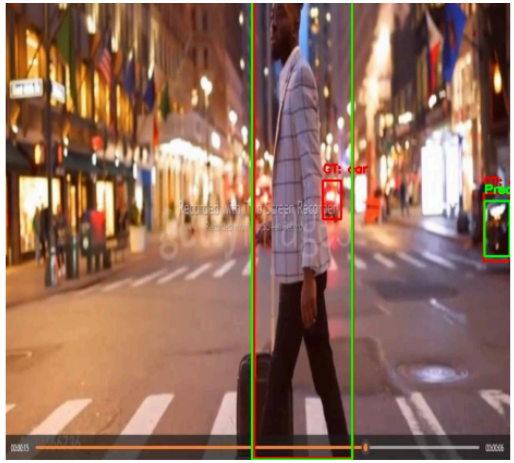
[illegible]mp4-0016_jpg.rf.87e95a6f710679dfdbaf[illegible]

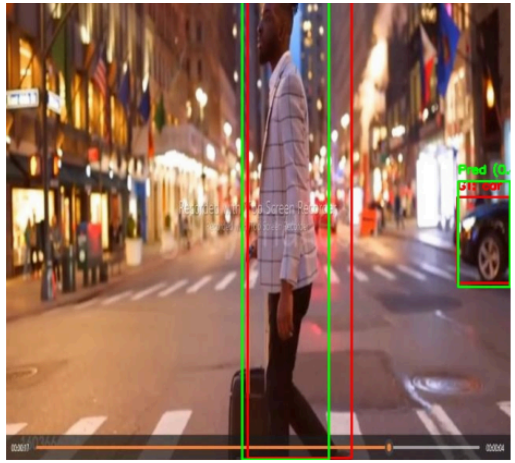


[illegible]mp4-0017_jpg.rf.f53f5cecb638f512257f5f5bd0fb3f4c.jpg

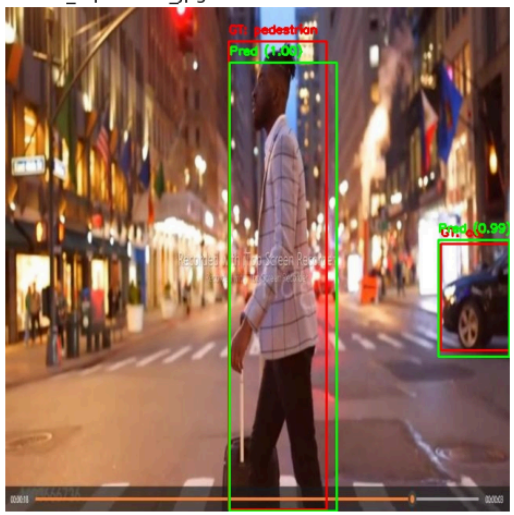


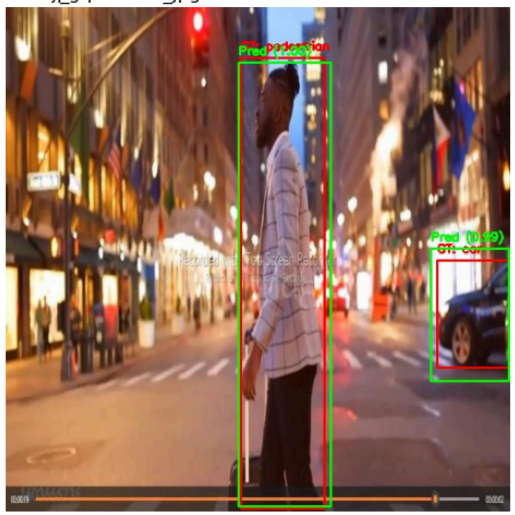


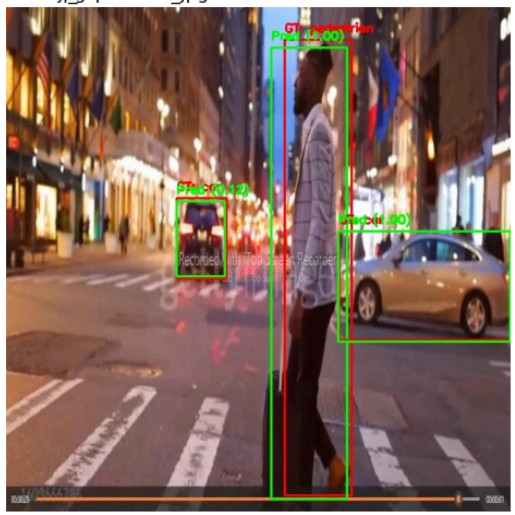




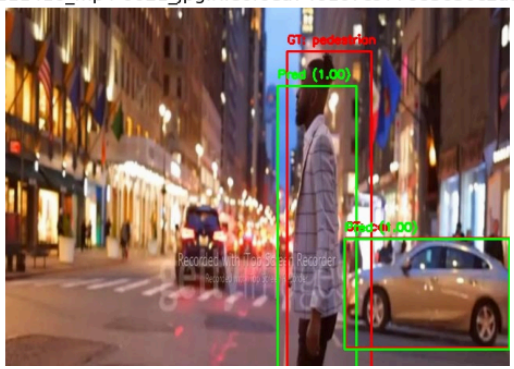
20250402_211416_mp4-0018_jpg.rf.3521f751402d0197a8[illegible].jpg 20250402_211416_mp4-0019_jpg.rf.01126a1e6c3871733e[illegible].jpg 20250402_211416_mp4-0020_jpg.rf.367b982ded591d3b59c2d1f66fe161f9.jpg







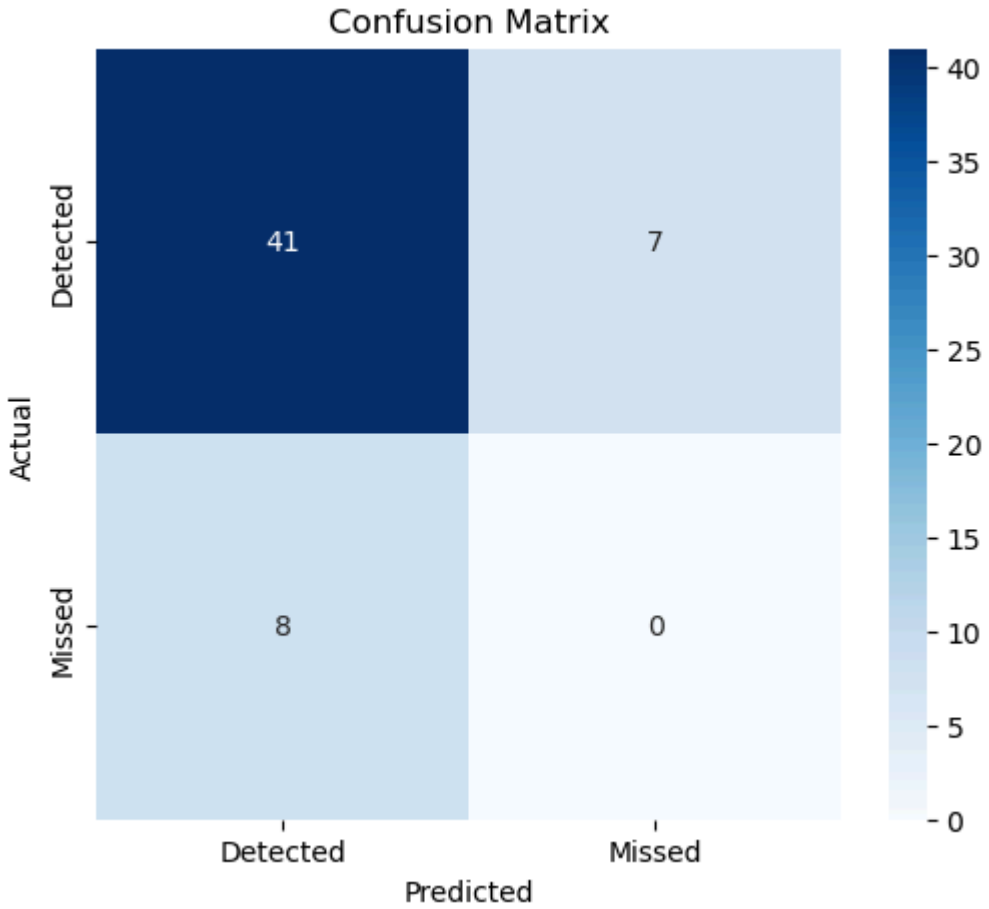
20250402_211416_mp4-0021_jpg.rf.e6f6ea746107c977683630c2adfe7916.jpg



In [29]:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

# 🔹 Load IoU evaluation file
iou_csv_path = r"E:\2. MDS DEAKIN UNIVERSITY\TASK\Task5\custom vision\evaluated_images\iou_evaluation.csv"
df_iou = pd.read_csv(iou_csv_path)

# 🔹 Define IoU threshold for TP, FP, FN
iou_threshold = 0.5

# 🔹 Calculate Confusion Matrix Elements
TP = sum(df_iou["IoU"] >= iou_threshold)   # True Positives: IoU >= 0.5
FP = len(df_iou) - TP                      # False Positives: IoU < 0.5
FN = 49 - TP  # False Negatives: Missed Ground Truths (49 total GT)

# 🔹 Create Confusion Matrix
conf_matrix = np.array([[TP, FP], [FN, 0]])  # No TN since we only have detections vs. GT
labels = ["Detected", "Missed"]

# 🔹 Plot Confusion Matrix
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# 🔹 Compute Precision, Recall, and F1-Score
precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0
f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
```

```
# 🔷 Display Metrics
print(f"🎯 Precision: {precision:.4f}")
print(f"🔍 Recall: {recall:.4f}")
print(f"⭐ F1 Score: {f1_score:.4f}")
```




```
🎯 Precision: 0.8542
🔍 Recall: 0.8367
⭐ F1 Score: 0.8454
```

In [21]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: