Computer Vision - Project 2

Visual surveillance of parking spaces on the street

Sîrbu Oana-Adriana
407 AI

1 Defining Paths

This section describes the process of setting up the directory structure for the second project. Thia project is divided into multiple tasks, and this code ensures that the necessary directories are created to store images, videos and solution files.

Main Directory

The main directory path, where the entire project is located, is defined and stored in the 'MAIN_DIR' variable. This path should be updated to reflect the actual location of the project.

Working Directory

The working directory, defined by 'WORK_DIR', is created to store the current code and the generated solution files. The path is constructed by appending 'work' to the main directory path.

Task 1 Directories

For Task 1, the paths to the images and solution directories are defined:

- 'task1_images_dir': Path to the directory containing Task 1 images.
- 'task1_sol_dir': Path to the directory where Task 1 solutions will be saved. The directory is created if it does not exist.

Task 2 Directories

For Task 2, the following directories are set up:

- 'video_dir': Path to the directory containing Task 2 videos.
- 'task2_data_dir': Path to the directory where the final frames of the videos will be stored as images. The directory is created if it does not exist.
- 'task2_sol_dir': Path to the directory where Task 2 solutions will be saved. The directory is created if it does not exist.

Task 3 & 4 Directories

For Task 3 and 4, the paths are adjusted similarly, as follows:

- ‘task3_videos_dir‘ / ‘task4_videos_dir‘: Path to the directory containing Task 3 (or 4) videos.
- ‘task3_sol_dir‘/ ‘task4_sol_dir‘: Path to the directory where Task 3 (or 4) solutions will be saved. The directory is created if it does not exist.

2 Extracting Parking Spots Boxes

The bounding boxes for the parking spots were generated using the LabelImg software. LabelImg is an open-source graphical image annotation tool. When launching LabelImg, I had to open the directory containing the images I wanted to annotate and select an image for annotation. I selected the training image 11_2 (as it had all parking spots occupied) and I drew bounding boxes by clicking and dragging the mouse. I also labeled each bounding box with the appropriate name. After annotating an image, LabelImg saves the annotations as an XML file, containing details about the bounding boxes and their labels.

Figure 1: Example of annotation for the parking spots using labelImg

Then, the XML file is parsed using the ‘xml.etree.ElementTree‘ module. The parsed XML tree is stored in the ‘tree‘ variable, and the root element is accessed via the ‘getroot()‘ method, stored in the ‘root‘ variable.

The parking spots are extracted from the parsed XML file by iterating through all ‘object‘ elements in the XML tree. For each object, the following information is retrieved:

- ‘name‘: The name of the parking spot (I named them: ’p1’, ’p2’, . . . , ’p10’).
- ‘bndbox‘: The bounding box coordinates, consisting of ‘xmin‘, ‘ymin‘, ‘xmax‘, and ‘ymax‘.

The extracted information is appended to the ‘parking_spots‘ list as tuples, where each tuple contains the name and a list of bounding box coordinates.

3 Model selection

For all four tasks, I chose to work with the YOLOv8 model.

The YOLOv8 model was loaded from the Ultralytics library, which is used for object detection tasks. Specifically, it loads the pre-trained model stored in the file ”yolov8m.pt”.

Additionally, the class indexes for cars and trucks within the YOLO model’s classes dictionary were defined. In this case, cars are represented by class ID 2, and trucks by class ID 7. These class IDs were important for identifying and detecting these specific objects in images or videos using the YOLO model.

Figure 2: Example of object detection realised by Yolov8 on the train image 01_2.jpg

4 Solution for Task 1 - Selected Parking Spot Occupancy Prediction

The solution for Task 1 involves predicting the occupancy status of parking spots based on images and query data provided.

1. The script begins by gathering paths to all JPEG images in the Task 1 directory (‘task1_images_dir‘). For each image, corresponding query data is read from a text file (‘_query.txt‘).

2. Using the YOLOv8m model from the Ultralytics library (‘model_8‘), the script detects objects within each image. Detected objects are classified, and their bounding boxes are determined.

3. The script reads the query data to determine the number and specific identities of parking spots (‘spots_to_check‘) for which occupancy predictions are required.

4. For each detected object (car or truck), the script calculates the Intersection over Union (IoU) with predefined parking spot coordinates (‘parking_spots‘). If the IoU exceeds 35%, the corresponding parking spot is marked as occupied (‘spot_status‘).

5. Results are written to output files (‘_predicted.txt‘) in the Task 1 solutions directory (‘task1_sol_dir‘). Each output file includes the total number of spots that needed to be analysed and their predicted occupancy status based on the analysis.

5 Solution for Task 2 - All Parking Spots Occupancy Predictions

The solution for Task 2 involves extracting the last frame from each of 15 videos and predicting the occupancy status of parking spots based on these frames.

1. The script iterates through each of the 15 videos available for Task 2. For each video, it retrieves the total number of frames and captures the last frame. This frame is saved as a JPEG image in the Task 2 data directory (‘task2_data_dir‘).

2. After extracting the last frame from each video, the script uses the YOLOv8m model (‘model_8‘) to detect objects within these images. It focuses on detecting cars and trucks, as specified by the class IDs (‘car_class_id‘ and ‘truck_class_id‘).

3. Similar to Task 1, the script calculates the Intersection over Union (IoU) between each detected object's bounding box and predefined parking spot coordinates (‘parking_spots‘). If the IoU exceeds 35%, the corresponding parking spot is marked as occupied.

4. Results are written to output files (‘_predicted.txt‘) in the Task 2 solutions directory (‘task2_sol_dir‘). Each output file indicates the predicted occupancy status of all 10 parking spots based on the analysis of the final frames from the videos.

6 Solution for Task 3 - Tracking cars in motion

Task 3 involves tracking objects across multiple video frames using the CSRT tracker from OpenCV and updating the bounding box using YOLOv8m detections at specified intervals (each 10 frames).

1. The script begins by initializing the CSRT tracker (‘cv2.TrackerCSRT_create()‘) and iterating through a list of video files (‘video_list‘) located in the Task 3 videos directory (‘task3_videos_dir‘).

2. For each video file, the script reads an initial annotation file to retrieve total frame count and initial bounding box coordinates. The CSRT tracker is initialized with the first frame and the initial bounding box.

3. The script iterates through each frame of the video, updating the tracker with the current frame. Every 10 frames (‘frame_index % 10 == 0‘), YOLOv8m model predictions are used to detect objects. The script selects the closest bounding box from YOLO detections to update the tracker's bounding box.

4. Results are written to an output file (‘_predicted.txt‘) in the Task 3 solutions directory (‘task3_sol_dir‘). Each output file contains frame-by-frame updates of the tracked object's bounding box coordinates.

5. If the CSRT tracker loses track of the object in any frame, no updates are written to the output file for that frame.

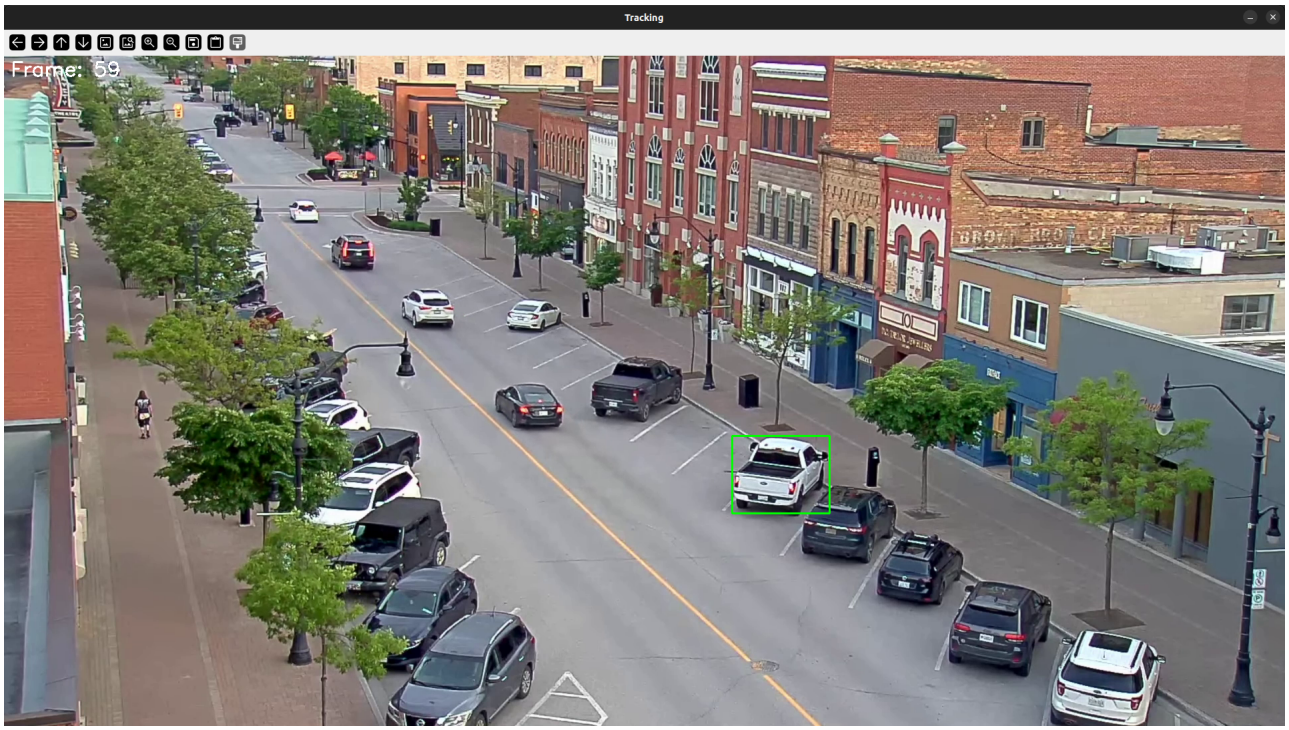

Figure 3: Example of object tracking on the fake test video 01.mp4 - initial frames

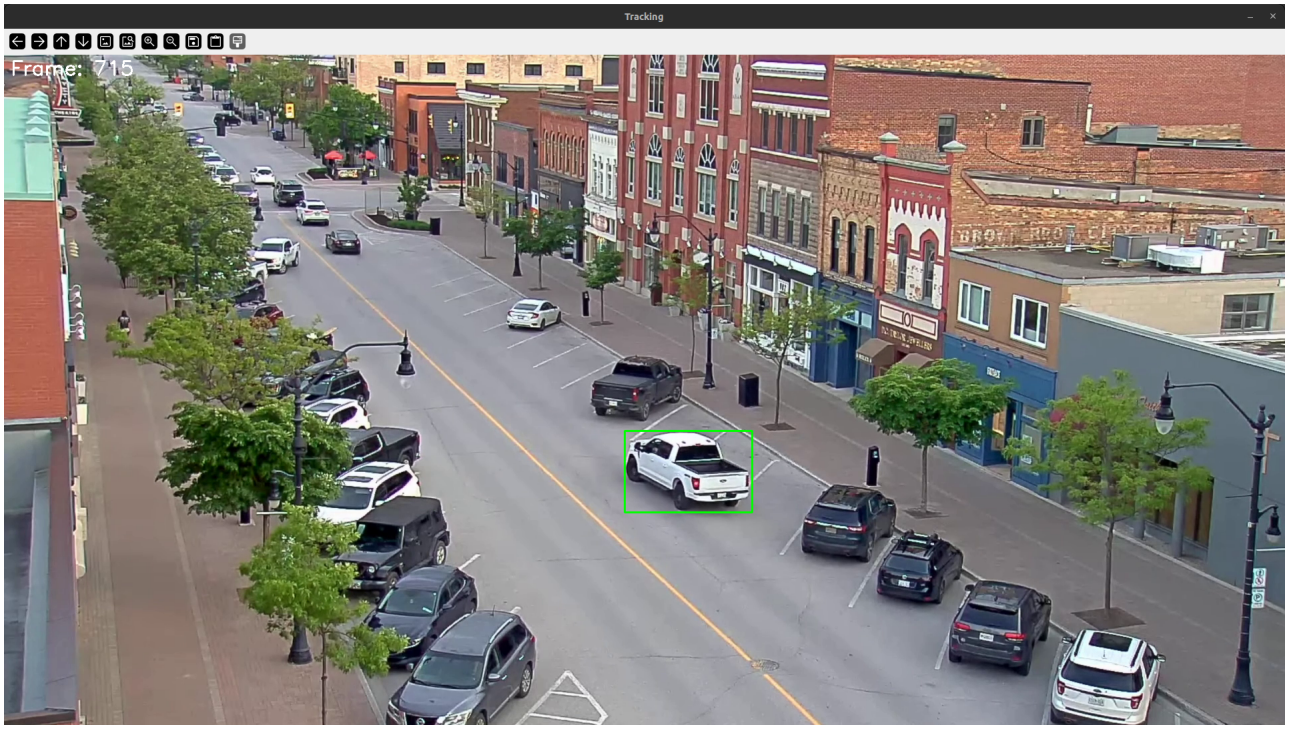

Figure 4: Example of object tracking on the fake test video 01.mp4 - middle frames

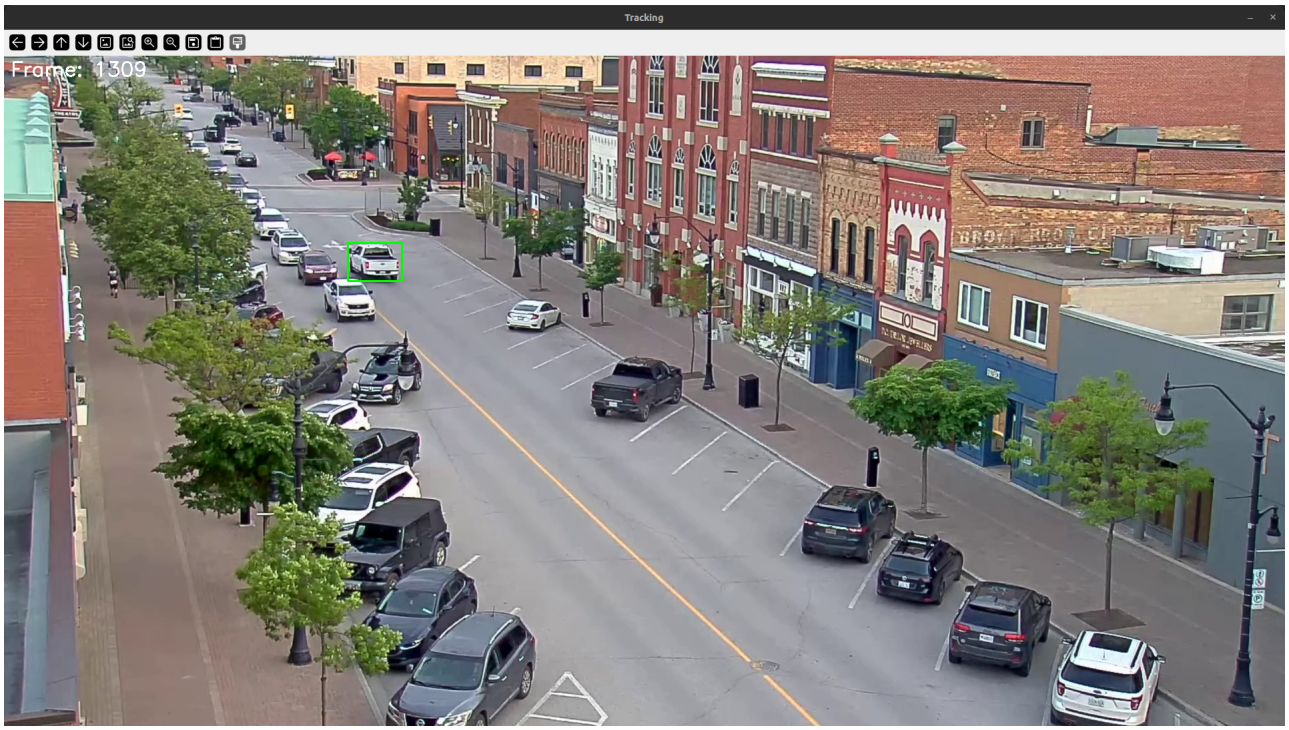

Figure 5: Example of object tracking on the fake test video 01.mp4 - last frames

7 Solution for Task 4 - Detect vehicles stopped at traffic light

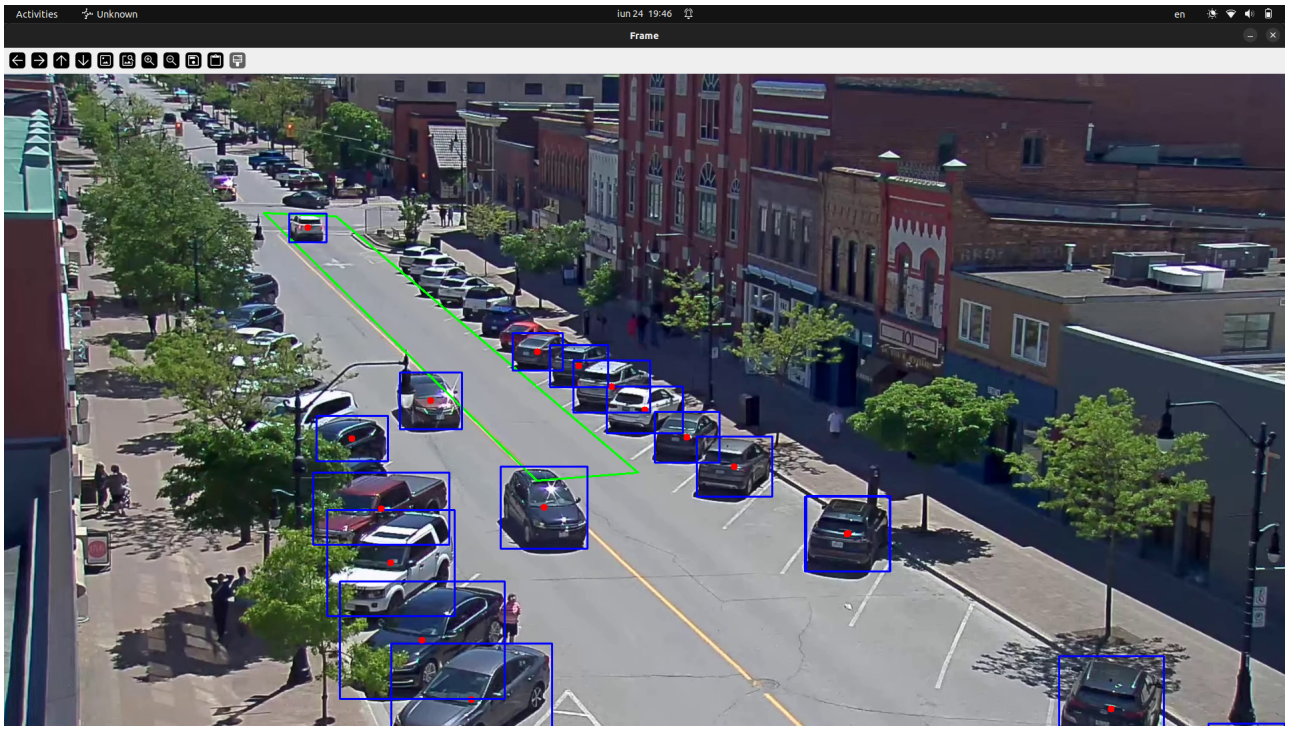

Figure 6: Example of the monitoring system on one of the training videos

Task 4 focuses on detecting stationary vehicles within a specified region of interest (ROI) in the final frames of video sequences. The solution for this task implies to:

1. Define a polygonal region in the frame where vehicle presence is to be monitored.
2. Each video is loaded, and the frames within the last specified duration are processed to detect vehicles.
3. The YOLOv8 model is used to detect cars and trucks in each frame.
4. Detected vehicles are checked for stationarity within the ROI by counting the number of frames they remain in the same position.
5. Vehicles that remain stationary for a defined threshold number of frames are recorded, and the results are written to an output file.

8 Conclusions

This project used the YOLOv8m model to detect cars and trucks in parking lots (Task 1) and predict their occupancy. Task 2 analyzed the final frames of 15 videos for the same purpose. Task 3 implemented real-time object tracking using the CSRT algorithm, with bounding boxes adjusted periodically by YOLOv8m for better accuracy. Task 4 focused on detecting stationary vehicles within a specified region of interest in the final frames of video sequences, identifying cars that remain stationary for a threshold number of frames. Initial results from train and fake tests demonstrated high accuracy in predicting parking spot occupancy and tracking vehicles.