Final Report

Final Project

ECSE415 - Introduction to Computer Vision

Autumn 2023
Electrical & Computer Engineering
McGill Univerity

Last update: 6 décembre 2023

Romuald Ricard (261194253)
Faiyad Irfan Hares (260914739)

1 Assignment

Write a python program that will analyze the two dashcam videos provided (`mcgill_drive.mp4` and `st-catherines_drive.mp4`, each are 30 frames per second, and are taken with the same car/dashcam) and provide the following analytics :

- Number of parked cars passed ;
- Number of moving cars passed ;
- Number of pedestrians passed ;
- Bonus output : Maximum speed in km/hour of the car with the dashcam.

You can use any software (except for software developed by other students in the class).

2 Problem Approach

Based on the analytic we want to extract, we can divide the original problem into simpler tasks and using a **bottom-up strategy** for developing our algorithm. We will first tackle the problem of detecting objects in the scene such as `persons` and `cars`. Then, we will go over tracking methods in order to be able to count objects in the scene. Finally, we will compare the performances of 2 algorithms we used to extract the analytic.

2.1 Object detection algorithm

The first thing we need for tracking objects in a scene is a way to **detect those objects** and get a **bounding-box** that surrounds them. We used several techniques during this class in order to perform that detection. We realized reading literature that most tracking algorithm use Convolutional Neural Netwoks (CNN) to perform that first operation. The two candidates here were `Mask-RCNN` and `YOLO` detectors. We decided to use the **YOLO** detector because of the speed at which it detects objects, but also because it features a list of detectors for various desired performance / speed criteria. **YOLO**

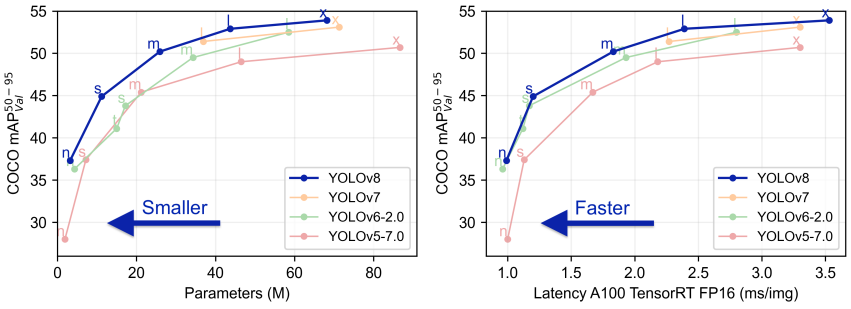

FIGURE 1 – Speed vs accuracy plots for different models (source : `YOLO`)

outputs a list of labeled bounding-boxes, along with the probability of the label being correct. In this study, we are only interested in the labels `car` and `person` with a probability value more than a threshold (generally 0.5).

Furthermore, in order to reduce the computational cost of the detection, we can first reduce the quality of the frames (usually 30% for the tests).

2.2 Software Packages and Routines Description

— **YOLOv5 (You Only Look Once, version 5)** : A cutting-edge real-time object detection system using Convolutional Neural Networks (CNNs). YOLOv5 is highly suitable for this purpose due to its balance between detection speed and accuracy, making it ideal for processing video in real-time or near-real-time scenarios. Its ability to detect small and fast-moving objects makes it a prime choice for analyzing dashcam videos where quick and reliable object detection is crucial.

— **OpenCV (Open Source Computer Vision Library)** : A comprehensive open-source computer vision and machine learning software library. OpenCV is selected for its extensive functionalities in image and video processing, which are essential for handling video data, such as reading video frames and performing image transformations. Its widespread use, robustness, and efficiency in handling real-time image and video analysis align perfectly with the needs of this method.

— **NumPy (Numerical Python)** : An essential package for scientific computing in Python. NumPy is chosen for its efficiency in handling large arrays and matrices of numerical data, a common requirement in computer vision tasks. The library's powerful mathematical functions simplify complex numerical calculations, such as computing centroids and distances, which are integral to the object tracking and speed estimation components of this method.

What are multi-trackers ? We will first start by counting how many different vehicles there are in the video. Obviously, counting the numbers of car detected on each frame does not give us a the final count we are looking for. Thus, the goal of a multi-tracking algorithm is to assign a unique id to the same object though multiple frames and being able to differentiate them (example shown in FIGURE 2). There exists a variety of multi-tracker types.

FIGURE 2 – Multi-tracking vehicles, taken from [1]

3 Tracking methods

As we mentioned earlier, there are a variety of tracking methods available in python libraries. **OpenCV** features trackers that can be adapted to work well in the context of this study. However, we wanted to program our own one in order to get a deeper understanding of the challenges and solutions we come across.

3.1 The problem of matching bounding-boxes

The output of our object recognition algorithm will be labeled bounding-boxes. The principle of object tracking is to match a known bounding-box at iteration i with a detected bounding-box at iteration $i+1$ (see FIGURE 3). There exists several methods to perform that matching operation. Suppose we have n tracked objects $\{t_i\}$ in memory, and we detect m objects $\{d_i\}$

FIGURE 3 – Bounding boxes of current (——) and previous (——) frame

using the object detection algorithm. In FIGURE 3, the tracked bounding-boxes are drawn in blue (——) and the detected bounding-boxes are drawn in red (——). We can divide these 2 sets into 3 different groups :

- The set of the pairs of **matched tracked and detected bounding-boxes** $M = \{(t_i, d_j)\}$;
- The set of all **un-matched tracked bounding-boxes** $U_t = \{t_i\}$;
- The set of all **un-matched detected bounding-boxes** $U_d = \{d_i\}$.

We can then update our tracking algorithm based on those 3 groups :

- Update the trackers with new position from the set of matched tracked and detected bounding-boxes M ;
- Mark the un-matched tracked bounding-boxes U_t as undetected ;
- Create new tracker object for un-matched detected bounding-boxes U_d

3.2 Matching based on center distances

The simplest and most crude method for matching bounding-boxes consists in using the distance between the center of all pairs of bounding-boxes. Suppose we have n tracked objects, and we detect m objects. We create an $n \times m$ matrix I whose element $(i\,j)$ is the distance between the centers of tracked bounding-box and detected bounding-box. For the bounding-boxes in FIGURE 3, we have a matrix that looks like :

$$I = \begin{bmatrix} \mathbf{14} & 404 & 574 & 1177 & 863 & 682 & 747 \\ 434 & \mathbf{16} & 154 & 762 & 444 & 262 & 328 \\ 595 & 178 & \mathbf{7} & 601 & 283 & 101 & 166 \\ 1187 & 775 & 605 & \mathbf{3} & 318 & 497 & 433 \\ 876 & 460 & 289 & 321 & \mathbf{0} & 180 & 115 \\ 704 & 287 & 116 & 492 & 173 & \mathbf{8} & 57 \end{bmatrix} \tag{1}$$

As we can see, there are 6 tracked bounding-boxes and 7 detected bounding-boxes. We can easily match every tracked bounding-box to a single detected bounding-box. In order to match the tracker i with a detected bounding-box j, we find the minimum of the i-th line of I.

— If the element is smaller than a threshold, we have a match ;

— If not, we have a undetected tracked bounding-box.

3.3 Rectangle Intersection Area Tracking

Another idea might be to compute the intersection area between all the bounding-boxes. Let b_1 and b_2 two bounding-boxes represented each by :

$$b_1 = \begin{bmatrix} x_{1,\min} & y_{1,\min} & x_{1,\max} & y_{1,\max} \end{bmatrix}$$
$$b_2 = \begin{bmatrix} x_{2,\min} & y_{2,\min} & x_{2,\max} & y_{2,\max} \end{bmatrix}$$

We can then compute the ratio between the intersection area of the two bounding-boxes and the area of the fist bounding-box with :

```
1   def compute_rectangle_intersection_area(bb1, bb2):
2     # Area of bb1
3     a = abs(bb1[2] - bb1[0])*abs(bb1[3] - bb1[1])
4
5     # Compute intersection sides
6     dx = min(bb1[2], bb2[2]) - max(bb1[0], bb2[0])
7     dy = min(bb1[3], bb2[3]) - max(bb1[1], bb2[1])
8
9     # If intersection
10    if dx >= 0 and dy >= 0:
11      return(dx*dy/a)
12    else:
13      return(0)
```

We can then construct a matrix similar to the distance matrix, but filled with the intersection area ratio calculated above. We can then make the assumption that the element $I_{i,j}$ represents

the **probability** that the i-th bounding-box represents the same object as the j-th bounding-box. On FIGURE 4, we have drawn the intersection area for matched bounding-boxes. We also give the

FIGURE 4 – Intersection area between matched bounding-boxes

intersection area ratio matrix :

$$I = \begin{bmatrix} \mathbf{0.96} & 0.05 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.08 & \mathbf{0.97} & 0.15 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.35 & \mathbf{0.95} & 0.0 & 0.0 & 0.08 & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.99} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & \mathbf{0.98} & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.02 & 0.0 & 0.0 & \mathbf{0.9} & 0.15 \end{bmatrix} \tag{2}$$

From this, we can quickly match the tracked bounding-boxes with detected bounding-boxes.

3.4 The Tracker class

In order to keep track of the objects in the scene, we create a `Tracker class` representing an object in the scene. We store those trackers in a dynamic list that will contain all the objects currently tracked in our scene.

3.4.1 Attributes and methods

This tracker class has attributes such as :

— The current bounding-box of the tracked object

— The confidence score (from 0 to 1)

— The label of the tracked object (`car`, `person`...)

And has methods :

— `update(bb)` : Updates the tracker when a matched detected bounding-box has been found ;

— `update_no_measure()` : Updates the tracker when no match was found.

3.4.2 Confidence score

It can happen that a tracked object is not detected (matched with no bounding-box at iteration i). There can be two main reasons to this :

— Either the object is occluded by another object, or have moved too far away to be detected

— Either the object has disappeared out of the scene (behind the car)

In order to remove the objects that are no longer visible, we use the `confidence` score. At the creation of the `Tracker` instance, we initialize this score to be 0. With each update, there are 2 cases :

— If the object is matched with a detected bounding-box, we increment the score by a certain amount (0.1 generally, and cap at 1) ;

— Else, we decrement the score by the same amount.

This `confidence` score then gives us a rough reading on two things :

— Weather or not the object is still visible in the scene ;

— Has the object been tracked for a long time (+10 iterations).

We can use that confidence score to perform two operations :

— Either to delete the tracker once the confidence score reaches below 0 ;

— Either assign an `id` to the tracker and confirm that it is indeed a confirmed tracked object.

If that `confidence` score goes below 0, the `Tracker` is deleted.

3.5 Counting vehicles and pedestrians

We evaluate the performances of this tracking algorithm by counting the number of cars (moving or not) and the number of pedestrians on the two videos :

Video	Cars	Pedestrians
`st-catherines_drive.mp4`	76	60
`mcgill_drive.mp4`	53	36

TABLE 1 – Count of vehicles and pedestrians in both scene

4 Distinction between parked and moving cars

Now that we can identify cars and track them, we need to distinguish between parked and moving cars. One way to do it is to compare the global velocity of the scene with the velocity of the tracked objects.

4.1 Global scene velocity

In this section, we are not yet interested in finding the speed of the car the camera is in. We need a reference of how the scene globally moves from the perspective of the camera. One way to obtain an estimate of this velocity is by computing the average magnitude of the scene optical flow.

```
def velocities_in_scene(current_frame, prev_frame, trackers):

    # Convert current and previous frames to grayscale
    prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
    current_gray = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)

    # Calculate dense optical flow using Farneback method
    flow = cv2.calcOpticalFlowFarneback(prev_gray, current_gray, None, 0.5, 3,
                                        15, 3, 5, 1.2, 0)

    # Compute magnitude and angle of the flow vectors
    magnitude, angle = cv2.cartToPolar(flow[..., 0], flow[..., 1])

    # Compute global scene velocity by averaging magnitude
    scene_velocity = np.mean(magnitude)

    return(scene_velocity)
```

4.2 Bounding-box relative velocity

We can also use the optical flow to compute the **velocity of a bounding-box** by averaging the optical flow magnitude over the bounding-box area. This relative velocity is displayed in FIGURE 5. We can then compute the speed of the bounding-box relative to the scene speed seen from the cameras perspective. This relative velocity gives us an indication of how objects are moving relative to the global movement of the scene.

4.3 Results of counting parked and moving cars

We gather the predictions into a table : We can see in both cases the number of parked cars

Video	Moving cars	Parked cars	Pedestrians
`st-catherines_drive.mp4`	46	30	60
`mcgill_drive.mp4`	27	35	38

TABLE 2 – Count of moving/parked cars and pedestrians in both scene

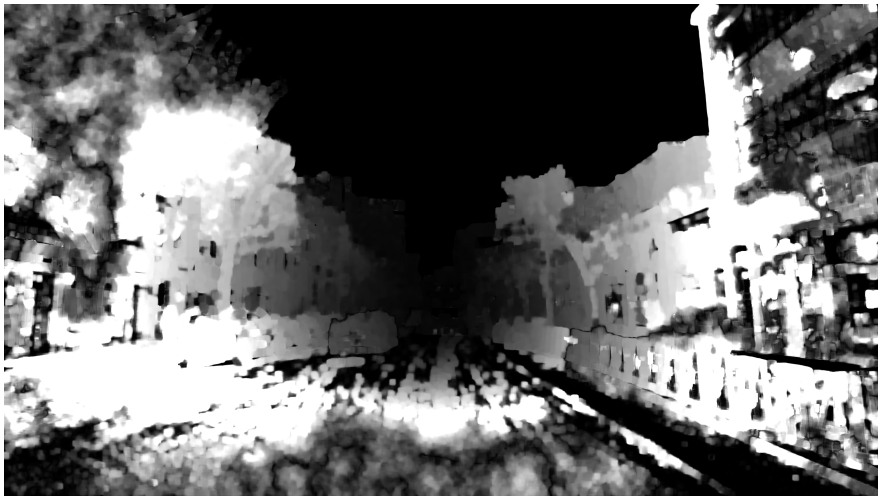

FIGURE 5 – Relative optical flow velocity (to average)

don't match with a manual count. We also see that the total number of cars is above the manual counts.

4.4 Limitations of the model

— The **occlusion problem** was not tackled in this model. Occlusion happens so when a car is no longer visible due to an obstruction in the scene. If an object is occluded for long enough, its tracker may be deleted thus it would be double detected once it reappears in the scene.

— The **velocity** issue. Our moving and parked car count seems to be wrong. This is due to the fact that the average value of the optical flow does not represent well the velocity in different regions of the screen. For example, the optical flow near the center of the screen is smaller than near the sides (see FIGURE 5). Thus, when a parked car moves from the center of the screen to the side of it, its relative velocity might change over time, making the classification difficult.

— The car identification could be more effective by storing **SIFT** features in the tracker object and using those to match detected an tracked bounding-boxes as discussed previously.

5 Alternative Approach based on center distance matching

This is a much simpler method of detection than the previous one. This method is based on the fact that since we are only concerned about cars and pedestrians that the car passes, we only need to count them when they are close to the frame edges. Again, we use the YOLOv5 model for object detection, with a focus on objects detected near the left, right, or bottom edges of the video frame. The detailed description of the method is outlined below.

5.1 Description of Method

5.1.1 Object Detection

- The method uses YOLOv5, a modern deep learning model, for detecting objects in each video frame.
- It processes the frames to detect cars and pedestrians, computing centroids for each detected object.

5.1.2 Tracking and Counting

- The method employs a tracking mechanism based on the centroids of objects, maintaining their identity across frames.
- It separately counts cars and pedestrians, with a count triggered only when an object's centroid is near the frame edge, as determined by the is_near_edge function.

5.1.3 Assumptions Made

- The method assumes that relevant objects (cars and pedestrians) predominantly appear near the video frame edges, typical of dashcam footage. This is based on the edge threshold, which determines how close an object must be to the frame's border to be considered for counting.
- It assumes parked cars are stationary, with their apparent motion indicating the dashcam car's movement. This is based on the movement threshold set to distinguish between stationary and moving cars.
- The movement threshold for differentiating parked and moving cars is manually set, assuming it accurately represents significant movement in the context of the video.
- The implementation assumes relative stability of the dashcam. Excessive movement could lead to inaccuracies in object tracking and speed estimation.

5.1.4 Results

The results of applying this method are summarized in the table below :

	mcgill_drive.mp4	**st-catherines_drive.mp4**
Parked Cars	13	51
Moving Cars	22	0
Total Pedestrians	37	56

TABLE 3 – Object detection and counting results for alternative method.

5.1.5 Comparison with Intersection Area Approach

The implemented method is compared with the previous, highlighting key differences in accuracy and efficiency :

- **Accuracy** :
  - *Current Method* : While effective in certain scenarios, it generally exhibits lower accuracy compared to the previous. This is primarily due to the reliance on manual thresholds and potential misclassifications (e.g., moving cars identified as parked), especially in complex or crowded scenes.
  - *Previous method* : The previous approach provides a more sophisticated mechanism for predicting and updating the state of moving objects. This results in better handling of object trajectories and velocities, leading to more precise tracking and classification.
- **Efficiency and Speed** :
  - *Current Method* : It offers higher efficiency and faster processing, mainly because it employs a simpler tracking mechanism and is less computationally intensive compared to the previous approach. This makes it more suitable for real-time or near-real-time video analysis where speed is a critical factor.
  - *Intersection area method* : While highly accurate, the previous approach is generally more computationally demanding due to its iterative prediction-correction process. This can result in slower processing times, making it less ideal for scenarios where speed is prioritized over precision.

5.1.6 Limitations of the Method

The method, while effective in its application, has several limitations that should be considered :

- **Detection Accuracy** : The method uses YOLOv5 for object detection, which generally provides high accuracy, but it sometimes encounters issues like double detection or missed detections in challenging conditions.
- **Misclassification of Moving Cars** : A notable problem is the misclassification of moving cars as parked. This occurs when the movement of a car between frames does not exceed the manually set movement threshold, leading to inaccurate categorization.
- **Tracking Mechanism Limitations** : The method relies on tracking the centroids of detected objects, which can be less accurate in crowded or complex scenes where objects overlap or frequently enter and exit the frame.

— **Manual Threshold Settings** : The use of manually set thresholds (like the movement threshold) for differentiating between parked and moving cars may not be universally effective across varying traffic conditions and video qualities.

— **Dashcam Stability** : The accuracy of object tracking and speed estimation is highly contingent on the stability of the dashcam, as excessive movement can lead to inaccuracies in the detection and tracking processes.

— **Real-time Processing Capability** : Depending on the computational power available and the complexity of the YOLOv5 model, the method might struggle with real-time processing of the video footage.

— **Environmental and Lighting Conditions** : The effectiveness of object detection can be impacted by varying environmental factors such as weather conditions and lighting, which can pose challenges to the consistency of the tracking and detection performance.

Références

[1] Pengfei Lyu, Minxiang Wei et Yuwei Wu. "Multi-Vehicle Tracking Based on Monocular Camera in Driver View". In : *Applied Sciences* 12.23 (2022). ISSN : 2076-3417. DOI : 10.3390/app122312244. URL : https://www.mdpi.com/2076-3417/12/23/12244.