Visual Object Tracking Report

Tom Genlis - EPITA - SCIA 2024

Introduction

This report provides an in-depth analysis of an implementation of a multi-object tracking (MOT) system. The MOT system is a crucial component in a variety of applications, including surveillance, autonomous vehicles, and robotics. It is designed to detect objects in sequential frames and maintain their identities throughout the sequence.

To evaluate the model, we will employ several metrics to provide a comprehensive analysis of the algorithm's performance:

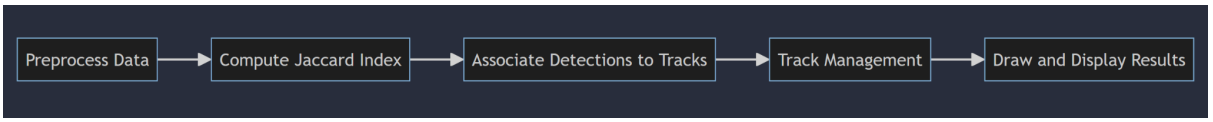
- **HOTA (Higher Order Tracking Accuracy)**: This metric evaluates the algorithm's ability to correctly associate detections with the same identity over time. It considers both detection and association accuracy.
- **MOTA (Multiple Object Tracking Accuracy)**: MOTA measures the tracking accuracy considering missed detections, false positives, and identity switches.
- **IDF1 (ID F1 Score)**: IDF1 is the harmonic mean of ID precision and ID recall. It measures the overall accuracy of identity assignments.
- **ID Switch (IDSW)**: This metric counts the number of times that tracked objects change their identity in the tracking process. A lower number of ID switches indicates better performance.
- **Fragmentation (Frag)**: Fragmentation refers to the number of times tracks are interrupted during the tracking process. Lower fragmentation is desirable as it indicates more consistent tracking.

System Overview

The multi-object tracking system is a comprehensive solution designed to track multiple objects across a sequence of frames. The system is built using Python and leverages various libraries such as OpenCV, Pandas, and NumPy. The system is divided into several components, each responsible for a specific task in the tracking process.

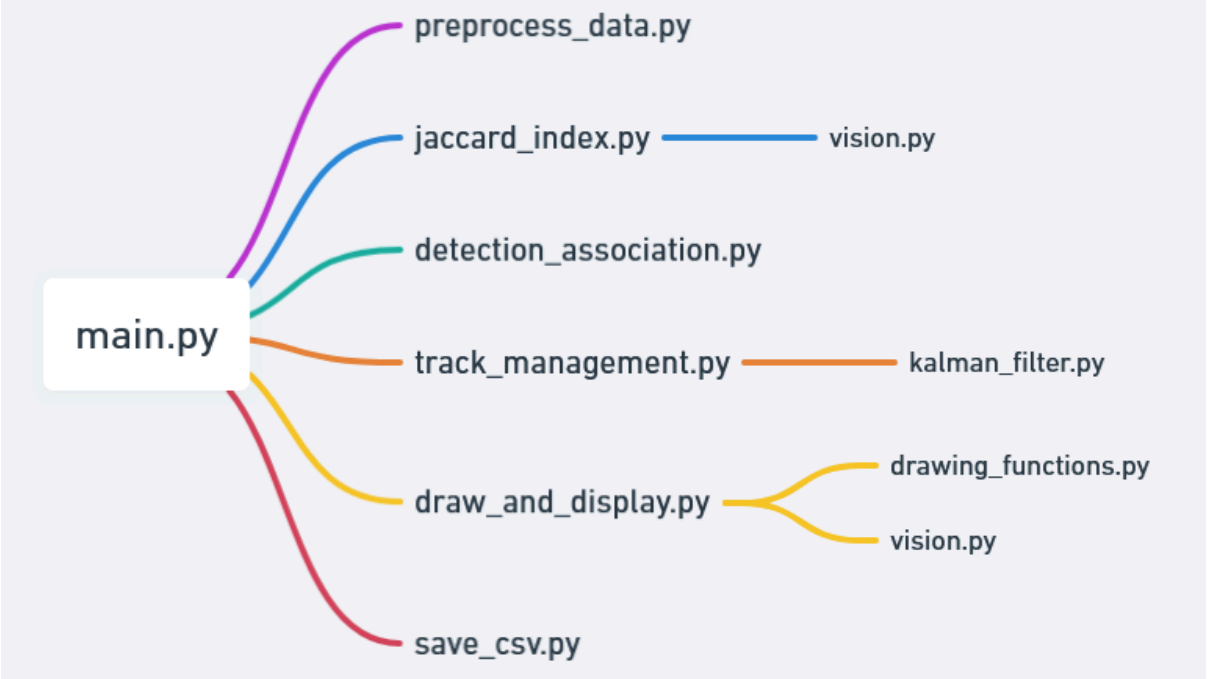
System Workflow

The following diagram illustrates the main components and workflow of the system:



Component Description

Here's the component dependency graph :



Components description :

- **preprocess_data** : takes the raw CSV file and removes non confident detection, rounds pixel values to ints, etc…
- **jaccard_index** : computes the "score of similarity" matrices for each frame for each bounding box par frame compared to their previous frame instances. Processed using ResNet18 embeddings cosine similarities, histogram correlations and IoU metrics.
- **vision** : YOLOV5 detection and ResNet18 bbox embedding using images.
- **detection_association** : using the Hungarian algorithm, orders bboxes for each frame in the lowest cost (highest similarity score) possible using the score of similarity matrices computed earlier.
- **track_management** : computes the new, active and inactive tracks for each frame, and computes their Kalman filters.
- **kalman_filter** : kalman filter implementation class
- **draw_and_display and drawing_functions** : paints the image, bounding boxes, scores, ids, arrows.
- **save_csv** : saves the detector's detections in the required 10 columns CSV format.

Steps

0. **Generate YOLO Detection Files** : This component generates the bounding boxes for each image of the video, then outputs it in a readable file.

```
generate_yolo_file(images)
```

1. **Preprocess Data**: This component is responsible for preparing the input data for the tracking process. It reads the detection and ground truth data from CSV files and organizes them into frames. This is done in the preprocess_frames function in the preprocess_data module.

```
det_frames, og_len = preprocess_frames(det)
```

2. **Compute Similarity Index**: The similarity index, is a combination of the jaccard index (IoU), ResNet embedding cosine similarity and histogram correlation. This is done in the *resnet_embedding_similarity_frames* function in the jaccard_index module. “Frames” are bounding boxes dictionaries that have a list of them for each frame.

```
jaccard_index_frames, similarity_frames, histogram_frames =
resnet_embedding_similarity_frames(images, det_frames)
```

3. **Associate Detections to Tracks**: This component associates each detection with a track. The association is based on the Similarity index computed in the previous step. This is done in the *associate_detections_to_tracks* function in the *detection_association* module.

```
det_tracks, det_jaccard_values = associate_detections_to_tracks(
        det_jaccard_index_frames,
        det_similarity_frames,
        det_histogram_frames,
        sigma_iou=0.73,
    )
```

4. **Track Management**: This component manages the tracks, updating them based on the associations made in the previous step. This is done in the track_management function in the track_management module.

```
    det_bboxes_for_each_frame = track_management(det_frames, det_tracks, det_jaccard_values)
```

5. **Draw and Display Results**: This component visualizes the tracking results. It draws the bounding boxes on the frames and displays the video. This is done in the draw_frames, show_video, and load_images functions in the draw_and_display module. It also saves the bounding box indexes in a 10 columns CSV file.

```
save_tracking_results(
        det_bboxes_for_each_frame, det_frames, save_name
    )
det_frame_imgs = draw_frames(og_len, images, det_bboxes_for_each_frame, det_frames)

show_video(file_name, det_frame_imgs)
```

Results

Presentation of evaluation results

Using TrackEval we go the following result at the last iteration of the tracker (TP5) :

- HOTA **8.8685**
  Has been improved thanks to the ResNet embeddings cosine similarities in association with the Hungarian Algorithm (ID consistency) and the YOLOV5 model (detection accuracy).

- MOTA **-21.781**
  Score is not great because of false positives (YOLOV5 artifacts), even with IoU and confidence filtering. The Kalman filters allow for a smoother detection, but sometimes lag behind some persons.

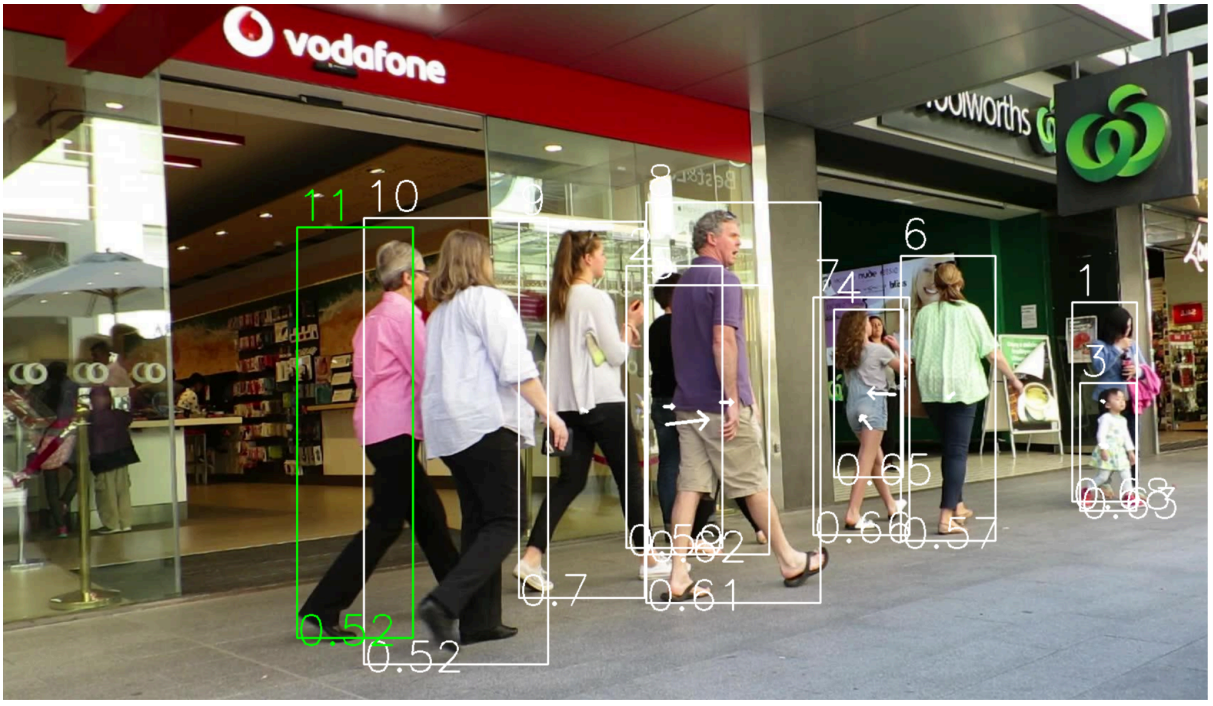
- IDF1 **8.3669**
  Same reasons as for HOTA, and also thanks to histogram correlation.

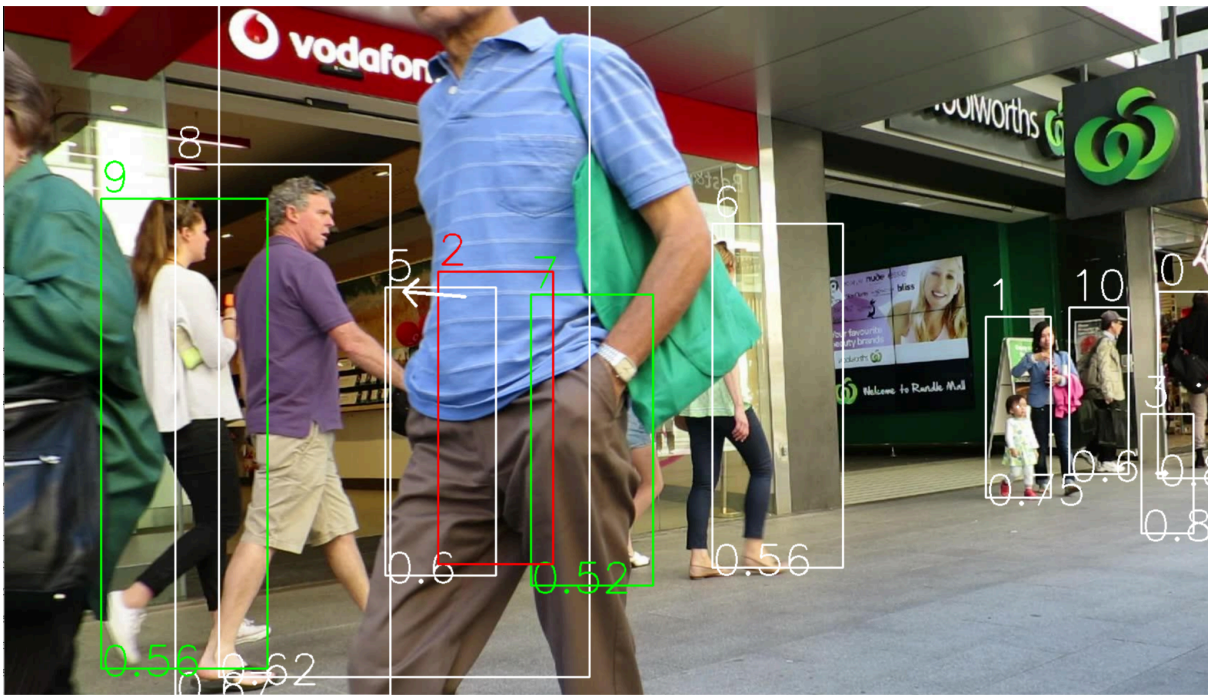
- IDSW **534**
  Same reasons as for HOTA and IDF1.

- Frag **649**
  Upgraded thanks to the Kalman filter, allows for a smoother movement and therefore reduces the number of times a ground-truth trajectory is interrupted.

Visualisations

Here are some screenshots of the YOLOV5 detections.





Tracks colors are :

- green if the track appeared
- white if the track is continuous
- red if the track disappeared

Tracks can have a trajectory arrow if they are old enough.
Tracks have an ID and a score which is the result of :

- if iou >= 0.7 then

  *(similarity / max_similarity) * 0.7 + histogram_score * 0.3*
- else

  0

Only tracks matching previous tracks with an IoU score above 0.7 are shown.

Discussion

The following improvements greatly improved the system :

1. Hungarian Algorithm to optimize the order of the bounding boxes
2. Kalman Filters to improve bounding boxes stability and keep track of overlapping ones
3. YOLOV5 detection to improve the bounding boxes accuracy over the span of the video
4. ResNet18 embedding processing and cosine similarity between them to better match the tracks
5. Histogram correlation to better match the images together

The results are not promising. In fact the instability of the bounding boxes is highlighted by the MOTA metric. This is mostly due to the way the Hungarian algorithm works. This is surely the cause of a bug, but after careful consideration and +50h sunk into this great project to my regret I have to stop there.

Conclusion

The detector plays a huge role in the performance of a tracker. Even if the matcher also plays a role, it can't magically improve the overall performance without a solid base of detections. This is why I think the best way to improve this system would have been to use YOLOV5 detections and filter them using another algorithm that can detect outliers.

Also, the current track management system has a memory of 2 frames, and only searches for the previous frame. Therefore the system completely forgets previously seen bboxes if they disappear for more than 1 frame.

Thank you for reading.