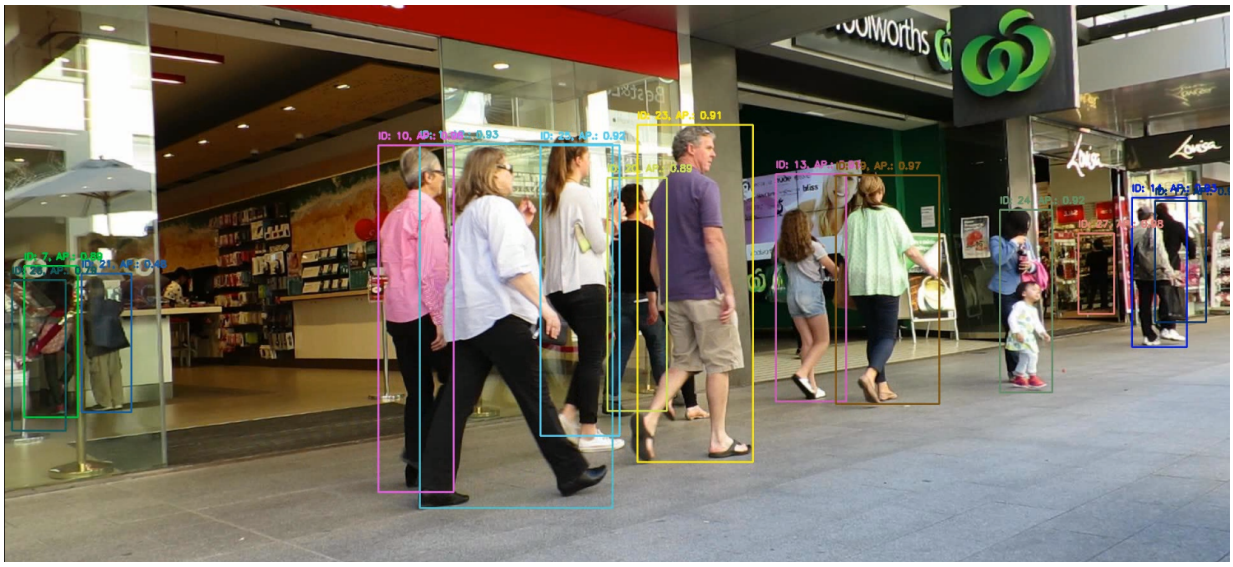
MLVOT Report

TP1 to TP5

Multi object tracking project series



AUTHOR :
alexandre.devaux-riviere

Contents

1 TP1 : Single Object Tracking with Kalman (Centroid-Tracker)

In this work, we had to implement a single object tracking system using Kalman filter for smooth and accurate tracking of object centroids, that would later be used as template for others TPs. The main objTracking file process video frames and track a simple ball object using the combination of object detection and state estimation.

Implementation details

— **KalmanFilter.py**

As mentioned in the subject, I created a KalmanFilter class with initialization, prediction, and update methods. In the __init__ method, I configured system matrices for optimal filtering performance using the values seen in the course :

A	(State transition matrix)
B	(Control input matrix)
H	(Measurement matrix)
Q	(Process noise covariance)
R	(Measurement noise covariance)

I handled both prediction and measurement update steps updating state space model with position and velocity components.

— **Detection.py**

I used the provided canny edge detection-based object detector.

— **objTracking.py**

I merged both features in order to get a successfully integrated detector output with Kalman filter with centroid-based tracking mechanism with clean visualizations.

Challenges and solutions

— **Matrix Dimensionality**

I got some problem ensuring correct dimensions for all matrices in the Kalman filter. To overcome the difficulty, I carefully implemented matrix operations following the mathematical model, using numpy and used a lot of debug to find out wrong shapes.

For example with the following issue :

```
# Incorrect implementation
self.u = np.array([u_x, u_y])

# Correct implementation
self.u = np.array([[u_x], [u_y]])
```

— **Parameter tuning**

I decided to keep default values for process noise and measurement noise because I had better results with these experimented values :

$$\text{Sampling time} : dt = 0.1$$
$$\text{Process noise (std. dev.)} : std\_acc = 1.0$$
$$\text{Measurement noise (std. dev.)} : x\_std\_meas = y\_std\_meas = 0.1$$

These parameters helped me achieve stable tracking.

2 TP2 : IOU-based Tracking (Bounding-Box Tracker)

The second TP focused on developing a simple single object tracking system using IoU over bounding boxes as the primary metric for association further extended to multiple object tracking (multiple tracks simultaneously). Here we only process pre-generated detections.

Implementation details

— **Track and TrackManager modules**

The implementation started with creating the TrackManager class that handles track (Track class) creation , updates, deletions and is able to change the tracks states to mark them visible or invisible.

— **The core : IOUTracker module**

When I assured myself that the Track manager was properly working, I continued with creating the IOUTracker class that manages the entire tracking pipeline. I worked on the core functionality : computing IoU between bounding boxes using coordinates and creating a similarity matrix for all track-detection pairs. In the update method (used for each frame), and as proposed in the subject, I utilized the Hungarian algorithm through scipy's linear_sum_assignment() function to find the best matches between existing tracks and new detections.

The way I implemented this systems allows the program to maintain active tracks and handle unmatched detections by creating new tracks. Unmatched tracks are marked as invisible until they exceed a maximum threshold (I decided to set it to self.max_invisible=10).

— **TrackVisualizer module**

The visualization component was simply implemented using OpenCV, with each track assigned a unique color for consistent visual representation across frames. For debug purposes, i also decided to add the IuO coef label on the top of each box along with its ID.

Challenges and solutions

Track management complexity

Managing multiple tracks simultaneously was very challenging, particularly when dealing with track updates (with ID changes) and history. I solved this issue by implementing the TrackManager class encapsulating all track related operations.

3 TP3 : Kalman-Guided IoU Tracking (Bounding-Box Tracker)

The goal of the third TP was to combine the strengths of both previous implementations : integrating Kalman filtering with IoU based tracking for improved accuracy / robustness.

Implementation details

— **KalmanIOUTracker module**

Building upon TP2, I integrated the Kalman filter from TP1 into the IoU based tracking system. The KalmanIOUTracker class extends my basic IoU tracker implementation by incorporating state estimation for each track. To do so, I just had to add few methods to convert between bounding box coordinates and centroids, allowing the Kalman filter to predict object positions in subsequent frames. The existing similarity computation matrix was modified to now enhance to use Kalman-predicted positions when calculating IoU scores. A new attribute inside Track was also added (self.kf) in order to help each track maintains its own Kalman filter instance, which is updated during successful detection matches and later used for prediction.

— **Better TrackVisualizer module**

For this TP, the visualization system was also changed through the TrackVisualizer class, which now includes additional features to display Kalman filter states. I implemented new methods to visualize both the predicted and actual states of objects (initially for debug purposes), including : draw_kalman_state() to show velocity vectors and predicted positions, draw_history_trace() to show the object trajectories over time, providing better insight into the tracking performance.

Challenges and solutions

Better visualization and performance optimization

During the IoU Tracking extension with Kalman filters, i had troubble debugging complex errors/ bad behavior. Implementing a better visualization module helped me a lot. I also had performance issues with updating Kalman filters for non active tracks.

4 TP4 : Appearance-Aware IoU-Kalman Object Tracker

The fourth TP added appearance based features using deep learning-based object re-identification (ReID) to further improve tracking robustness by combining it with our implementation of Kalman filter.

Implementation details

— **AppearanceKalmanTracker module**

In TP4, I enhanced the tracking by creating AppearanceKalmanTracker class, extensing the initial KalmanIOUTracker, by combining geometric information (IoU) / motion prediction (Kalman), and appearance similarity.

Now for each update, each detection is processed through the ReID model to generate feature vectors, which are then compared using cosine similarity / euclidean distance (I finally decided to go with the cosine similarity). Finally the tracking association now uses a weighted combination of IoU scores and appearance similarity in the similarity matrix computation, making the predictions more robust to occlusions and able to identity switches.

— **ReidFeatureExtractor module**

Thanks to the help of the subject, I implemented the ReidFeatureExtractor class required by the AppearanceKalmanTracker class to handle feature extraction using the given pre-trained neural network model (reid_osnet_x025_market1501.onnx). To make it work properly, I decided to use the given preprocessing of image patches, including : resizing, color space conversion, and normalization according to the ReID model's requirements.

Challenges and solutions

— **Similarity metric combination**

Finding the right balance between IoU, Kalman predictions, and appearance similarity required a lot of experimentation and I finally decided to go fifty / fifty ($\alpha = 0.5$ and $\beta = 0.5$).

— **Model loding and integration**

It was my first time using onnx like this, I struggled to find proper guides / documentation to finally integrate the pre-trained ReID model via a session 'ort.InferenceSession' (used inside extract_features() and detect() methods).

5 TP5 : Appearance-Aware IoU-Kalman Object Tracker : Detector Extension

The final TP was focused on improving the detection component using modern deep learning-based approaches with a bonus consisting of locally benchmarking our model to get performance metrics with a Tracker python lib.

Implementation details

PedestrianDetector

— **PedestrianDetector module**

For the TP, I chose to integrate YOLOv5 (a modern deep learning-based detection model) into my tracking pipeline.

Torch guide available here : `https://pytorch.org/hub/ultralytics_yolov5/`

Github repos availble here : `https://github.com/ultralytics/yolov5`

To do so, I implemented the PedestrianDetector class in which I configured the model to focus on the pedestrian class and includes confidence thresholding for reliable detections inside the generated det.txt file.

— **Metrics module**

I also integrated the TrackEval toolkit for comprehensive performance evaluation, allowing measurement of key metrics like HOTA, IDF1, and ID switches. But I didn't succeed in generating the intended Dataset architecture (i lost too much time on this part) so I sadly couldn't use it.

— **Main process**

In the main process, I only had to call PedestrianDetector to generate the required data and finally use it as base with the TP4 implementation of the project.

Challenges

Track Eval toolkit setup

As mentioned before, I sadly was unable to complete quantitative evaluation due to configuration parsing errors in the evaluation toolkit. To this day I still haven't succeeded without having an error somewhere in the process, even if I go and change the source code.

6 Where to find final results ?

Output videos location

All tracking results can be found in the following directories :

— TP1 : `tp1/results/output_tracking.avi`
— TP2 : `tp2/results/tracking_result.avi`
— TP3 : `tp3/results/tracking_result.avi`
— TP4 : `tp4/results/tracking_result.avi`
— TP5 : `tp5/results/tracking_result.avi`

Analysis of final implementation (TP5)

The system effectively maintains object identities across frames through the final pipeline (Kalman filtering / IoU, and ReID combination), even with spatial overlap, and appearance matching. But sadly, it does face some challenges with unpredictable movements that can lead to ID switches. Some boxes also appear from nowhere which can be explained by an error in my code / bad parameters fine tuning. We could also improve the performance by using more sophisticated model instead of the current lightweight YOLOv5 model.