Compte-rendu Projet MLVOT

TP1 - TP2 - TP3 - TP4 - TP5

Epita - MLVOT

Décembre 2024

Clovis Lechien *(clovis.lechien@epita.fr)*

Table des matières

1 TP1 **3**
1.1 Description du sujet 3
1.2 KalmanFilter.py 3
1.3 objTracking.py 4

2 TP2 **4**
2.1 Description du sujet 4
2.2 Loading and checking the data 4
2.3 Main logic 4

3 TP3 **5**
3.1 Description du sujet 5
3.2 Main logic 5

4 TP4 **6**
4.1 Description du sujet 6
4.2 Main logic 6

5 TP5 **7**

1 TP1

1.1 Description du sujet

The objective of this practical is to implement a `Single Object Tracking (SOT)` system in a 2D environment using a `Kalman Filter`. The object is represented as a point, specifically its centroid, which simplifies the tracking problem to maintaining the coordinates of this point over time.

The `Kalman Filter` serves as the core component for predicting the future position of the object and updating this prediction based on new measurements.

The filter operates in two phases :

— **Prediction** : Estimates the object's next state (position and velocity) based on a motion model and control inputs.

— **Update** : Corrects the prediction using new measurements from the object detection algorithm, computing the Kalman gain and updating the state estimate and error covariance.

1.2 KalmanFilter.py

This file implements the `Kalman Filter` with these three functions :

— `__init__(self, dt, u_x, u_y, std_acc, x_std_meas, y_std_meas)`

— `predict(self)`

— `update(self, z_k)`

First, we call the `__init__` function all of these parameters :

— `dt` : time for one cycle used to estimate state (sampling time)

— `u_x, u_y` : accelerations in the x-, and y-directions respectively

— `std_acc` : process noise magnitude

— `x_std_meas, y_std_meas` : standard deviations of the measurement in the x- and y-directions respectively

With which we can define these fields in our `KalmanFilter class` :

— `self.u` : Control input variables.

— `self.x_k` : Initial state matrix.

— `self.A, self.B` : Matrices describing the system model A,B with respect to the sampling time dt.

— `self.H` : Measurement mapping matrix H.

— `self.Q` : Initial process noise covariance matrix Q with respect to the standard deviation of acceleration (std_acc).

— `self.R` : Initial measurement noise covariance R.

— `self.P_k` : Covariance matrix P for prediction error.

The `predict` and `update` functions update the time state and calculate the error covariance.

1.3 objTracking.py

Serves as the main file that integrates object detection and tracking.
Uses the KalmanFilter class and the Detector alreay implement in the loop.

2 TP2

2.1 Description du sujet

The objective of this practical is to develop a `Simple IoU-Based Tracker` for object tracking using `bounding boxes`. The tracker is further extended to handle `Multiple Object Tracking (MOT)` by assigning unique identifiers to each tracked object and updating these tracks over time.

2.2 Loading and checking the data

To work properly with the data and choose the best file, the first step involves loading and checking the detection files.

The `load_data` function is used to read the CSV files containing detection data into a pandas DataFrame. The files are parsed with specified column names, including fields for frame number, bounding box coordinates, detection confidence, and other optional attributes.

Once the data is loaded, the `check_data` function provides an overview of its structure and quality. This function displays essential information such as general metadata (info()), statistical summaries (describe()), the count of null entries (isnull().sum()), and data types (dtypes).

The use of color-coded messages (via the `bcolors` class) ensures that the output is visually distinct and easily interpretable, helping to quickly identify potential issues like missing values or unexpected data types. The `plot_img` and `plot_img_bboxs` functions are used to visualize the images from the datasets.

With this in mind, after comparing the 3 files (`public-dataset, Yolov5s, Yolov5l`), I chose to work with `Yolov5s` since it has similar performances as Yolov5l with the added benefit of coming from a smaller model.

2.3 Main logic

The `get_current_list_bbox` function extracts bounding box information for a given frame from the dataset. This function converts the detection data into a list of bounding boxes, each defined by its top-left corner coordinates, width, and height.

The `get_IoU` function computes the Intersection-over-Union (IoU) between two bounding boxes. Using this metric, the `get_sim_matrix` function creates a similarity matrix for all pairs of tracked objects and new detections in a given frame, where each entry represents the IoU score.

The `get_assignments` function uses the Hungarian algorithm to solve the optimal assignment problem based on the similarity matrix. This step ensures that each detection is assigned to the most likely track or marked as unmatched.

The `update_tracks_with_ids` function manages and updates tracks by :
— Matching detections to existing tracks.
— Identifying unmatched detections and creating new tracks with unique IDs.
— Removing tracks that exceed the maximum number of missed frames.
— This step ensures the continuity and accuracy of tracking across frames.

The `get_all_tracked_bboxs` function processes the entire dataset, applying the tracking logic frame by frame. It generates a comprehensive list of tracked bounding boxes, similarity matrices, assignments, and matches for all frames.

The `generate_frames` function overlays bounding boxes and track IDs onto video frames, creating a visual representation of the tracking process. The `save_video` function compiles these annotated frames into a video for playback. The `save_tracking_results` function saves the tracking results in a text file formatted according to the MOT-challenge standard, enabling further evaluation and analysis. Main Workflow : The `main_loop` and `main_loop_full` functions tie together the tracking process. They load the detection data, perform tracking, and output the results as a video and a tracking result file. The extended `main_loop_full` function also includes data checking and intermediate visualizations to help verify the processing pipeline.

Overall, the file implements a complete IoU-based tracking system that accurately tracks multiple objects, visualizes the results, and saves the outcomes in standard formats.

3 TP3

3.1 Description du sujet

This file implements a Kalman-Guided IoU Tracking system, which combines the predictive capabilities of a Kalman Filter with IoU-based detection-to-track assignment.

By extending the IoU-based Multiple Object Tracker (MOT) from the second practical, this approach improves tracking accuracy and robustness by incorporating motion prediction for smoother and more consistent tracking.

We reuse a lot of components from the first and second practical, the explanations will be shorter.

3.2 Main logic

A `KalmanTracker` class is introduced to predict and update object states (bounding box properties like position and size) using the Kalman Filter. The Tracker class manages multiple `KalmanTracker` instances, updating their states based on new detections while handling matches, unmatched tracks, and new detections.

Each `KalmanTracker` instance maintains the state of an object, represented by the centroid (cx, cy) and dimensions (w, h).

Before assigning detections, the Kalman Filter predicts the next state of each tracked object using the motion model.

After associations, the Kalman Filter updates the state of matched tracks based on new detections.

Tracks that are successfully matched with detections are updated with the new state.

Tracks that are not matched are removed after exceeding a threshold of missed frames.

Unmatched detections create new tracks with unique IDs, initialized as KalmanTracker instances.

The tracker predicts object motion, improving robustness against occlusion and detection noise.

The association process considers both spatial overlap (IoU) and predicted motion, leading to more accurate tracking over time.

This enhanced tracker is more reliable for scenarios involving multiple moving objects with partial occlusions or noisy detections.

4 TP4

4.1 Description du sujet

This practical extends the IoU-Kalman Tracker to include object re-identification (ReID), enabling more robust tracking by integrating appearance similarity. The tracker combines geometric and appearance information to associate detected objects with tracked objects across frames.

4.2 Main logic

A pre-trained lightweight deep learning model is used to extract appearance feature vectors from detected object patches. The `FeatureExtractor` class performs this extraction using ONNX runtime.

Detected bounding box patches are resized to match the model's input dimensions. The patches are converted from BGR to RGB, normalized, and transformed into tensors suitable for model inference.

Feature vectors from the current frame's detections and the previous frame's tracked objects are compared using metrics like cosine similarity or Euclidean distance. These metrics quantify appearance similarity.

The `get_combined_sim_matrix` function combines IoU scores and normalized feature similarity scores to produce a robust similarity matrix.

The IoU metric captures geometric overlap, while the appearance similarity enhances robustness against occlusion or noisy detections.

Feature similarity scores are normalized to ensure comparability with IoU values.

The Hungarian Algorithm is applied to the combined similarity matrix to find the optimal assignments between tracked objects and new detections. This ensures that both geometric and appearance cues are considered.

The inclusion of ReID makes the tracker more resilient to challenges such as :

— Occlusions : Appearance similarity helps recover tracks when objects reappear.

— Ambiguities : Differentiating visually similar objects improves association accuracy.

— Long-Term Tracking : Appearance features allow re-identification of objects even after significant frame gaps.

5 TP5

This practical focuses on enhancing the Appearance-Aware IoU-Kalman Tracker by integrating an efficient deep learning-based object detector for pedestrian detection. The goal is to improve the accuracy and speed of detection while maintaining the system's ability to track multiple objects robustly. Additionally, the performance of the tracking system is evaluated using standard metrics from the ground truth dataset.