MLVOT

Individual Report

Matis Braun – SCIA 2025

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

Ici, nous devions implémenter le filtre Kalman. L'énoncé étant assez clair, il n'y a pas eu énormément de difficultés, et tout a pu être intégré.

La seule difficulté qu'il y a eu a été sur la définition de la matrice d'état initiale. Je l'avais initialement défini ainsi en ligne :

self.x_k = np.asarray([0, 0, 0, 0])

Alors qu'il fallait la définir en colonne comme ceci :

self.x_k = np.asarray([[0], [0], [0], [0]])

Comme on peut le voir sur la vidéo *'tracking_output_tp1.avi'*, tout fonctionne correctement, et nous avons la trajectoire sauvegardée au fil du temps.

La vidéo est sur le Git et est nommé *'Result_TP1_SingleObjectTracking.avi'*

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

Dans ce TP, nous devions créer un tracker IoU (Intersection Over Union) avec des bounding boxes. Nous allons implémenter un suivi de piétons dans une séquence vidéo à partir de détections déjà enregistrées, puis nous produirons un fichier de sortie contenant, pour chaque frame, les boundings boxes et leurs identifiants.

Pour cela, nous avons une vidéo dont chaque frame est stockée image par image pour que nous puissions itérer dessus dans l'ordre. On nous a également fourni aussi 3 différents fichiers textes contenant les résultats de détections sur chaque frame, venant de 3 modèles différents : Yolov5s, Yolov5l et un du site du challenge. Nous récupérerons ainsi les positions de chaque bounding box dans chaque frame pour faire notre tracker.

Voici les éléments que j'ai intégré pour cela :

- Un chargement des détections en lisant les fichiers et j'ai organisé les données en Dataframe. J'ai aussi fait une fonction pour extraires les coordonnées des bounding boxes dans le Dataframe pour chaque frame donnée.
- Le calcul de l'IoU entre les bounding boxes suivies et détectées puis la constrution de la matrice de similarité.
- L'algorithme Hugarian pour optimiser les correspondances.
- La gestion des tracks avec 3 possibilités : création d'un track pour chaque détection non associées / mise a jour de la position et remise à zero du compteur de frame manquées pour chaque track associé / suppression si le compteur de frames manquées du track est au maximum.
- Sauvegarde des résultats de détections sur chaque frame dans un fichier texte.

Pour ce TP, il y a eu beaucoup plus de difficultés. La gestion des tracks a été la plus grande difficulté. J'ai fait beaucoup d'erreurs lorsque j'ai codé l'algorithme, en utilisant dans un premier temps un tableau puis un dictionnaire pour plus de simplicité pour stocker les tracks, puis en codant mal la gestion des ID des tracks, qui faisait qu'ils n'étaient par exemple jamais réutilisés, que je les effaçais (donc les ID revenaient à 1), … Mais ceci sont des erreurs de codes que j'ai juste corrigé au fil du temps.

Dans un second temps, de nombreuses d'erreurs faisait que la similarité était presque toujours hors du seuil, donc le tracker n'associait jamais la détection à un track et donc générait un nouvel ID. J'ai donc augmenté la tolérance. Par exemple pour *'max_missed_frames'* je suis passé de 2 à 10, ou alors pour le seuil de similarité je suis passé de 0.7 à 0.3. Cela a nettement amélioré les résultats.

La vidéo est sur le Git et est nommé *'Result_TP2_IoUTracking.avi'*

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

Pour ce TP, nous avons amélioré le système de tracking du TP2 (basé sur l'IoU et l'algorithme Hungarian) en intégrant un filtre Kalman (celui du TP1) dans le processus. Cela permet de faire une meilleure prédiction de la position des piétons sur la vidéo entre 2 frames.

Voici les éléments que j'ai intégré pour cela :

- Une fonction qui convertit la représentation d'une bounding box en centroïde pour pouvoir utiliser le filtre Kalman et vice-versa.
- L'intégration du filtre Kalman que j'ai repris du TP1.
- L'ajout du filtre dans le processus de tracking en faisant l'IoU et Kalman. Le processus est :
    - On va lire dans un premier temps les bounding boxes dans le Dataframe

    - On utilise le filtre Kalman pour predire les positions en centroide qu'on convertit ensuite en bounding boxes.
    - On calcule l'IoU entre les boundings boxes prédites (par Kalman) et détectées et on construit la matrice de similarité
    - On fait l'algorithme Hungarian
    - On met a jour le filtre Kalman
    - On fait le gestion des tracks
- Sauvegarde des résultats de chaque frame dans un fichier texte.

Pour ce TP, il n'y a pas eu énormément de difficultés. L'astuce de représenter les boundings boxes par leurs centroïdes a bien aidé. Le plus difficile aura été l'ajout du filtre Kalman dans le processus.

Sinon, il y a un léger problème qui n'a pas été résolu, et qui est probablement dû à de mauvaise estimation du filtre Kalman. Comme on peut le voir sur la vidéo résultat du TP3, les bounding boxes changent parfois d'ID lorsqu'elles varient en taille. Cela est dû à une mauvaise prédiction du filtre Kalman, mais aussi à d'autres facteurs (IoU,...). A cause de cela, j'ai choisi de ne pas actualiser trop souvent la taille des boundings boxes, car cela faisait changer rapidement les id. Cela permet de garder une meilleure cohérence des ID tout en comprenant quelle est la cible piétonne de la bounding box même si elle est plus grande/petite.

La vidéo est sur le Git et est nommé *'Result_TP3_KalmanIoU.avi'*

TP4 : Appearance-Aware IoU-Kalman Object Tracker :

Pour ce TP, nous devions intégrer un modèle de Re-Identification pour améliorer le suivi. Nous combinons l'IoU, le filtre Kalman et le ReID, afin de ne pas nous baser uniquement sur l'information géométrique de l'IoU et de pouvoir ajouter la ressemblance d'apparence entre deux frames. Pour cela, nous allons utiliser le modèle OSNet en utilisant le ficher *'reid_osnet_x025_market1501.onnx'* fourni.

Voici les éléments que j'ai intégré pour cela :

- La configuration du modèle Re-ID
- L'extraction du patch de chaque bounding box, le prétraitement, la récupération du vecteur de caractéristiques et l'utilisation de la distance de métrique L2.
- On fait la formule de score en combinant l'IOU et la similarité
- On ajoute la gestion du filtre Kalman et du tracking
- Sauvegarde des résultats de détections sur chaque frame dans un fichier texte.

Pour ce TP, il y a eu beaucoup de problèmes. Cela a a été assez long à les résoudre. Dans un premier temps, la configuration du modèle a été fastidieuse, que ce soit dans

sa configuration, son utilisation, le prétraitement, … qui ont pu être réglé en regardant la documentation et internet. Il y a aussi des problèmes liés à différents paramètres comme le seuil de confiance, le *'max_missed_frame'*, les poids alpha et béta de la formule de score que j'ai dû régler individuellement. En revanche, l'implémentation dans le processus déjà existant n'a pas été difficile.

La vidéo est sur le Git et est nommé *'Result_TP4_AppearanceKalmanIoU.avi'*

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

Pour ce TP, nous allons ajouter un modèle de détection (Yolov8n). Cela va nous permettre de ne plus utiliser le fichier texte des détections déjà enregistrées, mais d'utiliser directement un modèle Deep Learning en temps réel.

Voici les éléments que j'ai intégré pour cela :

- Le chargement du modèle Yolo et sa configuration
- La fonction de détection des piétons
- Puis tout ce qui a déjà été fait dans le TP4

Pour ce TP, il n'y a pas eu de problème. Toutefois, pour la question bonus, il y en a eu. J'ai essayé d'évaluer mon projet de tracking, mais je n'ai pas pu les résoudre. J'ai essayé de régler les problèmes pendant plusieurs heures mais je n'ai pas réussi (problème de seqmap introuvable alors que j'ai reproduit les chemins exacts et les fichiers,…). Vu que c'était un bonus, je n'ai pas persévéré davantage.

La vidéo est sur le Git et est nommé *'Result_TP5_Extension.avi'*