TP3 – Séquences Vidéo & Tracking

Tracking multi-objets sur vidéos
OpenCV (CSRT) & YOLOv8 + Association IoU
Bonus : Filtre de Kalman & sweep de paramètres

Wassim Chikhi

Master 2 Vision et Machine Intelligente – 2025/2026

1. Objectifs du TP

L'objectif est de construire un pipeline reproductible de **tracking multi-objets** sur plusieurs vidéos (croisements / occlusions), en comparant :

- **Q1** : tracking *baseline* par trackers classiques OpenCV (CSRT), initialisés à la main (sélection de ROIs).
- **Q2** : tracking *detection-based* avec **YOLOv8** pour détecter à chaque frame, puis association temporelle par **IoU**.
- **Bonus 1** : lissage de trajectoire via un **Filtre de Kalman** (position + vitesse) appliqué sur une trajectoire (centroïdes YOLO).
- **Bonus 2** : étude de paramètres (conf et IoU threshold) : compromis **vitesse vs stabilité** (proxy : nombre de nouveaux IDs).

Dépôt GitHub : TP3 Multi-Object Tracking

2. Données et organisation du projet

2.1. Données

Les vidéos testées sont :

- `Tracking2min.avi` : scène multi-objets (petits objets, nombreux déplacements).
- `Walk1.mpeg` : suivi de personne(s) dans une scène intérieure.
- `redcaroverlay3.mp4` : suivi de véhicule (supporté par le pipeline Q2 en changeant la classe).

2.2. Organisation recommandée

- `tracking/` : vidéos d'entrée
- `tracking_results/` : sorties (`.mp4`, `.csv`, figures)
- `src/` : scripts (`q1_csrt.py`, `q2_yolo_iou.py`, `bonus_kalman.py`, `bonus_param_sweep.py`)

3. Question 1 – Tracking baseline multi-objets (CSRT)

3.1. Principe

Le tracker **CSRT** d'OpenCV est initialisé sur la première frame à partir de **bboxes sélectionnées manuellement**. Ensuite, les trackers évoluent frame par frame à partir de l'apparence locale (*template tracking*).

3.2. Procédure

— Lecture de la frame 0
— Sélection manuelle des ROIs (deux clics par objet : haut-gauche puis bas-droit)
— Initialisation de plusieurs trackers CSRT (`cv2.TrackerCSRT_create()`)
— Boucle vidéo : mise à jour, dessin des bboxes, export `mp4`

3.3. Résultat

La Figure 1 illustre un exemple de tracking CSRT multi-objets (IDs affichés).

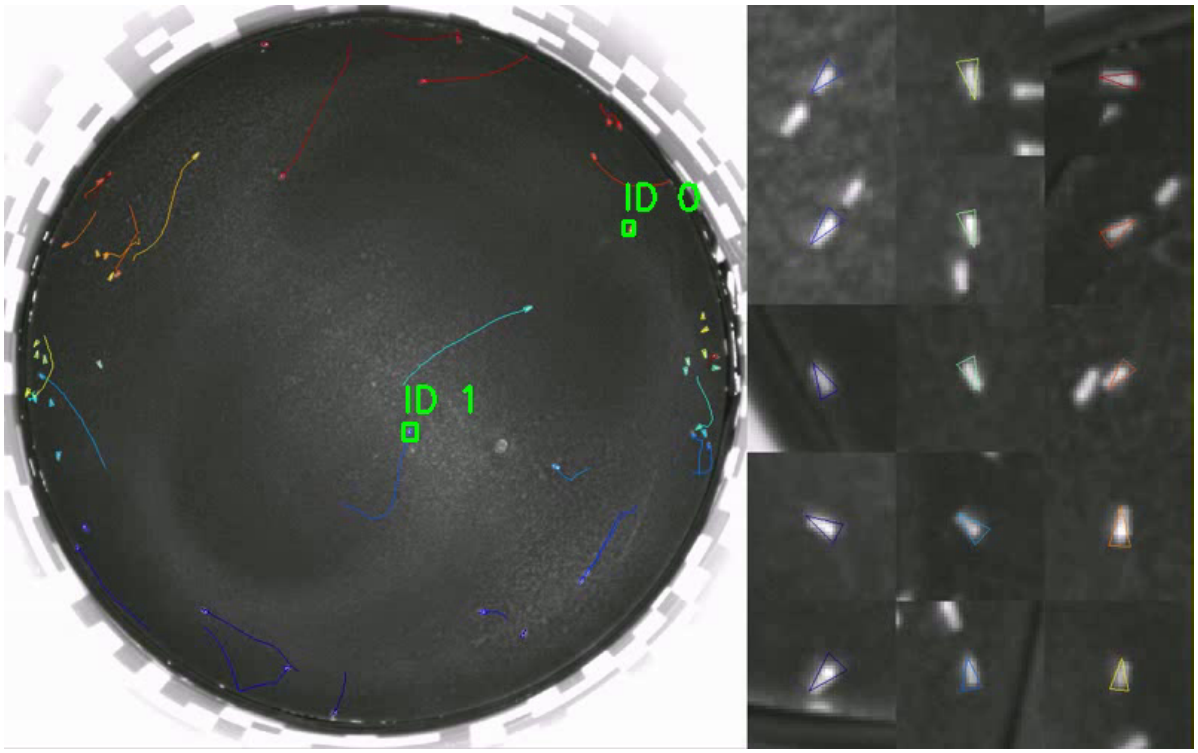

FIGURE 1. Q1 — Exemple de tracking baseline par CSRT (OpenCV) avec IDs.

3.4. Discussion (Q1)

CSRT peut donner de bons résultats à court terme mais reste sensible :

— aux changements d'échelle / illumination,
— aux occlusions longues,
— au drift lorsque l'apparence change.

Le point fort est sa simplicité et le fait qu'il n'exige pas de détecteur.

Bonus : Filtre de Kalman & sweep de paramètres

4. Question 2 – Détection YOLO + Tracking par association IoU

4.1. Principe

On détecte les objets à chaque frame avec **YOLOv8** (Ultralytics), puis on associe les détections entre frames via **IoU** :

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Chaque piste conserve sa bbox précédente, et une détection est assignée au track ayant le meilleur IoU au-dessus d'un seuil.

4.2. Détails d'implémentation

— Seuil de confiance YOLO : `conf` (ex : 0.25 ou 0.35)
— Seuil d'association : `iou_thr` (ex : 0.2–0.4)
— TTL (frames de tolérance si une détection manque) : ex 10

4.3. Résultat

Exemple sur `Walk1.mpeg` avec la classe `person` :

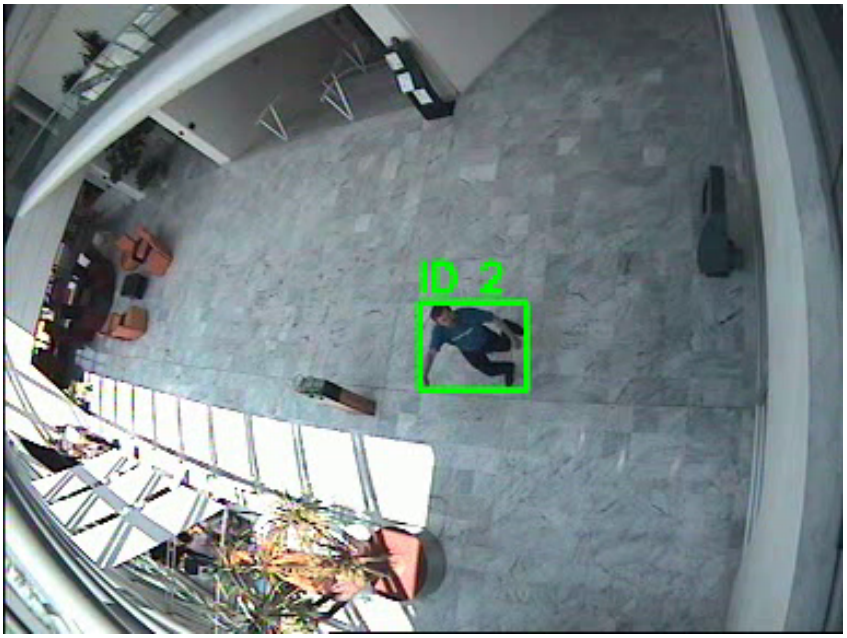

FIGURE 2. Q2 — Tracking par YOLOv8 + association IoU (exemple `person`).

4.4. Discussion (Q2)

Avantages :

— ré-acquisition possible après occlusion (si YOLO redétecte),
— meilleure robustesse aux variations visuelles qu'un tracker purement *template*.

Limites :

— risque de **switch d'ID** en cas de croisement,
— dépend de la qualité des détections (faux négatifs ⇒ tracks cassés).

5. Bonus 1 – Filtre de Kalman (position + vitesse) sur centroïdes YOLO

5.1. Principe

On considère un modèle à vitesse constante :

$$\mathbf{x}_t = [x_t, y_t, v_{x,t}, v_{y,t}]^T$$

Le filtre de Kalman prédit la position même si la mesure YOLO est manquante (données `NaN`).

5.2. Résultat

La figure ci-dessous compare les points mesurés (YOLO) et la trajectoire prédite (Kalman) sur 250 frames.

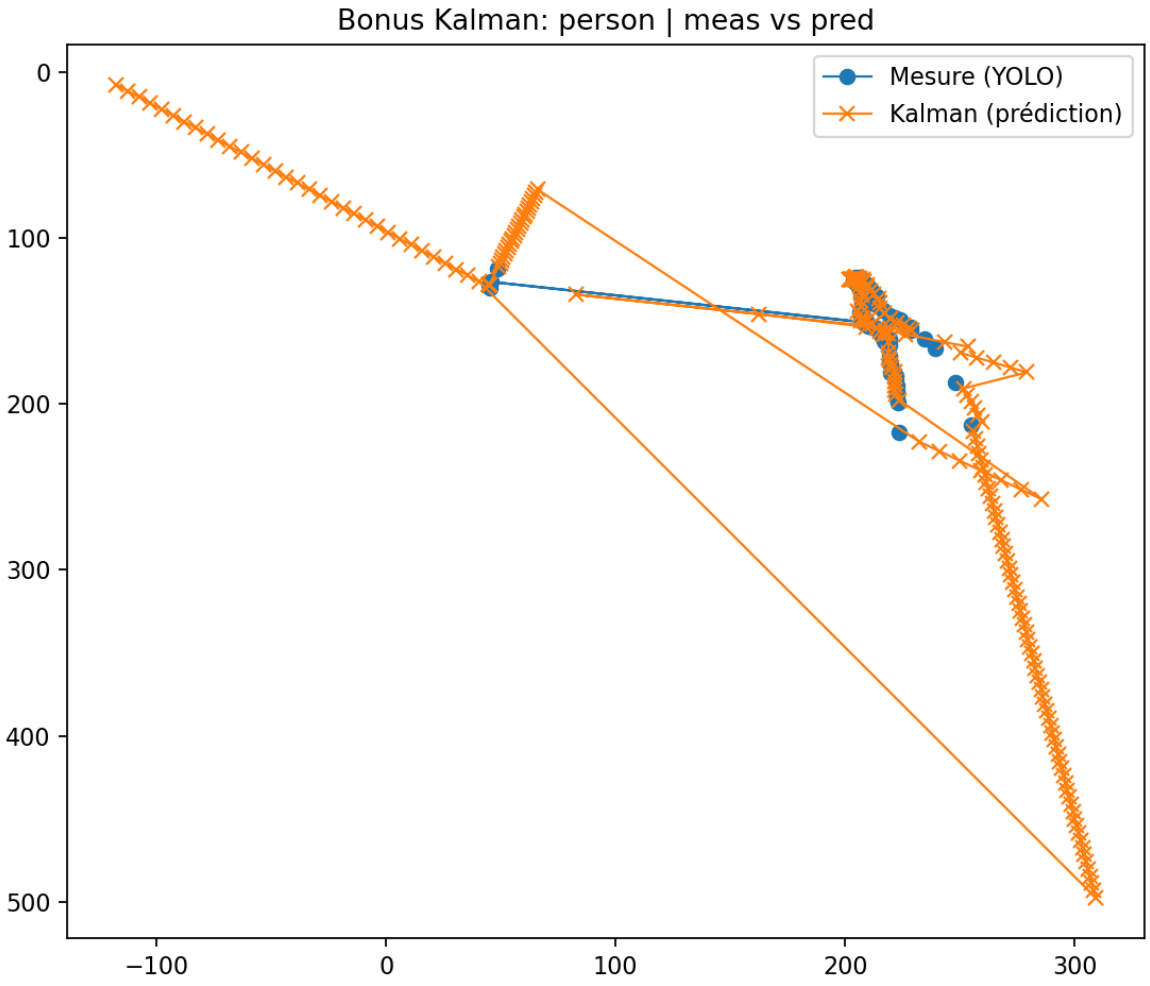

FIGURE 3. Bonus Kalman — comparaison trajectoire mesurée (YOLO) vs prédite (Kalman).

5.3. Commentaires

On observe que Kalman :

— **lisse** le bruit des mesures,
— **interpole** pendant les frames sans détection,
— peut diverger si les mesures sont très rares ou si un saut d'ID survient.

6. Bonus 2 – Sweep paramètres : vitesse vs stabilité

6.1. Protocole

On teste plusieurs couples $(conf, iou\_thr)$ sur `Walk1.mpeg`. On mesure :

— **fps** : vitesse moyenne de traitement,
— **newIDs_proxy** : nombre de nouveaux IDs créés (proxy d'instabilité),
— **tracks_end** : nombre de tracks actifs en fin de séquence.

6.2. Résultats (extraits CSV)

Les valeurs obtenues (60 frames) sont :

conf	iou_thr	frames	seconds	fps	newIDs_proxy	tracks_end
0.25	0.20	60	7.5267	7.9716	2	1
0.25	0.30	60	6.6808	8.9809	2	1
0.35	0.20	60	6.5583	9.1488	2	1
0.35	0.30	60	6.9323	8.6552	2	1

TABLE 1. Bonus 2 — Sweep paramètres (extrait de `bonus_param_sweep.csv`).

6.3. Conclusion bonus

Dans ces essais courts :

— Les FPS restent proches (8–9 fps).
— Le proxy d'instabilité (`newIDs_proxy`) est identique (2) ⇒ séquence relativement simple sur 60 frames.
— Une étude plus longue (600 frames) augmente le temps de calcul et peut révéler davantage de nouveaux IDs.

7. Conclusion

Ce TP met en évidence deux approches complémentaires :

— **CSRT (Q1)** : simple à lancer et efficace quand l'objet reste visible, mais sensible aux occlusions et au drift.
— **YOLO + IoU (Q2)** : plus robuste aux variations, permet de réinitialiser après perte, mais dépend des détections et peut générer des switchs d'ID.

Les bonus montrent :

— l'intérêt d'un **Kalman** pour lisser/interpoler,
— comment un **sweep** donne un compromis vitesse/stabilité.

Auteur : Wassim Chikhi **Année :** 2025/2026