Reporte proyecto Final DeepSORT

Profesores: Dr. Jean Bernard Hayet & Dr. Francisco Javier Hernández
Alumno: Mario Alberto Tapia de la Cruz

Maestría en Matemáticas Aplicadas

1. Introducción

El algoritmo de DeepSORT es una mejora al algoritmo SORT (mejor conocido como Simple Online and Realtime Tracker). Aquí se introduce una métrica de asociación basada en métodos de Deep Learning, esto nos permite identificar objetos a lo largo de objetos e incluso cuando se presentan oclusiones. Así podemos decir que DeepSORT aprende y entiende la apariencia visual de los objetos de forma más robusta y precisa. Teniendo así un mejor manejo de escenarios de seguimiento complejos donde otros algoritmos tienden a fallar.

2. Un repaso a SORT y sus fallos

SORT es un acercamiento al seguimiento de objetos donde se utilizan métodos rudimentarios como el Filtro de Kalman y el algoritmo Húngaro para el seguimiento de objetos, además pretende ser mejor que otros trackers. SORT tiene 4 componentes principales que se estudiarán a continuación:

1. Detección: Se detectarán los objetos en un frame a los que se les hará seguimiento. Para este paso se utiliza un detector de objetos como lo es FrRCNN o YOLO (aquí se hará detección de objetos usando YOLOv8 y YOLOv12

2. Estimación:Propagamos las detecciones entre frames, para ello estimamos la posición del objetivo en el siguiente frame, estamos tomando un modelo de velocidad constante. Una vez que la detección se asocia con un objetivo actualizamos la posición con una bounding box, estos componentes de velocidad se calculan a través de un Filtro de Kalman con el que entraremos a detalles más adelante.

3. Asosciación de Datos: Ya tenemos una bounding box de un objetiva detectada en frames anteriores, y una bounding box detectada para el frame actual. Ahora construimos una matriz de costos (por ejemplo la distancia IoU) entre detecciones y todas las predicciones de cajas delimitadoras de nuestros objetos existentes.

4. Creación y eliminación de Identidades del Tracking: Finalmente se crean nuevos IDs cuando aparecen nuevos objetos, y se eliminan IDs antiguos cuando los objetos dejan de estar a la vista por un tiempo prolongado. Todos estos IDs se crean o destruyen de acierdo con el umbral mínimo del IoU. Si el solapamiento entre una detección y un objetivo es menor que IoU_min, se considera un objeto no rastreado, así que se le asigna un nuevo ID; o bien, las trayectorias se terminan si no se detectan después de cierto número de cuadros perdidos (T_lost) (tú puedes especificar esta cantidad de cuadros). Si el objeto reaparece después de haber sido eliminado, se le asignará una nueva identidad. Justamente esta parte es donde DeepSORT comienza a tener muchos problemas.

Los objetos pueden ser rastreados exitosamente usando el algoritmo SORT, superando a muchos algoritmos del estado del arte. El detector nos da las detecciones, los filtros de Kalman nos proporcionan las trayectorias, y el algoritmo húngaro se encarga de la asociación de datos.

Entonces, ¿por qué necesitamos DeepSORT? Veámoslo en la siguiente sección.

3. DeepSORT

SORT funciona muy bien en términos de precisión y exactitud en el seguimiento. Pero SORT devuelve trayectorias con un alto número de cambios de ID y falla en casos de oclusión. Esto se debe a la matriz de asociación que utiliza. DeepSORT emplea una mejor métrica de asociación que combina tanto el movimiento como los descriptores de apariencia. DeepSORT puede definirse como un algoritmo de seguimiento que rastrea objetos no solo en función de su velocidad y movimiento, sino también de su apariencia.

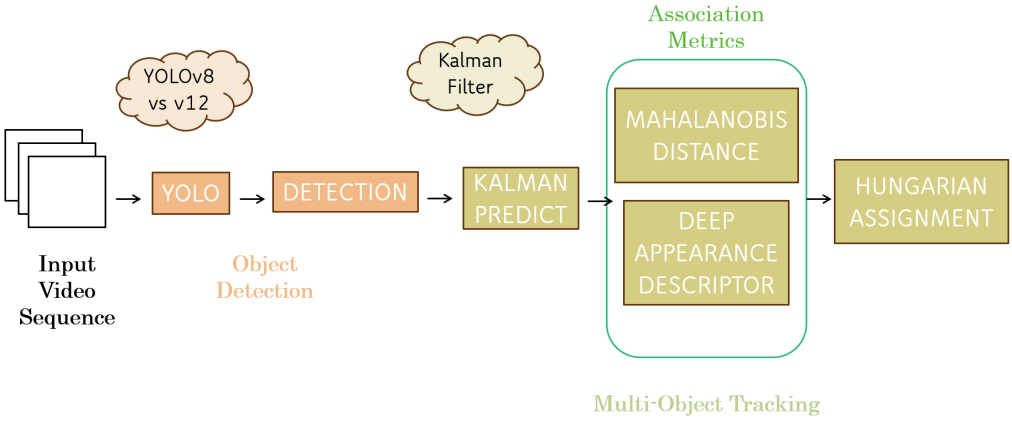

Figura 1: Pipeline de DeepSORT

Para este propósito, se entrena previamente (offline) un feature embedding bien discriminativo justo antes de implementar el seguimiento. La red se entrena con un conjunto de datos a gran escala de reidentificación de personas (la red usada es MARS), lo cual la hace adecuada para el contexto de seguimiento. Para entrenar el modelo de métrica de asociación profunda en DeepSORT, se utiliza un enfoque de aprendizaje de métrica con distancia coseno. Según el artículo de DeepSORT:

"La distancia coseno considera información de apariencia que es particularmente útil para recuperar identidades después de oclusiones prolongadas, cuando el movimiento es menos discriminativo." Como señalan los autores de DeepSORT [6]:

Esto significa que la distancia coseno es una métrica que ayuda al modelo a recuperar identidades en caso de oclusiones de largo plazo, incluso cuando la estimación por movimiento falla. Usar estos elementos simples puede hacer que el rastreador sea aún más potente y preciso.

4. Filtro de Kalman

Para esto vamos a considerar las ecuación de estado, estas son lineales y tienen procesos de ruido aditivo:

$$X_{k+1} = F_{k+1}X_k + W_{k+1}, \quad k = 0, 1, 2, \ldots, \tag{1}$$

$$Y_k = G_k X_k + V_k \quad k = 1, 2, \ldots. \tag{2}$$

Aquí suponemos que F_{k+1} y G_k son matrices conocidas. más aún, los vectores de ruido W_{k+1} y V_k son Gaussianos con medias y covarianzas conocidas. Sin pérdida de generalidad, suponemos que son vectores con media cero. Por último, los vectores de ruido son mutuamente independientes, esto es:

$$W_k \perp W_l,\ V_k \perp V_l,\ k \neq l$$

y

$$W_k \perp V_l$$

Esto quiere decir que W a diferentes instantes de tiempo no están correlacionados, similarmente para V en diferentes instantes de tiempo, y además que el ruido en la dinámica del sistema es independiente del ruido de las observaciones. Finalmente, la distribución de probabilidad de X_0 es conocida y Gaussiana, con media cero. Bajo estas restricciones el siguiente teorema se cumple.

Teorema 1. *Suponga que se cumplen las suposiciones anteriores. Denotemos* $x_{k|\ell} = \mathbb{E}(x_k \mid D_\ell)$ *y* $\Gamma_{k|\ell} = \mathrm{cov}(x_k \mid D_\ell)$ *y definamos* $\pi(x_0) = \pi(x_0 \mid D_0)$. *Entonces, las fórmulas de actualización de la evolución temporal y de la observación son:*

1. *Actualización de evolución temporal: Suponga que conocemos la distribución Gaussiana de*

$$\pi(x_k \mid D_k) \sim \mathcal{N}(x_{k|k}, \Gamma_{k|k}).$$

Entonces,

$$\pi(x_{k+1} \mid D_k) \sim \mathcal{N}(x_{k+1|k}, \Gamma_{k+1|k})$$

donde

$$x_{k+1|k} = F_{k+1} x_{k|k}, \tag{3}$$

$$\Gamma_{k+1|k} = F_{k+1} \Gamma_{k|k} F_{k+1}^{\top} + \Gamma_{w_{k+1}}, \tag{4}$$

2. *Actualización de Observación: Suponga que conocemos la distribución Gaussiana*

$$\pi(x_{k+1} \mid D_k) \sim \mathcal{N}(x_{k+1|k}, \Gamma_{k+1|k}).$$

Entonces,

$$\pi(x_{k+1} \mid D_{k+1}) \sim \mathcal{N}(x_{k+1|k+1}, \Gamma_{k+1|k+1}),$$

donde

$$x_{k+1|k+1} = x_{k+1|k} + K_{k+1} \left(y_{k+1} - G_{k+1} x_{k+1|k} \right), \tag{5}$$

$$\Gamma_{k+1|k+1} = \left(I - K_{k+1} G_{k+1} \right) \Gamma_{k+1|k}, \tag{6}$$

y la matriz K_{k+1}, conocida como la matriz de Ganancia de Kalman, está dada por

$$K_{k+1} = \Gamma_{k+1|k} G_{k+1}^{\top} \left(G_{k+1} \Gamma_{k+1|k} G_{k+1}^{\top} + \Gamma_{v_{k+1}} \right)^{-1},$$

En el contexto de DeepSORT, el estado $\mathbf{x_k}$ tiene una interpretación dada, cada componenten representa una característica de la *bounding box* o su velocidad. Este vector de estados queda definido como:

$$\mathbf{x} = \begin{bmatrix} x & y & a & h & \dot{x} & \dot{y} & \dot{a} & \dot{h} \end{bmatrix}$$

donde:

- x, y representan el centro de la *bounding box*
- a representa el aspect ratio
- h representa la altura $\dot{x}$, $\dot{y}$, $\dot{a}$, y $\dot{h}$ representan sus velocidades correspondientes.

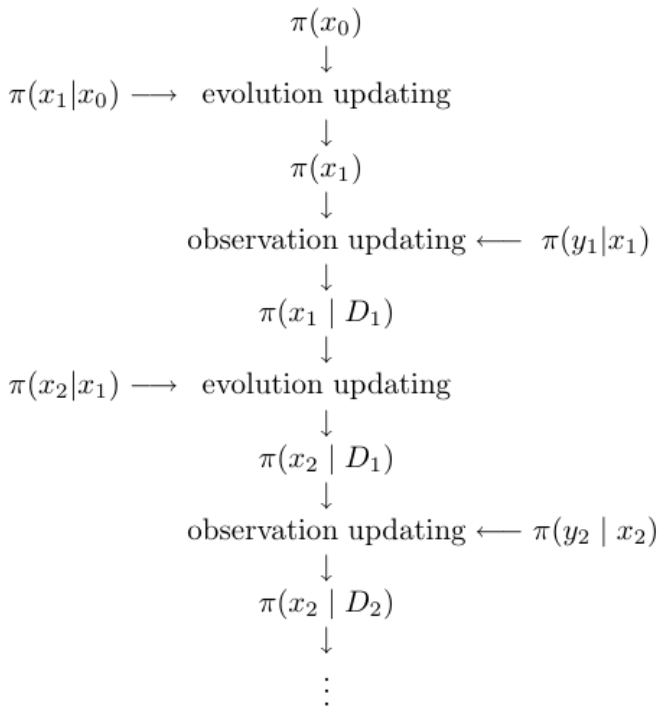

Figura 2: Diagrama de Filtro de Kalman

5. Experimentación

Para la experimentación se usaron tres vídeos a probar, *centro1_lq.MOV*, *centro2_lq.MOV* y *people.mp4*. Los primeros dos vídeos se tomaron con una cámara NIKON COOLPIX P610 y posteriormente se reescalaron los vídeos a la resolución de 640 por 380. El tercer vídeo (*people.mp4*) fue descargado de internet de un vídeo de referencia que se tomó de YouTube y tiene una resolución de 1280 por 720.

Además, se probaron para la detección los modelos YOLO pre-entrenados con COCO siguientes: *YOLOv8 nano, YOLOv12 nano* y *YOLO v12 medium* (En la carpeta de archivos del proyecto se puede encontrar los tres modelos cargados en archivos *.pt*. Además al hacer cada ejecución se le pedía que el `class_id == 0`, lo que significa que únicamente iba a detectar personas.

Todas las ejecuciones se realizaron en una CPU de 16 gb de memoria RAM, con un procesador Intel Core i7 13th.

Para hacer los conteos automáticos, en el for que me recorre los tracks en `tracker.tracks` se verificó con la función "*has attribute*"(`hasattr`) si el objeto track tiene el atributo "track_id", y si retorna `True` se guarda en un "`set()`"para evitar contar múltiples veces el mismo objeto. Luego, para hacer el cálculo del Accuracy para cada ejecución se utilizó

$$\text{Accuracy} = \left(\frac{1 - |CA - CM|}{CM}\right) \times 100\,\%$$

Donde CA son los conteos automáticos y CM los conteos manuales.

Modelo	Conteo Manual	Conteo Automático	Accuracy
YOLO v8 nano	42	71	30.95 %
YOLO v12 nano	42	67	40.48 %
YOLO v12 medium	42	73	26.19 %

Cuadro 1: Escena Centro1, resolución 640×380

Modelo	Conteo Manual	Conteo Automático	Accuracy
YOLO v8 nano	33	44	66.67 %
YOLO v12 nano	33	49	51.52 %
YOLO v12 medium	33	45	63.64 %

Cuadro 2: Escena Centro2, resolución 640×380

Modelo	Conteo Manual	Conteo Automático	Accuracy
YOLO v8 nano	56	79	58.93 %
YOLO v12 nano	56	52	92.86 %
YOLO v12 medium	56	33	59.93 %

Cuadro 3: Escena People, resolución 1280×720

6. Conclusiones

Los resultados de la Accuracy para cada ejecución fueron bastante variadas para los tres modelos de detección YOLO. En las escenas de *centro1* y *centro2* la cámara estaba posicionada de manera en que lo que grabara fuera el paso de peatones en su mayoría de izquierda a derecha y viceversa (salvo casos en donde se pueden ver peatones que vienen por un pequeño túnel a lo lejos de la escena). Mientras que en la escena de *people*, todos los peatones están siendo grabados desde una perspectiva "desde arriba", por lo que tal vez pueda ser más fácil el evitar solapamientos, pues hubo una gran cantidad de solapamientos en las escenas de *centro1* y *centro2*, lo que pudo haber ocasionado distintas detecciones distintas a una misma persona (por ejemplo en ocasiones pasaban familias juntas de a 3 o 4 personas posicionadas de manera muy cercana).

Para observar las funciones hechas, ir a la carpeta
\ProyectoVisionTapia\deep_sort\deep_sort\kalman_filter_tapia.py
Y para el archivo main accesar a: *\ProyectoVisionTapia\main.py*

Referencias

[1] Jari Kaipio and Erkki Somersalo. *Statistical and Computational Inverse Problems*, volume 160 of *Applied Mathematical Sciences*. Springer-Verlag New York, New York, 2004.

[2] Sanyam. Understanding multiple object tracking using deepsort, June 2022. Accessed: 2025-06-10.

[3] Yunjie Tian, Qixiang Ye, and David Doermann. Yolov12: Attention-centric real-time object detectors. *arXiv preprint arXiv:2502.12524*, 2025.

[4] Yunjie Tian, Qixiang Ye, and David Doermann. Yolov12: Attention-centric real-time object detectors, 2025.

[5] Nicolai Wojke and Alex Bewley. Deep cosine metric learning for person re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 748–756. IEEE, 2018.

[6] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.