P2: Multiple Object Traking for Video Sequences

Marlon Rodríguez Flor e Iván Pascual Shygapova

I. INTRODUCCIÓN

En este trabajo se aborda la tarea de seguimiento de múltiples objetos, específicamente personas caminando, en secuencias de video. Partimos de un modelo base de detección y seguimiento y utilizamos la base de datos MOT16, que contiene diversos escenarios para la detección de individuos. El objetivo principal de este trabajo es mejorar el modelo de partida mediante la implementación de varias técnicas avanzadas.

Para lograr esta mejora, se probaron las siguientes estrategias:

1) Detecciones de alta calidad: Asegurar que las detecciones iniciales sean precisas y confiables.
2) Uso del algoritmo YOLOv5: Entrenamiento e implementación de este modelo de detección para identificar personas en los videos, conocido por su rapidez y precisión.
3) Algoritmo húngaro para la asociación de datos: Utilización de este algoritmo para resolver el problema de asignación de detecciones a trayectorias, optimizando la correspondencia entre las detecciones de diferentes cuadros.
4) Manejo de trayectorias mediante contadores: Implementación de contadores para gestionar las detecciones fallidas y los falsos positivos, mejorando la robustez del sistema de seguimiento.

La métrica principal utilizada para evaluar el rendimiento del modelo mejorado es MOTA (Multiple Object Tracking Accuracy). Esta métrica proporciona una medida integral del desempeño del sistema, considerando falsos positivos, falsos negativos y errores de identidad. Con las técnicas aplicadas, se logró alcanzar un promedio de MOTA de 75.8% en los datos de prueba y un 68.1% en el conjunto completo de datos, lo cual representa una mejora significativa respecto al modelo base.

Además, se presenta el impacto de cada una de las técnicas implementadas mediante comparaciones visuales en cuadros específicos de las secuencias de video. Estas comparaciones demuestran cómo el modelo mejorado es capaz de detectar y seguir a las personas de manera más precisa y consistente en comparación con el modelo base.

En resumen, este trabajo no solo mejora el rendimiento del modelo base, sino que también muestra el potencial de las técnicas de detección y seguimiento utilizadas para optimizar la tarea de seguimiento de múltiples objetos en escenarios complejos. Las técnicas implementadas y los resultados obtenidos destacan la importancia de la combinación de detecciones de alta calidad, algoritmos eficientes de asociación de datos y un manejo robusto de trayectorias para lograr un sistema de seguimiento eficaz y preciso.

II. MÉTODO E IMPLEMENTACIÓN

En esta sección, exploraremos en detalle los métodos empleados y su implementación para mejorar el rendimiento en la tarea de seguimiento de múltiples objetos (MOT, por sus siglas en inglés), centrándonos específicamente en la detección y seguimiento de personas y sus respectivas trayectorias. Adicionalmente, es importante mencionar que la principal métrica utilizada para medir el rendimiento de los experimentos es la métrica MOTA.

A. Detección de Objetos

A continuación, se describen los enfoques implementados para mejorar el rendimiento en la tarea de detección de objetos. Adicionalmente, el código fuente de este apartado se encuentra en el archivo *object_detector.py*.

1) Detecciones de Alta Calidad: Para mejorar la detección de objetos en la implementación base del modelo Faster R-CNN con el *backbone* de ResNet-50, se incluyeron únicamente las detecciones de alta calidad, filtradas en función de la variable *scores*. Específicamente, se probaron diversas configuraciones de umbrales: 0.5, 0.6, 0.7, 0.8 y 0.9. El objetivo de estas pruebas fue identificar el umbral óptimo que maximice la precisión de las detecciones, asegurando que solo se consideren aquellas con una alta probabilidad de ser correctas. Al aplicar estos umbrales, el modelo filtra las detecciones de baja calidad, mejorando así la exactitud y confiabilidad de los resultados obtenidos.

La modificación en el código base para obtener detecciones de alta calidad se implementó en el método *detect* de la clase RCNN_FPN. A este método se le añadió el parámetro *threshold* como entrada. Posteriormente, se realiza el filtrado de *boxes* y *scores*, seleccionando las cajas y puntuaciones que superan el *threshold* asignado en función de la variable *scores*. Finalmente, se retornan las cajas y puntuaciones de alta calidad.

2) YOLOv5: YOLOv5 [1] es un conocido algoritmo de detección (y segmentación) de objetos. En el trabajo realizado, se ha utilizado el modelo medio (*yolov5m.pt*) aplicándole tanto *fine-tuning* como *transfer learning* utilizando los datos de entrenamiento; particularmente las secuencias MOT-02, MOT-04 y MOT-05 como set de entrenamiento y la secuencia MOT-09 como set de validación. Sin embargo, esta aproximación no ha resultado en un rendimiento superior y se ha quedado muy por detrás de la red Faster R-CNN con el *backbone* de ResNet-50 y el resto

de métodos implementados. Este método se ha dividido en dos submétodos:

1) Integración de YOLOv5 en el *tracker*: La integración de YOLOv5 se ha basado en el desarrollo de una clase *YOLOv5* dentro del módulo *object_detector.py*. Su estructura es sencilla, puesto que sólo se compone del *__init__* y el *detect*.
   a) Método *__init__*: El método es sencillo: solamente carga de PyTorch Hub (https://pytorch.org/hub/) el modelo de YOLOv5, indicando la ruta de los pesos a cargar; la única clase que queremos detectar y aplicando el *non maximum suppression threshold*.
   b) Método *detect*: El método *detect* es algo más complejo, puesto que pasa por transformar las imágenes a un formato que el modelo YOLOv5 pueda procesar. Toma como entrada un tensor de una sola imagen y este método lo transforma a una imagen PIL pasándola por una permutación y transformación a un array de *numpy*. Hecho esto, realiza la detección sobre la imagen, obtiene del tensor resultado separando los cuadros delimitantes de las confianzas y las devuelve.
2) Fine-tuning y transfer learning sobre YOLOv5: Se eligieron las secuencias MOT-02, MOT-04 y MOT-05 como corpus de entrenamiento y la secuencia MOT-09 para el de validación. En vídeo, es importante no mezclar fotogramas de una misma secuencia tanto en entrenamiento como en validación (como en test), porque entonces se estaría incurriendo en el error típico de revelarle al modelo la solución de validación. Después, siguiendo las instrucciones para el entrenamiento con datos propios [2], se convirtieron los ficheros *gt.txt* a varios ficheros de etiqueta con el formato YOLO. Para ello, se ha tenido que desarrollar el script *mot_to_yolo.py*, adjunto a la entrega, que lee los ficheros *gt.txt*, y va creando, por cada fotograma, un nuevo fichero de texto con el formato YOLO, con los valores normalizados entre [0, 1] como lo marca; para esto último, los valores de los llamados *bbox_left* y *bbox_width*, que son los de la posición en X y el ancho de la caja con respecto a X, se deben de dividir entre el ancho de la imagen y los valores de *bbox_top* y *bbox_height*, que son lo mismo que lo anterior para la Y y la altura, con el alto de la imagen, descritos en el propio fichero *gt.txt*. Además, estos valores se deben de centrar, puesto que YOLO describe las cajas delimitantes indicando, en vez de un extremo de la misma, el centro de ella. Por ello, la conversión suma *bbox_left* con *bbox_width* y lo divide entre dos para conseguir el centro en X, y lo mismo para Y. Hecho esto, se desarrolló el fichero YAML *dataset.yaml*, adjunto a la entrega, que indica al algoritmo dónde encontrar las imágenes de entrenamiento y validación y las clases a predecir (en este caso, sólo la de persona). Finalmente, se ejecutaron los entrenamientos sobre una GPU propia (una NVIDIA GeForce FTX 2060 de 6GB), para no estar sujetos al devenir de las GPUs de Google Colab, utilizando el script *train.py* y el resto de dependencias que se proporciona en el repositorio de YOLOv5 [3]. Para la aproximación del *fine-tuning* del modelo medio con los pesos ya entrenados que proporciona el repositorio, se llevó a cabo un entrenamiento por 20 épocas, cambiando el tamaño de las imágenes a 640x640 (por las limitaciones de memoria, a pesar de la pérdida de calidad), utilizando el tamaño de modelo medio (M, de nuevo, por temas de memoria), y congelando todas las capas excepto la última de detección (23 de las 24 que tiene el modelo), con el siguiente comando: *python train.py –img 640 –epochs 20 –data dataset.yaml –weights yolov5m.pt –freeze 23*. El mejor modelo conseguido durante este entrenamiento obtuvo un mAP_0.5:0.95 de 0.34 y un mAP_0.5 de 0.67. Al verlo insuficiente para el funcionamiento del algoritmo de detección, se pensó en el *transfer learning* como alternativa, pues podría suceder que el problema para el que se entrenó YOLOv5 (detección de objetos en el dataset COCO [4]) y el problema en cuestión se distanciaran mucho, igual por el tamaño de las imágenes y la cantidad de personas. Con el siguiente comando, se realizó el entrenamiento por otras 20 épocas, sobre el modelo pre-entrenado de Ultralytics y con el tamaño a 640x640: *python train.py –img 640 –epochs 20 –data dataset.yaml –weights yolov5m.pt*. Este modelo obtuvo un mAP_0.5:0.95 de 0.49 y un mAP_0.5 de 0.78, bastante superior, cuyos pesos se guardaron en un fichero *best.pt*, que luego se puede cargar desde el método *__init__* de *object_detect.py*.

B. Seguimiento de Objetos

A continuación, se describen los enfoques implementados para mejorar el seguimiento de objetos y manejo de trayectorias, con un orientación particular en la modificación de la asociación de datos. El código fuente correspondiente a este apartado se encuentra en el archivo *tracker.py*.

1) Cambio de matriz de costos: Con el objetivo de mejorar la asociación de datos, se experimentó con un cambio en la matriz de costos, la cual en el código base se implementa utilizando la métrica de Intersection over Union (IoU). Específicamente, se creó una matriz de costos en función de la distancia euclidiana entre los objetos seguidos existentes y los nuevos objetos detectados, con la esperanza de que esta métrica proporcionara una mejora en la precisión del seguimiento de objetos. La distancia euclidiana fue seleccionada porque puede capturar de manera efectiva las diferencias espaciales entre las posiciones de los objetos en diferentes cuadros, ofreciendo una perspectiva distinta a la de IoU, que se centra más en la superposición de áreas.

Para implementar este cambio, se modificó el método *data_association* en la clase *Tracker* del archivo *tracker.py*. Específicamente, se creó una matriz de costos basada en la distancia euclidiana, calculada utilizando la función *torch.dist*. Esta función se aplicó para medir la distancia entre los seguimientos existentes y las nuevas detecciones.

Sin embargo, los resultados obtenidos al realizar este cambio fue inferior a la matriz de costos del código base por lo que se mantuvo la matriz con IoU original.

2) Asociación de Datos: Para mejorar la asociación de datos, se utilizó el Algoritmo Húngaro, implementado mediante el método *linear_sum_assignment* de la librería Scipy [5]. Este algoritmo permite resolver el problema de encontrar la asociación óptima entre los objetos seguidos y las nuevas detecciones, garantizando que se minimicen los costos de asociación.

Este cambio se implementó en el método *data_association* de la clase *Tracker* en el archivo *tracker.py*. La nueva matriz de costos, que incorpora tanto la métrica de Intersection over Union (IoU), se utilizó para alimentar el algoritmo húngaro. Esto aseguró que las detecciones se asignaran de manera más precisa a los seguimientos existentes.

La utilización del Algoritmo Húngaro para la asociación de datos en el método *data_association* permitió una mejora notable en el rendimiento del sistema de seguimiento de objetos, demostrando ser una adición valiosa al algoritmo base.

3) Manejo de Trayectorias: Además, se incorporó un manejo eficiente de trayectorias mediante la inclusión de dos contadores que permiten rastrear detecciones falsas y faltantes. Además, se estableció un límite máximo para estas detecciones, de manera que se puedan eliminar trayectorias y detecciones falsas que superen estos umbrales.

Para implementar este manejo de trayectorias, se realizaron modificaciones en el archivo *tracker.py*, que afectan a las clases 'Tracker' y 'Track'. En la clase 'Track', se añadieron los atributos *misses* y *false_detections* para registrar las detecciones fallidas y las falsas, respectivamente, de cada seguimiento. Por otro lado, en la clase 'Tracker', se introdujeron los atributos *max_misses* y *max_false_detections*, que representan el máximo número permitido de detecciones fallidas y falsas consecutivas para un seguimiento. Se experimentó con valores máximos de 2, 3, 5, 8 y 10 para cada atributo, buscando maximizar los resultados obtenidos.

Además, se modificó la función *data_association* para implementar la eliminación de las trayectorias con detecciones fallidas y falsas excesivas. Se agregó un segmento de código donde se aumenta el conteo de detecciones fallidas si un seguimiento no se asocia con una detección, y viceversa, si las detecciones no se asocian con un seguimiento (lo que corresponde a un falso positivo).

Finalmente, todos los seguimientos que superan los límites máximos de detecciones fallidas y falsas son eliminados del proceso de seguimiento, garantizando así una gestión más precisa y eficiente de las trayectorias detectadas.

C. Ejecutando el código

La mejora y resolución del código base proporcionado se estructura en tres archivos principales. El primero, *APP-SIV_mot_p2.ipynb*, actúa como el "main" del proyecto, ya que carga y ejecuta todos los componentes relevantes del mismo. Al final de su ejecución, este archivo presenta los resultados obtenidos, consolidando así el flujo de trabajo del proyecto.

Por otro lado, el archivo *object_detector.py* aloja los detectores de objetos mencionados, implementados en los métodos FRCNN_FPN y YOLO. Este archivo es invocado por el archivo principal para llevar a cabo la detección de objetos necesaria para el seguimiento.

Asimismo, el archivo *tracker.py* contiene la clase Tracker, la cual gestiona el seguimiento de objetos. Este archivo es llamado por el archivo principal para llevar a cabo la tarea de seguimiento de objetos.

Finalmente, se incluyen los ficheros *mot_to_yolo.py* y *dataset.yaml* para la completitud de la evaluación del trabajo realizado, pero no son finalmente necesarios para ejecutar el programa, sólo lo serían para entrenar el modelo YOLO, el cual está referenciado en el código de *object_detector.py* como un enlace a Google Drive para descargar sus pesos.

Los archivos restantes son los mismos proporcionados en el código base del proyecto y no son mencionados aquí, ya que no influyen directamente en la estructura principal del proyecto. Tampoco se incluye copia del repositorio de Ultralytics, que contiene el script *train.py*, puesto que además se puede clonar desde GitHub [3].

III. DATOS

En este proyecto, se utilizó el conjunto de datos Multiple Object Tracking Benchmark (MOT16) [6], reconocido por su eficacia en el desarrollo y evaluación de algoritmos de seguimiento de objetos en escenarios complejos. MOT16 consta de 7 secuencias de video destinadas al entrenamiento y otras 7 para pruebas. Cada secuencia presenta una variedad de desafíos, incluyendo oclusiones, movimientos de cámara y una densa multitud en movimiento, lo que complica el seguimiento preciso de los objetos de interés.

Estas secuencias abarcan una amplia gama de situaciones, con variaciones en resolución y duración que reflejan la diversidad de condiciones encontradas en entornos del mundo real. La presencia de múltiples personas en cada secuencia proporciona una variedad de escenarios para evaluar la capacidad de los algoritmos de seguimiento para mantener la consistencia en la identificación y el seguimiento de objetos en movimiento.

El conjunto de datos MOT16 incluye una delimitación precisa de las personas en cada fotograma, así como la asignación de identificadores únicos a lo largo del tiempo para facilitar el seguimiento continuo de los objetos. Esto permite una evaluación rigurosa de la precisión y robustez de los algoritmos de seguimiento en condiciones diversas.

Específicamente para este proyecto, se utilizó únicamente el conjunto de datos de entrenamiento, seleccionando 4 secuencias (MOT16-02, MOT16-04, MOT16-05 y MOT16-09) para entrenamiento y 3 secuencias (MOT16-10, MOT16-11, MOT16-13) para pruebas. Al emplear el conjunto de datos MOT16 en este proyecto, fue posible evaluar y comparar el rendimiento de diferentes enfoques de seguimiento de objetos en escenarios del mundo real.

IV. RESULTADOS Y ANÁLISIS

En esta sección, analizamos los diferentes resultados obtenidos con los métodos presentados en la sección II de

este documento. En la tabla I, se muestran los resultados iniciales del modelo base proporcionado sobre la base de datos de prueba. Es destacable que la métrica MOTA, promediada en las tres secuencias, alcanza un 21.6%.

TABLE I: Resultados Modelo Base.

Secuencia	MOTA
MOT16-13	32.6%
MOT16-10	6.4%
MOT16-11	28.7%
OVERALL	**21.6%**

A. Detección de Objetos

En este apartado se probaron los siguientes métodos con el objetivo de mejorar las detecciones:

1) Detecciones de Alta Calidad: Para mejorar los resultados obtenidos con el modelo base, se probaron diferentes umbrales para obtener detecciones de alta calidad, combinándolos con el tracker del modelo base. El objetivo de estas pruebas fue maximizar los resultados en la tarea de detección. En la Tabla II se presenta un resumen de los resultados de la métrica MOTA promedio para los distintos umbrales utilizados.

TABLE II: Resultados para cada umbral de detección.

Umbral	MOTA
0.5	25.7%
0.6	**27.5**%
0.7	25.7%
0.8	25.3%
0.9	23.0%

Se puede observar que el mejor resultado se obtuvo con un umbral de 0.6. Por lo tanto, este umbral se utilizará para los experimentos subsecuentes. Es importante destacar el impacto de las detecciones de alta calidad en la métrica MOTA, aumentando en un 6%.

2) YOLOv5: Por otro lado, se probó el modelo YOLOv5 con el objetivo de mejorar el detector base Faster R-CNN. Este experimento se realizó en conjunto con el tracker del modelo base.

En la Tabla III se presentan los resultados obtenidos con YOLOv5. Sin embargo, los resultados fueron inferiores a los obtenidos con Faster R-CNN, por lo que se decidió continuar trabajando con el modelo original.

TABLE III: Resultados con YOLOv5 como detector.

Secuencia	MOTA
MOT16-13	32.8%
MOT16-10	13.6%
MOT16-11	23.8%
OVERALL	**23.0%**

B. Seguimiento de objetos

Partiendo del modelo de detección establecido en el apartado anterior, se llevaron a cabo una serie de modificaciones con el objetivo de optimizar el método de seguimiento. Estas mejoras incluyeron la implementación de algoritmos avanzados de asociación de datos y el manejo eficiente de trayectorias.

1) Asociación de Datos: En esta sección, se empleó el método húngaro para abordar el desafío de la asociación de datos, combinado con la métrica de Intersección sobre Unión (IoU) para generar la matriz de distancias. Los resultados obtenidos se presentan detalladamente en la tabla IV. Es importante destacar que se observa una mejora sustancial en la métrica MOTA, alcanzando un impresionante 62.8%.

Este incremento representa un aumento de más del 35% con respecto a los resultados anteriores. Estos hallazgos evidencian el impacto significativo que tiene resolver el problema de asociación como una optimización, lo que subraya la efectividad y la importancia de la implementación del método húngaro en el contexto del seguimiento de objetos.

TABLE IV: Resultados con Algoritmo Húngaro.

Secuencia	MOTA
MOT16-13	55.6%
MOT16-10	65.8%
MOT16-11	67.5%
OVERALL	**62.8%**

C. Manejo de Trayectorias

Por otro lado, se ha enriquecido el método húngaro con un manejo más sofisticado de trayectorias, lo cual implica la incorporación de contadores dedicados a registrar tanto las detecciones fallidas como las falsas positivas. Esta adición estratégica permite una evaluación más precisa del rendimiento del sistema de seguimiento. Además, se han introducido umbrales máximos para controlar el número de detecciones fallidas y falsos positivos permitidos en una trayectoria. De esta manera, aquellos seguimientos que excedan estos umbrales son identificados y removidos de manera automática, contribuyendo así a la depuración y optimización del proceso de seguimiento.

En el proceso de experimentación, se exploraron varios valores máximos para estos umbrales con el objetivo de maximizar la métrica MOTA. Los resultados de estas pruebas se presentan detalladamente en la tabla V, donde se puede apreciar cómo la elección de diferentes umbrales máximos impacta en el rendimiento del sistema de seguimiento. Este análisis permite identificar configuraciones óptimas que garantizan un equilibrio adecuado entre la detección precisa de objetos y la robustez del seguimiento a lo largo del tiempo.

TABLE V: Resultados Manejo de Trayectorias con distintos umbrales.

Umbral Fallos	Umbral Falsas Detecciones	MOTA
2	2	**75.8%**
3	3	75.4%
5	5	74.5%
8	8	73.3%
10	10	72.5%

Se puede apreciar que los mejores resultados fueron obtenidos utilizando un umbral de 2 tanto para fallos como para falsas detecciones. Es importante mencionar que,

aunque estos resultados son prometedores, podrían optimizarse aún más mediante la realización de un grid search exhaustivo. No obstante, con los parámetros actuales, hemos logrado alcanzar una MOTA del 75.8%, lo cual representa una mejora significativa del 13% en comparación con los resultados anteriores. Este incremento notable subraya la eficacia de los ajustes realizados y destaca el potencial de optimización adicional a través de técnicas de búsqueda de hiperparámetros más sofisticadas.

En la tabla VI se presentan los resultados del modelo final tomando en cuenta todas las métricas obtenidas sobre los datos de prueba. Con las tres secuencias de prueba se obtuvo un MOTA promedio de 75.8%.

Además, en la tabla VII se presentan los resultados obtenidos para todos los conjuntos de datos utilizados en el entrenamiento, que incluyen siete secuencias de video. En esta tabla se puede observar que se alcanza un MOTA promedio de 68.1% en todas las secuencias de video, lo cual demuestra el rendimiento consistente del modelo mejorado en diversos escenarios.

D. Comparación Modelo Mejorado vs Modelo Base.

A continuación, se presentarán comparaciones visuales en dos secuencias del conjunto de datos de prueba, destacando las detecciones obtenidas tanto con el modelo mejorado como con el modelo base. Estas comparaciones permitirán visualizar de manera clara las mejoras alcanzadas en la precisión y consistencia del seguimiento de objetos gracias a las optimizaciones implementadas en el modelo mejorado.

1) Secuencia MOT16-10: Se presenta el cuadro 30 de la secuencia de video MOT16-10 en la cual se pueden apreciar claras diferencias en las detecciones del modelo base y el modelo mejorado con los metodos mencionados en la seccion II. En la figura 1 se puede apreciar que el modelo base no es capaz de detectar a todas las personas en el cuadro, a simple vista se puede notar que omite 6 detecciones aproximadametente, mientras que el modelo mejorado de la figura 2 es capaz de detectar todas las omisiones del modelo base.

Sin embargo, el modelo mejorado no está exento de fallos. En el cuadro 30 de la figura 2, la detección número 4 es reasignada y considerada como una nueva detección debido a una oclusión, mientras que en los cuadros anteriores estaba etiquetada como 3. Estos errores causados por oclusiones podrían mejorarse aumentando el umbral de detecciones fallidas, permitiendo que las detecciones ocluidas permanezcan más tiempo en la matriz de asignación. Esta modificación ayudaría a mantener la coherencia en la identificación de objetos a lo largo de secuencias con oclusiones temporales.

2) Secuencia MOT16-13: A continuación, se presenta el cuadro 30 de la secuencia MOT16-13, donde se puede apreciar visualmente el rendimiento del modelo base y del modelo mejorado. En la figura 4, se observa que el modelo mejorado es capaz de detectar a las personas con etiquetas 21, 22 y 23, a pesar de que estas aparecen de manera difusa en la imagen. Por otro lado, la figura 3 muestra que el modelo base no logra detectar a estas personas. Esta comparación evidencia la superioridad del modelo mejorado en la identificación de objetos en condiciones difíciles.

Fig. 1: Detección Modelo Base Frame 30

Fig. 2: Detección Modelo Mejorado Frame 30

V. CONCLUSIONES

En conclusión, se ha logrado mejorar significativamente el rendimiento del modelo base proporcionado. Este éxito ha sido posible gracias a la adición de tres componentes fundamentales: detecciones de alta calidad, el algoritmo húngaro para resolver el problema de asociación, y el manejo óptimo de trayectorias. Estas tres mejoras han permitido alcanzar un promedio de MOTA del 75.8% en los datos de test y un 68.1% en todo el conjunto de datos, un resultado prometedor

TABLE VI: Resultados Modelo Final sobre Datos de Prueba.

Secuencia	IDF1	IDP	IDR	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP
MOT16-13	58.8%	62.7%	55.4%	83.7%	94.8%	110	79	26	5	537	1894	236	174	77.1%	0.135
MOT16-10	51.0%	56.6%	46.5%	78.8%	95.9%	57	36	20	1	429	2718	218	228	73.8%	0.147
MOT16-11	60.9%	68.3%	55.0%	79.1%	98.2%	75	40	27	8	135	1970	60	59	77.1%	0.081
OVERALL	56.5%	61.9%	51.9%	80.6%	96.1%	242	155	73	14	1101	6582	514	461	**75.8%**	0.124

TABLE VII: Resultados Modelo Final sobre todos los Datos.

Secuencia	IDF1	IDP	IDR	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP
MOT16-02	39.1%	58.0%	29.5%	50.5%	99.2%	62	9	39	14	75	9198	174	154	49.2%	0.088
MOT16-04	60.1%	70.7%	52.2%	73.4%	99.5%	83	39	29	15	178	12648	130	177	72.8%	0.099
MOT16-05	49.3%	59.4%	42.2%	67.9%	95.5%	133	51	66	16	222	2218	139	119	62.7%	0.140
MOT16-09	48.2%	60.0%	40.2%	66.0%	98.6%	26	11	14	1	49	1808	59	66	64.0%	0.082
MOT16-10	51.0%	56.6%	46.5%	78.8%	95.9%	57	36	20	1	429	2718	218	228	73.8%	0.147
MOT16-11	60.9%	68.3%	55.0%	79.1%	98.2%	75	40	27	8	135	1970	60	59	77.1%	0.081
MOT16-13	58.8%	62.7%	55.4%	83.7%	94.8%	110	79	26	5	537	1894	236	174	77.1%	0.135
OVERALL	52.5%	62.2%	45.8%	71.3%	97.9%	78	38	32	9	232	4636	145	140	**68.1%**	0.238

Fig. 3: Detección Modelo Base Frame 30

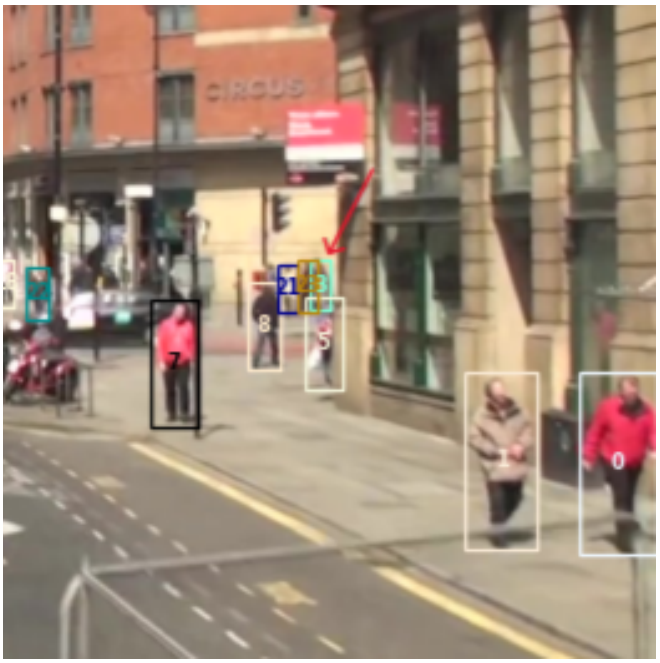

Fig. 4: Detección Modelo Mejorado Frame 30

y representativo de la eficacia del modelo mejorado en la detección y seguimiento de objetos.

Es importante destacar que estos métodos podrían ser aún más optimizados mediante técnicas exhaustivas, como el grid search, que permiten encontrar los parámetros óptimos para cada componente. Además, este ejercicio ha demostrado que la tarea de seguimiento de múltiples objetos puede ser considerablemente optimizada utilizando los métodos presentados en este trabajo.

En resumen, el uso de detecciones de alta calidad garantiza que el modelo tenga una base sólida de datos precisos sobre los cuales operar. La implementación del algoritmo húngaro optimiza la asociación de datos, minimizando los errores en la asignación de detecciones a trayectorias. Por último, el manejo adecuado de trayectorias, mediante el uso de contadores y umbrales para detecciones fallidas y falsas, asegura que el sistema mantenga su precisión y robustez, incluso en escenarios complejos.

VI. LOG DE TIEMPOS

A continuación, se detalla la cantidad de tiempo dedicado a cada aspecto del proyecto. Los tiempos incluyen la suma del esfuerzo de ambos participantes:

1) Estudio e investigación: 6 horas.
2) Implementación del código: 10 horas, incluyendo el diseño de la estructura del código, su implementación y las pruebas correspondientes.
3) Elaboración del informe: 6 horas, abarcando la redacción y ensamblaje del documento.

REFERENCES

[1] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, Y. Kwon, K. Michael, J. Fang, Z. Yifu, C. Wong, D. Montes *et al.*, "ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation," *Zenodo*, 2022.

[2] Ultralytics, "Train custom data," https://docs.ultralytics.com/yolov5/tutorials/train_custom_data, accedido: 2024-05-19.

[3] G. Jochner, "Yolov5 v7. github," https://github.com/ultralytics/yolov5/tree/v7.0, 2022, accedido: 2024-05-20.

[4] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[5] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[6] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," *arXiv preprint arXiv:1603.00831*, 2016.