Trabajo de Fin de Grado

Elaboración de mapas de calor basados en seguimiento de personas

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Álvaro Javier Afonso López

TUTORIZADO POR:
Dr. José Javier Lorenzo Navarro
Dr. Modesto Fernando Castrillón Santana

Fecha: 15 de enero de 2024

Agradecimientos

A mi familia, por aguantar mis momentos de estrés donde ni yo lo hacía y apoyarme siempre. Especial mención a mi madre, ella me lo ha dado todo.

A mi pareja, que me ha dado tanto cariño y muchísima felicidad.

A mis amigos, por aportarme las distracciones y sonrisas que uno siempre necesita.

Y por último, a mi, por haberme demostrado tantas veces cómo de equivocado estaba cada vez que pensaba que no podía.

Resumen

Este proyecto se enfoca en el ámbito de Visión por Computador, lo cual es muy útil a la hora de extraer información a partir de vídeos o imágenes. En la actualidad, con la explosión de surgimientos constantes de nuevas inteligencias artificiales, su uso ha tomado un rumbo muy interesante que aporta mucho valor.

El desarrollo de este trabajo está enfocado en la generación de mapas de calor a partir de vídeos de personas en tránsito, con la ayuda del método ByteTrack, disponible en Github. Este nos brinda la oportunidad de obtener información de los vídeos mencionados, tales como las coordenadas y el tamaño en la imagen, información que se puede usar de distintas maneras según el objetivo buscado, en nuestro caso, hacer que los datos sean más visibles para los usuarios.

Para dar una mayor flexibilidad en el resultado final y no forzar al usuario a tener un vídeo en una ubicación perfecta, se ha implementado una sencilla forma de escoger la perspectiva que se desea estudiar simplemente marcando sobre la imagen, lo que se conoce como transformación homográfica.

Abstract

This project focuses on the field of Computer Vision, which is very useful when extracting information from vídeos or images. Nowadays, with the explosion of constant emergence of new artificial intelligences, its use has taken a very interesting direction that provides a lot of value.

The development of this project is focused on the generation of heat maps from vídeos of people in transit, with the help of the ByteTrack tool, available on Github. This gives us the opportunity to obtain information from the mentioned vídeos, such as the coordinates and size in the image, information that can be used in different ways depending on the objective sought, in our case, making the data somewhat more visible to the users.

To give greater flexibility in the final result and not force the user to have a vídeo in a perfect location, a simple way has been implemented to choose the perspective you want to study simply by clicking on the image, which is known as transformation homographic.

Capítulo 1

Introducción

1.1. Motivación

Los mapas de calor son una herramienta gráfica que utiliza una jerarquía de colores para identificar los puntos de mayor y menor interés, resaltando la densidad y distribución en áreas específicas. Los colores cálidos como el rojo, naranja y amarillo representan los puntos de mayor interés, mientras que los colores fríos como el verde, azul y turquesa representan los puntos de menor interés. En la Ilustración 1.1 se puede ver un ejemplo genérico del resultado generado de un mapa de calor.

Estos gráficos son aplicables en una variedad de sectores, y uno de los campos de aplicación más significativos lo podemos encontrar en el sector comercial. Permiten comprender o analizar de una forma más eficiente los movimientos de la población en relación con la densidad de personas en distintos contextos geográficos, lo que aporta información muy valiosa. La capacidad de detectar los "puntos calientes", es decir, zonas que atraen la atención del usuario o del cliente, identificados en estos mapas pueden ser fundamentales para las empresas para establecer diferentes estrategias de ubicación de productos o servicios en función de la afluencia de usuarios, maximizando así su impacto en base a la afluencia de usuarios.

La investigación de esta tecnología sigue en constante avance, tratando de buscar nuevas aplicaciones y mejorar la precisión de los datos que estos mapas reflejan. El avance y aplicaciones no solo se aplica al ámbito comercial, sino en cosas tan fundamentales como la gestión de eventos, ciencia o planificación urbana.

Para conseguir estos resultados, la Visión por Computación ha resultado ser una herramienta completamente revolucionaria y eficaz en una amplia gama de aplicaciones, abarcando desde vigilancia hasta análisis de datos, así como optimizar rutas y espacios. En particular, ByteTrack es el pilar fundamental en el que sustenta este proyecto, siendo uno de los mejores en el sector.

Nos encontraremos desafíos tales como la precisión en la detección, escalabilidad y rendimiento al ser un proceso muy costoso de realizar. Tras una evaluación exhaustiva de esta

aplicación esperamos conseguir resultados muy favorables que permitan el uso de nuestra herramienta a personas que estén realizando algún tipo de estudio relacionado con el comportamiento del movimiento humano de una forma sencilla y rápida gracias a nuestra interfaz de usuario diseñada para dicho propósito.

Ilustración 1.1: Imagen de OZmap sobre puntos de venta (Fuente: [OZmap])

1.2. Objetivos

El objetivo del proyecto es ofrecer una aplicación con un aspecto gráfico sencillo y lo más intuitivo posible. Esta aplicación permitirá al usuario seleccionar un vídeo que desea analizar, indicar el tipo de análisis a aplicar y comenzar a ver resultados en tiempo real de forma inmediata.

Para conseguir el propósito general, se han propuesto unos objetivos específicos:

- ✔ Diseñar interfaz gráfica de usuario sencilla que simplifique las interacciones del usuario con la aplicación.
- ✔ Integrar ByteTrack para el seguimiento de personas en el vídeo previamente seleccionado, usando la información que este aporta como coordenadas y tamaño de las personas en la imagen.
- ✔ Mostrar el mapa de calor generado mediante la elección dos tipos de análisis, por tiempo o por movimiento, pudiendo ver tanto el vídeo original como el resultado al mismo tiempo.
- ✔ Permitir al usuario ver el progreso del proceso en tiempo real, sin necesidad de que este finalice, pudiendo interrumpirlo si así lo deseara.
- ✔ Permitir modificar la perspectiva en tiempo real sin interrumpir el proceso.
- ✔ Desarrollar la aplicación con el uso de las librerías gráficas que ofrece Python, lo que nos permite una implementación lo más eficiente posible en el manejo de datos.

1.3. Aportaciones

Tradicionalmente, cuando se quería generar mapas de calor, eran necesarios dispositivos de geolocalización específicos, lo que se traducía en una limitación y costes significativos. Ahora que contamos con la Visión por Computador, hemos podido dejar de depender de estos dispositivos, lo que nos permite una gran reducción en costes y los recursos derivados de este.

Este enfoque de Visión por Computador simplifica mucho el proceso de estudio, haciendo que no sean necesarios conocimientos técnicos específicos de cada dispositivo. Esto permite que el número de usuarios que pueden usar la herramienta aumente considerablemente, incluyendo a personas con conocimientos y recursos promedio sobre el uso de ordenadores y aplicaciones.

Con todo ello, estas aportaciones han conseguido poner al alcance de cualquiera la posibilidad de usar esta herramienta eficiente y económica como método de estudio del comportamiento humano. Así, se rompen las barreras que se encontraban en la realización de estudios previos.

1.4. Competencias específicas

CP03: Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.

CP04: Capacidad para conocer los fundamentos, paradigmas y técnicas propidas de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

CP05: Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes o entornos inteligentes.

CP06: Capacidad para desarrollar y evaluar sistemas interactivos y de presentación de información compleja y su aplicación a la resolución de problemas de diseño de interacción persona computadora.

1.5. Normativa y legislación

En el contexto en el que se desarrolla esta aplicación de seguimiento de personas sería imprudente no contar con la legislación y normativas aplicables. En este caso, la norma que se debe cumplir es la ley de protección de datos, como el Reglamento General de Protección de Datos (RGPD, fuente: [pro]) de la Unión Europea y la legislación española en la materia.

Este proyecto no guarda las imágenes que se le proporcionan ni tampoco ningún tipo de datos personales de las personas que aparecen puesto que no son necesarios. La herramienta únicamente analiza la secuencia de imágenes en busca de personas, pero no guarda datos más allá de lo estrictamente visual con relación a las posiciones con respecto a la imagen. Por consiguiente, al no almacenar, manipular ni usar ningún tipo de dato personal, cumple a la perfección en lo que respecta a la normativa de protección de datos, aunque se ha dejado una sección del código preparado para futuras implementaciones acerca de guardar únicamente el mapa de calor generado por esta herramienta.

La fuente de información que requiere la aplicación, es decir, los videos, corresponde a un ámbito externo al control de la aplicación, por tanto, para analizar esta información debemos asumir que el usuario que tiene en su poder estos datos cuenta con la autorización y permisos para el uso de estos de forma legal, y de no ser así deberá asumir la responsabilidad de las consecuencias legales.

Capítulo 2

Estado del arte

En este capítulo se explicará a nivel teórico los fundamentos en los que se sustenta este proyecto, tanto las técnicas como los conceptos.

2.1. Seguimiento de personas

Esta es una técnica fundamental en la Visión por Computador cuyo propósito es identificar, detectar y rastrear ciertos elementos en diferentes secuencias de imágenes o vídeos. Los últimos avances tecnológicos han conducido al desarrollo de modelos de aprendizaje profundo, tales así como redes neuronales convolucionales (CNN) o redes neuronales recurrentes (RNN), lo que ha sido uno de los principales responsables en la mejora de la precisión en el seguimiento de personas. Para ello, se han usado técnicas avanzadas de procesamiento de imágenes y aprendizaje automático que nos permiten reconocer características faciales, corporales o incluso la ropa de forma precisa y eficiente, facilitando el reconocimiento y seguimiento de los individuos en distintos contextos, tanto en interiores como en exteriores. Dichos modelos han aprendido de forma automática gracias a características almacenadas de forma masiva para realizar tareas de seguimiento más precisas y robustas.

A pesar de lo efectivos que pueden llegar a ser estos modelos, no dejan de encontrarse con ciertas barreras en su correcto funcionamiento, como pueden ser las condiciones de iluminación, la visibilidad, cambios en la apariencia de las personas o movimientos demasiado rápidos. Estas barreras son la motivación para continuar tratando de desarrollar modelos algorítmicos más robustos y técnicas más eficientes.

Hay una diferencia importante entre los conceptos de detección y seguimiento, si bien ambos requieren una detección inicial, el segundo es capaz de mantener ubicado el mismo elemento entre distintos fotogramas dentro de un mismo video, mientras que el primero haría su trabajo independientemente de los demás fotogramas, creando nuevas instancias de lo que sería el mismo objeto previamente catalogado en frames anteriores, entendiendo que es uno nuevo cuando no es así.

En la Ilustración 2.1 podemos ver un ejemplo de seguimiento de personas donde cada detección tiene un identificador único que se mantiene durante toda su aparición en el vídeo.

Ilustración 2.1: Ejemplo de seguimiento de personas generado por nosotros con ByteTrack

Los elementos que entran en juego para el seguimiento de personas son: entrada, detección de personas, etiquetado y seguimiento. El paso inicial es aportar información como puede ser una secuencia de imágenes o video, la llamada "entrada", y preprocesar cada frame antes de comenzar el análisis, lo cual es fundamental para que la herramienta cuente con datos consistentes para trabajar. El segundo paso sería la detección, donde entra en juego algún algoritmo que detecta y clasifica el objeto deseado dentro de una caja delimitadora que contiene los datos relacionados con la misma, como puede ser YOLO (You Only Look Once) [10] o cualquier otro modelo de su elección. A continuación, se prosigue con el etiquetado de los elementos previamente clasificados mediante una identificación única para cada uno de ellos. Y, por último, sería mantener ubicado el elemento clasificado con su respectiva etiqueta a lo largo de los frames que componen el video mientras se guarda la información recabada acerca de la trayectoria que ha seguido.

2.2. ByteTrack [2]

Es un método avanzado de seguimiento de objetos en vídeos, con especial diseño para la detección de personas. Las principales características son:

Precisión: Usa avanzados algoritmos de visión por computación para hacer un seguimiento los más preciso y eficiente posible de las personas que se pueden encontrar en una secuencia de imágenes interiores o exteriores.

Coordenadas y tamaño: Tras realizar el análisis, ByteTrack nos proporciona datos muy importantes como las coordenadas espaciales en la imagen, así como su tamaño para poder ubicar a la persona en su totalidad.

Este método usa la medida loU (Intersection over Union), lo que proporciona el porcentaje de acierto del área de predicción sobre la "bounding-box" que realmente se esperaba detectar. La mayoría de los métodos usan únicamente un nivel de umbral para asociar cajas de detección, como puede ser por ejemplo el 80 %. Sin embargo, ByteTrack tiene en cuenta todo tipo de detecciones, separándolo todo en predicciones de alta y baja confianza, que viene definida como "L" en el modelo, y la salida de que nos proporciona ByteTrack es "T", las trayectorias encontradas.

Para poner en contexto, explicamos los siguientes conceptos que se mencionarán más adelante:

YOLO: Modelo de detección de objetos a partir de vídeos o imágenes caracterizado por su precisión y eficiencia. A diferencia de otros modelos que dividen la imagen en diferentes regiones para hacer predicciones en cada una de ellas, YOLO aplica la detección sobre la imagen en su totalidad de forma simultánea, lo que permite una detección mucho más precisa y rápida.

Filtro de Kalman: Método matemático usado para medir el estado de un sistema dinámico en base a una serie de mediciones ruidosas e imprecisas. Con ello se consigue predecir el futuro estado del sistema usando tanto la información pasada como la actual. En este contexto del seguimiento de objetos, usamos este filtro para predecir la posible ubicación de un objeto en el siguiente frame considerando la trayectoria seguida hasta el momento.

En cada frame que compone el video se aplica el modelo de detección Yolo para obtener las predicciones asociadas a sus valores de confianza, las cuales se separan en dos grupos: "D_high" para las que la confianza es alta y "D_low" para las que tienen confianza baja. Tras esto, se aplica un filtro Kalman para encontrar las futuras ubicaciones de cada trayectoria.

La primera etapa de asociaciones se hace entre las cajas de alta puntuación de confianza y todas las trayectorias existentes, incluyendo las que puedan haberse perdido su detección, las cuales quedan en el grupo de "T_lost". Las detecciones que no se les asigna asociación con ninguna trayectoria se quedan en el grupo "D_remain" y "T_remain", respectivamente.

En la segunda etapa, se busca el emparejamiento de las detecciones de baja puntuación de confianza con las trayectorias en "T_remain" de la primera etapa y las que han quedado

sin emparejamiento se mantendrán en un nuevo grupo “T_re_remain”, mientras que las detecciones no emparejadas que además son de puntaje de confianza bajo se descartan finalmente. Las trayectorias que se han perdido en esta segunda etapa serán almacenadas en “T_lost”, y si esta se mantiene en dicho grupo por el número definido de 30 frames, se eliminará de “T”.

Por último, ByteTrack inicializa trayectorias nuevas a partir de las detecciones que cuenten con una alta confianza de puntuación que quedaron sin emparejar en “D_remain” de la primera etapa.

En la Ilustración 2.2 podemos apreciar la clara superioridad de esta herramienta en comparación con otras disponibles hasta principios de 2022, hoy en día superado por Bot-SORT [1] y luego por SMILETrack [7]:

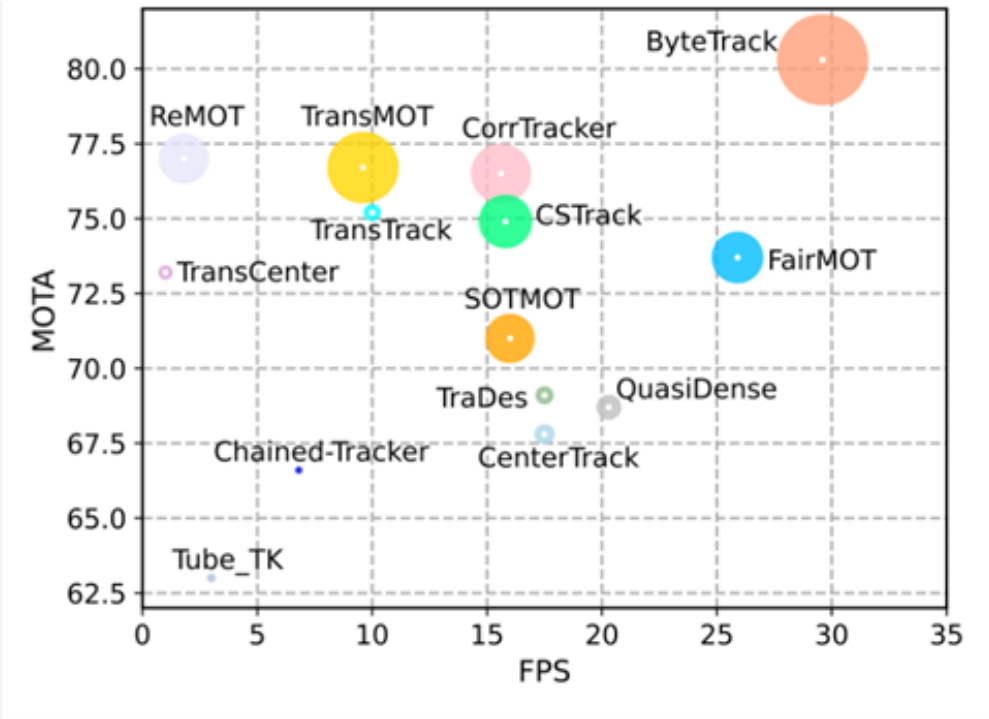

Ilustración 2.2: Comparativa de herramientas de tracking (Fuente: [com])

2.3. Transformación homográfica

Las coordenadas que nos aporta ByteTrack se encuentran dentro de la imagen, y para representar estas mismas coordenadas en el mundo real debemos aplicar la transformación homográfica, también conocida como proyección proyectiva. Esta es una técnica geométrica que transforma unas coordenadas determinadas en plano bidimensional a otras en un plano tridimensional. En este proyecto usamos esta técnica para permitir al usuario cambiar la perspectiva y corregir distorsiones provocadas por algún ángulo de visión no deseado del vídeo original, pudiendo hacer un mapa de calor desde un punto de vista diferente.

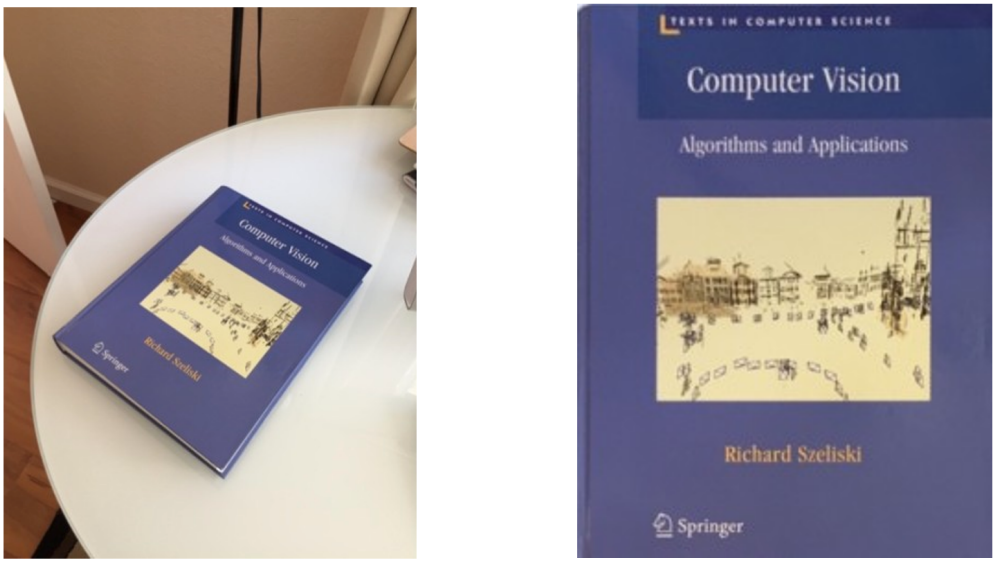

Ilustración 2.3: Ejemplo de transformación homográfica (Fuente: [per])

En la Ilustración 2.3 podemos apreciar que gracias al cambio de perspectiva homográfica, hemos sido capaces de ver la portada del libro con mucha más facilidad que en la imagen original.

Existen múltiples tipos de transformaciones geométricas, como la traslación o la rotación, pero la transformación homográfica es mucho más genérica y flexible, lo que permite cambiar la perspectiva, distorsiones y proyecciones más complejas.

Esta técnica consiste en mapear ciertos puntos de una imagen bidimensional para buscar el equivalente en un espacio tridimensional, tratando de trasponerlo a coordenadas del mundo real, lo que nos resulta sumamente útil cuando queremos ubicar algún objeto o persona contando únicamente con una imagen 2D.

Para conseguir el resultado mencionado, debemos aplicar el cálculo de una matriz de transformación, o también conocida como matriz homográfica. Esta matriz representa la proyección resultante entre dos planos, en este caso el plano bidimensional y el tridimensional y la usamos como medio para mapear puntos de un plano al otro, según el resultado que deseamos. Aplicado al caso de seguimiento de personas, resulta útil aplicar esta técnica dado que, al contar con imágenes únicamente bidimensionales, tras mapear las coordenadas según

las perspectivas que queremos modificar, nos permite obtener información más precisa y en un espacio tridimensional, lo que sería el mundo real.

La matriz mencionada consiste en una 3x3, la cual define como están relacionados los puntos de un plano con sus equivalentes en otro. Para cada punto específico en el primer plano (x,y) se calcula su equivalente (x',y') en el segundo plano gracias a una multiplicación matricial que observamos en Ilustración 2.4.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Ilustración 2.4: Cálculo de transformación homográfica

A la izquierda del igual, (x',y',w') representan las coordenadas transformadas del punto en el segundo plano, h representan los elementos que forman la matriz homográfica que relaciona ambos puntos y (x,y,1) representa el punto original del primer plano.

Hemos visto cómo se hace el cálculo del cambio homográfico, pero veamos cómo se calcula la matriz de transformación. Si bien la fórmula esta planteada para obtener un punto equivalente en otro plano, un primer paso es indicar unas condiciones iniciales para calcular la matriz. Para ello necesitaremos un mínimo de cuatro puntos en cada uno de los planos que deben ser correspondientes y no colineales, es decir, no pueden estar en una misma línea recta. Cada uno de los puntos se representa como (x,y) en un plano y (x',y') en el otro plano.

El cálculo de la matriz se puede hacer de varias maneras, como puede ser con el cálculo de mínimos cuadrados o por descomposición de matrices, aunque uno de los métodos más usados es el Direct Linear Transform (DLT), que explicamos brevemente a continuación:

1. Para aportar más precisión al resultado y reducir posibles errores, aplicamos una normalización a los puntos correspondientes. Con esto conseguimos que los puntos sean sometidos a una traslación y un escalado para obtener una media igual a cero.

2. Para cada pareja de puntos correspondientes entre los dos planos, (x,y) para el original y (x',y') para el transformado, se construye una fila en la matriz de ecuaciones según la relación que existe entre ambos.

3. Resolvemos el sistema de ecuaciones lineales que componen la matriz. El sistema es construido tomando todos los puntos correspondientes con sus respectivas parejas relacionadas.

4. Tras haber obtenido la matriz homográfica, invertimos la normalización aplicada en el primer paso.

El resultado final que obtenemos es una matriz de tres por tres que componen los coeficientes que describen la transformación de un plano al otro, para cualquier punto obtendremos el correspondiente al aplicarle estos coeficientes. Cabe decir que la matriz homográfica será tan precisa como lo hayan sido tanto los puntos usados como condición inicial, además de los respectivos cálculos de normalización y solución de las ecuaciones.

Capítulo 3

Metodología

3.1. Materiales necesarios para el desarrollo

Para poder usar plenamente esta aplicación serán necesarios dos elementos:

- ✔ Un dispositivo de grabación, como puede ser un móvil o una cámara.
- ✔ Un ordenador.

El ordenador con el que se ha desarrollado y realizado las pruebas consta con un procesador AMD Ryzen 7 5800H con Radeon Graphics, RAM de 16GB y una tarjeta gráfica NVIDIA GeForce RTX 3060. La ejecución de esta aplicación requiere de cierta potencia y al ser un portátil se ha notado considerablemente el rendimiento cuando está conectado a la fuente de alimentación y cuando no, teniendo que buscar el equilibrio entre rendimiento y duración, en ordenadores de escritorio se espera cierta mejoría.

3.2. Entorno de desarrollo

Hemos seleccionado el lenguaje de programación Python por varios motivos: el método ByteTrack del que dependemos está escrito en este lenguaje; en segundo lugar, este ofrece muchas facilidades para el manejo de datos con pocas instrucciones y de forma óptima, dado que tiene muchas librerías orientadas a ello; por último, es multiplataforma.

El entorno de desarrollo integrado (IDE) usado ha sido Visual Studio Code, o como también se le conoce comúnmente VS Code, creado por Microsoft para todo tipo de desarrollo: Web, Windows, Linux y macOS. Entre los principales motivos para haber escogido este editor podríamos mencionar las siguientes:

- ✔ Interfaz altamente intuitiva.
- ✔ Compatibilidad con múltiples lenguajes, que aun no siendo un motivo de peso específico para este proyecto, ha sido determinante porque ya conocía este IDE al haberlo usado

con otros lenguajes anteriormente, lo que me ha permitido ahorrarme ese tiempo de adaptación aunque sea con otro lenguaje.

- ✔ Personalización, ya que nos permite adaptar el editor en gran medida a nuestro gusto e incluso definir atajos de teclado definidos por nosotros mismos.
- ✔ Su popularidad sumada a la particularidad de que es un entorno de código abierto aporta que esté constantemente recibiendo nuevas aportaciones de la comunidad.
- ✔ Tiene soporte para la depuración del código, aportándonos herramientas para hacer pruebas en el propio editor, lo que aporta muchas facilidades para corregir errores cuando se producen en tiempo de ejecución.
- ✔ Control de versiones integrado por defecto, como Git, lo que permite un avance más eficiente del proyecto salvando aquellas versiones con las que se está conforme bajo algún pseudónimo para poder recuperarla si fuera posible.

3.3. Principales bibliotecas

3.3.1. tkinter

Es la librería que ofrece por defecto el lenguaje de Python para la implementación de interfaces de usuario gracias a los numerosos elementos gráficos que ofrece, como pueden ser botones, cajas de texto, entre otras. Nos permite no solo crear elementos a nuestro gusto sino ubicarlos en la disposición que nos resulte más agradable para nosotros.

3.3.2. OpenCV

OpenCV es una librería de código abierto muy usada en aplicaciones que emplean visión por computación, ya que cuenta con una multitud de herramientas para la lectura, procesado y escritura de vídeos o imágenes.

3.3.3. Threading

Esta librería permite ejecutar procesos en segundo plano de manera concurrente con la ejecución principal del código, lo cual es esencial para la mejora del rendimiento de las aplicaciones. Ha sido fundamental en la fase final del proyecto porque ha permitido mejorar considerablemente el rendimiento de la aplicación. Al ser un proceso muy costoso el analizar las imágenes y detectar un gran número de personas, la interfaz se ha visto ralentizada al tener que esperar a que finalice este proceso para poder actualizarse, y con esta herramienta hemos sido capaces de lanzar este proceso en segundo plano y realizar una espera activa, siendo esto mucho más agradable de cara al usuario.

3.4. Metodología de desarrollo

Para este proyecto se ha escogido una metodología de trabajo ágil Scrum, pensado para el desarrollo de proyectos complejos adaptativos en grupos de trabajo pequeños, permitiendo a los equipos entregar productos eficientemente y maximizando el impacto que puedan llegar a tener. El principal objetivo de esta práctica es fomentar la colaboración, la comunicación constante, adaptarse a los cambios, la entrega temprana y la mejora continua a lo largo del proyecto.

A continuación, enumeramos algunos de los aspectos fundamentales:

- ✔ Se prioriza la comunicación y la colaboración por encima de las herramientas o procesos específicos a usar.
- ✔ La documentación detallada no es tan importante como que el producto siempre esté en continuo desarrollo y operativo.
- ✔ Los planes propuestos siempre son flexibles, priorizamos la adaptación al cambio, aceptándolos y adaptándonos a ellos.
- ✔ Se trabaja en iteraciones, o también llamados sprints, que son los periodos de trabajo tras las reuniones donde se ha decidido las futuras implementaciones y que es lo que simboliza la sección "Sprint" en Ilustración 3.1. Además, durante los sprints, se hacen reuniones diarias rápidas para comentar los posibles problemas que se han encontrado y aportar nuevas soluciones o ideas que hayan podido surgir durante el desarrollo diario, simbolizado en la misma ilustración en la sección "24 h".
- ✔ Al emplear lo anterior, conseguimos que haya feedback continuo, mejorando así el producto.

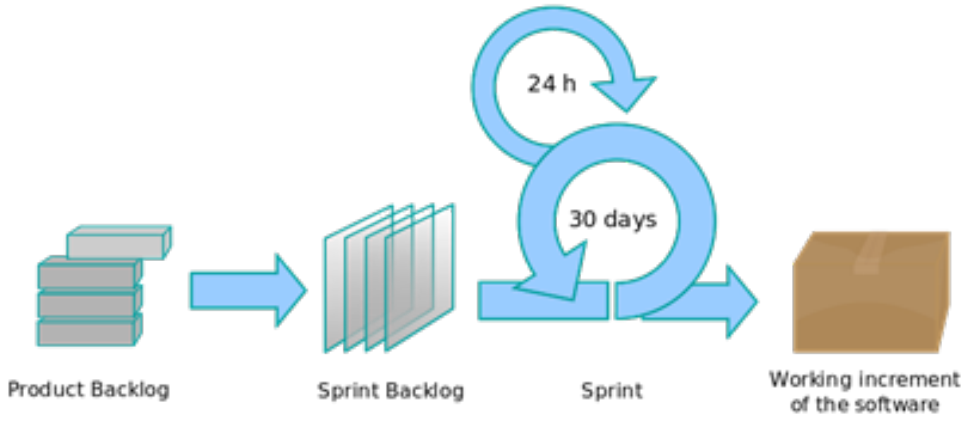

Ilustración 3.1: Metodología ágil Scrum (Fuente: [scr])

Esta forma de metodología ágil se ha vuelto muy popular en el desarrollo de software, aunque también se extendido a otros campos gracias a su enfoque flexible y dinámico para

proyectos complejos con constantes cambios y variables.

Es importante nombrar que hay 3 roles fundamentales en este proyecto, estos son:

Product Owner: El integrante que tiene la idea general y conocimiento del producto a desarrollar. Es el intermediario entre el equipo el cliente y el equipo, quien debe asegurarse que este último comprende los objetivos.

Master: La persona que lidera el equipo y lo guía, pudiendo incluso llegar a ayudar ante cualquier problema que surja durante el desarrollo.

Equipo de desarrollo: Desarrolladores que llevan a cabo la implementación, trabajando en equipo y en sprints, como ya mencionamos.

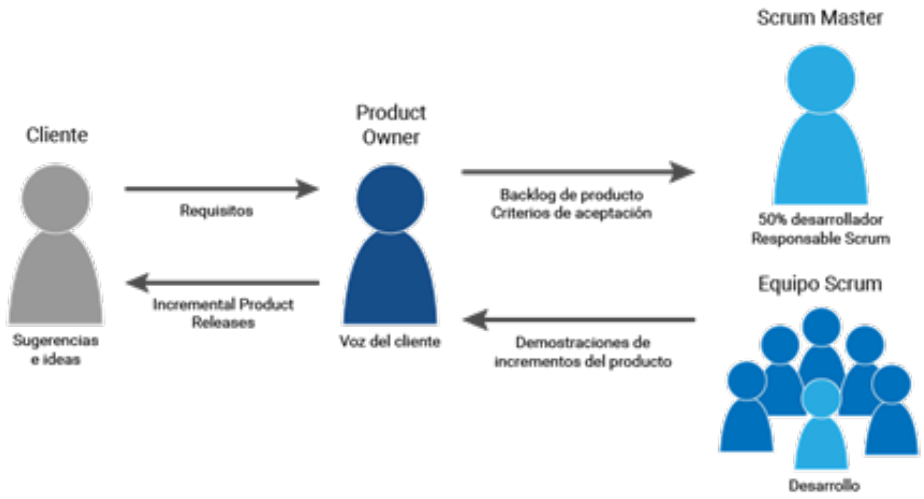

Ilustración 3.2: Roles Scrum (Fuente: [rol])

En este proyecto, el rol de Product Owner lo tomarían tanto el Dr. José Javier Lorenzo Navarro y el Dr. Modesto Fernando Castrillón Santana, ya que son las personas que contaban con la idea, y en cada reunión que hemos tenido su función ha sido darme su punto de vista del avance del proyecto, posibles modificaciones y mejoras, además de darme las indicaciones e información necesaria para el desarrollo del proyecto; el rol de Master lo ha tomado un poco más el Dr. José Javier, quien me ha ayudado con cada problema que he tenido durante el desarrollo; por último, yo soy el denominado equipo de desarrollo, que basándome en las indicaciones y documentos aportados por mis tutores he llevado a cabo las implementaciones de la aplicación. En la Ilustración 3.2 podemos ver de una forma visual las relaciones que existen entre los distintos integrantes de esta metodología de trabajo.

Capítulo 4

Planificación de trabajo

Las etapas que han definido el proceso de realización de este trabajo han sido:

1. **Introducción a ByteTrack:** Antes de comenzar el diseño e implementación del proyecto, se ha hecho un estudio en profundidad acerca de este método, lo que aportó una idea clara de cómo atacar el problema, viendo que clase de datos maneja, cómo los usa y decidir si todas las funcionalidades que ofrece son de utilidad para nuestro propósito.
2. **Introducción a OpenCV:** Estudiar la librería OpenCV y sus capacidades.
3. **Estudio de transformación homográfica:** Comprensión de lo que es y cómo aplicar la transformación homográfica.
4. **Diseño de Interfaz de Usuario:** Tratar de hacer un diseño lo más amigable posible para el usuario promedio.
5. **Implementación de mapas de calor:** Continuas pruebas de generación de mapas de calor con la información que aporta Bytetrack tras hacer el tracking de personas.
6. **Implementación de Interfaz de Usuario:** Desarrollar la interfaz mediante la librería de Python, tkinter.
7. **Cambio de perspectiva homográfica:** Como último paso de desarrollo, se implementó la idea de dar al usuario la posibilidad de seleccionar mediante clics la imagen a mostrar.
8. **Pruebas:** Comprobaciones varias de distintas ejecuciones para corroborar el correcto funcionamiento de la aplicación en distintos modos de análisis y diferentes escenarios.

Para poder poner en contexto de la duración estimada de las tareas, separadas por bloques, podemos verla en la siguiente tabla:

Fase	**Duración (horas)**	**Tarea**
Estudio previo	90	Tarea 1.1: Estudio de ByteTrack
		Tarea 1.2: Estudio de librería OpenCV
		Tarea 1.3: Estudio de transformación homógrafica
Diseño, desarrollo e implementación	140	Tarea 2.1: Diseño de Interfaz de Usuario
		Tarea 2.2: Desarrollo de la creación de mapas de calor con ByteTrack
		Tarea 2.3: Implementación de Interfaz de Usuario
Pruebas	30	Tarea 3.1: Comprobar funcionamiento en distintos escenarios
		Tarea 3.2: Comprobar rendimiento con distintos tipos de análisis.

Ilustración 4.1: Planificación de trabajo

Capítulo 5

Diseño

Como ya se ha mencionado anteriormente, algunos de los objetivos fundamentales en los que nos hemos centrado a la hora de definir la interfaz de usuario (IU) de esta aplicación es la sencillez de esta y hacerla lo más intuitiva posible, permitiendo adaptarla y acercarla a todo tipo de usuario que tenga un conocimiento promedio sobre el manejo de ordenadores convencionales, una primera idea aproximada podemos verla en el diseño de un mockup en la Ilustración 5.1. Para la implementación de la interfaz se ha usado la librería por defecto que ofrece Python llamada *TKinter*.

Ilustración 5.1: Mockup de diseño inicial

Los fundamentos del diseño en los que nos hemos basado son:

Simplicidad: Se ha tratado de hacer la interfaz lo más minimalista dentro de las posibilidades que ofrece la aplicación, lo que permite guiar al usuario de una forma más clara y directa. En la Ilustración 5.1 podemos ver que se han usado los elementos mínimos y necesarios.

Distribución de los elementos: El diseño está compuesto por seis botones, dos canvas donde se muestran las imágenes y una barra de progreso. La distribución de los mismos está dispuesta teniendo en cuenta la tendencia natural de las personas de ir de arriba a abajo y de izquierda a derecha, por ello, en la parte superior están las acciones iniciales, como elegir video o cámara, seguido un nivel por debajo de la siguiente manera: video original, acciones a realizar sobre el video seleccionado y resultado, en este orden de izquierda a derecha. Por último, en la parte inferior se verá el progreso del proceso en una barra de progreso que se rellena conforme avanza el vídeo y el mapa de calor generado.

Este diseño pensado para el propósito que hemos mencionado nos permite aportar al usuario una mayor claridad a la hora de usar la aplicación, y con ello, podemos definir el diagrama de flujo de la Ilustración 5.2.

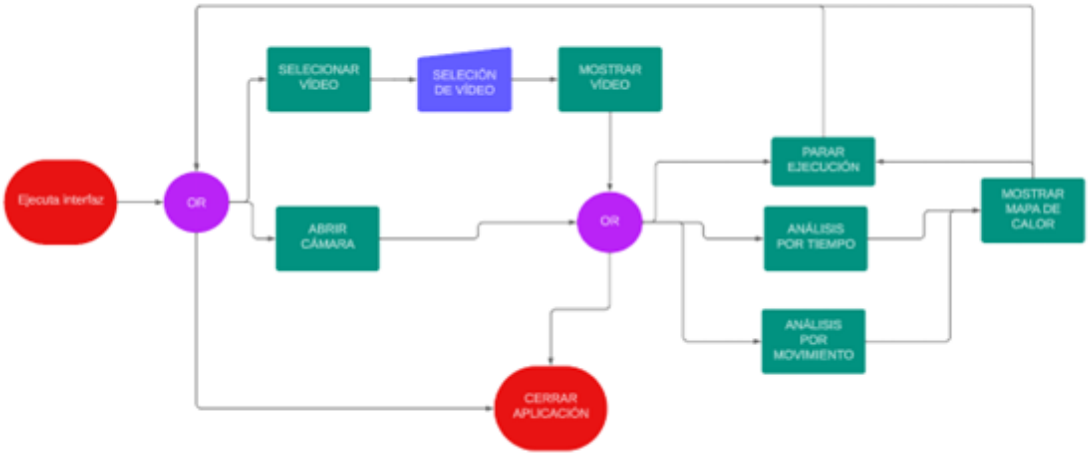

Ilustración 5.2: Diagrama de flujo

Como vemos, el flujo de acciones que puede realizar el usuario está medido, lo primero a destacar es que en cualquier momento del uso de la aplicación esta puede cerrarse, a diferencia de las demás acciones que cuentan con un cierto orden de ejecución. La única restricción que se le impone al usuario es que todo proceso iniciado debe ser interrumpido antes de realizar otro análisis, con esta parada se provoca que se limpie la pantalla de todo tipo de información que pudiera haber sido mostrada en análisis previos.

Capítulo 6

Desarrollo

Este capítulo estará orientado a detallar las diferentes etapas y procesos que ha seguido la implementación de esta aplicación.

6.1. Estructura del proyecto

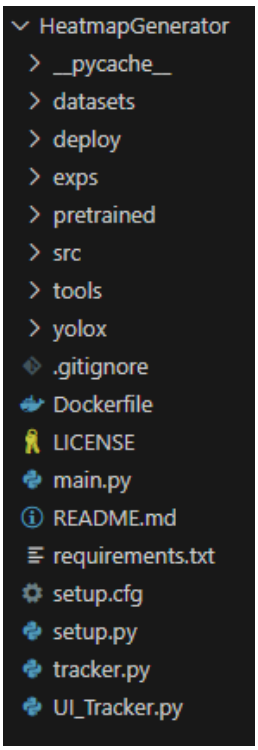

Ilustración 6.1: Estructura del proyecto

En la Ilustración 6.1 se puede ver como se han estructurado los componentes de esta aplicación. Gran parte de esta estructura ha sido creada por los autores del método ByteTrack,

y a partir de ello nosostros hemos realizado mofidicaciones para realizar nuestra aplicación.

En lo que respecta al método hemos querido dejarlo como ya se nos presentaba, y los ficheros específicos de nuestro proyecto los hemos creado en la carpeta raíz, siendo los siguientes:

tracker.py: Fichero que contiene el código que lee, procesa y analiza los vídeos que se le han indicado.

UI_Tracker.py: En este se ha realizado la implementación de la interfaz de usuario, así como la aplicación del cambio de perspectiva homográfica.

main.py: Creado únicamente como punto de arranque de la aplición y de esa manera dejar todos los componentes del proyecto con un único propósito.

6.2. ByteTrack

El estudio previo necesario para entender el funcionamiento de ByteTrack nos permitió que, al mismo tiempo que avanzábamos en compresión del manejo de datos, podíamos ir modificando el código que estábamos estudiando, suprimiendo o modificando algunas funcionalidades que o bien no necesitábamos, o las adaptábamos a la forma en que queríamos que funcionasen.

Inicialmente el código de ByteTrack estaba dispuesto en forma de script para ejecutarlo por comandos, mediante el uso de la biblioteca *argparse* esta tarea es mucho más agradable y sencilla de entender para alguien que aborda el código sin conocimiento previo acerca de la herramienta. De todos los argumentos que podía recibir, a nosotros solo nos era útil uno de ellos, el argumento "path" para indicar la dirección del vídeo a analizar. Dado que los demás parámetros se mantenían siempre estáticos para nuestro propósito, suprimimos la posibilidad de que fueran parámetros y modificamos el código para que siempre tomara los parámetros por defecto en caso de no indicárselos, lo que nos permitió borrar todo el código referido al manejo de parámetros y quedarnos con uno, además de suprimir la opción con la que contaba esta herramienta de guardar los resultados obtenidos, con lo que no se hubiera cumplido con la política de privacidad mencionada anteriormente.

En la Ilustración 6.2 vemos parte del código original de ByteTrack mencionado, donde el tercer parámetro (*path*) es el único que se mantuvo.

```
def make_parser():
    parser = argparse.ArgumentParser("ByteTrack Demo!")
    parser.add_argument(
        "demo", default="image", help="demo type, eg. image, video and webcam"
    )
    parser.add_argument("-expn", "--experiment-name", type=str, default=None)
    parser.add_argument("-n", "--name", type=str, default=None, help="model name")

    parser.add_argument(
        #"--path", default="./datasets/mot/train/MOT17-05-FRCNN/img1", help="path to images or video"
        "--path", default="./videos/palace.mp4", help="path to images or video"
    )
    parser.add_argument("--camid", type=int, default=0, help="webcam demo camera id")
    parser.add_argument(
        "--save_result",
        action="store_true",
        help="whether to save the inference result of image/video",
    )

    # exp file
    parser.add_argument(
        "-f",
        "--exp_file",
        default=None,
        type=str,
        help="pls input your expriment description file",
    )
    parser.add_argument("-c", "--ckpt", default=None, type=str, help="ckpt for eval")
    parser.add_argument(
        "--device",
        default="gpu",
        type=str,
        help="device to run our model, can either be cpu or gpu",
    )
```

Ilustración 6.2: Parámetros iniciales de ByteTtrack

Con lo anterior, hemos sido capaces de comprender las posibilidades que ofrece el método, cómo ejecutar el programa para que comience a analizar algún vídeo que se le indique e incluso modificar la forma de ejecutar para nuestro propósito. Ahora pasemos a entender el funcionamiento interno del método e intentar comprender qué información será la que utilicemos, e incluso qué información nos resultará de poca utilidad para proceder a suprimirla.

Tras un seguimiento lineal de la ejecución del código, desde su punto inicial, proceso tras proceso, llegamos a la parte del código donde ByteTrack devuelve los datos tras analizar el vídeo, los cuales son los que necesitamos para generar los mapas de calor. Esta información de la que hablamos se corresponde con la identificación de las personas detectadas en el análisis de seguimiento con sus respectivas coordenadas en la imagen, además de la altura y el ancho de cada una de ellas en píxeles, con esto fuimos capaces de generar un vídeo de prueba a modo de visualizar las detecciones, este es el que vimos en la Ilustración 2.1. Esta sección del código original corresponde con la que podemos ver la Ilustración 6.3.

```
236 def imageflow_demo(predictor, vis_folder, current_time, args):
237     cap = cv2.VideoCapture(args.path if args.demo == "video" else args.camid)
238     width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)  # float
239     height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)  # float
240     fps = cap.get(cv2.CAP_PROP_FPS)
241     timestamp = time.strftime("%Y_%m_%d_%H_%M_%S", current_time)
242     save_folder = osp.join(vis_folder, timestamp)
243     os.makedirs(save_folder, exist_ok=True)
244     if args.demo == "video":
245         #save_path = osp.join(save_folder, args.path.split("/")[-1])
246         save_path = osp.join(save_folder, args.path.split('\\')[-1])
247     else:
248         save_path = osp.join(save_folder, "camera.mp4")
249     logger.info(f"video save_path is {save_path}")
250     vid_writer = cv2.VideoWriter(
251         save_path, cv2.VideoWriter_fourcc(*"mp4v"), fps, (int(width), int(height))
252     )
253     tracker = BYTETracker(args, frame_rate=30)
254     timer = Timer()
255     frame_id = 0
256     results = []
257     while True:
258         if frame_id % 20 == 0:
259             logger.info('Processing frame {} ({:.2f} fps)'.format(frame_id, 1. / max(1e-5, timer.average_time)))
260         ret_val, frame = cap.read()
261         if ret_val:
262             outputs, img_info = predictor.inference(frame, timer)
263             if outputs[0] is not None:
264                 online_targets = tracker.update(outputs[0], [img_info['height'], img_info['width']], exp.test_size)
265                 online_tlwhs = []
266                 online_ids = []
267                 online_scores = []
268                 for t in online_targets:
269                     tlwh = t.tlwh
270                     tid = t.track_id
271                     vertical = tlwh[2] / tlwh[3] > args.aspect_ratio_thresh
272                     if tlwh[2] * tlwh[3] > args.min_box_area and not vertical:
273                         online_tlwhs.append(tlwh)
274                         online_ids.append(tid)
275                         online_scores.append(t.score)
276                         results.append(
277                             f"{frame_id},{tid},{tlwh[0]:.2f},{tlwh[1]:.2f},{tlwh[2]:.2f},{tlwh[3]:.2f},{t.score:.2f},-1,-1,-1\n"
278                         )
279                 timer.toc()
280                 online_im = plot_tracking(
281                     img_info['raw_img'], online_tlwhs, online_ids, frame_id=frame_id + 1, fps=1. / timer.average_time
282                 )
283             else:
284                 timer.toc()
285                 online_im = img_info['raw_img']
286             if args.save_result:
287                 vid_writer.write(online_im)
288             ch = cv2.waitKey(1)
289             if ch == 27 or ch == ord("q") or ch == ord("Q"):
290                 break
291         else:
292             break
293         frame_id += 1
```

Ilustración 6.3: Código original donde se manejan los datos del anális

En esta sección de código es donde se centrará todo nuestro proyecto en lo que respecta a ByteTrack, modificamos este código original para recabar los datos que necesitamos. Como vemos, al inicio de la función se concretan los datos acerca del vídeo que será analizado, como las dimensiones del vídeo y donde ocurre todo el proceso del análisis es en el bucle while que comienza en la línea 257. En orden de interés en lo que respecta a la utilidad final podemos ver que se lee el frame, el cual debe pasar por dos procesos, primero por la función inference

de la clase *predictor* y el resultado pasarlo al objeto *tracker*, que actualizará la información de seguimiento con este nuevo frame, y tras tratar los datos finalmente obtenemos toda la información separada para usar más cómodamente: *online_tlwhs*, son los datos de coordenadas y dimensiones de cada detección; y *online_ids*, que son las etiquetas identificativas de dichas detecciones, asociadas entre sí por posición en cada vector.

Dado que de esta manera realiza todo el análisis del vídeo sin poder acceder a los datos a medida que este avanza al ser un bucle infinito, se propuso una solución: crear una clase en la que pudiera quedar encapsulada toda esta información y se actualizase cada vez que se deseara, obteniendo la información cada actualización sin tener que esperar el proceso completo.

6.3. Clase Tracker

La única intención de hacer toda la refactorización del código no fue otra que poder obtener información en tiempo real, por tanto, se debía mantener la funcionalidad intacta, con lo que el equivalente a este bucle sería la función que observamos en la Ilustración 6.4, que pertenece a la clase creada, que hemos llamado *Tracker*, y que encapsula toda la información acerca del análisis, manteniendo el estado del análisis en todo momento mientras exista alguna instancia de la clase.

```
66      def getFrame(self):
67          ret_val, frame = self.cap.read()
68          if ret_val:
69              outputs, img_info = self.predictor.inference(frame, self.timer)
70              if outputs[0] is not None:
71                  online_targets = self.tracker.update(outputs[0], [img_info['height'], img_info['width']], self.exp.test_size)
72                  online_tlwhs = []
73                  online_ids = []
74                  for t in online_targets:
75                      tlwh = t.tlwh
76                      tid = t.track_id
77                      vertical = tlwh[2] / tlwh[3] > 1.6
78                      if tlwh[2] * tlwh[3] > 10 and not vertical:
79                          online_tlwhs.append(tlwh)
80                          online_ids.append(tid)
81                  self.timer.toc()
82                  #las detecciones por tiempo, aunque la persona no se mueva, se incrementa cada frame la aparición
83                  for item in range(len(online_tlwhs)):
84                      x,y,w,h = online_tlwhs[item]
85                      id = online_ids[item]
86
87                      min_move = 5 #parametro ajustable
88
89                      #caso en que las detecciones son por movimiento, una persona que permanece quieta no se seguira contando como detecciones en cada frame
90                      if not self.time_type and id in self.detection_dict and (abs(int(x) - self.detection_dict[id][0]) < min_move) and  (abs(int(y) - self.detection_dict[id][1]) < min_move):
91                          continue
92                      midPoint = int(x) + int(w/2)
93                      #sigue dentro de la imagen
94                      if midPoint < self.width and midPoint >= 0 and int(y) + int(h) < self.height and int(y) + int(h) >= 0:
95
96                          self.matrix_detection[int(y) + int(h), midPoint] += 30
97
98                          #coloreamos el entorno alrededor del punto detectado
99                          radius = 30 #parametro ajustable
100
101                         #todas las coordenadas para calcular las distancias
102                         coord_x, coord_y = np.meshgrid(np.arange(self.matrix_detection.shape[1]), np.arange(self.matrix_detection.shape[0]))
103
104                         #matriz que contiene todas las distancias con respecto al punto de deteccion
105                         distances_matrix = np.sqrt((coord_x-midPoint)**2+(coord_y-(int(y) + int(h)))**2)
106
107                         #suma detecciones en un radio con respecto al punto de detección original
108                         self.matrix_detection[np.where(distances_matrix <= radius)] += 20
109
110                         #en caso de que las detecciones sean por movimiento tendremos que llevar un histórico de la última posición contada
111                         if not self.time_type:
112                             self.detection_dict[id] = (int(x),int(y),int(w),int(h), self.matrix_detection[int(y) + int(h), midPoint])
113                 copy = self.matrix_detection.copy()
114                 copy[copy == 0] = np.nan
115                 logaritmic_version = np.log10(copy)
116                 logaritmic_version[logaritmic_version == np.isnan(logaritmic_version)] = 0
117
118                 live_colorMap = None
119                 live_colorMap = cv2.normalize(logaritmic_version, live_colorMap, alpha=0, beta=255, norm_type = cv2.NORM_MINMAX, dtype=cv2.CV_8U)
120                 live_colorMap = cv2.applyColorMap(live_colorMap, cv2.COLORMAP_JET)
121
122             self.frame_id += 1
123             return frame, live_colorMap
```

Ilustración 6.4: Función que devuelve análisis de cada frame

De esta manera, en lugar de tener que esperar a que finalice el proceso completo para poder generar un mapa de calor, podemos llamar a esta función repetidas veces desde la interfaz

de usuario para obtener el que tenemos generado hasta el momento y poder mostrarlo en tiempo real, cada llamada podría verse como una iteración del bucle de la Ilustración 6.3.

La clase *Tracker* que hemos creado contiene datos tales como el video que es sometido a análisis y un estado activo continuamente actualizado del proceso de análisis. La inicialización de esta clase recibe dos parámetros: dirección del archivo en la memoria local de nuestro ordenador, y el tipo de proceso analítico que se aplicará a dicha secuencia de imágenes, por movimiento o por tiempo, simbolizado por la variable booleana de la clase llamada *time_type*, la cual por defecto toma el valor falso, es decir, análisis por movimiento.

El proceso de generación de los mapas de calor comienza tras el tratamiento de los datos obtenidos en el frame, en la línea 152 hasta el final de la ilustración. El estado de los datos detectados se almacena en la variable *matrix_detection*, que se inicializa con un tamaño equivalente al del vídeo y todo a ceros, porque cada posición representa el número de detecciones por píxel. Como ya habíamos visto, el contenido de *online_tlwhs* es el total de detecciones del frame actual, debemos recorrerlo y con dicha información actualizar nuestra matriz de detecciones. Para facilitar el uso de los datos, los separamos en diferentes variables locales, líneas 153 y 154. La variable *min_move* y el diccionario *detection_dict* toman sentido cuando el mapa de calor que pretendemos generar es por movimiento. La segunda de estas llevará un control de la posición de cada persona detectada en el frame actual, información que usará el siguiente con el par identificación-ubicación. Al llegar a la condición de la línea 159, comprobamos que, en caso de estar actualizando en base a movimiento, tendremos que corroborar si contamos con la posición previa de la persona actual, y de ser así, comparar con el movimiento mínimo requerido (*min_move*) que se ha desplazado lo suficiente como para actualizar los datos, y si no existe historial previo, se tendrá en cuenta al ser su primera a aparición.

La generación de dos tipos de mapas de calor, por tiempo o por movimiento, se ha pensado e implementado contando con la posibilidad de que en los entornos capturados puede darse la posibilidad de que haya personas que permanezcan estáticas en un mismo punto durante casi toda la grabación, lo que podría distorsionar los datos generales del mapa al ser un pico demasiado elevado con respecto a los demás valores, que con mucha mayor probabilidad, estarán repartidas por la imagen. Inicialmente desarrollamos el análisis por tiempo, que consistía únicamente en contar el número de detecciones por coordenadas, y fue al realizar las pruebas que advertimos el daño que provocaba al resultado final las personas que permenecen estáticas. No obstante siendo un daño para la primera impresión que nos daba, nos percatamos de que puede ser un dato muy útil según el propósito de saber el movimiento o permanencia de las personas según el contexto, dado que no es lo mismo querer ubicar un negocio que una publicidad, siendo para el primera útil el análisis por tiempo y para el segundo por movimiento, provocando que se vea por una mayor cantidad de personas diferentes, y por ello se decidió mantener los dos tipos de análisis en lugar de uno más genérico.

Continuando con la ejecución del código, la variable *midPoint* se creó con la intención de tomar como detección los pies de las personas detectadas ya que las coordenadas devueltas por ByteTrack corresponden a la esquina superior derecha. La condición de la línea 163 es el comienzo de actualización de la matriz de detecciones una vez decidido si los datos actuales

de la persona son válidos, este bloque de código es común a ambos procesos, tanto por tiempo como por movimiento, el cual solo se ignora si las condiciones del segundo no se cumplen, de cualquier otra forma este bloque siempre se ejecuta.

Veamos la parte donde se actualiza la matriz de detecciones, contará como detección aquellas que cumplan con las condiciones nombradas hasta ahora además de que esta debe estar aún dentro de la imagen, dado que puede ser que la zona que nos interesa de la persona aún no se encuentra dentro de los límites de la imagen. Tras esto, si contásemos únicamente las coordenadas devueltas, las probabilidades de que gran número de personas pase por el mismo píxel de forma continuada son limitadas, por ello, con cada una de estas coordenadas, se incrementa considerablemente (en este caso un incremento de 30 en la línea 165) dicho punto, pero además incrementamos un radio en torno a este, aunque en menor medida, dando más peso al origen. Para conseguir esto último, usamos la función *meshgrid* de la librería *numpy*, que nos permite definir una malla de todas las coordenadas existentes en la imagen para calcular las distancias de cada píxel con respecto a la persona que hayamos detectado en cada momento, y con ello incrementar solo aquellos donde la distancia sea menor que la deseada, 30 píxeles en nuestro caso, pudiendo ampliar utilidades de cara a nuevas implementaciones permitiendo que el usuario escoja este valor.

Por último, la matriz de detecciones obtenida cuenta con un problema grave a la hora de visualizar los datos, las grandes diferencias entre valores provocan que se dificulte mucho visualizarlos cómodamente, por ello aplicamos una escala logarítmica para suavizarlos, aun representando las mismas proporciones, y de esta manera obtenemos un mapa de calor mucho más suave sin contraste tan pronunciados, para ver las diferencias entre aplicar o no escala logarítmica podemos dirigirnos a la Ilustración 6.5.

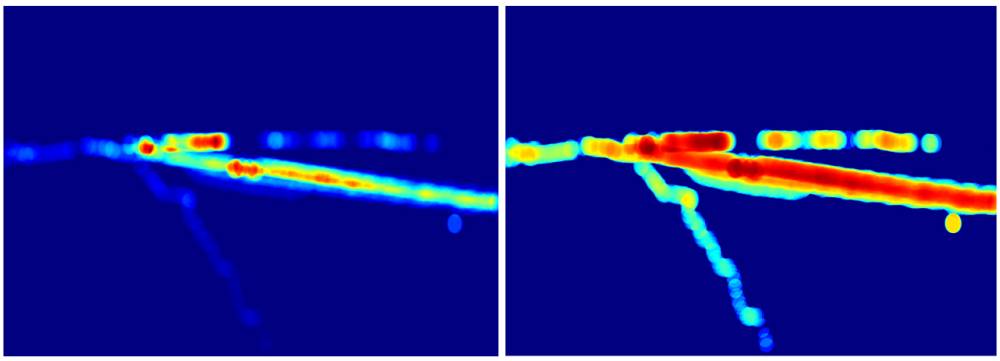

Ilustración 6.5: A la izquierda escala lineal y a la derecha escala logarítmica.

Esto ocurre por la enorme diferencia entre los valores pequeños y los grandes, ambas imágenes de la Ilustración 6.5 representan la misma información proporcional, lo único que cambia es que hemos homogeneizado las diferencias, dándole más importancia a los valores menores en relación con los más elevados.

6.4. Interfaz de usuario

Ya hemos mencionado que la interfaz gráfica ha sido desarrollada con la librería *tkinter*, que nos aporta un gran número de elementos gráficos muy útiles para la implementación.

Se ha creado una clase que contenga todo lo relacionado con la parte visual de la aplicación, que se ha llamado *UI_Tacker*, lo cual se ha hecho para separar todos los componentes de forma individual, y esta será el punto de inicio que ejecutará nuestro archivo main.

Todos los elementos que conforman la interfaz son objetos de *tkinter* y cada uno de ellos recibe ciertos parámetros según su propósito. Nuestra interfaz está compuesta por los siguientes elementos:

Botones: Se crean mediante la clase *Button* y reciben dos parámetros fundamentales, el primero de ellos es *text*, que define el texto que contendrá para informar al usuario de la acción que desempeña; y el *command*, que recibe el nombre de la función a ejecutar cuando se pulse el botón. Ejemplo de ello en la línea 46 de la Ilustración 6.6.

Canvas: Estos elementos se crean con la clase llamada *CTKCanvas*, que pertenece a la librería *customtkinter*, que hereda de *tkinter* y serán los encargados de mostrar tanto el vídeo original como el mapa de calor asociado a este. Para estos objetos, los parámetros a destacar son el ancho (*width*) y el alto (*height*), cuyo valor será el número de píxeles que ocuparán, línea 52. Estos valores los hemos definido de tal forma que se adapten a las dimensiones de la pantalla que esté usando el usuario.

Además de los parámetros mencionados referentes a cada objeto según relevancia, hay parámetros comunes que son esenciales para la ubicación de los elementos en la interfaz. En *tkinter* hay dos formas de definir el layout, y hemos usado ambas, la primera de ellas es definiendo una malla de filas y columnas, configuración que podemos ver en las líneas 41-44 de la Ilustración 6.6, donde cada celda sería el contenedor de un compenente, usando el número de fila y columna como coordenadas en la función *grid*, línea 53 de la Ilustración 6.6. La segunda forma sería mediante el método *place*, que recibe como argumentos las coordenadas (x, y) con respecto al contenedor, como puede ser en este caso la ventana completa de la interfaz, estos valores son el número de píxeles usando como referencia la esquina superior izquierda, un ejemplo de esto podemos verlo en las líneas 47 y 50 de la Ilustración previamente mencionada. Para este segundo caso se han calculado las coordenadas teniendo en cuenta las diferentes dimensiones que pueda llegar a tener la pantalla en uso, mientras que al configurar la malla de la primera forma, las filas y columnas se organizan de una forma automática.

En lo relacionado a mostrar el mapa de calor, el cambio de perspectiva homográfica se realiza en esta clase, usando una combinación de las funciones *findHomography* y *warpPerspective*, teniendo en cuenta los puntos que ha seleccionado el usuario habiendo indicado la región de la imagen que desea ver como resultado mediante clicks. Estos puntos que definen la nueva perspectiva se han recogido asociando un evento al canvas que muestra el vídeo original con la función *bind*, que podemos ver en las líneas 54-55 de la Ilustración 6.6, esta recibe como parámetros el tipo de evento y la función que llama cada vez que se active. Con el click izquierdo se añade un punto de perspectiva y con el derecho se borra el último añadido,

```
40        self.app.resizable(False, False)
41        self.app.grid_rowconfigure(1, weight=5)
42        self.app.grid_rowconfigure(0, weight=1)
43        self.app.grid_columnconfigure((0,1,3,4), weight = 2)
44        self.app.grid_columnconfigure(2, weight = 1)
45
46        select_video_button = tk.Button(master=app, text="Selección vídeo", font=("Times", 25), command=self.select_video, bg = "#c8c8c8", width=30)
47        select_video_button.place(x=0.1*self.screenDim[0], y=0.1*self.screenDim[1])
48
49        open_camera_button = tk.Button(master=app, text="Abrir cámara", font=("Times", 25), command=self.open_camera_button, bg = "#c8c8c8", width=30)
50        open_camera_button.place(x=0.6*self.screenDim[0], y=0.1*self.screenDim[1])
51
52        self.original_canvas = customtkinter.CTkCanvas(master = app, width = self.canvasDim[0], height = self.canvasDim[1], bg = "white")
53        self.original_canvas.grid(row = 1, column= 0, columnspan=2)
54        self.original_canvas.bind("<Button-1>", self.mouse_click)
55        self.original_canvas.bind("<Button-3>", self.mouse_remove)
```

Ilustración 6.6: Sección de código de *UI_Tracker*

cada uno de ellos tiene un color asociado que es equivalente a una esquina del canvas que muestra el resultado, en cada esquina de este segundo canvas se ha puesto un mismo punto de color para darle al usuario las referencias que definirán la nueva perspectiva.

Con este desarrollo de la interfaz finalmente conseguimos una implementación muy cercana a la idea inicial vista en la Ilustración 5.1. Podemos ver el resultado final de este desarrollo en mitad un proceso de análisis con cambio de perspectiva aplicado en la Ilustración 6.7.

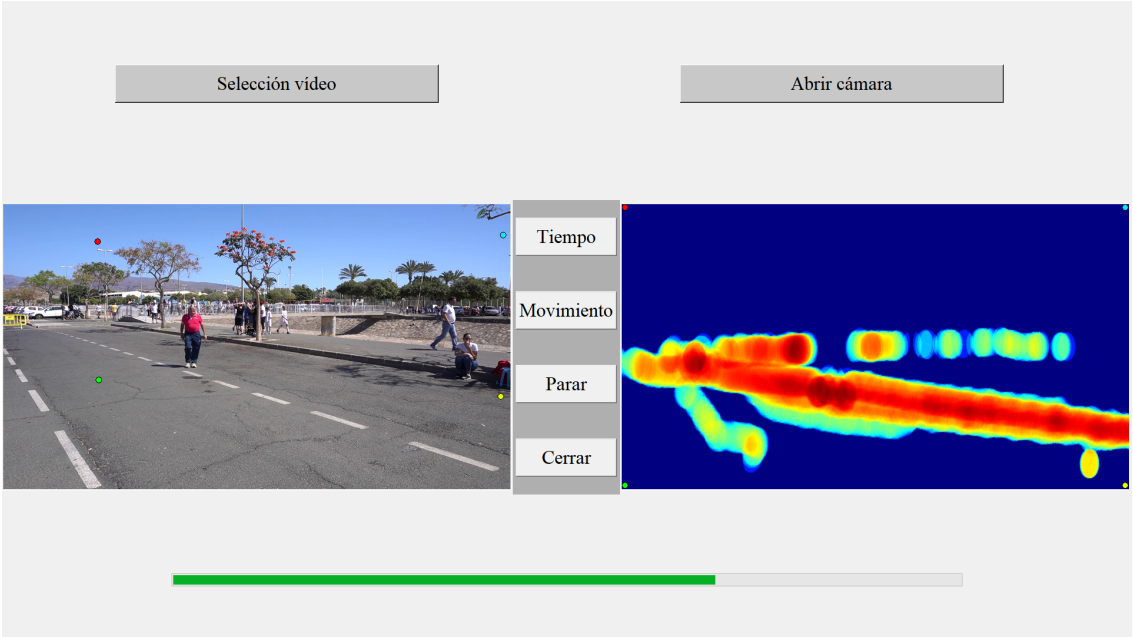

Ilustración 6.7: Ejecución de la versión final del proyecto

Cuando contábamos con la que creíamos que era la versión final del proyecto, al realizar las pruebas pudimos notar un rendimiento notablemente bajo una vez se iniciaba el proceso de análisis. Este problema ocurría porque como el código funciona de manera secuencial, al lanzar la herramienta de análisis la interfaz dejaba de actualizarse y quedaba en pausa a la espera de la devolución de los datos del análisis, lo que se traducía en una pantalla congelada que ignoraba las acciones del usuario hasta que este análisis se terminara, lo cual ocurría con cada frame. Para solucionar este poblema se decidió lanzar este proceso de análisis en segundo plano gracias a la librería *threading*, y en lugar de forzar a la interfaz a realizar una

espera, esta podía continuar actualizándose y en cada actualización realizar una consulta acerca de si el proceso lanzado había terminado y mostrar los nuevos datos actualizados o, de no ser así, seguir mostrando los últimos datos con los que contaba. Con esto se consiguió volver a una interfaz mucho más fluida, dejando que el costoso proceso de análisis no afectara a otros ámbitos de la aplicación. Este proceso podemos verlo en la función *execution*, donde se entra cada vez que se ha decidido el tipo de análisis a aplicar. La lógica que se ha seguido es la siguiente, si no existe ningún hilo quiere decir que debemos lanzar uno que ejecute el proceso que analiza el próximo frame a mostrar, condición de la línea 150 de la Ilustración 6.8, y si existe un hilo quiere decir que estamos esperando a que este finalice para mostrar la información que ha obtenido actulizando la interfaz, condición que podemos ver en la línea 154 de la Ilustración.

```
150         if self.thread is None:
151             self.thread = Thread(target=self.getFrame)
152             self.thread.start()
153 
154         if self.thread is not None and not self.thread.is_alive():
155             self.thread = None
156             self.frameCount += 1
157             self.proccessProgress.set((100 / self.totalFrames) * self.frameCount)
158             self.frame = cv2.resize(self.frame, self.canvasDim)
159             self.frame_rgb = cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB)
160             self.originalPhoto = ImageTk.PhotoImage(image=Image.fromarray(self.frame_rgb))
161             self.original_canvas.create_image(0, 0, image=self.originalPhoto, anchor=tk.NW)
162             self.draw_perspective_points()
```

Ilustración 6.8: Sección de código acerca del subproceso

6.5. Pruebas

Durante todo el desarrollo se han ido realizando pruebas unitarias relacionadas con cada nueva funcionalidad para comprobar el correcto funcionamiento, desde los pasos iniciales de conseguir una eficiente integración de ByteTRack para analizar un vídeo hasta un uso del resultado final de la aplicación con vistas a una pronta entrega del proyecto. La base fundamental en cada una de las pruebas unitarias tanto como en las etapas finales siempre ha consistido en contar con un amplio abanico de vídeos de distinto tipo y ver si se cumplen los objetivos con todos ellos. Hemos calculado el tiempo promedio por frame y ha quedado de la siguiente manera:

Análisis por tiempo: Este proceso ha tardado una media de 1.72 segundos por frame.

Análisis por movimiento: Este proceso ha tardado una media de 0.71 segundos por frame.

Estos datos se han dado así porque en el segundo hay muchas detecciones que se ignoran si no cumplen ciertos requisitos y este tiempo se ahorra, mientras que en el primero no hay filtros que pasar y se analiza todo, siendo muy costoso.

6.6. Requisitos e instalación

Los requisitos previos asi como dependencias que deban instalarse, además de los pasos a seguir para poder hacer uso de esta aplicación se podrán encontrar en el fichero *README* que esta en la raíz del proyecto. De esta manera dicho fichero será actualizado en el tiempo conforme se añadan nuevas funcionalidades.

Capítulo 7

Conclusiones e implementaciones futuras

El resultado de las pruebas de la aplicación una vez finalizado todo el desarrollo fue más que satisfactorio al haber conseguido casi a la perfección acercarnos a la idea inicial con la que contábamos sin habernos encontrado con demasiados problemas más allá del estudio de nuevas herramientas y la necesaria búsqueda de mejorar el rendimiento, el cual de por si ya se ve una clara diferencia cuando se escoge el análisis por tiempo o por movimiento. Hemos podido comprobar que el método nos aporta una gran precisión en las detecciones obtenidas.

Se ha dejado probar la interfaz a usuarios aleatorios, pudiendo ver que tras un tras unos pocos minutos y sin necesidad de explicación previa más que sabiendo el propósito de la aplicación, han sido capaces de hacer un uso esperado de la misma, lo que nos confirma que la interfaz cumple con el obejtivo de ser intuitiva para todo tipo de usuarios.

Algunas de las posibles mejoras futuras podría ser implementar el análisis de las imágenes captadas por cámara en tiempo real, sin necesidad de contar con un vídeo; mejorar aún más el rendimiento buscando algoritmos y métodos más eficaces o ampliar el número de funcionalidades desde la interfaz para aportar una experiencia de uso más completa.

Bibliografía

[1] Bot-sort github. https://github.com/NirAharon/BoT-SORT. Accessed: (19/12/2023).

[2] Bytetrack github. https://github.com/ifzhang/ByteTrack. Accessed: (19/12/2023).

[com] Comparativa de métodos. https://medium.com/@pedroazevedo6/object-tracking-state-of-the-art-2022-fe9457b77382. Accessed: (19/12/2023).

[scr] Metodología scrum. https://www.troopsf.com/scrum/. Accessed: (19/12/2023).

[pro] Reglamento general de protección de datos. https://www.boe.es/buscar/doc.php?id=DOUE-L-2016-80807. Accessed: (19/12/2023).

[rol] Roles scrum. https://desire.webs.uvigo.es/contenidos/scrum/. Accessed: (19/12/2023).

[7] Smiiletrack github. https://github.com/WWangYuHsiang/SMILEtrack. Accessed: (19/12/2023).

[per] Transformación homográfica. https://learnopencv.com/homography-examples-using-opencv-python-c/. Accessed: (19/12/2023).

[OZmap] OZmap. Imagen de ozmap. https://ajuda.ozmap.com.br/es-LA/support/solutions/articles/44001759401-mapa-de-calor. Accessed: (19/12/2023).

[10] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.