**Índice**

[**Introducción y objetivos** 2](#_Toc156071229)

[**Descripción Técnica** 2](#_Toc156071230)

[**Cámara Perfil: Conteo y Detección de Personas** 2](#_Toc156071231)

[*Detección* 2](#_Toc156071232)

[*Controlador y Contador* 3](#_Toc156071233)

[*Código Principal* 4](#_Toc156071234)

[**Cámara Frontal: Análisis de Género y Edad** 4](#_Toc156071235)

[*Clasificador de Caras* 4](#_Toc156071236)

[*Tratado de Imagen* 6](#_Toc156071237)

[*Código Principal* 6](#_Toc156071238)

[**Conclusión y Posibles Ampliaciones** 6](#_Toc156071239)

[**Tecnologías Usadas** 6](#_Toc156071240)

[**Repositorios** 6](#_Toc156071241)

**Introducción y objetivos**

El presente estudio se focaliza en la identificación, recuento y análisis de la edad y género de individuos mediante el empleo de *YOLOv8* y *MediaPipe*. El propósito fundamental es poder examinar y supervisar el tránsito de personas dentro de un establecimiento, posibilitando la generación de estadísticas detalladas sobre la cantidad y tipología de personas que ingresan a lo largo del día.

La disposición de una cámara posicionada lateralmente permite la detección de personas que atraviesan la entrada del establecimiento, llevándose a cabo un seguimiento para determinar si están ingresando o saliendo del edificio. Una vez que una persona cruza la línea límite, indicando su entrada, se activa una segunda cámara ubicada frontalmente en la entrada. Esta cámara captura una imagen y realiza un análisis para determinar el género y la edad de la persona detectada.

Los datos recopilados se almacenan en una base de datos dedicada, la cual se utilizará para generar estadísticas detalladas sobre el flujo de personas en el establecimiento a lo largo del año. La información, tanto en forma de vídeos de las cámaras como de métricas y gráficos, estará accesible a través de un servidor web especialmente desarrollado para este propósito.

**Descripción Técnica**

## **Cámara Perfil: Conteo y Detección de Personas**

Este proceso se divide en dos partes: detección de las personas que capture la cámara y el conteo de cuántas entran y salen del lugar. Solo se puede contar una persona que pase a la vez por lo que, en esta fase, el proyecto solo está pensado para establecimientos o servicios cuyas entradas solo den capacidad para una persona.

Este código se encuentra dentro del repositorio [vc-trabajo-final-cliente-deteccion-clientes](https://github.com/VC-ulpgc-JRP-y-SGR/vc-trabajo-final-cliente-deteccion-clientes).

### *Detección*

Para la detección, se han desarrollado dos clases fundamentales dentro del archivo *detection.py*: **Person** y **YOLOPersonDetector**.

***Clase Person***

La primera clase, denominada Person, desempeña un papel esencial en la gestión individual de cada persona detectada. Su función principal consiste en almacenar información relevante sobre la persona, como las coordenadas de la caja delimitadora de su figura, la posición central, la trayectoria, la dirección del movimiento, tiempo de vida de la persona siendo detectada y un indicador que señala si ha superado el tiempo máximo de seguimiento.

Este seguimiento temporal se realiza mediante métodos como **age\_one()**, que incrementa la edad de la persona en una unidad, y **is\_time\_out()**, que verifica si la persona ha excedido su tiempo de seguimiento máximo.

Asimismo, la clase posee el método **update\_coords()** para actualizar las coordenadas de la persona y el método **calculate\_dir()** para calcular su dirección de movimiento con relación a unos límites especificados.

***Clase YOLOPersonDetector***

La segunda clase, YOLOPersonDetector, se encarga de integrar el modelo YOLO para la detección de personas en una imagen.

Su método principal, **predict()**, acepta una imagen como entrada y devuelve una lista de las coordenadas de las personas detectadas. Este método utiliza la biblioteca *ultralytics* para realizar predicciones a partir del modelo YOLO, extrayendo las cajas delimitadoras de los objetos detectados y verificando la categoría del objeto para determinar si corresponde a una persona. En caso afirmativo, las coordenadas se agregan a la lista de las coordenadas de las personas detectadas y se devuelve como resultado.

### *Controlador y Contador*

Se han diseñado dos clases fundamentales para gestionar la visualización de información y llevar a cabo el seguimiento y conteo de individuos detectados en cada frame. Estas clases, denominadas **Painter** y **PersonCounterController**, cumplen roles clave en el sistema y se detallan a continuación.

***Clase Painter***

La clase **Painter** se encarga de la presentación visual de la información en la imagen procesada. Su función principal es dibujar líneas representativas, contar el número de personas detectadas y resaltar visualmente la presencia de cada individuo en la imagen.

Se utilizan diversos métodos, como **paint\_lines()**, que dibuja líneas divisorias y texto en la imagen para etiquetar las líneas de delimitan los límites de control, y **paint\_counter()**, que muestra un contador de personas en una esquina de la imagen. Además, el método **paint\_person()** resalta visualmente cada persona detectada, marcando su caja delimitadora, centro y etiqueta.

***Clase PersonCounterController***

La segunda clase, **PersonCounterController**, opera como el controlador principal de la aplicación. Se encarga de orquestar la interacción entre el detector de personas, el presentador visual y de avisar al servidor de que una persona ha entrado al local.

Al recibir un frame de la cámara, redimensiona la imagen según las especificaciones proporcionadas y realiza predicciones de detección de personas mediante el objeto **YOLOPersonDetector**. Luego, utiliza el objeto **Painter** para visualizar líneas divisorias, contadores y la presencia de personas en la imagen.

El método **track()** se encarga de seguir el movimiento de las personas detectadas y calcular su dirección en relación con límites específicos en la imagen con el método realizado en la clase **Person**, **calculate\_dir()**.

Por otro lado, el método **count()** actualiza el contador de personas en función de la dirección de movimiento de cada persona detectada y, a su vez, va enviando esta información al servidor, avisando a la **Cámara Frontal** de que tiene que realizar una foto cuando el contador suma 1.

### *Código Principal*

En el script principal, llamado **main.py**, se establece la lógica central para la ejecución del sistema de conteo y seguimiento de personas.

En la primera sección, se importan las clases esenciales desde los módulos **control** y **camera**. La clase **PersonCounterController** se encarga del control del conteo de personas, mientras que **CameraServer** se utiliza para el envío de frames al servidor desde la cámara.

La función principal, **main()**, inicializa instancias de las clases mencionadas y configura parámetros clave como el tamaño de la imagen, límites de conteo, edad máxima y el modelo YOLO a utilizar. Posteriormente, se utiliza la cámara del sistema, inicializada mediante OpenCV, para capturar frames en un bucle continuo. Cada frame capturado es procesado por el objeto **PersonCounterController**, llevando a cabo la detección de personas, así como el seguimiento y conteo.

En resumen, **main.py** funciona como el punto de entrada del sistema, coordinando la adquisición y procesamiento de frames, y presentando visualmente los resultados de la detección y conteo de personas en tiempo real. La modularidad de las clases facilita una fácil expansión y mantenimiento del sistema.

## **Cámara Frontal: Análisis de Género y Edad**

Este proceso realiza la detección de rostros y el análisis de género y edad a partir de imágenes capturadas por la cámara frontal.

Este código se encuentra dentro del repositorio [vc-trabajo-final-cliente-person-qualifier](https://github.com/VC-ulpgc-JRP-y-SGR/vc-trabajo-final-cliente-person-qualifier).

### 

### *Clasificador de Caras*

***Interfaz FaceDetector y Dataclasses***

En el archivo **face\_detector.py**, se ha diseñado una estructura modular para la detección de caras en imágenes. Se utilizan las bibliotecas **numpy** para manipulación de datos y **dataclasses** para definir clases inmutables. Además, se emplea la clase abstracta **ABC** del módulo **abc** para establecer una interfaz abstracta que las implementaciones específicas deben cumplir.

Tres clases de datos se definen utilizando la anotación **@dataclass**: **Point** para representar puntos en un plano, **BoundingBox** para delimitar objetos en una imagen, y **FaceDetectorResult** para almacenar los resultados de la detección de caras, incluyendo la imagen y la caja delimitadora.

La clase abstracta **FaceDetector** establece una interfaz que exige que cualquier implementación proporcione un método **detect()**. Este método recibe una imagen representada como un array de **numpy** y devuelve una lista de resultados de detección de caras.

***Clase MediaPipeFaceDetector***

En **mediapipe\_face\_detector.py**, la clase **MediaPipeFaceDetector** ofrece una implementación específica del detector de caras utilizando la biblioteca MediaPipe. A continuación, se describen detalladamente los métodos clave de esta implementación.

El método **detect()** implementa la detección de caras utilizando la biblioteca MediaPipe. Se inicia definiendo las opciones necesarias para el detector de caras, incluyendo el modelo específico (**blaze\_face\_short\_range.tflite**). Luego, se crea un objeto detector utilizando estas opciones y se procesa la imagen mediante el método **detect()** del detector de caras. La información resultante se convierte en instancias de la clase **FaceDetectorResult** mediante el método **\_convert\_to\_face\_detector\_result()**. Finalmente, se devuelve una lista de los resultados de la detección.

El método **\_convert\_to\_face\_detector\_result()** transforma los resultados de la detección proporcionados por MediaPipe en instancias de la clase **FaceDetectorResult**. Recibe las coordenadas de las caras detectadas y las convierte en instancias de la clase **BoundingBox** que representa una caja delimitadora. Luego, para cada caja delimitadora, se utiliza el método **ImageUtils.crop()** para recortar la imagen original y se crea una instancia de **FaceDetectorResult** que contiene la imagen recortada y la caja delimitadora correspondiente. La lista resultante de instancias de **FaceDetectorResult** se devuelve.

***Clase ViolaJonesFaceDetector***

La clase **ViolaJonesFaceDetector** implementa la detección de caras utilizando el clasificador de Viola-Jones mediante la biblioteca OpenCV. En el método **detect()**, se carga el clasificador previamente entrenado y se aplica para identificar caras potenciales en la imagen. Las coordenadas de las caras detectadas se convierten luego en instancias de la clase **FaceDetectorResult** mediante el método **\_convert\_to\_face\_detector\_result()**.

El método **\_convert\_to\_face\_detector\_result()** transforma las coordenadas de las caras en instancias de **BoundingBox** y utiliza el método **ImageUtils.crop()** para recortar la imagen original. Se crea una lista de instancias de **FaceDetectorResult** que contienen la imagen recortada y la caja delimitadora correspondiente.

Por otro lado, la función **\_convert\_image\_to\_gray** convierte una imagen a escala de grises utilizando la función **cv2.cvtColor()** de OpenCV.

Esta clase es una adición ya que realmente no se utiliza en el código final.

### *Tratado de Imagen*

En el archivo **utils.py** se ha desarrollado la clase **ImageUtils**, la cual proporciona funciones útiles para el tratado de imágenes. A continuación, se explica cada función de forma detallada.

El método estático **crop()** recibe una imagen representada como un array de **numpy** y una instancia de la clase **BoundingBox**. Utilizando las coordenadas de la caja delimitadora, se recorta la región correspondiente de la imagen y se devuelve como un nuevo array de **numpy**. Este método es útil para extraer partes específicas de una imagen, como las caras detectadas.

El método **overlay\_icons()** superpone un icono en una imagen en una posición específica. Toma la imagen original, la ruta del icono, un color, el tamaño del icono y las coordenadas del punto de superposición. El icono se lee y redimensiona según el tamaño proporcionado. Luego, se crea una máscara del icono donde los píxeles son 0, y se genera una capa de color del tamaño del icono. La capa de color se copia sobre la región de la imagen original especificada por las coordenadas del punto de superposición, respetando la máscara del icono. El resultado es una imagen con el icono superpuesto.

### *Código Principal*

En el script principal de clasificación de caras, **main.py**, se realiza la detección y resaltado de caras en tiempo real utilizando las clases creadas con MediaPipe y transmite los fotogramas procesados a través de un servidor.

Comienza importando las bibliotecas necesarias, incluyendo OpenCV para la manipulación de video, el módulo de registro (**logging**), la implementación de detección de caras con MediaPipe (**MediaPipeFaceDetector**), y el detector de caras basado en el clasificador de Viola-Jones (**ViolaJonesFaceDetector**). También se importa la clase **CameraServer** para la transmisión de video.

Luego, se inicializa la captura de video desde la cámara utilizando **cv2.VideoCapture(0)**. Se crea una instancia de **CameraServer** que inicia un servidor para enviar los fotogramas capturados.

En el bucle principal (**while True**), se lee cada fotograma de la cámara. Se utiliza la instancia de **MediaPipeFaceDetector** para detectar caras en el fotograma y se resalta cada cara detectada con un rectángulo azul en la imagen original.

Posteriormente, el fotograma modificado se envía al servidor mediante el método **server.send\_frame()**. Se maneja cualquier excepción que pueda ocurrir durante el proceso de envío, y se registra un mensaje de error si es necesario.

El bucle se ejecuta continuamente hasta que se interrumpe manualmente o se produce una excepción. Finalmente, se cierra la conexión del servidor con **server.close()**.

**Servidor Web**

# **Conclusión y Posibles Ampliaciones**

# 

# **Tecnologías Usadas**

# **Repositorios**