

Estimación de la altura de una persona basada en un objeto de referencia

Darío López Villegas
Juan Javier Sánchez Portillo
Paulo Felipe Rezende da Silva

Diciembre 2025

Resumen

Este trabajo presenta el diseño y la implementación de un sistema para la estimación de la altura humana en imágenes digitales, abordando la problemática de información de profundidad mediante el uso de un objeto de dimensiones conocidas, en el caso, un folio A4. El proyecto desarrollado en el ámbito de la asignatura de Procesamiento de Imágenes Digitales con una carga de 50 horas por persona, con la combinación de técnicas modernas de aprendizaje profundo, utilizando el modelo YOLOv8 para la detección de personas y objetos, con algoritmos geométricos clásicos para el cálculo de proporciones métricas. A lo largo del documento se explica la fundamentación teórica de proyección en perspectiva, con el software desarrollado en Python y la fase experimental llevada a cabo para validar la precisión de las mediciones en distintos escenarios concluyendo con un análisis de los resultados y las lecciones aprendidas durante el desarrollo del sistema.

Palabras claves: YOLOv8, persona, humano, imágenes digitales, procesamiento de imagen.

1. Introducción

El procesamiento de imágenes digitales ha evolucionado rápidamente en las últimas décadas, convirtiéndose en una herramienta fundamental para extraer información métrica del mundo real a partir de representaciones bidimensionales. Dentro de este campo, la metrología visual, que es la capacidad de medir objetos físicos mediante cámaras, presenta desafíos importantes, principalmente debido a la pérdida de información de profundidad, las variaciones de escalas inherentes a la proyección en perspectiva y también por problemas de eficiencia de algunos modelos.

El presente trabajo tiene como objetivo el estudio e implementación de un sistema capaz de estimar la altura de una persona en imágenes digitales y por una webcam. Para resolver el problema de la escala desconocida, el método propuesto se basa en el uso de un objeto de referencia, en el caso del proyecto, un folio tamaño A4 presente en escena junto a una persona.

Este proyecto ha sido concebido como un trabajo de profundización, con una carga de 50 horas de dedicación por persona, y busca didáctica la integración de técnicas clásicas

de visión por computador con algoritmos modernos de aprendizaje profundo y de procesamiento de imagen. En concreto, se utiliza el modelo YOLOv8 para la detección robusta de personas y objetos de referencia, combinando con scripts genéricos para el cálculo de proporciones.

2. Planteamiento teórico

En el desarrollo del proyecto diversas técnicas de procesamiento de imagen fueron utilizadas.

2.1. Transformaciones de color (RGB → LAB)

La transformación de valores RGB dependientes del dispositivo al espacio perceptualmente uniforme LAB no es directa y requiere una etapa intermedia del espacio XYZ [4]. El proceso sigue el flujo estándar sRGB (linealización) → XYZ → LAB [1].

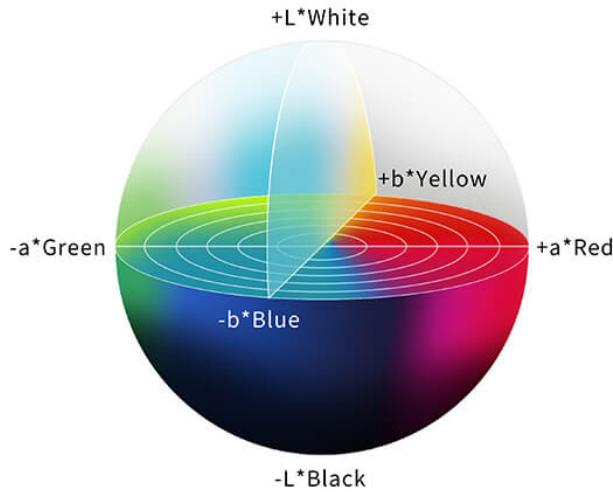


Figura 1: Espacio de colores de LAB. Fuente: Linshang technology y Kaizoudou

2.1.1. Linealización y Conversión sRGB → XYZ

Los valores sRGB de entrada pasan primero por una expansión de gamma (linealización) para eliminar la corrección gamma aplicada a los monitores. A continuación, estos valores lineales R_{lin} , G_{lin} , B_{lin} se convierten al espacio XYZ mediante una multiplicación matricial, asumiendo el iluminante estándar D65 [4]:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M \times \begin{bmatrix} R_{lin} \\ G_{lin} \\ B_{lin} \end{bmatrix} \quad (1)$$

2.1.2. Conversión XYZ → LAB

Los valores XYZ resultantes se normalizan con respecto al punto blanco de referencia (X_n, Y_n, Z_n) y se transforman para obtener las coordenadas L^* (luminosidad), a^* y b^*

(componentes cromáticos) utilizando la función de transformación no lineal $f(t)$ de LAB [4].

$$L^* = 116f(Y/Y_n) - 16, \quad a^* = 500[f(X/X_n) - f(Y/Y_n)], \quad b^* = 200[f(Y/Y_n) - f(Z/Z_n)] \quad (2)$$

La función $f(t)$ utiliza una raíz cúbica para valores superiores a un umbral ($\delta = 6/29$) y una función lineal para valores inferiores, garantizando la uniformidad perceptual [1].

2.2. Umbralización simple (inRange / threshold)

La umbralización es una técnica fundamental de segmentación que convierte una imagen en escala de grises a una imagen binaria (blanco y negro), separando los píxeles en dos grupos: los que están por encima de un valor de referencia (umbral) y aquellos que están por debajo [3]. La principal limitación es su sensibilidad a la iluminación regular y para mitigar eso se utilizan técnicas adaptativas como la Úmbralización Adaptativa el "Método de Otsu" [1].

Dada una imagen $f(x, y)$ y un umbral T , la imagen binaria resultante $g(x, y)$ se define como:

$$g(x, y) = \begin{cases} 255(\text{blanco}) & \text{si } f(x, y) > T \\ 0(\text{negr}) & \text{si } f(x, y) \leq T \end{cases} \quad (3)$$

2.2.1. Métodos Comunes

- **Threshold Simple:** Aplica un valor de corte global T a todos los píxeles. Si la iluminación no es uniforme, este método puede fallar, perdiendo detalles en áreas sombreadas o sobreexpuestas [3].

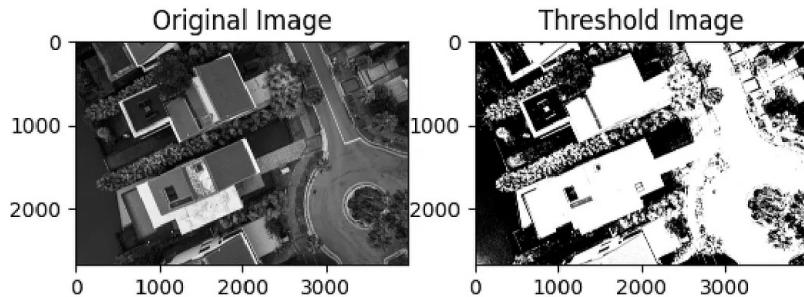


Figura 2: Método Threshold. Fuente: Ultralytics.

- **Rango de Color:** A diferencia del umbral simple que opera en un solo canal (intensidad), `inRange` permite segmentar basándose en un intervalo específico dentro de un espacio de color (ejemplo: HSV) [8].

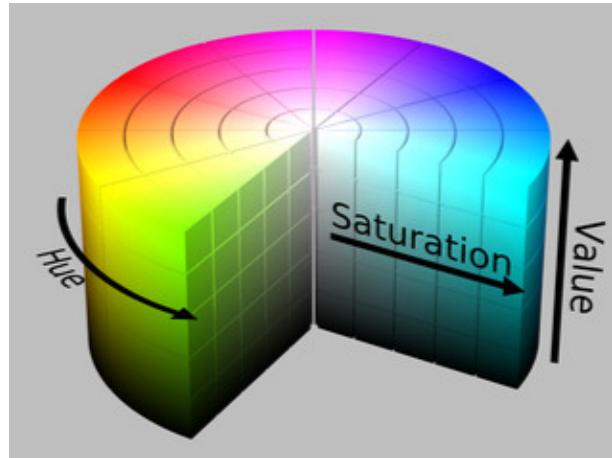


Figura 3: Rango de colores HSV. Fuente: Wikimedia Commons y OpenCV.

2.3. Operaciones morfológicas (dilatación)

La dilatación es una operación fundamental en el procesamiento morfológico de imágenes que .expande” las regiones blancas (primer plano) de una imagen binaria [5].

2.3.1. Concepto Matemático

Si A es la imagen binaria y B es el elemento estructurante, la dilatación de A por B (denotado $A \oplus B$) es el conjunto de todos los desplazamientos z tal cual la reflexión \hat{B} y A se superponen en al menos un píxel [6]:

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\} \quad (4)$$

En términos más simples para implementación: si al menos un píxel bajo el elemento estructurante (“kernel”) es blanco (1), el píxel central de la nueva imagen será blanco. Esto hace que las regiones blancas crezcan y los agujeros oscuros se reduzcan [5].

2.3.2. Efectos y Aplicaciones

- **Relleno de huecos:** Es muy eficaz para cerrar pequeños agujeros negros dentro de objetos blancos o reparar roturas en líneas discontinuas [18].

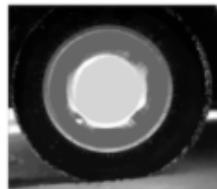


Figura 4: Relleno de huecos. Fuente: Mathworks.

- **Conexión de componentes:** Permite unir objetos que están muy cerca pero separados por una pequeña brecha (como caracteres de textos rotos) [6].

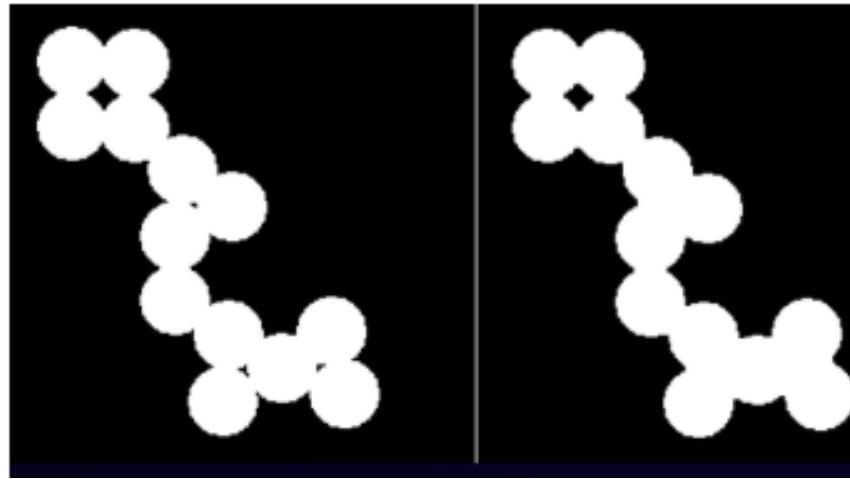


Figura 5: Conexión de componentes. Fuente: Mathworks.

- **Suavizados de contornos:** Tiende a suavizar las irregularidades hacia afuera, haciendo que los objetos sean más grandes y robustos [1].



Figura 6: Suavizados de contornos. Fuente: Mathworks.

2.3.3. Implementación Típica con OpenCV

Para la implantación se utiliza la función `cv2.dilate()` que desliza el kernel sobre la imagen [8]:

- **Kernel:** Define la forma y tamaño de la expansión (ej. un cuadrado de 3x3, 5x5). Un kernel más grande produce una dilatación más agresiva.
- **Iteraciones:** Aplicar la operación múltiples veces seguidas incrementa el efecto de engrosamiento.

2.4. Detectores de bordes (Canny)

El algoritmo de Canny es un detector multi-etapa para identificar bordes en imágenes con alta precisión y mínima sensibilidad al ruido [2]. Se aplica mediante `cv2.Canny(imagen, umbral_min, umbral_max)` [7].

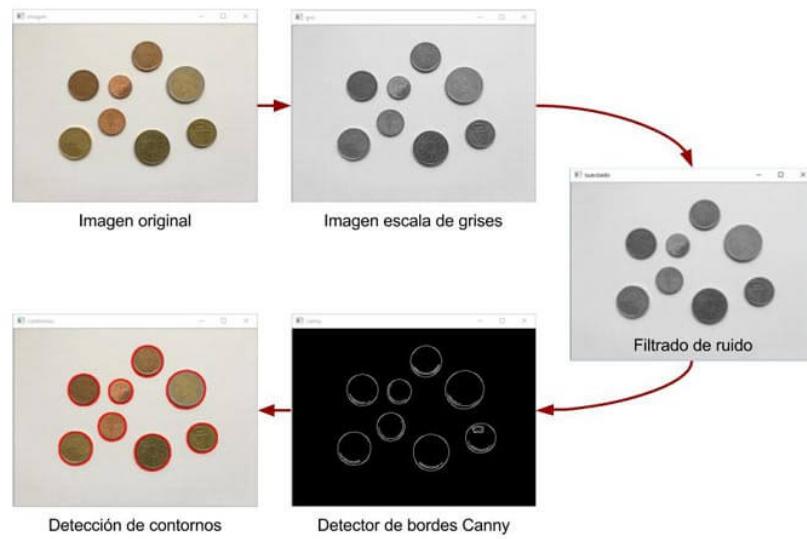


Figura 7: Fases del proceso de detección de bordes. Fuente: Programar Facil.

2.4.1. Proceso del Algoritmo

El método se ejecuta en cuatro pasos [2][7]:

- **Suavizado Gaussiano:** Reduce el ruido aplicando un filtro gaussiano [14].



Figura 8: Filtro gaussiano. Fuente: ResearchGate.

- **Cálculo del gradiente:** Detecta la intensidad y dirección de los cambios de píxeles (operador Sobel) [16].

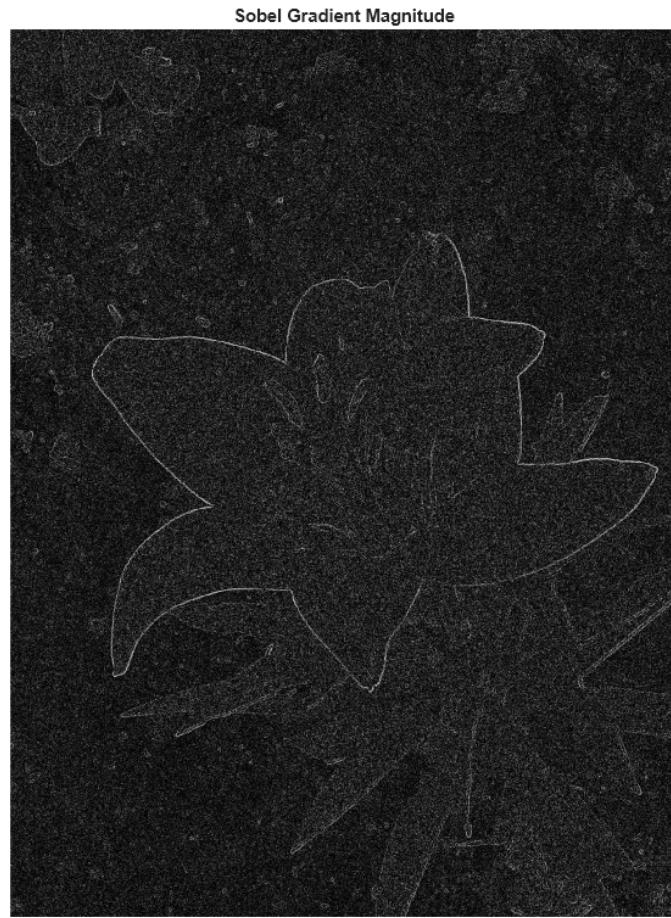


Figura 9: Operador Sobel. Fuente: MATLAB Help Center.

- **Supresión No-Máxima:** Adelgaza los bordes dejando solo los píxeles máximos locales [15].

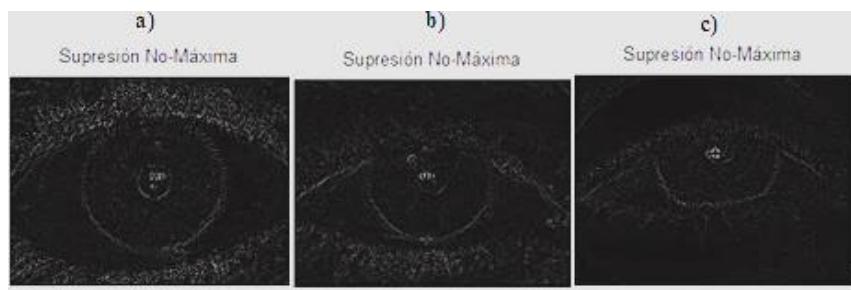


Figura 10: Supresión No-Máxima. Fuente: ResearchGate.

- **Umbralización por Histéresis:** Clasifica los píxeles como borde fuerte ($> \text{umbral_max}$), no-borde ($< \text{umbral_min}$) o borde débil (entre umbrales, aceptado sólo si conectado a un borde fuerte) [17].

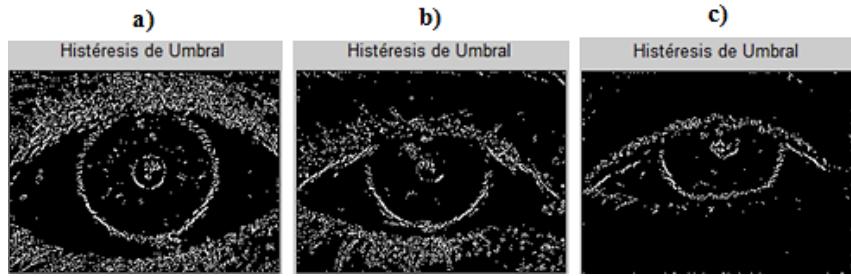


Figura 11: Umbralización por Histéresis. Fuente: ResearchGate.

2.5. Contornos y bounding boxes rotados (minAreaRect)

La función `cv2.minAreaRect(cnt)` permite encerrar un contorno en el rectángulo más pequeño posible, ajustando su ángulo de rotación a la orientación del objeto [8]. Esto se diferencia del *bounding box* estándar (`boundingRect`), que siempre es paralelo a los ejes y, por lo tanto, ocupa más área si el objeto está inclinado [12].

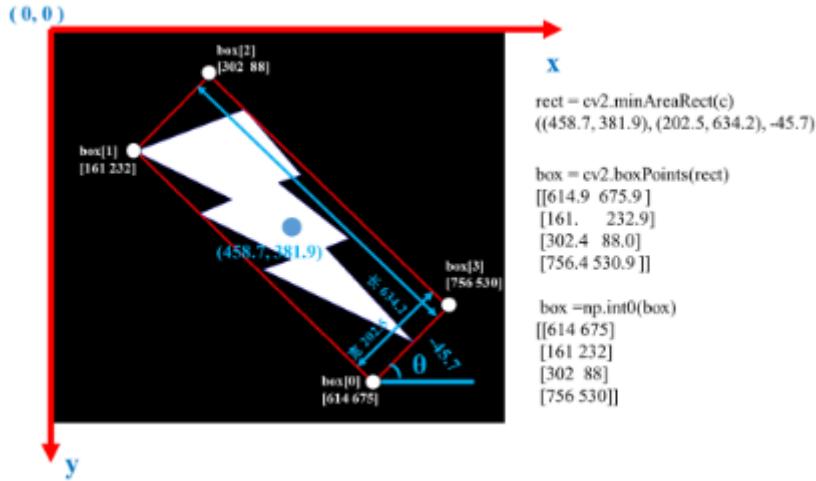


Figura 12: Función `cv2.minAreaRect`. Fuente: Jishuzhan

2.5.1. Estructura Técnica (Box2D)

- **Centro** (cx, cy): Coordenadas flotantes del centro geométrico del rectángulo [9].
- **Dimensiones** (w, h): Ancho y alto del rectángulo ajustado (el orden puede variar según la rotación) [8].
- **Ángulo**: Valor de inclinación en grados. Este valor oscila básicamente en el rango 0-90, indicando la rotación en sentido horario del eje horizontal [9].

2.5.2. Proceso de Dibujado

Dado que Box2D no define esquinas explícitas, se debe transformar antes de dibujar [9]:

- **Obtener Vértices:** Usar `cv2.boxPoints(rect)` para convertir la estructura Box2D en las 4 coordenadas de las esquinas.

- **Conversión Numérica:** Convertir los puntos flotantes a enteros (`np.int0`).
- **Renderizado:** Dibujar el polígono resultante con `cv2.drawContours` ya que la función `rectangle` no soporta rotación.

2.6. Análisis Geométrico (ratio A4)

Este análisis se utiliza en visión por computador para validar si una forma detectada corresponde a una hoja de papel estándar, basándose en sus proporciones físicas invariables [10].

2.6.1. Propiedad Geométrica (ISO 216)

El estándar internacional ISO 216 define que todos los formatos de la serie A (incluyendo el A4) mantienen una relación de aspecto única de $\sqrt{2}$ [11]. Esto significa que el lado largo siempre es 1,41 veces mayor que el lado corto, independientemente de la escala o resolución de la imagen [10].

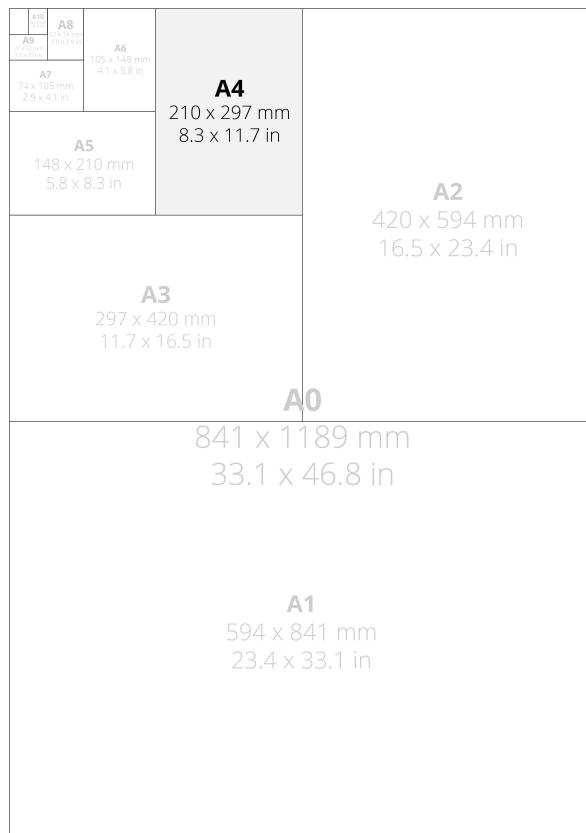


Figura 13: Dimensiones de todos los formatos de la serie A. Fuente: Hall of Print.

2.6.2. Aplicación: Filtrado de Contornos

Para discriminar entre ruido y un posible documento, se calcula la relación de aspecto (AR) del rectángulo mínimo que encierra el contorno (`minAreaRect`) [8]:

$$AR = \frac{\max(\text{ancho}, \text{alto})}{\min(\text{ancho}, \text{alto})} \quad (5)$$

Un contorno se acepta como candidato válido solo si su AR se aproxima a 1.41, dentro de un margen de tolerancia ϵ (típicamente ± 0.2) para compensar distorsiones de perspectiva o errores de segmentación [11].

$$|AR - 1.41| < \epsilon \quad (6)$$

2.7. YOLOv8

El YOLOv8 es un modelo de detección de objetos que se caracteriza por combinar alta precisión con velocidad de procesamiento en tiempo real, siendo adecuado para uso en aplicaciones industriales, robótica, vigilancia y vehículos autónomos. La Figura 14 muestra la comparación de YOLOv8 con las versiones anteriores [13].

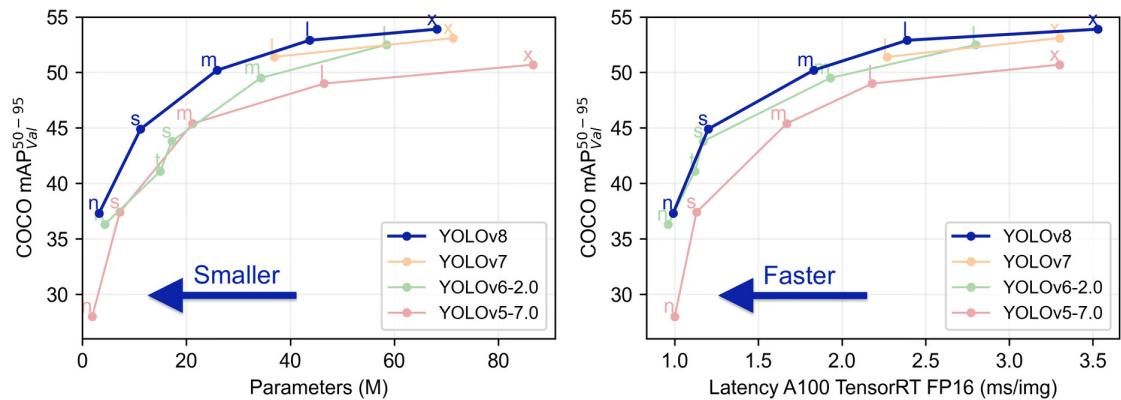


Figura 14: Comparación entre las versiones de YOLO. Fuente: Ultralytics

2.7.1. Características de YOLOv8

El YOLOv8 posee las siguientes características [13]:

- **Modelo Anchor-Free:** Abandona las cajas de anclaje predefinidas (*anchor boxes*) para predecir directamente el centro y las dimensiones del objeto, lo que simplifica la arquitectura y mejora la capacidad de generalización del modelo.
- **Utilización del módulo Backbone C2f:** Sustituye el antiguo módulo C3 por el C2f, que combina conexiones residuales con operaciones de convolución para enriquecer el flujo de gradientes y la extracción de características manteniendo la eficiencia computacional.
- **Cabezal Desacoplado:** Divide la etapa final de detección en dos ramas independientes, una para clasificación y otra para regresión de la caja, resolviendo conflictos de optimización entre estas tareas y mejorando la precisión global.
- **Task-Aligned Assigner (TAL):** Sustituye el IoU estático por una asignación dinámica de etiquetas durante el entrenamiento, seleccionando muestras positivas en función del alineamiento entre la puntuación de la clase y la precisión de la localización.

- **Losses Refinadas:** Utiliza CIoU (para superposición y consistencia geométrica) combinada con DFL (*Distribution Focal Loss*) para refinar los bordes de las cajas, permitiendo ajustes de localización más precisos incluso en objetos con formas o contornos ambiguos.

2.7.2. Ventajas del YOLOv8

Las ventajas de YOLOv8 son [13]:

- **Precisión Superior:** Actualmente ofrece uno de los mejores equilibrios entre precisión (mAP) y velocidad de inferencia en el mercado.
- **Anchor-Free:** Elimina la necesidad de calcular o ajustar anclajes manualmente para *datasets* específicos, haciendo que el modelo sea más generalista.
- **Versatilidad:** Soporta nativamente detección, segmentación, *pose estimation* y clasificación con la misma API.
- **Ecosistema:** La integración con el paquete Python de Ultralytics es extremadamente robusta, facilitando el despliegue y el entrenamiento vía CLI o código.

2.7.3. Inconvenientes del YOLOv8

Los inconvenientes de YOLOv8 son [13]:

- **Coste de Entrenamiento:** Debido a la complejidad extra de las *losses* y al *mosaic augmentation* pesado, el entrenamiento puede ser ligeramente más lento o exigir más memoria VRAM comparado con modelos más simples como YOLOv5n.
- **Objetos Pequeños:** Aunque mejor que sus predecesores, todavía enfrenta desafíos con objetos muy pequeños en resoluciones estándar (640x640), una limitación común en arquitecturas de una sola etapa (*One-Stage Detectors*).
- **Complejidad del Código:** La abstracción de la librería ultralytics facilita el uso, pero puede dificultar la personalización profunda de la arquitectura para investigadores que deseen alterar capas internas manualmente.

2.8. YOLOv8-SEGMENTACIÓN (YOLOv8-Seg)

La variante YOLOv8-Seg es la versión del modelo YOLOv8 diseñada específicamente para realizar segmentación de instancias: un proceso que asigna una máscara de píxeles a cada objeto detectado, en lugar de limitarse a una caja delimitadora (*bounding box*) rectangular. Esta capacidad resulta fundamental en aplicaciones donde la forma exacta del objeto aporta información crítica, como en análisis biométrico, conducción autónoma, medicina o medición precisa de figuras humanas (caso de este proyecto) [13].

Mientras que YOLOv8 estándar retorna únicamente coordenadas de caja y una probabilidad de clase, YOLOv8-Seg produce simultáneamente tres tipos de salida:

1. Bounding box
2. Clase predicha (confidence)

3. Máscara de segmentación, representada como un conjunto de protos combinados linealmente mediante coeficientes Learnables.

La Figura 15 ilustra la diferencia conceptual entre detección por bounding box y segmentación por máscaras.



Figura 15: YOLOv8 Segmentación. Fuente: Ultralytics

2.8.1. Arquitectura interna de YOLOv8-Seg

YOLOv8-Seg comparte la misma arquitectura general del YOLOv8 base (Backbone C2f + Neck FPN/PAN + Head desacoplado), pero introduce un *branch* adicional dedicado a la generación de máscaras, inspirado en modelos como YOLACT y Mask-R-CNN [13].

Su estructura interna puede resumirse en tres módulos principales:

a) Backbone (C2f + CNN convolucional) Extrae las características profundas de la imagen, generando mapas multiescala. Incluye:

- Convoluciones 3×3 aceleradas
- Bloques residuales ligeros
- C2f (Cross-Stage-Partial Fusion), que mejora la propagación del gradiente

b) Head de detección (Bounding Box + Clase) Produce:

- Coordenadas x, y, w, h del objeto
- Puntaje de clase (softmax o sigmoid depende de configuración)

c) Head de segmentación (Máscara) Este es el componente adicional respecto a YOLOv8 clásico. Consta de dos salidas:

- **Mask Coefficients (Coeficientes de Máscara):** Vectores de tamaño reducido que indican cómo combinar un conjunto de prototipos globales.
- **Mask Prototypes (Máscaras Base):** Producidos por una rama convolucional que genera entre 32 y 64 "protos". Cada máscara final se calcula por combinación lineal de estos protos. La máscara resultante se recorta a la bounding box para optimizar el tiempo de dibujado.

2.8.2. Flujo de inferencia en YOLOv8-Seg

El proceso de predicción ocurre en tres etapas:

1. **Inferencia base (Backbone + Neck):** Extrae características de la imagen.

2. **Predicción simultánea de:**

- Bounding boxes
- Clases
- Mask coefficients

3. **Reconstrucción de la máscara:** Se combinan los protos con los coeficientes para generar una máscara por instancia. La reconstrucción de máscara tiene una complejidad muy baja debido a la multiplicación eficiente de protos, por tanto, YOLOv8-Seg mantiene rendimiento en tiempo real.

2.8.3. Ventajas de YOLOv8-Seg respecto a Bounding Boxes

1. **Precisión espacial real** El bounding box puede dejar márgenes vacíos, especialmente en:

- Zona de la cabeza
- Hombros
- Pies
- Poses inclinadas

La máscara en cambio coincide con la silueta real del sujeto.

2. **Mediciones métricas más fiables** Para un sistema como este proyecto, donde se debe medir el punto más alto exacto de la cabeza y el punto más bajo exacto del pie, y estimar altura en píxeles sin márgenes falsos, YOLOv8-Seg elimina la incertidumbre generada por la caja rectangular.

3. **Mejor rendimiento en poses no verticales** La bounding box se expande si el sujeto levanta los brazos o ladea la cabeza. La máscara solo marca la región ocupada por el cuerpo, sin exageraciones.

4. **Contornos aptos para procesado geométrico** La máscara permite aplicar:

- Detección de puntos extremos
- Convex hull
- Recortes personalizados
- Extracción del esqueleto (skeletonization)
- Análisis de profundidad aproximada por área

2.8.4. Limitaciones de YOLOv8-Seg

Aunque notablemente más precisa, YOLOv8-Seg también presenta desafíos:

- Requiere más memoria VRAM que la versión de detección clásica.
- Las máscaras pueden perder detalle en casos de iluminación extrema.
- El rendimiento cae si la cámara produce ruido o motion-blur.
- La reconstrucción de protos puede suavizar bordes muy finos (ej. dedos).

Aun así, su relación coste-rendimiento continúa siendo ideal para aplicaciones en tiempo real.

2.8.5. Justificación del uso de YOLOv8-Seg en este proyecto

El uso de YOLOv8-Seg se justifica por:

1. **Necesidad de puntos extremos reales (top/bottom) del cuerpo:** La altura debe ser calculada desde la cabeza hasta la punta del pie, no hasta el borde artificial de una caja.
2. **Precisión crítica en mediciones métricas:** Un error de 3–5 píxeles en bounding box supone varios centímetros en la altura final.
3. **Robustez ante poses naturales del usuario:** La segmentación sigue el contorno incluso si la persona inclina la cabeza, mueve los brazos o flexiona rodillas mínimamente.
4. **Coherencia con las correcciones geométricas del Seguimiento 4:** Al combinar profundidad Z, longitud del pie L, ratio 1.414 del folio A4 y eliminación de la suela, se necesitaba una arquitectura capaz de proporcionar datos limpios de la silueta.

Por todo ello, YOLOv8-Seg se convierte en la opción óptima para la etapa final del sistema.

3. Implementación

Para el desarrollo de la solución propuesta se ha utilizado el lenguaje de programación Python 3, elegido por su amplia compatibilidad con bibliotecas de computación científica y visión artificial. La gestión de dependencias del proyecto se ha realizado mediante un archivo de requisitos estándar (`requirements.txt`), siendo las librerías fundamentales:

- **Ultralytics YOLOv8:** Utilizada como motor de inferencia de redes neuronales para la detección de objetos [13].
- **OpenCV (opencv-python):** Empleada para todo el preprocesamiento de imágenes, operaciones de visión artificial clásica (transformaciones de color, detección de bordes) y visualización de resultados [8].
- **NumPy:** Necesaria para las operaciones matriciales y el manejo de arrays numéricos que representan las imágenes y coordenadas.

El código fuente se ha organizado en diferentes scripts modulares que abordan el problema de forma secuencial, comenzando por validaciones en imágenes estáticas hasta llegar al procesamiento en tiempo real. A continuación, se detalla la lógica de cada módulo.

Para la utilización del modelo YOLOv8 fue hecho un *fork* del repositorio de GitHub **YOLOv8-HumanDetection** donde están los archivos base para hacer la experimentación [19]. Después fue creado un repositorio para almacenar el nuestro proyecto, llamado **Human-Height-Detection-Using-Reference** [20].

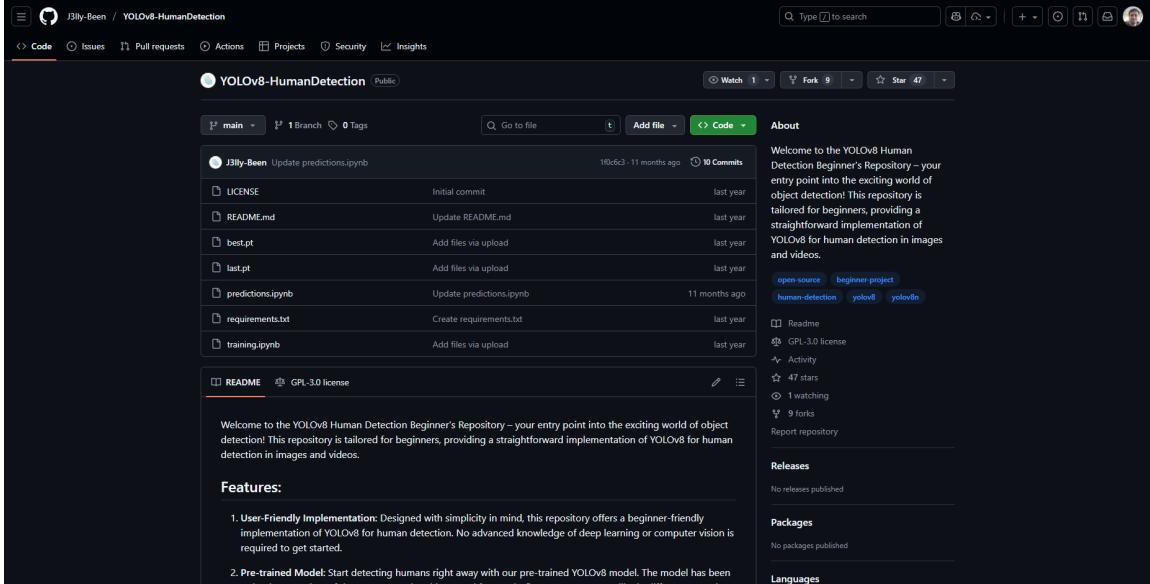


Figura 16: Repositorio “YOLOv8-HumanDetection”. Fuente: Github

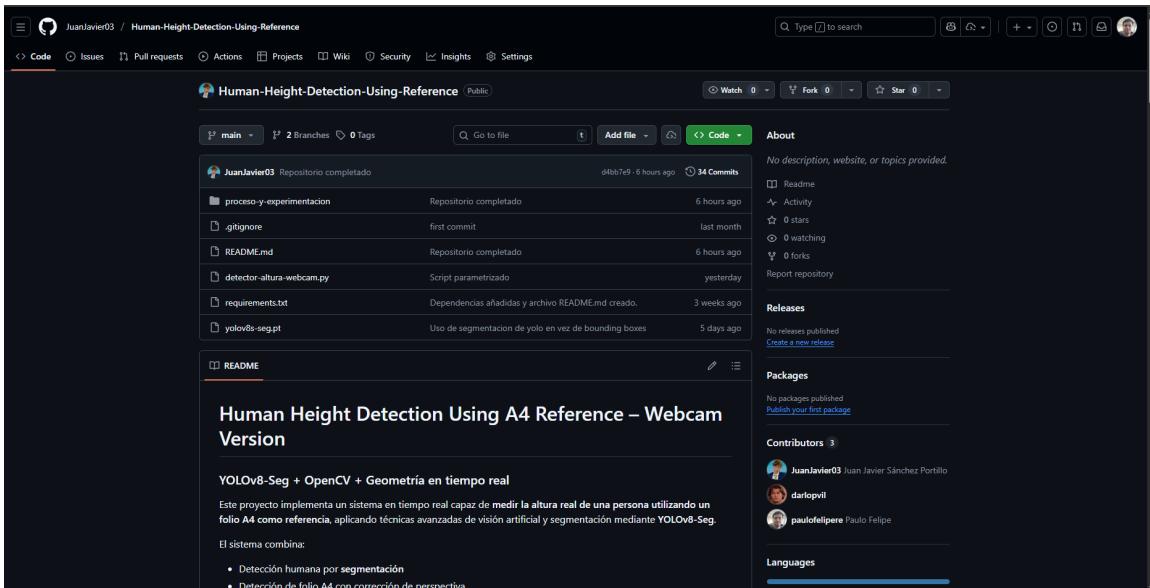


Figura 17: Repositorio “Human-Height-Detection-Using-Reference”. Fuente: Github

3.1. Detección de Humanos

Descripción de la solución: El problema a abordar (detección de la altura estimada de un humano) comenzó validando el concepto con imágenes estáticas antes de pasar al

flujo de vídeo en tiempo real.

Para la fase de detección, se tomó como base el repositorio *open-source* “YOLOv8-HumanDetection” [19]. De este repositorio no se ha bifurcado el código completo, sino que se ha adaptado su recurso más valioso: el modelo pre-entrenado. Específicamente, se utiliza el archivo de pesos `best.pt`, el cual contiene una red neuronal convolucional ya entrenada para distinguir figuras humanas con alta precisión.

Aportación original y lógica del script: La implementación original desarrollada para este proyecto reside en el script `human-detector.py`. A diferencia de los ejemplos genéricos que solo muestran la detección en pantalla, este módulo fue diseñado con un objetivo específico para la siguiente fase del proyecto: la extracción de datos.

El flujo lógico del script es el siguiente:

1. **Entrada:** Recibe una imagen estática.
2. **Inferencia:** Aplica el modelo `best.pt` con un umbral de confianza (*confidence threshold*) de 0.55 para evitar falsos positivos [13].
3. **Procesamiento:** Extrae las coordenadas de las “bounding boxes” (cajas delimitadoras) en formato (x_1, y_1, x_2, y_2) .
4. **Salida:** Devuelve estas coordenadas como una estructura de datos (lista de tuplas) y visualiza el resultado usando OpenCV para validación visual [8].

Detalles del modelo base: La elección del modelo `best.pt` sobre otros pesos disponibles (como `last.pt`) se debe a su rendimiento optimizado. Según la documentación original del repositorio, este modelo fue entrenado utilizando datasets importados y gestionados desde Roboflow, garantizando versatilidad en distintos escenarios de iluminación y fondo [19].

Las especificaciones técnicas del dataset de entrenamiento incluyen:

- **Distribución:** Un split del 88% para entrenamiento (4200 imágenes), 8% para validación y 4% para test.
- **Preprocesamiento:** Redimensionado (*stretch*) a 640x640 píxeles y auto-ajuste de contraste.
- **Aumentación de datos:** Para robustecer el modelo, se aplicaron técnicas como rotación ($\pm 15^\circ$), variaciones de saturación y brillo, y desenfoque (*blur*) de hasta 2.5px.



Figura 18: `human-detector.py`: Comparación entre la imagen original y la detección de un humano como salida. Fuente: Ultralytics



Figura 19: `human-detector.py`: Comparación entre la imagen original y la detección de varios humanos. Fuente: Ultralytics

Comando de ejecución:

```
python human-detector.py foto_persona.jpg
```

3.2. Detección del Referente (Folio A4)

Enfoque y Tecnologías: Una vez aislada la figura humana mediante el módulo anterior, el siguiente desafío técnico consiste en encontrar un elemento de referencia métrica dentro de esa región. Para esta tarea, a diferencia del uso de Deep Learning en la etapa previa, se ha optado por un enfoque de *Visión Artificial Clásica* utilizando exclusivamente la librería *OpenCV* y *NumPy* [1].

Diseño y Originalidad: El script desarrollado para esta función, `folio-detector-pasos.py`, es una implementación original diseñada específicamente para este proyecto. Se concibió como una herramienta de depuración visual (“paso a paso”) que permite descomponer el problema en capas lógicas, aplicando secuencialmente transformaciones de imagen basadas en el temario de la asignatura (transformaciones de color, operaciones morfológicas y análisis de contornos).

Lógica del algoritmo: La detección del folio no se realiza sobre la imagen completa, sino sobre la Región de Interés (ROI) recortada por YOLO. El proceso de decisión algorítmica sigue estos pasos:

1. **Transformación del Espacio de Color (BGR a LAB):** Las imágenes digitales suelen verse afectadas por cambios de iluminación. Por ello, el primer paso es convertir la imagen del espacio BGR al espacio *CIELAB (LAB)* [4].

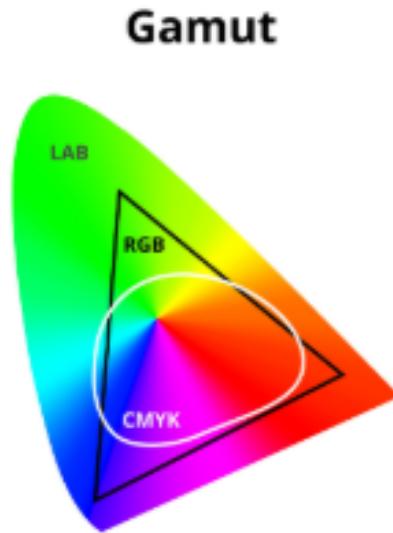


Figura 20: Diferentes escenarios de color. Fuente: Getty Images

Esta decisión de diseño es crítica: en el espacio RGB/BGR el color blanco es una mezcla de tres canales altos (255, 255, 255). En cambio, en LAB, sepáramos la **Luminosidad** (Canal L) de la información de color (Canales A y B). Esto nos permite buscar el folio basándonos en su propiedad física: es un objeto “sin color” (neutro en A y B) y “muy brillante” (alto en L), independientemente de si la luz es cálida o fría.

2. **Segmentación y Umbralización:** Se generan máscaras binarias combinadas. Se busca la intersección de píxeles que cumplen dos condiciones:
 - **Cromaticidad neutra:** Valores centrales en los canales A y B (lo que elimina colores vivos como la ropa o el fondo).
 - **Alta Luminosidad:** Valores altos en el canal L.
3. **Procesamiento Morfológico:** Una vez obtenida la “mancha” candidata a ser folio, se aplica un detector de bordes *Canny* [7]. Para asegurar que el contorno del papel sea continuo y robusto frente al ruido, se aplica una *dilatación* sobre los bordes detectados [5].
4. **Análisis Geométrico (Discriminación por Forma):** El paso final es puramente geométrico. El algoritmo extrae los contornos y calcula el rectángulo de área mínima rotado (`minAreaRect`) para cada candidato. Se descartan aquellos que no cumplen con el *Ratio de Aspecto* del estándar DIN A4. Dado que un A4 mide 210mm ×

$297mm$, su relación es de $\frac{297}{210} \approx 1,414$ [11]. El algoritmo solo acepta rectángulos cuya proporción esté dentro de un margen de tolerancia cercano a este valor matemático, descartando así otros objetos rectangulares o ruido espurio [10].

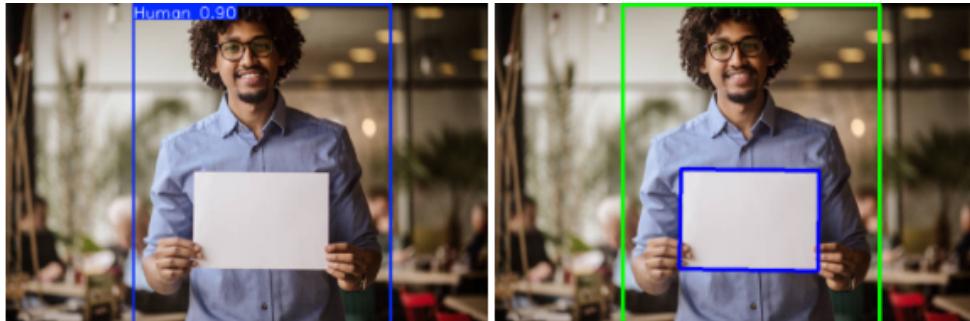


Figura 21: folio-detector-pasos.py: Imagen de entrada del Script 3.1 y de salida la detección del folio. Fuente: Ultralytics

Comando de ejecución:

```
python folio-detector-pasos.py imagen.jpg x1 y1 x2 y2
```

3.3. Implementación Final: Integración y Estimación Métrica

Objetivo y Diseño del Sistema: El script final de esta etapa, `detector-altura.py`, representa la culminación del procesamiento de imágenes estáticas. Su función es orquestar los dos módulos anteriores en un único *pipeline* de ejecución secuencial que automatiza el proceso completo: desde la carga de la imagen hasta la estimación numérica de la altura.

Lógica de Integración (Pipeline): El flujo de trabajo implementado sigue una estructura jerárquica estricta para optimizar recursos y evitar falsos positivos:

1. **Detección Primaria (IA):** Se invoca al modelo *YOLOv8* (`best.pt`) para localizar al sujeto. Si no se detecta presencia humana con una confianza superior al 55 %, el script detiene su ejecución inmediatamente, ahorrando cómputo [13].
2. **Extracción de ROI:** Se recorta la región de la imagen que contiene a la persona. Esto es crucial: el algoritmo de búsqueda del folio solo se ejecuta dentro de este recorte, no en la imagen completa. Esto reduce drásticamente el ruido y evita que se detecten objetos rectangulares blancos en el fondo de la habitación.
3. **Detección Secundaria (Visión Clásica):** Se aplica el algoritmo de color (LAB) y geometría (Ratio A4) descrito en el apartado 3.2 sobre el recorte.
4. **Transformación de Coordenadas:** Una vez localizado el folio dentro del recorte, el script realiza una transformación matemática para trasladar las *coordenadas locales* (relativas al recorte) a *coordenadas globales* (relativas a la imagen original), permitiendo dibujar el resultado final sobre la foto completa [8].

Fundamento Matemático de la Medición: Para la obtención del dato final, el sistema asume que el folio se encuentra en el mismo plano de profundidad que el sujeto (o sostenido por él). La estimación se realiza mediante una relación de proporcionalidad

directa (Regla de Tres), utilizando la dimensión vertical física del estándar A4 como constante de calibración [11].

La fórmula aplicada es:

$$Altura_{cm} = \frac{Altura_{pixel_sujeto}}{Largo_{pixel_folio}} \times 29,7 \quad (7)$$

Donde H_{pixel_folio} se obtiene del lado más largo del rectángulo rotado (`minAreaRect`), lo que permite medir correctamente el folio incluso si este está ligeramente inclinado en las manos del usuario [9].

Resultado: El sistema devuelve una visualización aumentada donde conviven la *Bounding Box* de la red neuronal (verde) y el contorno procesado por OpenCV (azul), junto con la estimación numérica impresa en consola.

Comando de ejecución:

```
python detector-altura.py foto.jpg
```

3.4. Transición al Tiempo Real: Validación Generalista

Justificación de la Prueba de Concepto: Antes de trasladar la lógica de medición métrica (desarrollada en los apartados anteriores sobre imágenes estáticas) a un entorno de vídeo en tiempo real, se consideró necesario realizar una validación técnica intermedia. El procesamiento de vídeo exige un rendimiento computacional mucho mayor que el de imágenes fijas, ya que el sistema debe procesar, inferir y dibujar resultados al menos a 15-30 fotogramas por segundo (FPS) para que la experiencia sea fluida.

Implementación del Script de Prueba: Para este fin se desarrolló el script `object-detection.py`. A diferencia de los módulos anteriores, este script no utiliza el modelo especializado en humanos (`best.pt`), sino que implementa el modelo oficial estándar `yolov8n.pt` (versión *Nano*) [13].

Esta elección tiene dos motivos:

1. **Evaluación de Velocidad:** El modelo *Nano* es el más ligero de la familia YOLO, optimizado para inferencia rápida en dispositivos con recursos limitados (como un portátil sin GPU dedicada). Esto nos permite verificar si el hardware disponible soporta la carga de trabajo.
2. **Capacidades de la Librería:** Al usar el modelo pre-entrenado con el dataset COCO, el sistema es capaz de detectar 80 clases de objetos distintos (personas, sillas, botellas, móviles, etc.). Esto sirvió para comprobar la robustez de la detección ante fondos complejos y en movimiento.

Aportación Funcional (Interfaz de Usuario): Además de la detección, en este script se introdujo por primera vez la interacción con el usuario mediante *OpenCV GUI*. Se programó una función de escucha de eventos del ratón (`cv2.setMouseCallback`) para implementar un botón virtual de “STOP” directamente sobre el *feed* de vídeo. Este mecanismo de control es esencial para gestionar el ciclo de vida de la aplicación de forma segura, evitando cierres forzados del proceso.

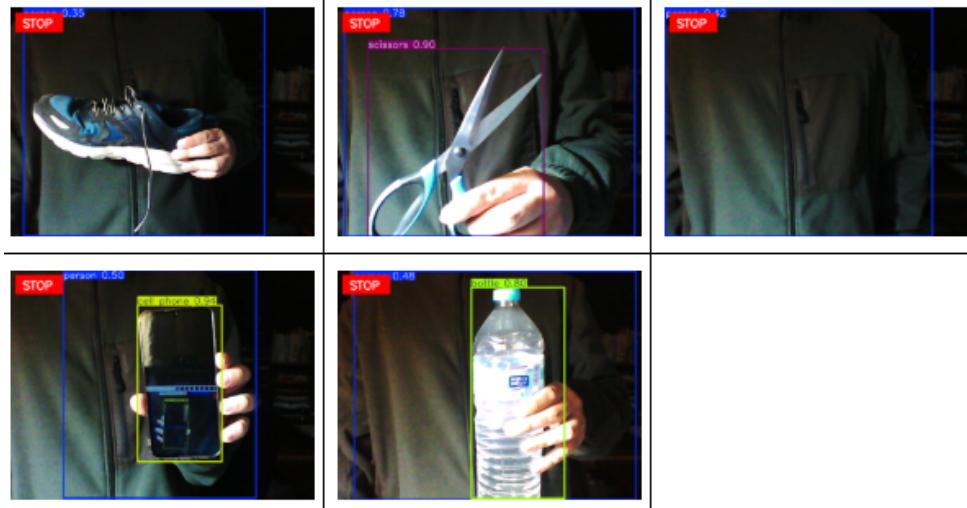


Figura 22: `object-detection.py`: Pruebas de detección de varios objetos (incluyendo ser humano de fondo). Fuente: Darío López

Comando de ejecución:

```
python object-detection.py
```

3.5. Detección Especializada en Tiempo Real

Fusión de Componentes y Diseño: Una vez validada la capacidad del hardware para ejecutar inferencia en tiempo real y tras haber caracterizado el modelo especializado (`best.pt`) en imágenes estáticas, se desarrolló el módulo `human-detector-webcam.py`.

Este script representa la convergencia de las dos etapas anteriores. Su diseño técnico consiste en trasladar la lógica de filtrado estricta, definida en la fase estática, al bucle de captura de vídeo en vivo.

Tecnologías y Adaptación: Se hace uso nuevamente de las librerías *OpenCV* (gestión del flujo de entrada/salida) y *Ultralytics* (motor de inferencia) [8]. La diferencia fundamental respecto a la prueba de concepto anterior (apartado 3.4) reside en el modelo neuronal cargado:

- Se descarta el modelo genérico (`yolov8n.pt`).
- Se carga el modelo adaptado `best.pt`, proveniente del repositorio externo *YOLOV8-HumanDetection* [19].

Lógica del Algoritmo (Frame a Frame): El reto en este módulo es mantener la consistencia de los datos en movimiento. Para ello, el script implementa un bucle de procesamiento continuo donde cada fotograma (*frame*) es tratado como una imagen independiente:

1. **Configuración de Captura:** Se fuerza una resolución de entrada de 1280×720 píxeles para garantizar que el sujeto tenga suficiente definición incluso si se aleja de la cámara.
2. **Inferencia con Parámetros Heredados:** Se aplica la detección utilizando exactamente los mismos hiperparámetros de confianza (`conf=0.55`) y superposición (`iou=0.7`) que se determinaron como óptimos en la fase de imágenes estáticas. Esto asegura que la calidad de detección sea consistente con los experimentos iniciales.

3. **Salida Dual de Datos:** El sistema no solo dibuja las cajas en la ventana de visualización (*feedback* visual para el usuario), sino que extrae e imprime las coordenadas (x_1, y_1) y (x_2, y_2) por la consola en tiempo real.

Parte Original: Aunque el modelo neuronal es externo, la implementación del bucle de control, la extracción de coordenadas y el contador de personas en pantalla (implementado mediante `cv2.putText`) son aportaciones originales desarrolladas para monitorizar el rendimiento del sistema antes de la integración final.

Comando de ejecución:

```
python human-detector-webcam.py
```

3.6. Iteración Experimental: Expansión del ROI

Antecedentes y Problemática (Seguimiento 3): Tras las pruebas iniciales con la primera versión del sistema, se detectaron disparidades inaceptables en los resultados dependiendo de la fisionomía del usuario. Mientras que con el sujeto *Paulo* la medición era precisa y con *Javier* el error se mantenía en un margen de $\pm 3\%$, con el sujeto *Darío* el sistema colapsaba, arrojando alturas erróneas que no superaban los 1,64 m (muy por debajo de su altura real).

En la sesión de control (“Seguimiento 3”), y tras consultar con la docente, se identificaron las causas y se propusieron tres vías de solución:

1. **Error de Profundidad:** Al adelantar el folio hacia la cámara, este se ve más grande, lo que por la regla de tres hace que el humano parezca más bajo.
2. **Error de Postura:** Se sugirió cambiar la forma de sujetar el folio (apoyado en las piernas e inclinándose) para igualar planos.
3. **Estrategia de Búsqueda Externa:** Para solucionar el desajuste específico en las medidas de *Darío*, se propuso modificar la lógica de detección: en lugar de buscar el folio únicamente dentro del recuadro de la persona, se debía intentar detectar el referente **fuerza del área de la *bounding box*** (específicamente en los laterales).

Implementación de la Solución (Script `detector-altura-webcam.py`): Este script corresponde a la implementación técnica de la **tercera propuesta** (evitar el recorte), desarrollada por el miembro del equipo *Javier* y posteriormente modificada por *Darío*.

La lógica principal añadida fue la **Expansión de ROI Horizontal**. El algoritmo modificaba las coordenadas entregadas por YOLO para ampliar artificialmente el área de búsqueda un 40% a cada lado del sujeto, intentando capturar el folio si este se encontraba en la periferia lateral.

Adicionalmente, para combatir el ruido en las mediciones, se introdujo en esta versión el cálculo de la **mediana estadística** (usando una cola de 5 muestras), buscando estabilizar los saltos numéricos en la predicción de altura.

Evaluación de Resultados y Limitaciones: Si bien esta versión consiguió solucionar teóricamente el problema geométrico (el folio ya entraba dentro de la imagen recortada), las pruebas de campo revelaron una **baja tasa de detección**. Al ampliar el área de búsqueda (`roi_expandido`), se introdujo más “ruido” visual en la imagen, y el algoritmo de visión artificial clásica (basado en umbrales de color y bordes) comenzó a fallar, siendo incapaz de encontrar el folio la mayoría de las veces.

Esta limitación nos obligó a pivotar hacia una nueva solución (“Seguimiento 4”), donde se integraría un nuevo modelo y se refinarián las propuestas restantes.

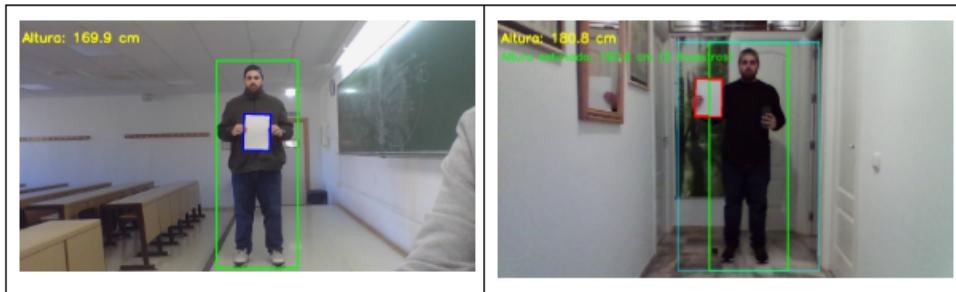


Figura 23: `detector-altura-webcam.py`: A la izquierda, una captura de la errónea estimación de la altura del sujeto Darío tras el Seguimiento 3. A la derecha, captura con solo 2 cms de error tras incorporar la estrategia de ampliación del ROI del folio, de cara al Seguimiento 4. Fuente: Darío López

Comando de ejecución: `python detector-altura-webcam.py`

3.7. Refinamiento del Sistema: Geometría, Segmentación y Nuevos Parámetros (Seguimiento 4)

Tras los resultados obtenidos en el Seguimiento 3, esta fase del proyecto se centra en una reconstrucción profunda del enfoque empleado hasta el momento. El objetivo principal ha sido eliminar las limitaciones observadas en la medición del sujeto, tanto desde el punto de vista geométrico como desde el propio proceso de detección del humano y del folio. Para ello, se han introducido nuevas variables, modificaciones matemáticas en la estimación de la altura y una sustitución estratégica del modelo neuronal utilizado en tiempo real.

A continuación, se describen de forma detallada las mejoras implementadas en el script `detector-altura-webcam.py`, que constituye la base de la versión actual del sistema.

3.7.1. Sustitución del modelo de detección: incorporación de YOLOv8-SEG

Una de las mejoras más significativas del Seguimiento 4 ha sido la transición desde el modelo `best.pt` (detección clásica basada en *bounding boxes*) hacia **YOLOv8-SEG**, un modelo que aporta un tipo de salida diferente: la máscara de segmentación [13].

Este cambio introduce dos ventajas críticas:

1. **Precisión en la silueta:** La máscara permite extraer el contorno exacto del sujeto, evitando el margen superior que YOLO introduce por defecto en sus bounding boxes.
2. **Altura basada en puntos reales:** A partir de la máscara se localizan directamente el punto más alto (cabeza real) y el punto más bajo (zona del pie), lo que genera una bounding box “propia”, ajustada al cuerpo y no a la predicción del modelo.

Gracias a esta modificación, la altura en píxeles deja de depender de las aproximaciones del detector y pasa a depender únicamente de la figura segmentada.

3.7.2. Corrección geométrica mediante profundidad Z y longitud del pie L

Durante las pruebas se observó un error sistemático: la zona del pie suele estar más cerca de la cámara que el resto del cuerpo, de modo que la proyección hace que la altura total parezca más pequeña.

Este fenómeno de perspectiva se corrige introduciendo dos nuevas variables:

- **Z:** Distancia estimada entre la persona y la cámara.
- **L:** Longitud aproximada del pie (distancia entre talón y punta).

El sistema ajusta la altura medida compensando la diferencia de profundidad entre la punta del pie y el cuerpo, reduciendo el acortamiento debido a la proyección en cámara. Este ajuste mejora la estabilidad de la medición incluso cuando la persona adopta posturas ligeramente asimétricas.

3.7.3. Corrección del folio mediante recuperación del ratio A4 (1.414)

El módulo anterior utilizaba `minAreaRect` para detectar el folio, pero este método no corrige la distorsión producida por inclinaciones del folio respecto al plano de la cámara. Para preservar la validez métrica del A4, se implementó un sistema de reconstrucción del ratio real del folio.

El procedimiento consiste en:

1. Obtener los cuatro vértices detectados.
2. Calcular los lados proyectados (ancho y alto).
3. Ajustar geométricamente el rectángulo para que recupere la proporción 1.414 del formato A4 [11].
4. Utilizar la dimensión corregida como referente métrico para la regla de tres.

Este método permite medir correctamente incluso cuando el usuario sostiene el folio inclinado o parcialmente rotado.

3.7.4. Eliminación de la suela del calzado en el cálculo de altura

Otra fuente de error detectada era la diferencia entre la altura real del sujeto (descalzo) y la altura medida con calzado. Para corregirlo, se añadió un parámetro configurable que descuenta la altura de la suela antes de mostrar el valor final.

De esta forma, el sistema entrega una altura real del sujeto independientemente del tipo de calzado utilizado.

3.7.5. Parametrización del código para permitir calibraciones dinámicas

Con el incremento de complejidad del sistema, se incorporó una parametrización completa del script, permitiendo ajustar en tiempo de ejecución:

- Longitud del pie L
- Altura de la suela

- Distancia estimada Z

Esta parametrización convierte el módulo en una herramienta flexible y adaptable, que puede calibrarse fácilmente para distintos usuarios.

3.7.6. Resultado: un sistema estable y preciso

La integración de los elementos anteriores da lugar a un sistema capaz de medir la altura con una base geométrica más robusta, corrigiendo errores de perspectiva, compensando inclinaciones del folio y ajustando la silueta humana con extrema precisión gracias a la segmentación.

El comportamiento final, validado con múltiples pruebas, muestra mediciones coherentes y estables en tiempo real.

4. Experimentación

Para validar la robustez y precisión del sistema desarrollado, se ha seguido una metodología experimental iterativa, dividida en cuatro bloques temporales. Estas pruebas han permitido ajustar los parámetros del algoritmo (umbrales de color, tamaño del ROI, filtro de mediana) y evaluar el desempeño en distintos entornos (iluminación doméstica controlada vs. iluminación fluorescente en el aula).

Como referencia para el cálculo del error, se establecen las alturas reales de los sujetos de prueba del equipo:

Sujeto	Altura Real (cm)
Darío	182
Javier	192
Paulo	180

Cuadro 1: Alturas reales de los integrantes del equipo para referencia experimental

4.1. Bloque 1: Validación Inicial en Entorno Controlado

- **Fecha:** 21-23 de noviembre.
- **Entorno:** Domicilio particular (Pasillo).
- **Sujeto de pruebas:** Darío.
- **Objetivo:** Probar la primera versión funcional del script con detección de folio estándar.

En esta fase se realizaron mediciones variando la distancia a la cámara (de 2 a +3 metros) y las condiciones de luz (Día/Noche). Los datos registrados mostraron que la iluminación juega un papel fundamental en la detección del folio mediante visión clásica (OpenCV) [1].

Día	Momento	Alt. Plat.	Dist.	Alt. Est.	Observaciones
Viernes	Día (Luz natural)	0,92 m	2,20 m	ERROR	No detecta el folio
Viernes	Día (Luz natural)	1,40 m	2,50 m	1,73 m	
Viernes	Noche (Luz artificial)	0,92 m	2,70 m	1,55 - 1,63 m	
Sábado	Día (Luz natural)	1,40 m	2,80 m	ERROR	Problemas de detección. Elección de fondos sobre objeto (pared blanca)
Sábado	Día (Luz natural)	1,40 m	+ 3 m	1,70 m aprox.	Más lejos ofrece resultados + ángulo inclinado hacia delante 20º
Domingo	Día (Luz natural)	1,40 m	3 m	1,60 m aprox.	Distancia mínima para que todo el cuerpo entre en la toma. Se ha actualizado el código para eliminar detecciones de cintura para abajo. Existe más claridad del sol.
Domingo	Día (Luz natural)	1,40 m	2,70 m	1,59 - 1,63 m aprox.	Se ha detectado mejor el folio.
Domingo	Día (Luz natural)	0,92 m	+ 3 m	1,60 m aprox.	Detecta bien el folio.
Domingo	Día (Luz natural)	Variaciones leves hasta los 2 m	Variable (desde 2,10 m hasta 3 m)	No más de 1,62 m	
Domingo	Día (Luz natural)	0,92 m	Variable (desde 2,10 m hasta 3 m)	Desde 1,53 m hasta 1,64 m	Detecta el folio desde 1,20 m hasta poco más allá de 3 m. A medida que me alejo del objetivo, aumenta la altura, pero no pasa de 1,65 m.

Cuadro 2: Cuadro de resultados del experimento del fin de semana del 21 de noviembre

Conclusiones: En esta primera toma de contacto con la experimentación del script principal (`detector-altura-webcam.py`), queda patente que, aunque no se consigue una estimación siquiera mínimamente aproximada a la altura real del sujeto de pruebas, es necesario jugar con las distancias hacia la cámara, así como subir y/o bajar la altura del objetivo de la webcam para obtener (como mínimo) unos resultados.

La gran mayoría de las pruebas se hicieron durante el día (con luz natural solar entrando directamente en el entorno de pruebas), lo que suponía un gran problema, puesto que los rayos incidentes del sol sobre el cuerpo del sujeto alteraban gravemente la detección del folio. No solo eso, sino que además, al ser el pasillo totalmente de color blanco y, si se

le suman los problemas de claridad/alta iluminación mencionados; el reconocimiento del folio terminaba por fallar a favor de patrones fantasma en paredes.

Es por ello que se tomó la decisión de modificar el código interno para poder ser reconocible (véanse los resultados del domingo). Sin embargo, esto no resolvió el problema de la estimación de altura.

4.2. Bloque 2: Pruebas de Integración

- **Fecha:** 26 de noviembre (Seguimiento 3).
- **Entorno:** Aula de clase (A2.10).
- **Sujetos de prueba:** Darío, Javier y Paulo.
- **Objetivo:** Pruebas cruzadas con todos los miembros del equipo.

Durante esta sesión se utilizó el código para medir a los tres integrantes. Aquí se identificó el fallo crítico de diseño mencionado en el apartado de implementación:

1. **Sujeto Paulo (177 cm):** Detección incorrecta ($\sim 180\text{-}181$ cm).
2. **Sujeto Javier (194 cm):** Error mínimo de +2 cm approx. (~ 192 cm).
3. **Sujeto Darío (169 cm): Fallo crítico.** El sistema seguía detectándolo muy por debajo de su altura real (182 cm).

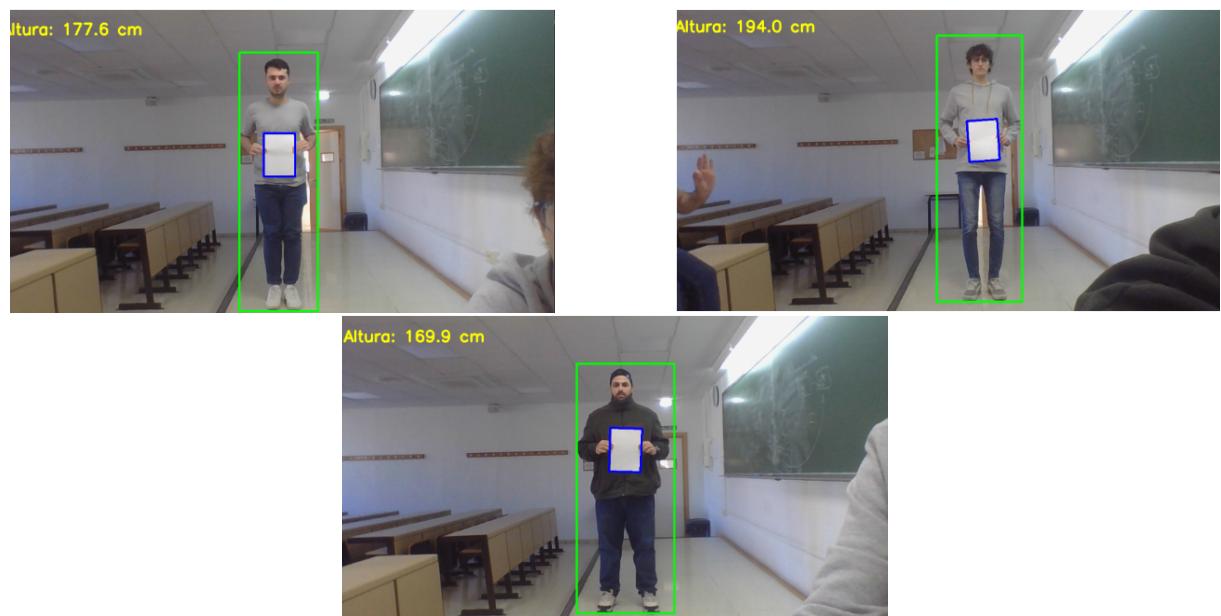


Figura 24: Bloque 2 de las pruebas. Fuente: Darío López

Como se muestra en las capturas tomadas en clase, las alturas de los tres integrantes fallaban de alguna manera, ya sea con un mayor o menor ratio de error. Sin embargo, tras una sesión de discusión con la profesora, donde se comentaron una serie de posibles soluciones a implementar de cara al siguiente Seguimiento; Javier señaló que deberíamos cambiar la localización del folio durante las capturas.

Prueba de ello es que si se alejaba o se acercaba el folio durante la detección, el sistema arrojaba un valor diferente cada vez. Más concretamente, si se acerca el folio (o se empuja) hacia delante, se obtiene una menor estatura. Aquí entra en juego el concepto de “*Error de profundidad*”, que también acusó el equipo japonés de *Takeda et al.* en su paper académico que usamos de referencia.

Con esta idea en mente, probamos a realizar otra sesión de mediciones, esta vez, sosteniendo el folio por la cintura. Debíamos bajar el objeto de referencia tanto como fuera posible, sin llegar a agacharnos en exceso o modificar nuestra postura de tal forma que agravara la correcta estimación de la altura.

Como puede apreciarse en las capturas que se muestran a continuación, la idea tuvo éxito, y las alturas se acercaban con mejores resultados a los valores reales de los sujetos.

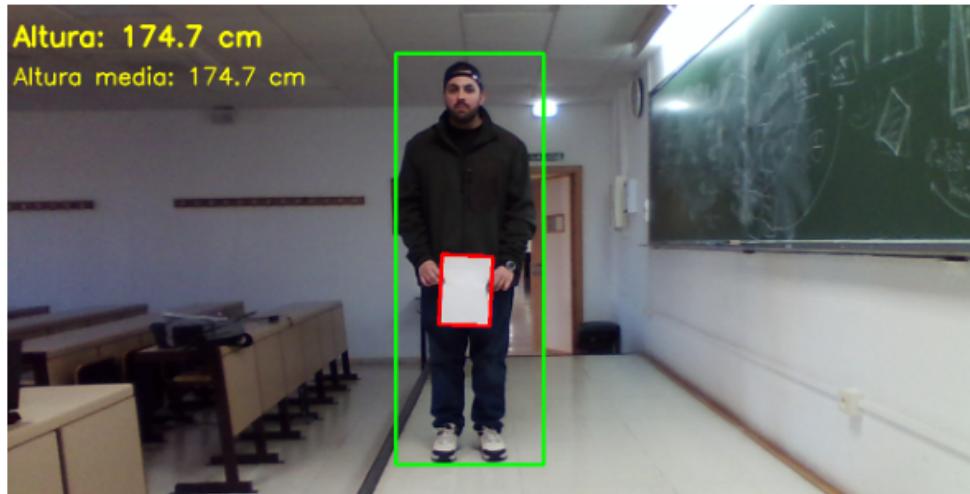


Figura 25: Sujeto *Darío* en la segunda sesión experimental. El cambio de altura es notable. Ha pasado de estimarse como 169 cms a 174 cms. Fuente: Darío López

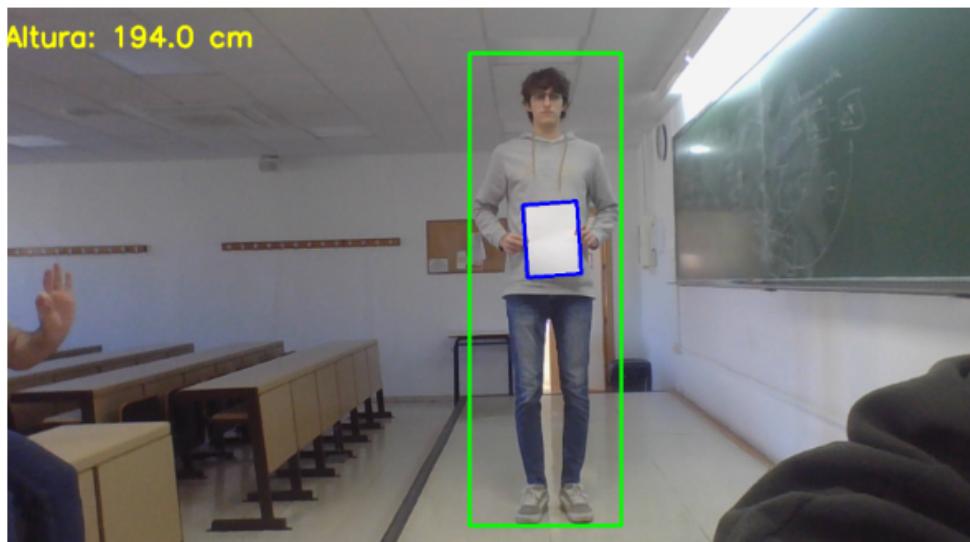


Figura 26: Sujeto *Javier* en la segunda sesión experimental. El error de estimación ha caído de 4 cms (por encima) a solo 2 cms (por debajo). Fuente: Darío López

Conclusiones: Aunque el sistema seguía fallando, en esta ocasión teníamos nuevas alternativas que explorar e introducir en el código. Además, con la última sesión de experimentos, quedó patente el hecho de que el problema radicaba en la posición del folio.

4.3. Bloque 3: Validación de Mejoras (ROI Expandido y Media-na)

- **Fecha:** 2 de diciembre.
- **Entorno:** Domicilio particular (Pasillo).
- **Sujeto de pruebas:** Darío.
- **Objetivo:** Verificar si la expansión lateral del ROI y el filtro de mediana solucionaban los problemas detectados en clase.

En esta tercera sesión, se comenzó por probar algunas opciones diferentes antes de entrar en la modificación del código.

En primer lugar, se utilizó un folio donde previamente se había escrito. Aunque no era blanco total, había sido utilizado. Esto se hizo con el propósito de ver si tal vez debería ser necesario cambiar ligeramente la naturaleza del objeto de referencia. Sin embargo, al no ser detectado siquiera por el ROI del folio, se desechó rápidamente esta idea.

A continuación, se consideró tomar algunas medidas inclinando el folio hacia delante y hacia atrás. Por último, y para probar diferentes escenarios, se llevaron a cabo una serie de mediciones en donde el sujeto no llevaba puesta la gorra. Esto tampoco arrojó ninguna mejora significativa a las estimaciones.

En la segunda parte de esta sesión experimental y, teniendo en cuenta que ya estaba entrando el ocaso de la tarde y había que encender las luces del pasillo (escenario de pruebas), se empezó a modificar el código para ampliar la bounding box del folio. Tras varios ajustes, se consiguió, y para el final de la sesión, se obtuvo una excelente estimación de la altura.

NOTA: En este tercer bloque de experimentos, se llevaron a cabo un total de 28 pruebas bajo condiciones diferentes. Tanto la hoja de cálculo con los resultados de cada prueba, así como las capturas tomadas en cada una de las pruebas, han sido colocadas en el ANEXO I al final de este documento.

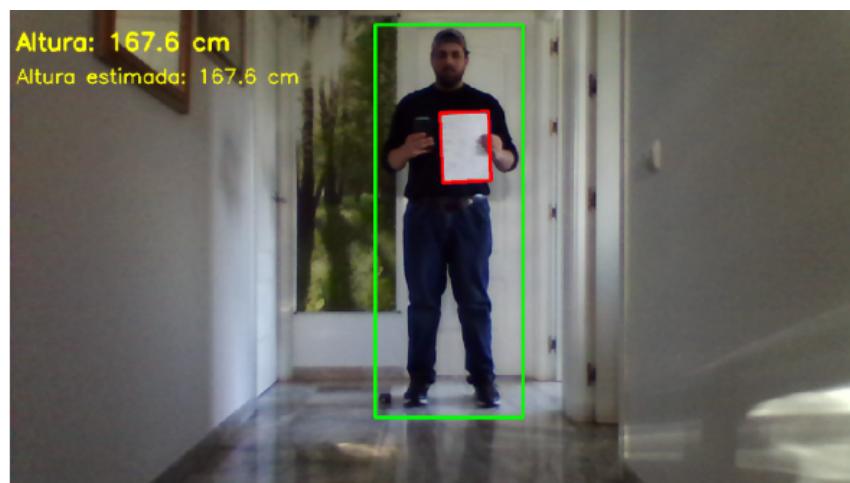


Figura 27: Imagen 23.1. Prueba de estimación empleando un folio blanco, pero escrito. Fuente: Darío López

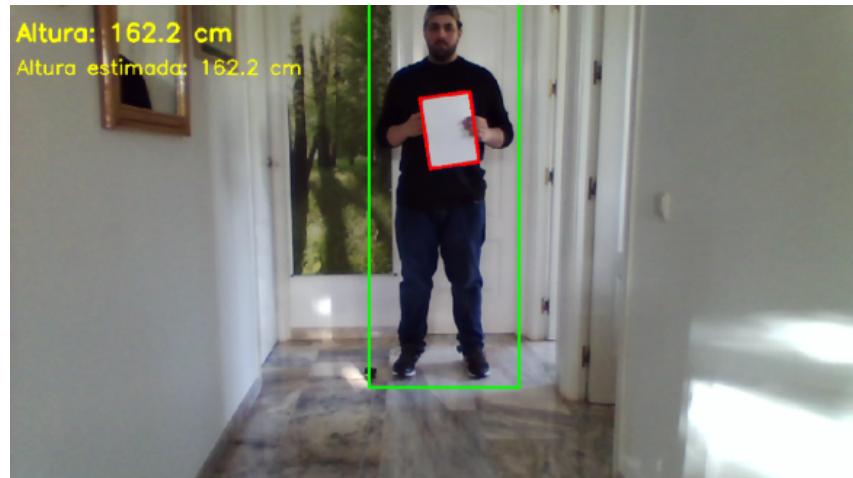


Figura 28: Imagen 23.2. Prueba de estimación basada en inclinar el folio hacia delante y atrás. Fuente: Darío López



Figura 29: Imagen 23.3. Prueba de estimación sin gorra. Fuente: Darío López

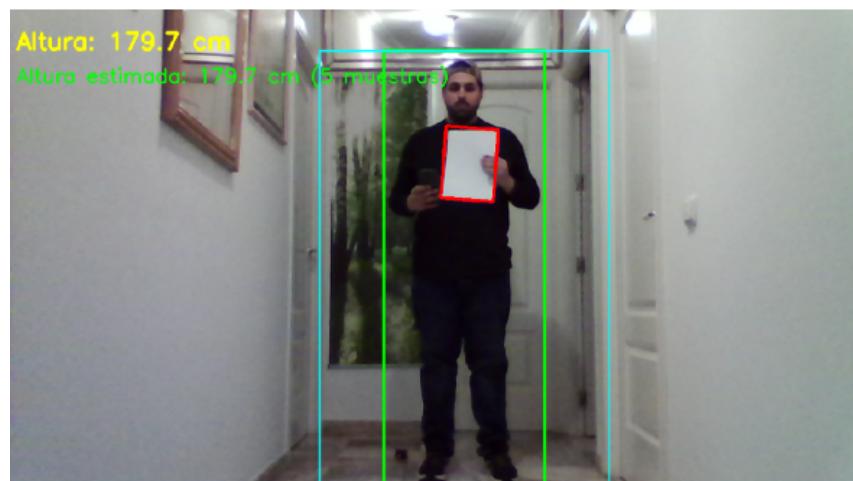


Figura 30: Imagen 24.4. Prueba de noche con luces encendidas y ROI del folio ampliado. Fuente: Darío López

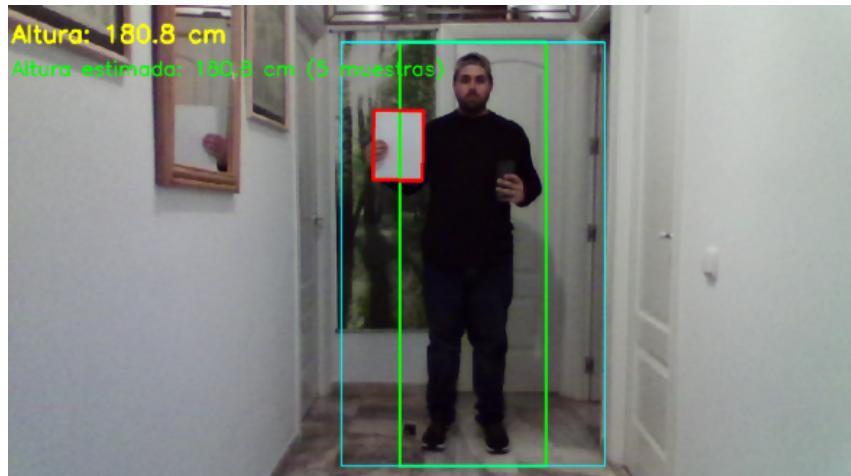


Figura 31: Prueba final de detección. Estimación muy precisa con solo - 2 cms de error. Fuente: Darío López

Conclusiones: Tras esta sesión de experimentos, se consiguió detectar la altura aproximada del sujeto de pruebas, que siempre había sido un problema recurrente en las anteriores sesiones.

Sin embargo, persistía una problemática: la detección del folio de día arrojaba unas medidas muy por debajo de la altura real del sujeto. Además, con la nueva implementación de detección del folio (ROI ampliado por los laterales), era bastante difícil de detectar, pues había que comenzar con el folio pegado justo delante del pecho y desplazarlo suavemente hacia algún lateral. Normalmente, en este proceso de desplazamiento del folio se perdía el foco del folio y había que comenzar de nuevo, situándolo delante del pecho.

4.4. Bloque 4: Validación Final y Comparativa de Escenarios

Fecha: 3 de diciembre (Seguimiento 4).

Entorno: Aula de clase (Aula A2.10).

Sujetos de prueba: Equipo de desarrollo (Darío, Javier, Paulo) y validación externa (Integrantes del Grupo 5).

Objetivo: Evaluar la robustez de la solución basada en ROI (Darío) frente a cambios drásticos de iluminación y fondo, y compararla con la nueva propuesta geométrica (Javier).

NOTA: En este cuarto bloque de experimentos, se llevaron a cabo un total de 13 pruebas bajo condiciones diferentes. En las fases que se muestran a continuación, solo se muestran 9 pruebas del total. Tanto la hoja de cálculo con los resultados de cada prueba, así como las capturas tomadas en cada una de las pruebas, han sido colocadas en el ANEXO II al final de este documento.

Fase A: Evaluación de la Solución basada en ROI En primera instancia, se desplegó la versión que había ofrecido buenos resultados en el entorno doméstico (Bloque 3). Sin embargo, el cambio de escenario al aula A2.10 supuso un desafío insuperable para el algoritmo de visión clásica.

Elementos como la ausencia y, simultáneamente, la existencia de iluminación fluorescente y la alteración del fondo del escenario causaron un deterioro significativo en el desempeño. Como se evidencia en las siguientes capturas, el sistema fue incapaz de mantener la precisión, arrojando mediciones erráticas o perdiendo el referente visual constantemente en los tres sujetos del equipo.



Figura 32: Sujeto: Darío. Escenario con baja iluminación. No existe detección del folio.

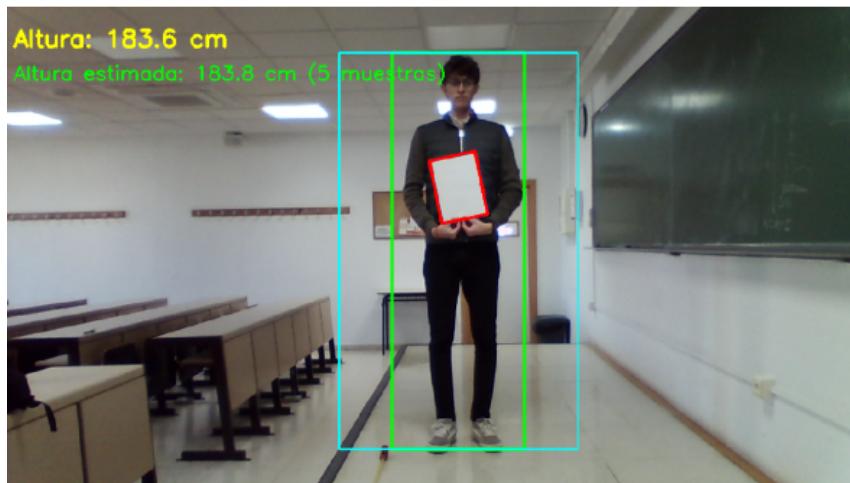


Figura 33: Sujeto: Javier. Escenario iluminado. Cambio de posición del folio.

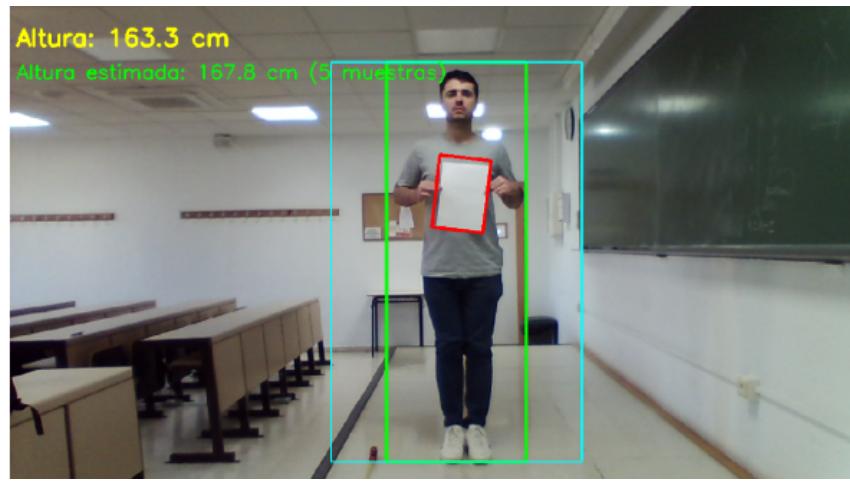


Figura 34: Sujeto: Paulo. Escenario iluminado. Sujetando el folio normalmente.

Como puede observarse, en este escenario los resultados arrojados son muy inestables. Quedó patente que la iluminación del ambiente era un asunto obligatorio a tener en cuenta

si se quería tener una detección del objeto de referencia. Además, el contorno del folio presentaba también ciertas inconsistencias, como se presenta en la última toma de Paulo.

Fase B: Implementación del Nuevo Modelo Geométrico

Ante la inestabilidad de la detección por color/ROI, se procedió a testear la nueva propuesta desarrollada por el integrante Javier. Esta versión introduce un cambio de paradigma:

- **Abandono del ROI del Folio:** Se elimina la dependencia de detectar el folio pixel a pixel mediante color.
- **Proyección Geométrica:** Se implementa un cálculo basado en variables del entorno introducidas manualmente, a saber:
 - distancia del sujeto al objetivo de la cámara.
 - Diferencia entre el torso y la puntera del calzado en el sujeto.
 - Grosor de la suela del calzado.
- Además, se presenta un nuevo modelo de detección del humano. Se trata de la versión “segmentación” de la versión 8 de YOLO.

Las pruebas realizadas sobre los mismos sujetos mostraron una corrección inmediata de los errores, estabilizando la medición en valores muy cercanos a la realidad antropométrica de los integrantes.

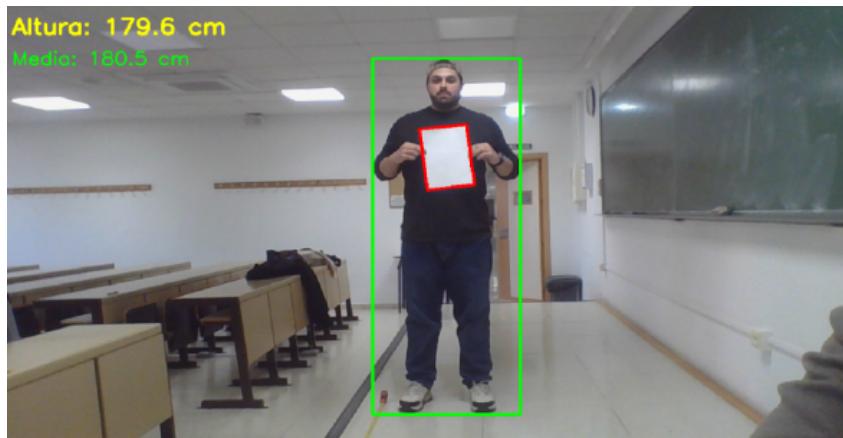


Figura 35: Sujeto: Darío. Entorno iluminado y folio torcido. Buena estimación con solo 2 cms de error de media.

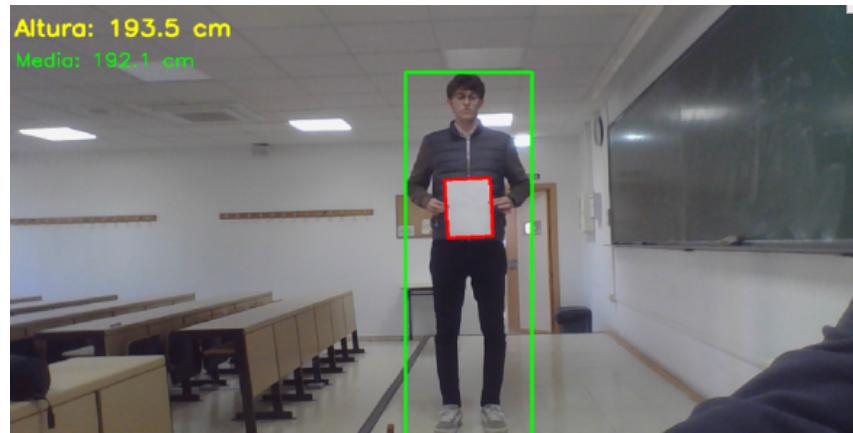


Figura 36: Sujeto: Javier. Entorno iluminado y folio recto. Estimación precisa.

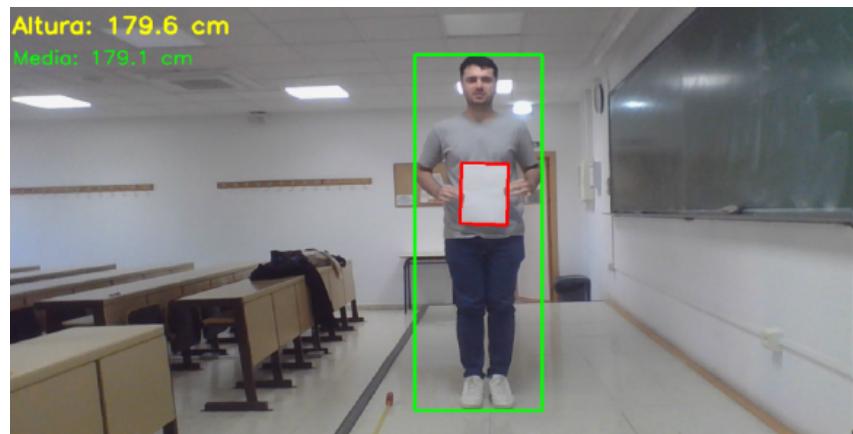


Figura 37: Sujeto: Paulo. Entorno iluminado y postura correcta. Error de estimación ± 1 cm.

Fase C: Validación Externa

Para descartar que el sistema estuviera "sobreajustado."^a los cuerpos de los integrantes, se realizó una prueba ciega con voluntarios externos al equipo (integrantes del Grupo 5). El sistema respondió correctamente a fisonomías y estaturas no ensayadas previamente, confirmando la generalidad y robustez de la solución final propuesta.

Alturas reales de los participantes:

- **Yue Liu:** 168 cm
- **Irene Ledi Leonés León:** 161 cm
- **Enrique Grijuelo Muñoz:** 178 cm



Figura 38: Sujeto: Yue. Altura calculada sin ningún error de aproximación.

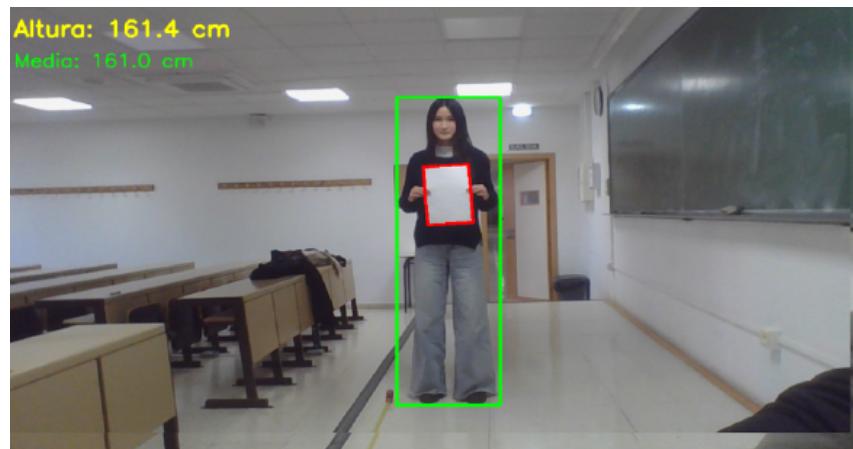


Figura 39: Sujeto: Irene. Igualmente, altura calculada sin ningún error presente.

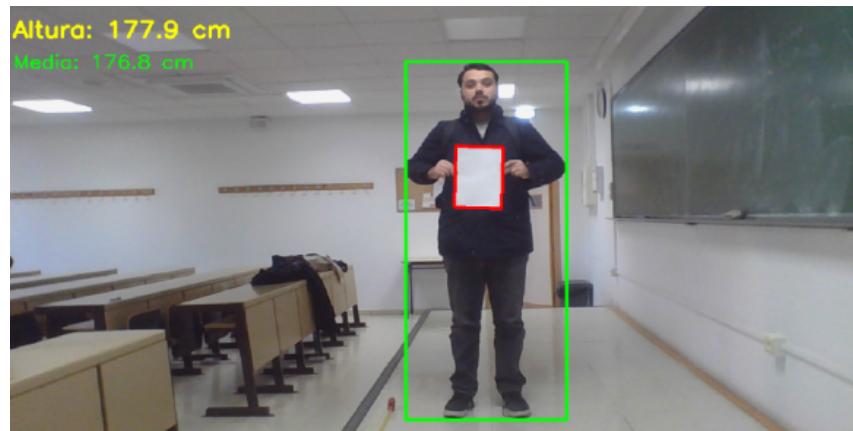


Figura 40: Enrique. Presenta un error de 1 - 2 cms (esperable).

Conclusiones: Esta última sesión experimental fue decisiva. Demostró que, si bien la solución basada en Visión Artificial Clásica (Color/ROI) es válida en entornos controlados, carece de la robustez necesaria para entornos ruidosos e iluminaciones variables. Por el contrario, la aproximación basada en Geometría Proyectiva y calibración manual (propuesta final de Javier) resultó ser la solución final óptima para el reto planteado en el aula.

5. Manual del usuario

Este manual describe el proceso necesario para ejecutar el sistema de detección de persona + detección de folio + medición de altura corregida, implementado en el script `detector-altura-webcam.py`.

5.1. Requisitos previos

Antes de ejecutar el proyecto, asegúrese de disponer de los siguientes elementos:

- **Modelo YOLOv8-Seg:**

- El archivo `yolov8s-seg.pt` debe situarse en la raíz del proyecto.
 - Si se usa otro modelo YOLOv8-Seg, el nombre debe actualizarse en el código.

- **Python 3.12.x:** (Versiones más recientes pueden no ser compatibles con las dependencias en las versiones especificadas).

- Versión probada: 3.12.6

- **pip:** Gestor de paquetes de Python (incluido normalmente con Python 3)

- **git:** (opcional, solo si clonas el repositorio).

Dependencias principales:

- OpenCV
- Ultralytics YOLOv8 (segmentación)
- NumPy

5.2. Crear entorno virtual “env”

Es altamente recomendable usar un entorno virtual para aislar las dependencias. En la raíz del proyecto, ejecute:

Windows (PowerShell)

```
py -3.12 -m venv env
```

Windows (cmd)

```
py -3.12 -m venv env
```

Linux / macOS

```
python3.12 -m venv env
```

5.3. Activar el entorno virtual

Windows (PowerShell)

```
.\env\Scripts\Activate.ps1
```

Windows (cmd)

```
env\Scripts\activate.bat
```

Linux / macOS

```
source env/bin/activate
```

Si el entorno está activado correctamente, la terminal mostrará algo como:

```
(env) C:\proyecto>
```

5.4. Instalar las dependencias

Con el entorno virtual activado, ejecute:

```
pip install -r requirements.txt
```

Esto instalará:

- OpenCV
- Ultralytics YOLOv8
- NumPy
- Otros módulos necesarios

5.5. Ejecutar el detector de altura

La funcionalidad completa se encuentra en: `detector-altura-webcam.py`
Este script:

- Abre la webcam
- Detecta a la persona mediante YOLOv8-Seg
 - *El script no funcionará si no se encuentra un archivo de pesos YOLOv8-Seg.*
- Detecta el folio dentro de la bounding box exacta
- Calcula la altura corregida usando:
 - Perspectiva
 - Relación geométrica del folio
 - Modelo $Z/(Z - L)$
 - Ajuste por suela

Parámetros obligatorios y opcionales El script acepta:

Parámetro	Tipo	Obligatorio	Descripción
-Z	float	Si	Distancia cámara-torso (en metros).
-L	float	No	Adelanto entre torso y punta del pie (default = 0.15 metros).
--suela	float	No	Corrección de suela/calzado en cm (default = 1.0 cm).

Cuadro 3: Parámetros del script detector-altura-webcam.py

Ejemplo de uso Con el entorno activado:

```
python detector-altura-webcam.py --Z 2.10 --L 0.10 --suela 2.0
```

Funcionamiento Al ejecutarse:

- Se abre una ventana llamada “Altura”
- Se mostrará en tiempo real:
 - Bounding box de la persona
 - Folio detectado
 - Altura estimada (cm)
 - Mediana móvil de los últimos valores (cm)
- Para salir, pulsa Q.

5.6. Notas importantes

- Asegúrese de que el folio esté dentro de la bounding box de la persona, ya que el algoritmo no busca fuera de ella.
- La cámara debe estar quieta para que la corrección geométrica sea estable.
- La medición final depende de la precisión de Z y L introducidos por el usuario.

5.7. Errores comunes y soluciones

YOLO no detecta persona

Aumenta iluminación o ponte completamente dentro del encuadre.

No detecta folio

Evita folios demasiado doblados o iluminaciones muy amarillas.

Altura varia demasiado

Revisa:

- Que Z esté bien medida
- Que L sea adecuado
- Que el folio esté recto y visible

6. Conclusiones

Este proyecto demostró la viabilidad de un sistema de estimación de altura humana que integra con éxito el aprendizaje profundo (modelo YOLOv8-SEG) para detección de personas con la visión artificial clásica (algoritmos OpenCV para procesamiento de imagen). El uso de una hoja A4 como objeto de referencia métrica abordó el problema de la escala desconocida inherente a las imágenes bidimensionales.

La fase de implementación destacó la importancia de decisiones de diseño, como la utilización del espacio de color LAB para aislar el referente con robustez contra las variaciones de iluminación, asegurando que el filtro geométrico y de color pudiera segmentar la hoja de papel blanco de manera consistente a partir de la Región de Interés (ROI) recortada alrededor del sujeto.

No obstante, la fase experimental reveló la limitación más crítica del sistema: el Error de Profundidad (depth error), que ocurre cuando la posición del objeto de referencia (la hoja) se mueve hacia adelante o hacia atrás del sujeto, falseando la medición por regla de tres. Esta limitación, que fue atenuada pero no resuelta por correcciones en la postura de sujeción de la hoja, indica que el enfoque puramente bidimensional es insuficiente.

En consecuencia, el trabajo futuro (apartado 3.7) se centrará en la introducción de modelos geométricos que consideren la profundidad z y en la evaluación de nuevos modelos de *Deep Learning* para la segmentación de la hoja, con el fin de mitigar la dependencia de los umbrales de color y del ruido del algoritmo de visión clásica.

7. Autoevaluación

La realización de este proyecto ha supuesto un desafío técnico que ha trascendido la mera implementación de librerías. A lo largo de las 50 horas de desarrollo por integrante, el equipo ha trabajado con el objetivo de trasladar a una aplicación real los conceptos teóricos propuestos por **Takeda et al. (2024)** en su investigación “*Calibration-Free Height Estimation for Person*”.

Nuestra propuesta ha consistido en validar experimentalmente si la premisa de dichos autores —la posibilidad de estimar la altura humana basándose en la relación con un objeto de referencia sin calibración compleja— es viable en un entorno doméstico y con hardware de bajo coste (webcam).

7.1. Cumplimiento de Objetivos y Validación Teórica

El objetivo principal se considera **cumplido satisfactoriamente**. Siguiendo la metodología de referencia basada en objetos de tamaño conocido (*Reference Object*), hemos logrado desarrollar un sistema que:

1. Identifica al sujeto y al referente (folio A4) automáticamente.
2. Calcula la relación píxel/cm en tiempo real.
3. Implementa correcciones geométricas (profundidad, perspectiva y calzado) que no estaban contempladas en los tutoriales básicos de YOLO.

La fase experimental (Sección 4) ha sido clave para desmitificar la simplicidad teórica. Hemos comprobado empíricamente que la *Visión Artificial Clásica* (detección por color) es insuficiente en entornos no controlados, validando así la necesidad de técnicas híbridas (Deep Learning + Geometría Proyectiva) que finalmente implementamos en el Seguimiento 4.

7.2. Aportaciones Individuales

El trabajo se ha distribuido de forma equitativa, aprovechando las fortalezas de cada miembro:

- **Juan Javier Sánchez:**

- **Liderazgo Técnico:** Desarrollo del núcleo algorítmico en Python. Implementación de los scripts de detección de folio, integración con YOLO y diseño del modelo matemático de corrección de perspectiva.
- **Explicación de código:** Al principio de cada clase, explicó todo el código con dibujos y pruebas a los compañeros para poder continuar con el proyecto.
- **Documentación:** Documentación del código final, README del repositorio y, apartados de YOLOv8-SEG e implementación del seguimiento 4.

- **Darío López:**

- **Concepción del Proyecto:** Investigación de material para el desarrollo del proyecto y verificación de la viabilidad técnica para la aplicación de las ideas de los compañeros.
- **Implementación y Pruebas:** Responsable de la implementación inicial del proyecto y de las primeras pruebas del código para verificar la viabilidad del uso de YOLO v8, así como de la participación en las pruebas con folios en el aula.
- **Documentación:** Desarrollo de la estructura del texto, con la redacción de los apartados de Introducción, planteamiento teórico y parte de la sección de implementación. También responsable de la conversión del texto a LaTeX y de la verificación ortográfica del mismo.

- **Paulo Felipe Rezende:**

- **Investigación:** Búsqueda exhaustiva del estado del arte, selección de papers académicos (como el de Takeda et al.) y documentación teórica de los algoritmos (Canny, Morfología, Espacios de Color).
- **Documentación y Presentación:** Redacción de la memoria técnica en LaTeX, elaboración de diagramas explicativos y preparación de los materiales visuales para la defensa del proyecto.

7.3. Líneas de Trabajo Futuro

Para acercar aún más nuestro prototipo a la precisión descrita por Takeda et al., se proponen las siguientes mejoras:

- **Detección Semántica del Referente:** Entrenar un modelo YOLO específico para detectar el folio A4, eliminando la dependencia del color y permitiendo usar el objeto en cualquier orientación.
- **Autocalibración:** Implementar detección de marcadores ArUco para obtener los parámetros intrínsecos de la cámara en tiempo real, sin intervención manual.

8. Tabla de tiempos individuales

8.1. Seguimiento 1

Darío López Villegas	
Tarea	Tiempo
Discusión de varias propuestas en grupo (clase del miércoles 5 de noviembre); estudiando a su misma vez la viabilidad, complejidad e integración de cada una de ellas con los conceptos estudiados en la asignatura, así como con aquellos que estudié personalmente de la librería de OpenCV	2h
Lectura comprensiva de cada una de las propuestas más prometedoras extraídas de la sesión arriba mencionada, además de las halladas por el compañero Javier durante su sesión de búsqueda	5h
Búsqueda, lectura e incorporación de fuentes adicionales científicas de inspiración y referencia para la realización del presente trabajo	1h
Total	8h

Juan Javier Sánchez Portillo	
Tarea	Tiempo
Elegir la idea inicial y comprobar viabilidad	2h
Búsqueda y estudio de artículos científicos, repositorios y guías	4h
Idea inicial de técnicas a utilizar	1h
Establecimiento de objetivos, explicación del proyecto, planificación inicial y desarrollo del PDF de seguimiento 1	2h
Lectura del resto de artículos, repositorios y guías aportados por mis compañeros	2h
Total	11h

Paulo Felipe Rezende da Silva	
Tarea	Tiempo
Búsqueda de artículos científicos y fuentes para utilizar en el proyecto	3h
Lectura de los resultados de la búsqueda, con la verificación inicial de la viabilidad del empleo de las tecnologías utilizadas	4h
Incorporación de las fuentes encontradas para el desarrollo del proyecto	1h
Lectura de las propuestas añadidas anteriormente por mis compañeros Darío y Javier	2h
Total	10h

8.2. Seguimiento 2

Darío López Villegas	
Tarea	Tiempo
Estudio comprensivo del artículo científico	10h
Lectura y estudio de YOLO	1h
Estudio y experimentación con el repositorio de detección empleando YOLOv8	3h
Total	14h

Juan Javier Sánchez Portillo	
Tarea	Tiempo
Corregir seguimiento 1	1h
Estudiar YOLO v8	2h
Estudiar repositorio elegido	1h
Creación de nuestro script + estructura del proyecto + documentación	5h
Ajuste del script para mayor precisión + pruebas	1h
Total	10h

Paulo Felipe Rezende da Silva	
Tarea	Tiempo
Estudio sobre el funcionamiento del YOLOv8	4h
Análisis del repositorio elegido	1h
Pruebas del script	3h
Ajuste del script	4h
Total	12h

8.3. Seguimiento 3

Darío López Villegas	
Tarea	Tiempo
Código: Corrección + Experimentos en casa	14 h
Experimentos en el aula (día 25)	2 h
Total	16h

Juan Javier Sánchez Portillo	
Tarea	Tiempo
Desarrollo script detector de folio (incluida elección y estudio de técnicas)	4 h
Desarrollo script completo de detección de altura	0,5 h
Pruebas y ajuste de parámetros	2,5 h
Documentación de todo el desarrollo para la comprensión por parte de los compañeros	1 h
Total	8h

Paulo Felipe Rezende da Silva	
Tarea	Tiempo
Pesquisa bibliográfica	8h
Desarrollo de la documentación	6h
Pruebas del proyecto	1,5h
Total	15,5h

8.4. Seguimiento 4

Darío López Villegas	
Tarea	Tiempo
Cambios en el código + Experimento Escenario A	4h
Experimentos en el aula - Escenario B - (día 03/12)	2h
Confección de “Implementación y Experimentación” de la Memoria	6h
Total	12h

Juan Javier Sánchez Portillo	
Tarea	Tiempo
Estudio de segmentación basada en máscaras (YOLO-SEG)	2 h
Estudio de geometría proyectiva aplicada al folio A4	4 h
Revisión de errores por perspectiva y proyección vertical	1 h
Diseño del modelo geométrico para corregir que los pies sobresalgan	2 h
Implementación de la corrección de los pies	1 h
Corrección del folio según ratio real + compensación de perspectiva	0,5 h
Integración completa en el módulo de webcam	1,5 h
Pruebas en casa + clase	6 h
Documentación del código	1 h
Documentación en la memoria	4 h
Implementación parametrización del script	2h
Total	25 h

Paulo Felipe Rezende da Silva	
Tarea	Tiempo
Pesquisa bibliográfica para la presentación	3h
Desarrollo de la documentación	6h
Pruebas del proyecto	1h
Desarrollo del archivo para la presentación	4h
Total	14h

8.5. Tiempos totales de cada componente

Nombre	Tiempo total
Darío López Villegas	50h
Juan Javier Sánchez Portillo	54h
Paulo Felipe Rezende da Silva	51,5h

Referencias

- [1] GONZALEZ, R. C.; WOODS, R. E. (2018). *Digital Image Processing*. 4. ed. Pearson. Capítulos 9 y 10.
- [2] CANNY, J. (1986). *A Computational Approach to Edge Detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, n. 6, pp. 679-698.
- [3] ULTRALYTICS. (2025). *Limiarização no Processamento de Imagens Explicada*. Disponible en <https://www.ultralytics.com/pt/blog/thresholding-in-image-processing>. Consulta: 26 noviembre 2025.
- [4] KAIZOUDOU. (2025). *From RGB to Lab color space*. Disponible en <https://kaizoudou.com/from-rgb-to-lab-color-space>. Consulta: 26 noviembre 2025.

- [5] VISÃO COMPUTACIONAL. (2025). *Morfologia Matemática para Processamento de Imagens*. Disponible en <https://visaocomputacional.com.br/morfologia-matematica-para-processamento-de-imagens>. Consulta: 24 noviembre 2025.
- [6] TIR - Universidad Federal de Pernambuco. (2025). *7. Morfología Matemática*. Disponible en <https://www.cin.ufpe.br/~tir/ComputacaoCientifica/7.Morfologia%20Matematica.pdf>. Consulta: 24 noviembre 2025.
- [7] PROGRAMAR FÁCIL. (2025). *Detector de bordes Canny, cómo contar objetos con OpenCV y Python*. Disponible en <https://programarfácil.com/blog/vision-artificial/detector-de-bordes-canny-opencv>. Consulta: 25 noviembre 2025.
- [8] OPENCV. (2025). *Contour Features*. Disponible en https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html. Consulta: 25 noviembre 2025.
- [9] OPENCV. (2025). *Creating Bounding rotated boxes and ellipses for contours*. Disponible en https://docs.opencv.org/3.4/de/d62/tutorial_bounding_rotated_ellipses.html. Consulta: 25 noviembre 2025.
- [10] HALL OF PRINT. (2025). *A4 Paper Size Guide*. Disponible en <https://www.hallofprint.com/a4-paper-size-guide>. Consulta: 23 noviembre 2025.
- [11] ORGANIZACIÓN INTERNACIONAL DE NORMALIZACIÓN (ISO). (2007). *ISO 216:2007 Writing paper and certain classes of printed matter — Trimmed sizes — A and B series, and indication of machine direction*.
- [12] STACKOVERFLOW. (2025). *What's the difference in results of cvBoundingRect and cvMinAreaRect?*. Disponible en <https://stackoverflow.com/questions/69911364/whats-the-difference-in-results-of-cvboundingrect-and-cvminarearect>. Consulta: 25 noviembre 2025.
- [13] JOCHER, G.; CHAURASIA, A.; QIU, J. (2023). *YOLO by Ultralytics*. Ultralytics.
- [14] ASKPYTHON. (2025). *3 Effective Methods for Applying Gaussian Filters to Images*. AskPython en línea. Disponible en <https://www.askpython.com/python-modules/applying-gaussian-filters-to-images>. Consulta: 26 noviembre 2025.
- [15] RESEARCHGATE. (2025). *Figura 4: Supresión No Máxima a) Persona No. 1 b) Persona No. 2 y c) Persona No. 3*. Disponible en https://www.researchgate.net/figure/Figura-4-Supresion-No-Maxima-a-Persona-No-1-b-Persona-No-2-y-c-Persona-No-3_fig4_272181378. Consulta: 26 noviembre 2025.
- [16] MATLAB. (2025). *Reducing Noise in Image Gradients*. MATLAB Documentation en línea. Disponible en <https://www.mathworks.com/help/images/reducing-noise-in-image-gradients.html>. Consulta: 8 diciembre 2025.
- [17] RESEARCHGATE. (2025). *Figura 5: Histéresis de Umbral a) Persona No. 1 b) Persona No. 2 y c) Persona*

No. 3. Disponible en https://www.researchgate.net/figure/Figura-5-Histeresis-de-Umbral-a-Persona-No-1-b-Persona-No-2-y-c-Persona-No-3_fig5_272181378. Consulta: 26 noviembre 2025.

- [18] MATHWORKS. (2025). *imfill - Fill image regions and holes*. MATLAB Documentation en línea. Disponible en <https://la.mathworks.com/help/images/ref/imfill.html>. Consulta: 8 diciembre 2025.
- [19] J3LLY-BEEN. (2024). *YOLOv8-HumanDetection software*. Versión 1.0. GitHub. Licencia GPL-3.0. Disponible en <https://github.com/J3lly-Been/YOLOv8-HumanDetection>. Consulta: 8 diciembre 2025.
- [20] JUANJAVIER03. (2024). *Human-Height-Detection-Using-Reference software*. GitHub. Disponible en <https://github.com/JuanJavier03/Human-Height-Detection-Using-Reference>. Consulta: 8 diciembre 2025.

ANEXO I: Tabla del resultado del bloque 3 de experimentos, así como las imágenes capturadas durante la realización del mismo.

El valor XXX.png en la columna archivo hace referencia al nombre de la imagen para cada caso en particular. Con el propósito de no romper la tabla y de que pueda verse en detalle el resultado de cada caso individualmente, se han adjuntado todos estos archivos de imagen al final de esta tabla.

Ambiente	Condiciones	Alt. Plat.	Dist.	Incl.	Alt. Est.	Arch.	Observaciones
Tarde (Sol natural)	Claridad	0,73 m	3 m	90°	1,65 m	1.png	
		0,73 m	3 m	120°	1,69 m	2.png	
		1,18 m	3 m	110°	1,67 m	3.png	Folio NO blanco.
		1,18 m	3 m	110°	1,71 m	4.png	
Tarde (Sol natural)	Claridad	1,18 m	3 m	80°	1,62 m	5.png	Folio inclinado levemente.
		1,18 m	3 m	100°	1,71 m	6.png	
		1,18 m	2,80 m	80°	1,61 m	7.png	Folio lo más recto posible.
		1,18 m	2,60 m	80°	1,67 m	8.png	Sin gorra.
Tarde (Sol natural)	Rayos de luz incidentes	1,18 m	2,80 m	80°	1,69 m	9.png	Sin gorra.
		1,18 m	3 m	80°	1,70 m	10.png	Sin gorra.
		1,18 m	2,80 m	80°	ERROR	11.png	Arreglada la función de altura media.
		1,18 m	3 m	80°	ERROR	12.png	
Tarde (ocaso)	Poca luz natural	0,73 m	3 m var.	120°	ERROR	13.png	
		0,73 m	3 m var.	90° aprox.	1,74 m	14.png	Sin gorra. Menos distancia es peor.
		1,18 m	3 m	90° aprox.	1,72 m	15.png	Con gorra. Menos distancia es peor.
		1,18 m	2,50 m	90° aprox.	1,69 m	16.png	Menos luz natural.
Tarde (Luz artificial)	Menos luz natural	1,18 m	3 m var.	—	1,74 m	17.png	
		1,18 m	3 m var.	90° aprox.	1,79 m	18.png	Bounding box del folio mayor.
		1,18 m	3 m	—	1,79 m	19.png	Bounding box del folio solo por los lados y al 50 % de la anchura.
		1,18 m	3 m juntos	90° aprox.	1,73 m	20.png	Detector menos restrictivo.
	Lámpara 100 %	1,18 m	3 m	—	1,62 m	21.png	Muestra peor detección del folio.
		0,73 m	2,60 m	90° aprox.	1,79 m	22.png	Código editado, detecta peor el folio.

Ambiente	Condiciones	Alt. Plat.	Dist.	Incl.	Alt. Est.	Arch.	Observaciones
		1,18 m	3 m	—	ERROR	23.png	No detecta el folio, le cuesta buscarlo.
		1,18 m	3 m var.	—	1,80 m	24.png	Tras jugar a detectar el folio, buenos resultados.
		1,18 m	3 m jus- tos	—	ERROR	25.png	Código como antes de cambiar el ROI del folio.
		1,18 m	3 m	—	1,77 m	26.png	Repetición de la instancia de arriba.
		1,18 m	3 m	—	1,80 m	27.png	Cuesta captar el folio colocado al lado.
		1,18 m	3 m	—	1,78 m	28.png	Cuesta captar el folio colocado delante del pecho.

Leyenda: Alt. Plat. = Altura de la plataforma; Dist. = Distancia cámara; Incl. = Inclinación; Alt. Est. = Altura estimada; Arch. = Archivo; var. = variable.

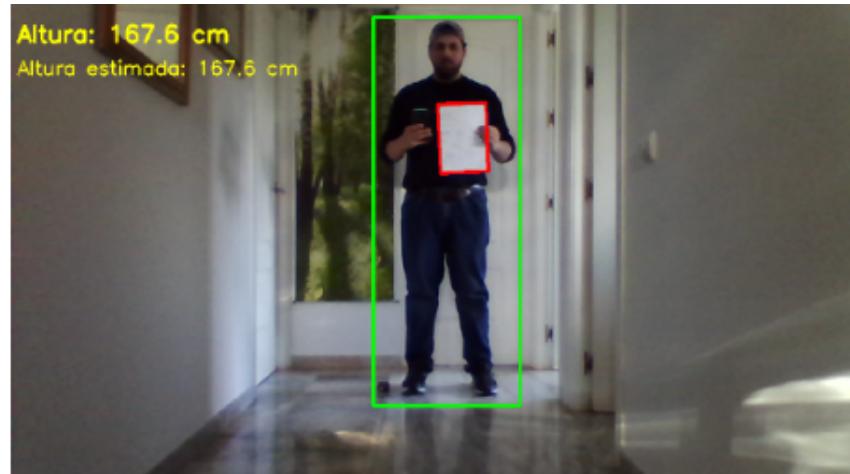
Imágenes del Anexo I



1.png



2.png



3.png



4.png



5.png



6.png



7.png



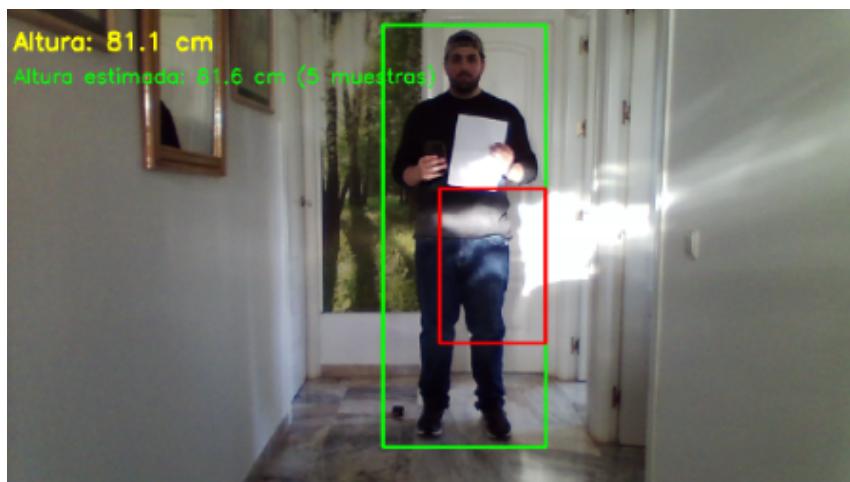
8.png



9.png



10.png



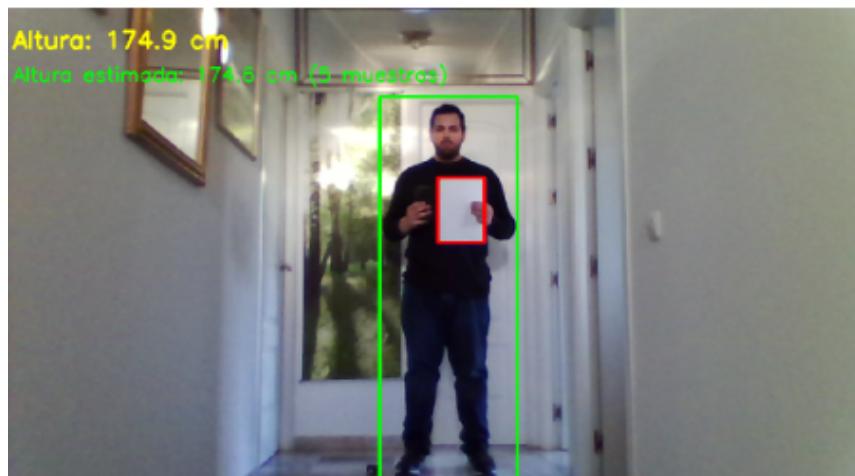
11.png



12.png



13.png



14.png



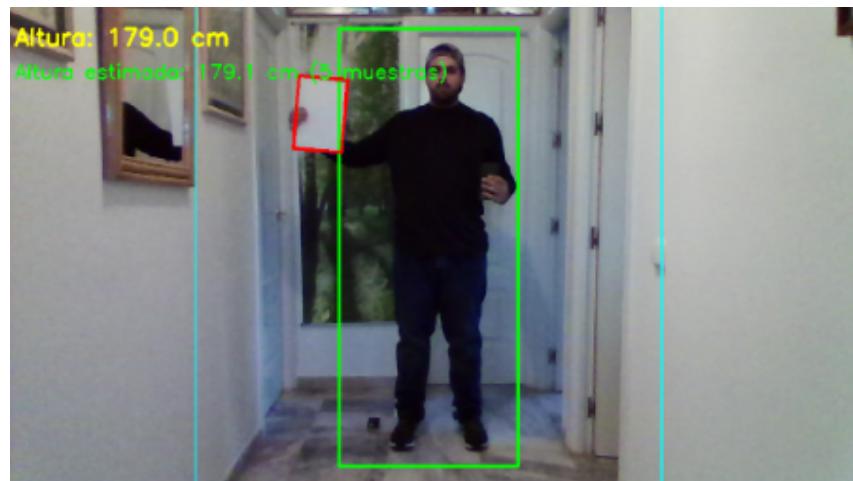
15.png



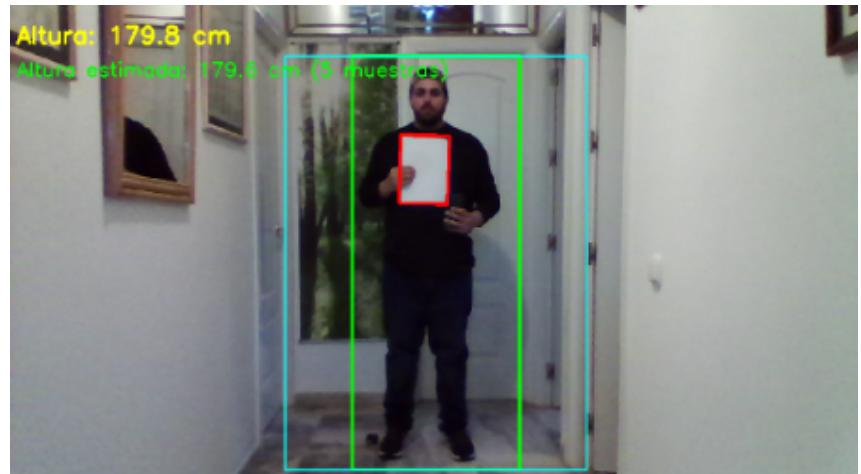
16.png



17.png



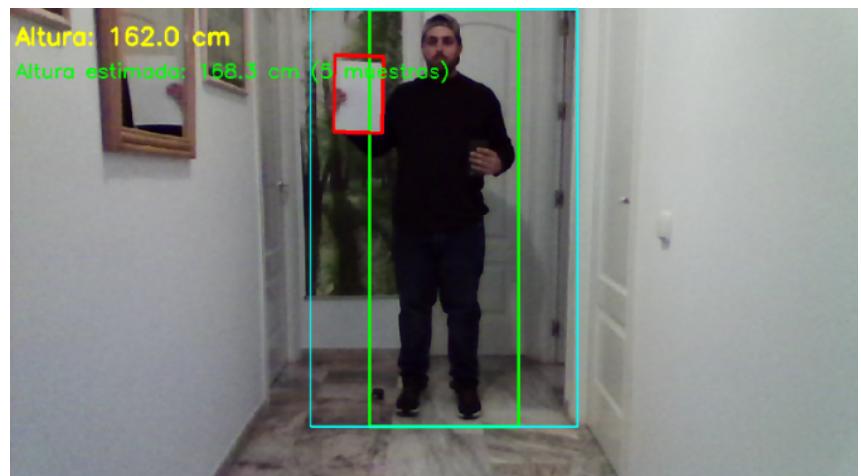
18.png



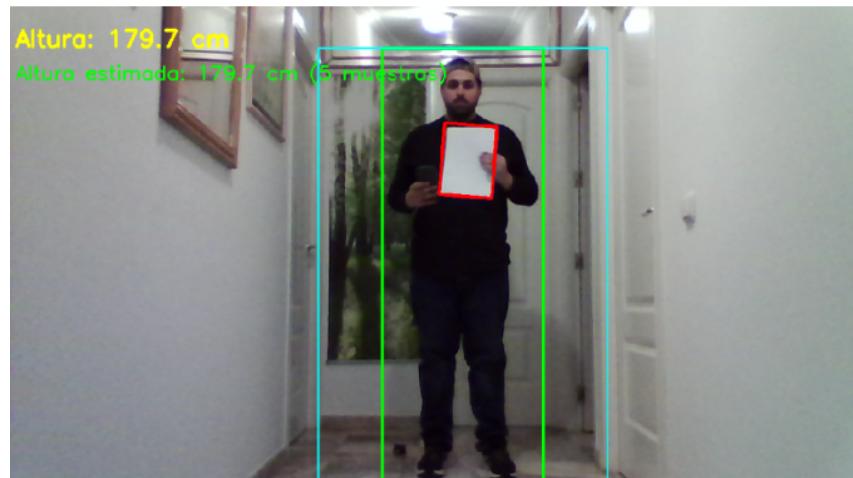
19.png



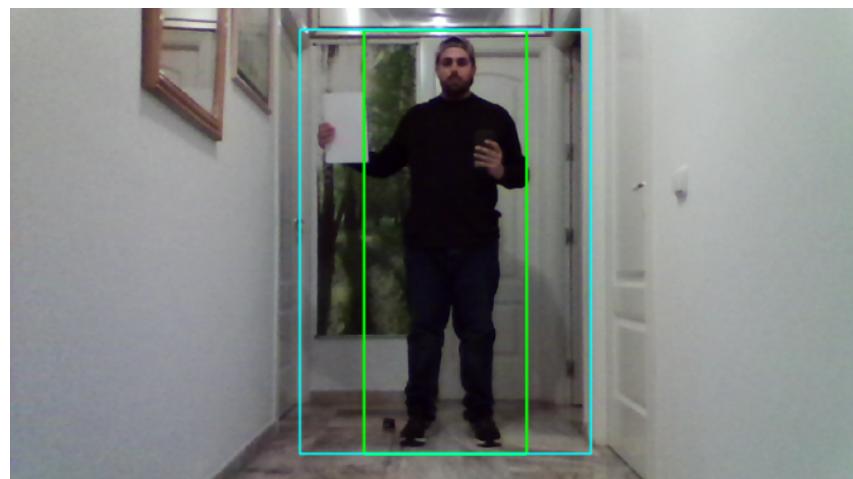
20.png



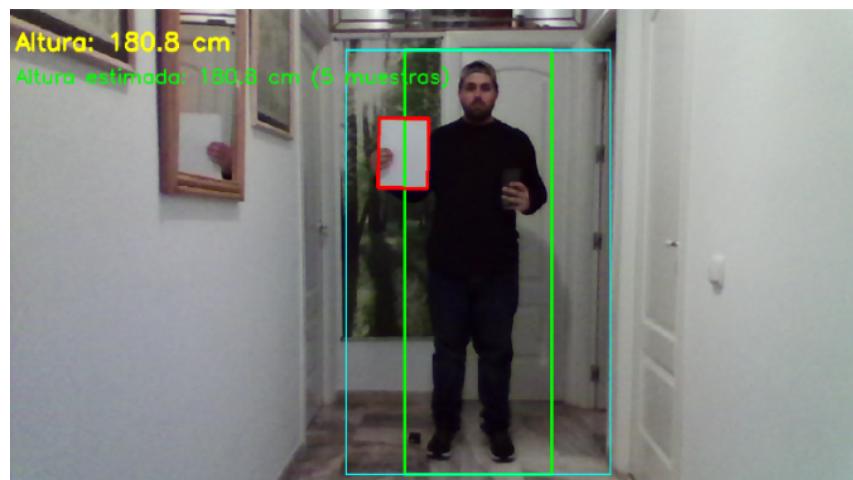
21.png



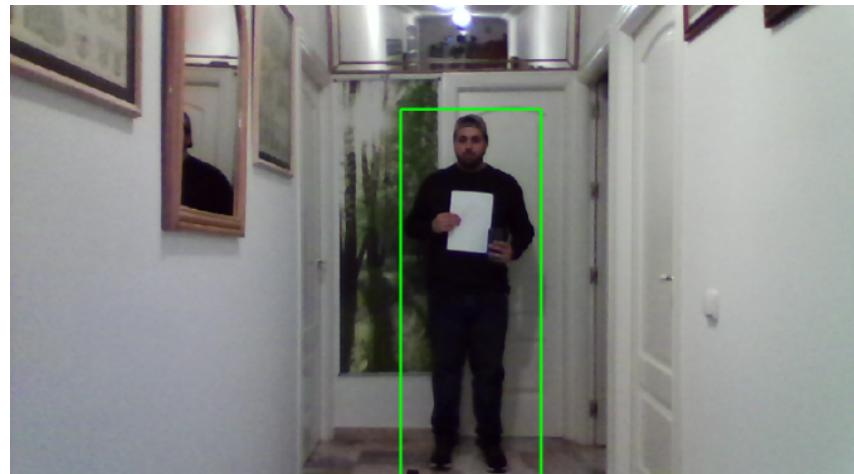
22.png



23.png



24.png



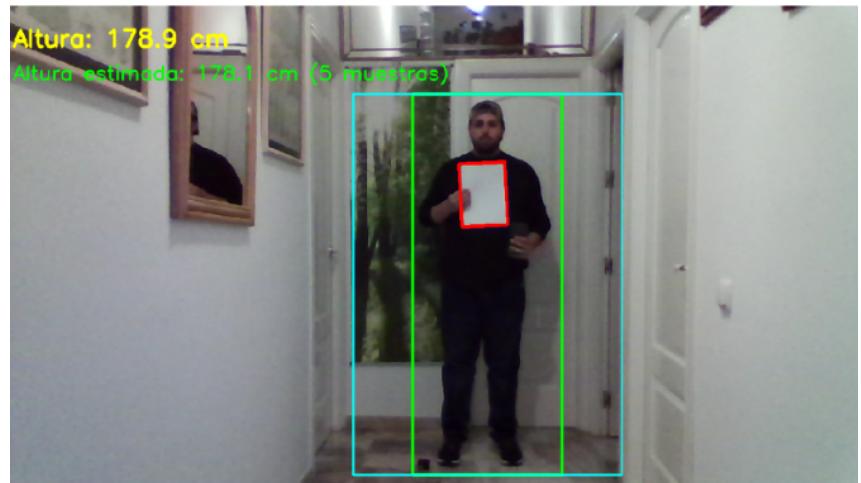
25.png



26.png



27.png



28.png

ANEXO II: Tabla del resultado del bloque 4 de experimentos, así como las imágenes capturadas durante la realización del mismo.

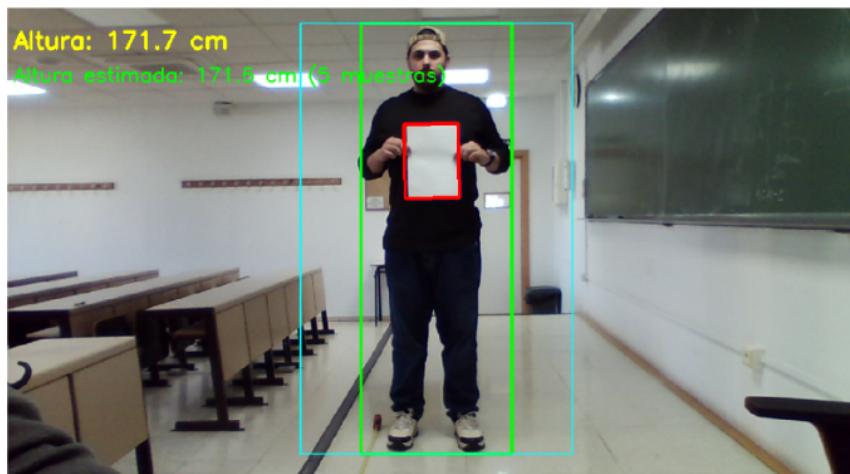
El valor XXX.png en la columna archivo hace referencia al nombre de la imagen para cada caso en particular. Con el propósito de no romper la tabla y de que pueda verse en detalle el resultado de cada caso individualmente, se han adjuntado todos estos archivos de imagen al final de esta tabla.

Ambiente	Condiciones	Alt. Plat.	Dist.	Incl.	Alt. Est.	Arch.	Observaciones
Tarde (Luces clase + claridad)	Penumbra	1,05 m	3 m	90º	ERROR	1.png	No se detecta la bounding box del folio.
	luces techo on	1,05 m	3 m	90º	1,71 m	2.png	Folio detectado.
				90º	1,85 m	3.png	Javier con el folio delante de las piernas.
				90º	1,83 m	4.png	Javier con el folio en el pecho.
				90º	1,78 m	5.png	Javier con el folio separado del pecho, moviéndolo más rápido.
				90º	1,63 m	6.png	Paulo con el folio en el pecho.
				90º	2,01 m	7.png	Javier (Primera versión del nuevo algoritmo y modelo).
				90º	1,93 m	8.png	Javier (Segunda versión del nuevo algoritmo y modelo).
				90º	1,68 m	9.png	Voluntaria: Yue.
				90º	1,77 m	10.png	Voluntario: Enrique.
				90º	1,79 m	11.png	Paulo con el folio delante del abdomen.
				90º	1,79 m	12.png	Darío con el folio torcido.
				90º	1,61 m	13.png	Voluntaria: Irene.

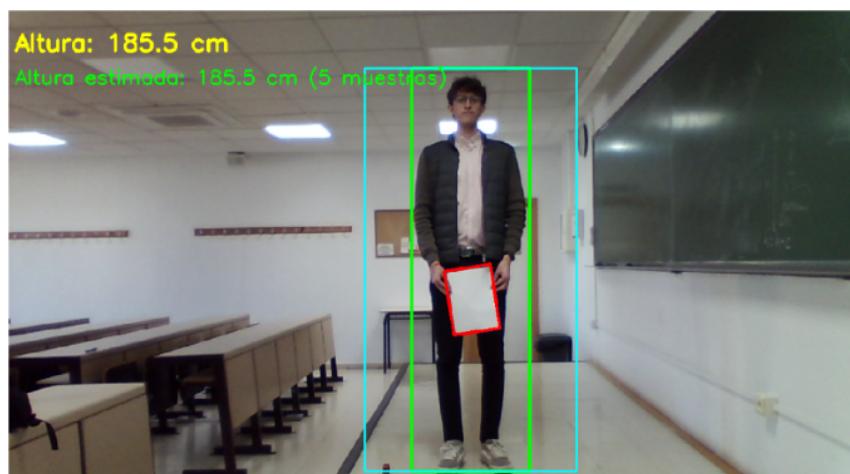
Imágenes del Anexo II



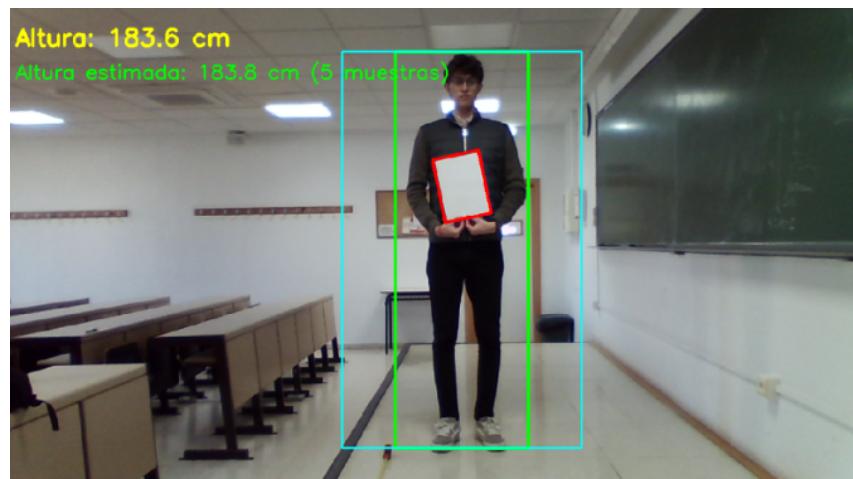
1.png



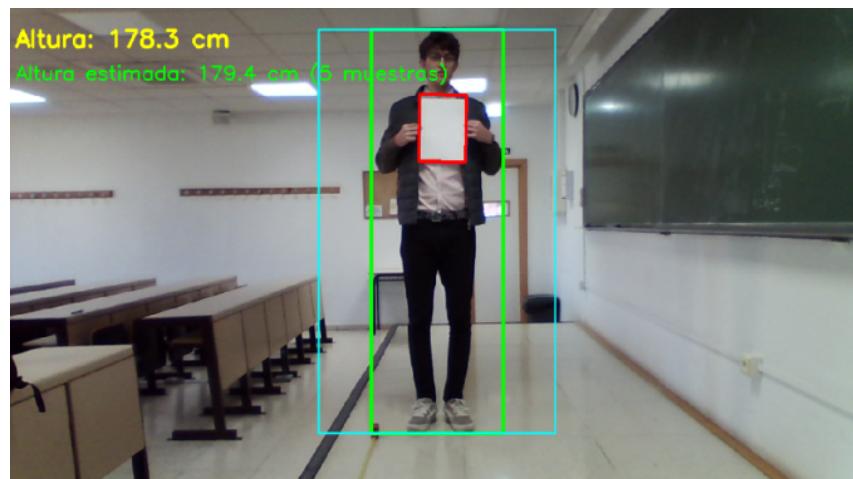
2.png



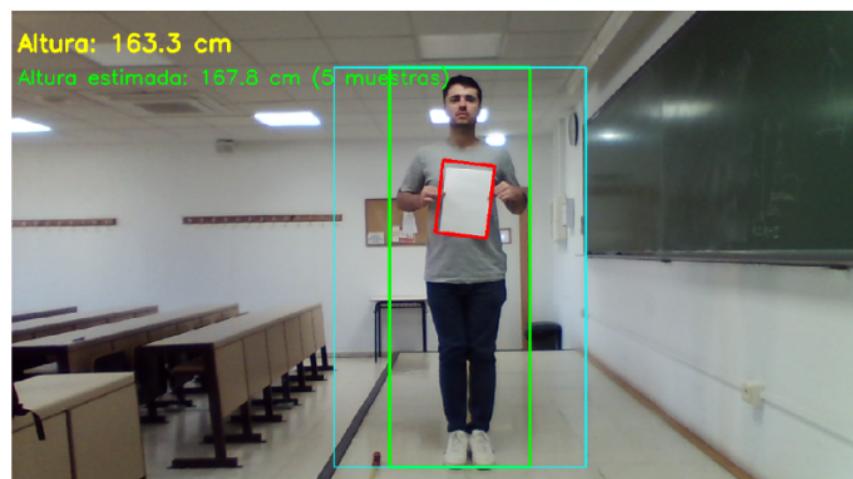
3.png



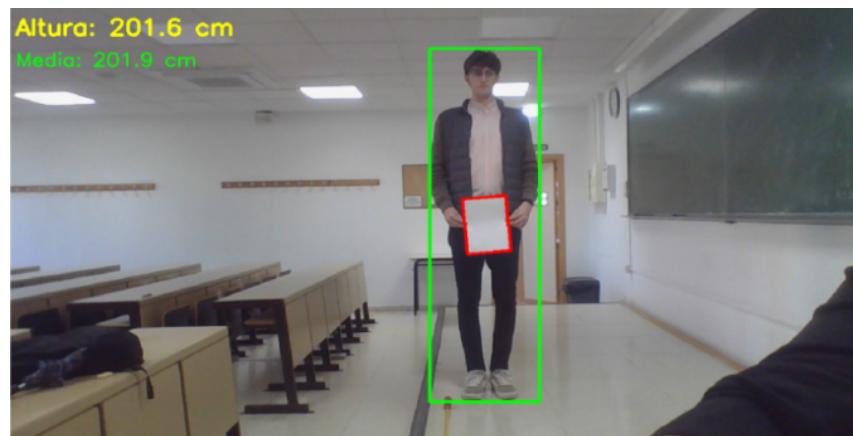
4.png



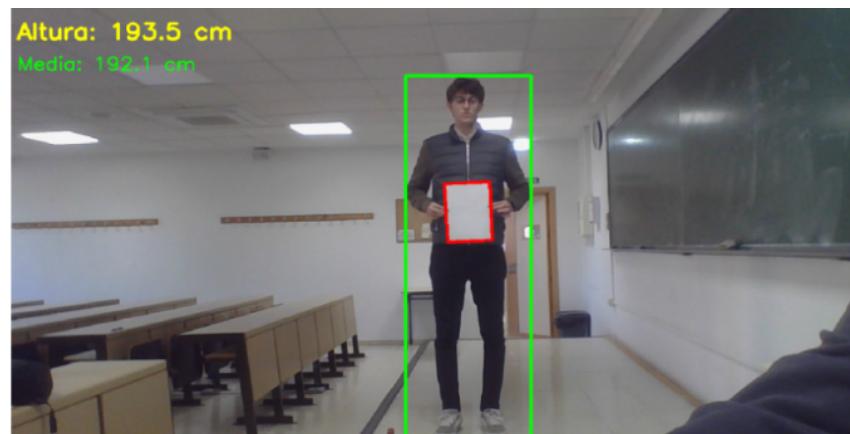
5.png



6.png



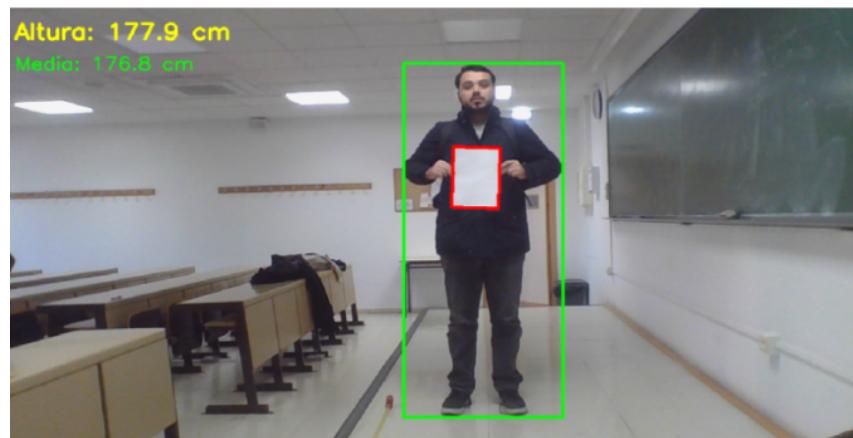
7.png



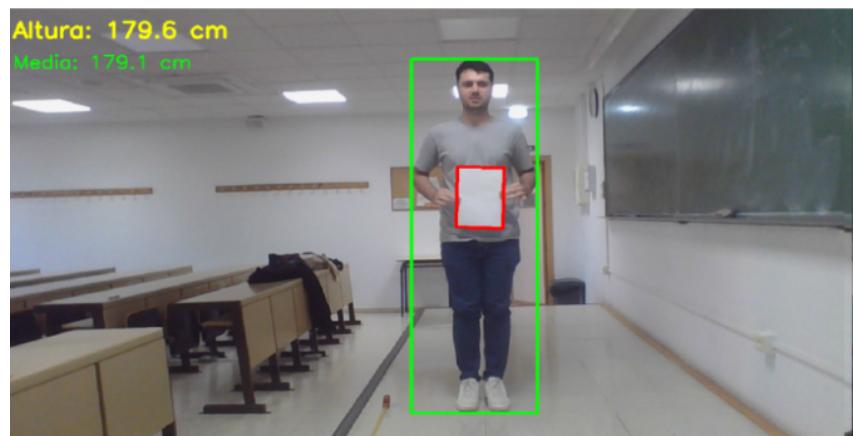
8.png



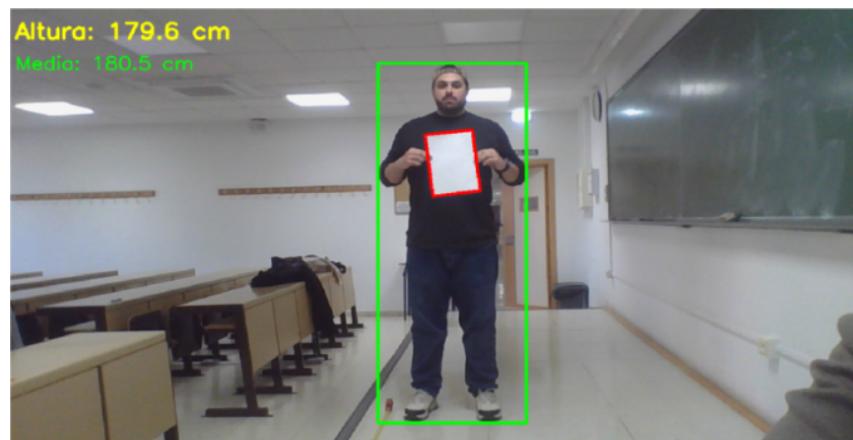
9.png



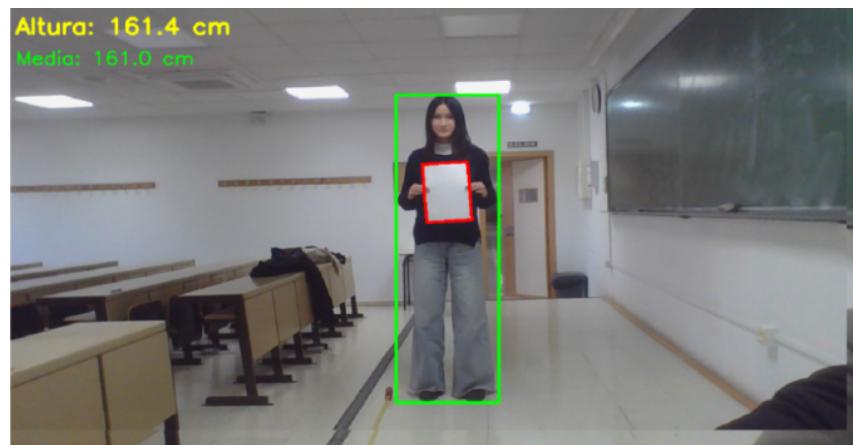
10.png



11.png



12.png



13.png