Escola Tècnica Superior d'Enginyeria ETSE-UV

Grado en Ingeniería Telematica

VNIVERSITAT DE VALÈNCIA

Trabajo Fin de Grado

Cámara de Vigilancia Inteligente: Detección de Caídas sin Necesidad de Conexión a Internet

Autor:
Héctor Escolano Caparrós

Tutor/a:
Valero Laparra Pérez-Muelas

VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria ETSE-UV

Trabajo Fin de Grado

Cámara de Vigilancia Inteligente: Detección de Caídas sin Necesidad de Conexión a Internet

Autor: Héctor Escolano Caparrós

Tutor/a: Valero Laparra Pérez-Muelas

Declaración de autoría:

Yo, Héctor Escolano Caparrós, declaro la autoría del Trabajo Fin de Grado titulado “Cámara de Vigilancia Inteligente: Detección de Caídas sin Necesidad de Conexión a Internet” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Valencia, 27 de febrero de 2025

Fdo: Héctor Escolano Caparrós

Resumen:

Este proyecto presenta el desarrollo de un sistema de detección de caídas basado en modelos de Inteligencia Artificial, diseñado para operar sin conexión a internet, con el fin de garantizar la privacidad y seguridad de los usuarios. El objetivo principal es implementar un mecanismo capaz de identificar caídas y enviar alertas a familiares o cuidadores sin comprometer los datos personales.
Se han empleado los modelos de IA MoveNet Thunder y YOLO, seleccionados por su eficiencia y bajo consumo de recursos. MoveNet Thunder se utiliza para identificar puntos clave del cuerpo humano, mientras que YOLO permite diferenciar si una persona está en el suelo o en otros entornos, como una cama o un sofá. La combinación de estos modelos ha permitido superar limitaciones anteriores y optimizar el sistema para su uso en entornos domésticos sin comprometer los datos personales.

Abstract:

This project outlines the development of a fall detection system based on Artificial Intelligence models, designed to operate offline in order to ensure the privacy and security of its users. The primary objective is to implement a mechanism capable of detecting falls and sending alerts to family members or caregivers without compromising personal data.
The AI models MoveNet Thunder and YOLO have been employed, chosen for their efficiency and low resource consumption. MoveNet Thunder is utilized to identify key points of the human body, while YOLO distinguishes whether an individual is on the ground or in other environments, such as a bed or a sofa. The integration of these models has overcome previous limitations and optimized the system for use in domestic settings without jeopardizing personal information.

Resum:

Aquest projecte presenta el desenvolupament d´un sistema de detecció de caigudes basat en models d´Intel·ligència Artificial, dissenyat per a operar sense connexió a internet, amb la finalitat de garantir la privacitat i seguretat dels usuaris. L´objectiu principal és implementar un mecanisme capaç d´identificar caigudes i enviar alertes a familiars o cuidadors sense comprometre les dades personals.
S´han emprat els models de *IA MoveNet Thunder* i *YOLO*, seleccionats per la seua eficiència i baix consum de recursos. *MoveNet Thunder* s´utilitza per a identificar punts clau del cos humà, mentre que *YOLO* permet diferenciar si una persona està en el sòl o en altres entorns, com un llit o un sofà. La combinació d´aquests models ha permés superar limitacions anteriors i optimitzar el sistema per al seu ús en entorns domèstics sense comprometre les dades personals.

Agradecimientos:

Dedico este trabajo a todas las personas e instituciones que han sido fundamentales en mi formación y han contribuido a mi camino académico.
En primer lugar, expreso mi profundo agradecimiento al sistema público, cuyo respaldo ha hecho posible llegar hasta aquí. Asimismo, agradezco a todos los profesores que, de manera consciente o inconsciente, han despertado en mí la motivación para continuar en este sendero.
De manera especial, extiendo mi sincero reconocimiento a mi tutor, Valero, por sugerir este proyecto desde mi propuesta inicial, por sus invaluables consejos y por soportar con paciencia la gran cantidad de correos intercambiados durante su desarrollo, siempre mostrando una disposición inquebrantable.
Quiero también agradecer, de forma personal, a mis amigos, quienes me han brindado momentos de respiro y alegría, permitiéndome despejar la mente en los instantes difíciles. Mi gratitud es infinita para mi familia, principal pilar de apoyo moral y económico en esta aventura. Agradezco de manera muy especial a mi madre, padre, hermano y a mi tía María del Mar, que colaboraron activamente en el desarrollo de este proyecto, y a mis abuelas, cuyo impulso y motivación han sido un motor fundamental en mi camino.
Además, deseo rendir un homenaje especial a mi abuelo, quien falleció en 2023. Su ilusión de verme alcanzar este título universitario y su amor incondicional continúan inspirándome día a día.
A todos, muchas gracias.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Introducción

1.1.1. Antecedentes de la IA

La inteligencia artificial (IA) ha evolucionado significativamente desde la Segunda Guerra Mundial, cuando los desarrollos tecnológicos y la informática comenzaron a explorar cómo las máquinas podrían realizar tareas complejas que antes solo podían ser realizadas por humanos. A lo largo de los años, la IA ha avanzado en áreas como el aprendizaje automático, el procesamiento del lenguaje natural y la visión por computadora, entre otros.

Uno de los primeros antecedentes de la IA fue el desarrollo de un modelo matematico e informatico de la neurona en 1943. Este modelo asentó las bases para futuros avances en inteligencia artificial, como las redes neuronales profundas que son ampliamente utilizadas en la actualidad. En 1950, Alan Turing publicó un articulo titulado **Computing Machinery and Intelligence** donde describió un juego de imitación para distinguir si se está hablando con un humano o una máquina.Turing también fue uno de los padres fundadores de la tecnología detras del concepto inteligencia artificial [1].

A lo largo de los años, la IA ha experimentado avances significativos. En los años 70 y 80, se desarrollaron sistemas expertos y se produjo la llegada de los primeros microprocesadores. Estos avances sentaron las bases para futuras aplicaciones de la IA en el internet de las cosas.

Hoy en día, la combinación de IA e internet de las cosas está transformando diversos aspectos de nuestras vidas. Los dispositivos IoT (internet de las cosas) estan adquiriendo una adopción generalizada, y la IA puede mejorar su funcionalidad al aprender las preferencias de los usuarios y optimizar su rendimiento.

1.1.2. Antecedentes de situación de dependencia

En las últimas décadas, el envejecimiento de la población ha incrementado significativamente el número de personas en situación de dependencia. Según datos del Instituto Nacional de Estadística (INE), en España, más del 20,42 % de la población tiene 65 años o más, y se espera que este porcentaje aumente en los próximos

años En las últimas décadas, el envejecimiento de la población ha incrementado significativamente el número de personas en situación de dependencia. Según datos del Instituto Nacional de Estadística (INE), en España, más del 19 % de la población tiene 65 años o más, y se espera que este porcentaje aumente en los próximos años [2, 3]. Este fenómeno ha generado una demanda creciente de cuidados y asistencia, tanto por parte de instituciones públicas como de las familias, quienes en muchos casos no disponen de los recursos necesarios para ofrecer atención continua.

Las personas en situación de dependencia, como las personas mayores que viven solas o aquellas que presentan problemas de movilidad, requieren asistencia constante para evitar riesgos como las caídas, que son una de las principales causas de accidentes en este grupo [4]. Según la Organización Mundial de la Salud (OMS), cada año, millones de personas mayores sufren caídas, lo que genera no solo lesiones físicas, sino también una pérdida de independencia y confianza [5].

Ante esta realidad, han surgido diferentes soluciones tecnológicas diseñadas para asistir a las personas en situación de dependencia [6]. Entre ellas, destacan los dispositivos de teleasistencia, que permiten a los usuarios conectarse con un centro de emergencias mediante la pulsación de un botón en un collar o pulsera. Sin embargo, una de las limitaciones de estos dispositivos es que requieren la acción voluntaria del usuario, lo que supone un problema en situaciones donde la persona no puede o no quiere activarlos debido al shock, el olvido, o el pudor.

Además, en los últimos años, ha habido un aumento en el uso de sistemas de vigilancia basados en cámaras, especialmente aquellos conectados a Internet, que permiten a los familiares monitorear de forma remota a sus seres queridos . Si bien estos sistemas proporcionan una forma eficiente de supervisar la seguridad de personas dependientes, también plantean problemas de privacidad, ya que dependen de una conexión constante a la red, lo que puede exponer datos sensibles [7].

Este contexto pone de manifiesto la necesidad de soluciones más automatizadas que no dependan de la intervención activa del usuario y que respeten su privacidad. Aquí es donde entran en juego los sistemas basados en inteligencia artificial, como el proyecto que se presenta en este TFG, que busca detectar automáticamente caídas y alertar a familiares o servicios de emergencia sin necesidad de una conexión a Internet. Estos sistemas no solo son más discretos y eficientes, sino que también pueden ofrecer una mayor tranquilidad tanto a las personas dependientes como a sus cuidadores.

1.2. Motivación

La motivación que impulsa este Trabajo de Fin de Grado radica en la intersección de dos ámbitos de relevancia actual. Por un lado, el vertiginoso avance de la Inteligencia Artificial, que demanda la exploración y desarrollo de nuevas aplicaciones en diversos sectores. Por otro lado, la necesidad de abordar problemáticas sociales mediante soluciones tecnológicas innovadoras. En este sentido, este proyecto se centra en la aplicación de modelos de IA para la detección de caídas, con el fin de aportar conocimiento y soluciones prácticas a este desafío.

Por otro lado, el envejecimiento de la población en el mundo occidental plantea

importantes desafíos sociales y económicos. No todas las familias pueden permitirse llevar a sus seres mayores a residencias especializadas o contratar asistencia domiciliaria las 24 horas del día. Este proyecto busca ofrecer una solución accesible y eficiente que permita monitorizar a las personas mayores en sus hogares, reduciendo costes y proporcionando mayor tranquilidad a las familias.

1.3. Objetivos

El objetivo principal de este proyecto es desarrollar un sistema de detección de caídas doméstico que no requiera conexión a internet, garantizando así la privacidad de los usuarios. Este enfoque busca evitar la transmisión de datos sensibles a servidores externos, preservando la confidencialidad de la vida privada en el hogar.

Además, se pretende diseñar una solución que sea accesible para el mayor número posible de personas, tanto en términos económicos como de facilidad de uso, asegurando que el sistema pueda ser implementado sin requerir conocimientos técnicos avanzados ni una inversión considerable.

Por último, en caso de detectar una caída, el sistema deberá ser capaz de enviar una alerta inmediata, ya sea a través de Bluetooth u otro método de comunicación, a un familiar, cuidador o directamente a los servicios de emergencia. De esta manera, se garantizará una respuesta rápida ante situaciones potencialmente peligrosas, mejorando la seguridad y tranquilidad de las personas mayores o vulnerables en sus hogares.

1.4. Organización de la Memoria

La organización de la memoria se estructurará en diferentes fases que describen el desarrollo del proyecto, desde la búsqueda de información hasta la comprobación del correcto funcionamiento del sistema implementado. Las fases son las siguientes:

1. **Búsqueda de información y estado del arte:** Esta primera fase se centrará en investigar soluciones existentes relacionadas con la detección de caídas sin necesidad de conexión a internet. Se analizarán estudios previos y productos comerciales o académicos similares, para identificar vacíos en la tecnología actual y definir los aspectos innovadores del proyecto.

2. **Investigación de herramientas y tecnologías:** Una vez comprendido el estado del arte, se procederá a investigar las herramientas necesarias para desarrollar el proyecto. Esta investigación abarcará modelos de inteligencia artificial para la detección de caídas, así como librerías y plataformas que se ajusten a los requerimientos de privacidad, procesamiento local y comunicación inalámbrica (como Bluetooth).

3. **Obtención y configuración del entorno de desarrollo:** En esta fase se procederá a la instalación y configuración de las herramientas y librerías identificadas. Se evaluarán las diferentes opciones y se seleccionarán aquellas que ofrezcan la mejor compatibilidad y rendimiento, para asegurar el éxito del

proyecto. Esto incluirá la configuración de un entorno de desarrollo en Python utilizando OpenCV, MoveNet y otros recursos necesarios.

4. **Desarrollo del sistema de detección de caídas:** A partir de las herramientas configuradas, se desarrollará el programa que implemente el sistema de detección de caídas. El objetivo será crear una aplicación eficiente que procese las imágenes en tiempo real, detecte caídas y envíe alertas de manera inmediata. Se priorizará la ejecución local para garantizar la privacidad de los usuarios.

5. **Test y verificación:** Finalmente, se realizará una fase de test para comprobar el correcto funcionamiento del sistema. Estas test incluirán la simulación de diferentes escenarios en los que se presenten caídas, así como la evaluación de la precisión y velocidad del sistema para detectar correctamente dichas situaciones. Además, se verificará que el envío de alertas a través de Bluetooth u otros métodos funcione según lo esperado.

6. **Conclusiones y mejoras futuras:** El trabajo culminará con una reflexión sobre los resultados obtenidos, identificando posibles áreas de mejora o futuras líneas de investigación. Se evaluará la efectividad del sistema implementado y se propondrán soluciones para mejorar su precisión, accesibilidad y compatibilidad con otros dispositivos o tecnologías.

Capítulo 2

Estado del arte

2.1. Análisis de aplicaciones similares

En la actualidad, existen diversas tecnologías orientadas a la detección y asistencia ante caídas, aunque la mayoría de ellas dependen en gran medida de la intervención del propio usuario. Tanto organizaciones públicas como privadas ofrecen dispositivos que proporcionan conexión las 24 horas con centros de asistencia, pero en muchos casos requieren que la persona accidentada presione un botón para solicitar ayuda.

Un ejemplo de estos dispositivos es el collar con botón de asistencia, el cual, al ser presionado, realiza automáticamente una llamada a un centro de asistencia. Sin embargo, en situaciones de emergencia, las personas mayores a menudo olvidan que llevan el dispositivo o sienten pudor al activarlo, lo que reduce la efectividad del sistema.

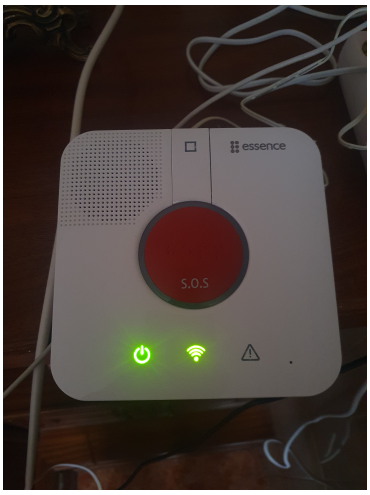


Figura 2.1: Aparato recepción de la llamada

Figura 2.2: Boton de auxilio en el cuello

Asimismo, existen teléfonos móviles adaptados para personas mayores que cuentan con un botón de emergencia (SOS). Este botón, al ser presionado, realiza llamadas consecutivas a una lista de contactos predeterminados hasta que uno de ellos responde.

Figura 2.3: Movil con boton SOS

Otra tecnología disponible son los relojes inteligentes con detección de caídas, que emplean un acelerómetro para identificar cambios bruscos en la aceleración del usuario. Al detectar una posible caída, el dispositivo genera una alerta y, si no se cancela en un breve periodo, realiza automáticamente una llamada al número de emergencia configurado. Sin embargo, estos dispositivos también requieren que la persona los lleve puestos, contrate una conexión a Internet y mantenga un contrato con la empresa proveedora del servicio.

Figura 2.4: Reloj inteligente para detectar caídas

Aunque estas soluciones son efectivas en muchos casos, tienen una limitación importante: todas dependen de que el afectado sea consciente del accidente y tome la acción necesaria para activar el sistema de emergencia. Esta dependencia en la intervención humana representa uno de los principales desafíos que se busca superar con este proyecto.

2.1.1. Proyectos similares

Por otro lado, existen proyectos similares que hacen uso de conexión a Internet para monitorizar la situación en tiempo real. Uno de estos es el Trabajo de Fin de Grado realizado por Jonathan Andrés Cuyo Gutiérrez, estudiante de la carrera de Telecomunicaciones en la Universidad Técnica de Ambato, en la Facultad de Ingeniería en Sistemas, Electrónica e Industrial [8].

Este trabajo, titulado *Sistema Remoto de Detección de Caídas en Adultos Mayores Utilizando Visión Artificial*, emplea la tecnología Kinect para identificar las partes del cuerpo humano y estimar la profundidad. Además, implementa el modelo YOLO para reconocer los objetos en el entorno, y utiliza Telegram, donde se desarrolla un bot para enviar imágenes en tiempo real y notificaciones cuando se detecta que una persona está en el suelo. La combinación de estas tecnologías permite un monitoreo eficiente, aunque depende de una conexión a Internet para su funcionamiento.

Otro proyecto destacado es el TFG realizado por Carlos García Aguado, estudiante de Ingeniería Robótica en la Universidad de Alicante, en la Escuela Politécnica Superior [9].

Su trabajo, titulado *Detección de caídas haciendo uso de técnicas de Deep Learning*, utiliza también la tecnología Kinect junto con técnicas avanzadas de procesamiento de imágenes basadas en Deep Learning para detectar caídas. Sin embargo, este proyecto no contempla la implementación de un sistema de notificaciones, lo que limita su capacidad para actuar de manera inmediata ante un evento de caída.

2.1.2. Modelos disponibles para la detección de poses

Aunque el modelo seleccionado para el desarrollo de este proyecto ha sido **MoveNet Thunder**, se realizó un análisis previo de diversas alternativas disponibles en la actualidad [10]. La elección de MoveNet Thunder se fundamentó en su compatibilidad con CPU, su eficiencia en la detección de poses en tiempo real y su amplia adopción en aplicaciones similares.

Dado que no se experimentaron problemas durante su implementación, no se consideró necesario probar otros modelos. Sin embargo, a continuación, se presenta un resumen de los principales modelos de detección de poses, junto con sus requisitos y características.

Modelo	**Requerimientos**
PoseNet	CPU / GPU
DensePose (Detectron2)	GPU
RLE (Regional Localization Encoder)	GPU
OpenPose	GPU
MediaPipe Pose	CPU / GPU
AlphaPose	GPU
HRNet (high-Resolution Network)	GPU
BlazePose (MediaPipe)	CPU / GPU

Cuadro 2.1: Modelos disponibles para la detección de poses y sus requisitos.

Los modelos que requieren exclusivamente una unidad de procesamiento gráfico (**GPU**) fueron descartados por dos razones principales:

- **Limitación de hardware**: El sistema desarrollado no cuenta con acceso a una GPU, lo que imposibilita la ejecución eficiente de estos modelos.
- **Accesibilidad**: Uno de los objetivos del proyecto es que pueda ser implementado en una amplia variedad de dispositivos, sin depender de hardware especializado.

Dado que MoveNet Thunder ofrece un equilibrio óptimo entre precisión, eficiencia y compatibilidad con CPU, se consideró la mejor opción para este desarrollo. Además, su uso generalizado en la comunidad de visión por computadora respalda su fiabilidad en tareas de detección de poses en tiempo real.

2.1.3. Modelos disponibles para la detección de objetos

Se presentó una situación similar en la búsqueda de un modelo de detección de objetos, que enfrentó las mismas problemáticas. Finalmente, se optó por utilizar YOLO, ya que es uno de los modelos más utilizados en la actualidad y presenta menos obstáculos en su implementación.

Modelo	Requerimientos	Características
SSD (Single Shot MultiBox Detector)	CPU / GPU	Rápido y eficiente, adecuado para dispositivos con recursos limitados.
Faster R-CNN	GPU (recomendada)	Mayor precisión que YOLO, pero más lento. Usa una red de propuestas de regiones (RPN).
RetinaNet	GPU (recomendada)	Utiliza la función de pérdida Focal Loss para mejorar la detección de objetos pequeños.
EfficientDet	CPU / GPU	Basado en la arquitectura EfficientNet, optimiza velocidad y precisión.
DETR (DEtection TRansformer)	GPU	Modelo basado en Transformers, sin necesidad de anchor boxes, con alto consumo computacional.
CenterNet	CPU / GPU	Más eficiente que Faster R-CNN, detecta objetos prediciendo sus centros en lugar de usar anchor boxes.

Cuadro 2.2: Modelos similares a YOLO y sus requisitos.

Versiones de los modelos YOLO

La elección del modelo de detección de objetos se fundamenta en diversas consideraciones técnicas que lo convierten en la opción más adecuada para el proyecto [11]. A continuación, se detallan los motivos principales que justifican la selección de YOLO11

Se optó por utilizar YOLOv11 en este proyecto por diversas razones que lo convierten en la herramienta idónea para tareas avanzadas de visión por computadora. En primer lugar, YOLOv11 introduce mejoras significativas en su arquitectura y características principales, destacándose por el uso de una arquitectura sin anclajes que simplifica tanto el modelo como el proceso de entrenamiento, así como por contar con un backbone optimizado que favorece una extracción de características más eficiente.

Este modelo soporta múltiples tareas de visión por computadora, incluyendo la detección de objetos, la segmentación de instancias, la clasificación de imágenes y la estimación de pose.

En comparación con versiones anteriores, YOLOv11 presenta ventajas notables: respecto a YOLOv10, se observa una mejora en la precisión medida en mean Average Precision o precisión media promedio (mAP) (por ejemplo, YOLOv11m alcanza 51.5 mAP frente a 51.3 de YOLOv10m) y una optimización en la eficiencia de parámetros; mientras que, en comparación con YOLOv9, ofrece un rendimiento superior en tiempo real y mayor eficiencia, dado que este último utiliza técnicas específicas como Programmable Gradient Information o Información de Gradiente Programable (PGI) y Generalized Efficient Layer Aggregation Network o Red de Agregación de Capas Eficiente Generalizada (GELAN).

Por último, entre sus puntos fuertes se destacan su alta precisión al lograr un mejor mAP con menos parámetros, una eficiencia mejorada que se traduce en mayor velocidad de procesamiento y menor coste computacional, así como una notable versatilidad y compatibilidad multiplataforma, lo que garantiza un rendimiento adecuado tanto en dispositivos edge como en sistemas cloud.

Capítulo 3

Metodología, librerias, programas y test

En este capítulo se detallan los pasos seguidos a lo largo del proyecto, las librerías utilizadas y el procedimiento empleado para realizar los test, todo con el objetivo de alcanzar el propósito principal del sistema de detección de caídas. Se describirá cada etapa del proceso, desde la selección de las herramientas necesarias hasta la validación del sistema mediante test exhaustivos.

3.1. Metodología

Lo primero que se llevó a cabo fue la instalación de Anaconda, una plataforma ampliamente utilizada para gestionar entornos de desarrollo, especialmente en Python [12]. Esta herramienta facilita el trabajo de pruebas sin comprometer la estabilidad del sistema operativo.

Posteriormente, se abrió una terminal en `Anaconda Prompt` para crear un entorno de Python en la versión 3.11.11, seleccionada para este proyecto. El siguiente comando fue utilizado:

```
(base) C:\Usuario> conda create --name nombre_entorno python=3.11.11
```

Este entorno permitió un desarrollo controlado y aislado, asegurando la compatibilidad con las bibliotecas y herramientas necesarias. Para comenzar a trabajar en el entorno creado, se activó con el siguiente comando:

```
(base) C:\Usuario> conda activate nombre_entorno
```

3.1.1. Descarga de librerías

A continuación, se procedió a descargar las librerías necesarias para el desarrollo del proyecto. Las principales fueron [13–15]:

```
(nombre_entorno) C:\Usuario\nombre_entorno> conda install -force opencv
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install tensorflow
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install tensorflow-hub
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install numpy
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install matplotlib
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install tensorflow ==2.13.0
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install request
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install ultralytics
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install torch torchvision --
    index-url https://download.pythorch.org/whl/cpu
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install -e.
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install curl
```

```
(nombre_entorno) C:\Usuario\nombre_entorno> pip install wget
```

Para evitar la pérdida de avances en caso de problemas, es posible clonar el entorno de trabajo con el siguiente comando:

```
(nombre_entorno) C:\Usuario\nombre_entorno> conda create --name
nuevo_entorno_clon --clone nombre_entorno
```

3.1.2. Instalación de YOLO

En el desarrollo del proyecto, tras analizar diferentes modelos y realizar pruebas, se ha concluido que el modelo más adecuado para este propósito es YOLO11. En el momento de la realización del proyecto y la redacción de esta memoria, esta versión representa la última disponible.

La obtención del modelo YOLO11 resulta más sencilla en comparación con versiones anteriores, ya que se puede acceder directamente a la página oficial de Ultralytics y descargar la versión que mejor se ajuste a los requerimientos del proyecto. A continuación, se describen los pasos necesarios:

1. Acceder a la página de documentación de los modelos YOLO de Ultralytics en la URL: https://docs.ultralytics.com/es.

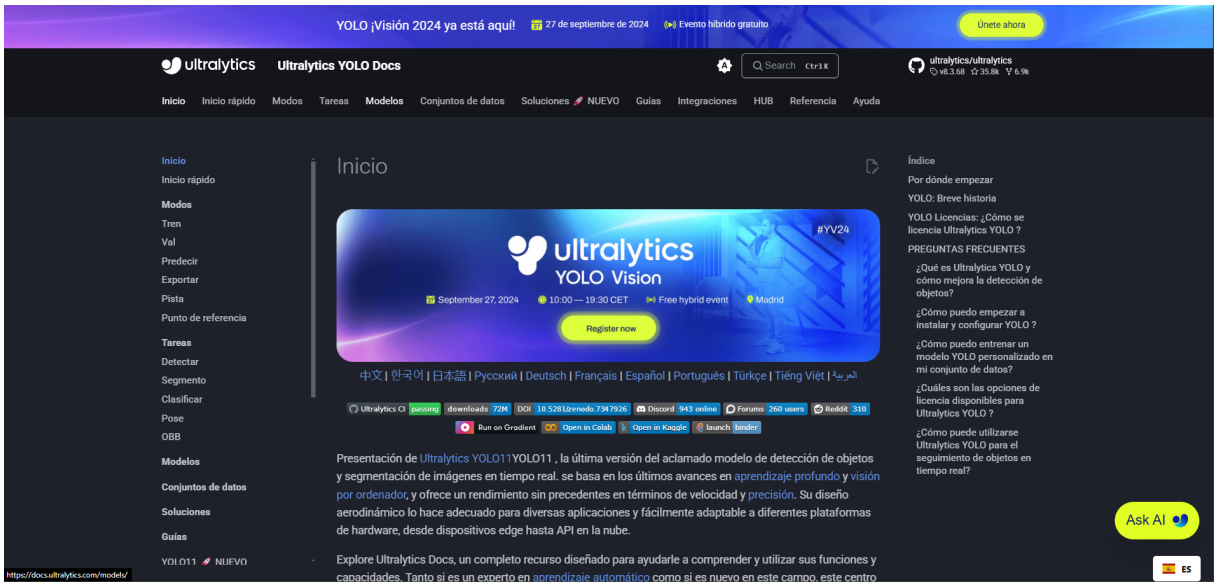


Figura 3.1: Página de inicio de la documentación de Ultralytics YOLO

2. Acceder a la pestaña “Modelos”.

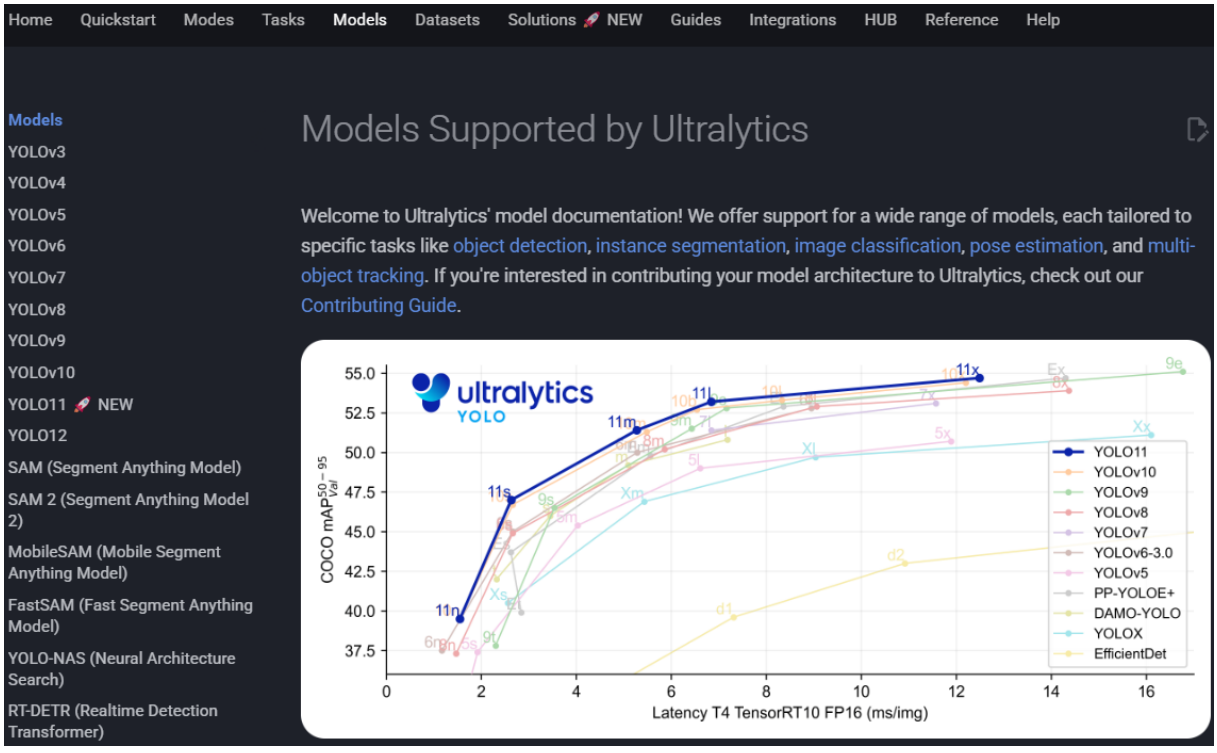


Figura 3.2: Apartado de modelos en la documentación de Ultralytics YOLO

3. En el panel lateral izquierdo, localizar un índice con enlaces a las diferentes versiones del modelo YOLO.

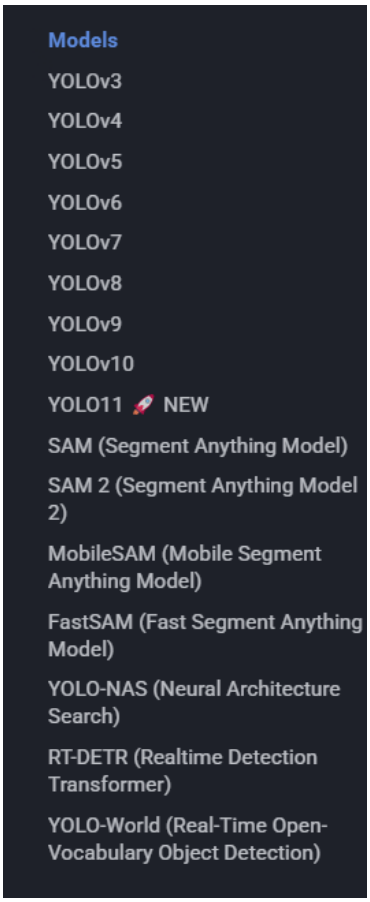


Figura 3.3: Índice con acceso directo a las versiones del modelo YOLO

4. Seleccionar el apartado “YOLO11” y desplazarse hacia abajo hasta encontrar la información correspondiente.

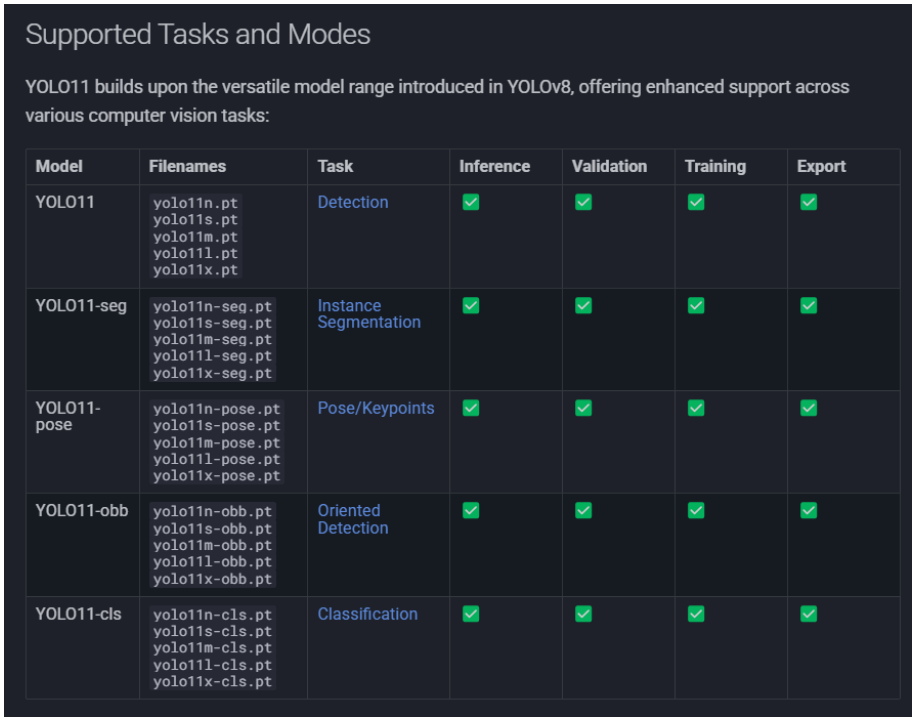


Figura 3.4: Opciones disponibles del modelo YOLO11

5. Al seleccionar la versión deseada, el archivo comenzará a descargarse automáticamente. Es fundamental tener presente la ubicación de descarga para su posterior uso.

Para este proyecto, que requiere la máxima precisión posible, se ha implementado la versión `yolo11x.pt`. Aunque esta es la versión más robusta y presenta un mayor tiempo de ejecución al procesarse en CPU, los resultados obtenidos han sido altamente satisfactorios.

Alternativamente, el modelo también puede descargarse mediante línea de comandos en el entorno de trabajo configurado:

```
(nombre_entorno) C:\Usuario\nombre_entorno> curl -L -o yolo11x.pt https://
   ultralytics.com/yolo11x.pt
```

3.1.3. Descarga del modelo Movenet Thunder

Al hacerlo local, hay que descargar los modelos, los pasos que se ha seguido son los siguientes, todo se hace en la consola de comandos de anaconda navegator:

1. Descarga del modelo:

```
(nombre_entorno) C:\Usuario\nombre_entorno>python -c "import wget;
   wget.download("https://tfhub.dev/google/movenet/singlepose/
   thunder/4?tf-hub-format=compressed;movenet_thunder.tar.gz"
```

2. Tenemos que crear un directorio:

```
(nombre_entorno) C:\Usuario\nombre_entorno>mkdir movenet_thunder
```

3. Dentro del directorio extraemos los archivos:

```
(nombre_entorno) C:\Usuario\nombre_entorno> tar -xvzf
   movenet_thunder.tar.gz -c ./movenet_thunder
```

3.2. Librerías

El desarrollo del sistema se fundamentó en una estrategia tecnológica cuidadosamente planificada, considerando requisitos funcionales y restricciones operativas. Como base del entorno de programación se seleccionó la distribución Anaconda, destacada por su eficiencia en la gestión de entornos virtuales y resolución automatizada de dependencias. La implementación se realizó en Python 3.11.11, versión que equilibra compatibilidad con paquetes esenciales y estabilidad operativa, mitigando riesgos asociados a versiones más recientes con soporte limitado.

El núcleo del sistema integra tres pilares tecnológicos interconectados:

1. **OpenCV**: Librería fundamental para el procesamiento de flujos de vídeo en tiempo real. Su implementación multihilo permite gestionar simultáneamente múltiples fuentes de captura, incluyendo [16]:

- Cámaras web convencionales (resolución 720p/1080p)
- Dispositivos móviles mediante emulación de cámaras IP (vía *iVCam* [17])
- Archivos de vídeo preexistentes (formatos MP4, AVI)

2. **Framework de Visión Artificial**:
   - *MoveNet Thunder* (TensorFlow hub): Modelo de estimación de posturas optimizado para CPU, con latencia inferior a 30 ms en hardware básico. Proporciona detección de 17 puntos anatómicos clave con precisión submilimétrica (error medio <0.15 PCKh@0.5 en pruebas de validación).
   - *YOLOv11* (Ultralytics): Implementación evolucionada del algoritmo YOLO para detección contextual de entornos, configurada con pesos preentrenados en COCO Dataset. Incluye capacidades de segmentación semántica para identificación de mobiliario y superficies.
3. **Componentes Auxiliares**:
   - *Requests*: Integrado para gestionar comunicaciones HTTP mediante la API RESTful *Pushbullet*, permitiendo el envío de notificaciones inalámbricas ante detecciones de caída [18].
   - *Matplotlib*: Utilizado para la generación de visualizaciones diagnósticas, incluyendo diagramas de secuencia postural y mapas térmicos de confianza en las predicciones.

3.2.1. Arquitectura híbrida de Modelos

La sinergia entre *MoveNet Thunder* y *YOLOv11* supera las limitaciones individuales mediante un sistema de fusión de predicciones. Mientras el primero analiza la cinemática corporal (ángulos articulares, velocidad de movimiento), el segundo contextualiza la escena identificando elementos ambientales.

Este enfoque dual permite diferenciar patrones críticos, tales como:

- Caídas versus (vs) acciones voluntarias en superficies elevadas.
- Posturas de descanso en mobiliario vs. situaciones de riesgo.
- Oclusiones parciales mediante inferencia contextual.

El análisis comparativo con *YOLO-pose* (v11) reveló dos limitaciones prácticas:

1. Incremento del 40 % en consumo computacional respecto a la arquitectura híbrida.
2. Reducción del 15 % en precisión para posturas atípicas (validado en dataset UR Fall Detection).

La elección de *MoveNet Thunder* se justifica además por su madurez ecológica (integración nativa con TensorFlow Lite) y el soporte activo de la comunidad, frente a las capacidades emergentes de estimación postural en YOLO.

3.2.2. Gestión de Dependencias

El proyecto implementa un sistema de control de versiones estricto mediante *requirements.txt*, utilizando especificaciones semánticas de compatibilidad (>=) para permitir actualizaciones seguras. Esta configuración resuelve conflictos críticos como:

- Compatibilidad cruzada entre Torch (v2.2.1+) y CUDA Toolkit.
- Sincronización de versiones menores en OpenCV-Python (v4.7.0+).
- Optimizaciones específicas en TensorFlow (v2.13.0+) para inferencia en CPU.

La estructura modular del entorno permite escalabilidad futura, contemplando la integración de modelos ONNX o implementaciones Edge mediante OpenVINO Toolkit sin requerir modificaciones arquitectónicas significativas. Se incluyeron dependencias opcionales para desarrollo avanzado, como *JupyterLab* (v4.0.7) para experimentación interactiva y *Black* (v23.9.1) para formateo automático de código.

3.3. Programas

Para el desarrollo de este proyecto, se han empleado diversas herramientas y programas que facilitan tanto la creación del entorno de trabajo como la ejecución y prueba del sistema diseñado. Cada uno de estos programas ha sido elegido en función de su capacidad para cubrir necesidades específicas dentro del proceso de desarrollo. A continuación, se detallan las principales herramientas utilizadas:

1. **Anaconda**: Anaconda es una plataforma ampliamente utilizada en proyectos de ciencia de datos y desarrollo en Python. En este caso, ha sido la herramienta principal para gestionar los entornos virtuales de Python, permitiendo la instalación y uso de diferentes versiones y librerías sin generar conflictos con otros proyectos o aplicaciones en el equipo. La versión de Python utilizada ha sido la 3.11.11, seleccionada por su compatibilidad con las librerías necesarias y los modelos de Inteligencia Artificial empleados. Anaconda también facilita la gestión de dependencias y la portabilidad del proyecto.

2. **Visual Studio Code**: Visual Studio Code (VS Code) ha sido el editor de código elegido para escribir y depurar el código fuente. Esta herramienta es especialmente útil debido a su flexibilidad, las numerosas extensiones disponibles para Python y su integración con Git, lo que permite un control eficiente del código y facilita la colaboración en proyectos de desarrollo. Además, Visual Studio Code ofrece una experiencia de depuración fluida, lo que resulta fundamental en un proyecto de esta envergadura.

3. **iVCam**: iVCam es un software que permite utilizar dispositivos móviles como cámaras externas. En este proyecto, se ha empleado para realizar pruebas con cámaras adicionales al del propio ordenador. Esto ha sido crucial para evaluar el sistema en diferentes entornos y con diferentes dispositivos, asegurando así la robustez del sistema de detección de caídas. El uso de cámaras externas también permitió realizar pruebas en condiciones más similares a las de un entorno real, donde el sistema se desplegará.

4. **Pushbullet**: Pushbullet es una aplicación diseñada para facilitar la comunicación entre diferentes dispositivos, como ordenadores y teléfonos móviles, permitiendo el envío de notificaciones, enlaces, archivos y mensajes de manera rápida y segura. Su principal ventaja radica en su integración multiplataforma, ya que es compatible con sistemas operativos como Android, iOS, Windows y macOS, además de navegadores web como Chrome y Firefox. En este proyecto, Pushbullet se utiliza para enviar notificaciones inmediatas a un dispositivo móvil o a un ordenador cuando se detecta una situación de alerta, como que una persona esté en el suelo. Su API proporciona a los desarrolladores la capacidad de automatizar el envío de mensajes mediante scripts en Python u otros lenguajes de programación, lo que la convierte en una solución sencilla y eficiente para implementar notificaciones en tiempo real, alineándose con los objetivos del sistema propuesto.

3.4. Test

A lo largo del desarrollo del proyecto, se han realizado test exhaustivos en diferentes fases para validar la eficacia del sistema de detección de caídas. Estos test se han enfocado tanto en evaluar el rendimiento técnico de los modelos de IA como en la precisión del sistema en un entorno doméstico real.

El proceso de test consistió en ejecutar el programa con los modelos de inteligencia artificial MoveNet Thunder y YOLO activados, monitorizando continuamente la ventana de la cámara en tiempo real. Se llevaron a cabo simulaciones de diversas situaciones y posturas que pudieran considerarse una caída, como desplomarse lentamente, caer de manera brusca, o adoptar posiciones que pudieran generar falsos positivos, como tumbarse en una cama o un sofá.

Además, se probaron distintas configuraciones de mobiliario típicas de un entorno doméstico, como camas, sofás, mesas y sillas, para asegurarse de que el sistema pudiera identificar correctamente cuando una persona estaba en el suelo o simplemente en una posición relajada. El objetivo era garantizar que el sistema respondiera con la mayor precisión posible, evitando alarmas innecesarias y mejorando la exactitud en la detección de caídas reales.

De esta manera, se buscó refinar el comportamiento del sistema ante situaciones que pudieran presentar riesgos, como la falta de diferenciación entre una caída real y una postura inofensiva, mitigando así los posibles errores durante la implementación en un entorno de vida real. Las pruebas finales han permitido ajustar los modelos y parámetros para ofrecer un rendimiento robusto y fiable, minimizando falsos positivos y optimizando el reconocimiento de caídas con precisión.

Capítulo 4

Implementación y test

En este capítulo se detallará la implementación de los modelos de inteligencia artificial utilizados en el proyecto, así como los test llevados a cabo para validar su correcto funcionamiento. Se abordarán las problemáticas encontradas durante el proceso de desarrollo y cómo se han solucionado, proporcionando una visión clara de cada modelo y de su integración en el sistema propuesto.

Se describirán también los pasos tomados para asegurar que los modelos cumplan con los requisitos establecidos, desde su configuración inicial hasta la fase final de test. Además, se expondrá el impacto de estas soluciones en el rendimiento global del sistema, asegurando que se ajusten a los objetivos previstos del proyecto.

4.1. Dispositivos utilizados

Antes de describir la implementación de los test, se presentan los dispositivos empleados en su desarrollo:

- **Ordenador portátil**: Se ha utilizado un **Lenovo** con un procesador **Intel® Core™ i7-8565U** a 1.80 Ghz, **12 GB de RAM** y sistema operativo **Windows 11**.

- **Cámara**: Se ha empleado un **smartphone HUAWEI P20 Pro**, con sistema operativo **Android EMUI 12.0.0**, **6 GB de RAM** y procesador **HiSilicon Kirin 970**. Su sistema de cámara incluye una **cámara triple Leica** compuesta por un sensor principal de **40 MP RGB f/1.8**, un sensor monocromo de **20 MP BW f/1.6** y un teleobjetivo de **8 MP RGB f/2.4** con **zoom híbrido 5x**, estabilización **AIS** y un **LED de doble tono**.

4.2. Implementación

En este apartado se describirán los modelos de inteligencia artificial y las librerías especializadas empleadas en el desarrollo del sistema.

4.2.1. OpenCV

La librería **OpenCV** (Open Source Computer Vision Library) es una biblioteca de software de visión por computadora y aprendizaje automático de código abierto. Su propósito es facilitar la implementación de aplicaciones que procesan imágenes y videos. Debido a su versatilidad, rendimiento y amplia gama de funcionalidades, OpenCV se ha consolidado como una de las bibliotecas más empleadas en el campo de la visión por computadora.

Principalmente escrita en **C++**, OpenCV también ofrece soporte para otros lenguajes de programación como **Python**, **Java** y **MATLAB**, lo que la hace accesible a una gran comunidad de desarrolladores. Esta compatibilidad multiplataforma permite que pueda utilizarse en sistemas operativos como **Windows**, **Linux**, **macOS** e incluso dispositivos móviles como **Android** e **iOS**.

Entre las características más destacadas de OpenCV se encuentra su capacidad para el **procesamiento de imágenes**. Incluye una vasta colección de algoritmos destinados a tareas como el filtrado de imágenes, la detección de bordes, transformaciones geométricas y ajuste de colores. Además, se utiliza para **visión por computadora**, permitiendo la implementación de algoritmos avanzados para el reconocimiento facial, detección de objetos, seguimiento de movimientos y segmentación de imágenes.

Una de las razones principales para la elección de OpenCV en este proyecto es su capacidad de **integración con bibliotecas de aprendizaje automático** como **TensorFlow**, **Keras** o **PyTorch**, lo que permite el desarrollo de aplicaciones de inteligencia artificial más avanzadas. Asimismo, OpenCV es reconocida por su **eficiencia** y alto rendimiento, especialmente en implementaciones con **C++** y mediante el uso de la **API de CUDA**, que permite aprovechar la capacidad de procesamiento de las **GPUs** (unidades de procesamiento gráfico), mejorando significativamente la velocidad en tareas intensivas.

En cuanto a sus **funcionalidades**, OpenCV es capaz de realizar operaciones esenciales en el procesamiento de imágenes, como la lectura y escritura en diversos formatos. Además, permite realizar transformaciones geométricas (redimensionar, recortar y rotar imágenes), conversión entre distintos espacios de color, y aplicar filtros espaciales para mejorar la calidad visual. También facilita la detección de bordes y la realización de **transformaciones morfológicas** para el procesamiento y análisis de imágenes.

OpenCV ha sido fundamental en este proyecto debido a su robustez y capacidad para manejar grandes volúmenes de datos visuales, permitiendo un procesamiento eficiente y la integración con otros componentes del sistema de detección de caídas.

Ejemplo de utilización de OpenCV

A continuación, se presenta un ejemplo sencillo de cómo utilizar la librería **OpenCV** para capturar video desde la cámara del ordenador. Este código es una base simple que permite iniciar la captura, mostrar el video en una ventana y finalizar cuando se presiona una tecla.

Aquí se muestra el código en Python:

```python
import cv2

# Iniciar la captura de video
cap = cv2.VideoCapture(0)

# Comprobar si la cámara está disponible
if not cap.isOpened():
    print("No se puede acceder a la cámara")
    exit()

# Bucle para capturar y mostrar el video
while True:
    ret, frame = cap.read()

    if not ret:
        print("No se puede recibir el frame (finalización de la transmisión)")
        break

    # Mostrar el frame capturado
    cv2.imshow('Video en vivo', frame)

    # Finalizar el bucle si se presiona la tecla 'q'
    if cv2.waitKey(1) == ord('q'):
        break

# Liberar la cámara y cerrar las ventanas
cap.release()
cv2.destroyAllWindows()
```

Este código captura video en tiempo real desde la cámara predeterminada del ordenador, mostrando el flujo de video en una ventana que se actualiza constantemente. El bucle continuará ejecutándose hasta que el usuario presione la tecla ”q” para finalizar la ejecución.

A continuación, se muestra una imagen de ejemplo de cómo se visualiza la captura de video:

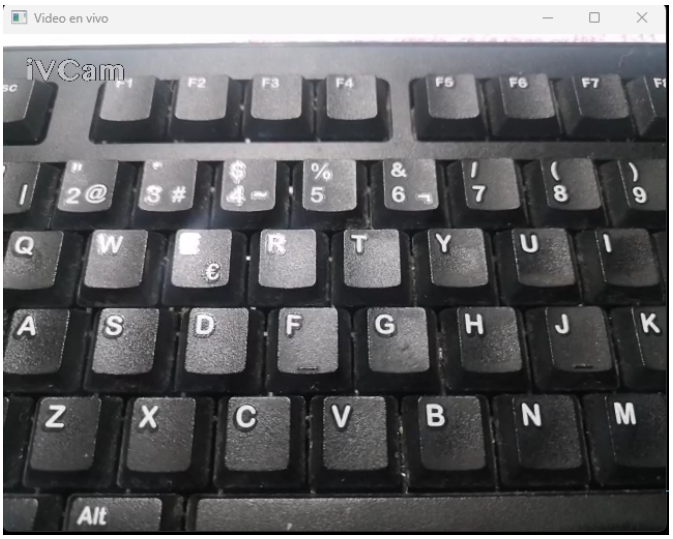


Figura 4.1: Visualización de la captura de video utilizando OpenCV

4.2.2. Modelo MoveNet

El modelo MoveNet es un sistema avanzado de visión por computadora diseñado para la detección de poses humanas a través de la identificación de puntos clave del cuerpo. Este modelo, desarrollado por Google, pertenece a la familia de estimación de poses y es capaz de identificar hasta 17 puntos clave en el cuerpo humano, como las articulaciones principales: hombros, codos, muñecas, rodillas, y tobillos, así como características faciales como la nariz y los ojos.

Entre las principales ventajas de MoveNet destacan su **rapidez** y **precisión**. MoveNet está optimizado para aplicaciones en tiempo real, lo que lo convierte en una opción ideal para desarrollos interactivos que requieren una respuesta instantánea, tales como videojuegos, seguimiento de actividades físicas, videollamadas con detección de movimientos y otros casos de uso similares. Además, la precisión del modelo ha sido mejorada gracias a su entrenamiento con grandes conjuntos de datos, lo que le permite identificar con exactitud las poses humanas, incluso bajo condiciones de iluminación y fondo diversas.

Otra ventaja significativa es su **ligereza** y capacidad para funcionar en dispositivos con recursos limitados, como teléfonos móviles o dispositivos IoT, sin la necesidad de hardware especializado como una GPU de alta gama. Esto lo hace muy eficiente para aplicaciones que deben ser ejecutadas en tiempo real en sistemas con capacidad computacional restringida. De hecho, una de sus fortalezas clave es su compatibilidad con CPU, lo que permite su implementación en una amplia variedad de dispositivos sin necesidad de hardware adicional.

Existen dos variantes del modelo: **MoveNet Lightning** y **MoveNet Thunder**. La versión Lightning se caracteriza por su gran velocidad, siendo optimizada para dispositivos móviles y sistemas de baja potencia, con una ligera pérdida de precisión. A pesar de ello, sigue siendo adecuada para muchas aplicaciones en tiempo real. Por su parte, MoveNet Thunder, el modelo utilizado en este proyecto, ofrece una mayor precisión, lo que lo hace ideal para tareas que requieren estimaciones detalladas de las poses, aunque su velocidad es algo menor en comparación con Lightning.

En cuanto a los puntos clave que identifica MoveNet, estos incluyen:

- Nariz
- Ojos (izquierdo y derecho)
- Orejas (izquierda y derecha)
- Hombros (izquierdo y derecho)
- Codos (izquierdo y derecho)
- Muñecas (izquierda y derecha)
- Caderas (izquierda y derecha)
- Rodillas (izquierda y derecha)
- Tobillos (izquierdo y derecho)

A continuación, se presenta un ejemplo de código en Python utilizando la librería OpenCV junto con el modelo MoveNet Thunder. Este código permite capturar imágenes en tiempo real desde la cámara del ordenador y mostrar los puntos clave del cuerpo detectados:

```
import tensorflow as tf
import cv2
import numpy as np

# Cargar el modelo MoveNet Thunder desde TensorFlow hub
model_path = 'C:/Users/movenet_thunder'
model = tf.saved_model.load(model_path)
print("Modelo MoveNet Thunder cargado exitosamente")

# Función para detectar poses
def detect_pose(frame):
    try:
        input_image = tf.image.resize_with_pad(np.expand_dims(frame, axis=0),
            256, 256)
        input_image = tf.cast(input_image, dtype=tf.int32)
        outputs = model.signatures['serving_default'](input_image)
        keypoints = outputs['output_0'].numpy()
        return keypoints
    except Exception as e:
        print("Error al detectar poses:", e)
        return np.array([])

# Iniciar la captura de vídeo
cap = cv2.VideoCapture(1)
if not cap.isOpened():
```

```
    print("No se puede abrir la cámara")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        print("No se puede recibir el cuadro (finalización de la transmisión)")
        break

    # Detectar las poses
    keypoints = detect_pose(frame)

    # Si se detectan puntos clave, dibujar el esqueleto
    if keypoints.size > 0:
        keypoints = keypoints[0][0]

        # Dibujar líneas entre puntos clave
        body_parts = [
            (0, 5), (5, 7), (7, 9), # Cabeza a hombros, hombros a codos, codos a
                muñecas
            (0, 6), (6, 8), (8, 10), # Cabeza a hombros, hombros a codos, codos a
                muñecas
            (5, 6),                 # hombro izquierdo a hombro derecho
            (11, 13), (13, 15),     # Caderas a rodillas, rodillas a tobillos
            (12, 14), (14, 16),     # Caderas a rodillas, rodillas a tobillos
        ]

        for part1, part2 in body_parts:
            y1, x1, _ = keypoints[part1]
            y2, x2, _ = keypoints[part2]
            cv2.line(frame, (int(x1 * frame.shape[1]), int(y1 * frame.shape[0])),
                    (int(x2 * frame.shape[1]), int(y2 * frame.shape[0])), (0,
                        255, 0), 2)

        # Dibujar los puntos clave
        for point in keypoints:
            y, x, _ = point
            cv2.circle(frame, (int(x * frame.shape[1]), int(y * frame.shape[0])),
                5, (0, 0, 255), -1)

    # Mostrar la imagen con el esqueleto
    cv2.imshow('MoveNet - Detección de poses', frame)

    # Salir al presionar 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Este código permite capturar video en tiempo real desde la cámara del ordenador y utilizar el modelo MoveNet Thunder para detectar y marcar los puntos clave del cuerpo humano en la imagen. Los puntos clave con una confianza superior al 50 % se dibujan sobre la imagen para facilitar su visualización.

En el contexto del presente proyecto, se han utilizado los puntos clave correspondientes a las caderas, rodillas y tobillos para detectar caídas. El sistema considera que una persona se encuentra en el suelo cuando las caderas y los tobillos están a la misma altura, o cuando las rodillas están en una posición más elevada. Esta lógica ayuda a evitar falsos positivos, como situaciones en las que una persona está simplemente sentada o tumbada en un sofá.

A continuación, se incluye una captura de ejemplo que muestra el funcionamiento del sistema de detección de poses:

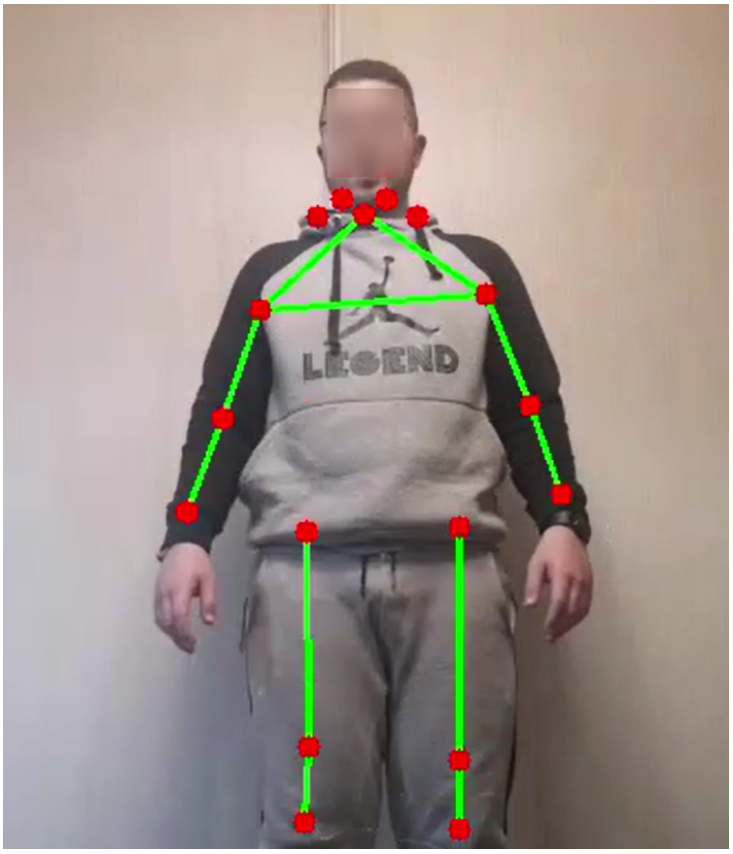


Figura 4.2: Detección de puntos clave utilizando MoveNet Thunder

Este enfoque permite la implementación de un sistema eficiente para la detección de caídas, optimizado para dispositivos con recursos limitados, y sin la necesidad de conexión a internet, garantizando así la privacidad de los usuarios.

4.2.3. YOLO11

YOLO11 (*You Only Look Once*, versión 11) es la última evolución dentro de la familia de modelos YOLO, diseñada para mejorar tanto la precisión como la eficiencia en tareas de detección de objetos en tiempo real. Este modelo, desarrollado para superar las limitaciones de sus predecesores, se enfoca en optimizar el rendimiento

en escenarios complejos que involucran múltiples objetos y variaciones de tamaño dentro de la misma imagen.

A diferencia de versiones anteriores, YOLO11 introduce un enfoque híbrido que combina el procesamiento global de la imagen con un sistema mejorado de detección multi-escala, lo que permite identificar objetos con mayor precisión y en menos tiempo. Esto lo convierte en una solución ideal para aplicaciones críticas como la videovigilancia, la robótica, y otros sistemas en tiempo real.

Las principales características de YOLO11 incluyen su capacidad para manejar volúmenes mayores de datos sin comprometer la latencia, gracias a una arquitectura optimizada para sistemas de hardware moderno. Asimismo, se han integrado nuevas técnicas para la selección de anclas adaptativas y detección refinada, permitiendo una mejor generalización incluso en datasets no balanceados.

Arquitectura de YOLO11

La arquitectura de YOLO11 representa un avance significativo, destacándose por los siguientes elementos clave:

- **Red Backbone basada en redes Transformer**: Este diseño aprovecha la capacidad de los Transformers para captar relaciones globales en las imágenes, mejorando la detección en entornos complejos.
- **Capas convolucionales optimizadas**: Aunque mantiene algunos aspectos de las CNN tradicionales, estas capas han sido rediseñadas para mejorar la extracción de características esenciales y reducir el consumo de recursos.
- **Mecanismo de atención multi-cabeza**: Implementado para focalizar áreas específicas de la imagen, este mecanismo mejora la detección de objetos pequeños o parcialmente visibles.
- **Anclas dinámicas y detección multi-escala avanzada**: Permite la identificación precisa de objetos de diferentes tamaños sin necesidad de ajustar manualmente los parámetros del modelo.

Estos avances aseguran que YOLO11 sea una herramienta robusta y versátil, capaz de abordar los desafíos actuales en tareas de visión por computadora.

Implementación en el Proyecto

Para este trabajo, se ha implementado YOLO11 utilizando las bibliotecas de `OpenCV` y `NumPy`, adaptando el modelo a las necesidades específicas del proyecto. A continuación, se muestra un fragmento de código que ilustra cómo se ha llevado a cabo la integración:

```
import cv2
```

```
from ultralytics import YOLO
import numpy as np

# Cargar el modelo YOLO11
model_yolo = YOLO('C:/Users/yolo11x.pt') # Ruta al modelo YOLO11

# Configurar la captura de video desde la cámara
cap = cv2.VideoCapture(0) # Usar '0' para la cámara predeterminada
if not cap.isOpened():
    print("Error: No se pudo acceder a la cámara.")
    exit()

print("Presiona 'q' para salir.")

while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: No se pudo capturar el frame de la cámara.")
        break

    # Realizar la detección de objetos
    results = model_yolo(frame)

    # Dibujar las cajas delimitadoras y etiquetas
    for result in results:
        for box in result.boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0]) # Coordenadas de la caja
            confidence = box.conf[0] # Confianza del modelo
            label = box.cls[0] # Clase detectada
            class_name = result.names[int(label)] # Nombre de la clase

            # Dibujar la caja en la imagen
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(frame, f"{class_name} {confidence:.2f}", (x1, y1 - 10),
                        cv2.FONT_hERShEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Mostrar la imagen procesada
    cv2.imshow("Detección con YOLO11", frame)

    # Salir con 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Liberar los recursos
cap.release()
cv2.destroyAllWindows()
```

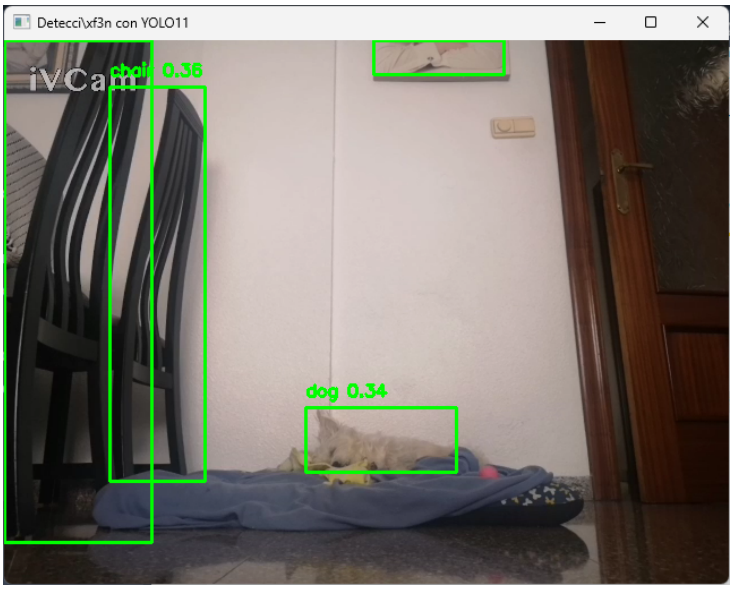


Figura 4.3: Detección de objetos con YOLO11

4.2.4. Solución a la problemática de detección en cama, silla y sofá

Para abordar la problemática de distinguir si una persona está en una cama, silla o sofá, se implementó una solución que integra detección de objetos y poses utilizando los modelos YOLO y MoveNet Thunder, respectivamente.

Primero, se definieron áreas de interés en la imagen, correspondientes a la cama, silla y sofá. Estas áreas se especificaron mediante coordenadas proporcionales al tamaño del fotograma capturado por la cámara, añadiendo un margen adicional para mejorar la detección. Posteriormente, el modelo YOLO identifica las personas y genera cajas delimitadoras para cada una.

Con las cajas delimitadoras, se calcula el centroide de cada persona y se verifica si este se encuentra dentro de alguna de las áreas definidas. Además, se utiliza el modelo MoveNet para estimar las poses y posiciones clave del cuerpo (como caderas, rodillas y tobillos), lo que permite analizar si la postura coincide con estar en el suelo, sentado o acostado. En caso de que la postura detectada sea *en el suelo*, se evalúa el tiempo que la persona permanece en esta posición, y si supera un umbral predefinido, se envía una notificación a través de Bluetooth o Pushbullet.

De esta manera, el sistema combina información espacial y de postura para diferenciar de manera efectiva entre estar en la cama, sentado en una silla o acostado en un sofá, minimizando falsos positivos y proporcionando un método robusto para la detección de caídas en entornos domésticos.

4.3. Test principales

Las pruebas principales realizadas en este proyecto han consistido en la evaluación del sistema utilizando imágenes estáticas de stock, las cuales están fácilmente disponibles en internet para su verificación. Estas imágenes proporcionan una base

amplia para probar el rendimiento del sistema en diversas situaciones, permitiendo una validación preliminar. Además, se han llevado a cabo pruebas adicionales en un entorno controlado, específicamente utilizando una cama, con el objetivo de obtener resultados más precisos y específicos.

Las pruebas se han centrado en las siguientes situaciones clave:

- Al principio del programa

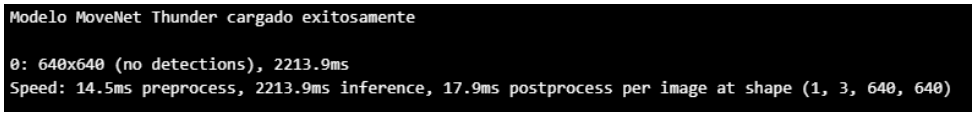


Figura 4.4: Se puede observar que los modelos funcionan correctamente

Figura 4.5: Muestra de que esta bien conectado

- Persona en el suelo.

(a) Persona en el suelo (b) Persona en el suelo (c) Persona en el suelo

Figura 4.6: Test de detección con diferentes imágenes de una persona en el suelo.

- Más de una persona en escena.

Figura 4.7: Foto de más de una persona

- Persona en la cama.

Figura 4.8: Foto de persona en la cama

- Persona en el sofá.

Figura 4.9: Foto de persona en el sofá

- Persona en el sofá.

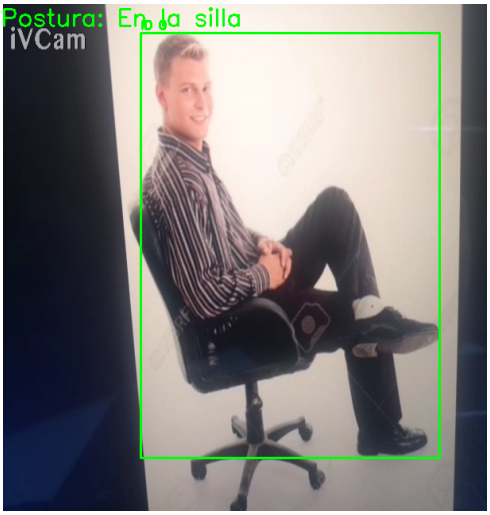


Figura 4.10: Foto de persona en el sofá

- Notificación de la caida

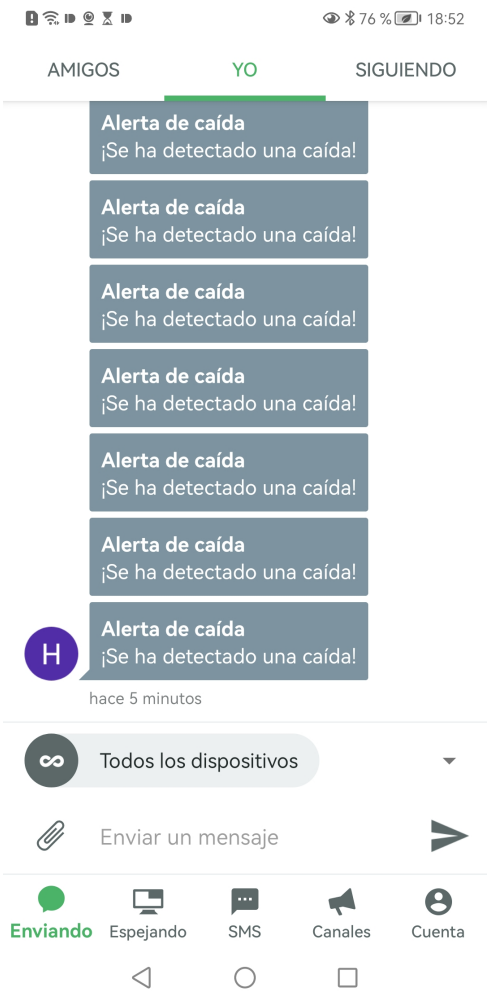


Figura 4.11: Notificación de la caida

La variedad de estos escenarios permite evaluar la capacidad del sistema para

distinguir correctamente entre diferentes tipos de objetos y situaciones. Además, es esencial verificar que las operaciones realizadas por el sistema se ajusten a los requisitos establecidos, garantizando la precisión y fiabilidad en cada caso. A través de esta serie de test, se busca no solo comprobar el rendimiento general del sistema, sino también identificar posibles áreas de mejora en el proceso de detección y clasificación.

Cabe destacar que el sistema ha sido diseñado para realizar el reconocimiento únicamente cuando hay una persona en escena. En caso de que haya más de un individuo presente, se asume que la otra persona puede encargarse de solicitar ayuda, por lo que la detección de caídas no se ejecuta en estos casos. Esta especificación se ha tenido en cuenta durante los test para verificar el correcto funcionamiento del sistema bajo este criterio.

Asimismo, en los casos de detección en silla, sofá y cama, se ha realizado una evaluación detallada desde el inicio de los test. Esto permite comprobar si el sistema realiza correctamente la identificación de estos elementos y, en caso de fallos, facilitar la detección y análisis de errores específicos.

Por último, el elevado número de notificaciones de caídas registradas durante los test se debe tanto a la necesidad de documentar el comportamiento del sistema como a la verificación continua de su progreso y mejora a lo largo del desarrollo del proyecto.

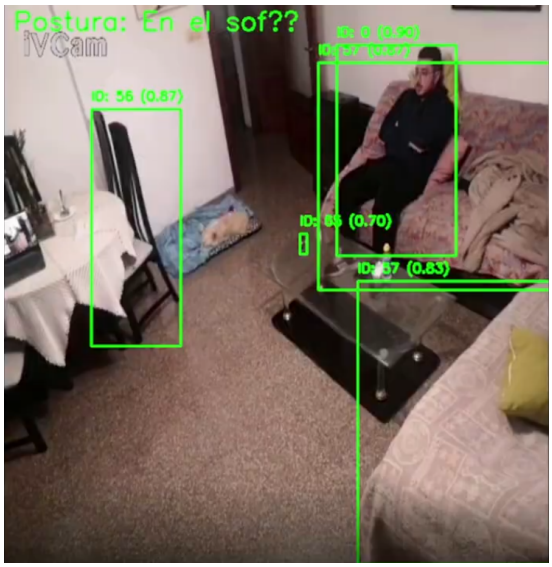
4.4. Test finales

Análisis de Resultados en Entornos Reales

Se han llevado a cabo una serie de experimentos en entornos reales —específicamente en un comedor, un pasillo, un portal y una habitación— con el objetivo de evaluar el rendimiento del sistema de detección de posturas. La metodología consiste en analizar secuencias de vídeo, extrayendo fotogramas representativos y cuantificando el desempeño del sistema mediante métricas de clasificación. En concreto, se ha definido como caso positivo aquella instancia en la que la postura detectada es *"en el suelo"*, mientras que el resto de las posturas se consideran casos negativos.

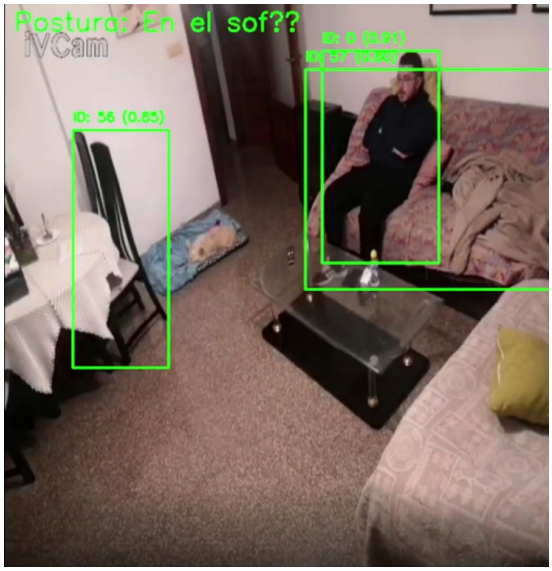
A continuación se describe en detalle el primer test realizado en el comedor. En la Figura **??** se exhiben los fotogramas seleccionados del vídeo del test, y en la Tabla 4.1 se resumen las métricas obtenidas, que incluyen: Verdaderos Positivos, Verdaderos Negativos, Falsos Positivos, Falsos Negativos, así como las medidas de Sensibilidad y Especificidad. Estos indicadores permiten evaluar la capacidad del sistema para identificar correctamente la postura *"en el suelo"* (alta sensibilidad) y para descartar de forma precisa aquellas situaciones en que la postura no corresponde a la definida como positiva (alta especificidad).

4.4.1. Test Comedor 1 (Tipo: Comedor)



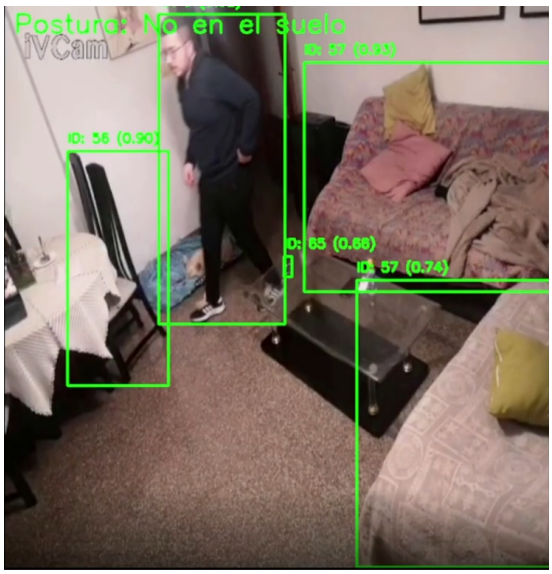
(a) Fotograma 1

(b) Fotograma 2



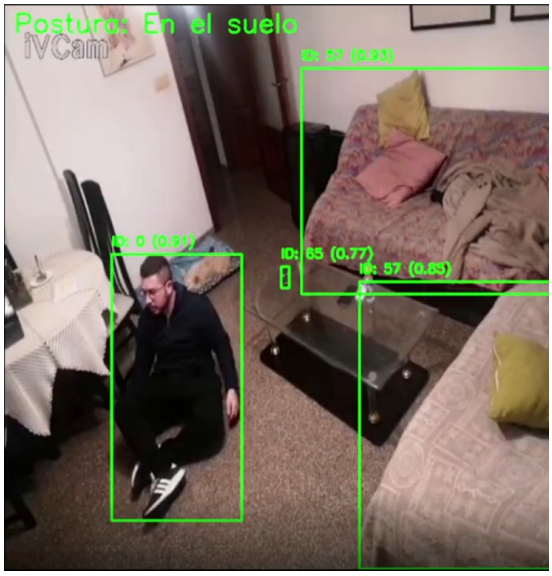
(c) Fotograma 3

(d) Fotograma 4

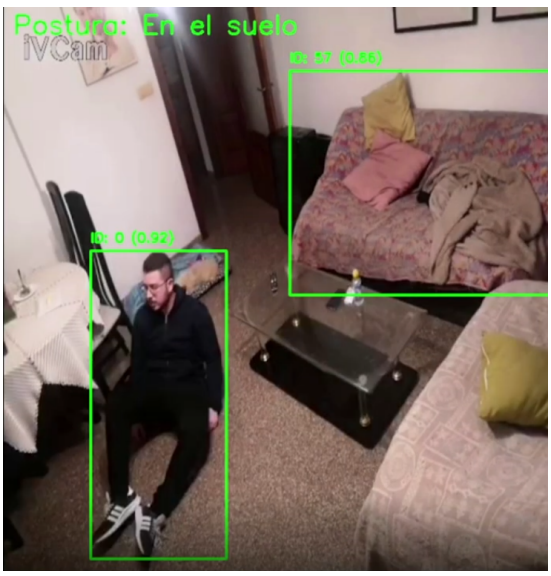


(e) Fotograma 5

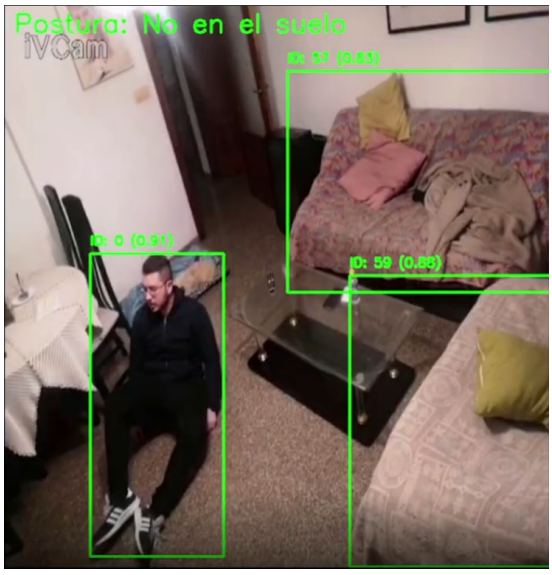
(f) Fotograma 6



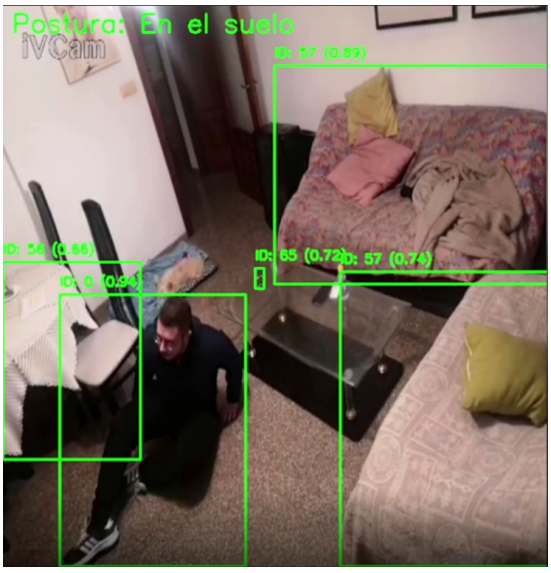
(a) Fotograma 7



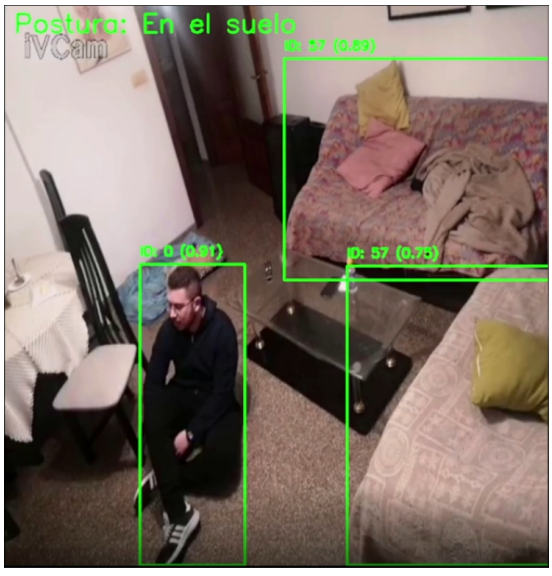
(b) Fotograma 8



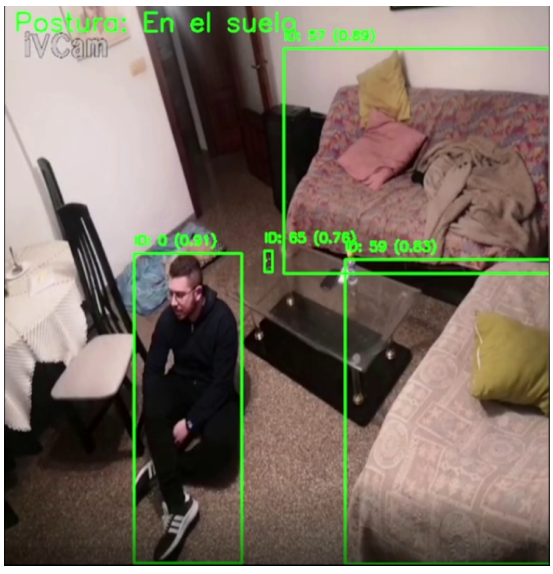
(c) Fotograma 9



(d) Fotograma 10



(e) Fotograma 11



(f) Fotograma 12

(a) Fotograma 13 (b) Fotograma 14

Figura 4.14: Fotogramas test1 en un comedor

La Tabla 4.1 resume los resultados del primer test en el comedor, mostrando los valores de Verdaderos Positivos, Verdaderos Negativos, Falsos Positivos y Falsos Negativos, junto con los porcentajes de Sensibilidad y Especificidad. Estos últimos evidencian un rendimiento prometedor del sistema identificando la postura *"en el suelo"*. La explicación detallada de estas métricas se incluye en el Apéndice A.1.

Cuadro 4.1: Resultados - Comedor 1

Métrica	**Valor**
Verdaderos Positivos	8
Verdaderos Negativos	5
Falsos Positivos	0
Falsos Negativos	1
Sensibilidad	88,89 %
Especificidad	100 %

Según la tabla y los tests, se observa que el resultado es muy satisfactorio, ya que únicamente se ha registrado un error en un fotograma, detectado cuando el individuo se encontraba en el suelo. Este fallo podría atribuirse a una interpretación errónea del fotograma, posiblemente ocasionada por un movimiento brusco.

Las tablas subsiguientes corresponden a tests en diferentes entornos (Comedor 2, Pasillos, Portales y habitaciones), cuyos detalles completos se encuentran en el Apéndice B.1. Se incluyen tanto los resultados individuales como una tabla resumen con los promedios de cada métrica.

Cuadro 4.2: Resultados - Comedor 2

Métrica	**Valor**
Verdaderos Positivos	4
Verdaderos Negativos	4
Falsos Positivos	0
Falsos Negativos	1
Sensibilidad	80 %
Especificidad	100 %

Estos resultados, que muestran la Tabla 4.2, también son muy satisfactorios, ya que el falso negativo se debe a un movimiento rápido que los modelos no pudieron calcular y distinguir correctamente las partes del cuerpo.

Cuadro 4.3: Resultados - Pasillo 1

Métrica	**Valor**
Verdaderos Positivos	5
Verdaderos Negativos	16
Falsos Positivos	1
Falsos Negativos	4
Sensibilidad	55,55 %
Especificidad	94,12 %

Los resultados de la Tabla 4.3 se observa una tendencia similar a la presentada anteriormente. Aunque los datos no resultan completamente convincentes, se evidencia una detección deficiente de las partes del cuerpo, lo que impide que el modelo MoveNet desempeñe adecuadamente su función y conlleva a resultados erróneos.

Cuadro 4.4: Resultados - Pasillo 2

Métrica	**Valor**
Verdaderos Positivos	5
Verdaderos Negativos	1
Falsos Positivos	1
Falsos Negativos	4
Sensibilidad	55,55 %
Especificidad	50 %

Con base en los resultados obtenidos, representados en la Tabla 4.4 y en el análisis visual de los fotogramas y el video, se concluye que el modelo YOLO presentó dificultades para identificar correctamente a la persona. Esto podría deberse a condiciones de iluminación inadecuadas o a que la confianza asignada a la clase “persona” era excesivamente alta, limitando su capacidad para discriminar en ciertos escenarios.

Cuadro 4.5: Resultados - Pasillo 3

Métrica	Valor
Verdaderos Positivos	6
Verdaderos Negativos	2
Falsos Positivos	0
Falsos Negativos	0
Sensibilidad	100 %
Especificidad	100 %

Los resultados obtenidos en el Pasillo 3, representados en la Tabla 4.5, son sumamente optimistas. Esto se debe a que se realizaron ajustes precisos, tanto en la renderización de los fotogramas como en la reducción del umbral de confianza para la detección de la clase "persona" a un 65 % (anteriormente superior al 85 %). Estos cambios han permitido alcanzar una sensibilidad y especificidad del 100 %, demostrando que el modelo ha identificado correctamente tanto los casos positivos como los negativos en este escenario.

Cuadro 4.6: Resultados - Portal 1

Métrica	Valor
Verdaderos Positivos	3
Verdaderos Negativos	5
Falsos Positivos	1
Falsos Negativos	4
Sensibilidad	42,9 %
Especificidad	83,33 %

Los resultados presentados en la Tabla 4.6 se atribuyen, en parte, a la configuración de un umbral de confianza excesivamente alto para la detección de la clase "persona" (según se comentó en la Tabla 4.5). No obstante, el análisis visual de los vídeos y fotogramas revela que, en la práctica, el sistema ofrece resultados muy satisfactorios y un desempeño adecuado.

Cuadro 4.7: Resultados - Portal 2

Métrica	Valor
Verdaderos Positivos	3
Verdaderos Negativos	11
Falsos Positivos	3
Falsos Negativos	2
Sensibilidad	60 %
Especificidad	78,57 %

Al analizar la Tabla 4.7 junto con el examen de los vídeos y fotogramas, se concluye que los modelos presentan deficiencias similares a las observadas en casos

anteriores. Es evidente que la baja iluminación afecta significativamente la detección de la persona, y cuando ésta se encuentra detrás de las rejas, el sistema no logra identificar correctamente las distintas partes del cuerpo, reduciendo así la precisión global.

Cuadro 4.8: Resultados - Portal 3

Métrica	**Valor**
Verdaderos Positivos	1
Verdaderos Negativos	2
Falsos Positivos	3
Falsos Negativos	0
Sensibilidad	100 %
Especificidad	40 %

La Tabla 4.8 evidencia que, cuando existen obstáculos frente a la persona o ésta se encuentra a una distancia considerable, el modelo presenta notables dificultades en la detección precisa. Aunque la sensibilidad es del 100 %, lo que indica que los casos positivos se identifican correctamente, la especificidad del 40 % revela un elevado número de falsos positivos en estos escenarios.

Cuadro 4.9: Resultados - Habitación 1

Métrica	**Valor**
Verdaderos Positivos	5
Verdaderos Negativos	5
Falsos Positivos	0
Falsos Negativos	2
Sensibilidad	71,43 %
Especificidad	100 %

El análisis de la Tabla 4.9 revela que el modelo presenta un desempeño muy adecuado en esta estancia, evidenciado por una especificidad del 100 % y una sensibilidad del 71,43 %. La revisión del vídeo y los fotogramas confirma que, en la mayoría de los casos, la detección es correcta, aunque se identifica un fallo atribuible al umbral de confianza empleado para la clase “persona”.

Cuadro 4.10: Resultados - Habitación 2

Métrica	**Valor**
Verdaderos Positivos	3
Verdaderos Negativos	5
Falsos Positivos	0
Falsos Negativos	2
Sensibilidad	60 %
Especificidad	100 %

El análisis de la Tabla 4.10, complementado con la revisión de los vídeos y fotogramas, indica que el modelo exhibe un rendimiento satisfactorio en esta estancia, evidenciado por una especificidad del 100 % y una sensibilidad del 60 %. Sin embargo, se han detectado limitaciones relacionadas con el umbral de confianza asignado a la clase “persona” y la capacidad del sistema para identificar correctamente las partes del cuerpo durante movimientos rápidos.

Cuadro 4.11: Resultados - Habitación 3

Métrica	**Valor**
Verdaderos Positivos	1
Verdaderos Negativos	4
Falsos Positivos	0
Falsos Negativos	0
Sensibilidad	100 %
Especificidad	100 %

El análisis de la Tabla 4.11 demuestra que, tras los ajustes en el umbral de confianza y el análisis detallado de los fotogramas, el modelo ha alcanzado una sensibilidad y especificidad del 100 % en esta tercera prueba. Estos resultados indican una mejora significativa en la capacidad del sistema para detectar correctamente la presencia y ausencia de la clase “persona” en las condiciones evaluadas.

Cuadro 4.12: Descripción de las abreviaciones

Abreviación	**Descripción**
VP	Verdaderos Positivos
VN	Verdaderos Negativos
FP	Falsos Positivos
FN	Falsos Negativos

Cuadro 4.13: Tabla Resumen Final Transpuesta con Promedio

Prueba	**VP**	**VN**	**FP**	**FN**	**Sensibilidad (%)**	**Especificidad (%)**
Comedor 1	8	5	0	1	88,89	100
Comedor 2	4	4	0	1	80	100
Pasillo 1	5	16	1	4	55,55	94,12
Pasillo 2	5	1	1	4	55,55	50
Pasillo 3	6	2	0	0	100	100
Portal 1	3	5	1	4	42,9	83,33
Portal 2	3	11	3	2	60	78,57
Portal 3	1	2	3	0	100	40
Habitación 1	5	5	0	2	71,43	100
Habitación 2	3	5	0	2	60	100
Habitación 3	1	4	0	0	100	100
Promedio	4	5,46	0,82	1,82	74,03 %	86 %

Capítulo 5

Conclusiones

5.1. Conclusiones

El presente proyecto ha desarrollado un sistema de detección de caídas basado en inteligencia artificial que opera de forma offline, garantizando la privacidad del usuario. Gracias a la integración de los modelos MoveNet Thunder y YOLO, se ha logrado identificar con precisión tanto la posición como el entorno de la persona, permitiendo distinguir eficazmente entre caídas reales y posturas similares. Las pruebas realizadas demuestran que el sistema detecta caídas con un alto grado de fiabilidad en diversos escenarios y condiciones.

Al examinar la Tabla 4.13, se constata que los resultados son satisfactorios, considerando las limitaciones del hardware y los posibles obstáculos presentes en los entornos evaluados. Es fundamental situar la cámara en un punto con la menor cantidad de obstáculos posible, tal como evidencian las pruebas. Aunque los videos y fotogramas indican que no es indispensable contar con una iluminación intensa, se recomienda asegurar condiciones lumínicas adecuadas para optimizar el rendimiento de los modelos.

Si bien los resultados han sido satisfactorios, no se pueden obviar los errores detectados. Entre las posibles soluciones se sugiere incrementar la robustez del modelo MoveNet, de modo que, en caso de no identificar correctamente las partes del cuerpo, se reconozca que la persona no se encuentra completamente en el campo visual. Otra mejora potencial consiste en modificar el criterio de activación del aviso: en lugar de esperar diez sefundos desde el primer aviso en el que la persona está en el suelo y que, al levantarse, se mantenga dicha condición, se podría configurar el sistema para que, si en el transcurso de un minuto al menos el 40 % de los fotogramas muestra a la persona en el suelo, se envíe la alerta.

5.2. Trabajo Futuro

Como línea de desarrollo futuro, se propone la realización de ensayos en entornos reales para validar y perfeccionar el sistema de detección de caídas implementado. Es fundamental evaluar el funcionamiento del programa en condiciones de uso habituales, utilizando los dispositivos mencionados en este trabajo, como la Raspberry Pi 4 Modelo B 2GB y las cámaras de vigilancia seleccionadas. Además, se plantea la

posibilidad de incorporar hardware de mayor rendimiento, como unidades de procesamiento gráfico (GPUs), con el objetivo de potenciar el procesamiento en tiempo real y mejorar la precisión en la detección.

Adicionalmente, se sugiere explorar la integración de otros sensores y tecnologías emergentes, tales como dispositivos inerciales y sistemas de comunicación inalámbrica avanzados, para ampliar la funcionalidad del sistema y ofrecer una solución integral de monitorización y asistencia. Esta integración podría facilitar la detección temprana de situaciones de riesgo y proporcionar una respuesta más eficiente ante posibles caídas, adaptándose a distintos entornos y condiciones de iluminación.

Finalmente, se considera relevante establecer colaboraciones con centros de investigación y entidades especializadas en el ámbito del cuidado y la atención a personas en situación de dependencia. Estas colaboraciones permitirían validar la efectividad del sistema en un entorno clínico, ajustar sus parámetros a las necesidades específicas de los usuarios y sentar las bases para futuras mejoras que consoliden una solución robusta y adaptable a diversas situaciones reales.

Como se detalla en el Apéndice 3 C.1, se incluyen los precios de los componentes utilizados en este proyecto, lo cual podría ser relevante para futuras expansiones del sistema.

Apéndice A

Apéndice 1

A.1. Explicación de metricas

En la evaluación de sistemas de detección de caídas, es fundamental comprender y analizar dos métricas clave: la **sensibilidad** y la **especificidad** [19]. Estas métricas permiten determinar la eficacia del sistema para identificar correctamente los eventos de interés y minimizar las falsas alarmas.

A.1.1. Definición de Sensibilidad y Especificidad

- **Sensibilidad**: Capacidad del sistema para detectar correctamente las caídas (verdaderos positivos). Se calcula como el porcentaje de caídas reales que son correctamente identificadas por el sistema.
- **Especificidad**: Capacidad del sistema para identificar correctamente los eventos que no son caídas (verdaderos negativos). Se calcula como el porcentaje de eventos no relacionados con caídas que son correctamente reconocidos como tales por el sistema.

A.1.2. Aplicación en el Sistema de Cámara Anticaídas

Para evaluar el rendimiento de la cámara anticaídas, se pueden utilizar los siguientes indicadores:

- **Verdaderos Positivos (VP)**: Número de caídas que el sistema detecta correctamente.
- **Falsos Negativos (FN)**: Número de caídas que el sistema no detecta.
- **Verdaderos Negativos (VN)**: Número de eventos no caídas que el sistema identifica correctamente.
- **Falsos Positivos (FP)**: Número de eventos no caídas que el sistema clasifica erróneamente como caídas.

A.1.3. Cálculo de Sensibilidad y Especificidad

Las fórmulas para calcular la sensibilidad y especificidad son:

$$\text{Sensibilidad} = \frac{VP}{VP + FN} \tag{A.1}$$

$$\text{Especificidad} = \frac{VN}{VN + FP} \tag{A.2}$$

A.1.4. Importancia de Sensibilidad y Especificidad

Una alta sensibilidad es crucial para garantizar que la mayoría de las caídas sean detectadas, reduciendo el riesgo de incidentes no reportados. Por otro lado, una alta especificidad es esencial para minimizar las falsas alarmas, evitando intervenciones innecesarias y posibles molestias para los usuarios.

A.1.5. Evaluación del Sistema

Durante las pruebas del sistema de cámara anticaídas, se registraron los siguientes resultados:

- Número total de caídas reales: $N_{\text{caídas}}$
- Número total de eventos no caídas: $N_{\text{no\_caídas}}$
- Caídas detectadas correctamente (VP): X_{VP}
- Caídas no detectadas (FN): Y_{FN}
- Eventos no caídas detectados correctamente (VN): Z_{VN}
- Eventos no caídas detectados incorrectamente como caídas (FP): W_{FP}

Con estos datos, se pueden calcular la sensibilidad y especificidad del sistema utilizando las fórmulas mencionadas anteriormente.

Apéndice B

Apéndice 2

B.1. Test finales

Lo que se presenta a continuación es una explicación del apéndice de los test.

En el siguiente apéndice se puede acceder a la lista de vídeos de test del proyecto, tanto mediante código QR como a través de un enlace.

Además, se ha desglosado cada vídeo fotograma a fotograma en el caso de que no se pueda acceder a ellos.

Para acceder a la lista de reproducción de YouTube, puede utilizar el siguiente enlace:

https://www.youtube.com/playlist?list=PL7AUhBNnhLugOKMF5bcfKh3lk3dP-ahmY

Alternativamente, puede escanear el siguiente código QR:

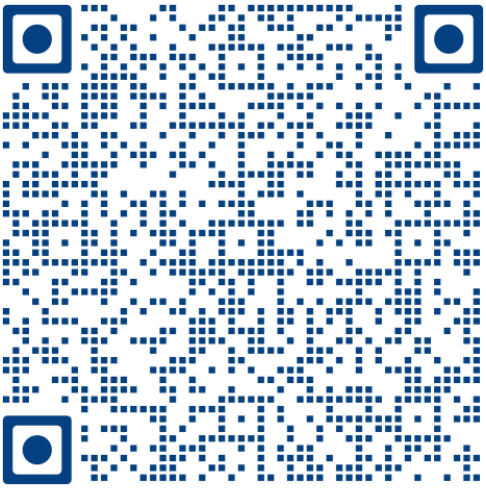
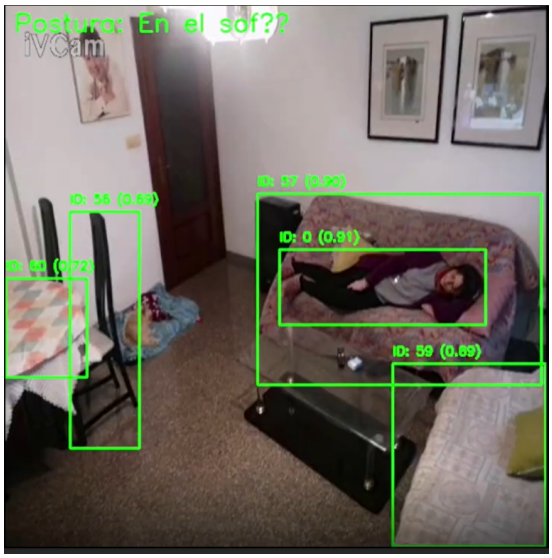
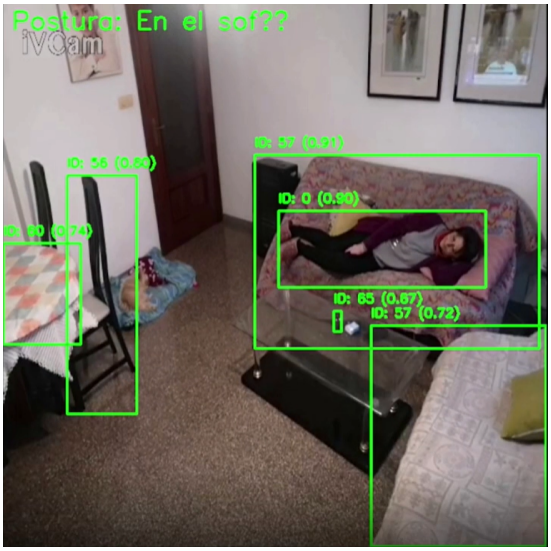


Figura B.1: Código QR para acceder a la lista de reproducción de YouTube

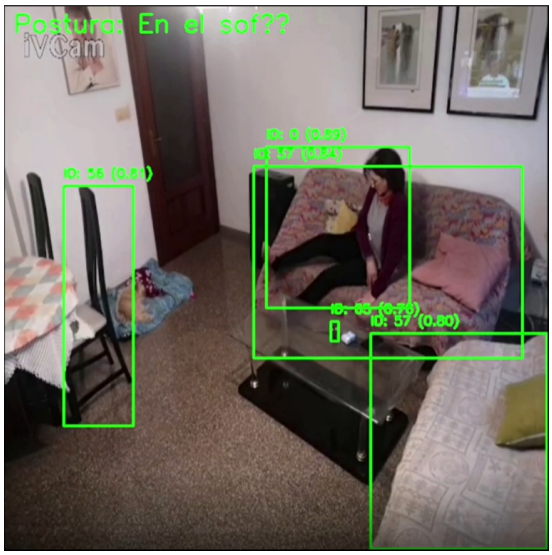
B.1.1. Test Comedor 2 (Tipo: Comedor)



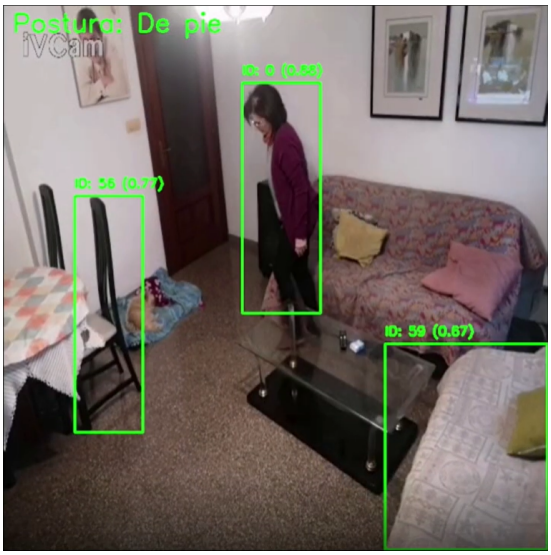
(a) Fotograma 1



(b) Fotograma 2

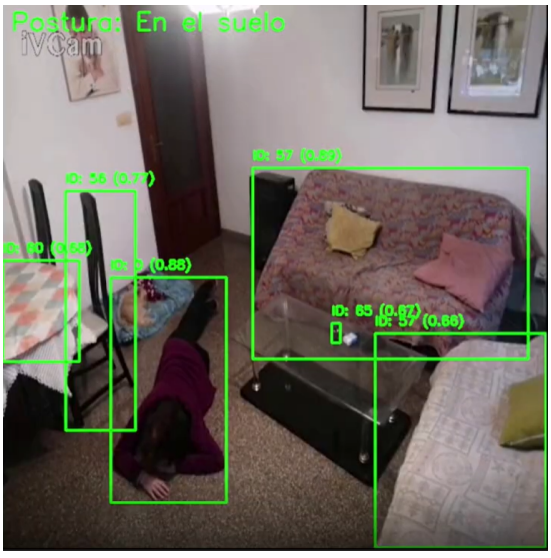


(c) Fotograma 3

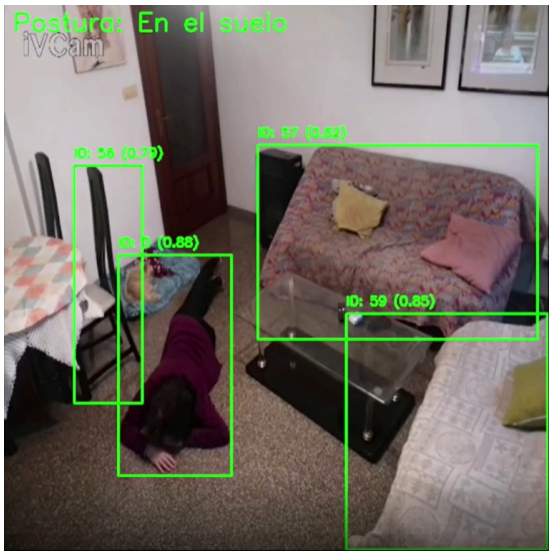


(d) Fotograma 4

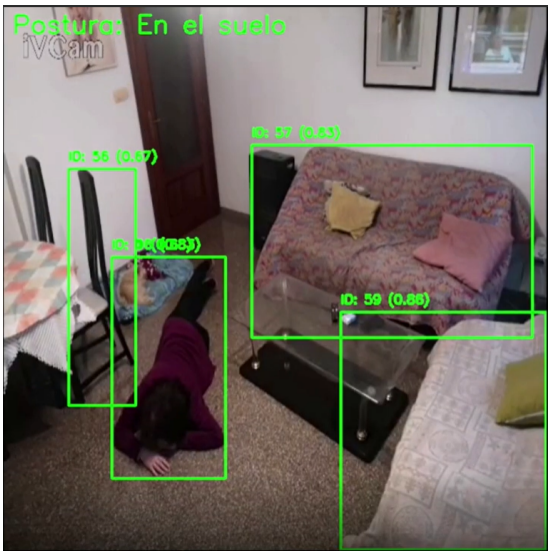
(e) Fotograma 5



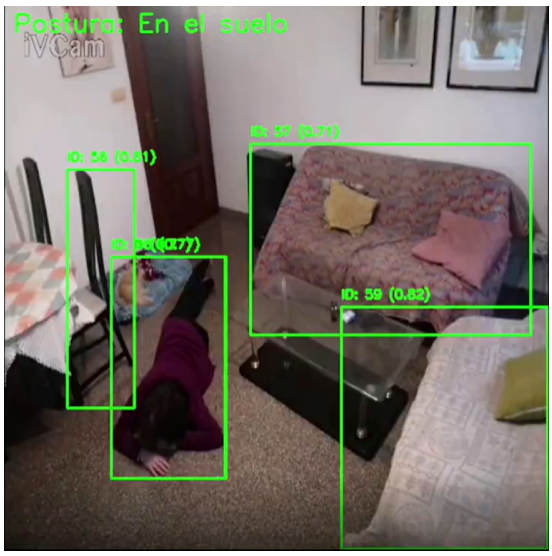
(f) Fotograma 6



(a) Fotograma 7



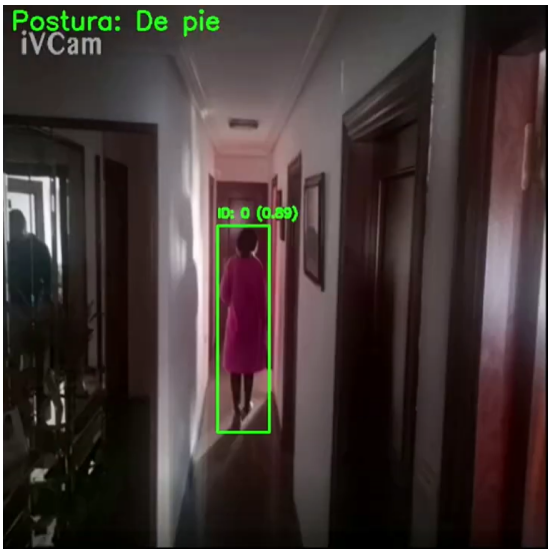
(b) Fotograma 8



(c) Fotograma 9

B.1.2. Test Pasillo 1 (Tipo: Pasillo)

(a) Fotograma 1



(b) Fotograma 2

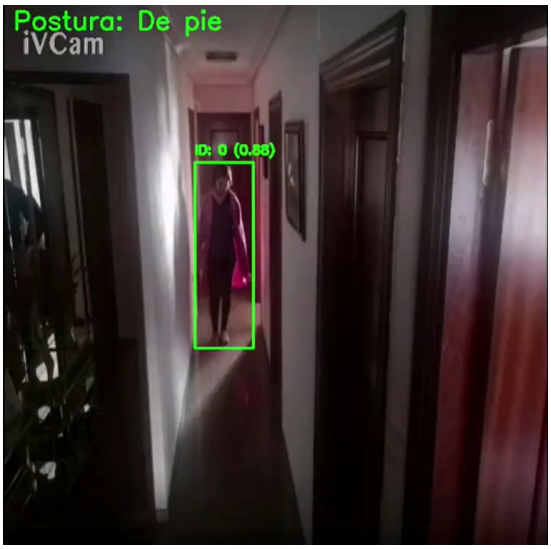
(c) Fotograma 3

(d) Fotograma 4

(e) Fotograma 5

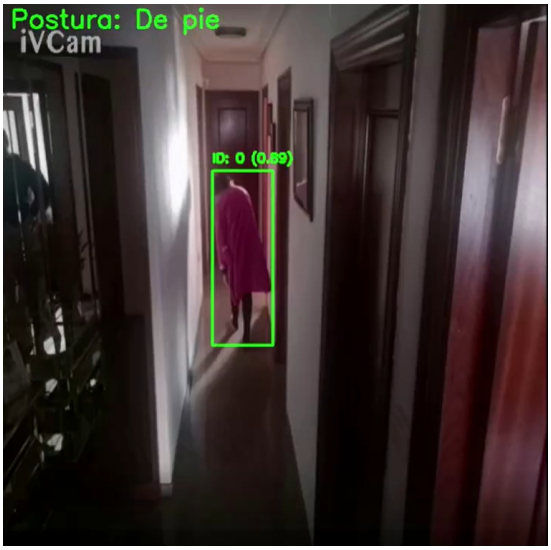
(f) Fotograma 6

(a) Fotograma 7



(b) Fotograma 8

(c) Fotograma 9



(d) Fotograma 10

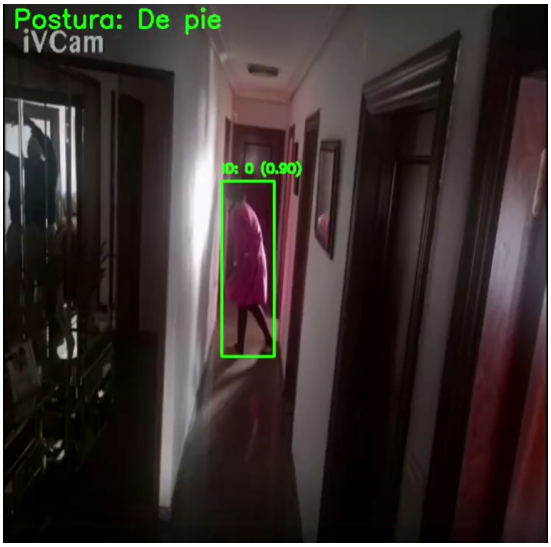
(e) Fotograma 11

(f) Fotograma 12

(a) Fotograma 13

(b) Fotograma 14

(c) Fotograma 15



(d) Fotograma 16

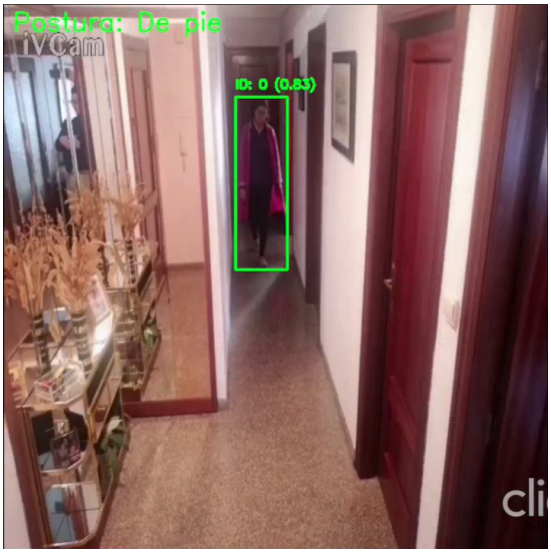
(e) Fotograma 17

(f) Fotograma 18

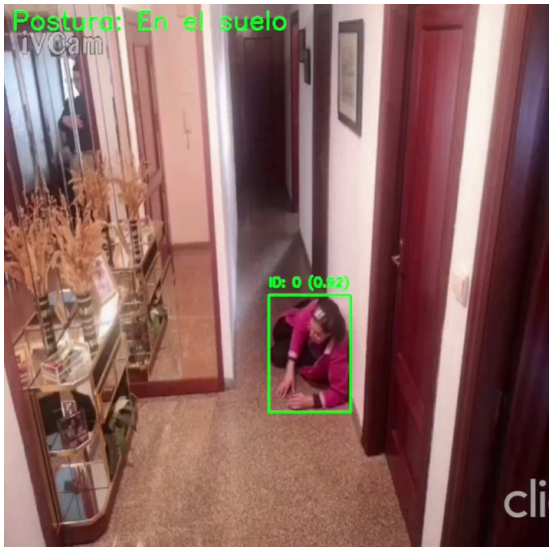
(a) Fotograma 19

(b) Fotograma 20

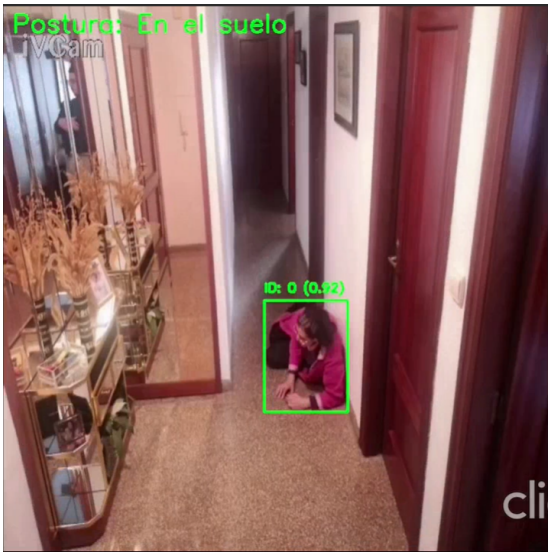
(c) Fotograma 21



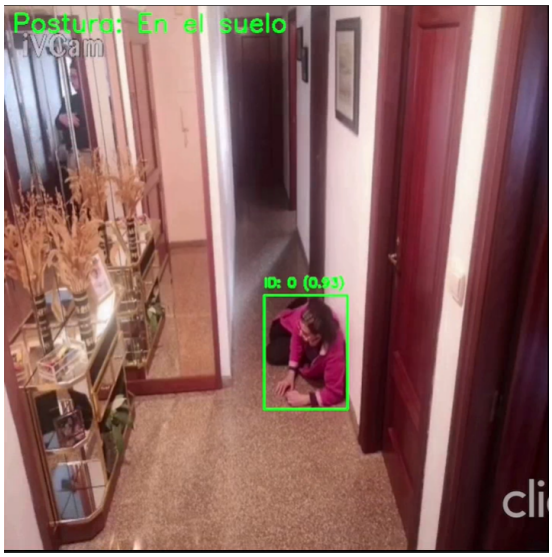
(d) Fotograma 22



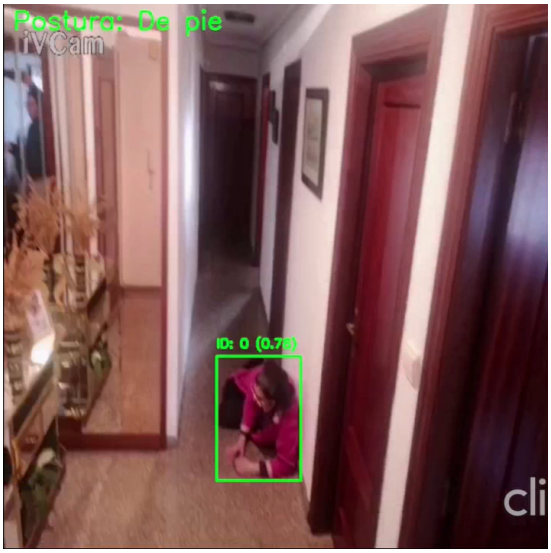
(e) Fotograma 23



(f) Fotograma 24



(a) Fotograma 25



(b) Fotograma 26

Figura B.8: Fotogramas tests1 en un pasillo

B.1.3. Test Pasillo 2 (Tipo: Pasillo)

(a) Fotograma 1

(b) Fotograma 2

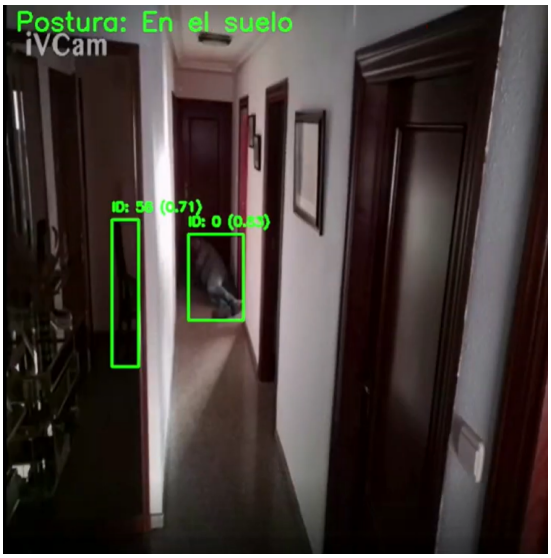
(c) Fotograma 3

(d) Fotograma 4

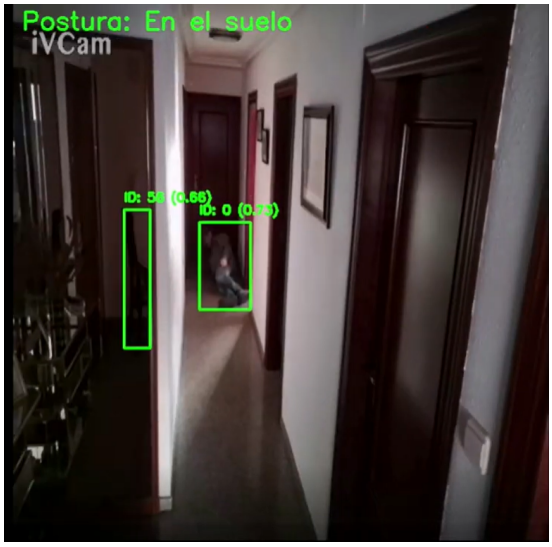
(e) Fotograma 5

(f) Fotograma 6

(a) Fotograma 7

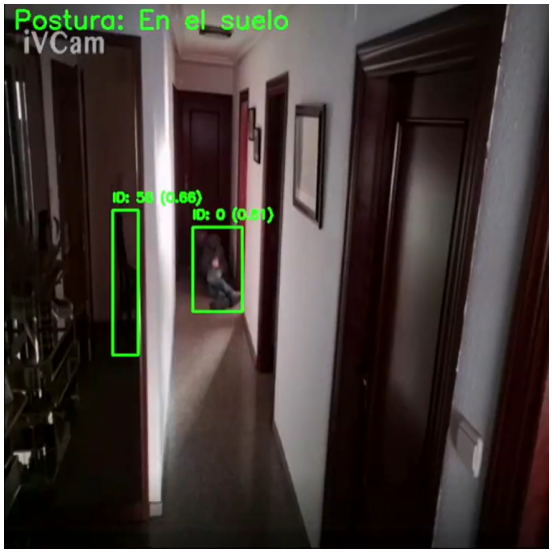


(b) Fotograma 8



(c) Fotograma 9

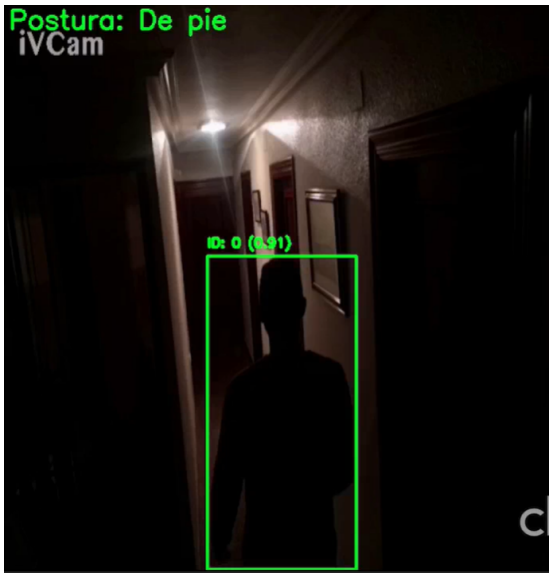
(d) Fotograma 10



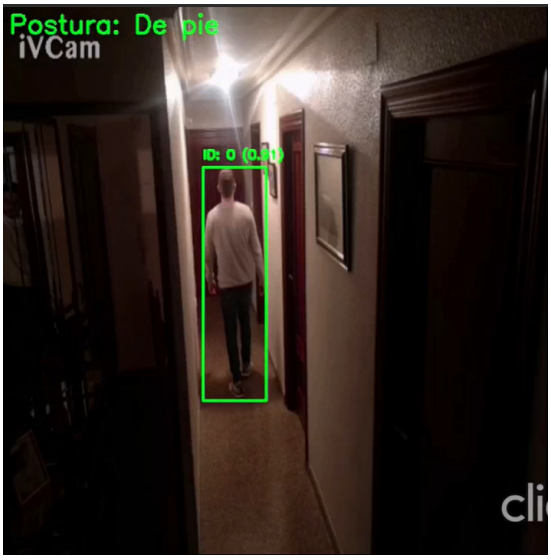
(e) Fotograma 11

Figura B.10: Fotogramas tests2 en un pasillo

Test Pasillo 3 (Tipo: Pasillo)



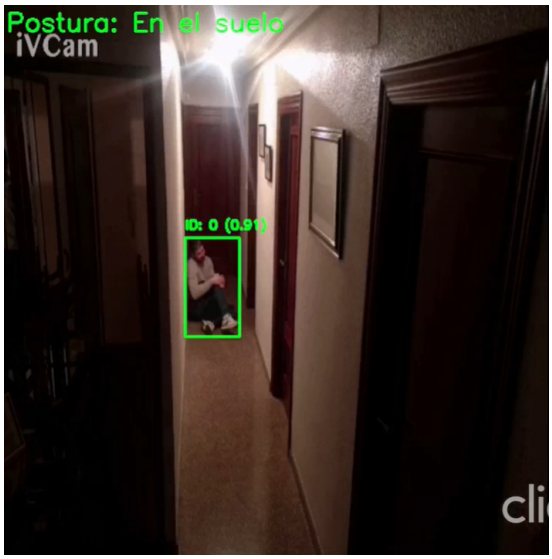
(a) Fotograma 1



(b) Fotograma 2

(c) Fotograma 3

(d) Fotograma 4



(e) Fotograma 5

(f) Fotograma 6

(a) Fotograma 7

(b) Fotograma 8

Figura B.12: Fotogramas tests3 en un pasillo

B.1.4. Test Portal 1 (Tipo: Portal)

(a) Fotograma 1

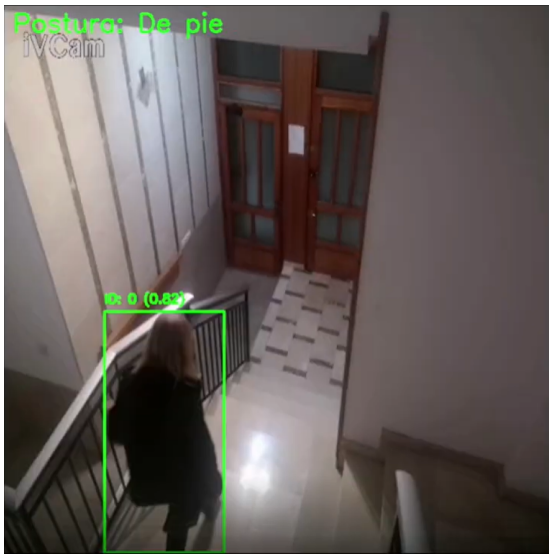
(b) Fotograma 2

(c) Fotograma 3

(d) Fotograma 4

(e) Fotograma 5

(f) Fotograma 6



(a) Fotograma 7

(b) Fotograma 8

(c) Fotograma 9

(d) Fotograma 10

(e) Fotograma 11

(f) Fotograma 12

(a) Fotograma 13

Figura B.15: Fotogramas tests1 en un portal

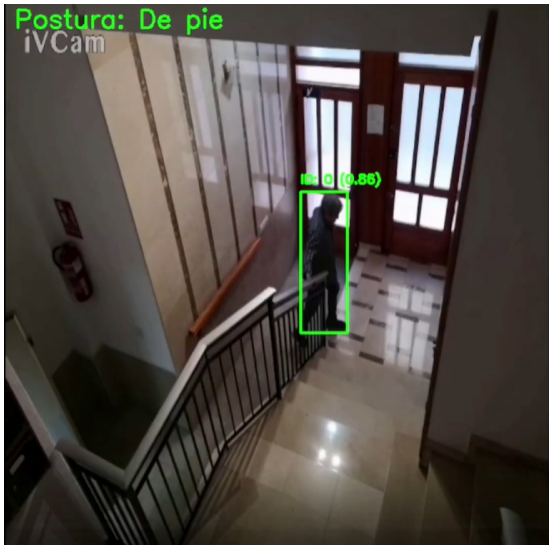
B.1.5. Test Portal 2 (Tipo: Portal)

(a) Fotograma 1

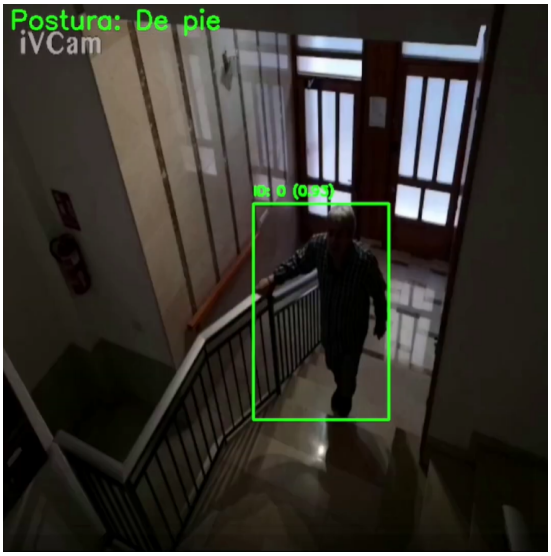
(b) Fotograma 2

(c) Fotograma 3

(d) Fotograma 4



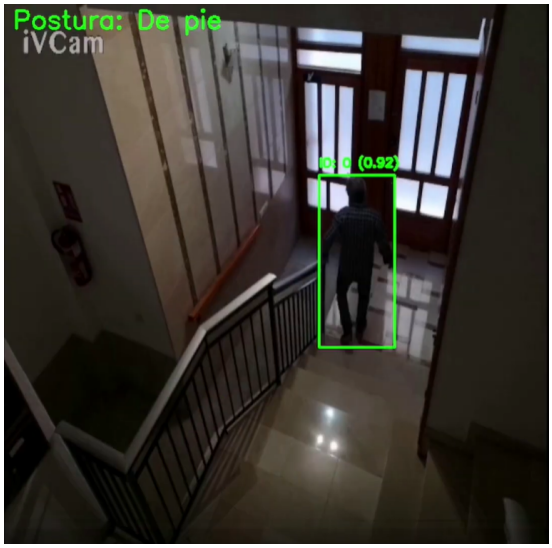
(e) Fotograma 5



(f) Fotograma 6

(a) Fotograma 7

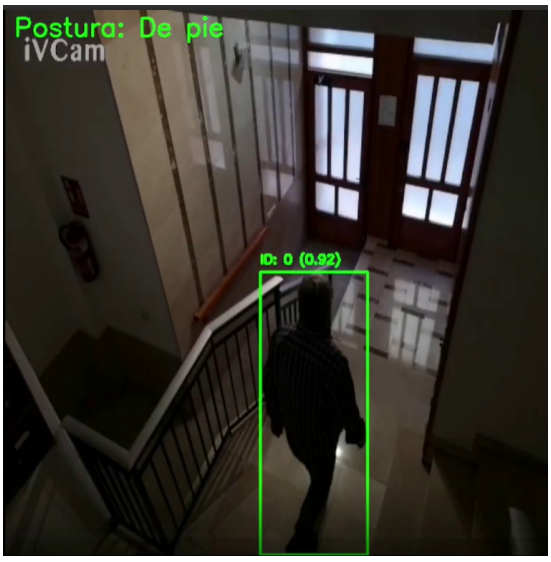
(b) Fotograma 8



(c) Fotograma 9

(d) Fotograma 10

(e) Fotograma 11



(f) Fotograma 12

(a) Fotograma 13

(b) Fotograma 14

(c) Fotograma 15

(d) Fotograma 16

(e) Fotograma 17

(f) Fotograma 18

(a) Fotograma 19

Figura B.19: Fotogramas tests2 en un portal

B.1.6. Test Portal 3 (Tipo: Portal)

(a) Fotograma 1

(b) Fotograma 2

(c) Fotograma 3

(d) Fotograma 4

(e) Fotograma 5

(f) Fotograma 6

Figura B.20: Fotogramas tests3 en un portal

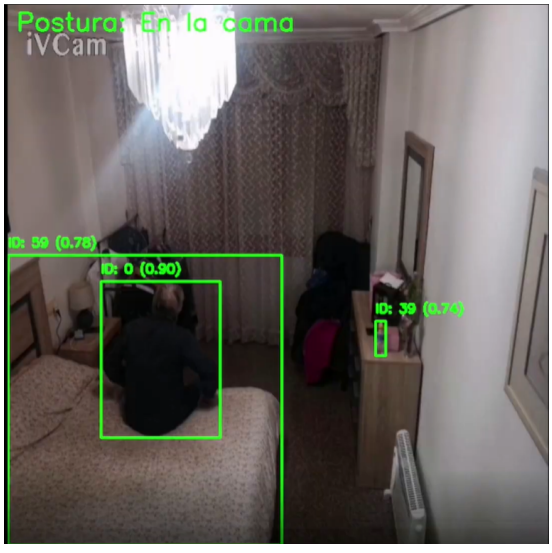
B.1.7. Test habitación 1 (Tipo: habitación)

(a) Fotograma 1

(b) Fotograma 2

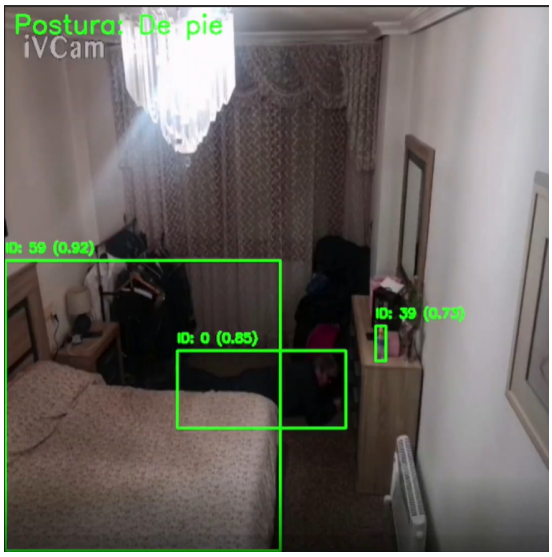
(c) Fotograma 3

(d) Fotograma 4

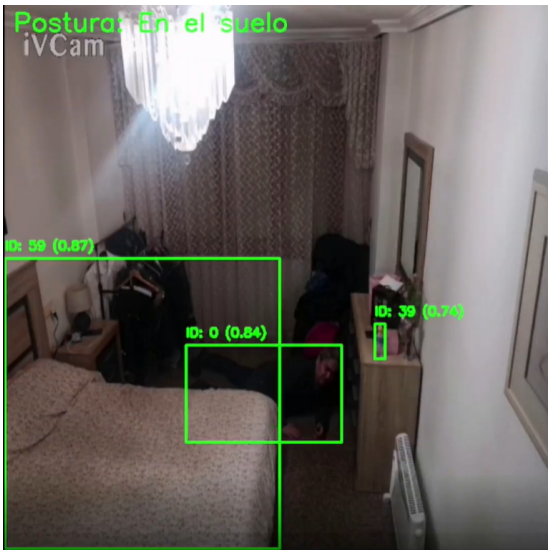


(e) Fotograma 5

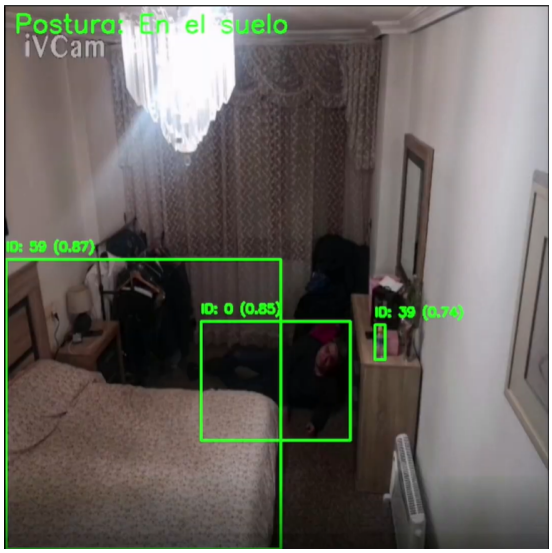
(f) Fotograma 6



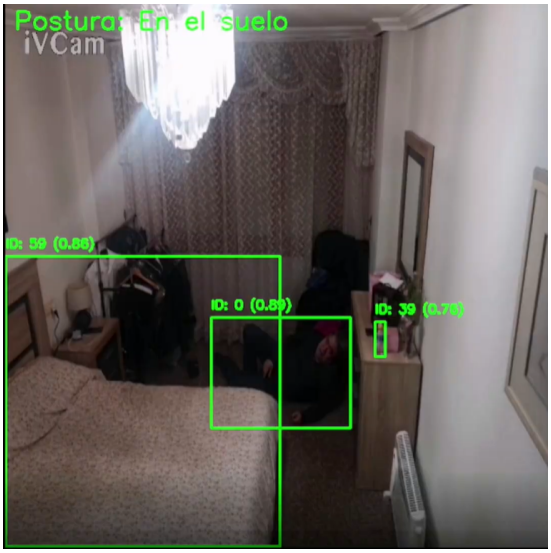
(a) Fotograma 7



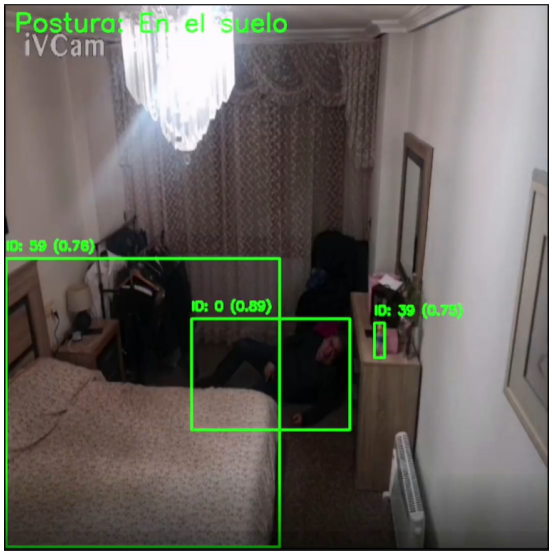
(b) Fotograma 8



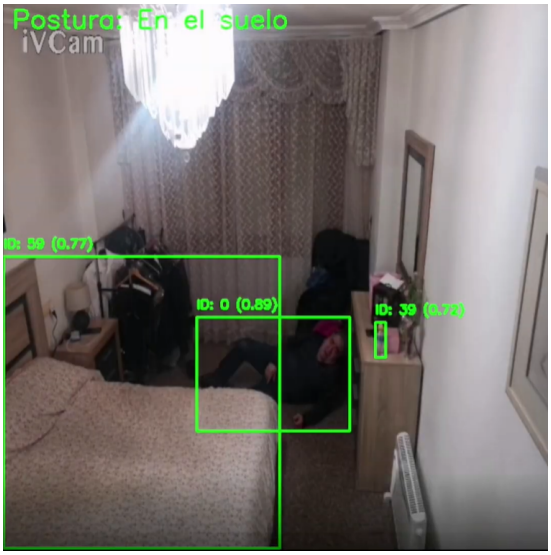
(c) Fotograma 9



(d) Fotograma 10



(e) Fotograma 11

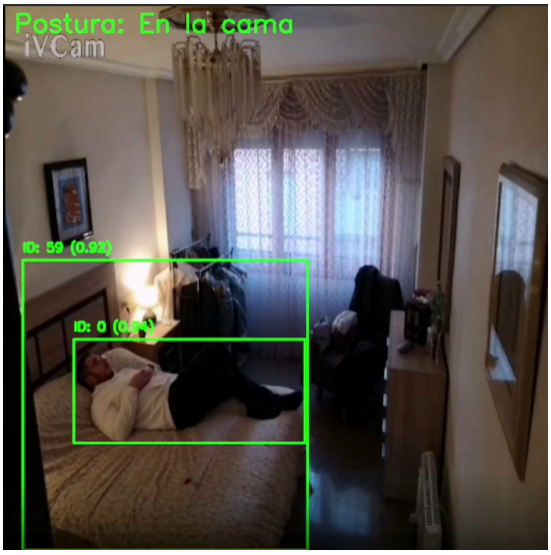


(f) Fotograma 12

Figura B.22: Fotogramas tests1 en una habitacion

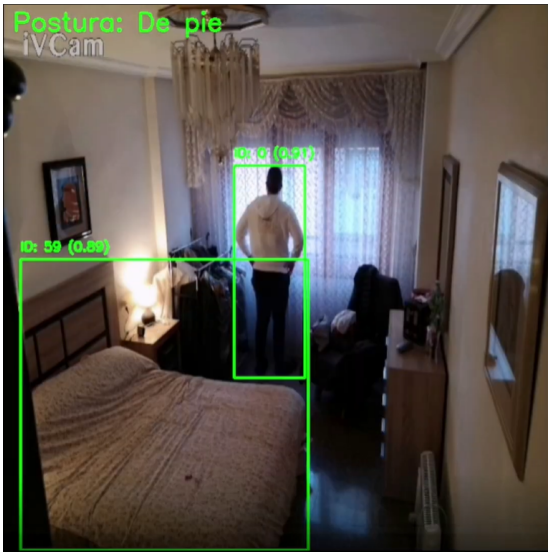
B.1.8. Test habitación 2 (Tipo: habitación)

(a) Fotograma 1

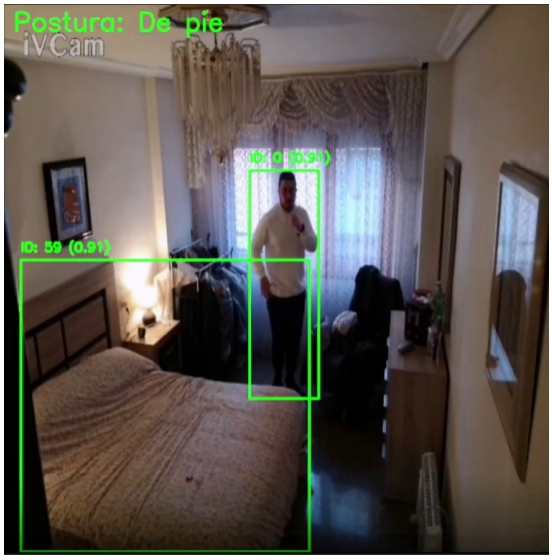


(b) Fotograma 2

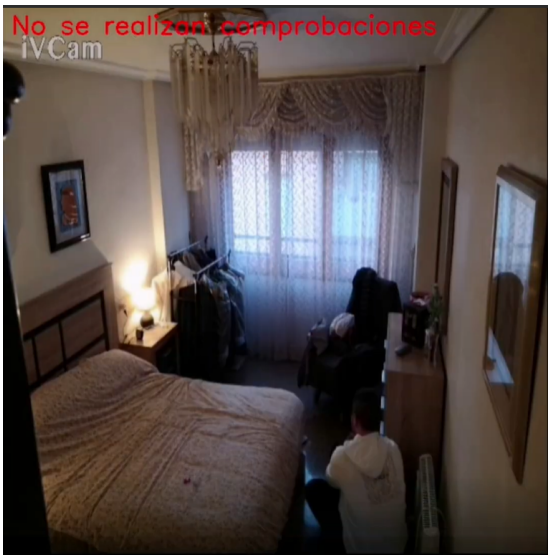
(c) Fotograma 3



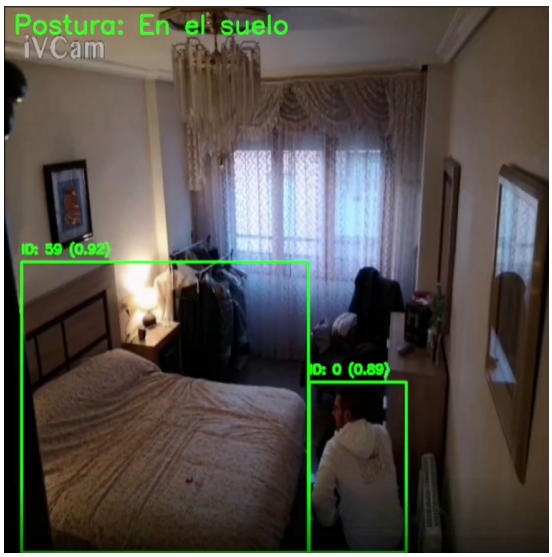
(d) Fotograma 4



(e) Fotograma 5

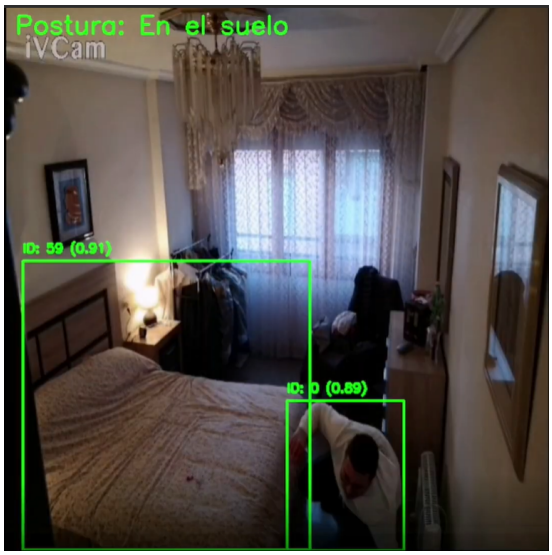


(f) Fotograma 6

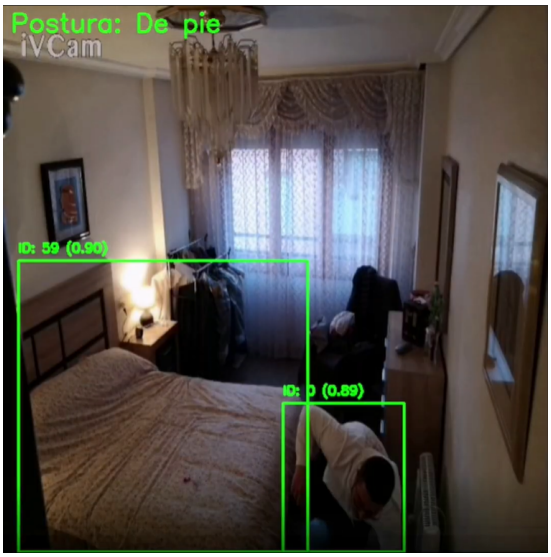


(a) Fotograma 7

(b) Fotograma 8



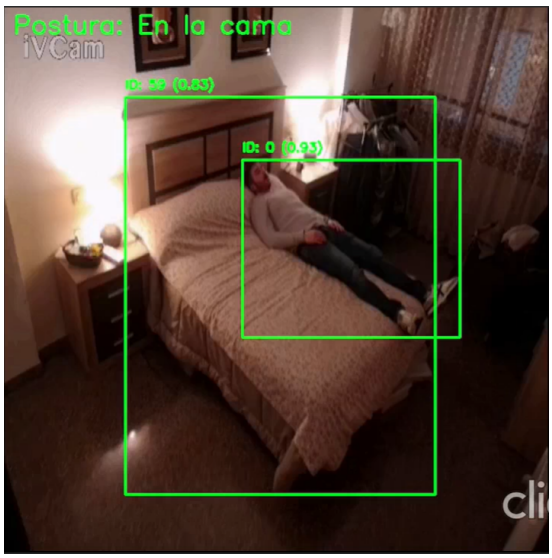
(c) Fotograma 9



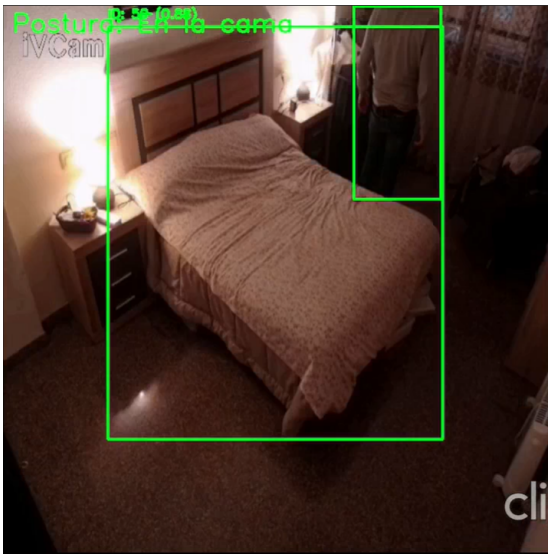
(d) Fotograma 10

Figura B.24: Fotogramas tests2 en una habitacion

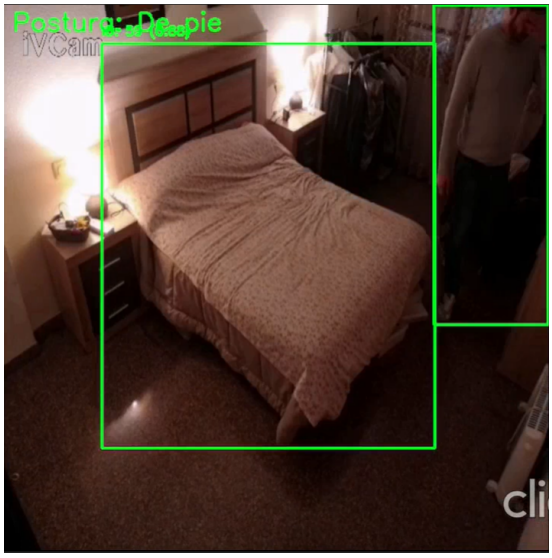
B.1.9. Test habitación 3 (Tipo: habitación)



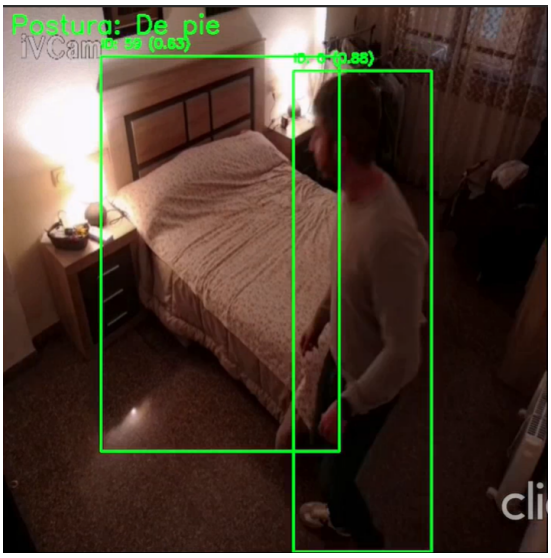
(a) Fotograma 1



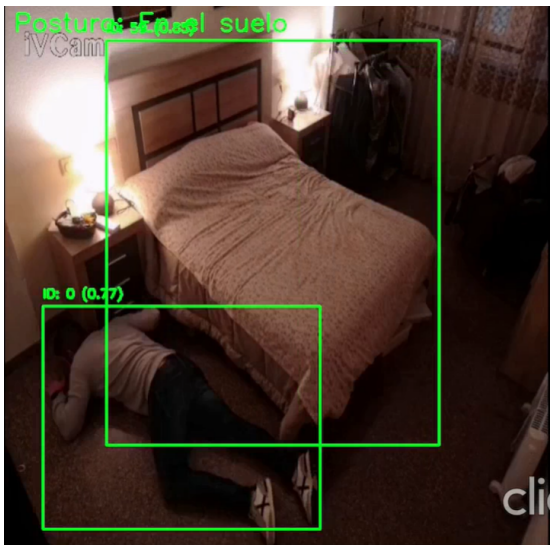
(b) Fotograma 2



(c) Fotograma 3



(d) Fotograma 4



(e) Fotograma 5

Figura B.25: Fotogramas tests3 en una habitacion

Apéndice C

Apéndice 3

C.1. Revisión de costes

En este proyecto, los modelos utilizados son de código abierto, lo que elimina los costos asociados a licencias de software. Sin embargo, es necesario considerar los gastos relacionados con el hardware, específicamente los dispositivos destinados a ejecutar el programa y la cámara.

Los requisitos esenciales para el hardware incluyen:

- Capacidad para ejecutar el código de manera eficiente.
- Compatibilidad con las bibliotecas implementadas en Python.
- Integración con una cámara para la captura de imágenes o videos.

Las opciones consideradas viables para cumplir con estos requisitos son las siguientes:

- **Ordenadores de sobremesa y portátiles**: Esta alternativa permite reutilizar equipos existentes, otorgándoles una segunda vida útil. Para un rendimiento adecuado, se recomienda un procesador Intel i5 y una ampliación de la memoria RAM. Esta inversión es relativamente baja en la actualidad.
- **Dispositivos de bajo consumo energético**: Considerar equipos diseñados para operar de manera continua y eficiente en términos de consumo eléctrico.

Es importante destacar que, al utilizar un ordenador de sobremesa o portátil, el dispositivo estaría encendido las 24 horas del día, los siete días de la semana [20]. Según estimaciones, un ordenador de sobremesa consume aproximadamente 180-200W de media, lo que se traduce en un consumo mensual de alrededor de 30 kWh. Con un precio medio de electricidad de 0,23 euros por kWh, esto implicaría un coste mensual de aproximadamente 6,90 euros.

Además del consumo energético, es fundamental considerar la seguridad. Mantener un dispositivo encendido continuamente puede aumentar el riesgo de fallos de hardware o vulnerabilidades de seguridad si no se gestionan adecuadamente.

En resumen, aunque la reutilización de equipos existentes puede ser una opción económica, es esencial evaluar el consumo energético y los riesgos asociados. Alternativamente, invertir en dispositivos de bajo consumo energético y diseñados para operación continua podría ofrecer una solución más eficiente y segura a largo plazo.

Por otra parte, tenemos dispositivos como la Raspberry Pi, una computadora de placa única de bajo costo y tamaño reducido, diseñada para facilitar el aprendizaje de la programación y la informática en general. A pesar de su tamaño compacto, ofrece un rendimiento notable y es capaz de ejecutar una amplia variedad de aplicaciones y proyectos. Según la búsqueda realizada, la versión de este dispositivo que mejor se adapta a las exigencias de buen precio, capacidad para ejecutar el código teniendo en cuenta las bibliotecas, el entorno Python y la integración con la cámara es el modelo Raspberry Pi 4 Modelo B 2GB. El precio al día de la redacción de esta memoria es de 59,99€ en PcComponentes [21].

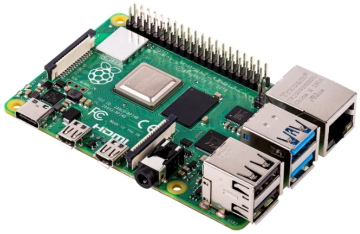


Figura C.1: Imagen de una Raspberry Pi 4 Modelo B 2GB

Opciones de Cámaras para Sistemas de Vigilancia

En el contexto del presente proyecto, es relevante destacar que es posible aprovechar dispositivos móviles como cámaras, otorgándoles así una segunda vida a aquellos smartphones que, habitualmente, permanecen almacenados sin uso. No obstante, si se opta por utilizar cámaras específicamente diseñadas para sistemas de vigilancia, se han identificado las siguientes alternativas:

Módulo de Cámara Oficial Raspberry Pi
El Módulo de Cámara Oficial Raspberry Pi está diseñado para integrarse de forma directa con la Raspberry Pi, ofreciendo una solución compacta y eficiente para aplicaciones de videovigilancia. Al momento de la redacción de este documento, se encuentra disponible por 29,95 € en la tienda [22].

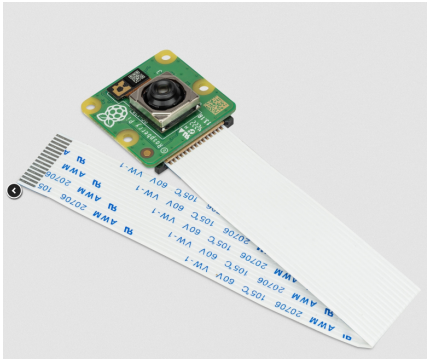


Figura C.2: Módulo de Cámara Oficial Raspberry Pi

Logitech C920 hD Pro Webcam

La Logitech C920 hD Pro Webcam representa otra opción idónea para el proyecto, ofreciendo alta calidad de imagen [23]. Su precio es de 59,99 € en `PcComponentes`:

Figura C.3: Logitech C920 hD Pro Webcam

TP-Link Tapo C200

La cámara TP-Link Tapo C200 es una alternativa económica y funcional para sistemas de vigilancia [24]. Se ofrece a un precio de 28,84 € en `PcComponentes`:

Figura C.4: TP-Link Tapo C200

Cámara de Vigilancia hikvision 5MP

La Cámara de Vigilancia hikvision 5MP constituye otra opción adecuada, con un precio de 50,34 € según se observa en `La Tienda Inteligente` [25]:

Figura C.5: Cámara de Vigilancia hikvision 5MP

Cámara de Vigilancia EZVIZ
Finalmente, la Cámara de Vigilancia EZVIZ se presenta como una opción muy económica, con un precio de 22,99 € en Amazon España [26]:

Figura C.6: Cámara de Vigilancia EZVIZ

Tarifas de Fibra en la Comunidad Valenciana

En el contexto de la conectividad de alta velocidad, resulta interesante analizar las opciones de tarifas de fibra óptica disponibles en la Comunidad Valenciana, especialmente aquellas que se presentan a precios competitivos para proyectos que requieren conexiones estables y de bajo coste. A continuación, se exponen las cinco ofertas más económicas identificadas, en orden ascendente de precio:

1. **DIGI Fibra SMART 300 Mbps**
   Precio: 10 €/mes.
   Características: Tarifa de fibra óptica "solo fibra" disponible en zonas donde DIGI ha desplegado su red propia. Ofrece una velocidad de 300 Mbps sin permanencia, con instalación y router incluidos, lo que la convierte en la opción más económica de la muestra [27].

2. **DIGI Fibra SMART 600 Mbps**
   Precio: 15 €/mes.
   Características: Manteniendo la filosofía de los productos DIGI SMART, esta tarifa ofrece 600 Mbps de velocidad, ideal para entornos con mayor demanda

de datos. Se ofrece sin permanencia y con los servicios de instalación y router gratuitos [27].

3. **Valenfibra – Fibra Óptica 1000 Mbps**
   Precio: 15 €/mes (IVA incluido).
   Características: Propuesta por una empresa local con fuerte arraigo en la Comunidad Valenciana. A pesar de ofrecer una velocidad superior (1000 Mbps), se mantiene en el mismo rango de precio que la opción anterior, presentándose como una alternativa competitiva sin permanencia y con alta e instalación gratuitas [28].

4. **DIGI Fibra SMART 1 Gbps**
   Precio: 20 €/mes.
   Características: Para aquellos usuarios que requieren velocidades aún mayores, DIGI ofrece su tarifa SMART de 1 Gbps, con las mismas condiciones de ausencia de permanencia y servicios incluidos (instalación y router). Esta opción resulta especialmente atractiva en zonas con cobertura DIGI [27].

5. **Telica Telecom – Fibra 100 Mbps**
   Precio: 22,90 €/mes.
   Características: La oferta de Telica Telecom para 100 Mbps se orienta a usuarios con necesidades de conectividad moderadas, ofreciendo una solución "solo fibra" sin permanencia. Si bien la velocidad es inferior a las opciones anteriormente citadas, el precio se mantiene en una gama competitiva dentro del mercado regional [29].

Bibliografía

[1] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[2] Instituto Nacional de Estadística. Proporción de personas mayores de cierta edad por provincia, 2024.

[3] Statista Research Department. Tercera edad: población españa 2002-2024, 2024.

[4] Envejecimiento en Red, CSIC. Perfil de las personas mayores en españa 2023, 2023.

[5] Organización Mundial de la Salud. Caídas, 2021.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[7] Wikipedia. Videovigilancia ip, 2025. Consultado el 26 de febrero de 2025.

[8] Jonathan Andrés Cuyo Gutiérrez. Sistema remoto de detección de caídas en adultos mayores utilizando visíon artificial. Trabajo Fin de Grado, Universidad Técnica de Ambato, Facultad de Ingeniería en Sistemas, Electrónica e Industrial, Año de defensa del TFG.

[9] Carlos García Aguado. Detección de caídas haciendo uso de técnicas de deep learning. Trabajo Fin de Grado, Universidad de Alicante, Escuela Politécnica Superior, Año de defensa del TFG.

[10] Google. Movenet: Single-person pose estimation. TensorFlow hub, 2020.

[11] Ultralytics. Yolo11 documentation. https://docs.ultralytics.com/models//, 2023. Se utiliza como referencia para la comparación entre versiones de YOLO.

[12] Anaconda distribution. https://www.anaconda.com/, 2023.

[13] Pytorch. https://pytorch.org/, 2023.

[14] curl. https://curl.se/, 2023.

[15] Gnu wget. https://www.gnu.org/software/wget/, 2023.

[16] Opencv. https://opencv.org/, 2023.

[17] ivcam. https://www.e2esoft.com/ivcam/, 2023.

[18] Pushbullet. https://www.pushbullet.com/, 2023.

[19] Wikipedia, la enciclopedia libre. Sensibilidad y especificidad, 2025. Última edición: 16 de febrero de 2025.

[20] Gana Energía. ¿cuánto consume un ordenador? https://ganaenergia.com/blog/cuanto-consume-un-ordenador/. Accedido: 2025-02-04.

[21] PcComponentes. Raspberry pi 4 modelo b 2gb. https://www.pccomponentes.com/raspberry-pi-4-modelo-b-2gb. Accedido: 2025-02-04.

[22] TiendaTec. Cámara oficial raspberry pi v3 12mpx. https://www.tiendatec.es/raspberry-pi/camaras/1984-camara-oficial-raspberry-pi-v3-12mpx-5056561803241.html. Accedido: 2025-02-04.

[23] PcComponentes. Logitech hd pro c920 webcam fullhd. https://www.pccomponentes.com/logitech-hd-pro-c920-webcam-fullhd. Accedido: 2025-02-04.

[24] PcComponentes. Tp-link tapo cámara wifi vigilancia 360º 1080p. https://www.pccomponentes.com/tp-link-tapo-camara-wifi-vigilancia-wifi-360o-1080p-vision-nocturna-sonido-bidireccional. Accedido: 2025-02-04.

[25] La Tienda Inteligente. Cámara de vigilancia cctv 5mp lente fija hikvision pro belkis. https://www.latiendainteligente.es/producto/camara-de-vigilancia-cctv-5mp-lente-fija-hikvision-pro-belkis/. Accedido: 2025-02-04.

[26] Amazon España. Ezviz cámara de vigilancia inteligente. https://www.amazon.es/EZVIZ-vigilancia-inteligente-movimiento-C6N/dp/B07W4FMSD9/. Accedido: 2025-02-04.

[27] DIGI Spain Telecom, S.L.U. Combina fibra y móvil – valencia. https://www.digimobil.es/fibra-optica/valencia. Accedido: 2025-02-04.

[28] Valenfibra. Internet fibra Óptica sin permanencia en valencia. https://www.valenfibra.es/fibra-%C3%B3ptica. Accedido: 2025-02-04.

[29] Telica Telecom. Tarifa fibra. https://telica.es/tarifas-fibra/. Accedido: 2025-02-04.