UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DESARROLLO Y EXPERIMENTACIÓN DE UN PROGRAMA DE MONITORIZACIÓN Y CONTROL MEDIANTE VISIÓN ARTIFICIAL Y REDES NEURONALES SIAMESAS.

TFG Grado en Ingeniería Electrónica Industrial y Automática

Autor: Damián Ibáñez Fernández

Tutor: Houcine Hassan Mohamed

Cotutor: Carlos Pascual Domínguez Montagud

Fecha: Valencia, septiembre 2019

Agradecimientos

Quisiera agradecer a todas las personas que me han ayudado a llevar a cabo este proyecto. En especial, gracias a todas las personas que me ayudaron a realizar los diferentes experimentos, los compañeros con los que he compartido este grado y sobre todo gracias a mi amiga Elisa.

Por último, quiero agradecer a mi madre el apoyarme y permitirme realizar estos estudios a pesar de ser lejos de casa.

Índice

Capítulo 1

Introducción

En el siguiente texto se expone el proyecto llevado a cabo para diseñar y crear un programa de seguridad y vigilancia utilizando visión artificial. Para el desarrollo de este programa se han estudiado las diferentes herramientas que se utilizan en la actualidad para solucionar este tipo de problemas.

En este trabajo se ha creado un programa en Python incluyendo la interfaz gráfica de usuario con la intención de controlar la escena grabada por una cámara de vigilancia de diversas formas contando y clasificando todas las personas que aparezcan. También es capaz de crear distintos tipos de zonas, entre ellas zonas en las que solamente el personal autorizado en la base de datos podrá acceder mediante reconocimiento facial.

Uno de los problemas principales en la seguridad por videovigilancia clásica es la gran cantidad de información que recibe la persona encargada de este trabajo. Tener un algoritmo potente capaz de detectar correctamente personas individuales, ser capaz de seguirlas a lo largo de la escena y clasificarlas es muy útil para ayudar a automatizar todo este proceso. Esto hace que las redes neuronales, que no son más que modelos capaces de buscar la combinación de parámetros que mejor se ajusta a un problema concreto nos sean muy útiles, ya que son muy flexibles. Por ello han sido una de las herramientas principales en este proyecto.

Además, las redes neuronales utilizadas han sido entrenadas para solo necesitar una foto de la persona autorizada en la base de datos para poder ser reconocida. Esto es muy útil, ya que la mayoría de redes neuronales necesitan miles de imágenes para poder crear un modelo efectivo.

Capítulo 2

Motivación

Este proyecto ha sido realizado gracias a la propuesta del tutor de crear una aplicación de seguridad. Tras varias reuniones y diferentes propuestas para ello se decidió diseñar un programa que utilizase la visión artificial para la videovigilancia de forma flexible y automática.

En este trabajo el alumno ha sido capaz de poner en práctica, así como ampliar sus conocimientos en visión artificial, programación y automatización. Además, pudo investigar sobre diferentes herramientas de interés que se utilizan en la actualidad como son las redes neuronales, el seguimiento de objetos y el reconocimiento facial.

Capítulo 3

La visión artificial en la seguridad

3.1. ¿Qué es la visión artificial?

La visión artificial es un campo de estudio cuyo objetivo es conseguir que los ordenadores sean capaces de ver, es decir entender y poder trabajar con el contenido de imágenes digitales o vídeos. Conseguir esto es bastante más difícil de lo que pudiera parecer, ya que para los seres humanos comprender las cosas que suceden en nuestra visión es algo trivial hasta para un niño, pero nuestros limitados conocimientos de la visión biológica y la complejidad de la percepción dinámica del mundo lo convierten en un problema que sigue sin resolver.

Así pues, la visión artificial se encarga de adquirir, procesar, analizar y comprender imágenes digitales para muchas y diversas aplicaciones como la inspección de maquinaria, modelado 3D, captura de movimiento, seguridad del automóvil, vigilancia...

Estas aplicaciones se consiguen a través de modelos matemáticos, algoritmos, estadística y otras herramientas que permiten analizar los diferentes componentes de la imagen, como por ejemplo usar un algoritmo capaz de buscar bordes en la imagen teniendo en cuenta la posición de los píxeles. Añadiendo más y más de estos algoritmos podemos llegar a conseguir solventar problemas bastante complejos.

Uno de los problemas que tiene la visión artificial actualmente es el tiempo de procesado de la imagen, ya que ejecutar algoritmos muy complicados resulta en tiempos de procesado muy largos que no permiten por ejemplo trabajar en tiempo real o cumplir ciertas metas. Un balance entre los tiempos de procesado y la eficacia de los algoritmos es una de los objetivos que se intentan cumplir hoy en día.

3.2. La visión artificial en servicios profesionales

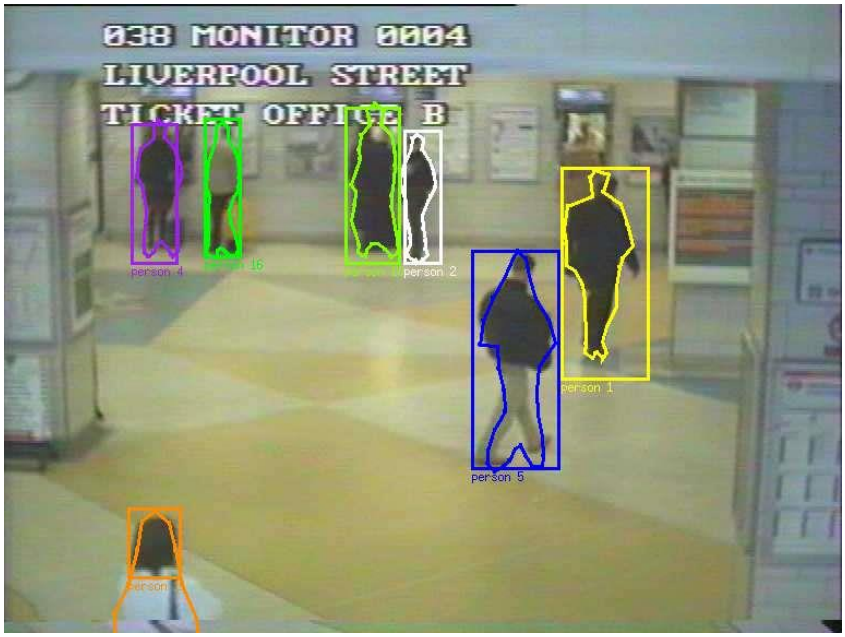
El cambio en los últimos años de equipamiento analógico por digital, tanto en cámaras, sensores u otros instrumentos ahora más asequibles ha ayudado a este avance y mayor uso de la visión artificial. Hoy en día podemos encontrar empresas que realizan servicios profesionales utilizando visión artificial mayoritariamente en las siguientes categorías:

1. Sistemas de vigilancia
2. Control del tráfico
3. Robótica y cadenas de montaje
4. Automatización en agricultura
5. Vehículos autónomos
6. Smartphones
7. Deportes
8. Medicina

1. Sistemas de vigilancia

La visión artificial se ha convertido en una de las herramientas más avanzadas y utilizadas en la industria de la seguridad y la vigilancia. Esto es gracias a la mejora en los niveles de confianza y fiabilidad de los algoritmos de reconocimiento facial, la detección de elementos peligrosos y la inclusión de las redes neuronales en los últimos años. Además, la gran cantidad de datos disponibles y su continuo aumento hacen que cada vez se utilice más en este campo.

Otra parte importante a tener en cuenta es la velocidad de reacción de una detección por visión artificial comparada a la de un operario. Por estos motivos existen ya muchas empresas dedicadas a la venta de estos servicios tanto para propiedades privadas como lugares amplios como centros comerciales o aeropuertos. Por ejemplo, en muchos aeropuertos actualmente para atravesar la frontera se reconoce al individuo mediante una cámara en lugar de tener a un empleado verificando la identidad del individuo.



`Figura 3.2.1. Sistema de seguridad por Nils T Siebel.

2. Control del tráfico

Los llamados sistemas de control de tráfico adaptativos (ATCS) utilizan técnicas de procesado de video para numerar los vehículos y seguir sus trayectorias, calcular su velocidad, el nivel de saturación del tráfico o incluso reconocer vehículos buscados de forma autónoma. Con esta información y teniendo control de las señales de tráfico se espera optimizar el flujo de vehículos además de detectar infracciones como violaciones del límite de velocidad estipulado, accidentes u otras infracciones de forma mucho más rápida y segura que actualmente.

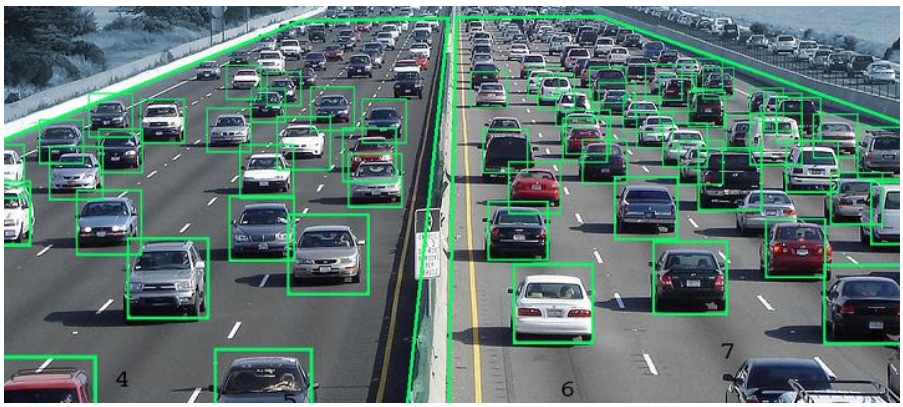


Figura 3.2.2. Sistema de control de tráfico adaptativo.

3. Robótica y cadenas de montaje

La primera aplicación comercial de la visión artificial en la industria fue en este tipo de trabajos. Su uso principal es revisar de forma automática que los productos están en buen estado realizando tareas de inspección y calidad. Otro uso importante es el de automatizar procesos de ensamblaje, detectando la correcta orientación de las diferentes piezas, medidas y defectos en las cadenas de ensamblaje con mayor consistencia, velocidad y precisión que el que un trabajador puede ofrecer.

Figura 3.2.3. Control de calidad mediante visión artificial.

4. Automatización en agricultura

El uso de la visión artificial en este campo es bastante similar al que se le da en la robótica y las cadenas de montaje, los objetivos principales que se buscan en este caso es identificar de forma automática los cultivos, monitorización de cultivos y ganado, así como su clasificación o incluso calcular la geografía del terreno y la composición del suelo.

En algunos casos se están utilizando de forma combinada la visión artificial con drones para conseguir este tipo de información. Todo esto ayuda a aumentar la eficiencia no solo de la obtención de esta información, sino del uso de la maquinaria también.

Figura 3.2.4. Control de cultivos mediante visión artificial, por Blue River Technology.

5. Vehículos autónomos

Los vehículos autónomos necesitan una gran cantidad de información para poder funcionar correctamente, y la mayoría de esta información se consigue mediante cámaras y visión artificial junto con otros tipos de sensores. Pero actualmente está tecnología está en desarrollo por varios motivos.

Sin embargo, actualmente la visión artificial sí se encuentra en el mercado de la automoción con otros tipos de aplicaciones de automatización, como por ejemplo la detección de puntos ciegos, los sistemas de visión nocturna, detección de somnolencia o sistemas de prevención de colisión.

Figura 3.2.5. Sistema de detección y reconocimiento en un coche autónomo por NVIDIA AI Labs.

6. Smartphones

Las grandes empresas de smartphones están en una continua carrera por mejorar las cámaras frontales y traseras, y a su vez el software que hay detrás de ellas. Entre las diversas aplicaciones que estos aparatos ofrecen podemos encontrar detección facial, estabilización de video, anticipación de movimiento, autoenfoque, control automático de la iluminación, detección de profundidad... El caso de los smartphones es probablemente el contacto más cercano que tiene la población general con la visión artificial.

Figura 3.2.6. Reconocimiento facial del IPhone X.

7. Deportes

En el caso del deporte el uso de la visión artificial es mayormente ayudar a los atletas y árbitros. Por ejemplo, para los árbitros el uso de cámaras es ya muy útil en deportes como el tenis o en carreras de cualquier tipo para verificar el ganador y detectar con mayor seguridad faltas e infracciones. Para los propios atletas la visión artificial se utiliza para obtener datos, estadísticas y probabilidades que utilizan para mejorar y prepararse contra otros atletas.

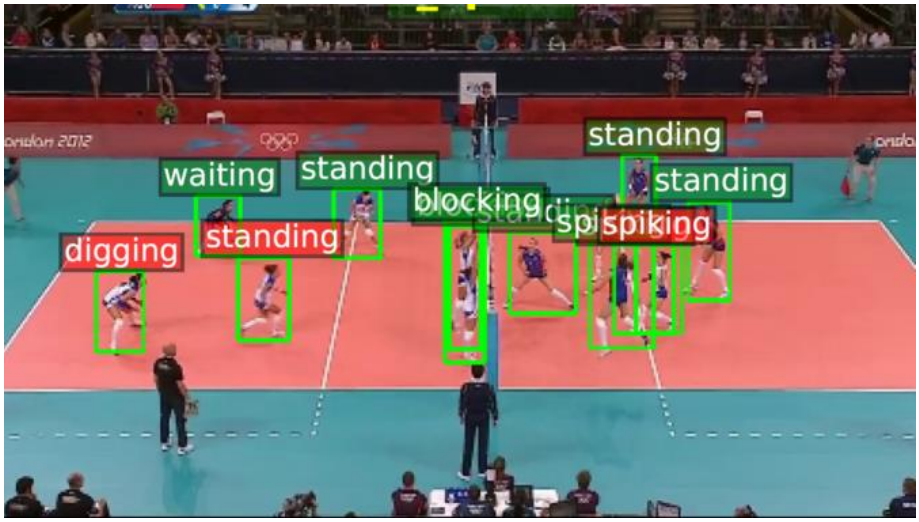


Figura 3.2.7. Detección de individuos y juego en baseball por NVIDIA AI Labs.

8. Medicina

En el campo de la medicina el uso de la visión artificial lo encontramos en el análisis de las imágenes que se obtienen de tecnología como los escáneres, rayos X o resonancias magnéticas. Usando esta tecnología se pueden identificar tumores y otras anomalías de forma mucho más rápida y segura que las convencionales.

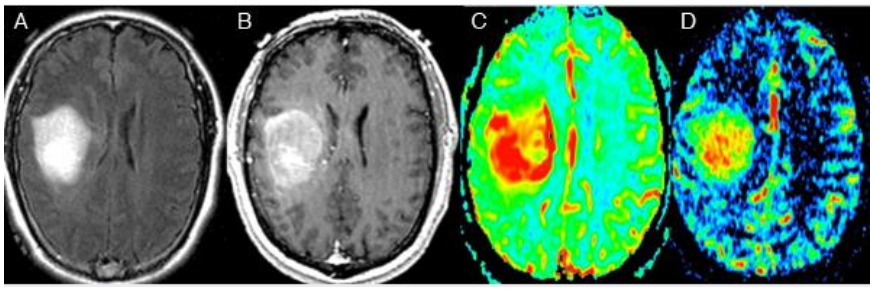


Figura 3.2.8. Análisis de un tumor cerebral mediante visión artificial.

3.3. Análisis de mercado de seguridad mediante visión artificial

El mercado de la visión artificial y la inteligencia artificial ha aumentado sobre todo en los últimos años. Podemos observar esto en esta estadística de las ganancias por empresas en Estados Unidos en los últimos años y su previsión, por Fortune Business Insights.

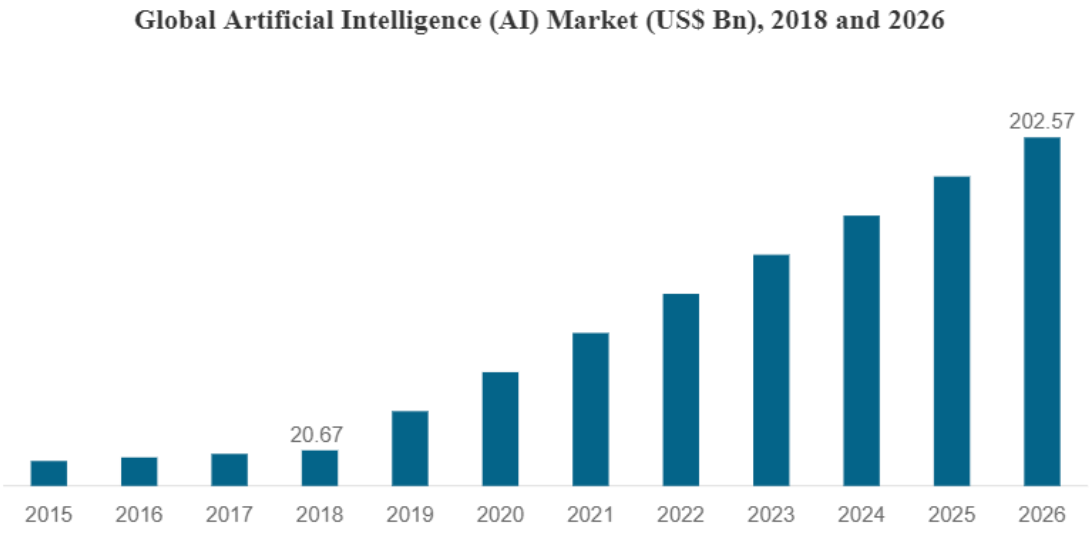


Figura 3.3.1. Mercado de la IA en Estados Unidos por Fortune Business Insights.

Por otro lado, también se conoce en qué campos se utilizan más la inteligencia artificial, siendo la visión artificial uno de los tres campos principales en los que se utiliza.

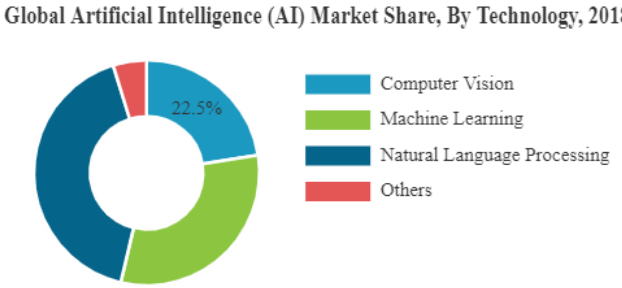


Figura 3.3.2. Diferentes mercados de la IA Fortune Business Insights.

Con estos datos podemos calcular como en 2018 las ganancias de sistemas de inteligencia artificial usadas en visión artificial alcanzan el valor de 4,650,750,000 $, siendo este el 22.5% de 20,670,000,000 $.

En el caso concreto de la vigilancia, el país que más está invirtiendo en este tipo de tecnología es China. En el siguiente gráfico vemos por ejemplo como han aumentado el número de patentes en reconocimiento facial para vigilancia comparado con la seguridad por cámaras de operario clásicas:

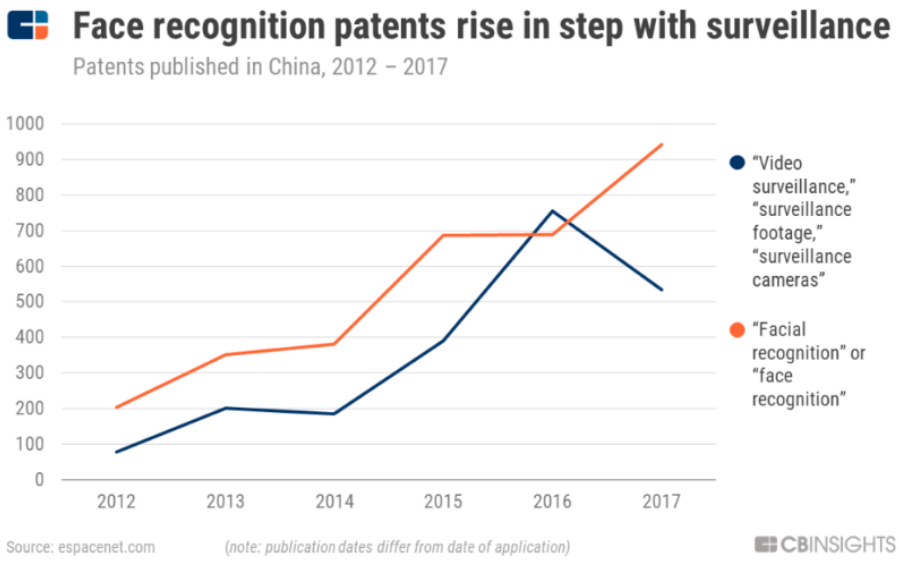


Figura 3.3.3. Grafica comparativa de patentes de reconocimiento facial y videovigilancia.

En este otro gráfico podemos observar el gran crecimiento en la inversión en visión artificial que ha hecho China en los últimos años:

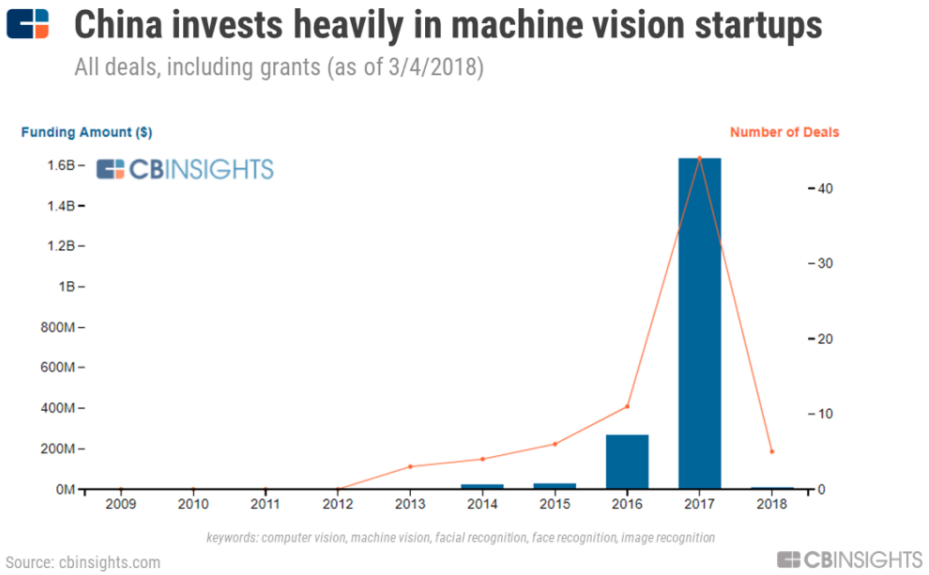


Figura 3.3.4. Crecimiento de la inversión en visión artificial en China por CBINSIGHTS.

Capítulo 4

Soluciones alternativas

Dentro del campo de la seguridad y la vigilancia existen muchas soluciones distintas, además de la videovigilancia existen varios tipos de sensores utilizados para detectar personas o complementar la videovigilancia.

Por otra parte, también se comentarán soluciones de software alternativas al utilizado en este proyecto.

4.1. Sensores

Dentro de los sensores, para la detección de personas los que se utilizan más comúnmente son los siguientes:

1. Sensor de movimiento activo
2. Sensor de movimiento pasivo

1. Sensor de movimiento activo

Existen varios tipos de sensores de movimiento activo dependiendo del tipo de señales que utilicen, pero el funcionamiento de todos estos es el mismo. El sensor envía una señal que debe rebotar en una superficie a una distancia conocida, si algo interfiere con el envío de esta señal el sensor detecta un cambio en el tiempo de retorno de la señal y la alarma se activa. Las señales utilizadas más comúnmente son las microondas y ultrasonidos.

Figura 4.1.1. Sensor de movimiento por microondas a la derecha, por ultrasonidos a la izquierda.

Por otro lado, tenemos también los sensores reflexivos, que utilizan haces de luz con un dispositivo emisor y un receptor, y si este haz de luz es cortado por algún objeto se activa la señal de alarma.

Figura 4.1.2 Sensor de movimiento fotoeléctrico.

2. Sensor de movimiento pasivo

Los sensores de movimiento pasivo son los más comunes en los sistemas de seguridad domésticos. Estos sistemas llamados PIR funcionan detectando cambios de movimiento o calor a través de infrarrojos.

Figura 4.1.3 Sensor de movimiento pasivo PIR.

4.2. Sistemas mixtos

La mayoría de sistemas de seguridad domésticos combinan el uso de los sensores mencionados anteriormente con la videovigilancia, para así obtener una mayor fiabilidad en las detecciones. Estos sistemas de seguridad también suelen ir acompañados de cierres de puertas magnéticos, alarmas de sonido, y sistemas de aviso a las autoridades.

4.3. Software alternativo

Durante el diseño del proyecto se plantearon diferentes alternativas para el software a utilizar. Para más información sobre el software que finalmente se utilizó ver capítulo 8.1. Diseño.

A continuación, se expondrán las más relevantes para cada uno de los siguientes casos:

1. Lenguaje de programación
2. Interfaz de usuario
3. Detección de personas
4. Seguimiento de personas
5. Reconocimiento facial

1. Lenguaje de programación

Al inicio del proyecto se planteó utilizar como lenguaje de programación para el código C++. Este es un lenguaje orientado a objetos con base en C versátil, potente y general. En este lenguaje existen librerías de visión artificial como OpenCV a pesar de no ser el lenguaje de programación más utilizado para este fin.

2. Interfaz de usuario

Para la creación de la interfaz de usuario se contempló utilizar la librería PyQt, por su capacidad de crear aplicaciones similares a las de otras plataformas y porque además funciona del mismo modo que Qt en otros lenguajes. También se contempló el uso de WxPython por tener un diseño más moderno y sencillo.

3. Detección de personas

En el caso de la detección de personas se estudió utilizar **Haar cascade.** Este es el algoritmo que ofrece OpenCV para la detección de objetos, tanto para cuerpo completo, o solo medio cuerpo. Su funcionamiento se basa en un modelo pre entrenado que ha comparado imágenes con personas y otras sin personas como positivas y negativas, y obteniendo unas características básicas de lo que debería ser una persona.

También se planteó el uso de **histogramas de gradientes orientados**, que al igual que el algoritmo Haar cascade viene en las librerías de OpenCV. Este algoritmo busca una función global para describir una persona en lugar de un conjunto de características locales.

4. Seguimiento de personas

Para el seguimiento de personas un algoritmo alternativo es el **filtro de Kalman**. Este algoritmo utiliza mediciones observadas a lo largo del tiempo con ruido y otras imprecisiones y produce estimaciones de variables. Comparando estas estimaciones se obtienen valores empíricos bastante precisos. Estos filtros se utilizan mayoritariamente en navegación, visión artificial y procesado de señales.

5. Reconocimiento facial

Se consideró el **algoritmo LBPH** de reconocimiento facial. Este algoritmo utiliza cuatro parámetros por grupo de píxeles para realizar el reconocimiento: el radio alrededor del pixel central, los pixeles vecinos para encontrar el patrón y el número de píxeles en horizontal y en vertical.

Capítulo 5

Redes neuronales en la visión artificial

5.1. ¿Qué es una red neuronal?

Una red neuronal artificial es como su nombre indica, un conjunto de neuronas artificiales conectadas. Básicamente son algoritmos con muchos parámetros distintos, que al ajustarse de la forma adecuada nos ofrecen soluciones bastante fiables.

Por lo tanto, el uso actual de las redes neuronales no es imitar el funcionamiento de un cerebro, sino resolver problemas en los que el programa debe aprender algo. Por ejemplo, queremos un programa capaz de distinguir entre imágenes que contengan el número 3 del resto de imágenes. Podemos conseguir un programa basado en una red neuronal que sea capaz de diferenciarlas con bastante exactitud.

Esto se logra, por ejemplo, de forma simplificada, teniendo un programa que vaya probando en este caso la respuesta de la red neuronal ante imágenes con el numero 3 y sin este, y vaya modificando los parámetros de estas neuronas según estos resultados. Al final obtenemos una red neuronal capaz de diferenciar estas imágenes, aunque no sepamos cómo, cuántos o por qué se han variado estos parámetros. Este es, uno de los actuales problemas que tienen las redes neuronales, y por lo que no son fiables para casos en los que necesitamos saber el funcionamiento exacto del algoritmo.

Dependiendo del tipo de aprendizaje que buscamos y el tipo de resultados, es más útil un tipo de configuración de red neuronal u otro. Por este motivo existen una amplia variedad de tipos de redes neuronales. Las variaciones en estos tipos van desde cómo funcionan sus neuronas individualmente, a como se realizan las diferentes conexiones entre ellas.

La arquitectura en la que combinamos estas neuronas artificiales, por lo tanto. también es de gran importancia. La forma más básica es teniendo una capa de neuronas de entrada, varias capas internas y por último una única neurona de salida. Cada una de las neuronas de la capa de entrada están conectadas a cada una de las neuronas de la primera capa oculta, teniendo cada una de estas segundas neuronas unos valores de peso y sesgos diferentes para poder obtener una característica concreta de los valores de entrada. Tras una o varias capas más de este tipo se reduce el número de neuronas según las características buscadas hasta terminar en la capa final.

5.2. Tipos de redes neuronales y elección.

Dentro de los diversos tipos de redes neuronales que hay, se explicarán solamente los tipos más relevantes e interesantes en relevancia a este proyecto.

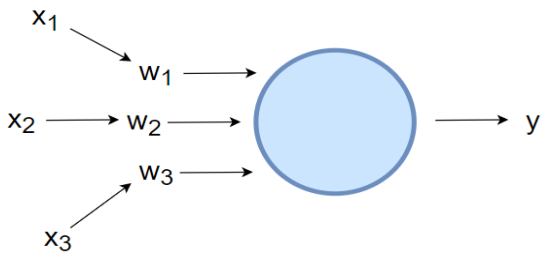
1. Perceptrones
2. Neuronas sigmoides
3. Redes neuronales prealimentadas
4. Redes neuronales recurrentes
5. Redes neuronales convolucionales
6. Redes neuronales convolucionales siamesas

1. Perceptrones

Los perceptrones son el primer modelo de neurona, desarrollado en los cincuenta por Frank Rosenblatt. Entender su funcionamiento es imprescindible para comprender sistemas más complejos de neuronas.

Estas neuronas funcionan de la siguiente forma. Cada una de ellas tiene diferentes entradas binarias (x) y una única salida (y). Cada una de estas entradas es multiplicada por un número real diferente, llamado peso (w).

La suma de todas estas multiplicaciones más un valor de umbral (b) puede ser positiva o negativa. Modificando este sesgo (b) y los diferentes pesos (w) para cada variable obtenemos modelos totalmente distintos. Esto se puede observar en el esquema 5.2.1.



$$y = \begin{cases} 0 \; si \; \sum_j w_j x_j + b \leq 0 \\ 1 \; si \; \sum_j w_j x_j + b \leq 1 \end{cases}$$

Figura 5.2.1. Esquema del funcionamiento de un perceptrón y su ecuación.

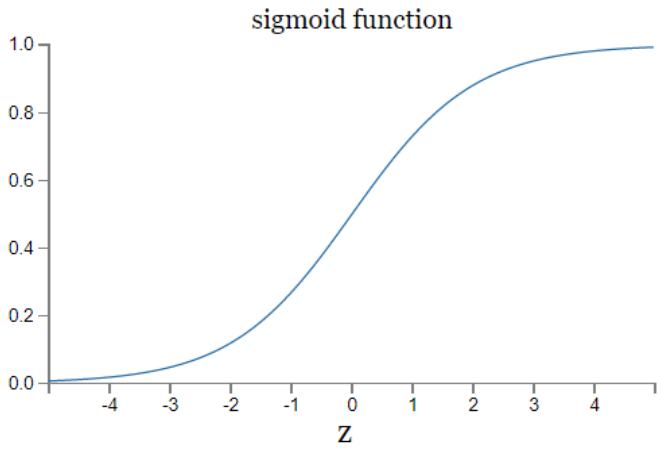
Además, con un conjunto de perceptrones se puede conseguir resolver cualquier operación lógica, ya que podemos formar puertas NAND.

Sin embargo, actualmente no se utilizan perceptrones, porque tener que modificar los pesos y sesgos de forma manual por cada neurona en redes neuronales que son conjuntos de varias capas de cientos de neuronas es demasiado tedioso. Hay que tener en cuenta que cada cambio en uno de estos pesos o sesgos afecta a los siguientes conectados.

2. Neuronas sigmoides

Las neuronas sigmoides son la forma más básica de neurona que se utiliza actualmente. La diferencia con los perceptrones es que los pequeños cambios en las variables de pesos y sesgos no afectan tanto a la salida.

Esto se consigue haciendo que la salida deje de ser una señal binaria y aparezca como resultado de la función sigmoide.



$$\sigma(z) = \frac{1}{1 + e^{z}}$$

Figura 5.2.2.1. Función sigmoide.

Concretamente la salida de una neurona sigmoide viene dada por la función siguiente:

$$\sigma = \frac{1}{1 + e^{(-\sum_j w_j * x_j - b)}}$$

Figura 5.2.2.2. Ecuación de salida de una función sigmoide.

Si consideramos que la salida de los perceptrones tiene una función de escalón, la sigmoide podríamos verla como un escalón más redondeado y curvado, haciendo que como se mencionó, pequeños cambios en los diferentes pesos y sesgos harán un menor cambio en la salida final de la neurona. Esto es un gran cambio, ya que la salida deja de ser un valor binario para ser un valor comprendido entre 0 y 1. Con esto se abre la posibilidad a muchos enfoques diferentes, como por ejemplo obtener como salida la media de intensidad de unos píxeles en una imagen digital.

3. Redes neuronales prealimentadas

Esta es la arquitectura básica de la que hemos hablado anteriormente, con una capa inicial, varias ocultas y una última capa final con una sola neurona. Se denominan prealimentadas porque solo reciben información de la capa anterior.

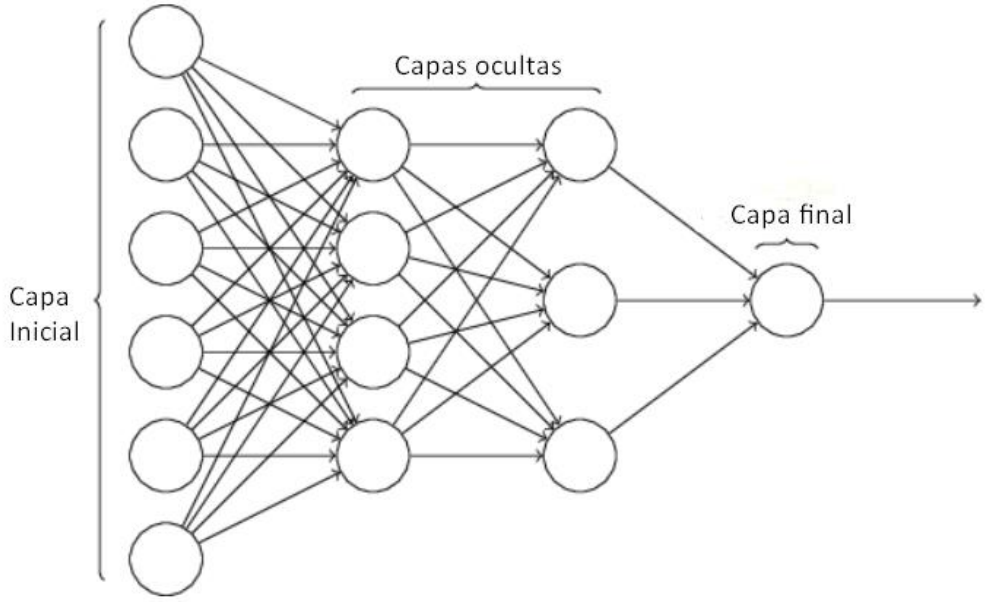


Figura 5.2.3. Esquema de una red neuronal prealimentada.

Los otros tipos de redes neuronales más complejos se diferencian en la arquitectura de estas capas ocultas, ya que con ellas es con lo que creamos los diferentes algoritmos, jerarquías y características.

4. Redes neuronales recurrentes

Las redes neuronales recurrentes están organizadas de forma totalmente diferente a las prealimentadas, por lo que son dos tipos totalmente distintos.

A este tipo de redes neuronales se les llama retroalimentadas ya que son capaces de modificar los valores de peso y sesgo con bucles en cada una de las neuronas, en lugar de utilizar los valores de las capas anteriores.

El problema con estas redes neuronales es que actualmente los algoritmos de aprendizaje que tienen no son tan potentes como los de otros tipos, sin embargo, tienen un gran potencial.

Figura 5.2.4. Esquema de una red neuronal recurrente.

5. Redes neuronales convolucionales

Las redes neuronales convolucionales están específicamente diseñadas para el trabajo con imágenes, ya que, si queremos trabajar por píxeles en una imagen, si esta tiene simplemente 120 píxeles de altura, 120 píxeles de anchura y tres canales de color (120x120x3) necesitaríamos que la primera capa interna tuviese 43200 pesos a configurar, y esto solo para una imagen pequeña.

Estas redes neuronales en lugar de estar formadas por capas internas hechas por un conjunto unidimensional de neuronas como las vistas hasta ahora, están formadas por capas internas tridimensionales, es decir, cada capa tiene una anchura, una altura y una profundidad. Existen varios tipos de estas capas, siendo la principal la capa convolucional.

A las matrices formadas por los distintos pesos de estos conjuntos tridimensionales de neuronas se les llama filtros. Dividiendo la imagen por zonas y haciendo distintas combinaciones se consigue reducir el número de neuronas utilizadas a como sería con una red prealimentada.

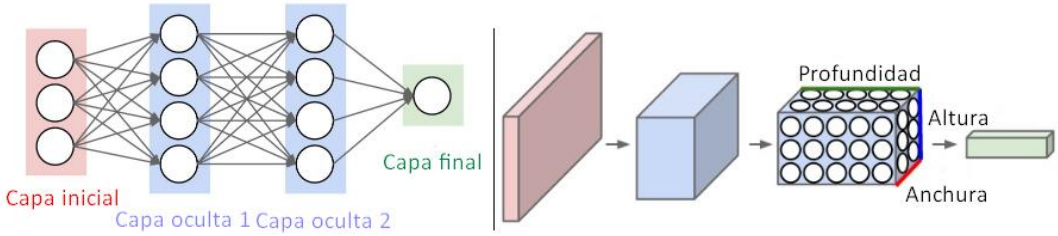


Figura 5.2.5. Esquema de una red neuronal convolucional.

6. Redes neuronales convolucionales siamesas

La característica principal de este tipo de redes neuronales convolucionales se encuentra en que tienen dos o más subredes en las que tanto la arquitectura como los valores de pesos y sesgos son idénticos, de ahí su nombre.

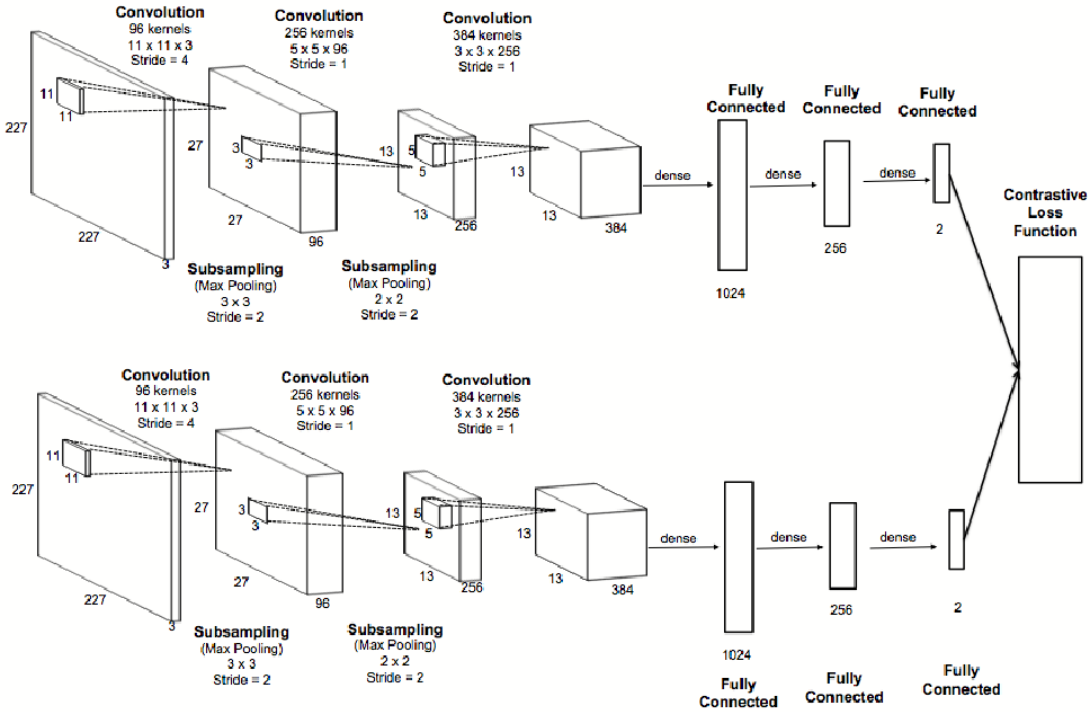


Figura 5.2.6. Esquema de una red neuronal convolucional siamesa.

Para este proyecto se ha decidido utilizar este último tipo de red neuronal convolucional. La justificación es que tenemos dos objetivos a cumplir, primero el reconocimiento facial, y segundo el reconocimiento de personas.

Las redes neuronales siamesas son muy buenas encontrando similitudes, en lugar de aprender por ejemplo lo que es un perro, gato... aprende qué hace iguales a dos perros, dos gatos... y además en lugar de necesitar miles de imágenes de ejemplo como las redes convolucionales normales pueden trabajar con un número muy limitado.

Por estos motivos este tipo de redes están utilizándose mucho en reconocimiento, ya que podemos trabajar incluso con one shot learning, es decir, usando una sola imagen de la persona que queremos reconocer y que sea bastante robusto, ya que puede aprender similitudes semánticas.

Capítulo 6

Requerimientos para el software

La idea inicial de este proyecto fue crear un programa de seguridad y vigilancia que fuese útil tanto para espacios pequeños como para espacios abiertos, haciendo uso de la visión artificial y otras herramientas aprendidas a lo largo de la carrera o por cuenta propia. Las funcionalidades que se buscaban conseguir con este programa son las siguientes:

- Detección de personas
- Seguimiento de personas
- Reconocimiento y detección facial
- Interfaz gráfica de usuario
- Control de la escena a través de la creación de diferentes zonas

Otros requerimientos clave fueron que la exigencia de procesamiento del programa fuese la menor posible, pero con un modelo lo más robusto posible o que la interfaz fuese sencilla de utilizar a pesar de la complejidad que tenga el código, además este código debía ser lo más intuitivo posible para futuros cambios.

El programa también tenía que ser capaz de mostrar cuando alguna persona entraba en una zona restringida, o en la escena en si durante el modo de vigilancia.

Con tal de comprobar el correcto funcionamiento del programa diseñado se realizaron las siguientes pruebas:

1. Evaluación de la detección y seguimiento
2. Evaluación de la detección y reconocimiento facial
3. Evaluación de las diferentes zonas

Capítulo 7

Instalación

El código del programa está preparado para poder ser ejecutado desde Windows Powershell 10, pero para poder iniciar el programa primero se ha de instalar el lenguaje de programación Python 3. Para esto descargaremos la última versión disponible desde la página oficial: www.python.org/downloads/windows.

Una vez instalado Python procedemos a instalar las librerías necesarias para poder ejecutar el código. Para instalarlas se han de seguir los siguientes pasos:

- Iniciar Windows PowerShell. Esta interfaz de consola viene ya instalada en todos los equipos Windows, así que solo es necesario buscarlo en Windows e iniciarlo.

- Una vez dentro de Windows PowerShell, vamos al directorio en el que se encuentra la carpeta con los archivos del programa a través del comando "cd" seguido del nombre de la carpeta.

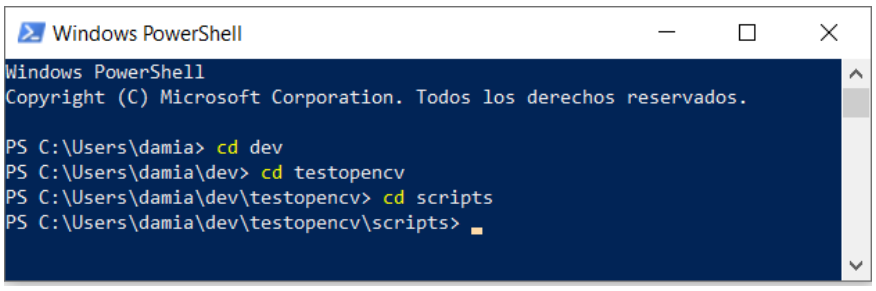


Figura 7.1. Entrar en el directorio en Windows PowerShell.

- Una vez en la carpeta de los archivos descargamos e instalamos las librerías necesarias listadas a continuación con los siguientes comandos en Windows PowerShell:

  Primero "python -m pip install --upgrade pip" para actualizar el instalador de librerías y módulos. A continuación, instalamos todas las librerías ejecutando el comando "pip install -r requirements.txt". Con este comando instalaremos directamente todas las librerías necesarias que ya están listadas en ese documento en lugar de tener que instalarlas una a una. En caso de que alguna esté desactualizada, la propia consola nos avisará y nos dará el comando para actualizarla.

  Finalmente, para iniciar el programa solo es necesario ejecutar el siguiente comando en caso de querer utilizar la primera cámara configurada en el PC, normalmente la CAM:

  "python testgui.py --prototxt
  mobilenet_ssd/MobileNetSSD_deploy.prototxt --model
  mobilenet_ssd/MobileNetSSD_deploy.caffemodel"

  O la siguiente línea en caso de querer utilizar uno de los vídeos de ejemplo:

  "python testgui.py --prototxt
  mobilenet_ssd/MobileNetSSD_deploy.prototxt --model
  mobilenet_ssd/MobileNetSSD_deploy.caffemodel --input
  videos/example_01.mp4"

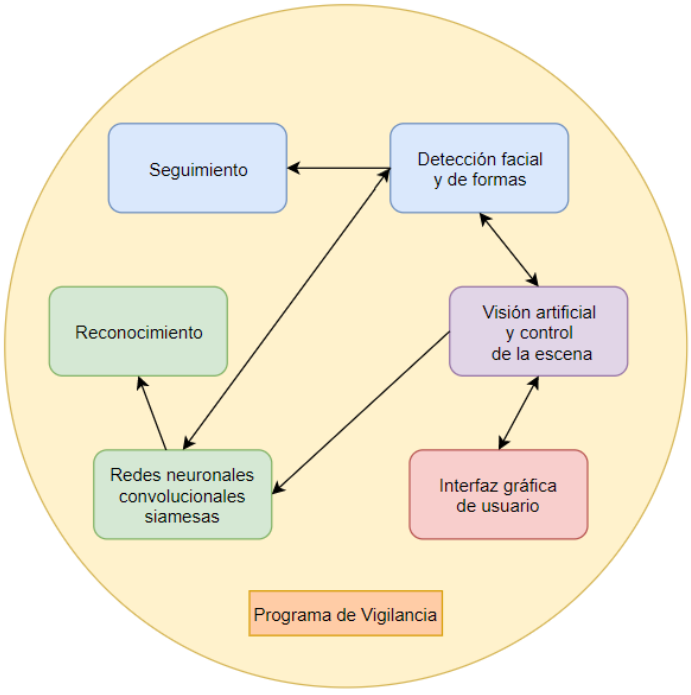
  Siendo "example_01.mp4 el nombre del vídeo que se va a pasar por el programa.

Capítulo 8

Desarrollo de la aplicación de vigilancia

8.1. Diseño

Para el diseño de este proyecto se debían crear y unificar los diferentes módulos requeridos en un solo programa. Para ello se buscó un lenguaje de programación en el que fuese sencillo trabajar con estas herramientas y del que existiese documentación. Por estos motivos se decidió utilizar Python, ya que es muy intuitivo y existen muchas librerías y documentación sobre redes neuronales y visión artificial.

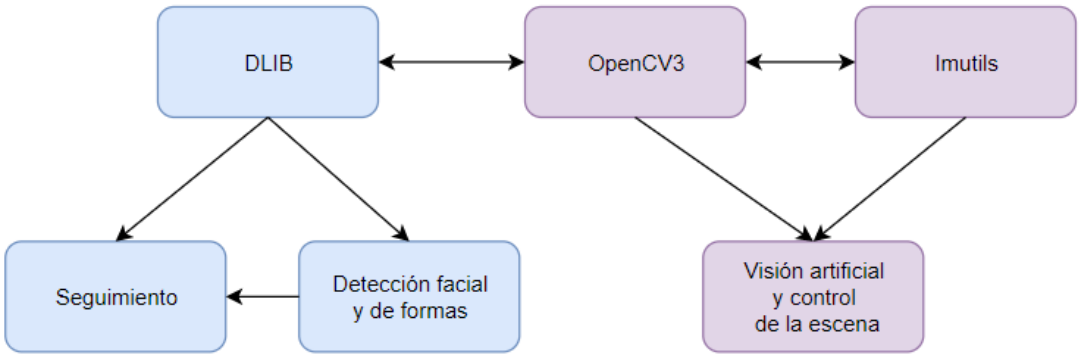


Esquema 8.1.1. Esquema básico del programa.

Para el diseño de estas funciones, se utilizaron diferentes librerías de Python. La principal en este proyecto es la librería OpenCV para la visión artificial, ya que existe mucha documentación y es una librería multiplataforma con funciones propias para el reconocimiento de objetos entre otras. Del mismo modo para toda la interfaz de usuario se ha usado la librería Tkinter, que es la interfaz gráfica por defecto y más utilizada en Python.

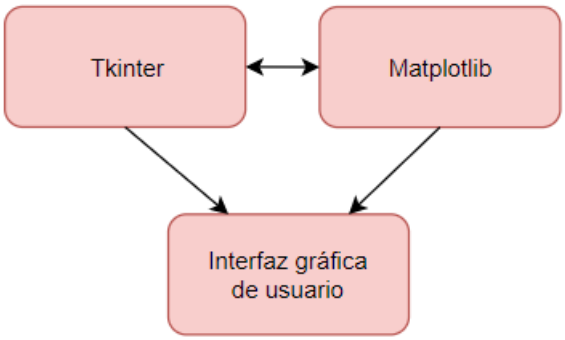
Junto con OpenCV se utilizó la librería imutils, que añade varias funciones para el procesado de imagen y reajustado entre otras.

Para los modelos de detección, tanto la detección facial, como la detección de personas y el seguimiento se ha usado la librería DLIB, que se suele utilizar en conjunto con OpenCV. Para el diseño de este programa se realizaron pruebas con estos modelos, los modelos propios de detección de OpenCV y algunos más de otras librerías. Pero los modelos de DLIB fueron los que dieron mejores resultados de detecciones, tanto en detecciones correctas como en falsos positivos, falsos negativos y malas detecciones.



Esquema 8.1.2. Esquema de las librerías para visión, detección y seguimiento.

Del mismo modo se usó la librería Matplotlib en conjunto a Tkinter para el diseño gráfico del programa.



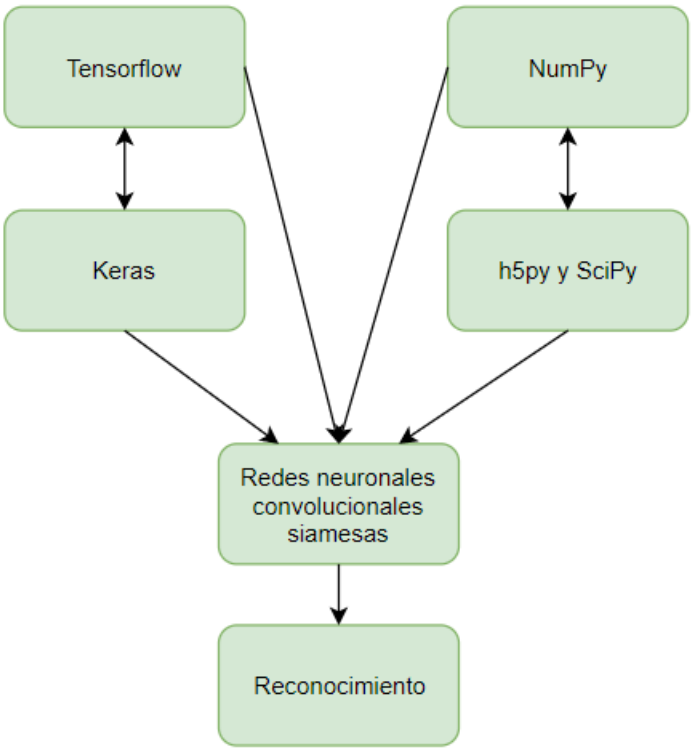
Esquema 8.1.3. Esquema de las librerías de interfaz de usuario.

Se utilizó la librería NumPy para leer y trabajar con las matrices que forman los modelos de las redes neuronales. Es una de las librerías más conocidas y utilizadas para usar de forma eficiente contenedores de información multidimensional.

También se han usado las librerías h5py y SciPy que trabajan de forma conjunta con NumPy para manipular de forma más sencilla la información almacenada.

La primera librería utilizada para el manejo de redes neuronales es TensorFlow. Esta es una de las librerías de aprendizaje profundo más populares, sobre todo por la cantidad de soporte y documentación disponible ya que fue creada por Google.

Por último, con la librería TensorFlow se ha utilizado la librería Keras. Es de esta librería de dónde hemos elegido los modelos de redes neuronales siamesas para este proyecto. Keras permite añadir de forma simple nuevos módulos de capas de redes neuronales, funciones de coste, de activación... Además, tiene una amplia documentación y es simple e intuitiva.



Esquema 8.1.4. Esquema de las librerías de redes neuronales y reconocimiento.

8.1.1. Interfaz de usuario

La misión de este módulo es hace más sencillo el uso del programa para realizar todas las funciones disponibles a un usuario. Estas funciones incluyen la creación de las diferentes zonas, modificar los parámetros necesarios, activar o desactivar los modos de vigilancia, añadir personas a la base de datos, reconocer personas designadas, etc. Para más información sobre el uso de la interfaz de usuario ver anexo 1.

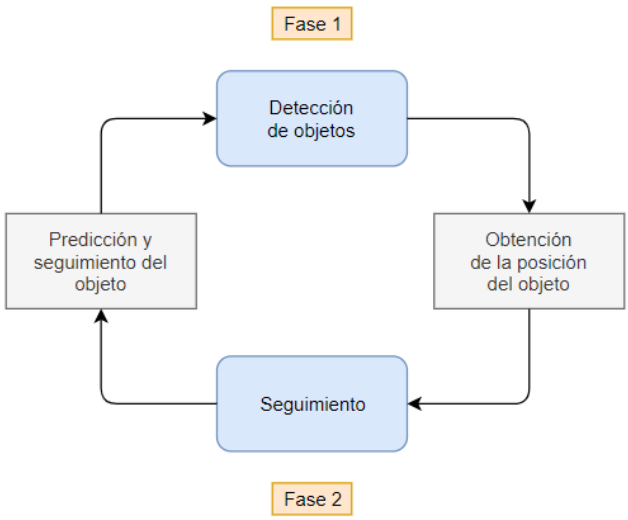
8.1.2. Detección y seguimiento

La detección de objetos, en este caso personas, en visión artificial trata de determinar dónde se encuentra este objeto en una imagen o fotograma. Los algoritmos de detección de objetos son computacionalmente más demandantes y por lo tanto más lentos que los algoritmos de seguimiento.

Por otro lado, el algoritmo de seguimiento lo que hace es una vez conocida la posición de un objeto en la imagen, asignar una identificación a ese objeto en especial, y seguir su movimiento a lo largo del video prediciendo la localización nueva del objeto en los siguientes fotogramas a través de atributos de los pixeles.

En este proyecto se han combinado ambos dividiendo el algoritmo de seguimiento en dos fases. Una primera fase de detección, que es computacionalmente más exigente para encontrar los objetos que se buscan mediante una red neuronal siamesa, y ver si hay algún objeto perdido en la fase de seguimiento anterior. Para no malgastar recursos esta fase solo ocurre cada varios fotogramas, en los que para cada objeto detectado actualizamos el seguimiento con los nuevos límites de la detección. En la segunda fase se hace el seguimiento de los diferentes objetos detectados anteriormente. El algoritmo de seguimiento debe ser más rápido y eficiente que el detector. Una vez terminado el proceso se vuelve a repetir.

Con este método mixto conseguimos un método de detección y seguimiento eficiente en cuanto a recursos computacionales y precisión.



Esquema 8.1.1.2. Esquema del algoritmo de detección y seguimiento.

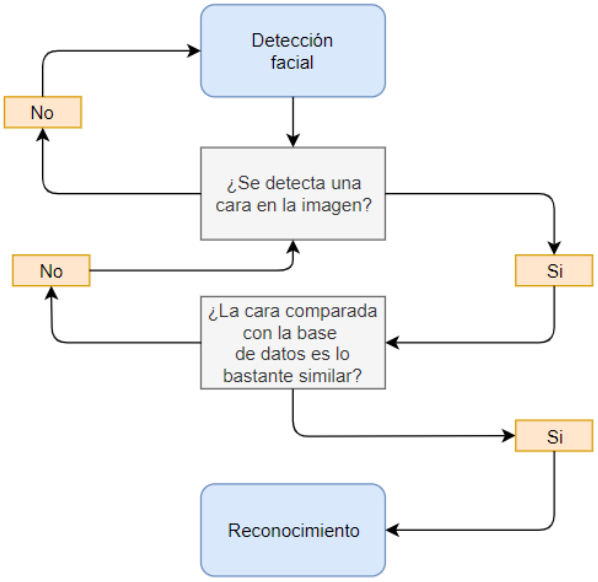
8.1.3. Detección y reconocimiento facial

La detección facial funciona del mismo modo que la detección de objetos, pero concretamente buscando formas similares a caras en la escena. En este caso utilizamos un modelo para detectar rostros de frente para facilitar el reconocimiento, pero existen modelos para detectar perfiles y otras perspectivas.

El reconocimiento facial pasa por varias fases para evitar falsos positivos, primero se ha de detectar que hay una persona en la escena. Después se detecta la cara de esta persona, y finalmente se compara la cara detectada con las caras en la base de datos, a través de la red neuronal convolucional siamesa.

Esta base de datos se puede ampliar desde el propio programa de forma sencilla, y se puede variar la cantidad de imágenes del individuo que se desean utilizar para que el algoritmo las compare.

La discriminación entre si el individuo ha sido reconocido o no se realiza a través de un umbral de coincidencia, con un valor obtenido empíricamente. El algoritmo calcula la similitud entre la cara detectada del fotograma actual y la base de datos y devuelve un valor. Si este valor es superior al umbral se considera que la persona ha sido reconocida como la de la base de datos, en caso contrario no es reconocida.



Esquema 8.1.1.3. Esquema del algoritmo de detección y reconocimiento facial.

8.1.4. Creación de diferentes zonas

Las diferentes zonas que se pueden crear con el programa utilizan las funciones de reconocimiento, detección y seguimiento explicadas en los apartados anteriores en conjunto con la interfaz de usuario, por lo que no se han incluido en el esquema 7.1. para simplificarlo. Se diseñaron los siguientes tipos de zonas:

- Zona prohibida. Cuando se detecta una persona y esta entra dentro de esta zona delimitada se activa una alarma. Esta zona tiene la intención de servir para proteger zonas concretas de la escena, como zonas de alta tensión que son peligrosas dentro de una fábrica, o puertas a las que no se debería acceder entre otros.

- Zona de interacción. Una señal se activa cuando hay más de una persona dentro de esta zona lo suficientemente cerca. La utilidad de esta zona está pensada más para el estudio y la obtención de datos sobre interacciones en zonas públicas o centros comerciales para obtener estadísticas.

- Zona de trabajo. Para crear esta zona primero se debe definir el número de personas que deben ocupar esta zona de trabajo y la duración en la que deben estar en ella. Una vez definido esto y creada, se activará una alarma cuando el número de personas dentro de la zona no sea el estipulado durante el tiempo diseñado. Esta zona de trabajo tiene la intención de servir para controlar mejor los fichajes de personas con puestos fijos, o también por ejemplo la atendencia de alumnos a una clase.

- Zona exclusiva. Esta es una versión avanzada de la zona prohibida, donde solo podrá acceder a esta zona una persona que haya sido previamente identificada por el reconocimiento facial como una persona con autoridad para entrar en dicha zona.

8.1.5. Otras utilidades

Además de las funciones explicadas previamente, el programa tiene capacidad para definir un aforo máximo de personas detectadas, y si este es superado alertar.

Otra de las utilidades que tiene el programa son dos modos de vigilancia. Un modo de vigilancia en el que si cualquier persona es detectada en la escena se active una alarma, y otro modo de vigilancia avanzado en el que además de hacer esto, el programa obtiene una imagen de la persona que entre en la escena y la envía a un correo electrónico previamente introducido.

El programa también tiene la capacidad de cambiar el objeto de detección, a perros y/o gatos además de personas para poder controlar comportamientos no deseados de estos animales, que vayan a zonas no deseadas o incluso para poder estudiar posibles problemas de ansiedad.

También se han incluido algunas opciones para el operador del programa para modificar el brillo y contraste de la imagen, y el canal de color para obtener unas detecciones mejores dependiendo del escenario de forma más sencilla.

Finalmente, se añadió la posibilidad de añadir trazos de perspectiva cónica con varios focos y cantidad de aristas para una posible comparación de la situación de la persona en la pantalla con la situación en el mundo real en un futuro proyecto. Para más información ver anexo 1.

8.2. Implementación

Se ha dividido la implementación de las funciones más importantes del programa en cinco apartados:

8.2.1. Interfaz de usuario

La interfaz de usuario, programada utilizando Tkinter en Python está dividida en tres pestañas distintas, una para los usos más simples, otra con más aplicaciones y una última con utilidades para mejoras y futuras funciones. Para más información ver anexo 1.

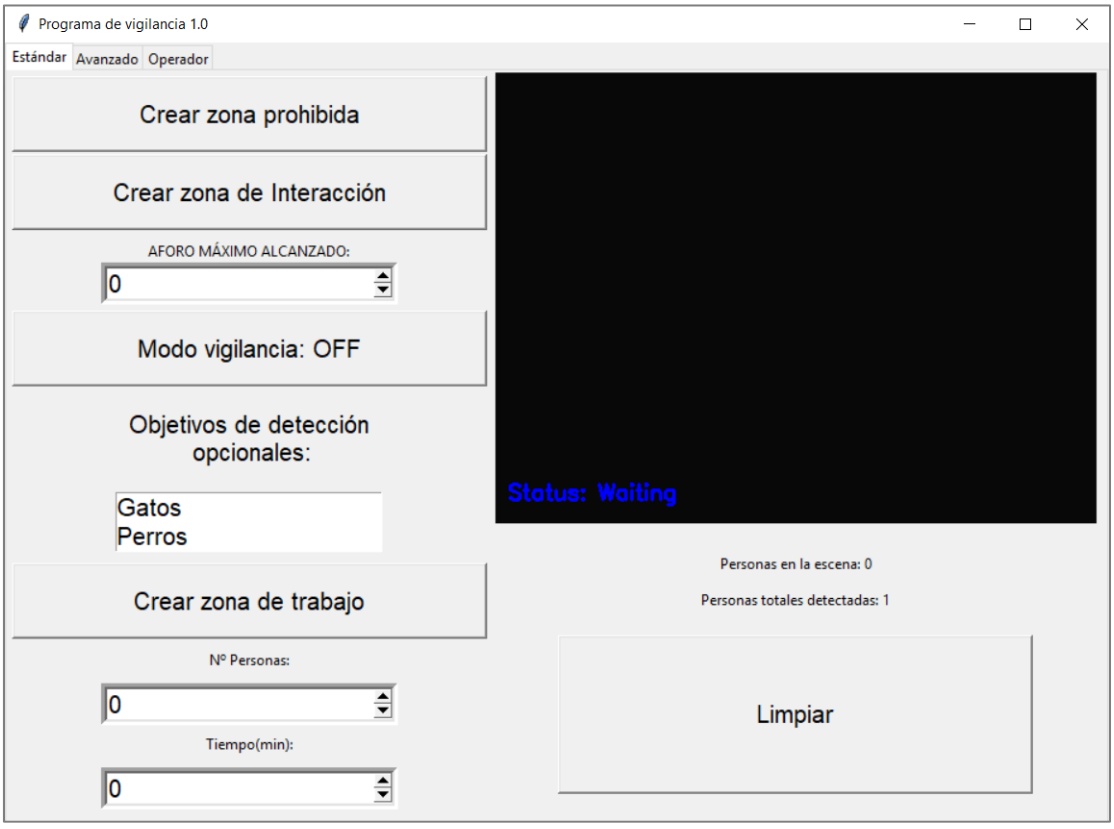


Figura 8.2.1. Interfaz gráfica de usuario, pestaña standard.

8.2.2. Detección y seguimiento

A continuación, se mostrarán y explicará la implementación en el código más importante de estas dos funciones. Primero definimos modelo de la red neuronal siamesa utilizado para detectar las personas y el algoritmo para predecir las trayectorias en el seguimiento.

En este caso el modelo concreto se especifica en la ejecución del código, el modelo es "mobilenet_ssd/MobileNetSSD_deploy.caffemodel". Del mismo modo creamos una instancia del seguidor del centroide del objeto detectado, y una lista para almacenar los seguidores de los distintos centroides y asignar una identificación especial a cada uno.

```
shape_predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
face_aligner = FaceAligner(shape_predictor, desiredFaceWidth=200)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

Código 8.2.2.1. Inicialización de modelos.

```
ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
trackers = []
trackableObjects = {}
```

Código 8.2.2.2. Inicialización del seguidor y almacenamiento.

Una vez hecho esto se inicia el estado del programa en modo espera para ahorrar recursos, y cada número definido de fotogramas cambia el estado al modo de detección. Dentro de este modo se convierte la imagen y se compara con el modelo de la red neuronal para obtener las detecciones.

```
# Inicializar el estado actual y la lista de delimitación
status = "Waiting"
rects = []
cantidad = 0

#  Metodo de detección para ayudar al tracker
if totalFrames % args["skip_frames"] == 0:
    # Cambiar el estado y definir el tracker
    status = "Detecting"
    trackers = []
    # Convertir el frame en un blob y pasar este através de la Red
    # neuronal para obtener las detecciones
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
    0.007843, (300, 300), 127.5)
    net.setInput(blob)
    detections = net.forward()
```

Código 8.2.2.3. Iniciar el estado y las detecciones.

Ahora que ya tenemos las detecciones se realiza un bucle a través de todas ellas, se calcula el valor de confianza para compararlo con el umbral definido, y si lo superan se dibuja una caja a su alrededor y se les da un ID para poder visualizarlo.

```
# Bucle a través de las detecciones
for i in np.arange(0, detections.shape[2]):
    # Extraer el valor de confianza (probabilidad) asociado
    # con la predicción
    confidence = detections[0, 0, i, 2]
    # Filtrar detecciones debiles bajo el nivel de confianza
    if confidence > args["confidence"]:
        # Obtener el indice de la lista de clases de
        # posibles detecciones
        idx = int(detections[0, 0, i, 1])
        # Calcula y dibuja un rectangulo por objeto detectada
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        cv2.rectangle(frame, (startX, startY), (endX, endY),
        COLORS[idx], 2)
```

Código 8.2.2.4. Bucle a través de las detecciones.

Una vez tenemos las detecciones que queremos se inicia el seguidor, que utilizará el centroide del objeto detectado anteriormente. Una vez comienza un seguimiento el estado del programa pasa al de seguimiento, y el seguidor se actualiza a lo largo de los fotogramas.

```
tracker = dlib.correlation_tracker()
                rect = dlib.rectangle(startX, startY, endX, endY)
                tracker.start_track(cv2image, rect)
                # Añadir el tracker a nuestra lista para poder usarlo
                # durante frames faltantes
                trackers.append(tracker)
    # En caso contrario usar los trackers en lugar de los detectores
    # para mejorar el rendimiento del procesamiento de frames
    else:
        # Bucle entre los trackers
        for tracker in trackers:
            # Cambiar el estado del sistema a 'tracking' en lugar de
            # 'waiting' o 'detecting'
            status = "Tracking"
            # Actualizar el tracker y coger la posición actualizada
            tracker.update(cv2image)
            pos = tracker.get_position()
            # Sacar la posición del objeto
            startX = int(pos.left())
            startY = int(pos.top())
            endX = int(pos.right())
            endY = int(pos.bottom())
            # Añadir las coordenadas limitantes y actualizar el rectangulo
            rects.append((startX, startY, endX, endY))
    #  Usa el centroide del tracker para asociar el objeto detectado antiguo
    # con el centroide nuevo
    objects = ct.update(rects)
```

Código 8.2.2.5. Seguimiento del objeto a través del centroide

Finalmente almacenamos la posición del seguidor junto con su identificación para utilizarlo en la siguiente iteración.

```
to = trackableObjects.get(objectID, None)

# Si no existe crea uno
if to is None:
    to = TrackableObject(objectID, centroid)
# Si existe lo podemos usar para determinar la dirección
# Guarda el objeto en el diccionario
trackableObjects[objectID] = to
```

Código 8.2.2.6. Actualización del seguidor.

8.2.3. Detección y reconocimiento facial

Para la detección de caras también necesitamos definir qué algoritmo de detección vamos a utilizar, así como el modelo de red neuronal que vamos a usar. En nuestro caso utilizamos un detector de la librería DLIB, y el clasificador de OpenCV Haar cascade para detectar caras frontales en conjunto.

```
# Definir el detector facial usado de la libreria dlib
detector = dlib.get_frontal_face_detector()
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml')
# Definir el modelo de red neuronal preentrenado para el reconocimiento facial
FR_model = load_model('nn4.small2.v1.h5')
```

Código 8.2.3.1. Definir el detector y la red neuronal para el reconocimiento facial.

Una vez definidos, dentro del bucle de detección de personas explicado en el punto anterior, creamos otro bucle que busque formas similares a caras. De esta forma no buscamos caras por todo el fotograma.

Si se activa el reconocimiento facial, para cada una de las caras detectadas se obtiene un valor de similitud con todas las caras de la base de datos a través de la red neuronal siamesa. En caso de que alguna de estas coincidencias sea superior al valor umbral definido, se identifica a la persona, en caso contrario se cierra el bucle. Cuando la persona es identificada, se muestra en la pantalla el nombre asociado en la base de datos, así como el valor de coincidencia obtenido.

```
for (objectID, centroid) in objects.items():
    if buscandoNombre:
        faces = face_cascade.detectMultiScale(frame, 1.3, 5)
        for(xface,yface,wface,hface) in faces:
            cv2.rectangle(frame, (xface, yface), (xface + wface, yface + hface),
            (255, 255, 0), 2)
            roi = frame[yface:yface+hface, xface:xface+wface]
            encoding = img_to_encoding(roi, FR_model)
            min_dist = 100
            identity = None

            for(name, encoded_image_name) in face_database.items():
                dist = np.linalg.norm(encoding - encoded_image_name)
                if(dist < min_dist):
                    min_dist = dist
                    identity = name

            if min_dist < 0.08:
                cv2.putText(frame, "Face : " + identity[:-1],
                (xface, yface - 50), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)
                cv2.putText(frame, "Dist : " + str(min_dist),
                (xface, yface - 20), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)

                if identity[:-1] == identname:
                    recontext = True
                else:
                    recontext = False

            else:
                cv2.putText(frame, 'No se reconoce',
                (xface, yface - 20), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 0, 255), 2)
```

Código 8.2.3.2. Bucle de detección y reconocimiento facial.

8.2.4. Creación de diferentes zonas

Para la creación de estás zonas de una forma sencilla e intuitiva se utilizó el puntero, después de seleccionar el tipo de zona que se desea crear se hace clic en la imagen con el puntero, y la zona se dibujará como un rectángulo cuando se vuelva a realizar un segundo clic. De esta forma se crea un rectángulo siendo las posiciones elegidas esquinas opuestas. Para más información ver anexo 1.

Por lo tanto, primero se debe obtener la posición del puntero. Para ello se creó la siguiente función:

```
def motion(event):
    global x
    global y
    x, y = event.x, event.y
```

Código 8.2.4.1. Obtención de la posición del puntero.

Además de esto, se ha definido una clase para las zonas, que incluye esta posición inicial del puntero y su posición final. Para cambiar el efecto de cada una de las zonas se creó una máquina de estados, que dependiendo de la zona que se elija crear cambia el estado para realizar una función u otra.

```
class zona():
    def __init__(self, box, vectorInicio, vectorFinal):
        self.box          = box;
        self.vectorInicio = vectorInicio
        self.vectorFinal  = vectorFinal
```

Código 8.2.4.2. Definición de la clase zona.

```
# Definir la función que cambia al estado a zona prohibida
def estadoEliminar():
    global estado
    estado = 1
# Definir la función que cambia el estado a zona de interacción
def estadoInteraccion():
    global estado
    estado = 2
# Definir la función que cambia el estado a zona de trabajo
def estadoTrabajo():
    global estado
    estado = 3
# Definir la función que cambia el estado a vigilancia avanzada
def estadoAvanzado():
    global estado
    global buscandoNombre
    global identnameP
    estado = 4
    identnameP = EntryName2.get()
    buscandoNombre = not buscandoNombre
```

Código 8.2.4.3. Funciones para cambiar el estado según el tipo de zona.

```
def crearCuadro(event):
    global posInicial
    global posFinal
    global x
    global y
    global estado

    if(estado == 0):
        return
    if(posInicial == False):
        posInicial = {"x":x, "y":y}
    else:
        posFinal = {"x":x, "y":y}
        if(estado != 0):
            if(estado == 1):
                listaDeZonasProhibidas.append(zona(0, posInicial, posFinal))
            if(estado == 2):
                listaDeZonasDeInteraccion.append(zona(0, posInicial, posFinal))
            if(estado == 3):
                listaDeZonasTrabajo.append(zona(0, posInicial, posFinal))
                start = time.time()
            if(estado == 4):
                listaAvanzada.append(zona(0, posInicial, posFinal))

            posInicial = False
            posFinal   = False
            estado = 0
```

Código 8.2.4.4. Definición de la máquina de estado y obtención de la posición de las zonas.

En todas las zonas se debe comprobar si hay personas dentro, para esto comprobamos si la posición del centroide del objeto detectado para el seguimiento de personas está dentro de la zona definida previamente con la siguiente función:

```
# Función para comprobar si el centroide está dentro de la zona creada
def comprobarSiEstaEnZona(zona,x,y):
    for p in zona:
        if((x >= p.vectorInicio["x"]) and (x <= p.vectorFinal["x"]) and (y >= p.vectorInicio["y"]) and (y <= p.vectorFinal["y"])):
            return True
    return False
```

Código 8.2.4.5. Función para comprobar la posición de personas dentro de las zonas.

Una vez sabemos la cantidad de personas dentro de la zona, dependiendo del tipo de zona que se trate realizamos una acción u otra. En la zona que detecta interacciones si las hay, en este caso dos o más personas, lo bastante cerca se da un aviso.

En el caso de la zona prohibida se avisa si se detecta cualquier persona en su interior, y en la zona prohibida avanzada se da un aviso si la persona que ha entrado en la zona está autorizada mediante el reconocimiento facial o si no lo está.

En la zona de trabajo se define el tiempo que esta zona permanecerá activa y el número de personas que debe haber dentro, en caso de que este número cambie a mayor o menos se avisará.

```
# Si hay varias personas muy cerca en la zona avisa
if(personasJuntas > 1):
    cv2.putText(frame, "INTERACCIÓN ENTRE PERSONAS", (220, 20),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

# Si hay alguien en la zona prohibida crea una alerta
if(comprobarSiEstaEnZona(listaDeZonasProhibidas, centroid[0] - 10, centroid[1] - 10)):
    cv2.putText(frame, "ALERTA ZONA PROHIBIDA", (20, 20),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

# Si hay alguien en la zona prohibida crea una alerta a no ser
# que sea una persona autorizada
if(comprobarSiEstaEnZona(listaAvanzada,centroid[0] - 10, centroid[1] - 10)):
    if (identity[:-1] == identnameP):
        cv2.putText(frame, "PERSONA AUTORIZADA", (20, 20),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
    else:
        cv2.putText(frame, "ALERTA ZONA PROHIBIDA AVANZADA", (20, 20),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

# Alerta si no se cumplen los requerimientos de la zona de trabajo
if(comprobarSiEstaEnZona(listaDeZonasTrabajo,centroid[0] - 10, centroid[1] - 10)):
    personasTrabajo = personasTrabajo + 1
    #Calcula el tiempo
    if(WorkStart == 0):
        StartTime = time.time()
        EndTime = int(EntryTiempo.get())*60
    Time1 = time.time()
    if(EndTime-(Time1-StartTime) <= 0):
        WorkStart = 0
        Time1 = 0
        listaDeZonasTrabajo = []
    WorkStart = WorkStart + 1
    if(personasTrabajo > int(EntryPersonas.get())):
        cv2.putText(frame, "ALERTA DEMASIADAS PERSONAS", (220, 20),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

    if(personasTrabajo < int(EntryPersonas.get())):
        cv2.putText(frame, "ALERTA FALTAN PERSONAS", (220, 20),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
```

Código 8.2.4.6. Activación de avisos dependiendo del tipo de zona.

Por último, para dibujar en la pantalla los diferentes recuadros en la zona creada se utiliza un bucle en el que se dibuja un rectángulo de diferentes colores según el tipo de zona con los vectores obtenidos con el puntero.

```
    # Crea el rectangulo de las diferentes zonas
for cosas in listaDeZonasProhibidas:
    cv2.rectangle(frame, (cosas.vectorInicio["x"], cosas.vectorInicio["y"]),
    (cosas.vectorFinal["x"], cosas.vectorFinal["y"]),(255,255,255) , 3)

for cosas in listaDeZonasDeInteraccion:
    cv2.rectangle(frame, (cosas.vectorInicio["x"], cosas.vectorInicio["y"]),
    (cosas.vectorFinal["x"], cosas.vectorFinal["y"]),(31,96,250) , 3)

for cosas in listaDeZonasTrabajo:
    cv2.rectangle(frame, (cosas.vectorInicio["x"], cosas.vectorInicio["y"]),
    (cosas.vectorFinal["x"], cosas.vectorFinal["y"]),(0,0,0) , 3)

for cosas in listaAvanzada:
    cv2.rectangle(frame, (cosas.vectorInicio["x"], cosas.vectorInicio["y"]),
    (cosas.vectorFinal["x"], cosas.vectorFinal["y"]),(20,20,80) , 3)
```

Código 8.2.4.7. Función para dibujar los diferentes rectángulos.

8.2.5. Otras utilidades.

La función de controlar el aforo máximo compara la cantidad de gente detectada en la escena con el valor máximo que se define en la interfaz. En caso de que haya un mayor número de personas en la escena se realiza un aviso.

```
# Si el aforo detectado es mayor al definido crea un aviso
if(int(SpinAforo.get()) > 0 and cantidad > int(SpinAforo.get())):
    cv2.putText(frame,"AFORO MÁXIMO ALCANZADO!", (90,10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

Código 8.2.5.1. Función avisar en caso de aforo máximo.

El modo de vigilancia básico detecta cualquier persona en la escena y realiza un aviso del mismo modo que el aforo máximo. Sin embargo, el modo de vigilancia avanzado además ha de obtener una imagen del intruso y enviarla a un correo definido en la interfaz. Esto se consigue de la siguiente forma. Primero se definen las características del correo.

```
# Función para enviar un mail al correo designado
def captureToEmail():
    msg = MIMEMultipart()
    msg['From'] = "test@andaungato.meow"
    msg['To'] = EntryMail.get()
    msg['Subject'] = Header('Un intruso ha aparecido en el campo de visión de la camara', 'utf-8').encode()
    msg_content = MIMEText('Esta la cara detectada por el programa.', 'plain', 'utf-8')
    msg.attach(msg_content)
    directory = "emailImage"
```

Código 8.2.5.2. Definición del correo.

Varios fotogramas más tarde, para evitar que la imagen se vea borrosa se obtiene una imagen de la cara del intruso, se añade al correo y es enviado.

```
if not os.path.exists(directory):
    os.makedirs(directory, exist_ok = 'True')
# Capturar una imagen para enviar al correo
frameaa = vs.read()
frameaa = imutils.resize(frameaa, width=500)
frameaa = cv2.flip(frameaa, 1)
framea_gray = cv2.cvtColor(frameaa, cv2.COLOR_BGR2GRAY)
# Detectar la cara del intruso
faces = face_cascade.detectMultiScale(frameaa, 1.3, 5)
faces = detector(framea_gray)
# Si se detecta una cara enviar el correo
if len(faces) == 1:
    face = faces[0]
    (xaa, yaa, waa, haa) = face_utils.rect_to_bb(face)
    face_img = framea_gray[yaa-50:yaa + haa+100, xaa-50:xaa + waa+100]
    face_aligned = face_aligner.align(frameaa, framea_gray, face)
    nombreArchivo = ''.join(random.choices(string.ascii_uppercase + string.digits, k=30))
    cv2.imwrite(os.path.join(directory, str(nombreArchivo+'.jpg')), face_aligned)
    with open('emailImage/'+nombreArchivo+'.jpg', 'rb') as f:
        mime = MIMEBase('image', 'png', filename='img1.png')
        mime.add_header('Content-Disposition', 'attachment', filename='img1.png')
        mime.add_header('X-Attachment-Id', '0')
        mime.add_header('Content-ID', '<0>')
        mime.set_payload(f.read())
        encoders.encode_base64(mime)
        msg.attach(mime)

    server = smtplib.SMTP('smtp.gmail.com:587')
    server.ehlo()
    server.starttls()
    server.set_debuglevel(1)
    server.login("user", "pasword")
    server.sendmail("user", msg["To"], msg.as_string())
    server.quit()
    return False
```

Código 8.2.5.3. Obtención de la imagen y envío del correo.

El cambio de objeto a detectar, entre personas, perros y gatos se realiza cambiado el modelo de la red neuronal asignado a buscar mediante la interfaz de usuario.

```
ObjectToDetect = DetectionCheck.curselection()
if (len(ObjectToDetect)>=1):

    if ((ObjectToDetect[0])==1 and len(ObjectToDetect)==1):
        StringClass1 = "dog"
        StringClass2 = "person"
    if ((ObjectToDetect[0])==0 and len(ObjectToDetect)==1):
        StringClass1 = "cat"
        StringClass2 = "person"
    if (len(ObjectToDetect)>1):
        StringClass1 = "cat"
        StringClass2 = "dog"
# Si esta clase no es una persona, perro o gato ignorarlo
options = ["person" ,StringClass1 ,StringClass2]
try:
    value_index = options.index(CLASSES[idx])
except:
    value_index = -1;
if (value_index < 0):
    continue
```

Código 8.2.5.4. Cambio del objetivo a buscar en la escena.

Las modificaciones de canal de color se realizan a través de OpenCV.

```
if(bnw):
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame = frame_gray
else:
    b,g,r = cv2.split(frame)
    ChannelColor = ColorCheck.curselection()
    if (len(ChannelColor)==1):
        if ((ChannelColor[0])==0 and len(ChannelColor)==1):
            frame = r
        if ((ChannelColor[0])==1 and len(ChannelColor)==1):
            frame = g
        if ((ChannelColor[0])==2 and len(ChannelColor)==1):
            frame = b
    if(len(ChannelColor)==2):
        if ((ChannelColor[0])==0 and (ChannelColor[1])==1 and len(ChannelColor)==2):
            frame = r+g
        if ((ChannelColor[0])==0 and (ChannelColor[1])==2 and len(ChannelColor)==2):
            frame = r+b
        if ((ChannelColor[0])==1 and (ChannelColor[1])==2 and len(ChannelColor)==2):
            frame = g+b
    if (len(ChannelColor)==3):
        frame = cv2.merge((r,g,b))
```

Código 8.2.5.5. Cambio del canal de color.

Y las modificaciones en brillo y contraste se hacen siguiendo la siguiente fórmula matemática que las relaciona:

```
# Modifica el contraste y el brillo en el frame
frame = frame * (ContrastScale.get()/127 + 1) - ContrastScale.get() + BrightScale.get()
frame = np.clip(frame,0,255)
frame = np.uint8(frame)
```

Código 8.2.5.6. Cambio de brillo y contraste.

Por último, la creación de la perspectiva cónica con varios focos y aristas variables es distinta según la cantidad de focos, por lo que dependiendo de ello se dibuja de una forma u otra:

```
FocusValue = FocoCheck.curselection()
if (len(FocusValue)>=1):
    if ((FocusValue[0])==0 and len(FocusValue)==1):
        if(persp):
            cv2.line(frame,(0,HorizontScale.get()),(500,HorizontScale.get()),(255, 255, 255),thickness=1)
            cv2.circle(frame, (Foco1Scale.get(), HorizontScale.get()), 4, (255, 255, 255), -1)
            for x in range(int(EntryAristas.get())):
                startpoint=((500/int(EntryAristas.get())+1)*x+1)+250
                cv2.line(frame,(int(startpoint),370),(Foco1Scale.get(),HorizontScale.get()),(255, 255, 255),thickness=1)
                startpoint2=(250-(500/int(EntryAristas.get())+1)*x+1)
                cv2.line(frame,(int(startpoint2),370),(Foco1Scale.get(),HorizontScale.get()),(255, 255, 255),thickness=1)
                horizontalpoint=(HorizontScale.get()+(500/int(EntryAristas.get())+1)*x*(x/20))
                cv2.line(frame,(0,int(horizontalpoint)),(500,int(horizontalpoint)),(255, 255, 255),thickness=1)
    if ((FocusValue[0])==1 and len(FocusValue)==1):
        if(persp):
            cv2.line(frame,(0,HorizontScale.get()),(500,HorizontScale.get()),(255, 255, 255),thickness=1)
            cv2.circle(frame, (Foco1Scale.get(), HorizontScale.get()), 4, (255, 255, 255), -1)
            cv2.circle(frame, (Foco2Scale.get(), HorizontScale.get()), 4, (255, 255, 255), -1)
            for x in range(int(EntryAristas.get())):
                startpoint=((500/int(EntryAristas.get())+1)*x+1)+250
                cv2.line(frame,(int(startpoint),370),(Foco1Scale.get(),HorizontScale.get()),(255, 255, 255),thickness=1)
                startpoint2=(250-(500/int(EntryAristas.get())+1)*x+1)
                cv2.line(frame,(int(startpoint2),370),(Foco1Scale.get(),HorizontScale.get()),(255, 255, 255),thickness=1)
                for x in range(int(EntryAristas.get())):
                    startpoint=((500/int(EntryAristas.get())+1)*x+1)+250
                    cv2.line(frame,(int(startpoint),370),(Foco2Scale.get(),HorizontScale.get()),(255, 255, 255),thickness=1)
                    startpoint2=(250-(500/int(EntryAristas.get())+1)*x+1)
                    cv2.line(frame,(int(startpoint2),370),(Foco2Scale.get(),HorizontScale.get()),(255, 255, 255),thickness=1)
```

Código 8.2.5.7. Dibujo de las aristas y focos para la perspectiva.

Capítulo 9

Evaluación

Con tal de comprobar el correcto funcionamiento del programa desarrollado se han evaluado sus funciones principales. Así pues, se ha evaluado la capacidad del programa para detectar personas y su correcto seguimiento, la eficacia de la detección facial, el funcionamiento de los diferentes tipos de zonas y algunas funciones más.

9.1. Evaluación de la detección y seguimiento

El primer experimento que se realizó fue para comprobar la efectividad en la detección y el seguimiento de personas. Este experimento se realizó de la siguiente forma:

- Se obtuvieron tres videos de vigilancia de lugares distintos, con iluminación diferente y flujo de personas distinto.
- Se hicieron pasar por el programa para que realizase las detecciones y el seguimiento.
- Se anotaron las detecciones y seguimientos correctos, cuando solamente se detectó a la persona y no se realizó un seguimiento y cuando no se realizó la detección a pesar de haber una persona
- Se muestran los resultados en la figura 9.1.1.

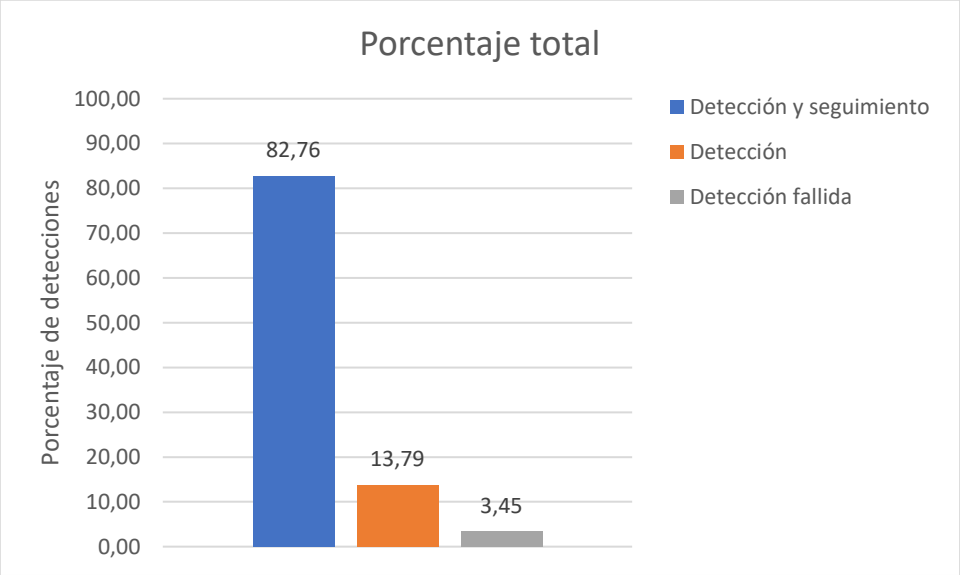


Figura 9.1.1. Resultados del experimento de seguimiento y detección.

La cantidad total de personas en el conjunto de los tres videos fue de 29 personas. De estas 29 personas 24 fueron detectadas y seguidas correctamente, 4 no fueron seguidas de forma correcta y 1 no fue detectada.

De este experimento se extrae también que el programa no está preparado para cámaras de visión nocturna, que es donde hubo una detección fallida. También que cuando hay varias personas que se solapan completamente se producen perdidas momentáneas de una detección y por lo tanto del seguimiento, pero en solapamientos parciales el programa es capaz de seguir a diferentes personas.

Por último, se intentó experimentar en un video de calidad 4K, pero el ordenador no fue capaz de procesar suficientemente rápido las detecciones.

9.2. Evaluación de la detección y reconocimiento facial

La evaluación del reconocimiento facial se dividió en dos experimentos. El primero se realizó para comprobar la capacidad de reconocer usuarios de la base de datos.

- Se añadieron 5 imágenes del usuario a la base de datos.
- El usuario se coloca delante de la cámara en diferentes orientaciones, distancias y posiciones.
- Si el reconocimiento facial detecta a esa persona como la añadida en la base de datos se considera un reconocimiento correcto, si no se identifica como incorrecto.
- Se realiza el mismo experimento, pero con una sola imagen y comparan los datos.
- Los resultados se muestran en la figura 9.2.1.

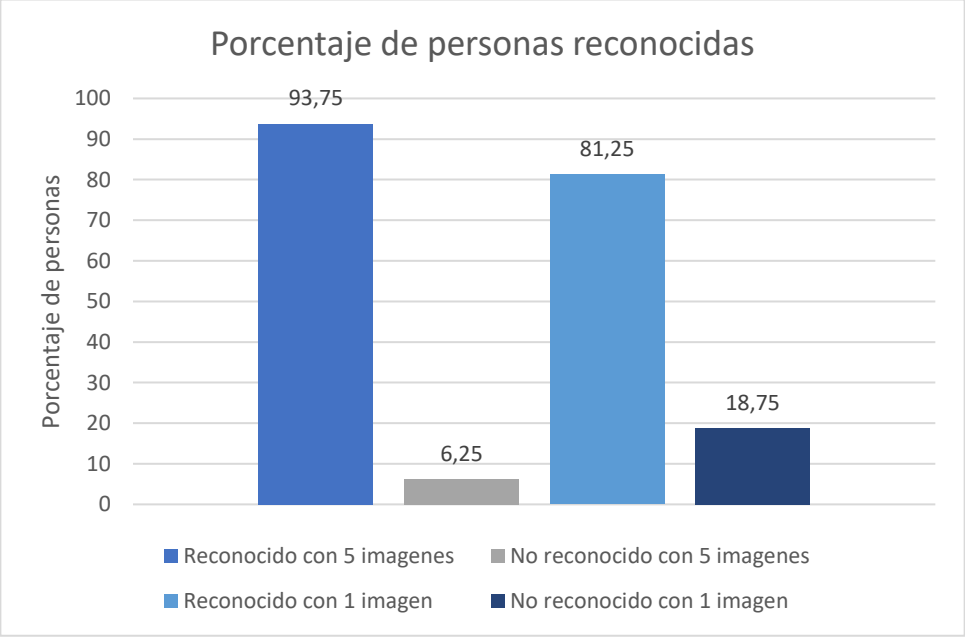


Figura 9.2.1. Resultados del experimento de reconocimiento facial.

El segundo experimento tuvo la finalidad de comprobar la capacidad del programa para identificar como no usuarios a personas no incluidas en la base de datos.

- Varias personas no identificadas en la base de datos se sitúan delante de la cámara en diferentes orientaciones, distancias y posiciones.
- Si el reconocimiento facial detecta a esa persona como una añadida en la base de datos se considera un falso positivo, si no se le reconoce se considera como correcto.
- Se realiza el mismo experimento, pero diferentes valores de tolerancia. En este caso 0.08 y 0.06.
- Se comparan los resultados en la figura 9.2.2.

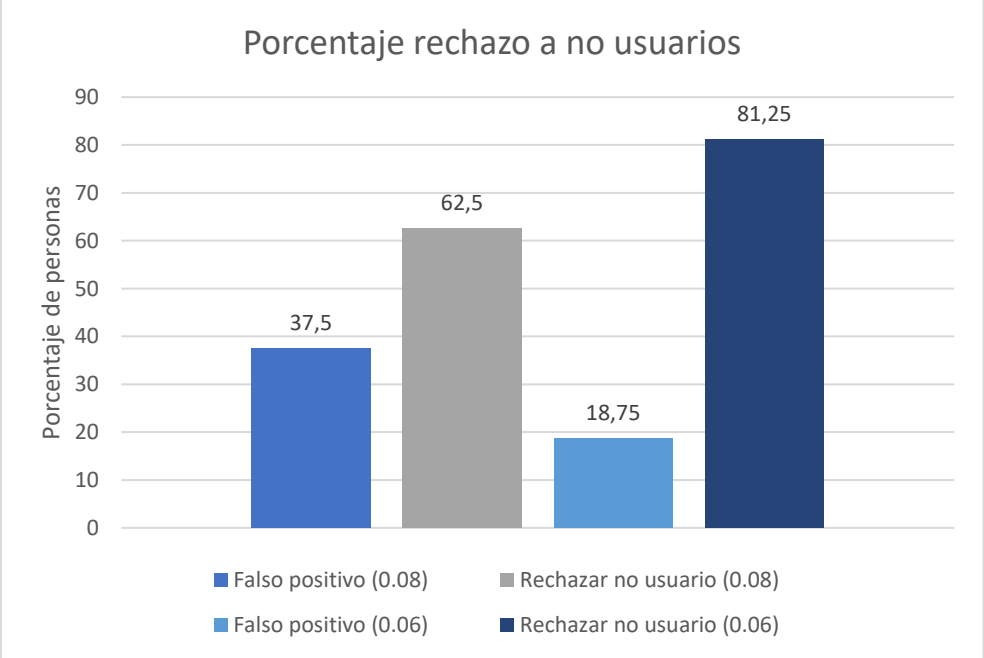


Figura 9.2.2. Resultados del experimento de reconocimiento facial.

Para estos dos experimentos se utilizaron a 16 personas de diferentes géneros, edades y características físicas.

Del primer experimento concluimos que el reconocimiento es más efectivo utilizando un mayor número de imágenes del usuario, pero aun así la efectividad utilizando one shot learning es alta. Y en el segundo experimento se observa como con una tolerancia menor, el reconocimiento produce un menor número de falsos positivos.

El reconocimiento facial de usuarios con distintas tolerancias tuvo unos resultados idénticos, por lo que no se ha incluido está estadística.

9.3. Evaluación de las diferentes zonas

Para la evaluación del funcionamiento de las distintas zonas se ha seguido el siguiente procedimiento con cada una de ellas.

- Se crea la zona del tipo a evaluar en una escena en un lugar amplio.
- Se comprueba que cuando una persona es detectada en su interior se avisa mediante el programa de forma correcta.
- En el caso de la zona de trabajo se comprueba que varían los avisos dependiendo de la cantidad de gente dentro de la zona y dura el tiempo seleccionado.

Estas pruebas fueron satisfactorias, por lo que se determinó que los distintos tipos de zona funcionan de forma adecuada.

9.4. Evaluación de otras utilidades

También se comprobó el correcto funcionamiento de otras utilidades del programa. Entre ellas cabe mencionar la utilidad de cambiar el objetivo de reconocimiento a perros o gatos. Se hizo una prueba con tres perros y los tres fueron detectados. Como ejemplo se puede observar la figura 8.4.1.

Figura 9.4.1. Resultados del reconocimiento de un perro.

Capítulo 10

Planificación

Antes del desarrollo de este proyecto se realizó una planificación inicial del tiempo a utilizar en el proyecto, teniendo en cuenta los objetivos iniciales. Sin embargo, en la planificación final se han tenido en cuenta distintos problemas encontrados a lo largo del diseño del proyecto y otros retrasos.

10.1. Planificación inicial

- Definición del proyecto y reuniones. Se realizarían 5 reuniones con el profesor tutor y el cotutor, las primeras para decidir el tipo de aplicación y las siguientes reuniones para revisar el progreso y añadir diferentes detalles.
  Tiempo estimado: 10 horas.

- Estudio del lenguaje Python. Estas horas se dedicarían a aprender a manejar mejor este lenguaje y a crear aplicaciones con él.
  Tiempo estimado: 30 horas.

- Investigación sobre la detección, el seguimiento y el reconocimiento facial. En esta fase se investigaría sobre diferentes métodos para conseguir estas funciones en Python.
  Tiempo estimado: 60 horas.

- Diseño de la aplicación. Tras decidir los métodos a utilizar diseñar el funcionamiento concreto del programa.
  Tiempo estimado: 20 horas.

- Desarrollo de la aplicación. Tras tener claro el diseño, implementarlo en el código y hacer que el programa funcionase.
  Tiempo estimado: 40 horas.

- Experimentación de la aplicación. Una vez creada, realizar diferentes pruebas para comprobar el correcto funcionamiento del proyecto.
  Tiempo estimado: 20 horas.

- Documentación. Tiempo destinado a la redacción de la memoria, planificación, requerimientos, etc.
  Tiempo estimado: 60 horas.

- Presentación. Este tiempo sería designado a diseñar la presentación del TFG y su explicación frente al jurado.
  Tiempo estimado: 10 horas.

La suma total de las horas en la planificación inicial es de 250 horas.

10.1. Planificación final

A continuación, se detallan las horas que se utilizaron finalmente para cada uno de los apartados anteriores.

- Definición del proyecto y reuniones. Durante esta fase se realizó una reunión extra más debido a algunos problemas por lo que se utilizaron algunas horas más de las esperadas.
  Tiempo estimado: 10 horas.
  Tiempo empleado: 15 horas.

- Estudio del lenguaje Python. En el estudio del lenguaje Python se utilizaron más horas de las previstas por el uso de más librerías.
  Tiempo estimado: 30 horas.
  Tiempo empleado: 40 horas.

- Investigación sobre la detección, el seguimiento y el reconocimiento facial. También en este apartado se utilizaron más horas de las esperadas.
  Tiempo estimado: 60 horas.
  Tiempo empleado: 70 horas.

- Diseño de la aplicación. En esta sección se utilizaron las horas estimadas.
  Tiempo estimado: 20 horas.
  Tiempo empleado: 20 horas.

- Desarrollo de la aplicación. Para el desarrollo de la aplicación tampoco fueron necesarias más horas de las preestablecidas.
  Tiempo estimado: 40 horas.
  Tiempo empleado: 40 horas.

- Experimentación de la aplicación. En este caso también se utilizaron exclusivamente las horas estimadas.
  Tiempo estimado: 20 horas.
  Tiempo empleado: 20 horas.

- Documentación. Para la documentación fueron necesarias más horas de las esperadas, en concreto por la redacción de la memoria.
  Tiempo estimado: 60 horas.
  Tiempo empleado: 65 horas.

- Presentación. En este caso no fueron necesarias más horas de las estimadas.
  Tiempo estimado: 10 horas.
  Tiempo empleado: 10 horas.

Por lo tanto, se utilizaron 260 horas para completar el proyecto.

Capítulo 11

Presupuesto

Para el proyecto se han realizado dos presupuestos, un presupuesto inicial teniendo en cuenta las horas y el trabajo estimado y otro presupuesto final teniendo en cuenta las horas utilizadas realmente para el proyecto.

11.1. Presupuesto inicial

Como para este proyecto todo el software que se ha utilizado era software libre, tanto las librerías, el lenguaje de programación como los editores de código solamente ha habido coste de recursos humanos.

En este trabajo se ha considerado un ingeniero en prácticas para el desarrollador del trabajo, con un salario de 20€/h que es el salario medio de este tipo de profesionales en España. Por otra parte, al trabajo hecho por el tutor y cotutor se ha considerado de ingeniero senior, cuyo trabajo tiene un coste de 50€/h.

Al no haber más gastos en la siguiente tabla se muestra el presupuesto inicial completo.

	Precio (€) /Tiempo (h)	Tiempo (h)	Subtotal (€)
Ingeniero en prácticas	20	240	4800
Ingeniero sénior	50	10	500
		Total (€)	5300

Tabla 10.1. Tabla de presupuesto inicial.

11.2. Presupuesto final

En el presupuesto final se tienen en cuenta las horas realizadas tras la ejecución del proyecto. En este caso el resultado ha sido el siguiente:

	Precio (€) /Tiempo (h)	Tiempo (h)	Subtotal (€)
Ingeniero en prácticas	20	245	4900
Ingeniero sénior	50	15	750
		Total (€)	5650

Tabla 10.2. Tabla de presupuesto final.

En conclusión, la diferencia entre el presupuesto inicialmente planteado y el final asciende a 350€, lo cual es una cantidad aceptable de variación en la ejecución de un proyecto de este tipo. Así mismo, el precio final del proyecto para tratarse de un trabajo de ingeniería con tecnología actual e investigación se trata de un precio aceptable.

Capítulo 12

Conclusiones

Los objetivos que debía cumplir el software como programa de monitorización y vigilancia han sido alcanzados. El seguimiento y detección de personas funcionó correctamente en más de un 80% de los casos. La detección de personas, a pesar de ser una función más compleja también obtuvo buenos resultados en la experimentación.

El resto de funciones también lograron aplicarse correctamente, entre ellas cabe destacar la creación de las diferentes zonas para un control más preciso de la escena, el modo de vigilancia avanzado dónde se envía una imagen del intruso detectado en la escena por correo, y el control del aforo máximo.

Por lo tanto, podemos concluir que el desarrollo y los resultados de los experimentos realizados con el programa han sido satisfactorios.

Capítulo 13

Trabajos futuros

Para un futuro mayor desarrollo del proyecto se han considerado diversas mejoras. En cuanto al seguimiento de personas, se podría utilizar un algoritmo de un filtro de Kalman para mejorar la precisión del seguimiento, pero sería necesario un procesador más potente o quizás utilizar una GPU. Del mismo modo podría solucionarse que el programa sea capaz de procesar videos de mayor calidad o con un mayor número de personas a la vez en la escena.

Otra opción para ayudar a mejorar las detecciones sería añadir más opciones para modificar con otros filtros la escena, ya sea para obtener detalles más precisos o una mayor adaptabilidad a la luz del escenario en el que se utilice la cámara.

También para un mayor control de la escena, continuar el desarrollo de añadir una perspectiva que permita conocer una posición aproximada de la persona detectada en coordenadas reales y no solo de coordenadas en pixeles en la pantalla podría ser muy útil.

Capítulo 14

Anexos

14.1. Manual de usuario

En el siguiente manual de usuario se describirá de forma detallada como utilizar las funciones del programa desarrollado en este proyecto. El programa está formado por tres pestañas, cada una con funciones diferentes.

3. Pestaña estándar
4. Pestaña avanzado
5. Pestaña operador

En todas estas pestañas se realiza la detección y seguimiento de cualquier persona que entre en la escena de forma automática, estas personas serán numeradas con una identificación distinta por cada nueva detección.

1. Pestaña estándar

En esta pestaña se encuentran las funciones básicas del programa. Entre estas funciones se encuentra la creación de zonas prohibidas, de zonas de interacción, el control del aforo máximo, el modo de vigilancia, la creación de zonas de trabajo, la opción de cambiar de objetivo de detección y la detección y seguimiento de objetivos en la escena, así como el recuento de personas actuales en la escena y el recuento total de personas que han aparecido.

El formato en el que se presenta es el siguiente:

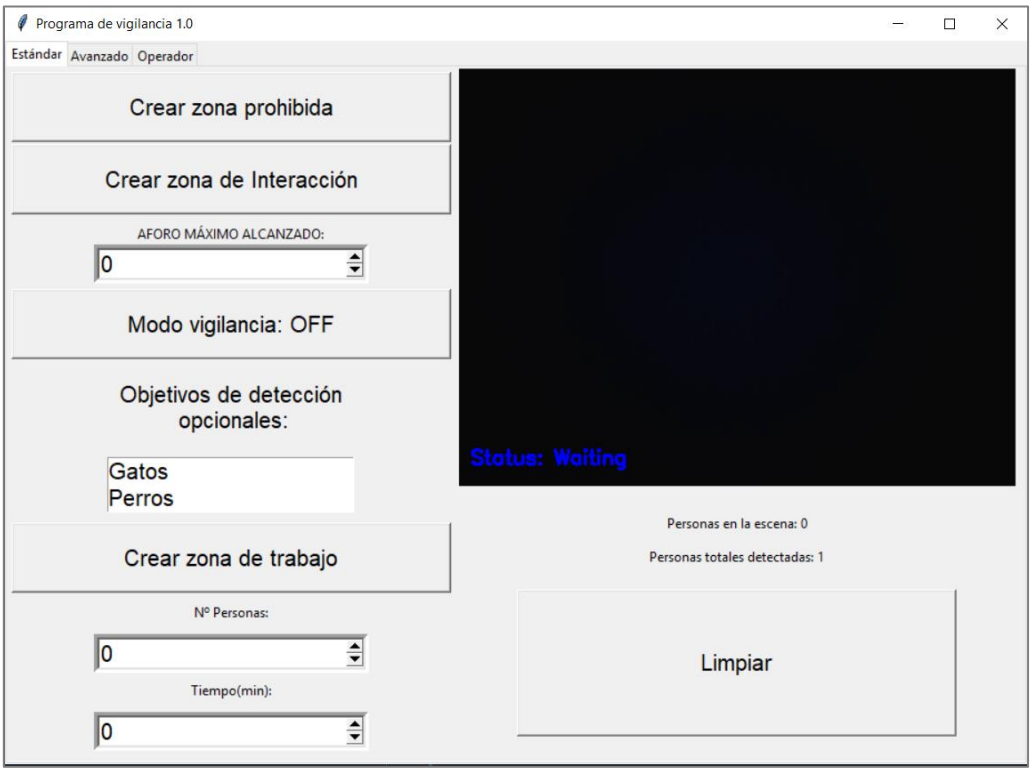


Figura 14.1.1.1. Diseño de la interfaz de usuario en la pestaña estándar.

Para **crear una zona prohibida** primero se debe hacer clic en el botón de "Crear zona prohibida". A continuación, se realiza un clic sobre la pantalla para poner una de las esquinas del rectángulo que queremos que forme dicha zona, y después realizamos un segundo clic donde queremos la esquina opuesta como se puede ver en la figura 14.1.1.2. y 14.1.1.3.

Si se ha creado una zona prohibida correctamente el rectángulo debería ser de color blanco como se puede observar en la figura 14.1.1.3. En caso de que se detecte una persona en el interior de esta zona se activará un aviso como se puede observar en la figura 14.1.1.4. Finalmente, si se desea eliminar esta zona solo se debe pulsar el botón de "Limpiar" y todas las zonas en la pantalla se eliminarán.

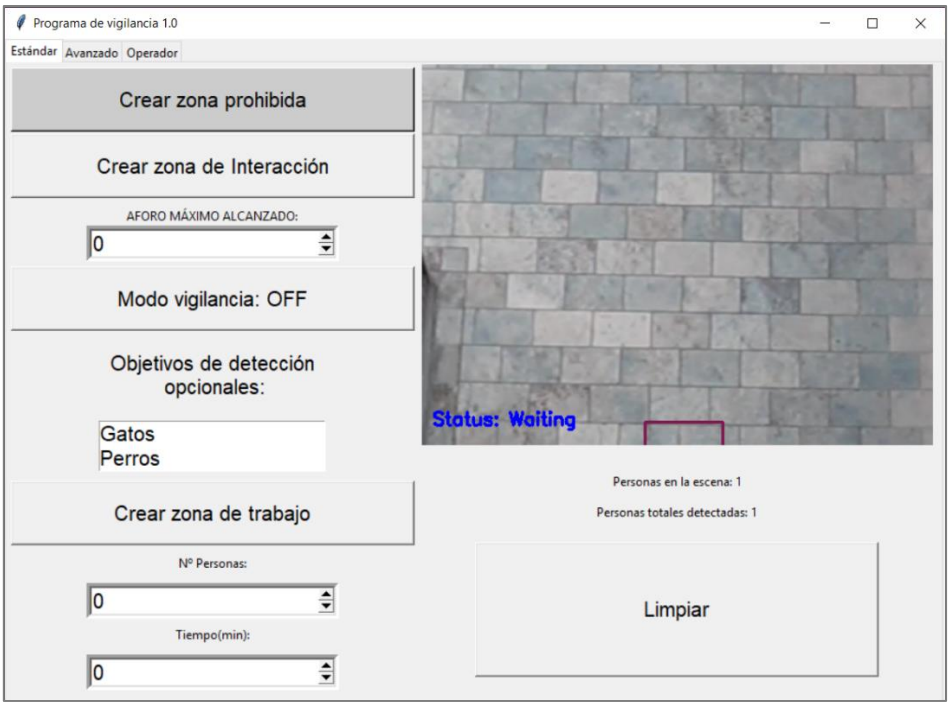


Figura 14.1.1.2. Botón de crear zona prohibida.

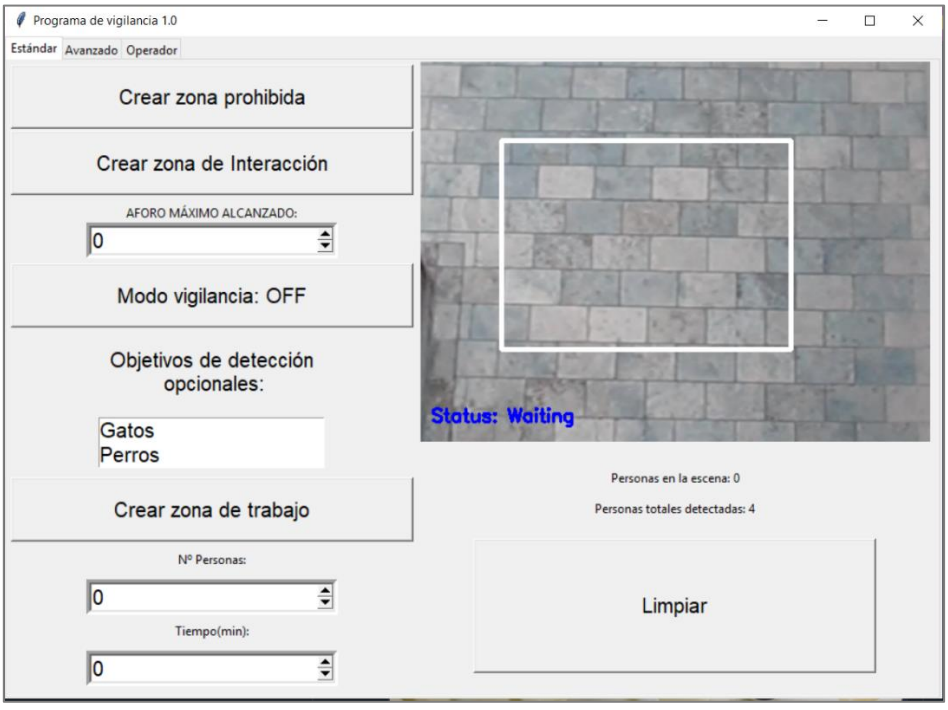


Figura 14.1.1.3. Zona prohibida creada.

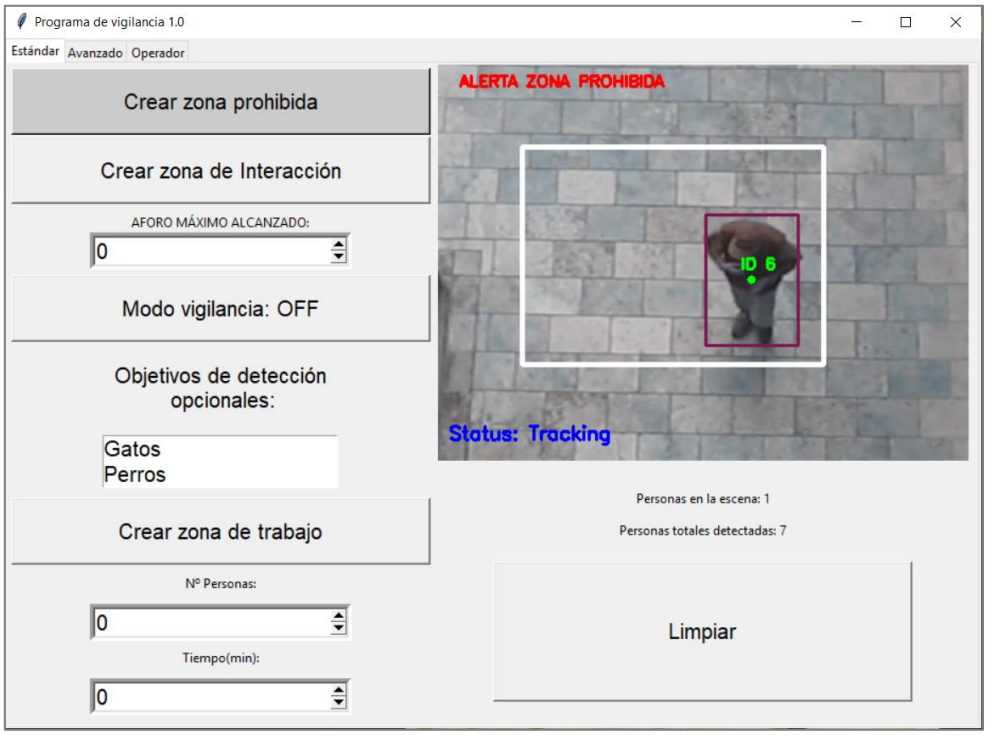


Figura 14.1.1.4. Zona prohibida en funcionamiento

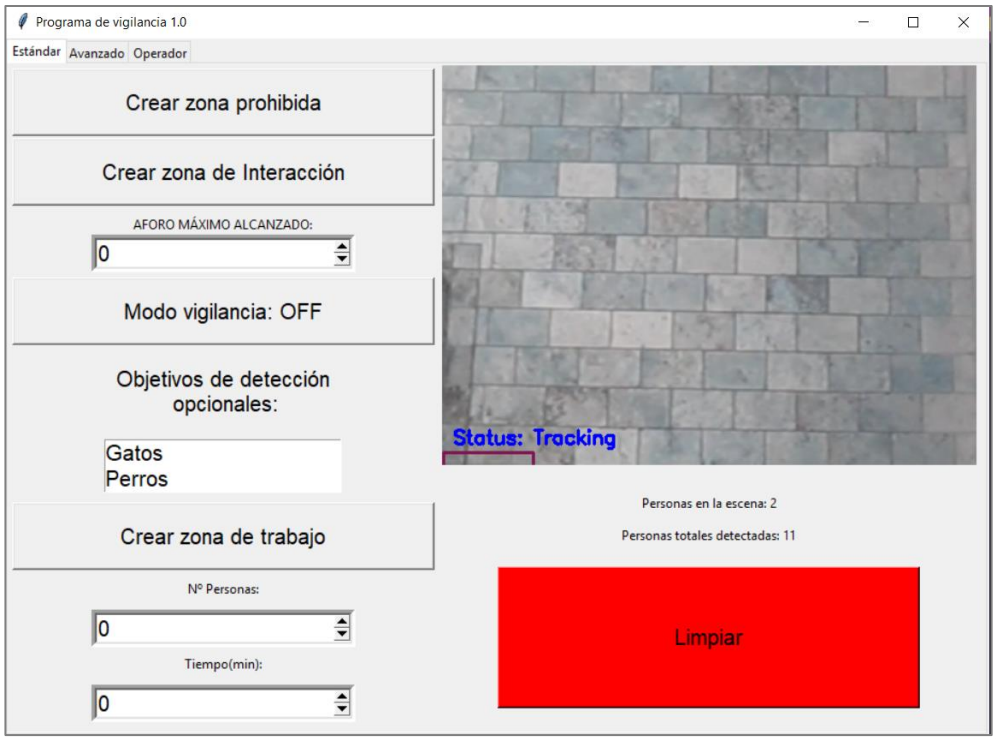


Figura 14.1.1.5. Limpieza de zonas.

La creación del resto de zonas se hace del mismo modo. En el caso de la **zona de interacción**, la zona creada será de color azul, el aviso que genera al activarse puede verse en la figura 14.1.1.6.

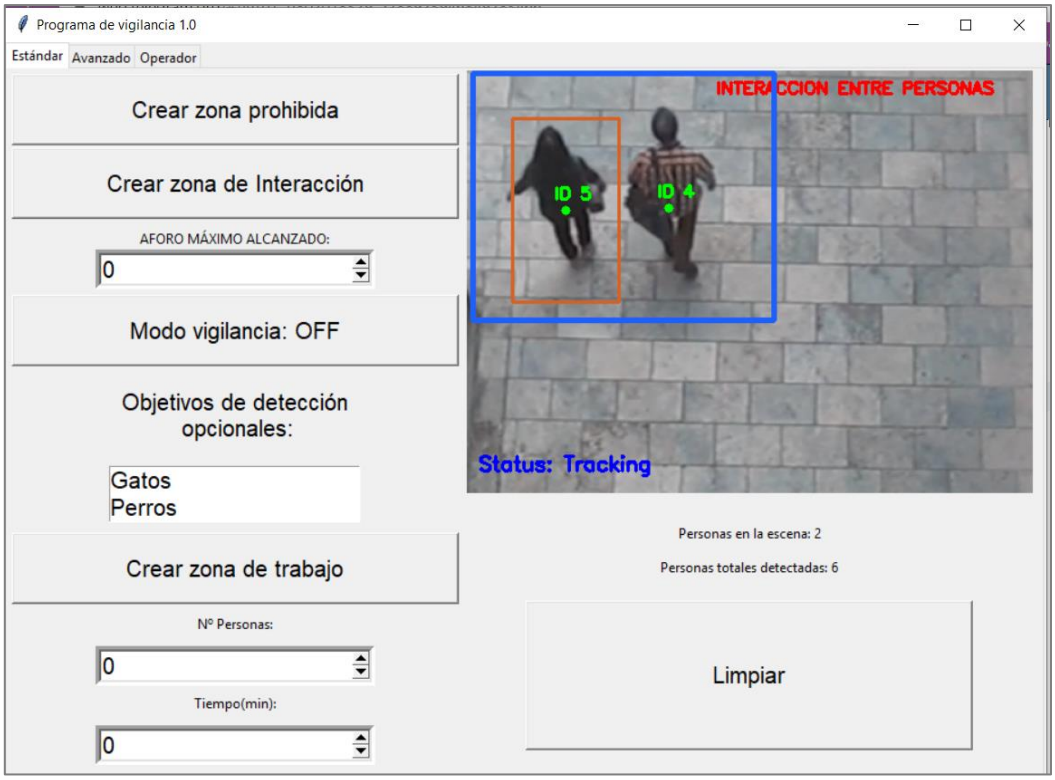


Figura 14.1.1.6. Creación de zona de interacción.

Si se desea **cambiar el objetivo de detección** a perros y/o gatos solo es necesario realizar un clic sobre la opción que se desee, como se observa en la figura 14.1.1.7.

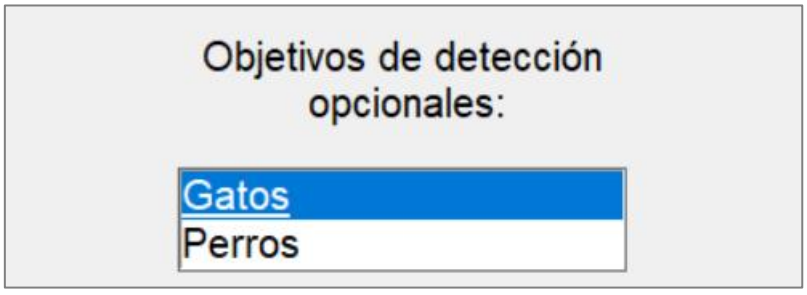


Figura 14.1.1.7. Objetivos de detección opcionales

Para designar el **aforo máximo** es necesario modificar el valor utilizando las flechas o cambiando el número manualmente. En caso de que se alcance se alerta como se puede observar en la figura 14.1.1.8.

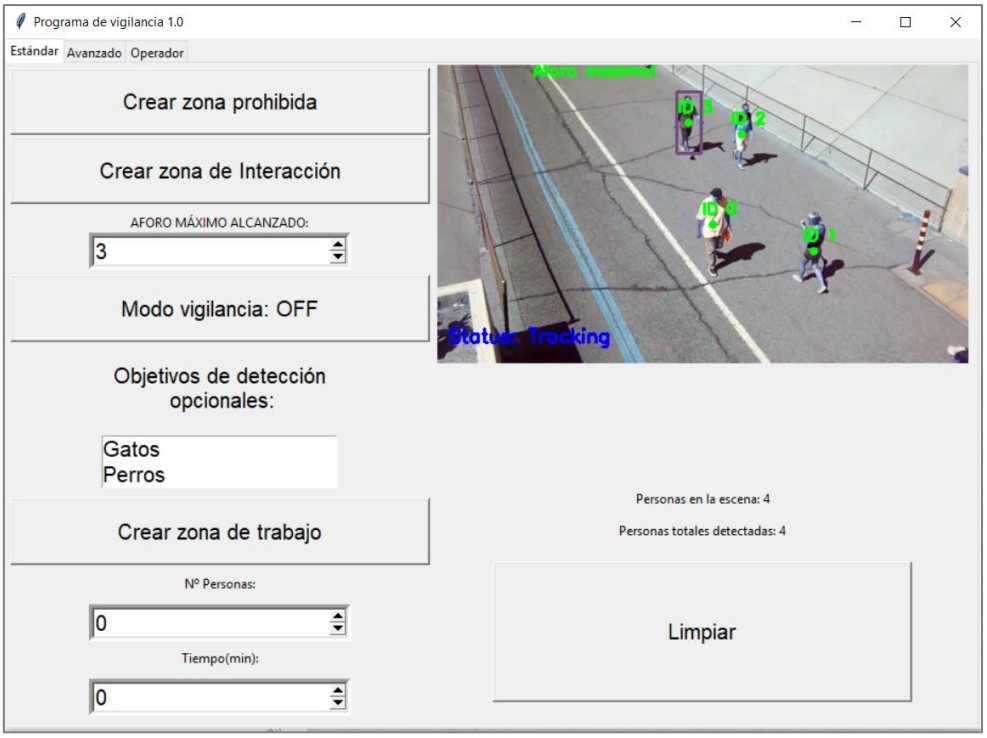


Figura 14.1.1.8. Alerta de aforo máximo.

Para activar el **modo vigilancia** estándar se ha de realizar clic en el botón modo vigilancia. Una vez activado en el mismo botón se nos mostrará como activado diciendo "ON" en vez de "OFF". Se puede observar su funcionamiento en la figura 14.1.1.9.

Por último, para crear una **zona de trabajo** primero debemos señalar el número de personas que queremos que estén en esta y el tiempo que queremos que dure en las dos casillas debajo del botón de crear zona de trabajo. En este caso la zona creada será de color negro. Esto se muestra en la figura 14.1.1.10.

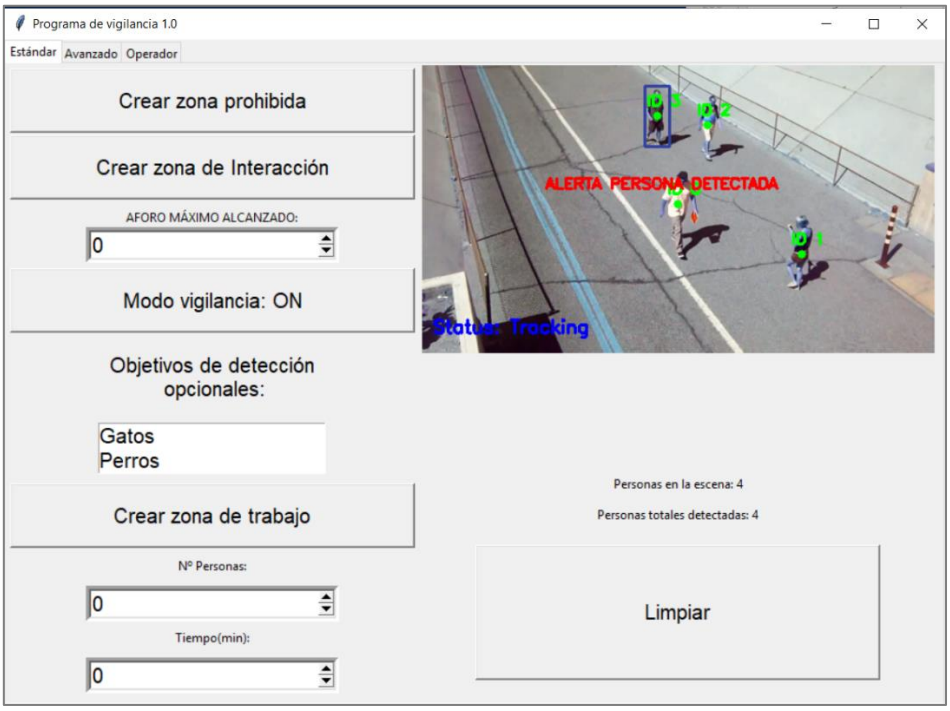


Figura 14.1.1.9. Funcionamiento de modo vigilancia.

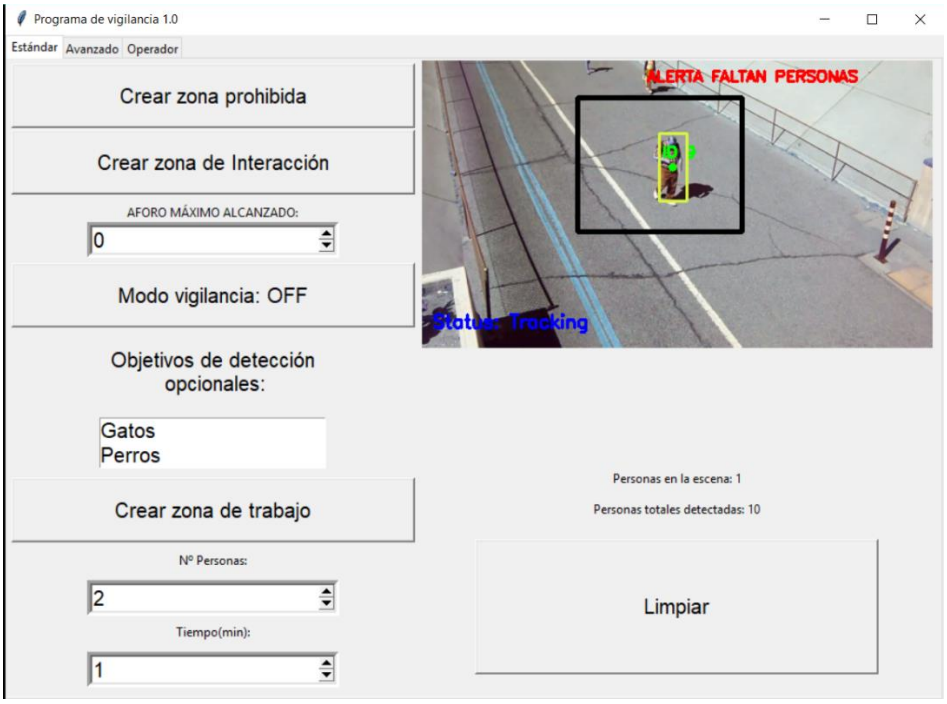


Figura 14.1.1.10. Funcionamiento de la zona de trabajo.

2. Pestaña avanzado

Dentro de esta pestaña se encuentran las funciones que utilizan reconocimiento facial y otras opciones más avanzadas. Entre ellas encontramos la función de añadir personas a la base de datos, la creación de zonas exclusivas a usuarios en la base de datos, el modo de vigilancia avanzado y el reconocimiento facial. La interfaz de esta pestaña se puede ver en la figura 14.1.2.1.

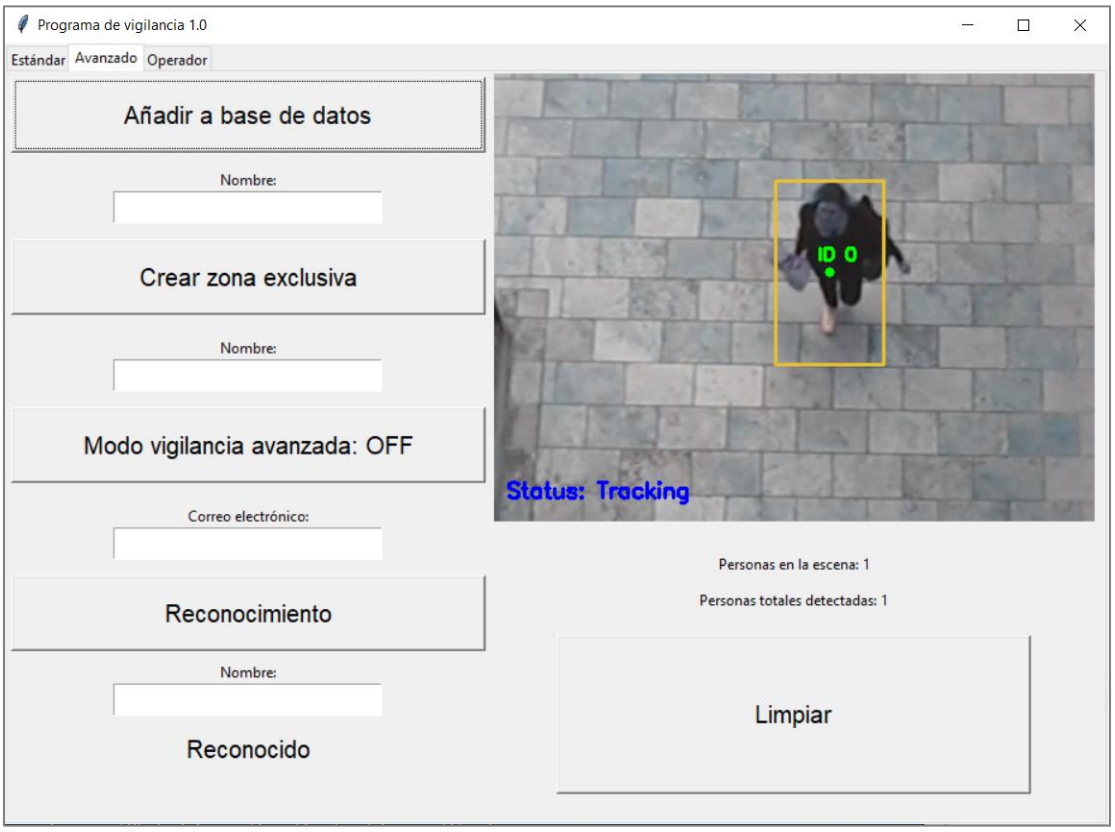


Figura 14.1.2.1. Pestaña avanzado.

. Para **añadir una persona a la base de datos** primero es necesario escribir su nombre en la casilla de "Nombre" justo debajo de este botón, una vez escrito la persona a añadir deberá acercarse a la cámara hasta que esta sea capaz de detectar su cara. Entonces se pulsará el botón de añadir a base de datos y el programa tomará una foto de esta persona y la guardará con el nombre asignado a la base de datos. Podemos observar este proceso en la figura 14.1.2.2.

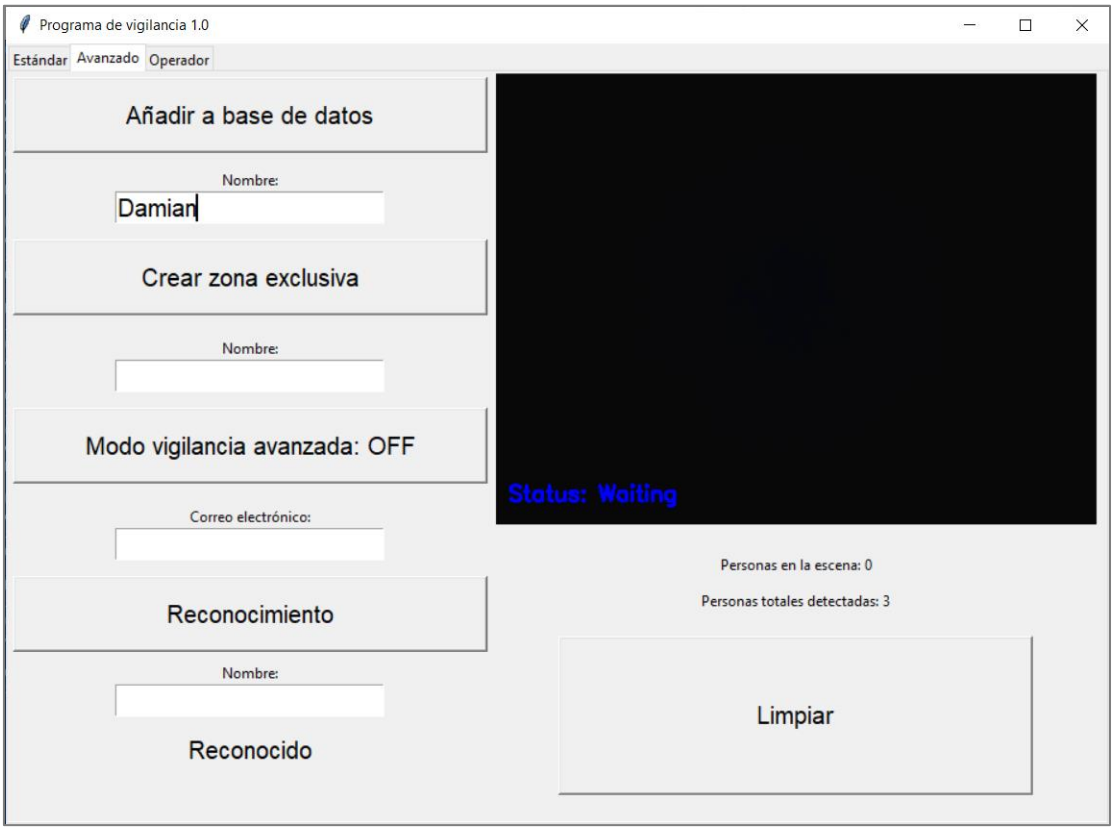


Figura 14.1.2.2. Ejemplo de añadir persona a la base de datos.

. **Crear una zona exclusiva** se hace de la misma forma que el resto de zonas, pero antes hay que introducir el nombre de la persona apta para entrar en esta zona en la casilla "Nombre" debajo del botón "Crear zona exclusiva". Esta zona es de color azul oscuro. Se puede observar el funcionamiento de esta zona en la figura 14.1.2.3.

El modo de vigilancia avanzada funciona del mismo modo que el modo de vigilancia. Al activarlo el nombre del botón cambiará a "ON" en lugar de "OFF". Cuando una persona es detectada en este caso, se obtendrá una imagen de la cara del intruso y se enviará al correo electrónico que se deberá haber introducido previamente en la casilla inferior al botón "Correo electrónico". El funcionamiento de este botón se puede ver en la figura 14.1.2.4. y el correo que envía el programa se puede observar en la figura 14.1.2.5.

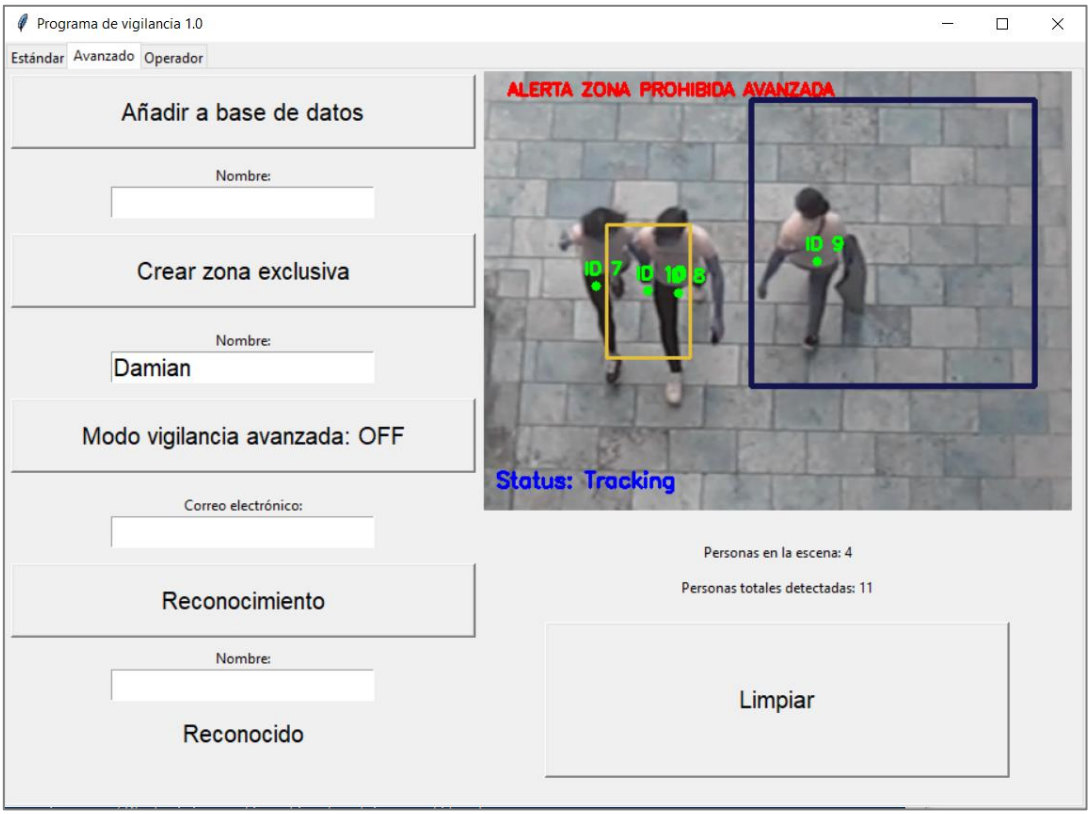


Figura 14.1.2.3. Funcionamiento de la zona exclusiva.

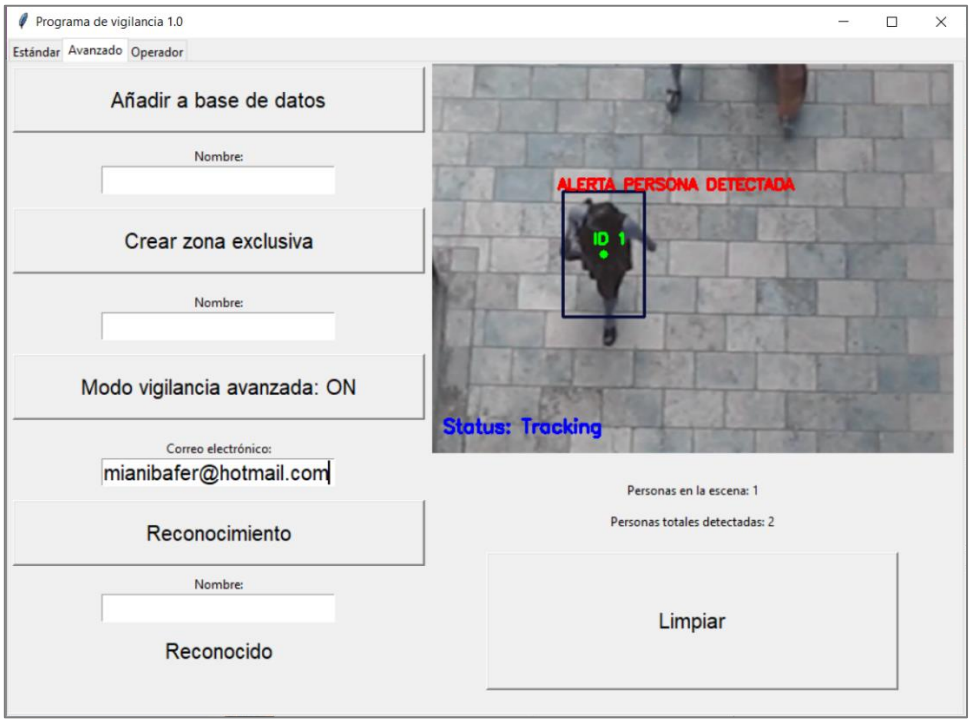


Figura 14.1.2.4. Funcionamiento del modo de vigilancia avanzado.

Figura 14.1.2.5. Correo enviado por el modo de vigilancia avanzada.

Por último, tenemos el **reconocimiento** facial. Para activar esta función se ha de escribir el nombre del individuo a reconocer en la casilla de "Nombre" debajo del botón de reconocimiento. Una vez hecho esto se pulsa el botón de reconocimiento, y en caso de que la persona detectada sea reconocida el texto debajo del nombre que hayamos introducido dirá "Reconocido". En caso contrario el texto dirá "No reconocido.".

Además, una vez activado el reconocimiento se puede modificar el nombre de la persona que se desea identificar en cualquier momento. Las personas identificadas en la imagen aparecerán con el nombre detectado y la similitud con la base de datos escritos encima del recuadro que rodeará su cara, en caso contrario aparecerá el texto "No reconocido". Se puede observar un ejemplo en la figura 14.1.2.6.

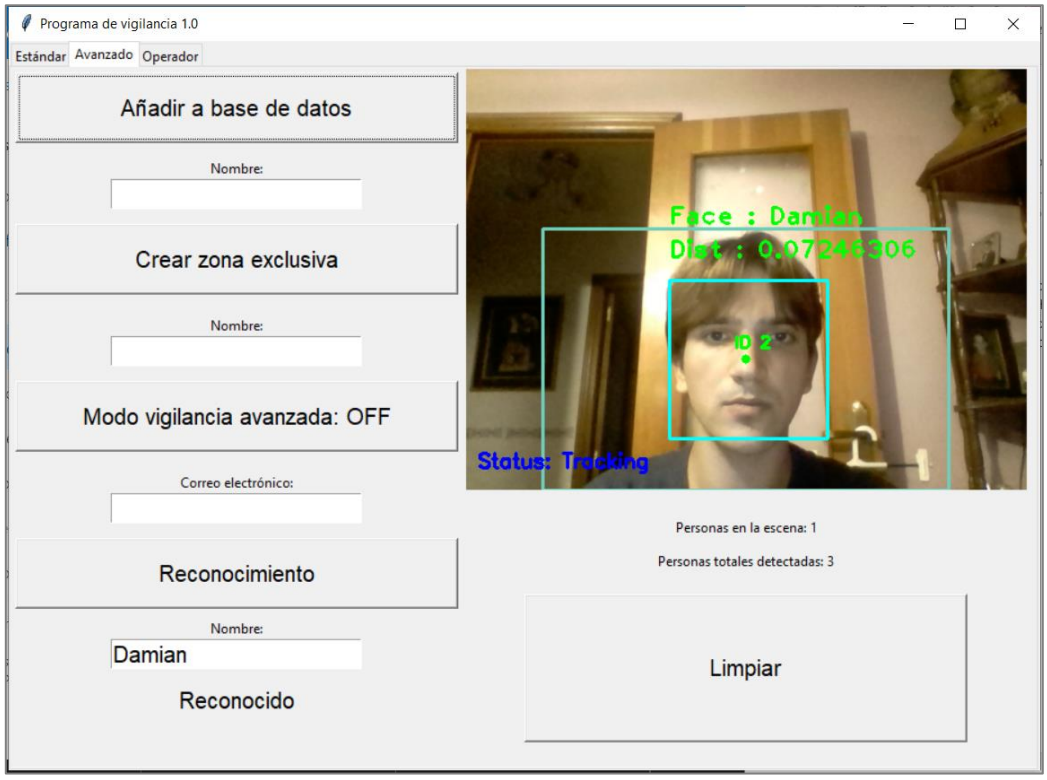


Figura 14.1.2.6. Funcionamiento del reconocimiento facial.

3. Pestaña operador

En esta pestaña se pretenden incluir varias herramientas para que sea más fácil la detección y el seguimiento de personas a través de varias herramientas. Las herramientas actualmente disponibles son la modificación de la imagen a

blanco y negro, el cambio de brillo y contraste, el cambio de canal de color y la posibilidad de añadir una perspectiva.

Para modificar la imagen a **blanco y negro** se ha de pulsar botón "Blanco y negro". Una vez hecho esto el botón también habrá cambiado a llamarse "Color", pulsarlo de nuevo devolverá la configuración de color anterior.

Para cambiar el **brillo** y el **contraste** tenemos dos barras que al deslizarla se modifican estos valores, además se nos indica con que valor se están modificando. Para cambiar el **canal de color** se ha de hacer clic encima del canal de color deseado, además se pueden utilizar varios a la vez. Podemos ver varias de estas opciones en la figura 14.1.3.1.

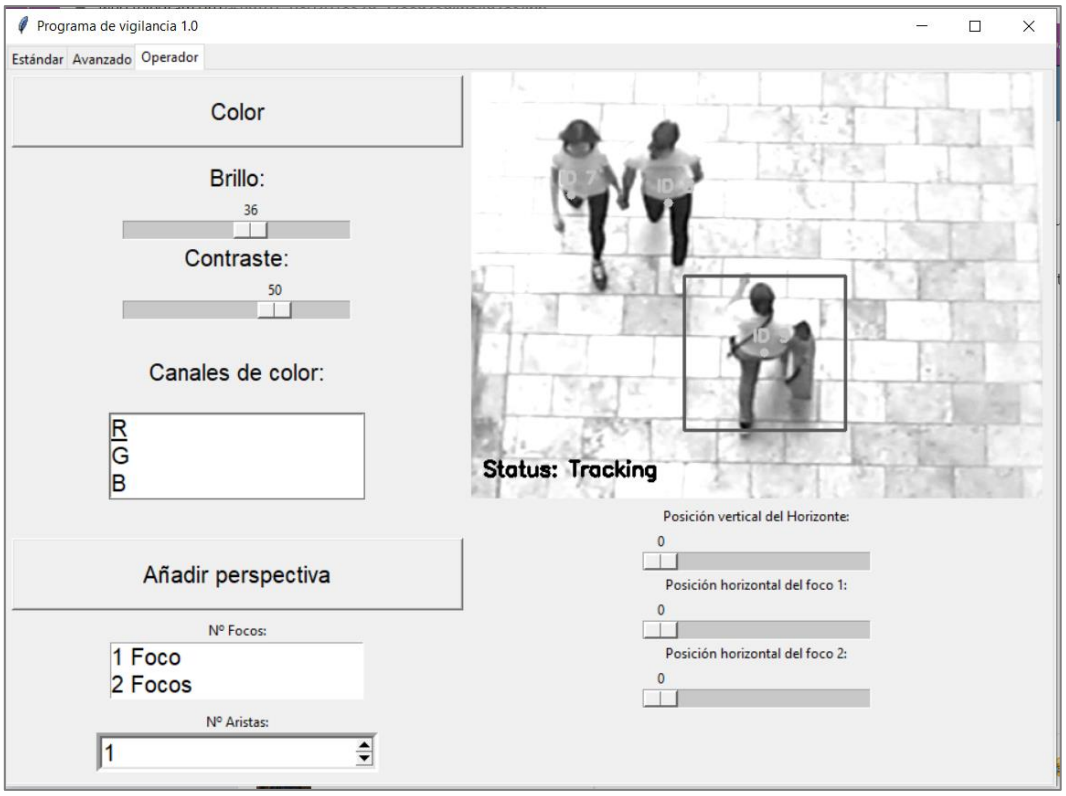


Figura 14.1.3.1. Distintas modificaciones en brillo y contraste en blanco y negro.

Por último, para **añadir una perspectiva** a la escena, primero se ha de elegir el número de focos que tendrá. Después se realiza clic sobre el botón añadir perspectiva. Una vez creada podemos modificar de nuevo si queremos uno o dos focos, la cantidad de aristas por foco para que la cuadricula se ajuste mejor a lo

deseado, la posición vertical del horizonte y la posición horizontal de ambos focos en el horizonte. Podemos ver un ejemplo de esta perspectiva en la figura 14.1.3.2.

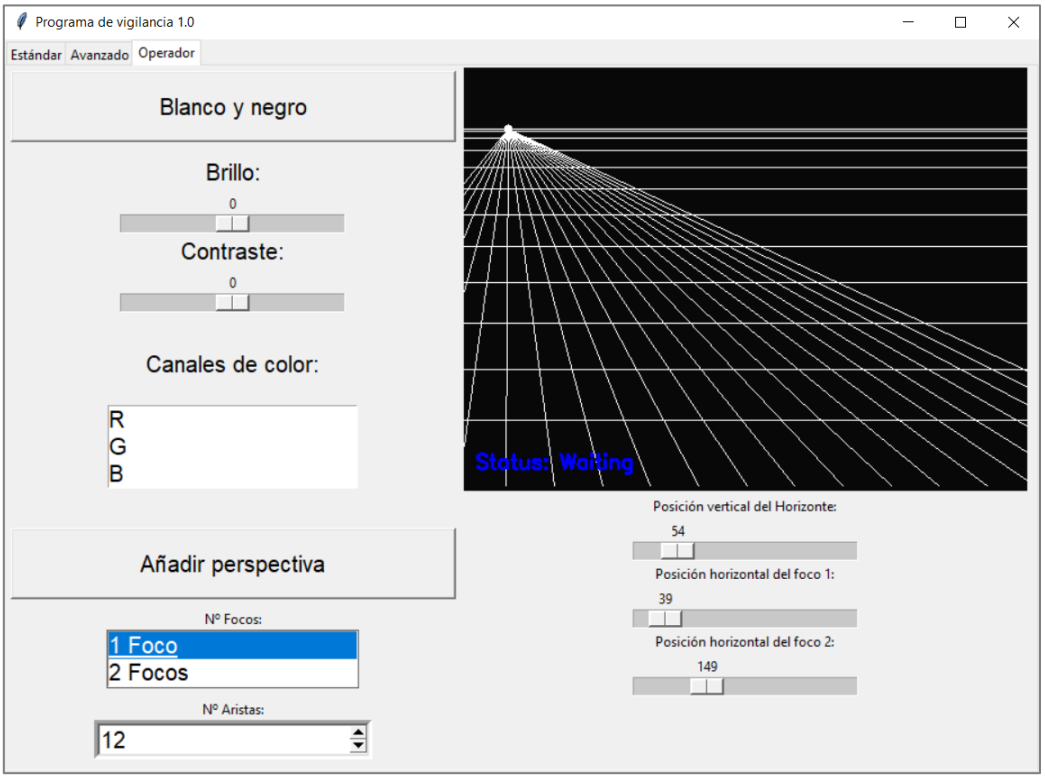


Figura 14.1.3.2 Ejemplo de perspectiva añadida a la imagen.

14.2. Bibliografía

[1] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov. (2016). Siamese Neural Networks for One-shot Image Recognition. Department of Computer Science, University of Toronto.

[2] Página informativa sobre redes neuronales. (2019): http://neuralnetworksanddeeplearning.com/

[3] Página con cursos sobre visión artificial, OpenCV y aprendizaje profundo. (2017): https://www.pyimagesearch.com

[4] Página con información sobre redes neuronales. (2017):

https://towardsdatascience.com/

[5] Página sobre el uso de redes neuronales en la visión artificial. (2019):

http://cs231n.github.io/

[6] Página con tutoriales sobre inteligencia artificial (2019):

https://becominghuman.ai/

[7] Página con información sobre visión artificial, seguimiento e inteligencia artificial (2019):

https://iitmcvg.github.io/

[8] Página con información sobre visión artificial y redes neuronales (2019):

https://computervision.tecnalia.com/en/

[9] Página con información sobre usos de la visión artificial en el mercado (2019):

https://fullscale.io/computer-vision-uses-companies/

[10] Página con información sobre visión artificial y aprendizaje automático (2019):

https://machinelearningmastery.com/ç

[11] Página con estadísticas e información sobre el mercado de la inteligencia artificial y la visión artificial (2019):

https://www.fortunebusinessinsights.com/industry-reports/artificial-intelligence-market-100114

[12] Página con estadísticas e información sobre la vigilancia por visión artificial en China (2018):

https://www.cbinsights.com/research/china-surveillance-ai/

[13] Página con información sobre distintos métodos de detección de personas en tiempo real. (2018):

https://medium.com/@madhawavidanapathirana

[14] Página con información sobre redes neuronales. (2018):

https://ml4a.github.io/

[15] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, Philip H.S. Torr. (2017). End-To-End Representation Learning for Correlation Filter Based Tracking.

[16] Página oficial con cursos y documentación sobre OpenCV. (2019):

https://opencv.org/

[18] Página oficial con cursos y documentación sobre TensorFlow. (2019):

https://www.tensorflow.org/

[19] Página oficial con cursos y documentación sobre Keras. (2019):

https://keras.io/

[20] Página con cursos y documentación sobre Tkinter y Python (2019):

https://www.tutorialspoint.com/python/

[21] Página con cursos sobre Python (2019):

https://www.codingforentrepreneurs.com/