UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

IMPLEMENTACIÓN DE UN SISTEMA SOFTWARE DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN DE OBJETOS EN MOVIMIENTO.

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Silke Fresno Moreno

TUTORIZADO POR

Patricia Balbastre Betoret

CURSO ACADÉMICO: 2019/2020

Índice

Índice de figuras.

1. Objeto del proyecto.

El objeto del presente proyecto de fin de grado consiste en el diseño e implementación de un sistema software de visión artificial para la detección de objetos en movimiento mediante un sistema de hardware estándar y un software desarrollado para este fin.

El objetivo específico del proyecto es la detección de movimiento y el posterior tratamiento de las imágenes capturadas.

Para ello se ha realizado la programación mediante la interfaz gráfica Qt y desarrollando las librerías OpenCV, destinada a la visión artificial mediante ordenador, así como el lenguaje de programación C++.

2. Estudio de necesidades, factores a considerar: limitaciones y condicionantes.

2.1. ¿Qué és la visión artificial?

La visión artificial es una tecnología punta con una clara tendencia a una mayor presencia, con mucho potencial y gran cantidad de aplicaciones, siendo esta uno de los tres campos principales de la inteligencia artificial junto al aprendizaje automático (machine learning) y la lógica difusa (fuzzy logic).

Es un sistema que permite la obtención de imágenes de forma automatizada y el análisis de esas imágenes para adquirir los datos necesarios que permitan analizar y/o controlar un proceso. Este sistema tiene como cualidad primordial el hecho de ser automático de modo que todas las decisiones que se toman son sin intervención humana, es decir se trata de un sistema autosuficiente para el propósito mencionado, siendo también unas de sus características la eliminación de la subjetividad en los procesos de inspección, y la flexibilidad de los procesos productivos, ya que pueden ser modificados para adaptarse a diversos procesos. Por lo cual el coste de estos sistemas tiende a ser más bajo que los demás, siendo a su vez fiables y robustos.

Los objetivos que la visión artificial comprende, según el libro "Machine Vision – Applications and Systems" son:

- *La detección, segmentación, localización y reconocimiento de objetos en imágenes.*
- *La evaluación de resultados.*
- *Registro de diferentes imágenes de una misma escena u objeto, es decir, hacer concordar un mismo objeto en diversas imágenes.*

- *Seguimiento de un objeto en una secuencia de imágenes.*
- *Mapeo de una escena para generar un modelo tridimensional de la escena.*
- *Estimación de las posturas tridimensionales de humanos.*
- *Búsqueda de imágenes digitales por su contenido.*

Estos objetivos se logran mediante el reconocimiento de patrones, el aprendizaje estadístico, la geometría de proyección, el procesamiento de imágenes, la teoría de grafos y otros campos.

2.2. Aplicaciones visión artificial.

Así, mediante la aplicación de diversas técnicas y según las necesidades de las industrias, la visión artificial ha ido avanzando e incorporandose en diferentes ámbitos profesionales, tan diversos como:

- Robótica y cadenas de montaje: los procesos industriales precisan la automatización para la detección de piezas en el proceso de fabricación o para la verificación de ciertos controles de calidad, evitando defectos de producción.

- Sector alimentario: la visión artificial es fundamental en el control de calidad de los alimentos, no solo para el estado de los productos sino también para asegurar el correcto envasado y cierre de los productos enlatados.

- Telefonía móvil: los smarthphones utilizan diferentes aplicaciones como la detección de rostros para desbloquear la pantalla, la estabilización de video, etc.

- Medicina: usa la visión artificial para el análisis de imágenes para detectar anomalías en resonancias o escáneres.

- Videovigilancia: es una de las aplicaciones más recurrentes en la visión artificial seguridad y la vigilancia, debido a los algoritmos de reconocimiento facial, así como el control de acceso.

2.3. Ventajas y beneficios de la visión artificial.

En todos los sectores y campos aplicados, la visión artificial ha logrado beneficios y mejoras para el posicionamiento de la industria.

Como pueden ser:

- Aumentar la producción: optimizan los tiempos de producción.
- Evita los fallos humanos: evita la subjetividad humana, la falta de atención, los errores visuales.
- Reduce costes: costes de tiempo, de personal, de procesos productivos.
- Aumenta la calidad en la producción: la producción es fiable, objetiva y constante.
- Aumenta la satisfacción del cliente: al lograr productos de mejor calidad y menor coste.

2.4. Objetivos.

Aparte del desarrollo de esta herramienta de obtención, procesamiento y análisis de información a través de imágenes, se ha planteado la necesidad de un objetivo más específico en el proyecto, se ha decido incorporar una aplicación de video vigilancia, en la cual se incluye un sistema de grabación, capaz de grabar y guardar las imágenes capturadas en video automáticamente cuando el programa detecta movimiento.

Además, el usuario debe ser capaz de visionar con posterioridad dichas grabaciones mediante una interfaz sencilla e intuitiva.

Se ha optado por esta utilización, ya que el campo donde se está utilizando mayormente la visión artificial es el de la seguridad y la vigilancia. La tarea de las cámaras de vigilancia es monitorear el área. Se conectan a un dispositivo de grabación, una dirección IP o envían la grabación a un monitor que es atendido por un guardia de seguridad.

Hasta hace poco, estos dispositivos eran costosos y requerían personal cualificado para su mantenimiento, pero recientemente los precios de estos dispositivos han descendido. Dependiendo de la tarea que se desee realizar, no es necesario contratar personal de vigilancia, y ahora están controlados por un sistema de visión artificial.

2.5. Limitaciones.

Una de las limitaciones que tiene nuestro proyecto es el tiempo de procesado de las imágenes captadas, ya que dichas imágenes no serán adquiridas en tiempo real.

A su vez, la iluminación es la parte más crítica dentro de un sistema de visión, dependiendo de las condiciones de iluminación, las imágenes digitales a menudo tienen señales de ruido deficientes, iluminación desigual, impurezas de enfoque, deslumbramiento, cambio de color y otros problemas que distorsionan la calidad general de la imagen.

Por lo que se tendrá en cuenta la intensidad de luz necesaria, la reflectividad del objeto y el tipo de cámara utilizada.

Por otro lado, ha de considerarse también las interferencias técnicas, como puede ser el ruido, o las interferencias debidas al contexto, que son la escala, la oclusión, un objeto puede estar en segundo plano, o el fondo.

Debido a que gran parte del contenido del programa de detección de movimientos tiene que ver con el análisis de imágenes, se deben tener en cuenta las limitaciones que ello conlleva en cuanto a captación de datos.

Esto nos lleva a plantear si las imágenes de personas captadas mediante un sistema de videovigilancia pueden considerarse como dato de carácter personal.

En la normativa sobre cámaras de videovigilancia, la imagen de una persona constituye un dato de carácter personal en la medida que pueda identificarse a dicha persona, por lo tanto, puede ser objeto de tratamiento para diversas finalidades y debe aplicarse la normativa sobre protección de datos.

Esta norma debe aplicarse siempre que exista grabación, captación, transmisión o almacenamiento de imágenes, incluidas su retransmisión en tiempo real.

No obstante, hay excepciones para el cumplimiento de la LOPD (Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales).

Una de las cosas a tener en cuenta para realizar el tratamiento de la información de acuerdo con la normativa es la finalidad de la aplicación.

Se ha realizado este trabajo de fin de grado con intenciones educativas y de búsqueda de información y tendrá aplicación en el ámbito personal o doméstico, y de acuerdo con el Reglamento General de Protección de Datos no se aplica esta norma al tratamiento de imágenes grabadas por una persona física en el ejercicio de actividades exclusivamente personales o domésticas no es necesario formalizar el registro de los datos.

3. Planteamiento de soluciones alternativas y justificación de la solución adoptada.

Para la realización de este trabajo se han analizado diversas soluciones existentes en el mercado para la detección de movimiento a parte de los sistemas software, como pueden ser los sensores de detección de movimiento en un área determinada mediante la emisión y/o recepción de señales.

3.1. Sensores.

Existen muchos tipos de sensores para la detección de movimiento, entre ellos podemos encontrar dos categorías, sensores pasivos (PIR) que son aquellos que no emiten radiaciones, sino que las captan, miden la radiación electromagnética infrarroja que emiten los cuerpos que se encuentran dentro de su campo de visión, y los sensores activos, que están basados en la combinación entre un emisor (diodo LED infrarrojo) y un receptor (fototransistor).

- Sensores de presencia ultrasónicos: a través de una onda ultrasónica que recorre el área captan variaciones en el espacio, dicha oda vuelve al detector tras rebotar en cada objeto que se encuentra en el ambiente, si se detecta un nuevo objeto se activa.

Figura 1. Sensor ultrasónico.

- Sensores de movimiento de infrarrojos: utilizan luces infrarrojas para detectar cambios de calor. Al detectar el movimiento de una masa, a luz registra el cambio y envía una señal a la central receptora. Este sistema se utiliza mayoritariamente en alarmas.

Figura 2. Sensor infrarrojo.

- Sensores de presencia E27: son sensores que captan el movimiento debido a las emisiones de señal de luz del infrarrojo procedente de cualquier fuente de calor existente en el área.

Figura 3. Sensor de proximidad por infrarrojos E27.

.

3.2. Software.

Debido a las cualidades de nuestro proyecto, en el que es necesario realizar la grabación de los movimientos para su posterior visualización y teniendo en cuenta las posibilidades de expansión de un sistema desarrollado mediante programación sobre hardware estándar se han tenido que considerar diversas opciones respecto al software sobre el que se trabajará.

.

3.2.1. Librerías de Visión artificial.

Exisen varias librerías, de uso libre para su uso en la visión artificial, entre ellas proponemos 3, completas para la mayoría de los propósitos, de forma breve las explicamos a continuación:

- OpenCV: biblioteca más conocida, contiene más de 500 algoritmos. Dispone de mucha documentación y está disponible para Linux, Android y Windows.

- Sherlock: software que se basa en una interfaz gráfica para realizar toda la programación, las herramientas que incorpora lo hacen útil para la programación de aplicaciones sin un profundo conocimiento de la visión artificial.

- Halcon: software para el desarrollo de aplicaciones industriales de visión, creado y comercializado pro MVTec. Tiene complejidad media-alta.

3.2.2. Lenguaje de programación.

Para el desarrollo del software de visión artificial se pueden utilizar diversos lenguajes de programación como lo son C++, lenguaje Java o Python.

Las características de cada uno son:

- C++: lenguaje compilado, orientado a objetos y en el cual se pueden encontrar gran variedad de bibliotecas que facilitan la labor del programador.

- Java: posiblemente el lenguaje de programación más usado, puede funcionar en diversas plataformas, pero adolece de la ausencia de una biblioteca de Visión Artificial suficientemente probada y funcional.

- Python: lenguaje fácilmente portable a otras plataformas, con una gran biblioteca estándar, además de sencillo de utilizar debido a su sintaxis, que es muy limpia y se parece mucho al inglés.

3.2.3. Entorno de desarrollo.

A su vez, también se debe seleccionar el entorno de desarrollo integrado para la aplicación, en este caso al utilizar la librería OpenCV es necesario que el entorno que se utilice sea compatible con el compilador MinGW, que es una implementación del compilador Gcc para la plataforma Windows, que permite un desarrollo de aplicaciones sobre un conjunto de la API de Win32 y que puede generar tanto ejecutables como librerías utilizando la mencionada API de Windows; para este proyecto se ha tenido que elegir entre:

- Dev C++: entorno de desarrollo integrado (IDE) para programar en lenguaje C/C++. Usa MinGW como compilador.

- Qt Creator: IDE multiplataforma para desarrollar aplicaciones utilizando Interfaces gráficas de Usuario (GUI). Es compatible con los sistemas operativos GNU, Windows XP y versiones posteriores que requieren el compilador MinGW.

3.3. Justificación de la solución adoptada.

Para la realización de este proyecto, se ha optado por la realización de un programa mediante visión artificial, usando las librerías de OpenCV, utilizando el lenguaje C++.

La primera selección del trabajo ha sido la del sistema operativo, en nuestro caso se ha optado por Windos, ya que es el sistema operativo más utilizado del mundo y es el que viene instalado por defecto en nuestro ordenador.

Con el fin de que el sistema pueda desarrollarse sobre una interfaz de programación ampliamente probado y fiable, se utilizará OpenCV, una librería para el procesamiento de imágenes y manipulación de video que incluye las funciones necesarias para el control y la captura de datos mediante la cámara de un ordenador

La elección de este lenguaje de programación viene dada a que al ser un lenguaje compilado no necesita de una máquina virtual para ejecutarse, es un lenguaje orientado a objetos lo cual permite la programación modular con sus ventajas.

Además, al ser un lenguaje popular y muy utilizado existen multitud de herramientas y repositorios de ayuda que facilitan su utilización, igualmente es un lenguaje muy flexible que permite programar con múltiples estilos, además tiene acceso a memoria de bajo nivel mediante el uso de punteros y variables estáticas, y también debido a que es de los más utilizados por los programadores facilita así la comprensión y ampliación o mantenimiento del código.

Seguidamente se ha elegido como entorno de desarrollo el programa Qt Creator para la aplicación porque ya se ha trabajado con anterioridad con él por lo que se conoce su funcionamiento, es un IDE open source por lo que hay continuas mejoras sobre él además de una comunidad de usuarios activa con multitud de ejemplos de utilización y código, es sencillo de utilizar y es muy intuitivo, y se amolda muy bien a los requisitos necesarios para este proyecto sobre la interfaz gráfica ya que utiliza el editor integrado (Qt Designer) para el desarrollo de aplicaciones con interfaz de usuario basado en widgets. También incluye un asistente de proyecto y un sistema avanzado de ayuda contextual.

4. Descripción detallada de la solución adoptada.

En este apartado se procede a describir detalladamente la librería de visión artifical empleada en el proyecto, OpenCV, así como los diversos módulos de que está compuesta esta.

Igualmente se reseñarán las diferentes fases por las que ha transcurrido el desarrollo del proyecto para, finalmente, realizar una explicación de la interfaz gráfica.

4.1. OpenCV. Librerías OpenCV.

OpenCV, cuyas siglas provienen de los términos Open Source Computer Vision), es una biblioteca libre (de código abierto) de visión artificial y machine learning, originalmente creada por Intel, y lanzada bajo licencia BSD, licencia de software permisiva ya que tiene menos restricciones en comparación con otras y permite el uso del código fuente en software no libre, que lo hace disponible para fines comerciales y de investigación, proporcionando una infraestrucutra común para aplicaciones de visión artificial y acelerando su uso en productos comerciales.

Esta librería es multiplataforma, por lo que proporciona soporte para diversos sistemas operativos como GNU/Linux, Windows, Android y Mac OS X, así como para varias arquitecturas de hardware como x86, x64 o ARM, además admite diferentes lenguajes de programación, aunque esté integramente desarrollado en C++.

OpenCV tiene más de 2500 algoritmos que se pueden utilizar para aplicaciones de análisis y procesamiento de imágenes o video, detección de objetos, detección de rostros y análisis de formas. También cuenta con una amplia fuente de documentación de referencia para desarrolladores actualizada, además de una gran cantidad de ejemplos, tutoriales e incluso manuales de las funciones accesibles para todo el mundo.

Figura 4. Logo OpenCV.

OpenCV presenta una estructura modular, lo que implica que el paquete incluye varias bibliotecas estáticas o compartidas.

En este apartado vamos a explicar brevemente los módulos en los que está dividida la biblioteca, y que usaremos en nuestra aplicación.

- Módulo core: este módulo contiene las funciones principales de la librería, como son las estructuras de datos básicas y las funciones aritméticas que serán utilizadas por el resto de los módulos.

    - En este módulo están las estructuras que posteriormente utilizaremos para el procesado de imágenes de nuestro trabajo, que son la clase Mat y QPixmap.

- Módulo imgproc: módulo de procesamiento de imágenes, permitiendo la manipulación y transformación de cualquier tipo de imagen (cambio de tamaño, deformación…), conversión de espacio de color etc.

    - Nuestro trabajo incluirá este módulo para conversión de color sobre algunas de las imágenes.

- Módulo highgui: contiene las funciones para la lectura y escritura de imágenes (imread, imwrite) y de videos (VideoCapture), así como otras funciones de interfaz de usuario fácil de usar. Este módulo ofrece la posibilidad de mostrar los resultados de las operaciones sin tener que utilizar a componentes de la propia aplicación.

- Módulo video: análisis de video que comprende la estimación de movimiento, la sustracción de fondo y algoritmos para el seguimiento de objetos.

- Módulo features2d: módulo con el que se consigue la detección de y descripción de una serie de características sobre la imagen, además de poseer un framework para la unión de esos puntos.

- Módulo calib3d: este módulo incorpora algoritmos básicos de geometría de múltiples vistas, calibración de cámara única y estéreo, estimación de pose de objeto, algoritmos de correspondencia estéreo y elementos de reconstrucción 3D.

En la figura 5, se muestran los diferentes módulos en los que se estructura OpenCV.

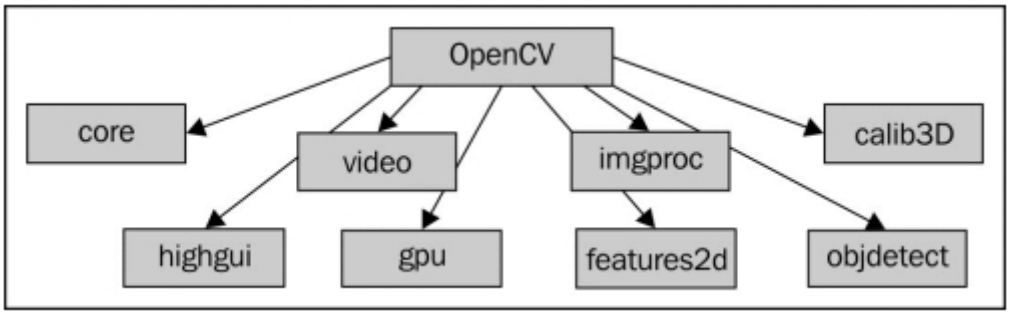


Figura 5. Estructura de OpenCV.

4.2. Diseño del proyecto.

El proyecto ha transcurrido por distintas etapas hasta alcanzar el objetivo final propuesto.

En los siguientes apartados se describen los bloques principales de actuación que han originado las funciones para desarrollar el código para la detección de movimientos.

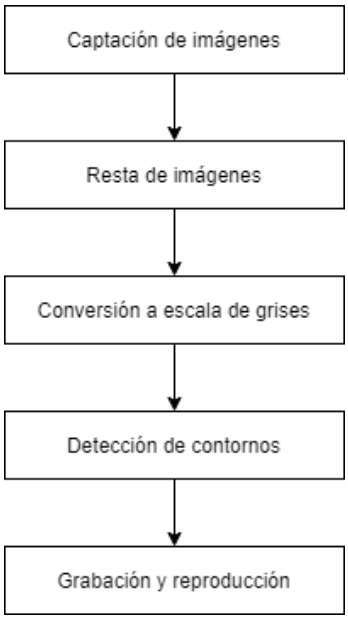


Figura 6. Fases del proyecto.

4.2.1. Captación de imágenes.

El primer paso para alcanzar el objetivo del proyecto es obtener las imágenes que componen los videos de entrada.

Usando la biblioteca OpenCV podemos acceder a la cámara web y cada una de las imágenes capturadas podrán almacenarse.

Para ello es necesario introducir la clase cv::Mat, que es la que nos permite almacenar y manipular los píxeles de una imagen, declarando matrices de 1, 2 o más dimensiones.

Las imágenes digitales están representadas por matrices, estas matrizes están compuestas por píxeles en escala de grises cuyo brillo variará según el valor numérico que tenga, el rango de los valores oscila entre el 0 y el 255, siendo 0 el color negro y el 255 el blanco [Figura 7]. Si se trata de una imagen a color estará constituida por una matriz tridimensional en formato BGR (el formato OpenCV por defecto) [Figura 8].

90	67	68	75	78	98	185	180	153	139	132	106	70	80	81	69	69	67	35	34
92	87	73	78	82	132	180	152	134	120	102	106	95	75	72	63	75	42	19	29
63	102	89	76	98	163	166	164	175	159	120	103	132	96	68	42	49	46	17	22
45	83	109	80	130	158	166	174	158	134	105	71	82	121	80	51	12	50	31	17
39	69	92	115	154	122	144	173	155	105	98	86	82	106	83	76	17	29	41	19
34	80	73	132	144	110	142	181	173	122	100	88	141	142	111	87	33	18	46	36
37	93	88	136	171	164	137	171	190	149	110	137	168	161	132	96	56	23	48	49
66	117	106	147	188	202	198	187	187	159	124	151	167	158	138	105	80	55	59	54
127	136	107	144	188	197	188	184	192	172	124	151	138	108	116	114	84	46	67	54
143	134	99	143	188	172	129	127	179	167	106	118	111	54	70	95	90	46	69	52
141	137	96	146	167	123	91	90	151	156	121	93	78	82	97	91	87	45	66	39
139	137	80	131	162	145	131	129	154	161	158	149	134	122	115	99	84	35	52	30
137	133	56	104	165	167	174	181	175	169	165	162	158	142	124	103	67	19	31	23
135	132	65	86	173	186	200	198	181	171	162	153	145	135	121	104	53	14	15	33
132	132	88	50	149	182	189	191	186	178	166	157	148	131	106	78	28	10	15	44

Figura 7. Matriz de píxeles.

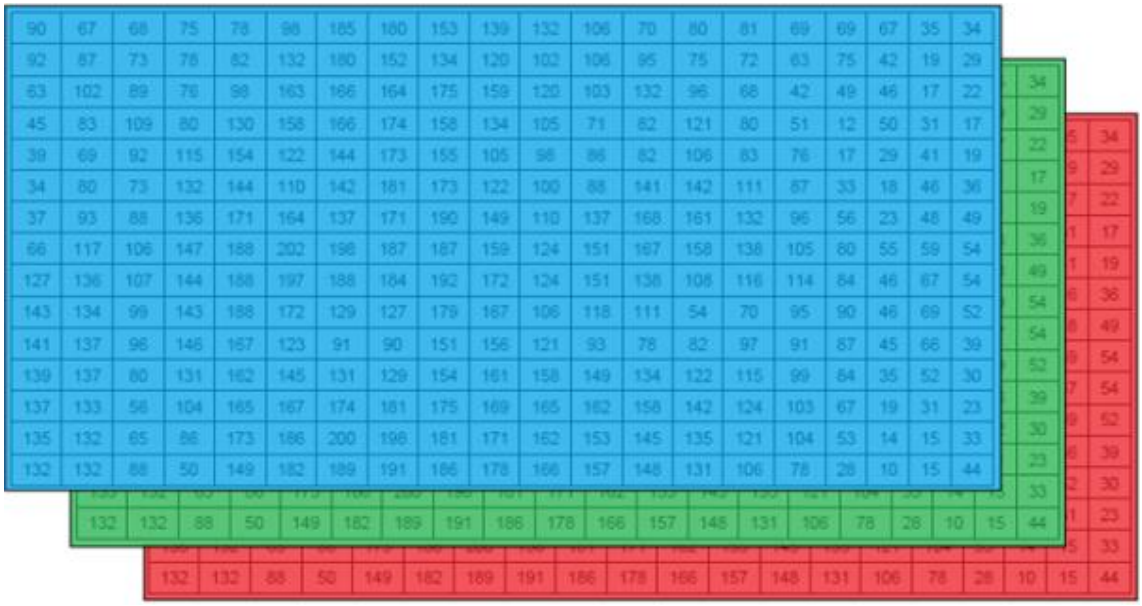


Figura 8. Matriz de píxeles tridimensional BGR.

El primer paso para captar las imágenes provenientes de la cámara web es utilizar la clase, VideoCapture (), y abriremos la cámara en la cual debemos indicar el índice de la cámara que vayamos a utilizar, en nuestro caso al ser la única cámara disponible, la cámara web del ordenador, el valor de dentro del paréntesis será 0.

Para ello la inicializamos:

```
VideoCapture camaracapture;

camaracapture.open(0);
```

El siguiente paso será la creación de los objetos de clase Mat que vayamos a utilizar:

Mat frame; → imagen capturada que mostraremos continuamente.

Mat previousframe; → primera imagen capturada.

Mat nextframe; → segunda imagen capturada.

Mat resta; → imagen resultante de la resta entre la primera y la segunda imagen.

Mat initframe; → imagen inicial capturada, ésta imagen nos permitirá calcular los contornos con posterioridad.

Mat resta_init; → imagen resultante de la resta entre initframe y nextframe para hallar los contornos.

Hay que mencionar que los objetos previousframe y nextframe son lo que nos sirven para la detección del movimiento puesto que los utilizamos para el cálculo de la diferencia entre ellos, si hay alguna diferencia arrancamos la grabación de las imágenes en video y además establecemos un contador de movimientos que nos permite calcular el periodo de tiempo en el cual no hay diferencias entre previousframe y nextframe lo que indica ausencia de movimiento y por lo tanto podemos parar la grabación. De este modo la grabación sólo se realiza de los movimientos más un tiempo prudencial que se añade para evitar la generación de demasiados ficheros separados por instantes muy breves.

Así mismo es importante reseñar la función del objeto initframe que nos permite calcular la diferencia entre la última imagen capturada (en la práctica equivale a nextframe) y la primera tomada en initframe, con ésta diferencia podemos calcular los contornos con claridad puesto que representarían el contorno del sujeto en movimiento (capturado en nextframe) respecto al entorno vacio y estático (capturado en initframe).

Tras verificar que la cámara se haya iniciado correctamente, añadimos dos un timers (reloj interno para que ejecute una acción mediante una interrupción), para ello usaremos la clase QTimer, y los conectaremos con las funciones correspondientes, de este modo al activarse la interrupción con el periodo especificado para el timer se ejecuta automáticamente la función asociada.

Incluiremos la biblioteca <QTimer> y posteriormente se creará las clases y los objetos en el archivo de cabecera MainWindow.h

Biblioteca → `#include <QTimer>`
Clase y objeto →
```
QTimer *imageTimer;
QTimer *differenceTimer;
imageTimer = new QTimer(this);
```

El timer, imageTimer, servirá para mostrar continuamente los frames por pantalla y para grabar en el caso de que se detecte movimiento, mientras que el timer differenceTimer servirá para comprobar las diferencias entre el primer y el segundo frame.

Es conveniente la utilización de dos Timers diferenciados puesto que mostrar un video continuo, que es para lo que se utiliza el primer Timer, requiere de un periodo corto que permita una visualización natural de al menos 12 frames por segundo, pero no es posible calcular la diferencia entre los frames al mismo tiempo pues saturaríamos innecesariamente el procesador, por lo tanto, se introduce un segundo timer especificamente para este cálculo y con un periodo de al menos un segundo.

Aunque sería posible la programación de ambos procesos mediante threads separados, no es totalmente necesario para esta primera aproximación a la detección de movimientos, se explicará en el apartado 6.2 de posibles mejoras.

Prosiguiendo, guardamos las imágenes previamente descritas utilizando la línea de código:

```
camaracapture.read(previousframe);
```

Que situará la imagen captada por la cámara web en el objeto Mat indicado.

Tras ello se hará una redimensión para que las imágenes captadas puedan ser visionadas por completo en las diversas ventanas.

En nuestro caso, hemos utilizado un tamaño de 250x250 píxeles.

```
cv::resize (previousframe,previousframe,Size(250,250));
```

Para la visualización del video por la pantalla, se ha decidido que sea por labels, una etiqueta para un elemento en una interfaz de usuario que se utiliza para mostrar texto o una imagen, por lo que es necesario la incorporación de una librería externa llamada mat2qimage.cpp, para poder convertir un objeto Mat de OpenCV a una imagen de la clase qImage de Qt.

Para empezar, la conversión en sí no se realiza a través de la función mat2QPixmap (), sino que esta función utiliza el método estático fromImage () de QPixmap, que recibe la imagen Qimage como parámetro de entrada, y devuelve el objeto QPixmap en cuestión. Esta imagen QImage se obtiene a su vez llamando al método mat2QImage () que recibe la estructura Mat de origen como parámetro de entrada.

El método mat2QImage () se basa en la creación de un nuevo objeto QImage a partir de su constructor, en el que recibe como argumentos un conjunto de parámetros de la imagen Mat. Aunque en diferente orden, se corresponden con los parámetros recibidos por el constructor de copia de Mat en la operación de conversión inversa.

Las siguientes funciones son fundamentales para poder aplicar las operaciones sobre la imagen cargada en el label:

```
QImage qImage = Mat2QImage(frame);
QPixmap pixmap = QPixmap::fromImage(qImage);
```

Una vez realizados estos pasos, podemos mostrar la imagen por un label situado previamente en nuestra interfaz.

```
ui->label_continuo->clear ();
ui->label_continuo->setPixmap(pixmap);
```

Realizaremos el mismo procedimiento con los demás objetos Mat, ya que los utilizaremos para mostrar la resta de frames y la imagen con contornos.

4.2.2. Substracción de fondo.

La siguiente fase es la substracción de fondo, que consiste en tomar una imagen como base e ir restando los sucesivos fotogramas capturados para obtener las diferencias entre ellos. El resultado es un fodo negro y donde se detecta movimiento el color es diferente. Sin embargo, la substracción de fondo es muy sensible a los cambios de iluminación.

Existen dos modalidades de la técnica de substracción de fondo, según si se obtiene el fondo con una imagen de referencia o con un fotograma anterior.

En nuestro caso se ha realizado con la técnica de substracción con fotogramas anteriores, ya que la técnica con imagen de referencia es más sensible a los cambios de luz, ya que las condiciones de luz pueden variar excesivamente si la comparación de las imágenes se realiza en un periodo de tiempo excesivamente grande. Por ejemplo, si se toma la imagen de referencia por la mañana en un día soleado y el fotograma por la tarde.

En la substracción con fotogras anteriores, el fotograma se compara con un fondo o imagen de referencia tomada en un periodo anterior de tiempo muy pequeño.

Dado que comparar pixel a pixel no sería óptimo, utilizaremos la función cv::absdiff, que calcula la diferencia absoluta por elemento entre dos matrices, siendo su sintaxis:

absdiff(InputArray src1 *,InputArray* src2 *,Matriz de salida* dst *)*

Los parámetros:

- src1: matriz o escalar de entrada
- src2: segunda matriz o escalar de entrada.
- dst: matriz de salida que tendrá el mismo tamaño y tipo que las matrices de entrada.

En nuestro caso utilizamos la función para restar dos frames capturados sucesivamente, nextframe para la segunda imagen captada y previousframe para la primera, la resta de las imágenes se guardará en el objeto resta de la clase Mat, y posteriormente saber si ha habido movimiento (diferencia) entre una y otra.

```
cv::absdiff(nextframe,previousframe,resta);
```

4.2.3. Conversión a escala de grises.

Con el objeto de simplificar los cálculos y evitar la detección involuntaria debida a modificaciones en las condiciones de iluminación del entorno capturado por la cámara es necesario convertir a escala de gris las imágenes capturadas.

Aplicamos la función threshold que se basa en la comparación de cada valor de intensidad de píxeles con respecto a un umbral determinado, todos aquellos que superen el umbral serán pixeles blancos y los que no lo superen serán píxeles negros. Esta función se usa para obtener una imagen de dos niveles de una imagen en una escala de grises o para eliminar el ruido.

threshold (InputArray src, OutputArray dst, double thresh, double maxval, int type)

Siendo los parámetros:

- src: matriz de entrada (punto flotante de múltiples canales, 8 o 32 bits).
- dst matriz de salida del mismo tamaño y tipo y el mism número de canales que src.
- thresh: valor umbral.
- maxval: valor máximo para usar con los tipos de umbral (THRESH_BINARY o THRESH_BINARY_INV).
- type: tipo de umbral.

El umbral en su forma más simple se llama umbral binario (CV_THRESH_BINARY). Además de la imagen de entrada (src) y el valor umbral (thresh), se requiere otro parámetro de entrada, llamado valor máximo (maxValue).

En cada posición de píxel (x, y), el valor de píxel src (x, y) se compara con el umbral. Si src (x, y) es mayor que thresh, establece el valor del píxel de la

imagen de destino dst (x, y) en maxValue; de lo contrario, lo establece en cero. Podemos ver esto reflejado en la figura 9.

Esta operación de umbral se puede expresar como:

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

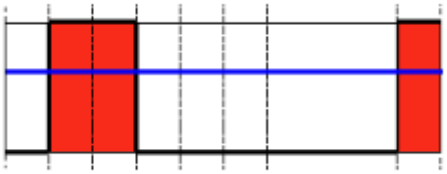


Figura 9. Umbral Binario.

En el caso de nuestro programa, el valor umbral se establece en 80, mientras que el valor máximo se sitúa en 255, que es el valor máximo que puede alcanzar la escala de grises.

No obstante, el usuario podrá modificar dicho umbral mediante el deslizador horizontal.

```
int value = ui->horizontalSlider->value();

cv::threshold(resta,resta,value,255,cv::THRESH_BINARY);
```

Lo siguiente es realizar una operación morfológica, una operación que procesa imágenes basadas en formas y aplica un elemento estructurante a una imagen de entrada y genera una imagen de salida. Tienen una amplia gama de usos como, por ejemplo: eliminación de ruido, aislamiento de elementos individuales y unión de elementos dispares en una imagen.

En nuestro caso utilizaremos la erosión.

La función erode () usa un núcleo especificado para erosionar la imagen, que determina la vecindad de un píxel sobre el cual se toma el mínimo. La función getStructuringElement (MORPH_RECT, Size (3, 3)) se utiliza para obtener el núcleo rectangular con el tamaño de 3 x 3 para esta operación morfológica. La imagen resultante se almacena en resta.

```
cv::erode (resta, resta,cv::getStructuringElement(cv::MORPH_RECT,cv::Size(3,3)));
```

Como ya se ha explicado en el apartado de la captación de imágenes, la imagen de color BGR está formada por una matriz tridimensional, mientras que una imagen en una escala de grises está compuesta por una matriz bidimensional.

Por ello es necesario la utilización de una función que transforme la imagen en color BGR a una imagen en escala de grises.

Un espacio de color o modelo de color es un código que permite que los números representen colores. Los espacios de color más frecuentes son:

- RGB. Una matriz tridimensional con cada una de ellas almacenando los valores de 8 bits (0 a 255) para la intensidad. Cada matriz guarda los colores R – Rojo, G – Verde (Green) y B – Azul (Blue).
- HSV. Es una matriz tridimensional, con cada matriz almacenando el color, la saturación y la intensidad.
- HLS.
- Lab.

Los espacios de color están vistos desde perspectivas diferentes. El espacio BGR o RGB se basa en el hecho de que los colores están formados desde la mezcla por adición de los colores primarios, siendo estos rojo, azul y verde, y su representación tridimensional se puede apreciar en la figura 10.

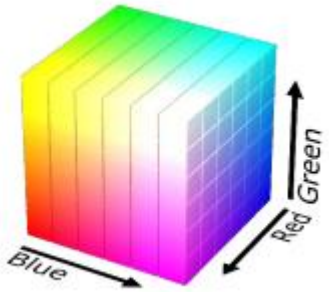


Figura 10. Espacio BGR.

El espacio de escala de grises, no tiene representación tridimensional, porque en él solo influyen dos variables, como lo son el blanco y negro, y las tonalidades que se pueden encontrar en imágenes usan este espacio de color, están representadas en la figura 11.

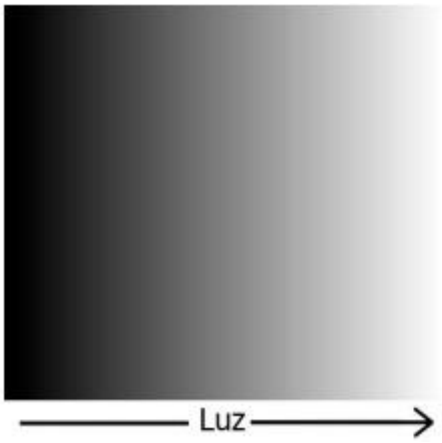


Figura 11. Espacio de grises.

La función cv::cvtColor () se usa para convertir de un espacio de color a otro mientras conserva el mismo tipo de datos.

cvtColor (InputArray src, OutputArray dst, int code, int dstCn = 0)

Consta de cuatro partes:

- matriz de entrada (src), que puede ser una matriz de 8 bits, una matriz sin signo de 16 bits o una matriz de punto flotante de 32 bits.
- matriz de salida (dst) tendrá el mismo tamaño y profundidad que la matriz de entrada.
- código (code).
- parámetro final (dstCn), es el número de canales requeridos en el destino.

La operación de conversión a realizar se especifica mediante el argumento de código, con posibles valores mostrados en la Tabla 1.

Conversion code	Meaning
cv::COLOR_BGR2RGB cv::COLOR_RGB2BGR cv::COLOR_RGBA2BGRA cv::COLOR_BGRA2RGBA	Convert between RGB and BGR color spaces (with or without alpha channel)
cv::COLOR_RGB2RGBA cv::COLOR_BGR2BGRA	Add alpha channel to RGB or BGR image
cv::COLOR_RGBA2RGB cv::COLOR_BGRA2BGR	Remove alpha channel from RGB or BGR image
cv::COLOR_RGB2BGRA cv::COLOR_RGBA2BGR cv::COLOR_BGRA2RGB cv::COLOR_BGR2RGBA	Convert RGB to BGR color spaces while adding or removing alpha channel
cv::COLOR_RGB2GRAY cv::COLOR_BGR2GRAY	Convert RGB or BGR color spaces to grayscale
cv::COLOR_GRAY2RGB cv::COLOR_GRAY2BGR cv::COLOR_RGBA2GRAY cv::COLOR_BGRA2GRAY	Convert grayscale to RGB or BGR color spaces (optionally removing alpha channel in the process)
cv::COLOR_GRAY2RGBA cv::COLOR_GRAY2BGRA	Convert grayscale to RGB or BGR color spaces and add alpha channel

Tabla 1. Conversiones disponibles por cvtColor.

Todas las conversiones de espacio de color utilizan los siguientes convenios: las imágenes de 8 bits están en rango de 0 a 255, las imágenes de 16 bits están en el rango de 0 a 65,536 y números de punto flotante están en el rango de 0.0 a 1. 0.

Cuando convertimos una imagen en escala de grises a una imagen en color, todos los componentes de la imagen resultante se consideran iguales; pero para la transformación inversa (es decir, convertir RGB o BGR, una imagen en color, a escala de grises), el valor se calcula a través de la fórmula ponderada de forma conceptual:

$$Y\ (0.299)\ R + (0.587)\ G + (0114)\ B$$

Por lo tanto, nuestro código quedaría de la siguiente manera:

```
cv::cvtColor (resta, resta, CV_BGR2GRAY);
```

siendo los parámetros, la matriz de entrada src el objeto frame2, fgMask como matriz de salida dst y la operación de conversión a realizar, CV_BGR2GRAY, que cambia del espacio de color BGR a escala de grises, como aparece en la Tabla 1.

A continuación, mostraremos los resultados que realizan las funciones anteriormente descritas sobre una imagen.

En la figura 12, podemos observar el efecto del threshold predeterminado, a 80, aplicado junto a la función de cambio a escala de grises cv::cvtColor.

Así como en la figura 13 y 14 podemos observar el efecto del umbral cuando se situa en un número bajo, o muy alto.

Figura 12. Resultado función Threshold.

Figura 13. Resultado función Threshold a 15

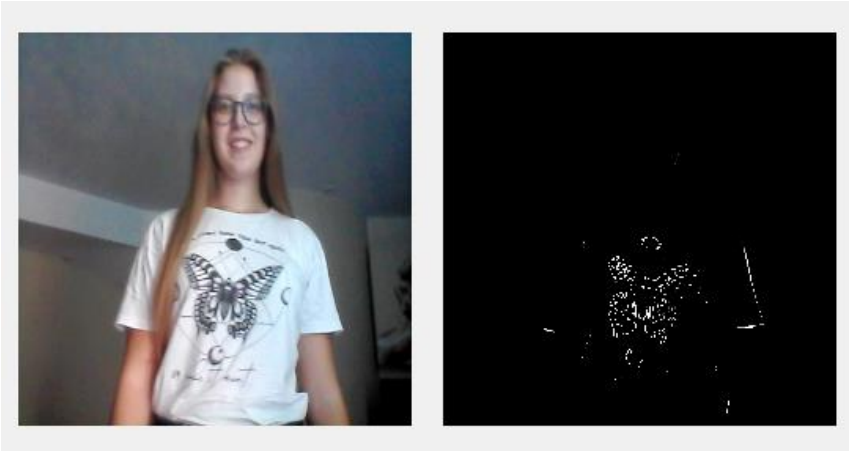


Figure 14. Resultado función Threshold a 125.

En la imagen siguiente, figura 15, podemos observar el funcionamiento del erode junto a la función de cambio a escala de grises cv::cvtColor.

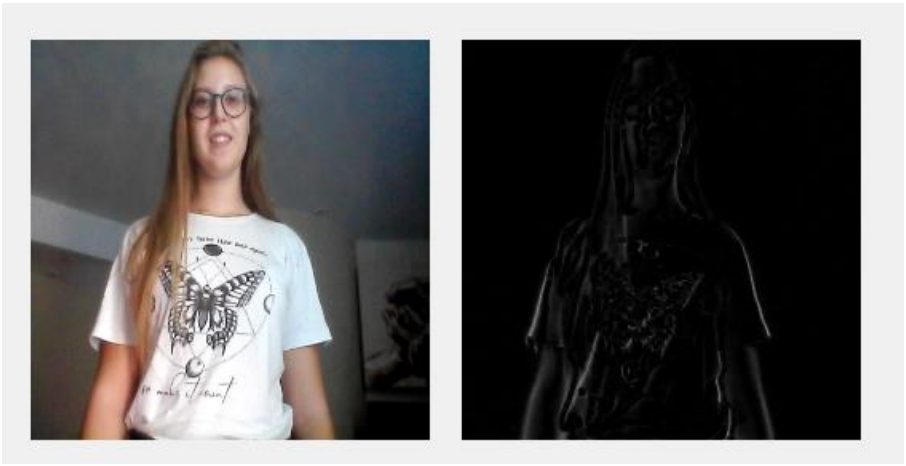


Figura 15. Resultado función erode.

4.2.4. Detección de contornos.

La detección de contornos es la siguiente etapa del proyecto, ya que es la que se responsable de hacer saber al usuario si se ha detectado movimiento.

Según la documentación de OpenCV, “Los contornos se pueden explicar simplemente como una curva que une todos los puntos continuos (a lo largo del límite), que tienen el mismo color o intensidad. Los contornos son una herramienta útil para el análisis de formas y la detección y reconocimiento de objetos”.

Después de la aplicación de las operaciones anteriores se obtiene una imagen binaria, con el objeto que se desea seguir en color blanco y las imágenes restantes en color negro, ya que countNonZero solo funciona con imágenes en gris, de un solo canal.

Para comprobar si hay movimiento para posteriormente añadir los contornos a la imagen, se usa la función countNonZero (), que devuelve el número de elementos de la matriz distintos a cero, es decir, píxeles que no son negros.

En nuestro caso se utilizará para saber si hay movimiento, utilizando una variable con un número fijo comparándolo con el número de píxels encontrados por la función countNonZero ().

```
cv::countNonZero(resta);
```

Una vez nos hemos asegurado que hay movimiento, procedemos a dibujar los contornos sobre la imagen.

La función de la biblioteca de OpenCV que se va a utilizar es la siguiente: *findContours,* para obtener los bordes de los objetos que se encuentran en la imagen binaria.

La función consta de los siguientes parámetros:

findContours (image (InputOutputArray), contours (OutputArrayOfArrays), hierarchy (OutputArray), mode, method)

- Image: imagen de entrada (binaria) a la que se va a aplicar la detección de contornos.

- Contours: es un vector de contornos, donde cada contorno es un vector de puntos. Se considera como un vector de vectores de puntos que guardan al contorno.

- Hierarchy (jerarquía): incluye información sobre la topología de la imagen, es decir, cómo se relacionan los contornos detectados. Almacena la organización de los contornos de la imagen. Contiene tantos elementos como el número de contornos detectados.

- Mode: permite definir un modo de recuperación de contornos. Existen modos que permiten eliminar las relaciones de jerarquía, o establecer nuevas relaciones que identifiquen sólo los 33 contornos externos, etc. En el caso de la detección de la persona propiamente dicho se va a aplicar la operación en el modo “CV_RETR_EXTERNAL”, que devuelve el contorno exterior extremo como salida.

- Method: método de aproximación de contornos. Dispone de varios modos de trabajo, por ejemplo: no realizar aproximación y almacenar todos los puntos del contorno, realizar una aproximación simple comprimiendo horizontal y verticalmente para reducir los puntos del contorno, o incluso aplicar algoritmos de aproximación muy complejos. En este caso se aplicará el modo “CHAIN-APPROX-SIMPLE”, que considera solo los segmentos horizontales, verticales y diagonales dejando sólo sus puntos que forman dichos segmentos. Ofrece resultados muy buenos y con el mínimo de datos del contorno posible. De esta forma se consigue reducir en gran medida los cálculos que el procesador tiene que llevar a cabo, fomentando la fluidez del programa de seguimiento. Por ejemplo: un rectángulo constaría de 4 puntos.

En nuestro proyecto quedaría de la siguiente forma, formado por resta_init, la imagen binaria de entrada y el vector de vectores declarado llamado contornos, que almacenará los puntos de los contornos encontrados en la imagen binaria:

```
vector<std::vector<cv::Point>> contornos;
findContours (resta_init, contornos,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_SIMPLE)
```

Para cada contorno encontrado, ahora aplicamos una aproximación de una curva poligonal con otra con menos vértices para que la distancia entre ellos sea menor o igual a la precisión especificada.

```
approxPolyDP (curva InputArray, OutputArray approxCurve, doble épsilon ,
bool cerrado )
```

Los parámetros que utiliza son:

-Curva, que es el vector de entrada de un punto almacenado en un vector o Mat.

-Curva aproximada, que será el resultado de la aproximación

-Épsilon, es un parámetro que especifica la precisión de aproximación. Es la distancia máxima entre la aproximación y la curva original

- Cerrado, es un parámetro booleano en el cual si es verdadero la curva aproximada está cerrada, es decir los primeros y últimos vértices están conectados.

```
vector<vector<Point> > contornos_poly (contornos.size() );

approxPolyDP (Mat(contornos[i]), contornos_poly[i], 3, true );
```

Después de eso, realizamos la función que calcula y devuelve el rectángulo límite mínimo vertical superior derecho para un conjunto de puntos o píxeles distintos de cero de la imagen en escala de grises.

```
boundRect[i] = boundingRect (Mat(contornos_poly[i]) );
```

El siguiente paso, es dibujar los rectángulos obtenidos sobre la imagen para posteriormente mostrarlos al usuario.
Para ello se elegirá un color aleatorio, dado por la Clase RNG, que genera números aleatorios y que con la clase Scalar se pasarán los valores de los píxeles.

```
RNG rng (12345);

Scalar color = Scalar (rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
```

Se creará un ciclo for que permitirá para los i contornos encontrados, dibujar el contorno.

La función rectángulo, dibuja un contorno rectangular o un rectángulo relleno cuyas dos esquinas opuestas son pt1 y pt2.
Los parámetros de la función rectangle () son los siguientes:

rectangle(InputOutputArray (img), Point (pt1) ,Point (pt2), const Scalar & (color), int (thickness = 1), int (lineType = LINE_8), int (shift = 0))

- img: la imagen.
- pt1: vértice del rectángulo.
- pt2: vértice opuesto a pt1 del rectángulo.
- color: rectángulo de color o brillo (imagen en escala de grises).
- grosor: espesor de las líneas que componen el rectángulo, en el caso de valor negativo se dibujará un rectángulo relleno.
- tipo de línea
- cambio: número de bits fraccionarios en las coordenadas del punto.

```
rectangle (resta, boundRect[i].tl(), boundRect[i].br(), color, 1, 8, 0 );
```

4.2.5. Grabación y reproducción de video.

Para la grabación de imágenes capturadas por la cámara, utilizamos la clase VideoWriter (), que proporciona API para escribir archivos de video o secuencias de imágenes.

VideoWriter (const String& filename, int fourcc, doble fps, tamaño frameSize, bool isColor = true)

- filename: nombre del archivo de video de salida.
- fourcc: código de 4 caracteres utilizado para comprimir las tramas. Por ejemplo, VideoWriter :: fourcc ('P', 'I', 'M', '1') es un códec MPEG-1, VideoWriter :: fourcc ('M', 'J', 'P', 'G ') es un códec motion-jpeg, etc.
- fps: velocidad de fotogramas de la secuencia de video creada
- frameSize: tamaño de los cuadros de video
- isColor: si no es 0, el codificador esperará y codificará marcos de color, si no, funcionará con marcos de grises.

En nuestro caso, el nombre del archivo será un string, que cambiará según la hora en el que se vaya a captar el video, por lo que cada archivo se guarda como grabación_yyyy-MM-dd_HH.mm.ss.

```
const QString filename = "grabacion_" + QDateTime::currentDateTime().toString("yyyy-MM-dd_HH.mm.ss") + ".avi";
std::string sfilename = filename.toLocal8Bit().constData();

video = VideoWriter (sfilename, CV_FOURCC('M','J','P','G'), 10, Size(frame_width,frame_height));
```

Seguidamente se guarda el frame captado, escribiéndolo en el archivo creado en el paso anterior. Cabe destacar que la imagen que se guarda es almacenada en el directorio donde se encuentra el programa, por lo que no es necesario crear o definir la ruta de almacenamiento con antelación.

```
video.write(frame);
```

La siguiente función facilita la reproducción del video anteriormente guardado, siendo éste elegido por el usuario para ser mostrado por pantalla.

Para ello se han utilizado las siguientes líneas de código:

```
QString fileName = QFileDialog::getOpenFileName(this,tr("Abre
            grabación"),"",tr("Videos (*.avi)"));

 std::string sfileName = fileName.toLocal8Bit().constData();

                VideoCapture cap(sfileName);
```

En ellas se utiliza una función estática que llama a las funciones estándar del sistema para mostrar cuadros de diálogo (API del sistema operativo) devolviendo como valor el nombre del archivo que ha sido seleccionado por el usuario, éste es el cometido de la clase QFileDialog, y especificamos qué tipos de archivos puede seleccionar el usuario garantizando de este modo que sean compatibles con nuestra función de reproducción de videos, en nuestro caso .avi, ya que se trata de una grabación de video mediante el estándar Audio Video Interleave.

En la siguiente línea se convierte el QString seleccionado en un String, ya que para capturar el video del archivo es necesario un String como parámetro dentro de VideoCapture ();

Por último, se leen los frames de la grabación del video capturado y se muestra en una ventana:

```
                Mat frame_grabacion;

              cap.read(frame_grabacion);

          imshow("Video_grabación", frame_grabacion);
```

4.3. Interfaz gráfica.

La interfaz se ha desarrollado utilizando la herramienta Qt Designer para el diseño del GUI (Graphic User Interface o interfaz gráfica de usuario), usando objetos e imágenes gráficas para representar la información y acciones disponibles en la interfaz.

Con ello se proporcionará un entorno visual sencillo y facilitará la interacción del usuario con el sistema operativo de un computador.

La interfaz de nuestro programa se compone de una ventana, tal como podemos observar en la figura 16, en la cual podemos percibir diferentes apartados.

Figura 16. Interfaz gráfica.

Por un lado, se sitúan tres botones en la parte inferior, cada uno de ellos conectados con una función.

El botón “ver cámara”, abrirá la cámara y nos permitirá ver las imágenes en los diferentes labels situados arriba, en el primero se puede observar la imagen capturada, en el segundo la diferencia de imágenes en gris si se detecta movimiento y en el tercero mostrará donde está el movimiento mediante contornos rectangulares.

En el caso de que no haya movimiento, solo se mostrará el primer label, tal como vemos en las figuras 17 y 18 que son fondos estáticos, pero en el caso contrario se habilitarán los demás labels mostrando las imágenes anteriormente descritas (imagen en gris y contornos, figura 19, y a su vez se grabará el video de la imagen en movimiento en formato .avi.

Para llevar un registro, en el momento que se detecte movimiento se escribirá automáticamente la fecha y la hora exacta del inicio del movimiento en el PlainTextEdit, situado abajo a la izquierda, figura 20, así como en el video grabado que estará situado en la carpeta del proyecto.

Figura 17. Interfaz gráfica con fondo estático.

Figura 18. Interfaz gráfica con persona estática delante.

Figura 19. Interfaz gráfica con persona en movimiento.

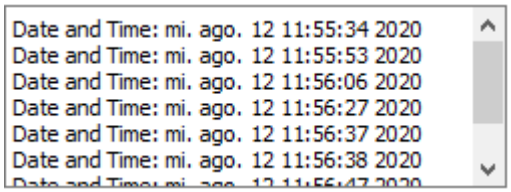


Figura 20. PlainTextEdit.

Con el botón “cerrar cámara”, se cerrará la cámara y se limpiarán los labels y el PlainTextEdit, como si se volviera a reiniciar el programa.

Por último, el botón “ver grabación”, abrirá la carpeta de archivos, tal como se muestra en la figura 21, donde se sitúan todas las grabaciones hechas por el programa en formato .avi, realizadas cuando se ha detectado movimiento, para que el usuario pueda seleccionar el video que desee visualizar. Posteriormente emergerá una ventana que reproducirá el video seleccionado por el usuario, figura 22.

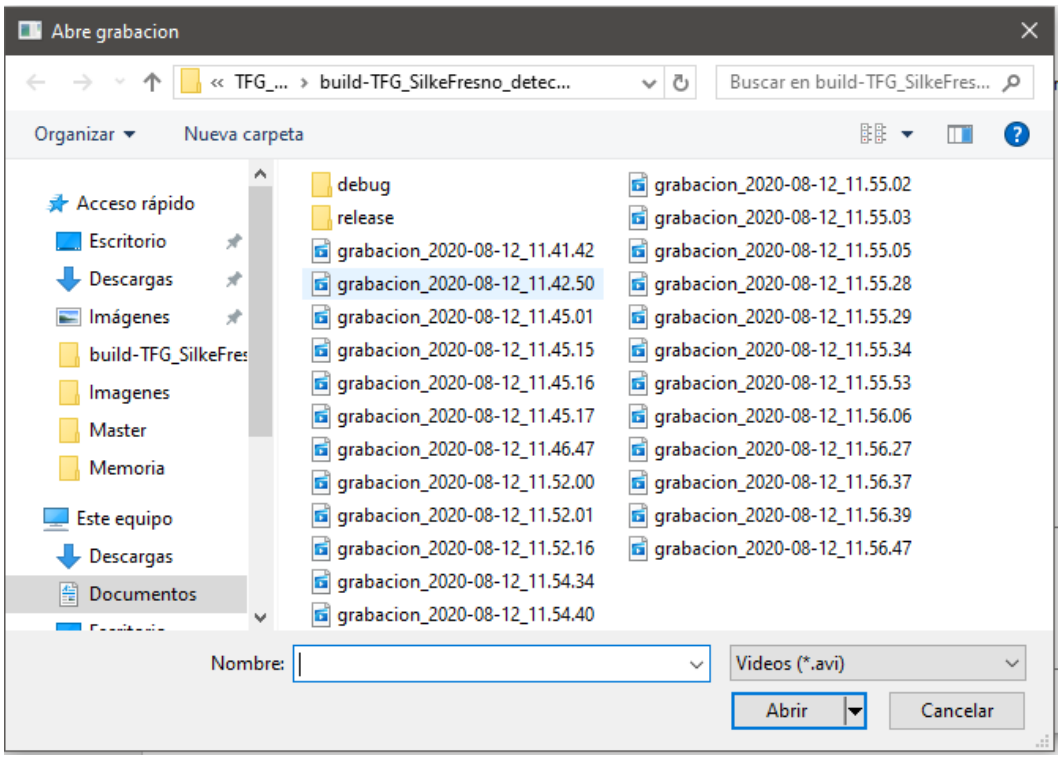


Figura 21. Carpeta de archivos.

Figura 22. Ventana emergente con la grabación.

5. Presupuesto.

El presupuesto elaborado se compone de varias partes, que son, la parte de software y hardware y la parte de coste personal, y que constituyen el coste total del proyecto.

5.1. Coste de Software y Hardware.

El coste comprende los diversos gastos que tienen que ver con el equipo y programas empleados para el desarrollo del trabajo.

Descripción	Cantidad	€/ud	Importe €
Qt creator *	1	Gratuito	Gratuito
OpenCV	1	Gratuito	Gratuito
Ordenador portátil Lenovo con cámara incorporada	1	500	500
TOTAL			500

Figura 23. Presupuesto Coste Software y Hardware.

La licencia de Qt está disponible gratuitamente para código abierto bajo licencias LGPL v3, GPL v2 y v3, que hace que puedas contribuir al desarrollo de software de la comunidad de Qt.

Por lo tanto, en los gastos en este apartado han sido solamente de hardware.

5.2. Coste de Personal.

El coste de personal incluye los tiempos empleados para la realización del trabajo, y varía en función de la tarea desarrollada.

Debemos tener en cuenta el realizador del proyecto, ya que ha sido elaborado por una estudiante de ingeniera electrónica, de modo que los costes de personal irán supeditados a este hecho.

Descripción	Horas	Precio/h	Importe €
Instalación Qt y librerías OpenCV	10	15	150
Concocimiento librerías OpenCV	50	15	750
Implementación programa	135	15	2025
Pruebas simulación	15	15	225
Redacción memoria	100	15	1500
TOTAL			4650

Figura 24. Presupuesto Coste Personal.

5.3. Coste total.

El sumatorio de los importes descompuestos anteriormente dan como resultado el presupuesto total del proyecto de implementación de un sistema software de visión artificial para la detección de objetos en movimiento.

Descripción	Importe €
Coste de Software y Hardware	500
Coste de personal	4650
TOTAL	5150

Figura 25. Presupuesto Total.

6. Anexos.

6.1. Configuración Qt Creator y OpenCV en Windows.

En este apartado se expondrá la instalación del entorno de programación Qt, así como la configuración de la librería OpenCV para sistema operativo Windows 10. Para el correcto funcionamiento del programa se han utilizado las versiones de Qt 5.12.2 con MinGW 7.3.0 y la versión de OpenCV 4.0.1.

Figura 26. OpenCV con Qt creator.

Se ha seguido la guía de la página web www.wiki.qt.io.

Los pasos a seguir para la instalación son:

- 1. Instalación de Qt:

Desde la página de Qt www.qt.io, descargar el instalador de Qt.

Ejecutar el instalador y configurar los siguientes pasos:

- Bienvenido al instalador en línea de Qt: siguiente
- Cuenta Qt: inicio de sesión unificado para Qt: omitir.
- Setup-Qt: siguiente.
- Carpeta de instalación: elegir carpeta en el explorador de archivos (Recomendación en carpeta C).
- Selección de componentes: Qt-Qt5.12.2-MinGW 7.3.0 32 bit.
- Selección de componentes: Qt-Tools-MinGW 7.3.0.
- Acuerdo de licencia: de acuerdo y siguiente.
- Accesos directos del menú: siguiente.
- Listo para instalar: instalar.

Comprobar la correcta instalación de Qt, ejecutando un programa nuevo.

- File → New File or Project→ Qt Widgets Application: elegir.
- Seleccionar la localización del proyecto y nombrarlo.
- Define build system: siguiente.
- Class information: MainWindow (por defecto): siguiente.
- Kit Selection: Desktop Qt 5.12.2 MinGW 32-bit: siguiente.
- Project managment: finalizar.

Empezar el debug y comprobar que aparece una ventana vacía, acabar el debug presionando la cruz roja encima de esa ventana.

- 2. Cmake:

Descargar cmake desde https://cmake.org/download/. Ejecutar el instalador y configurar el cmake de la siguiente manera:

- Bienvenido a CMake setup Wizzard: siguiente.
- Acuerdo de licencia de usuario: [X] aceptar y siguiente.
- Opciones de instalación: [X] Agregar Cmake a la ruta del sistema para todos los usuarios: siguiente.
- Carpeta de destino: C:\Program Files\CMake (default): siguiente.
- Listo para instalar CMake: instalar.

- 3. OpenCV:

Descargar OpenCV, versión 4.0.1, desde la página https://sourceforge.net/projects/opencvlibrary/.

Extraer en C:\. Se creará la carpeta c:\opencv

- Abrir panel de control
- Sistema y seguridad
- Sistema → configuración avanzada del sistema
- Variables del entorno
- Variables del sistema → PATH → Editar
- Nueva variable: C:\Qt_v5\Tools\mingw730_32\bin

Para compilar OpenCV, hay que iniciar la aplicación Cmake que está en la ruta C:\Program Files\CMake\bin\cmake-gui.exe.

Configurar la ruta donde está el código fuente y donde construir los binarios (la carpeta opencv-build se creará en el directorio seleccionado).

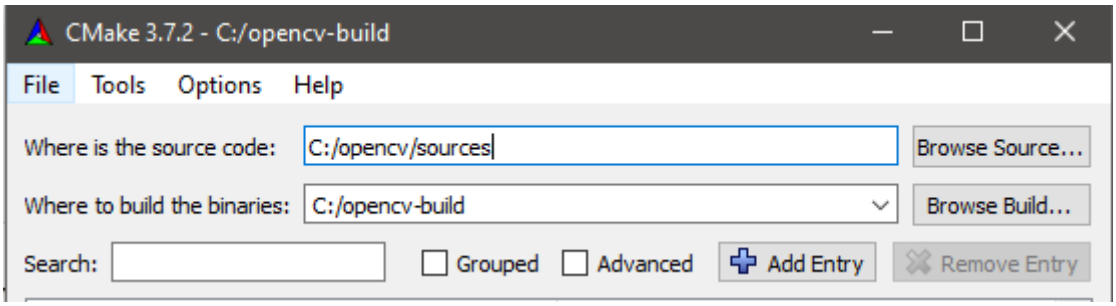


Figura 27. Configuración Cmake.

Dar clic en Configurar y modificar las siguientes configuraciones de la ventana emergente:

- Especificar el generador para este proyecto: MinGW Makefiles.

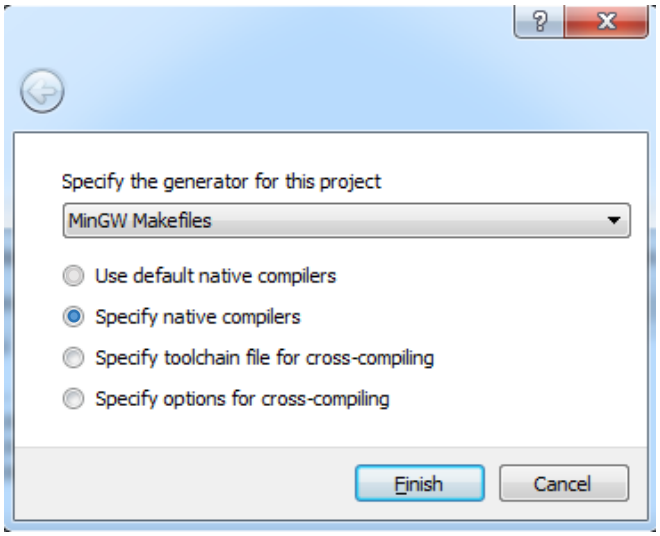


Figura 28. Generador para el proyecto.

- Especificar compiladores nativos.
- Compiladores C: C:\Qt_v5\Tools\mingw730_32\bin\gcc.exe
- Compiladores C++: C:\Qt_v5\Tools\mingw730_32\bin\g++.exe
- Terminar

Modificar las casillas especificadas, que son:

- Marcar la casilla: WITH_QT
- Marcar la casilla: WITH_OPENGL
- Configurar Qt5_DIR: C:/Qt_v5/5.12.2/mingw73_32/lib/cmake/Qt5

- Desmarcar la casilla: ENABLE_PRECOMPILED_HEADERS.

Dar clic a Configurar y después cambiar las siguientes líneas:

- Configurar QT_MAKE_EXECUTABLE:
  C:/Qt_v5/5.12.2/mingw73_32/bin/qmake.exe
- Configurar Qt5Concurrent_DIR:
  C:/Qt_v5/5.12.2/mingw73_32/lib/cmake/Qt5Concurrent
- Configurar Qt5Core_DIR:
  C:/Qt_v5/5.12.2/mingw73_32/lib/cmake/Qt5Core
- Configurar Qt5Gui_DIR:
  C:/Qt_v5/5.12.2/mingw73_32/lib/cmake/Qt5Gui
- Configurar Qt5Test_DIR:
  C:/Qt_v5/5.12.2/mingw73_32/lib/cmake/Qt5Test
- Configurar Qt5Widgets_DIR:
  C:/Qt_v5/5.12.2/mingw73_32/lib/cmake/Qt5Widgets
- Configurar Qt5OpenGL_DIR:
  C:/Qt_v5/5.12.2/mingw73_32/lib/cmake/Qt5OpenGL
- Configurar CMAKE_BUILD_TYPE: release
- Crear y configurar OPENCV_VS_VERSIONINFO_SKIP = 1

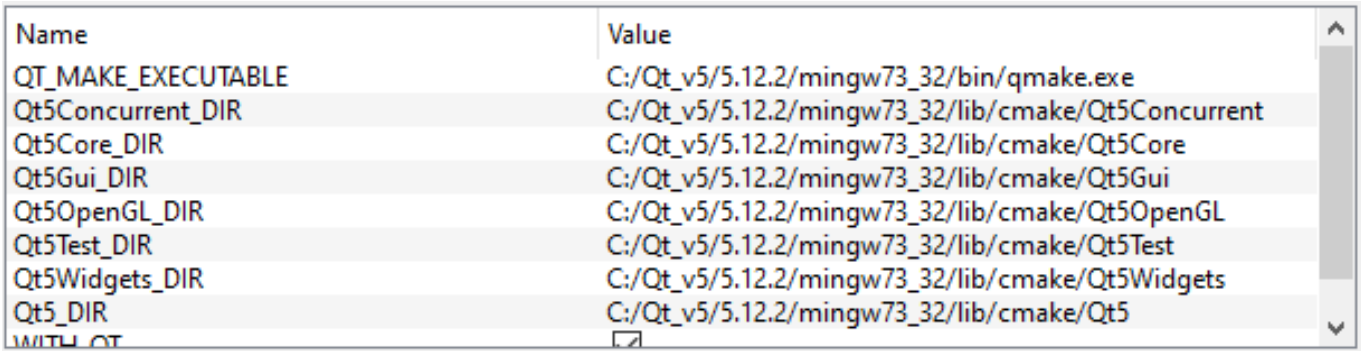


Figura 29. Configuraciones Cmake.

Para finalizar, volver a configurar y clicar el botón de generar.

Por último, abrir la consola de mandos (cmd) de Windows, escribir los siguientes comandos:

- cd c: \

- cd opencv-build
- mingw73-make -j 8
- mingw73-make instalar

Cuando finalize la compilación, los binarios de OpenCV están listos para ser utilizados.

- 4. Bibliotecas compiladas OpenCV a la variable PATH de Windows:

Para agregar las bibliotecas compiladas OpenCV,

- Abrir panel de control
- Sistema y seguridad
- Sistema → configuración avanzada del sistema
- Variables del entorno
- Variables del sistema → PATH → Editar
- Nueva variable: C:\opencv-build\install\x86\mingw\bin

Para concluir, ejecutar un nuevo proyecto, e incluir las librerías tal como se explica en el siguiente apartado.

6.2. Instalación de librerías en Qt.

➢ Librería mat2qimage.

Para la conversión de imágenes de Mat a Qpixmap es necesaria la inclusión de las librerías externas mat2qimage al proyecto.

Para ello, primero se descargan y se guardan los archivos en la misma carpeta donde tengamos guardado el código fuente de nuestro proyecto.

Después en el programa Qt se selecciona la carpeta “sources” → “Add Existing Files”. Buscamos los archivos mat2qimage.cpp y mat2qimage.h en la carpeta donde los hemos guardado previamente y seleccionamos para agregarlos.

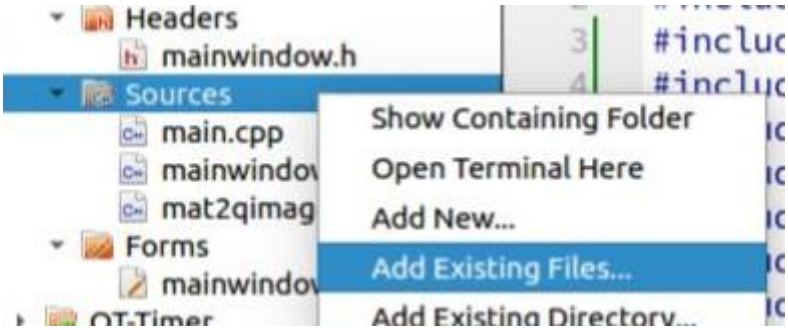


Figura 30. Inclusión de librería Mat2Qimage.

- Inclusión librerías OpenCV en el programa:

Modificar el fichero .pro para incluir la ruta donde están las librerías, y posteriormente incluir las librerías o módulos dentro del proyecto.

Las librerías incluidas son las relativas a los módulos de OpenCV explicadas anteriormente, así como también se incluirán las cabeceras .h de las clases incluidas en las librerías para que se puedan importar directamente en el código.

Existen dos formas para hacerlo, escribiendo nosotros las líneas o añadiendo la librería como librería externa.

Para la primera forma, se debe editar el fichero .pro escribiendo las siguientes líneas, tal como indica la figura 26:

```
INCLUDEPATH += C:\opencv\build\include


LIBS += C:\opencv-build\bin\libopencv_core320.dll
LIBS += C:\opencv-build\bin\libopencv_highgui320.dll
LIBS += C:\opencv-build\bin\libopencv_imgcodecs320.dll
LIBS += C:\opencv-build\bin\libopencv_imgproc320.dll
LIBS += C:\opencv-build\bin\libopencv_features2d320.dll
LIBS += C:\opencv-build\bin\libopencv_calib3d320.dll
LIBS += C:\opencv-build\bin\libopencv_videoio320.dll
LIBS += C:\opencv-build\bin\libopencv_video320.dll
LIBS += C:\opencv\sources\3rdparty\ffmpeg\opencv_ffmpeg.dll
```

Figura 31. Ruta y librerías en .pro.

Para hacerlo de forma externa:

- Realizamos click con el botón derecho sobre la carpeta del proyecto, seleccionamos “Add Library” → “External Library”.

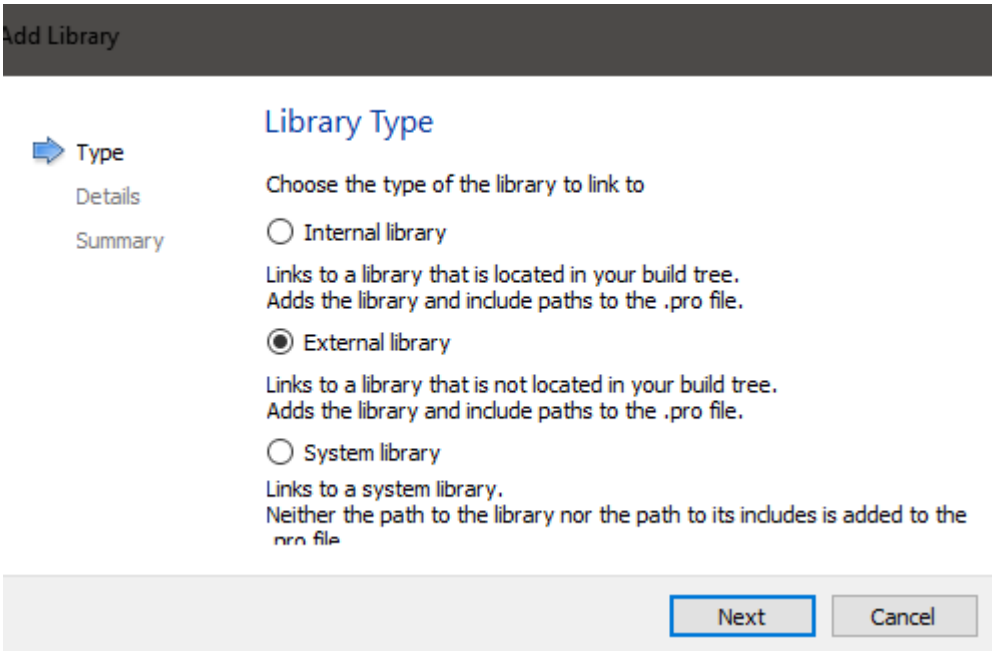


Figura 32. Selección Biblioteca Externa.

- En el archivo de biblioteca (Library File), navegar hasta encontrar el archivo opencv_worldXXXd.lib. Puede estar situado en la siguiente ruta: C:\opencv\build\x64\vc14\lib.
- Para incluir la ruta (Include Path), hay que buscar la carpeta incluir. Esta está situada en la siguiente dirección: C:\opencv\build\include
- Finalizar seleccionando el sistema operativo y biblioteca dinámica.

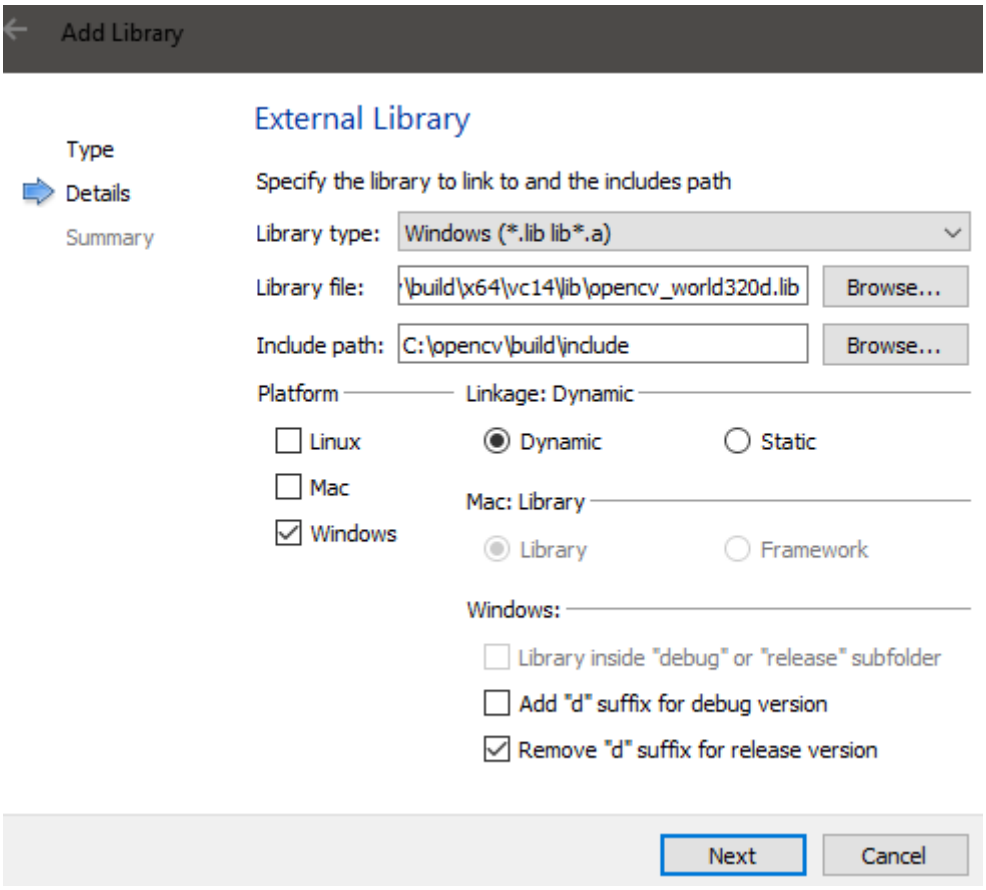


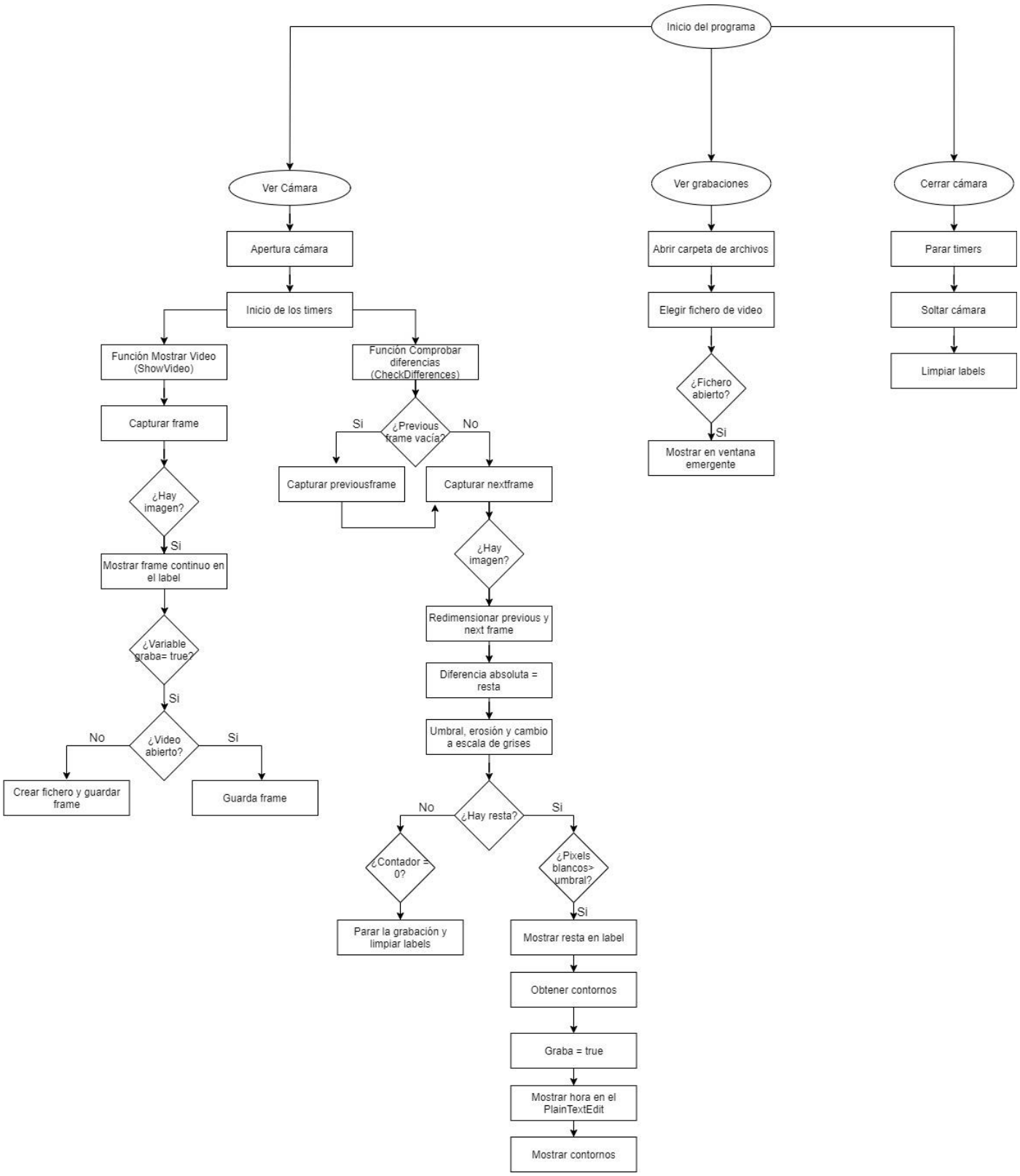
Figura 33. Selección archivos.

6.3. Enlace video demostración.

En el siguiente enlace, se puede observar el funcionamiento de la simulación del programa.

https://clipchamp.com/watch/NL9YDJiD0Z3

6.4. Diagrama de flujo.



6.5. Código.

- MainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

// bibliotecas de opencv
#include<opencv2/core/core.hpp>
#include<opencv2/ml/ml.hpp>
#include<opencv/cv.h>
#include<opencv2/imgproc/imgproc.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/videoio.hpp>
#include<opencv2/imgcodecs.hpp>
#include<opencv2/video/video.hpp>
//biblioteca para cambiar las imágenes de Mat a pixmap
#include "mat2qimage.h"

#include <QTimer>
#include <QtCore>
#include <iostream>
#include <sstream>
#include <stdio.h>
#include <string>
#include <QImage>
#include <QPixmap>
#include <fstream>
#include <QDebug>
#include <QDateTime>
#include <QFileDialog>
#include <QMessageBox>
#include <QSlider>

using namespace cv;
using namespace std;

QTimer *imageTimer;
QTimer *differenceTimer;


VideoCapture camaracapture;
VideoWriter video;

Mat frame;
Mat previousframe;
Mat nextframe;
Mat resta;
Mat resta_init;
Mat initframe;

int contadordeperiodosenmovimiento;
bool graba=false;
bool lreloj=false;
```

```
void MainWindow::on_Iniciar_Videovigilancia_clicked()
{
    camaracapture.open(0);

    if(!camaracapture.isOpened()) //Si la cámara no se puede abrir, se mostrará mensaje
    {
        ShowMessage("No se puedo iniciar la cámara");
    }
    else
    {
        //Si la cámara está abierta
        const int imagePeriod = 1000/25;
        imageTimer = new QTimer(this); //crea temporizador
        imageTimer->setInterval(imagePeriod);
        //Conecta el temporizador con la función que muestra continuamente el frame
        connect(imageTimer,SIGNAL(timeout()),this,SLOT(ShowVideo()));
        imageTimer->start(10);

        //Comprueba la imagen de hace 1s a la imagen de hora
        const int differencePeriod = 1000;  //Tiempo entre cálculo de diferencias en ms
        differenceTimer = new QTimer(this);
        differenceTimer->setInterval(differencePeriod);
        //Conecta el temporizador con la función que capta y analiza las imágenes
        connect(differenceTimer,SIGNAL(timeout()),this,SLOT(CheckDifferences()));
        differenceTimer->start(30);
        contadordeperiodosenmovimiento = 0;
        //Inicializamos intiframe puesto que puede cerrarse la cámara y volver a abrirla en una posición diferente
        initframe = Mat();

    }
}
void MainWindow::CheckDifferences()
{
    if(previousframe.empty()) //Si el primer frame esta vacío entra(la primera vez)
    {
        camaracapture.read(previousframe); //Capturamos frame
    }
    else
    {
        //Capturamos el segundo frame
        camaracapture.read(nextframe);
        if(!nextframe.empty())
        {
            //Redimensionamos los frames
            cv::resize(nextframe,nextframe,Size(250,250));
            cv::resize(previousframe,previousframe,Size(250,250));

            //calculamos la diferencia absoluta con un umbral (absdiff)
            //y extraemos esa diferencia como imagen (resta).
            cv::absdiff(nextframe,previousframe,resta);

            //Aplicamos umbral
            //El valor del threshold vendrá dado por el slider.
            int value = ui->horizontalSlider->value();
            QString val = QString::number(value);
            ui->label_valorumbral->setText(val);
            cv::threshold(resta,resta,value,255,cv::THRESH_BINARY);

            //Aplicamos erosión para eliminar ruido, unir elementos dispares...
            cv::erode(resta,resta,cv::getStructuringElement(cv::MORPH_RECT,cv::Size(3,3)));

            //Cambio a escala de grises
            cv::cvtColor(resta, resta, CV_BGR2GRAY);
```

```
        if(!resta.empty())
        {
            //Determinar el initframe que es el que se usará
            //para calcular el objeto sobre el fondo, se supone
            //que previousframe tiene la imagen del fondo antes de detectar el primer movimiento
            //de este modo descartamos las imágenes vacias si hicieramos la captura nada más
            //encender la cámara o tendríamos que incluir otro timer
            if(initframe.empty())
            {
                //creamos un clon de previousframe en initframe
                cv::resize(previousframe,initframe,Size(250,250));
            }

            int movementthreshold = 200;
            //Cuanto mas bajo mas movimiento detecta per también más falsos positivos
            //si el array contiene al menos 200 elementos no negros podemos decir
            //que hay una diferencia clara entre ellos y por lo tanto movimiento

            if((cv::countNonZero(resta)) > movementthreshold)
            {
                //Llamar a la función para encontrar y dibujar los contornos
                ShowContours();

                //cuando detecta movimiento se incrementa
                contadordeperiodosenmovimiento = contadordeperiodosenmovimiento + 1;

                ui->label_deteccion->setText("Grabando");

                //Grabar video
                //Iniciamos la variable para que grabe en ShowVideo()
                graba = true;
                //Reloj
                if(!lreloj)
                {
                    //Obtener la fecha y hora actuales
                    lreloj=true;
                    QDateTime dateTime = QDateTime::currentDateTime();
                    QString dateTimeString = dateTime.toString();
                    //Mostramos en el TextEdit la hora en la que se inicia la grabación
                    const QString EndofLine = "\n";
                    ui->plainTextEdit->insertPlainText("Date and Time: " + dateTimeString + EndofLine);
                }
            }
            else
            {
                //contador de periodos nos sirve para cortar la grabación
                //cuando no ha habido movimiento durante algún tiempo
                contadordeperiodosenmovimiento = contadordeperiodosenmovimiento - 0.1;
                contadordeperiodosenmovimiento = qMax(0,contadordeperiodosenmovimiento);

                if(contadordeperiodosenmovimiento == 0)
                {
                    //Para la grabación y limpia la variable del reloj
                    ui->label_sustract->clear();
                    ui->label_contorno->clear();
                    ui->label_deteccion->setText("");

                    graba=false;
                    lreloj=false;
                }
            }
        }
        //Como calculamos el movimiento entre dos instantes cambiantes ahora
        //debemos actualizar el frame inicial de cálculo de diferencias
        camaracapture.read(previousframe);
    }
}
}
```

```
void MainWindow::ShowVideo()
{

    camaracapture.read(frame);

    if(!frame.empty())
    {
        //Muestra el frame continuamente en el label
        cv::resize(frame,frame,Size(250,250));
        QImage qImage = Mat2QImage(frame);
        QPixmap pixmap = QPixmap::fromImage(qImage);
        ui->label_continuo->clear();
        ui->label_continuo->setPixmap(pixmap);

        //Grabar
        if(graba)
        {
            if(!video.isOpened())
            {
                //Inicializa la grabación con el nombre dado por fecha y hora en la que se realizó
                int frame_width = 250;
                int frame_height = 250;
                const QString filename = "grabacion_" + QDateTime::currentDateTime().toString("yyyy-MM-dd_HH.mm.ss")
                        + ".avi";
                std::string sfilename = filename.toLocal8Bit().constData();
                //Guarda los frames
                video = VideoWriter(sfilename, CV_FOURCC('M','J','P','G'),10, Size(frame_width,frame_height));
            }

            if(video.isOpened()) //Si ya estaba abierto escribe en el mismo fichero
            {
                video.write(frame);
            }
        }



        else
        {
            if(video.isOpened())
            {
                video.release();
            }
        }
    }
}
```

```
void MainWindow::ShowContours()
{

    RNG rng(12345);

    //mostramos la diferencia capturada en el checkDiferences (entre previousframe y nextframe)
    QImage qImage = Mat2QImage(resta);
    QPixmap pixmap = QPixmap::fromImage(qImage);
    ui->label_sustract->clear();
    ui->label_sustract->setPixmap(pixmap);


    //Realizamos la resta del primer frame con el next frame para tener el contorno
    //de la imagen real y no del movimiento
    //findContours necesita una imagen en gris no un BGR

    cv::absdiff(nextframe,initframe,resta_init);
    cv::threshold(resta_init,resta_init,50,255,cv::THRESH_BINARY);   //127
    cv::erode(resta_init,resta_init,cv::getStructuringElement(cv::MORPH_RECT,cv::Size(6,6)));
    cv::cvtColor(resta_init, resta_init, CV_BGR2GRAY);

    //Obtenemos los bordes de los objetos
    vector<std::vector<cv::Point>> contornos;
    findContours(resta_init,contornos,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_SIMPLE);


    //Obtenemos contornos aproximados a polígonos
    vector<vector<Point> > contornos_poly( contornos.size() );
    vector<Rect> boundRect( contornos.size() );
    int rectsNum=0;
```

```
    Mat drawing = Mat::zeros( resta_init.size(), CV_8UC3 );

    //Dibujamos contorno poligonal
    for( int i = 0; i< rectsNum; i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        rectangle( drawing, boundRect[i].tl(), boundRect[i].br(), color, 1, 8, 0 );
    }

    //muestra los contornos en la etiqueta(label)
    QImage qcImage = Mat2QImage(drawing);
    QPixmap cpixmap = QPixmap::fromImage(qcImage);
    ui->label_contorno->clear();
    ui->label_contorno->setPixmap(cpixmap);
}


void MainWindow::ShowMessage(QString myText)
{    QMessageBox msgBox;
     msgBox.setText(myText);
      msgBox.exec();
}
```

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    //Rango del slider y valor predeterminado
    ui->horizontalSlider->setRange(0,255);
    ui->horizontalSlider->setValue(80);
    connect(ui->horizontalSlider, SIGNAL(valueChanged(int)), this, SLOT(CheckDifferences()));

}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_Detener_Videovigilancia_clicked()
{
    //Limpiar las etiquetas y parar los timers
    imageTimer->stop();
    differenceTimer->stop();
    camaracapture.release();
    ui->label_continuo->clear();
    ui->label_contorno->clear();
    ui->label_sustract->clear();
    ui->plainTextEdit->clear();
}
```

```
void MainWindow::on_Ver_grabaciones_clicked()
{
    //Muestra cuadro de diálogo
    //Devuelve el archivo seleccionado por el usuario
    QString fileName = QFileDialog::getOpenFileName(this,tr("Abre grabacion"),"",tr("Videos (*.avi)"));
    std::string sfileName = fileName.toLocal8Bit().constData();
    VideoCapture cap(sfileName);

    if(cap.isOpened())
    {
        while(1)
        {
            namedWindow("Video_grabacion");
            Mat frame_grabacion;
            cap.read(frame_grabacion);

            //Si no hay más frames en el video, finaliza.
            if(frame_grabacion.empty())
                break;

            //Muestra el frame de la grabación en una ventana
            imshow("Video_grabacion", frame_grabacion);

            int delaybetweenframes = 1000/5;
            if(waitKey(delaybetweenframes) == 'c') break;
        }
    }
}
```

- MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
public slots:

    void on_Iniciar_Videovigilancia_clicked();
    void ShowMessage(QString);
    void ShowVideo();
    void CheckDifferences();
    void ShowContours();
    void on_Detener_Videovigilancia_clicked();
    void on_Ver_grabaciones_clicked();


private slots:


private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

7. Conclusiones.

7.1. Valoración personal.

El proyecto ha conseguido su objetivo de realizar un programa de video vigilancia, capaz de detectar movimiento y grabar y reproducir secuencias de video.

La visión artificial es una herramienta que depende mucho de las condiciones del entorno, donde se aplique, la iluminación…

Para el correcto funcionamiento el programa se debe tener en cuenta la iluminación y el entorno, ya que estas variables han de ser controladas para evitar fallos en la detección del movimiento o reflejos de luz que den lugar a la detección de un movimiento inexistente.

Se puede ver que el programa no es competente cuando no hay iluminación natural suficiente, o cuando el fondo es del mismo color que el objeto situado al frente de la cámara, creando resultados erróneos en la detección de movimiento, sobretodo en el dibujo de los contornos. Tampoco lo será si el usuario modifica los valores del umbral hasta numeros muy bajos o demasiado altos, ya que la detección no será efectiva.

En cuanto a opinión personal la realización de este proyecto ha sido satisfactoria ya que el descubrimiento de esta librería de visión artificial, OpenCV, ha abierto una ventana de conocimiento para posteriores trabajos, ya que posee un gran potencial y una gran variedad de aplicaciones.

El desarrollo del trabajo ha sido duro, en gran parte a las condiciones imperantes debido a la crisis sanitaria actual, y debido a la necesidad de aprender una herramienta nueva y desconocida para llegar a familiarizarse con el manejo de las librerías y funciones incluidas necsarias para el desarrollo del proyecto. Aunque el amplio abanico de posibilidades de esta biblioteca (OpenCV), deja entrever una futura mejora de los resultados obtenidos.

7.2. Posibles mejoras.

Como posible mejora del trabajo se podría considerar la optimización del programa.

Como se comentó en el apartado 4.2.1, se podría modificar la programación de los procesos para mostrar los frames por pantalla y para comprobar las diferencias entre frames mediante threads separados y poder utilizar los múltiples procesadores del equipo ejecutando los procesos de forma concurrente, cosa que sería deseable si se incrementa la resolución de las

imágenes o se disminuyen los periodos, consiguiendo así un algoritmo más rápido.

También como posible mejora sería la integración en el programa de la detección de rostros, y su almacenamiento para aumentar así el objetivo de la seguridad y la vigilancia. Asimismo, se podría implementar el conteo de personas para situaciones de control de aforo, una aplicación muy demandada ahora debida a la crisis sanitaria actual.

8. Bibliografía.

[1] Visión artificial. [Online].

https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial

[2] BOE - Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales. [Online].

https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf

[3] Normativa camaras de vigilancia. [Online].

https://www.grupoadaptalia.es/blog/articulos-explicativos-legales/normativa-camaras-de-vigilancia/

[4] Tipos de sensores de movimiento. [Online].

https://www.comparadoralarmas.com/tipos-de-sensores-de-movimiento-cual-necesitas/

[5] OpenCV. [Online].

https://opencv.org/

[6] Qt – Cross platform software development for embedded systems. [Online].

https://www.qt.io/

[7] OpenCV – Comunidad de OpenCV. [Online].

https://docs.opencv.org/3.4/index.html

[8] Foro OpenCV. [Online]

https://answers.opencv.org/questions/

[9] A. Kaebler & G. Bradski, Learning Open CV: Computer vision in C++with the OpenCV library., 2013. [Online].

https://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf

[10] Intech, Machine Vision- Applications and systems., 2013.

[11] Wiki Qt – comunidad de Qt. [Online].

https://wiki.qt.io/Main

[12] Página con información sobre VideoCapture. [Online].

https://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html

[13] Página con información sobre el Threshold. [Online].

https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html

[14] Página con información sobre Erode. [Online].

https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

[15] Página con información sobre cambios de color. [Online].

https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab

[16] Página con información sobre encontrar contornos. [Online].

https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours

[17] Página con tutoriales OpenCV. [Online].

https://hetpro-store.com/TUTORIALES/category/programacion/opencv/

[18] Configuración Qt y OpenCV en Windows. [Online].

https://wiki.qt.io/How_to_setup_Qt_and_openCV_on_Windows

[19] Biblioteca externa mat2qimpage.cpp. [Online].

https://github.com/JorgeAparicio/ImageQ/blob/master/mat2qimage.cpp