Universidad Zaragoza
1542

Proyecto Fin de Carrera

Desarrollo de un sistema de reconocimiento visual para sistemas Linux embebidos.

Autor/es

Jorge Vega Sánchez

Director/es y/o ponente

José Neira Parra
David Gascón

EINA Escuela de Ingeniería y Arquitectura

2012

Índice

Índice **1**

Índice de figuras **3**

Índice de cuadros **5**

1. Introducción **7**
1.1. Detección de personas 8
1.2. Detección de matrículas 8
1.3. Deteccion de fuego 9

2. Descripción detallada del sistema **9**
2.1. Arquitectura del sistema 9
2.2. Comunicación entre la cámara y el dispositivo Meshlium 10
2.3. Servidor de peticiones TCP y Cola de mensajes 12
2.4. Detección de personas 13
2.4.1. Algoritmo 16
2.4.2. Resultados 17
2.5. Detección de matrículas 18
2.5.1. Búsqueda de candidatos a matrículas 18
2.5.2. Cálculo de la homografía frontal 20
2.5.3. Identificación de los caracteres 20
2.5.4. Resultados 22
2.6. Detección de fuego (fundamento teórico) 24
2.6.1. Búsqueda de imágenes candidatas a contener fuego 24
2.6.2. Selección de la región de llama 25
2.6.3. Confirmación de llama de fuego basada en contornos . . . 25

3. Conclusiones **27**
3.1. Valoración crítica 27
3.2. Posibilidades de continuación y/o ampliación 28
3.3. Incidencias 28
3.4. Opinión personal 28

4. Bibliografía **29**

Índice de figuras

1. Dimensiones y visión general de la cámara 11
2. Esquema del sistema 11
3. Diagrama de clasificador en cascada. 13
4. Esquema aplicación clasificador Haar. 14
5. Ejemplo de análisis completo. 16
6. Ejemplo de Falsos positivo y negativo. 18
7. Ejemplos resultados detección de bordes 19
8. Prueba error detección sencilla blob dentro de otro. 19
9. Ejemplo aplicación operación homografía. 20
10. Imagen con los caracteres modelo. 21
11. Imagen con los caracteres modelo. 21
12. Resultado del programa en terminal. 22
13. Ejemplo de análisis completo. 23
14. Resultados intermedios: a) Imagen original b) Escala de grises c) Imagen binaria d) Imagen binaria suavizada e) Imagen con las regiones pequeñas eliminadas y f) borde de las llamas. 27

Índice de cuadros

1. Características router Meshlium 10
2. Características cámara Ip inalámbrica 10
3. Resultados cuantitativos 17
4. Matriz de confusión 17
5. Resultados cuantitativos globales 22
6. Matriz de confusión etapa búscador 22
7. Resultados de identificacion de caracteres 23

1. Introducción

El objetivo del proyecto consiste en desarrollar un sistema capaz de realizar en tiempo real el tratamiento de imágenes suministradas por una cámara de vídeo, de forma que se puedan generar alarmas ante la aparición de un determinado patrón, para la empresa **Libelium**. Esta empresa se dedica al diseño e implementación de soluciones basadas en conectividad inalámbrica distribuida y monitorización y control de datos del entorno.
El funcionamiento se ejemplifica con la detección y lectura de matrículas y la detección de personas. Además se ha realizado un estudio para una futura implementación para la detección de fuego.
El sistema tendrá que ser eficiente en términos de computación y uso de recursos ya que va estar ejecutado por unaarquitectura de bajas prestaciones y bajo consumo: la plataforma Meshlium. Meshlium es un router multiprotocolo que corre un sistema Linux basado en Debian. Este proyecto también incluye la selección de una webcam que se adecuara a nuestras necesidades, es decir hacer una búsqueda en el mercado, seleccionar una candidata y ponerse en contacto con los distribuidores en España.
El router Meshlium trabaja con redes inalámbricas y por ello la cámara utilizada es diferente a las utilizadas normalmente, necesitamos una cámara con conectividad inalámbrica. Las cámaras con conectividad inalámbricas son novedosas en el mercado porque aparte de poseer esta conectividad sin cables la gran mayoría de ellas corre un sistema Linux empotrado, lo que les otorga mucha más versatilidad. Imprescindible en nuestro caso dado que necesitamos trabajar con las imágenes en un sistema Linux.
Antes de aplicar algoritmos debemos enviar las imágenes de la cámara al dispositivo Meshlium. Esto se realiza buscando entre el código fuente de la interfaz web. Posteriormente configuramos los *eventos* 2.2 de la cámara para pedir imágenes sólo cuando sucede algo extraño, por ejemplo se ha detectado movimiento o se ha producido un ruido. Cuando sucede esto hemos configurado la cámara para que nos envíe un mensaje TCP [1] a un servidor TCP que espera la llegada de estos mensajes. Una vez recibido el mensaje, nos descargamos la imagen de la cámara y la guardamos de forma ordenada y enviamos una señal para que la parte de análisis se ponga a analizar esta imagen. En resumen, el programa principal incluye un proceso que se encarga de comunicar con la cámara y otro para el análisis de las imágenes. Para comunicar ambos procesos se usa una cola de mensajes [2].
Los elementos más destacados de los algoritmo de procesamiento de imágenes son:

1. Algoritmo para la detección de personas:
   a) Tratamiento para adecuar la imagen de entrada al clasificador HAAR.
   b) Uso de clasificadores HAAR para el reconocimiento de caras.
2. Algoritmo para la detección y lectura de matrículas:
   a) Detección de la región de interés en la imagen.
   b) Cálculo de la homografía para obtener la imagen frontal equivalente.
   c) Análisis de conectividad para obtener los blobs de interés (letras y números).

d) Extracción de los blobs y tratamiento (Erosión / Dilatación).

e) Reconocimiento de las letras y números utilizando un clasificador de patrones.

3. Sistema de reconocimiento de fuego. *(prueba conceptual)*:

a) Búsqueda de las llamas por características de color propias.

b) Extracción de estas llamas mediante blobs.

c) Análisis de las características de los blobs de las llamas (perímetro, área, crecimiento,)

Todos los algoritmos se programaron en C/C++ utilizando una librería de 'Visión por Computador'. La biblioteca elegida para desarrollar los algoritmos de visión por computador es OpenCV debido a su solidez y su amplia implantación dentro del extenso mundo de la visión por computador. Su solidez se basa en que fue creada y mantenida por Intel. Posteriormente y ya con una base de desarrollada fue donada a la empresa incubadora de software libre *Willow Garage* que cada 6-8 meses saca una actualización.

1.1. Detección de personas

Para la detección de personas realizamos una búsqueda de rostros dentro de las imágenes. Para la búsqueda/detección de caras miramos la búsqueda de patrones (caras) en la imagen [3] aunque dada la variabilidad del rostro humano la descartamos y seleccionamos utilizar un clasificador en cascada [4] por sus buenos resultados, que se encuentra definido en OpenCV [5]. Este clasificador trabaja con características Haar [6] para la detección. La propia librería OpenCV dota de unos ficheros con las características Haar de varias partes del cuerpo humano, entre ellas el rostro. Realizamos un análisis de que fichero/s de características Haar nos dará un buen ratio detección/coste computacional. Entre los parámetros de este clasificador también debemos fijar un tamaño mínimo de búsqueda. Esto implica que no buscaremos rostros de un tamaño menor al fijado y por lo tanto el análisis será más veloz.

1.2. Detección de matrículas

Por lo general la búsqueda de matrículas se realiza con sistema inteligentes que han sido entrenados. En nuestro caso basamos nuestro algoritmo en la búsqueda de matrículas a partir de las características de este objeto. Nos basamos en la búsqueda de contornos con diferentes métodos para posteriormente extraerlos usando técnicas de extracción de blobs. Un blob es un punto o región en la imagen que se diferencia en sus propiedades como el brillo o el color en comparación con las zonas que le rodean. Para trabajar con blobs existen dos librerías complementarias a OpenCV: cvBlobsLib[7] y cvblob[8]. La decisión fue elegir la biblioteca *cvblob* dado que la otra no tenía versión para trabajar en Linux mientras que cvblob además de trabajar para Linux, Mac o Windows es creada y mantenida por un español, facilitando las consultas sobre dudas.
Primero búscamos blobs candidatos a ser una matricula mediante detección de bordes, posteriormente esta primera selección pasa por el bloque de reorientación (homografía) para poner esas zonas de forma frontal y así facilitar la tarea

de extracción de caracteres y su posterior identificación. Esto se conoce como *homografía [9]* que es un concepto geométrico para la reproyección de ciertas partes de una imagen partiendo del conocimiento de las 4 esquinas frontales del objeto, se pueden ver en los ejemplos [10] y [11].
Con la imagen rectificada debemos extraer los caracteres de la matrícula y para ello aplicamos operaciones como la segmentación[12] y la binarización mediante el método Otsu[13] [14] de binarización dinámica dado que a priori no podemos fijar un umbral para la binarización (consiste en transformar una imagen en escala de grises, 256 valores, a blanco y negro, 2 valores, fijando un umbral para el cambio). La extracción de los caracteres se realiza mediante la extracción de blobs. Posteriormente hay que identificar los caracteres extraídos, esto se realiza mediante una comparativa con unos caracteres modelo extraídos de una imagen. Esta comparativa se llama comparativa de patrones o MatchTemplate [15] [3].

1.3. Deteccion de fuego

Por último y como caso de estudio teórico tenemos la detección de fuego y humo. Hablamos de detección de fuego partiendo de imágenes, dado que existen sensores de calor o CO_2 que servirían para detectar fuego pero que requieren de una cercanía al foco del fuego. Nuestro estudio trata de localizar fuego desde la distancia que nos pueda proporcionar una cámara. Tenemos varios papers [16] [17] [18] que nos dan varias ideas pero el más interesante es [19]. Este paper trata la detección de fuego a partir de características de color y de contorno (perímetro y área). Estas operaciones son asequibles de aplicar dado que ya las he usado en el algoritmo para la detección de matrículas y se basan en operaciones de fácil aplicación con la librería *cvblob*.
Los resultados se escriben en un fichero de logs o de actividad donde escribimos mensajes con la hora. Y hay un fichero de actividad por cada día de actividad del sistema. Además las imágenes descargadas también las guardamos de forma ordenada según su fecha y hora.

2. Descripción detallada del sistema

2.1. Arquitectura del sistema

En lo que concierne a los equipo que vamos a utilizar partimos de un hardware fijo que es el dispositivo Meshlium de la empresa Libelium. Las características de este equipo las vemos en el cuadro 1:

Por el contrario la elección de la cámara web inalámbrica es libre y debe estar basada en las características que necesitamos. Su característica principal es que debe ser una cámara IP inalámbrica. Esta característica es fundamental dado que lo que buscamos es la versatilidad y el poder colocarla sin ninguna limitación. La obligatoriedad de usar este tipo específico de cámara limita mucho el número de cámaras donde elegir porque muchas marcas todavía no han entrado en este segmento del mercado. Tras un estudio del mercado elegimos una cámara de la marca AXIS [20], a elegir entre el modelo M1011-W (básica) y el modelo M1031-W (avanzada). Nos decantamos por la marca AXIS por su buena reputación en el mundo de las cámaras IP y por sus muchos años en el sector. La decisión final nos decantó por el modelo avanzado dado que nos daba

Procesador	500 Mhz (x86)
Memoria RAM	256 MB (DDR)
Disco duro	8 Gb/16 Gb/32 Gb
Potencia	5W (18v)
Rango de temperatura	-20ºC/ 50ºC
Sistema	Linux, Debian. OLSR Mesh protocolo de comunicación.Drivers Madwifi
Seguridad	Autenticación WEP, WPA—PSK, HTTPS y acceso SSH

Cuadro 1: Características router Meshlium

más versatilidad de trabajo, AXIS M1031-W 2. Luego buscamos distribuidores españoles de AXIS y seleccionamos aquel que nos dio el menor precio (170 euros).

Las características de la **cámara IP inalámbrica AXIS M1031-W**.

Sensor Imagen	CMOS RGB de barrido progresivo de 1/4”
Sensibilidad Lumínica	1-10000 lux, F2.0
Compresión Video	H.264 (MPEG-4 parte 10/AVC), Motion JPEGMPEG-4 Parte 2 (ISO/IEC 14496-2)
Resolución	640 x 480 a 160 x 120
Entrada/Salida video	Micrófono y altavoz incorporado
Interfaz Inalámbrica	AXIS M1011-W/M1031-W: IEEE 802.11g/b Antena integrada invisible
Procesador y memoria	ARTPEC-B, 64 MB de RAM, 32 MB de memoria flash
Alimentación	4,9 - 5,1 V CC, 6,5 W máx
Conectores	Toma de CC, RJ-45 10BASE-T/100BASE-T
Sensor PIR (infrarrojos)	Sensor de movimiento de infrarrojos pasivo (PIR) con sensibilidad configurable. Alcance máx.: 6
Condiciones de funcionamiento	Humedad relativa: 20 % – 80 % (sin condensación) Temperatura: 0ºC/ 50ºC

Cuadro 2: Características cámara Ip inalámbrica

2.2. Comunicación entre la cámara y el dispositivo Meshlium

La cámara trae implementada internamente una interfaz web para visualizar las imágenes y para configurar los principales parámetros de la cámara. Primero configuramos la webcam para que se conecte por defecto a la red wifi que crea el dispositivo Meshlium (de nombre meshlium igualmente). Analizando la interfaz web sacamos la ruta donde se encuentran alojadas las fotografías para poder

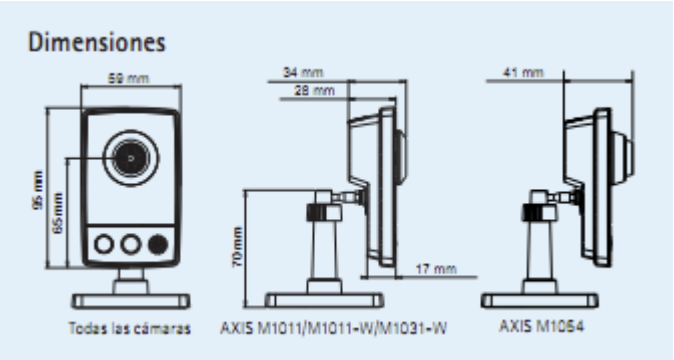


Figura 1: Dimensiones y visión general de la cámara

extraerlas mediante código de terminal que posteriormente podremos ejecutar como una instrucción dentro de nuestro código del programa. A su vez configuramos las alertas/*eventos* de la cámara para que nos envíe un mensaje TCP cuando el sensor de movimiento salte y cuando se supere un umbral de ruido. Cuando configuramos las peticiones TCP de los eventos hay que marcar la casilla para que la cámara mande peticiones consecutivas mientras dure el evento activo. De no marcar esta casilla sólo recibimos una petición aunque se activara el evento varias veces. Estos eventos como los parámetros de la cámara los podemos modificar utilizando el interfaz web o también de una forma más arriesgada modificando los ficheros internos correspondientes de la cámara directamente, esta última opción es la más arriesgada porque trabajas directamente sobre los ficheros del sistema Linux empotrado de la cámara. No obstante teniendo cuidado y tras reiniciar la cámara los cambios surten efecto de la misma manera que si lo hacemos desde la interfaz web.

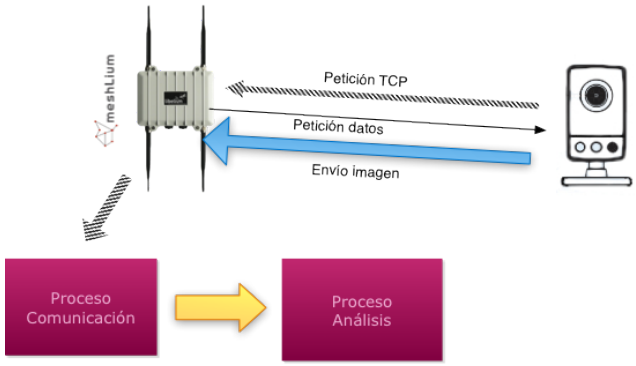


Figura 2: Esquema del sistema

La siguiente etapa es crear una estrategia de comunicación entre la cámara y el dispositivo Meshlium. En este caso dado que la cámara la hemos configurado para enviar peticiones TCP debemos configurar un servidor TCP [1] en nuestro código. Este servidor TCP recibirá las peticiones provenientes de la cámara y descargará una fotografía de la cámara que será la entrada a las funciones de análisis de imágenes. Como vemos tenemos dos partes bien diferenciadas, una

parte de comunicación cámara-Meshlium y otra parte de análisis. La parte de comunicación es asíncrona y la parte de análisis sólo se activará tras haber recibido una petición TCP. En la figura 2 vemos un visión global de la comunicación del sistema.

El programa principal consta de dos módulos:

1. Interacción con la cámara IP Inalámbrica.
    a) Recepción de señales TCP procedentes de la cámara.
    b) Descarga de imágenes de la cámara.
    c) Comunicación con el proceso de análisis.
2. Análisis de las imágenes.
    a) Recepción de mensaje con datos de imagen descargada.
    b) Análisis de la imagen.
    c) Escritura de resultados en el fichero de log.

Con estas características hemos montado una configuración de dos procesos independientes, uno encargado de la comunicación cámara-Meshlium y otro encargado del análisis. Los puntos fuertes de esta estrategia es que estamos permanentemente atentos a la posible llegada de peticiones TCP provenientes de la cámara incluso cuando nos encontremos analizando otras imágenes. Posteriormente tenemos que comunicar ambos procesos para hacerle llegar la imagen al proceso de análisis una vez la hayamos descargado de la cámara. Lo hacemos mediante una cola de mensajes[2]. Realmente mediante la cola de mensajes no envíamos la fotografía de un proceso, solamente comunicamos la referencia (nombre del archivo, en nuestro caso la hora, minutos y segundos de cuando se recibió la petición TCP) de la última fotografía, esto lo guardamos en un buffer (cola de mensajes) para evitar dejarnos alguna imagen sin tratar. Por si queda alguna duda, al llegar una petición TCP, nos descargamos la imagen actual de la cámara y la renombramos con la hora, minutos y segundos y la guardamos en la carpeta IMÁGENES dentro de sucesivas carpetas cuyo nombre corresponden al año, mes y día correspondiente a la fecha actual. En la cola de mensajes sólo pasamos el nombre de este fichero de imagen dado que el resto de parámetros ya los conocemos. Con esto tenemos un espacio donde guardar las imágenes de forma correcta y ordenada. Analizamos mientras hay elementos en la cola.

2.3. Servidor de peticiones TCP y Cola de mensajes

Un **socket** no es más que un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador. Desde el punto de vista de programación, un socket no es más que un "fichero" que se abre de una manera especial. Una vez abierto se pueden escribir y leer datos de él con las habituales funciones de *read()* y *write()* del lenguaje C.
Existen básicamente dos tipos de *canales de comunicación* o sockets, los orientados a conexión y los no orientados a conexión, conocidos como TCP o UDP. Cualquiera de ellos puede transmitir datos en cualquier momento, independientemente de que el otro programa esté "escuchando" o no. En nuestro programa utilizamos socket orientado a conexión TCP[1] para comunicar la camara con el

dispositivo Meshlium. Al programa que actúa de esta forma se le conoce como *servidor*. Su nombre se debe a que normalmente es el que posee la información y la sirve al que se la pida. En nuestro caso particular la cámara es la que manda las peticiones al dispositivo Meshlium, que tras recibirlas opera para extraer la imagen actual de la cámara.
En cuanto a la comunicación entre dos procesos hay varias opciones, entre ellas la memoria compartida o la **cola de mensajes** [2]. Nosotros hemos elegido una cola de mensajes porque se adecua mejor a nuestras necesidades, en la cola vamos almacenando de forma ordenada los nombres de las imágenes que van llegando y las vamos cursando en orden. Se fundamenta en que los procesos introducen mensajes en una cola y se van almacenando en ella. Cuando un proceso extrae un mensaje de la cola, extrae el primer mensaje que se introdujo y dicho mensaje se borra de la cola. Es decir una cola de mensajes sigue una estructura FIFO (First Input First Output).
Unos comandos de *linux* que nos pueden resultar de utilidad para ver el número de colas creadas y su características son *ipcs*[21] y *ipcrm*[22]. *ipcrm* nos permite eliminar las memorias que se nos han quedado "pendientes" en nuestras pruebas. *ipcs* sirve para mostrar las colas compartidas que hay actualmente en uso con sus respectivos identificadores.

2.4. Detección de personas

En esta parte vamos a explicar la parte de detección de personas. Algo que a priori parece muy sencillo pero que posee mucha complejidad implícita. El ser humano se diferencia fácilmente a otros seres de su misma especie por un proceso de aprendizaje que dura toda la vida. De hecho cuando se producen cambios físicos en la persona notamos como nos cuesta más reconocerla. Un sistema de visión por computador es como un recién nacido, a priori no sabe nada y tienen que ser agentes externos los que le digan las pautas para reconocer los diferentes objetos que nos rodean. Para diferenciar a una persona de cualquier otra cosa usamos la detección de rostros o caras dado que es la partes más característica del ser humano.
El rostro humano es un objeto dinámico que tiene un alto grado de variabilidad en su apariencia lo cual hace que su detección sea un problema difícil de tratar en visión por computador. El detector de objetos usado en este trabajo es un

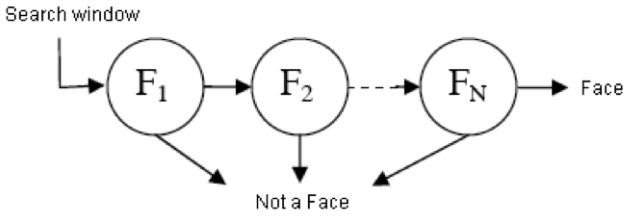


Figura 3: Diagrama de clasificador en cascada.

clasificador, llamado **cascade of boosted classifiers working with haar-like features** [6]. Un clasificador es entrenado con unos cientos de imágenes de ejemplos de un objeto en particular (ej. una cara o un automóvil), llamados ejemplos positivos, que son escalados al mismo tamaño (ej. 20 x 20), y también

entrenado con ejemplos negativos (imágenes arbitrarias del mismo tamaño). Después de que el clasificador es entrenado, puede ser aplicado a regiones de interés (del mismo tamaño que el usado durante el entrenamiento) en una imagen de entrada. La salida del clasificador marca "1" si la región es congruente con el objeto (ej. cara/automóvil), y "0" en el caso contrario. Para buscar el objeto en la totalidad de la imagen, se puede mover la ventana de búsqueda a lo largo de la imagen y revisar cada localición usando el clasificador. El clasificador es diseñado para que pueda ser fácilmente redimensionado en orden de ser capaz de encontrar objetos de interés de diferentes tamaños, lo cual es mas eficiente que redimensionar la imagen por si misma. Por lo tanto, para encontrar un objeto de tamaño desconocido en la imagen, la búsqueda se realiza varias veces con ventanas de diferentes tamaños. La palabra **cascade** en el nombre del clasificador significa que el clasificador resultante consiste en varios clasificadores simples que son aplicados subsecuentemente a una región de interés (también llamada ROI del inglés región of interest) hasta que en alguna etapa el candidato es rechazado o todas las etapas son pasadas, como podemos ver en la figura 3. La palabra **boosted** significa que los clasificadores correspondientes a cada etapa son a la vez complejos y están construidos de clasificadores simples usando una de las cuatro diferentes técnicas de boosting (peso por voto) como *Discrete Adaboost*, *Real Adaboost*, *Gentle Adaboost* y *Logitboost*. En función de los clasificadores base que se utilicen, las distribuciones que se empleen para entrenarlos y el modo de combinarlos, podrán crearse distintas clases del algoritmo genérico de boosting. El algoritmo de boosting empleado por *Viola y Jones* en OpenCV es AdaBoost.
Los clasificadores más básicos son árboles de decisión con al menos dos ramas. Un proceso de reconocimiento puede ser mucho más eficiente si está basado en la detección de características más significativas del tipo de objeto que debe ser detectado. Estas características son llamadas tipo Haar debido a que son computadas de manera similar a los coeficientes de las transformadas *wavelet* de Haar.

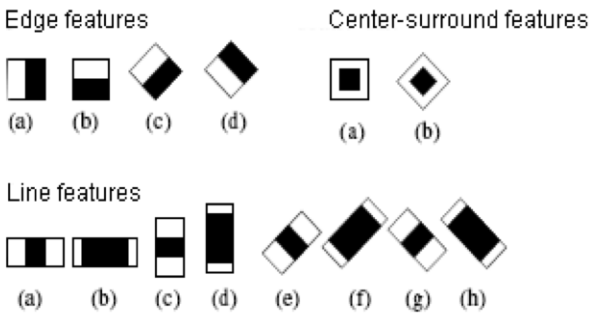


Figura 4: Esquema aplicación clasificador Haar.

En la figura 4 muestra un ejemplo de características Haar. Las usadas en un clasificador en particular dependen de su forma (1a, 2b, etc.), posición en la región de interés y del tamaño. Por ejemplo, en el caso de la característica (2c), la respuesta es calculada como la diferencia entre la suma de los pixeles de la imagen bajo el rectángulo cubriendo la característica completa (incluyendo las franjas blancas y negras) y la suma de los pixeles de la imagen bajo la franja

negra multiplicada por 3 en orden de compensar la diferencia de tamaños entre áreas. La suma de los valores de los pixeles sobre las regiones rectangulares es calculada rápidamente usando imágenes integrales [4].
Las últimas versiones de la librería OpenCV ofrecen una completa serie de clasificadores y ficheros XML que incluyen los ficheros con las características Haar para las caras de frente, caras de perfil, ojos, boca, nariz, parte superior del cuerpo (busto), piernas, cuerpo entero, etc ... Buscamos en un principio la detección de personas usando ficheros de características Haar para las siguientes partes del cuerpo:

1. Cara
2. Ojos
3. Busto
4. Piernas
5. Cuerpo Entero

En principio se busca usar el mayor número de marcadores para obtener una detección lo más fiable posible. Realizamos un análisis en profundidad para descartar los marcadores menos precisos y ver que clasificadores nos aportan más información y de cuales podemos prescindir. Tras diferentes pruebas con múltiples imágenes se llegó a la conclusión de que la gran mayoría de los marcadores eran ineficaces en la mayor parte de las imágenes analizadas y que el que obtenía el mejor rendimiento era un marcador referido a la cara, contenido en el fichero *haarcascade_frontalface_alt.xml*. Esto tiene por contrapunto que no se detectaran las personas que den la espalda a la cámara. Que si pensamos un poco sólo podrían ser reconocidas por los marcadores de las piernas o del cuerpo entero, aunque en la realidad no es así.

Total de clasificadores analizados:

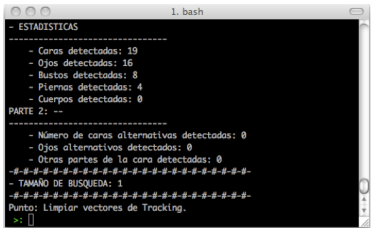
1. Clasificadores primarios:
   a) Cara → *haarcascade_frontalface_alt.xml*
   b) Ojos → *haarcascade_eye.xml*
   c) Busto → *haarcascade_upperbody.xml*
   d) Piernas → *haarcascade_lowerbody.xml*
   e) Cuerpo Entero → *haarcascade_fullbody.xml*
2. Clasificadores alternativos y secundarios
   a) Cara
      1) Cara Frontal Alternativa → *haarcascade_frontalface_alt_tree.xml*
      2) Cara Frontal Alternativa 2 → *haarcascade_frontalface_alt2.xml*
      3) Cara Frontal por defecto → *haarcascade_frontalface_default.xml*
   b) Ojos
      1) Ojo izquierdo → *haarcascade_mcs_lefteye.xml*
      2) Ojo derecho → *haarcascade_mcs_lefteye.xml*

3) Ambos ojos pequeños → *haarcascade_mcs_lefteye.xml*

c) Otras partes de la cara

1) Nariz → *haarcascade_mcs_nose.xml*
2) Boca → *haarcascade_mcs_mouth.xml*

(a) Imagen a analizar



(b) Resultados del análisis

Figura 5: Ejemplo de análisis completo.

Resumiendo, los clasificadores secundarios (nariz y boca) y alternativos (cara y ojos) no nos aportan ventajas apreciables y nos basta con utilizar un clasificadores primario (*haarcascade_frontalface_alt.xml*) para tener una muy buena detección de personas. El clasificador en cascada tiene un parámetro que es el tamaño mínimo de búsqueda, realizamos pruebas con tamaños de ventana entre 5 y 50 pixels en intervalos de 5 y se llegó a la conclusión de que el tamaño mínimo de búsqueda más idóneo era 30x30 pixels, esto implica que no detectaremos rostros de dimensiones inferiores pero si mayores. Como resultado final tenemos algo que a priori parece muy simple pero que llega como conclusión de un análisis mucho más intensivo. Además debido a las limitaciones de hardware que tenemos cumplimos un buen ratio de detección/consumo de recursos.

2.4.1. Algoritmo

Partes del código:

1. Cargar los ficheros con las características Haar (XML)
2. Tratar la imagen de entrada
3. Aplicar los clasificadores de Haar
4. Recuadrar las caras en la imagen (opcional, en Meshlium desactivado)
5. Guardado de la imagen con las caras recuadradas (desactivado en Meshlium)
6. Escribir el resultado en el fichero de log

La étapa de **tratamiento de la imagen** se encarga de transforma la imagen a escala de grises y de realizar un ecualizado del histograma.

2.4.2. Resultados

Hemos sometido al algoritmo a unas pruebas preliminares para comprobar su comportamiento. Tomamos una muestra de 200 imágenes con diferentes iluminaciones, enfoques, resolución y perspectiva de las personas para someter al algoritmo a la mayor variabilidad posible de entradas. De estas 200 imágenes, 100 son con personas y 100 sin personas, o dicho de otra forma imágenes positivos y negativas. A su vez en este conjunto de muestras hemos tratado de buscar la mayor variabilidad de personas atendiendo a su raza (caucásica, negra, asiática, hindú) y añadiendo complejidad con el uso de complementos en el rostro humano como puede ser la barba, gafas (vista o de sol), gorra o sombrero, etc.
En el cuadro 3 debemos tener en cuenta que en cada imagen pueden aparecer una o más personas, es por eso que salen unas cifras altas y mayores que 200, número de imágenes analizadas.

Dato	**Valor**
caras totales	605
caras detectadas	467
caras erróneamente detectadas	41

Cuadro 3: Resultados cuantitativos

Verdadero Positivo (VP)	**Verdadero Negativo (VN)**
426	83
Falso Positivo (FP)	**Falso Negativo (FN)**
41	155

Cuadro 4: Matriz de confusión

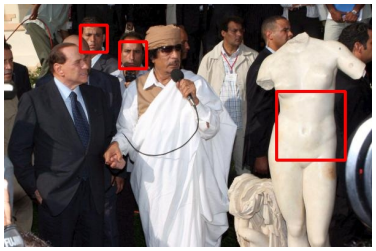
Explicación de la matriz de confusión del cuadro 4:
Verdadero Positivo: objeto detectado (p) y coincide con el objetivo (v).
Verdadero Negativo: objeto no detectado (p) y no hay objetivo (v).
Falso Positivo: objeto detectado (p) pero no es el objetivo (f).
Falso Negativo: objeto no detectado (n) pero hay objetivo en la imagen (f).

$$precision = \frac{vp}{vp + fp} = \frac{426}{426 + 41} = 0{,}9122 \tag{1}$$

$$sensibilidad = \frac{vp}{vp + fn} = \frac{426}{426 + 155} = 0{,}7332 \tag{2}$$

Los valores de *precisión* y *sensibilidad* son buenos, están cercanos a la unidad. Sobretodo el valor de *precisión* que nos indica que se producen muy pocos falsos positivos (detección de personas cuando no hay). El valor de *sensibilidad*, o *recall* en inglés, nos dice que aún no detectamos algunas personas.

Aparte de estas conclusiones, analizando más en profundidad las caras recuadradas por el algoritmo en las diferentes imágenes podemos decir que los puntos débiles de este algoritmo están en la detección de personas de perfil y también con las personas de raza negra con la tez especialmente oscura. Luego en las



(a) Ejemplo de Falso Positivo

(b) Ejemplo de Falso Negativo

Figura 6: Ejemplo de Falsos positivo y negativo.

imágenes aleatorias ha habido algún falso positivo destacando alguna cara de algún animal, pero en general los resultados con estas 100 imágenes aleatorias son buenos, sólo 17 falsos positivos.

2.5. Detección de matrículas

Este estudio viene a detectar e identificar matrículas de vehículos, trata de ir más allá del software que usan lectores de matrículas instalados en estaciones, campus universitarios o entradas a polígonos. Estos lectores trabajan en unas circunstancias muy fijas dado que los vehículos están siempre a una distancia casi fija y en una orientación también fija. Este estudio tratará de detectar e identificar sin limitaciones de distancia y orientación.
El trabajo se ha dividido en tres bloques bien diferenciados:

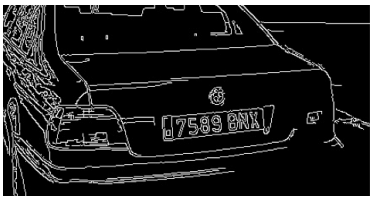
1. Búsqueda de candidatos a matrículas
2. Cálculo de la homografía frontal
3. Identificación de los caracteres

2.5.1. Búsqueda de candidatos a matrículas

Este bloque busca partes de la imagen (blobs) candidatas a ser una matrículas. Decimos candidatos porque no será hasta el bloque de identificación de caracteres donde se tome la última decisión. En este algoritmo buscamos mediante diferentes métodos de **detección de bordes**. Como algoritmos de detección de bordes hemos utilizado Canny, para las matrículas, y Laplace, para los caracteres. En la figura 7 vemos un ejemplo de la aplicación de estos métodos de detección de bordes sobre una imagen. la detección por una parte de caracteres y por otra de *marcos* o matrículas. Posteriormente seleccionamos los marcos que

contengan un número mínimo de caracteres en su interior, el objetivo consiste en obtener al final de este bloque 1 o 2 blobs candidatos a ser una matrícula para continuar el análisis; si este bloque tuviera como salida muchos blobs candidatos se sobrecargarían computacionalmente hablando, los siguiente bloques.

(a) Imagen resultando de Laplace

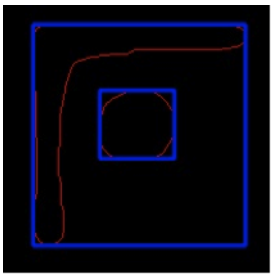


(b) Imagen resultante de Canny

Figura 7: Ejemplos resultados detección de bordes

Ahora tratamos el **sub-algoritmo** de selección, es la parte más crucial de este bloque. Como ya hemos comentado antes tratamos de buscar blobs grandes que contengan blobs más pequeños en su interior. Para eso usamos un análisis en 2 etapas, un primer análisis de tamiz grueso y rápido y un posterior análisis de grano fino y más lento. El análisis grueso y rápido se basa en comprobar la inclusión usando los rectángulos exteriores a los blobs, por lo tanto hacemos simples comparaciones numéricas. El problema que tiene este método es que dá falsos positivos de inclusión como podemos ver en la figura 8.

(a) Imagen de ejemplo



(b) Contornos que circundan los blobs

Figura 8: Prueba error detección sencilla blob dentro de otro.

En la figura 8 creada a tal efecto para ejemplificar estos falsos positivos de forma clara. En la imagen podemos apreciar dos blobs en los ninguno contiene/encierra al otro. En la siguiente imagen muestro los bordes exteriores a los blobs. Como vemos al mostrarlos los rectángulos exteriores a los blobs menos como aparece una inclusión de un rectángulo dentro del otro. Es decir vemos un falso positivo. Es por esto que sólo con este método no obtendríamos una buena selección de los blobs candidatos a ser matrícula.
Ahora vemos la necesidad de aplicar un método más preciso, el método de análisis más fino y lento. Este método se basa es ver si hay inclusión dentro de un blob dentro de otro usando los bordes de los blobs. Este método es más lento porque hacemos más comprobaciones. Para esto hemos creado un método pro-

pio basado en una instrucción de OpenCV, *cvPointPolygonTest* [23]. La función determina si el punto está dentro de un contorno, fuera o se encuentran en contacto (o coincide con un vértice). Devuelve un valor positivo, negativo o cero. Positivo cuando el punto esta fuera del contorno, negativo cuando esta dentro y cero cuando el punto está sobre el contorno.

2.5.2. Cálculo de la homografía frontal

Homografía es un concepto matemático del área de la geometría. Una "homografía", si podemos traducirlo así, es una transformación invertible de un espacio de proyección (por ejemplo, el plano proyectivo real) igual que los mapas de líneas rectas con líneas rectas. Los sinónimos son co-alineación, transformación proyectiva y proyectividad.
La operación consiste, una vez conocidas las cuatro esquinas del objeto/blob, en calcular su matriz de transformación de orientación y posteriormente aplicar esta matriz para reorientar el objeto y disponerlo en una orientación frontal. OpenCV tiene definidas variables específicas para trabajar con matrices asi como métodos para operar con ellas. En la figura 9 vemos un **ejemplo** de aplicación:

(a) Imagen original con las esquinas de la matrícula marcadas

(b) Imagen tras aplicar homografía

Figura 9: Ejemplo aplicación operación homografía.

2.5.3. Identificación de los caracteres

Esta es la etapa final y está formada por dos sub-etapas:

1. Extracción de los caracteres de la imagen
2. Identificación de los caracteres

Primero debemos **extraer los caracteres** de la matrícula. Los caracteres de los caracteres de una matrícula destacan sobre el color de fondo y en algunos casos son de color negro, por ejemplo en las matrículas europeas.

Binarizamos la imagen de forma dinámica, es decir sin difinir un umbral fijo sino que este se calcula de forma dinámica para cada caso particular dado que las condiciones cambian de una imagen a otra. Para eso usamos el método de Binarización Otsu [14]. Una vez binarizada extraemos la matrícula eliminado los posibles blobs exteriores a ella. Para ello buscamos los blobs y me quedo con el más grande, que será la matrícula. Una vez seleccionada la matrícula objetivo

Figura 10: Imagen con los caracteres modelo.

extraigo todos los blobs que contiene en su interior. Posteriormente los ordeno de menor a mayor según la ordenada x de su centro. Realizamos comprobaciones por si no hemos detectado ningún blob, en caso de no detectar ningún blob saltamos a analizar el siguiente blob candidato a ser matrícula. Posteriormente buscamos una secuencia de blobs que tenga una altura muy similar. Con esto elimino todos los blobs detectados que no sean caracteres. Tras la selección de blobs marcados como caracteres elimino los que son clasificados como no caracteres. Por último comprobar que el número de caracteres finales detectados está dentro de la normalidad de matrícula, si es mayor que 4 o menor de 12 caracteres. Si no estamos dentro de los parámetros saldríamos y seguiríamos analizando otra matrícula o esperando a otra foto para analizarla.
La parte crucial de la **identificación** se basa en la comparativa de patrones a partir de la correlación normalizada entre imágenes. Pero antes de la comparativa de patrones realizamos un redimensionamiento del caracter para que la comparativa sea entre dos imágenes del mismo tamaño y mejorar la detección. Los caracteres modelo para realizar la comparativa se extraen de una imagen, figura 10, que contiene todos los caracteres posibles a identificar. Algunos países tiene pequeñas variaciones sobre estos caracteres para evitar falsificaciones pero básicamente la estructura es la misma. La inclusión de otros tipos de letras y números se realizaría en esta etapa. Para realizar esta operación utilizamos el siguiente método de OpenCV, *cvMatchTemplate* [15]. Realizamos la comparativa con los 26 caracteres modelo y definimos el carácter detectado como aquel que ha obtenido el valor más alto de correlación en la comparativa. El algoritmo de identificacion también guarda el segundo caracter más probable. El caracter identificado se guarda en un vector y una vez se han identificado todos los caracteres de la imagen se escribe el resultado en el fichero de *log* o se muestra por pantalla.

Figura 11: Imagen con los caracteres modelo.

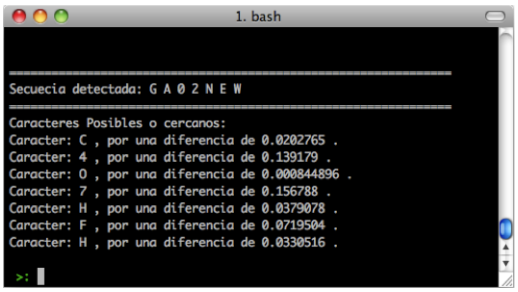


Figura 12: Resultado del programa en terminal.

2.5.4. Resultados

Hemos sometido al algoritmo a unas pruebas preliminares para comprobar su comportamiento. Tomamos una muestra de 350 imágenes con diferentes iluminaciones, enfoques, resolución y orientaciones de los vehículos para someter al algoritmo a la mayor variabilidad posible, además trabajamos con matrículas de muy diversos países porque el algoritmo trata de ser lo más general posible. De estas 350 imágenes, 250 son de coches con su correspondiente matrícula y 100 son imágenes no relacionadas, nos interesa estudiar como se comporta en todas las situaciones posibles.

Dato	**Cantidad**	**Porcentaje**
Matrículas detectadas	109	43,60 %
Matrículas no detectadas	141	57,40 %
Matrículas Detectadas e identificadas	51	20,40 %

Cuadro 5: Resultados cuantitativos globales

Verdadero Positivo (VP)	**Verdadero Negativo (VN)**
109	59
Falso Positivo (FP)	**Falso Negativo (FN)**
103	141

Cuadro 6: Matriz de confusión etapa búscador

Ahora vamos con los valores de *precisión* y *sensibilidad* a partir de la matriz de confusión 6:

$$precision = \frac{vp}{vp + fp} = \frac{109}{109 + 103} = 0{,}5141 \tag{3}$$

$$sensibilidad = \frac{vp}{vp + fn} = \frac{109}{109 + 141} = 0{,}4360 \tag{4}$$

En el cuadro 6 y más concretamente en los valores extraídos de ella como son la *precisión* y el *sensibilidad* son aceptables. Estos valores son mejores cuanto más cercanos estén a la unidad. El factor *sensibilidad* es el cociente entre las matrículas detectadas y el total de matrículas en la muestra de análisis (109+141=250

numero total de imágenes con matrículas en el análisis). Mientras que en la *precisión* entran en juego también los posibles falsos positivos que se hayan podido dar en las muestra de imágenes aleatorias (100). En general los resultados que nos indican que tenemos una mejor detección frente a falsos positivos mientras que sería importante mejorar la étapa de búsqueda de matrículas 2.5.1.

(a) Ejemplo de Falso Positivo + Falso Negativo

(b) Ejemplo de Falso Negativo

Figura 13: Ejemplo de análisis completo.

En la figura 13 vemos ejemplos de falso positivo (detectamos como matrícula algo que no lo es) y falso negativo (no se detecta matrícula y si la hay) que se produce en nuestro método de busqueda. La diferencia entre matrículas detectadas e identificadas se produce porque las detectadas son el resultado de la salida del primer bloque, búsqueda 2.5.1, y en el último bloque, identificacion 2.5.3, se pueden descartar si no se confirma que contienen un número de caracteres dentro del rango de caracteres normal de una matrícula (más de 4 y menos de 12). No obstante la principal causa de esta diferencia se produce en el paso del bloque de búsqueda de matrículas 2.5.1 al de homografía 2.5.2. Los blobs candidatos a ser matrículas no se detectan de manera perfecta y por ello al calcular sus puntos esquina, que son la entrada necesaria para aplicar la operación de homografía, no coinciden con los de la matrícula. Esto lleva a que la proyección realizada con la operación de homografía no nos de una matrícula en vista frontal y con posterioridad al extraer los caracteres 2.5.3 no obtengamos un número de caracteres "normal" y entonces el algoritmo descarte ese blob como matrícula.
Resumiéndo, en la figura 9a hemos marcado las cuatro esquinas de la matrícula y en muchas ocasiones, aplicando nuestro algoritmo, obtendremos un punto o puntos en demarcaciones diferentes que haría que la imagen resultante 9b fuera diferente, torcida en la mayoría de los casos donde la extracción de los caracteres nos hará descartar este blob como matrícula.
Por último hemos realizado un análisis también del acierto en la identificación de caracteres. Para obtener estos resultados tomamos el número de caracteres

Dato	**Valor**
Identificacion correcta	65.58 %
Identificacion erronea	34,42 %

Cuadro 7: Resultados de identificacion de caracteres

identificados correctamente entre el total de caracteres de la matrícula. Si se

identifican correctamente 3 caracteres en una matrícula que contiene 6 caracteres, el resultado parcial sería $3/6 = 0.5 = 50\,\%$.
De la identificación de caracteres también podemos sacar algunas conclusiones. Se producen errores de identificación por parecido en la forma de algunos caracteres. La letra B es parecida al número 8, lo mismo sucede con el número 2 y la letra Z o el número 0 (cero) con la letra O (letra o mayúscula) y con la letra Q y también aunque en menor medida con la letra D. En *menor* medida hay dificultades en la identificación entre el número 7 y la letra T, igualmente entre el número 6 y la letra G o entre las letras H y M y con la letra N y la letra V. O también entre los caracteres Y, K e X. Esto se debe a que tras aplicar sucesivas operaciones sobre las imágenes estas sufren pequeños cambios en su forma original.
Por último comentar que normalmente en Europa la tipografía de las matrículas tiene pequeñas modificaciones para evitar su falsificación. Para manejar esto bastaría con añadir el juego específico de caracteres en la etapa de identificación 2.5.3.

2.6. Detección de fuego (fundamento teórico)

Este es el fundamento teórico para una futura implementación de este algoritmo. Esta sección esta basada en el artículo [19]. El proceso de detección de incendios en su conjunto abarca tres partes:

1. Búsqueda de imágenes/frames candidatas a contener fuego
2. Selección de la región de llama
3. Confirmación de llama de fuego basada en contornos

En el **primer paso**, las imágenes candidatas son detectadas. El **segundo paso** detecta los píxeles de la llama en las imágenes de candidatas a contener fuego mediante un juego de reglas. En el **último paso**, se aplican cuatro operaciones morfológicas (dilatación, erosión, eliminació de regiones pequeñas y la detección de bordes mediante Canny) a todas las regiones clasificadas como llama para obtener el contorno exacto de la llama. Las reglas para decidir si hay fuego se basan en tres características (área, perímetro y grado de redondez de los contornos de la llama).

2.6.1. Búsqueda de imágenes candidatas a contener fuego

En la ecuación 5 la fórmula a aplicar en las imágenes o fotogramas del video a analizar.

$$\begin{cases} p & \text{fuego} & p_R > R_T, p_G > G_T, p_B > B_T \\ p & \text{no-fuego} & \text{resto} \end{cases} \tag{5}$$

En la ecuación refeq2 buscamos si el fotograma j contiene fuego, donde, R_T, G_T y B_T son los umbrales de los canales R, G y B, que se definen de acuerdo a resultados empíricos, que son los intervalos 120 hasta 180, 70-110 y 30-50 [19], respectivamente.

$$\begin{cases} F_j & \text{fuego} & S_j > S_T, S_j > S_{j-1} \\ F_j & \text{no-fuego} & \text{resto} \end{cases} \tag{6}$$

Donde S_T es un umbral de área de fuego de color de identificación según resultados empíricos, que depende de las condiciones ambientales. No obstante debemos de tener en cuenta la siguiente propiedad, el fuego siempre crece por lo que el área de fuego en el fotograma j será mayor que en el instante anterior *(j-1)*. Esta compración de áreas es muy sencilla de realizar con la libreria *cvblob* [8] porque al detectar los blobs en una imagen automaticamente calcula su área entre otros parámetros.
Si tres imagenes consecutivas F_{j-2}, F_{j-1}, F_j se seleccionan como imágenes que contienen fuego, seleccionamos 20 fotogramas consecutivos, desde F_{j-2} hasta F_{j+17}, como candidatos para su posterior procesamiento en las siguientes etapas. De lo contrario, probamos con los siguientes tres fotogramas hasta que todos los fotogramas del video sean procesados.

2.6.2. Selección de la región de llama

Para cada fotograma F_j, comprobamos cada píxel p = (P_R, P_G, P_B). Obtenidos los valores de cada pixel se aplica la fórmula definida en la ecuación 7:

$$\begin{cases} p & \text{fuego} \quad p_R > R_T, p_R > p_G > p_B, p_v > (255 - p_R)\frac{V_T}{R_T} \\ p & \text{no-fuego} \quad \text{resto} \end{cases} \tag{7}$$

Donde R_T es el umbral de la componente tal como se indica en 5, V_T es el umbral de la saturación, que van desde 115 hasta 135 [19] y V_A es el valor de la saturación del píxel p. Aquí, la regla $P_R > P_G > P_B$ se deriva del hecho de que R (red) se convierte en la principal componente en una imagen a color del fuego.

2.6.3. Confirmación de llama de fuego basada en contornos

Esta fase se puede dividir en **cuatro sub-fases**. En **primer lugar**, convertir cada fotograma candidato en el espacio de color RGB en una imagen en escala de grises y posteriormente en una imagen binaria. En **segundo lugar**, aplicar las operaciones de dilatación y erosión en cada imagen binaria para suavizar la imagen. En **tercer lugar**, se eliminan los blobs pequeños y se rellenan los agujeros pequeños en las regiones de la llama para mejorar la eficiencia del detector de bordes. **Finalmente**, la decisión de si hay un incendio o nó se basa en el contorno que se obtiene por el detector de bordes Canny.
Para cada fotograma F_j, j = *1, 2, . . . , 20* lo convierten de RGB a escala de grises GL_J. Para cada píxel p = (P_R, P_G, P_B) en el fotograma j, se obtiene el valor correspondiente en escala de grises pGL mediante la aplicación de la ecuación 8:

$$p_{GL} = \begin{cases} 0{,}59p_R + 0{,}3p_G + 0{,}11p_B & \text{p es un pixel del fuego} \\ 0 & \text{p no es pixel del fuego} \end{cases} \tag{8}$$

Según [19] nos encontraremos que hay pequeños agujeros y puntos aislados en la imagen binaria. Con el fin de limpiar estos pixeles ruidosos o regiones, suavizamos la imagen binaria aplicando las operaciones de la dilatación y erosión, operadores morfológicos básicos.

Después de las operaciones anteriores, tendremos blobs correspondientes a llamas sin agujeros internos. Sin embargo, puede haber algunas regiones muy pequeñas que afectarían el cálculo de características y podría reducir la velocidad de decisión. De hecho, las regiones pequeñas no son tan importantes en el proceso de decisión, por lo que se pueden borrar con el fin de mejorar la eficiencia en la detección. Para quitar los pequeñas blobs de llamas, se tiene en cuenta si el número de pixeles del blob es menor que un umbral predefinido, entonces, la región correspondiente a esta llama se descarta mediante el ajuste de los pixeles en esta región 0 (negro) o lo que es lo mismo borrándolos. En la práctica borrar zonas de una imagen es complejo y todavía no se ha creado nada a tal efecto en la biblioteca cvblob [8] pero si podemos eliminar blobs del deque [24] donde los guardamos porque la librerí trata cada blob como un objeto.
Hasta ahora, hemos obtenido las regiones llama suavizando la imagen y eliminando las regiones pequeñas. A continuación, pasamos a decidir si hay un incendio en los 20 fotogramas sucesivos mediante la detección de las características dinámicas derivadas de los contornos de la llama. En este estudio, se obtiene el contorno de la llama a partir del detector de bordes de Canny. Con los contornos de todas las regiones llama seleccionados, podemos realizar la decisión basándonos en la siguiente regla, ecuación. 9:

$$\begin{cases} \text{Hay fuego} & P_R > P_{ST}, P_C > P_{CT}, P_R > P_{RT} \\ \text{No hay fuego} & \text{resto} \end{cases} \tag{9}$$

Donde P_S, P_C y P_R son los porcentajes de la expansión del área del blob de la llama en los fotogramas, expansión del perímetro en los fotogramas y si el grado de redondez de la llama está dentro de un intervalo predefinido en el conjunto de 20 fotogramas, respectivamente. P_{CT}, P_{ST} y P_{RT} son umbrales predefinidos empíricamente en [19] y que son diferentes según las condiciones ambientales, se establece su valor para todos ellos en 0.7. En este paso el calculo de la expansión de área es muy sencillo como hemos comentado antes, obtener el perímetro es otra operación muy sencilla con *cvblob*, pero el concepto de grado de redondez es nuevo y no está definido en *cvblob*. No obstante es un concepto asequible, tal como se explica en [25] se limita a aplicar a cada blob la siguiente fórmula matemática $grado - redondez = \frac{4\pi area}{perimetro^2}$.

$$N_j^{(S)} = \begin{cases} 1 & S_j > S_{j-1} \\ 0 & \text{resto} \end{cases} \tag{10}$$

$$P_S = \frac{\sum_{j=0}^{20} N_j^{(S)}}{20} \tag{11}$$

En la ecuación 10 S_j es el número de pixeles de la llama en el fotograma F_j y $N_j^{(s)}$ es el flag que indica si el número de pixeles llama aumenta o no frente a su marco anterior. Del mismo modo, la variable P_C se calcula en 12 y 13:

$$N_j^{(C)} = \begin{cases} 1 & C_j > C_{j-1} \\ 0 & \text{resto} \end{cases} \tag{12}$$

$$P_C = \frac{\sum_{j=0}^{20} N_j^{(C)}}{20} \tag{13}$$

donde C_j es el perímetro total de todos los contornos de la llama en el fotograma F_j, es decir, el número de pixeles en el borde de la llama en el fotograma F_j. $N_j^{(c)}$ es el flag que indica si el aumento del perímetro general o no frente a su marco anterior.

$$R_j = \frac{(C_j)^2}{4\pi S_j} \tag{14}$$

$$N_j^{(R)} = \begin{cases} 1 & R_{Tmin} < R_j < R_{Tmax} \\ 0 & \text{resto} \end{cases} \tag{15}$$

$$P_R = \frac{\sum_{i=0}^{20} N_j^{(R)}}{20} \tag{16}$$

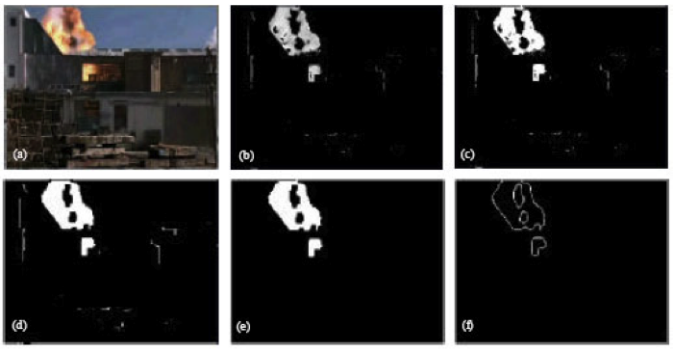


Figura 14: Resultados intermedios: a) Imagen original b) Escala de grises c) Imagen binaria d) Imagen binaria suavizada e) Imagen con las regiones pequeñas eliminadas y f) borde de las llamas.

En la ecuación 15 R_j es la redondez general de todos los contornos de la llama en el fotograma F_j y $N_j^{(R)}$ es el flag que indica si la redondez general se encuentra en el intervalo $[R_{Tmin}, R_{Tmax}]$ o no. Aquí, los valores típicos de R_{Tmin}, R_{Tmax} utilizados son 1.7 y 6 [19], respectivamente. En la figura 14 (*Imagen extraída de [19].*) vemos una secuencia de imágenes que nos muestra las imágenes intermedias de este algoritmo de detección de fuego.
Para confirmar la detección de fuego, se debe c de cumplir la ecuación 9 para los tres parámetros, con los valores umbrales igual a 0.7.

3. Conclusiones

3.1. Valoración crítica

El objetivo del proyecto más allá del desarrollo de un sistema de procesado visual en tiempo real, se ha completado. Los módulos que se han construído como constituyen puntos de partida para su evolución y mejora. En la detección de personas el algoritmo funciona muy bien y con los análisis realizados vemos que funciona muy bien para todo distintas razas. La parte de detección de matrículas no arroja resultados tan buenos pero es un paso inicial hacia la generalidad y no en casos específicos (estilo Campus o estación Intermodal o solamente con matrículas de un país en concreto).Y como ya hemos comentado en 2.5.4 vemos donde se encuentran los puntos débiles a mejorar en un futuro. Debemos tener en cuenta que hemos trabajado con imágenes de baja resolución (640x480 o 0,5 MPx por lo general) para adecuarnos a las condiciones de trabajo normales de la cámara actual y futura de la empresa Libelium. En un futuro habría que mejorar la parte de búsqueda para obtener mejores resultados para conseguir que los puntos esquina del blob candidato a ser matrícula fueran correctos y al aplicar la operación de homografía el rectificado fuera “perfecto”.

3.2. Posibilidades de continuación y/o ampliación

Las posibilidades de continuación son muchas, comenzando por implementar la detección de fuego y pudiendo continuar por la lectura de códigos QR entre muchas otras opciones. El mundo de la visión por computador es muy amplio y con tiempo y el suficiente apoyo se pueden realizar proyectos muy innovadores y con gran valor para la empresa. La sola implementación del método de detección de fuego ya sería una gran continuación.

3.3. Incidencias

Durante la realización de este proyecto ha habido un par de incidencias relevantes. La primera de ellas se refiere a la parte de detección de matrículas en su parte de búsqueda 2.5.1. La **primera** incidencia relevante fue la detección de un error y su corrección de un método de la biblioteca *cvblob* (*ver anexo*) en la obtención de los puntos de contorno de un blob. El método original devolvía entorno al 10-20 % de los puntos del contorno de un blob. La corrección no es muy compleja una vez sabes donde está el error y consiste en modificar unas líneas de un método de la librería. La **segunda** incidencia se produjo igualmente en 2.5.1 y era encontrar un método eficiente para chequear que un blob está contenido dentro de otro más grande. No valía una comprobación punto a punto por su coste computacional. Se buscó mucho hasta encontrar el método [23].

3.4. Opinión personal

Personalmente este proyecto ha sido un reto personal muy importante. Para su realización se ha trabajado muy duro durante un año, con momentos duros cuando no se encontraban los errores o cuando me quedé atascado buscando una solución para poder seleccionar las matrículas. Es decir, este proyecto me ha traído malos y buenos momentos de los que hemos aprendido a saber enfocarlos y a tener esperanza cuando las cosas no funcionan. Además al ser un servidor ingeniero en Telecomunicación, realizar este proyecto tan basado en la programación ha servido para aprender y afianzar muchos conceptos y para ver que soy capaz de desarrollar proyectos informáticos de envergadura.
En lo profesional creo que he crecido al aumentar mis conocimientos en programación y en uso de entornos basados en Linux.

4. Bibliografía

[1] "Artículo sobre sockets TCP y UDP.," 2008. http://www.chuidiang.com/clinux/sockets/sockets_simp.php.

[2] J. Abellán, "Artículo sobre colas de mensajes entre procesos.," 2007. http://www.chuidiang.com/clinux/ipcs/colas.php.

[3] N. Ruddin, "Búsqueda y comparativa con patrones o plantillas.," 2008. http://nashruddin.com/searching-icons-in-a-screenshot-using-template-matching.html.

[4] M. García Guevara, D. Echeverry, Julian, and W. Ardila Urueña, "Grid characteristics and uses: a grid definition," BSc Thesis, Universidad Tecnológica de Pereira, Portugal, June 2008.

[5] "Librería de visión por computador, opencv," 2006. http://opencv.willowgarage.com/.

[6] R. Mungía, R. Mungía, and A. Grau, "Visca: seguimiento de caras servocontrolado," BSc Thesis, Universidad Politécnica de Cataluña, España.

[7] "cvBlobsLib: Librería para el tratamiento de Blobs," 2006. http://opencv.willowgarage.com/wiki/cvBlobsLib.

[8] C. Carnero, "cvBlob: Librería para el tratamiento de Blobs," 2008. http://cvblob.googlecode.com.

[9] M. Grosse, "Ejemplo de la operación Homography con OpenCV escrito en Python.," 2009. http://ioctl.eu/blog/2009/05/13/opencv_homography.

[10] "Definición concepto de Homography en la Wikipedia.." http://en.wikipedia.org/wiki/Homography.

[11] A. Mckay, "Ejemplo de la operación Homography con OpenCV escrito en Python.," 2008. http://ashleymckay.com/ashleymckay/cms/?q=node/126.

[12] D. M. Escrivá, "Segmentation and feature extraction. Contours and blob detection.," 2010. http://blog.damiles.com/2010/12/segmentation-and-feature-extraction-contours-and-blob-detection/.

[13] "Segmentación por umbralización: Binarización otsu, universidad nacional de quilmes, argentina," 2005. http://iaci.unq.edu.ar/materias/vision/archivos/apuntes/Segmentaci%C3%B3n%20por%20umbralizaci%C3%B3n%20-%20M%C3%A9todo%20de%20Otsu.pdf.

[14] K. Mehrez, "Ejemplo de la operación Homography con OpenCV escrito en Python.." http://mehrez.kristou.org/opencv-otsu-thresholding.

[15] N. Ruddin, "Template Matching with OpenCV.," 2008. http://blog.damiles.com/2010/12/segmentation-and-feature-extraction-contours-and-blob-detection/.

[16] T. B. Ugur, D. Yigithan, G. Ugur, and C. A. Enis, "Real-time fire and flame detection in video," BSc Thesis, Bilkent University(Ankara), Turkey, 2005.

[17] S. Verstockt, P. Lambert, R. Van de Wall, and B. Merci, "State of the art in vision-based fire and smoke detection," BSc Thesis, Ghent University, Beligum.

[18] J. Ahlén and S. Seipel, "Early recognition of smoke in digital video," BSc Thesis, Department of Building, Energy andEnvironmental Engineering, University of Gavle, Sweeden, 2010.

[19] Z. Xiao-Lin, Y. Fa-Xin, W. Yu-Chun, L. Zhe-Ming, and S. Guang-Hua, "Early fire detection based on flame contours in video," *Information Technology Journal*, vol. 9, no. 5, pp. 899–908, 2010.

[20] "Marca Axis - Cámaras IP." `http://www.axis.com/es/index.htm`.

[21] C. Llamas, "Comando ipcs.." `http://www.infor.uva.es/~cllamas/concurr/concurrencia.html`.

[22] G. Póo-Caamao, "Comando ipcs.," 2002. `http://www.infor.uva.es/~cllamas/concurr/concurrencia.html`.

[23] "Método de OpenCV para ver si punto dentro de polígono.." `http://opencv.willowgarage.com/documentation/python/imgproc_structural_analysis_and_shape_descriptors.html#pointpolygontest`.

[24] "Información sobre la estructura de datos Deque.," 2006. `http://www.cplusplus.com/reference/stl/deque/`.

[25] "Adición de imágenes." `http://inperc.com/wiki/index.php?title=Roundness`.

[26] P. A. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[27] "OpenCV Wiki page of Cascade Classification." `http://opencv.willowgarage.com/documentation/c/objdetect_cascade_classification.html`.

[28] "Opencv face detection," February 2009. `http://nashruddin.com/OpenCV_Face_Detection`.

[29] N. Kuntz, "Detección de contornos," 2009. `http://dasl.mem.drexel.edu/~noahKuntz/openCVTut7.html`.

[30] N. Kuntz, "Métodos para la detección de contornos 2," 2009. `http://dasl.mem.drexel.edu/~noahKuntz/openCVTut5.html`.

[31] Q. Che, "PowerPoint con ejemplos de código sobre las principales operaciones básicas en OpenCV," 2007. `http://www.discover.uottawa.ca/~qchen/my_presentations/A%20Basic%20Introduction%20to%20OpenCV%20for%20Image%20Processing.pdf`.

[32] N. Kuntz, "Adicción de imágenes," 2009. http://dasl.mem.drexel.edu/~noahKuntz/openCVTut2.html.

[33] N. Kuntz, "Gestion eventos de ratón en OpenCV," 2009. http://dasl.mem.drexel.edu/~noahKuntz/openCVTut4.html.

[34] D. M. Escrivá, "Múltiples ejemplos con OpenCV," 2008. blog.damiles.com.

[35] "Documentación sobre OpenCV," 2008. www.emgu.com/wiki/files/2.0.0.0/html/.

[36] Multiple, "Lista de correo OpenCV ," 2008. OpenCV@yahoogroups.com.

[37] "Información sobre el comando ldd.," 2008. http://www.kotti.es/2006/01/ldd-dependencias-dinamicas-de-binarios/.

[38] "Información más completa sobre el comando ldd.," 2008. http://www.kotti.es/2006/01/ldd-dependencias-dinamicas-de-binarios/.

[39] A. Sensada, "Informacin sobre los makefiles.," 2009. http://es.debugmodeon.com/articulo/compilar-en-c-y-hacer-makefiles.

[40] G. Póo-Caamao, "Búsqueda y comparativa con patrones o plantillas.," 2002. http://calcifer.org/documentos/make/makefile.html.

[41] F. J. G. Pealvo, "Información sobre la herencia de clases.," 2002. http://zarza.usal.es/~fgarcia/docencia/poo/02-03/P5.pdf.

[42] "Método específico Contorno de OpenCV.," 2006. http:/http://opencv.willowgarage.com/documentation/structural_analysis_and_shape_descriptors.html#pointpolygontest/zarza.usal.es/~fgarcia/docencia/poo/02-03/P5.pdf.

[43] "Información sobre la estructura de datos Vector.," 2006. http://www.cplusplus.com/reference/stl/vector/.

[44] "Información sobre el dispositivo Meshlium.." http://www.libelium.com/documentation/mesh_extreme/meshlium-datasheet_esp.pdf.

[45] "Tienda online de cámaras Ip inalámbricas 1.." http://www.wlanmall.com/ip-video-surveillance/ip-cameras/.

[46] "Tienda online de cámaras Ip inalámbricas 2.." http://www.ipcamerasupply.com/wireless-ip-cameras.

[47] "Tienda online de cámaras Ip inalámbricas 3.." http://www.brickhousesecurity.com/wireless-network-cameras.html.

[48] "Tienda online de cámaras Ip inalámbricas 4.." http://www.kintronics.com/neteye/wirelessipcameras.htm.

[49] "Tienda online de cámaras Ip inalámbricas 5.." http://www.planetsecurityusa.com/ps-txcam58i.

[50] "Tienda online española sobre vigiliancia y cámaras IP.." http://www.seguridadplus.com.

[51] "Marca Scati Labs - Cámaras IP." http://www.scati.com.

[52] "Marca Mobotix - Cámaras IP." http://www.mobotix.com/esl_ES/.

[53] "Marca Vivotek - Cámaras IP." http://www.vivotek.com.

[54] "Marca MagoDSP - Cámaras IP." http://www.magodsp.com.

[55] "Onvif -¿coalición de empresas para crear un estándar de comunicaciones para sistemas de seguridad." http://www.onvif.org.

[56] "PSIA: Physical Security INTEROPERABILITY ALLIANCE." http://www.psialliance.org/.