UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

I.T. TELECOMUNICACIÓN (SONIDO E IMAGEN)

"DESARROLLO DE SOFTWARE DE RECONOCIMIENTO DE MATRÍCULAS DE COCHE"

TRABAJO FINAL DE CARRERA

Autor/es:
Arnau Campos Albuixech

Director/es:
José Ignacio Herranz Herruzo

GANDIA, 2014

ÍNDICE

1. **Introducción 5**
    1. Objetivos 6
    2. Organización de la memoria del proyecto 6

2. **Fundamentos teóricos 7**
    1. Introducción a la visión artificial 7
    2. Introducción a los sistemas OCR 8
    3. Sistemas OCR 9
    4. Que es un sistema ANPR?? 10
    5. Fundamentos teóricos de los ANPR 11
        1. Descripción del sistema de captación de imágenes 11
        2. Algoritmos del ANPR 13
            1. Localización de la matrícula 13
            2. Corrección de inclinación de la matrícula..... 14
            3. Segmentación de los caracteres 14
            4. Reconocimiento óptico de caracteres 15
        3. Problemas típicos de estos sistemas 15

3. **Herramientas utilizadas..... 17**

1. Lenguaje de programación: C++ 17
2. Bibliotecas de software: OpenCV 18
3. Eclipse..... 20
4. Qt Creator 21

4. **Descripción del software 22**

1. Interfaz de usuario 22
2. Funcionamiento 23
3. Función sacar matrícula 25
    1. Cargar imagen 25
    2. Detector matrícula 25
    3. Detector de números 28
    4. Determinar que valor es cada carácter 28
    5. Resumen de la función 30

5. **Aplicación en casos reales 31**

6. **Conclusiones 36**

7. **Bibliografía 37**

1

INTRODUCCIÓN

Los vehículos están identificados por sus números de matrícula, los cuales son fácilmente legibles para los humanos pero no para las máquinas. Para las máquinas, un número de matrícula de una placa es simplemente un conjunto de números repartidos por una matriz, los cuales representan una región de una imagen, con una intensidad y luminosidad determinadas. Debido a esto, es necesario diseñar un sistema matemático robusto capaz de percibir y extraer lo que deseamos de la imagen capturada.

Estas funciones o diseños matemáticos están implementados en lo que se llaman en inglés ”ANPR Systems” (Sistemas de Reconocimiento Automático de Placas de Matrícula) y significan una transformación entre el entorno real que se percibe y los sistemas de información que necesitamos para guardar y manejar toda esa información.

El diseño de estos sistemas es uno de de los campos de investigación en áreas como la inteligencia artificial, la visión por computador, el reconocimiento de patrones y las redes neuronales.

Estos sistemas de reconocimiento automático de placas de matrículas es un tema de indudable interés comercial con numerosas aplicaciones como el control de aparcamientos, acceso a instalaciones, tarificación de peajes, cálculo de la velocidad media entre puntos de una carretera, etc.

Los métodos de reconocimiento toman una imagen fija (o una secuencia de ellas), localizan en ella la placa que corresponde a la matrícula, y proceden a la extracción y reconocimiento de los caracteres que contiene.

1.1. Objetivos

En el presente trabajo final de carrera se pretende desarrollar un software, utilizando un sistema de visión artificial, que permita detectar automáticamente matrículas a partir de imágenes de coches y extraer el numero con un formato de texto legible por el ordenador. El trabajo se dividirá en dos etapas:

- La primera etapa de adaptación con C++, y en concreto con la librería de OpenCV, en el entorno de eclipse kepler (Versión 4.3). Una vez familiarizados con las herramientas se elaborara un primer programa funcional.
- En la segunda etapa con el Qt creator se elaborara una pequeña interfaz gráfica que permita realizar las funciones básicas del programa: cargar imagen, detectar matrícula y números y devolver en formato texto la matrícula.

Siendo así el resultado final del proyecto un pequeño programa que funcione correctamente y con el mínimo error posible.

1.2. Organización de la memoria del proyecto

Esta memoria estará estructurada en diversos apartados. En la primera parte de la memoria nos centraremos en los fundamentos teóricos en los que se basará este proyecto, presentando en líneas generales que es la visión artificial y profundizando en los sistemas OCR, más concretamente en los sistemas ANPR que son los que nos interesan. A continuación, realizaremos una pequeña explicación de las herramientas que he utilizado para realizar el proyecto como son el lenguaje de programación y el entorno de desarrollo.

Siguiendo el orden de la memoria entraremos en la descripción de mi software, mostrando como es la interfaz gráfica elegida, como funciona y una pequeña explicación del programa por dentro.

Por último, solo quedara mostrar una galería capturas de pantalla con ejemplos reales, en la que se mostraran la efectividad y las limitaciones. Y las conclusiones finales en la que daremos cuenta de todos los posibles errores.

2

FUNDAMENTOS TEÓRICOS

En esta sección explicare los fundamentos teóricos que fundamentan el proyecto con la mayor brevedad posible pero intentando no omitir ningún punto importante. Pasando desde lo más básico de que es la visión artificial a profundizar en cómo debería ser el montaje de un sistema de reconocimiento automático de matrículas y los algoritmos del software necesarios.

2.1 Introducción a la visión artificial

La visión artificial, es un subcampo de la inteligencia artificial, el propósito de la cual es programar un ordenador para que "entienda" una escena o las características de una imagen.

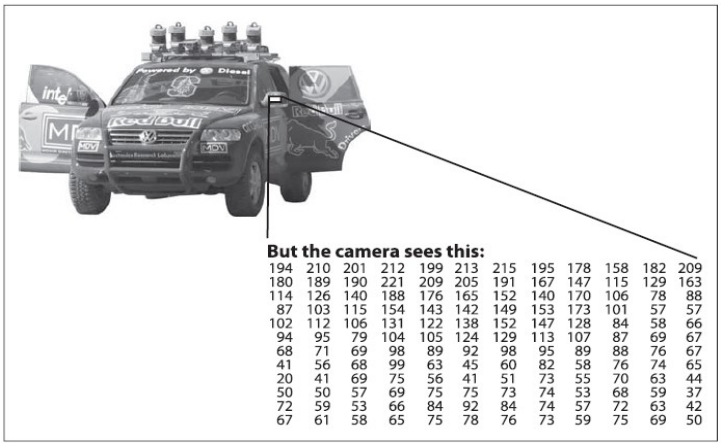


Figura 1: Para un ordenador esta imagen es solo una cuadricula de números

Dentro de la visión artificial podemos encontrar distintas motivaciones incluyendo algunas como:

- La detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes. Por ejemplo, caras humanas, o el caso que nos atañe matrículas en los coches.
- La evaluación de los resultados. Por ejemplo, segmentación, registro.
- El registro de diferentes imágenes de una misma escena u objeto, es decir, hacer concordar un mismo objeto en diversas imágenes.
- El seguimiento de un objeto en una secuencia de imágenes.
- El mapeo de una escena para generar un modelo tridimensional de la escena; este modelo podría ser usado por un robot para navegar por la escena.
- La estimación de las posturas tridimensionales de humanos.
- La búsqueda de imágenes digitales por su contenido.

Estos objetivos se consiguen por medio de reconocimiento de patrones, aprendizaje estadístico, geometría de proyección, procesamiento de imágenes, teoría de grafos y otros campos. La visión artificial cognitiva está muy relacionada con la psicología cognitiva y la computación biológica.

2.2. Introducción al reconocimiento óptico de caracteres (OCR)

El reconocimiento óptico de caracteres, abreviado a partir de ahora como OCR por sus siglas en ingles (*Optical Character Recognition*), es un proceso computarizado de análisis dirigido a la digitalización de textos. Esta tecnología permite convertir una imagen digital, ya provenga de un escaneo de una hoja de texto como de una fotografía, la cual por si sola no es más que una matriz de puntos, en datos manejables para un ordenador. Con estos datos se puede interaccionar en un procesador de textos o similar.

2.3. Sistemas OCR

El proceso para realizar el OCR lo podemos diferenciar en 3 pasos generales:

1. **El Pre-procesado**: en este punto se trabajara en la imagen de entrada recibida según sus características hasta obtener como resultante una imagen binaria de cada carácter individualmente. En este punto se realizará, en caso de ser necesario, la corrección de imágenes torcidas, eliminación de manchas de la imagen con un suavizado, binarización de la imagen, detección de la posición de los caracteres, segmentación y normalización de relación de aspecto.

2. **Reconocimiento de caracteres**: en este punto debemos averiguar de qué carácter se trata y para realizar esto podemos diferenciar estos modos de llevarlo a cabo:

   - Pattern matching: Este procedimiento implica la comparación píxel a píxel entre la imagen obtenida del carácter en el paso anterior con una imagen patrón almacenada anteriormente. Dará como resultado el carácter con más coincidencia de píxeles. Esta técnica funciona con textos escritos a máquina con fuentes definidas, para textos escritos a mano y aplicaciones que tengan que detectar distintas fuentes dará mal resultado.

   - Feature detection: Descompone la imagen de el carácter en una lista de características, el conjunto de las cuales nos permite diferenciar de que se trata. Utilizando este método se pueden conseguir aplicaciones más complejas, lo que permita aplicaciones que reconocieran textos escritos a mano o detección de caracteres más degradados.

3. **El Pos-procesado**: Dependiendo del sistema puede hacer falta un sistema de pos-procesado según nuestras necesidades. Normalmente lo que se hace es acotar al máximo las opciones de salida, por ejemplo haciendo una lista de palabras posibles. Si se realiza este apartado correctamente se mejorara mucho el resultado final.

Son muchas las aplicaciones que implementan algoritmos de reconocimiento óptico de caracteres, i cada día son más las aplicaciones que las utilizan ya sea para mejorar su

rendimiento o como base. Estas son algunas de las aplicaciones más destacables que utilizan algoritmos OCR:

- Reconocimiento de texto manuscrito: Digitalización de documentos.
- Indexación en bases de datos: El gran aumento de información publicada que ha tenido lugar en los últimos años, provoca que introducir los meta-datos manualmente a todas las imágenes resulte imposible. Por eso ahora podemos generar la información de los meta-datos de imágenes almacenadas en bases de datos mediante el texto que aparezca en esta.
- Reconocimiento de datos estructurados: Se usa para digitalizar de forma masiva grandes cantidades de documentos estructurados o semiestructurados (facturas, nóminas, albaranes, pólizas, justificantes bancarios, etc.), catalogando automáticamente los documentos con los meta-datos obtenidos y archivando-los en formato digital de forma indexada para facilitar su posterior búsqueda. Tiene el inconveniente de que es necesario diseñar previamente las plantillas, pero con una buena configuración se ahorra mucho tiempo en el proceso de digitalización.
- Reconocimiento de matrículas: Este es el sistema que nos interesa en este proyecto y que veremos a partir de ahora con profundidad.

2.4. Que es un sistema ANPR??

Para procesar, clasificar o analizar datos todo el mundo piensa en usar los ordenadores, y si los datos están ya en el ordenador, la mayor parte de estas tareas son fáciles de realizar. Para cualquier sistema informático que trate con vehículos el dato más importante es el numero de matrícula, ahí es donde entra el sistema de reconocimiento automático de matrículas también llamado sistema ANPR (*Automatic number plate recognition* en inglés), que es una tecnología que permite la vigilancia en masa a través del uso de un software de reconocimiento óptico de caracteres para escanear imágenes y leer las matrículas de los vehículos. Este sistema sustituye la tarea de mecanografiar manualmente el número de la placa del vehículo en el sistema informático, evitando así que una supervisión humana.

Estas técnicas son utilizadas por las diversas fuerzas de policía y como método de recaudación electrónica de peaje en las autopistas de pago o para vigilar la entrada y salida de aparcamientos privados, entre otros ejemplos.

2.5. Fundamentos teóricos de los ANPR

Los sistemas automáticos de reconocimiento de matrículas son un conjunto especial de componentes de hardware y software, que procesan una señal de entrada que contiene una representación gráfica, tanto imágenes estáticas como secuencias de video y reconocen los caracteres de la placa. Transformando la información de esa imagen en un texto que el ordenador sea capaz de leer.

2.5.1. Descripción del sistema de captación de imágenes

La parte del sistema de captación de imágenes forma parte de hardware del sistema ANPR, que consiste típicamente en una unidad de cámara, el procesador de imagen, el disparador de la cámara y la comunicación y almacenamiento.

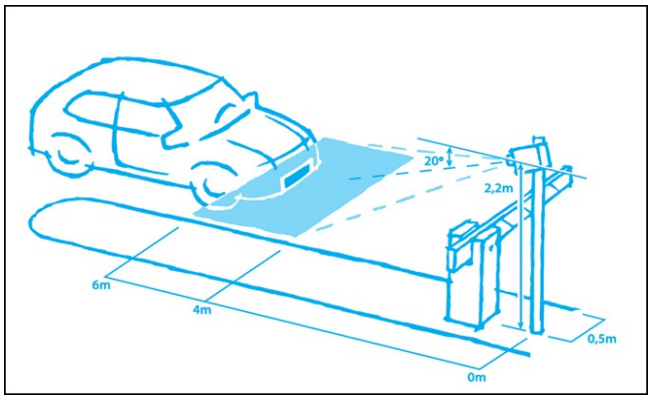


Figura 2: Esquema de colocación de hardware a la entrada de un aparcamiento

El hardware del disparador lo controla físicamente un sensor directamente instalado en un carril. Cada vez que el sensor detecta un vehículo en una distancia adecuada de la cámara, se activa un mecanismo de reconocimiento. Una alternativa a esta solución es la implementación de un software de detección de coches entrantes, o la detección por software del vehículo con el procesamiento continuo de la señal de video, aunque esto

consumirá más recursos del sistema, no hará necesario equipo de hardware adicional como el disparador.

El procesador de imagen, es la parte donde se ejecuta el software, reconoce las instantáneas estáticas captadas por la cámara, y devuelve una representación de texto de la placa de matrícula detectada. Unidades ANPR pueden tener procesadores de imagen dedicados propios, o pueden enviar los datos capturados a una unidad central de procesamiento para su posterior procesamiento.

Debido a que uno de los campos de aplicación es un uso en la carretera para el control de la velocidad, es necesario en estos casos utilizar una cámara especial con un tiempo de obturación extremadamente corto. Ya que de lo contrario, la calidad de las instantáneas capturadas se degradaría por un efecto de desenfoque de movimiento no deseado causado por el movimiento del vehículo. Por ejemplo, el uso de la cámara estándar con el obturador de 1/100 segundos para capturar un vehículo con velocidad de 80 km/h causará un desenfoque de movimiento significativo(el coche se habrá desplazado 0,22m con el obturador abierto). lo cual dificulta en gran medida la capacidad de reconocimiento.

Figura 3: Ejemplo de distorsión por poca velocidad de obturación

Para asegurar que los resultados no varíen independientemente de las condiciones de luz no se suelen utilizar cámaras normales para la captura de instantáneas en la oscuridad o de la noche, debido a que operan en el espectro de luz visible, estos sistemas a menudo se basan en cámaras que operan en una banda infrarroja del espectro de luz. El uso de la cámara de infrarrojos combinado con una iluminación de infrarrojos es lo

mejor para lograr este objetivo, ya que baja iluminación, las placas, que están hechas de materiales reflexivos, destacan mucho más que el resto de la imagen. Este hecho hará la detección de matrículas mucho más fácil.

2.5.2. Algoritmos del ANPR

Ahora explicare los algoritmos principales que harán que, una vez recibida la imagen, el software detecte la matrícula. La complejidad de cada una de estas subdivisiones del programa determina la precisión del sistema.

1. Localización de la matrícula

El primer paso es una detección de un área de número de placa. Esta problema incluye algoritmos que son capaces de detectar un área rectangular de la placa de matrícula en una imagen original. Las maquinas no entienden la definición de matrícula que podríamos extraer de un diccionario, por eso hay una necesidad de encontrar una definición alternativa de una placa de número basado en descriptores que ser comprensible para las máquinas.

Una definición adecuada para definir la matrícula a la maquina seria "área rectangular con una mayor aparición de bordes horizontales y verticales". Ya que la alta densidad de los bordes horizontales y verticales en un área pequeña en la mayoría de los casos son causados por los caracteres de una matrícula, pero no en todos los casos. En este proceso a veces podemos detectar un área equivocada que no corresponde a un número de matrícula. Por a esto, a menudo detectaremos varios candidatos a ser la matrícula por este algoritmo, y luego escoger la mejor opción por un análisis de las características de cada candidato, como por ejemplo las proporciones que debería tener una matrícula.

2. Corrección de inclinación de la matrícula

Para la corrección de la inclinación utilizaremos la transformada de Hough, normalmente esta transformada es utilizada para detectar líneas, pero aquí la utilizaremos para detectar el ángulo de la inclinación de la matrícula. Como vemos en el ejemplo de la imagen 4 en la que se ve la imagen antes y después de la transformación. En la imagen de la

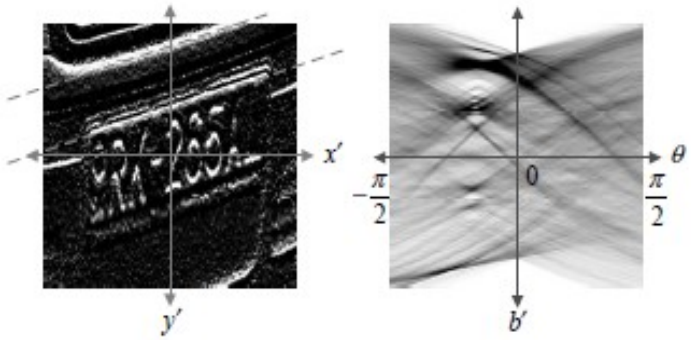


Figura 4: Antes y después de la transformada de Hough a una matrícula

transformada se observa que las manchas más oscuras es donde se encuentran las líneas más largas. Así que cogemos el punto máximo de la transformada y suponemos que el ángulo de esta línea determina el ángulo general por ser la más larga. En cuanto sepamos el ángulo nos bastara con utilizar algún algoritmo de corrección de ángulos para tener nuestra matrícula dentro de un rectángulo perfecto y poder recortarla del resto de la imagen.

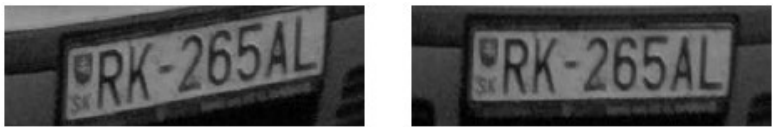


Figura 5: Antes y después de la corrección de ángulo

3. Segmentación de los caracteres

Para la segmentación de caracteres asumiremos que los procesos anteriores han funcionado y que como principio de este paso tenemos la imagen de la matrícula en blanco y negro solo.

La segmentación es uno de los procesos más importantes si la segmentación falla, un carácter puede ser in-apropiadamente dividido en dos, o dos caracteres pueden ser in-apropiadamente fusionados. Existen procedimientos más complejos, pero vamos a

explicar la extracción de los caracteres utilizando la proyección horizontal por su sencillez, facilidad de implementar y buenos resultados.

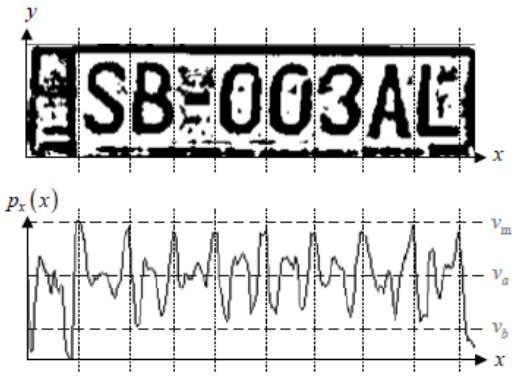


Figura 6: Matrícula umbralizada arriba y su proyección horizontal abajo

Primero umbralizaremos la matrícula y sacaremos su proyección horizontal como en la imagen 6. En esta proyección se puede observar a simple vista que en los mínimos hay caracteres y en los máximos es por donde deberíamos cortar, sin olvidarnos de la fisonomía de la matrícula(acordándonos que delante va la letra del país y que contiene 4 números y 3 letras)

4. Reconocimiento óptico de caracteres

Este es un sistema OCR como los vistos en el punto 2.3, en el cual el pre-procesado es los tres pasos realizados anteriormente, y utilizando alguno de los sistemas ahí mencionados obtendremos como resultado la matrícula en texto para ser tratada por el ordenador.

2.5.3. Problemas típicos de estos sistemas

Existen ciertos problemas comunes en los ANPR en los que se tendrá que tener especial atención para intentar que se produzcan el mínimo posible en los sistemas. Entre estos problemas se incluyen:

- Resolución de imagen pobre, a menudo porque la matrícula está demasiado lejos, aunque a menudo es resultado del uso de una cámara de baja calidad.

- Imágenes desenfocadas, en particular desenfoque de movimiento, ocurre muy a menudo en unidades móviles.
- Iluminación pobre y bajo contraste debido a reflexión o sombras.
- Un objeto que oscurece parte de la matrícula, a menudo una barra del remolque, o suciedad en la matrícula.
- Técnicas de evasión que puedan utilizar los conductores.

Por lo tanto, según lo complejo y efectivo de nuestro software de detección deberemos tener mejor instalación de hardware o mejor controlado el entorno donde lo instalemos (iluminación constante, que el coche se tenga que detener, etc.) y es muy importante que estos tres puntos estén compensados para que funcione correctamente.

3

HERRAMIENTAS UTILIZADAS

En esta sección damos paso a explicar las herramientas utilizadas para realizar este proyecto. Estas han sido elegidas debido a la simplicidad y comodidad que nos proporcionan a la hora de realizar el sistema que quería construir.

3.1. Lenguaje de programación: C++

Para hablar de C++ antes es necesario explicar que un lenguaje de programación. Un lenguaje de programación es una herramienta que nos permite comunicarnos e instruir a la computadora para que realice una tarea específica. Cada lenguaje de programación posee una sintaxis y un léxico particular, es decir, forma de escribirse que es diferente en cada uno por la forma que fue creado y por la forma que trabaja su compilador para revisar, acomodar y reservar el mismo programa en memoria.

La definición oficial del lenguaje nos dice que C++ es un lenguaje de propósito general basado en el C, al que se han añadido nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones inline, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería (en realidad la librería Estándar C es un subconjunto de la librería C++).

Entre las características principales de este lenguaje, por las que ha cosechado un notable éxito en muchas aplicaciones, sistemas operativos, compiladores e intérpretes

que han sido escritos en C++ (el propio Windows y Java por ejemplo), encontraremos las siguientes:

- **Programación orientada a objetos**: Aunque se suele decir que es un lenguaje híbrido, ya que permite la programación estructurada. La posibilidad de orientar la programación a objetos permite al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real.
- **Velocidad**: Los compiladores de C++ generan código nativo con un alto grado de optimización en memoria y velocidad, lo que lo convierte en uno de los lenguajes más eficientes.
- **Programación modular**: Un cuerpo de aplicación en C puede estar hecho con varios ficheros de código fuente que son compilados por separado y después unidos. Además, esta característica permite unir código en C con código producido en otros lenguajes de programación como Ensamblador o el propio C.
- **Brevedad**: El código escrito en C++ es muy corto en comparación con otros lenguajes, sobre todo porque en este lenguaje es preferible el uso de caracteres especiales que las palabras clave.

3.2. Bibliotecas de software: OpenCV

Una biblioteca de software consiste en un conjunto de funciones descritas en un lenguaje de programación en específico que permiten agilizar el trabajo y brindar una interfaz de desarrollo al programador o API (*Application Program Interface*). En el tema de visión por computador existen distintas de estas bibliotecas, en mi caso la utilizada es OpenCV.

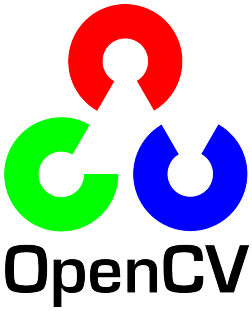


Figura 7:
Logotipo OpenCV

OpenCV es un conjunto de bibliotecas de código abierto u Open Source bajo licencia BSD (Es una licencia de software libre permisiva en comparación con otras estando muy cercana al dominio público, permite el uso del código fuente en software no libre) desarrolladas en un principio por Intel, disponibles desde 1999. La biblioteca está escrita en C y C++ y se pueden ejecutar desde diversos sistemas operativos como GNU/Linux, Windows y Mac OS X, también existe un desarrollo de interfaces para otras lenguas como Python, Ruby, Matlab, etc. OpenCV fue diseñado ofreciendo un código diseñado muy eficientemente y con un fuerte enfoque a aplicaciones capaces de ejecutarse en tiempo real. Puede tomar ventaja de los procesadores multi-núcleo.

Uno de los objetivos de OpenCV es proporcionar una infraestructura de visión por ordenador fácil de usar que ayude a las personas a construir aplicaciones bastante sofisticadas rápidamente. Para eso, la biblioteca contiene más de 500 funciones que abarcan muchas áreas. Es usada ampliamente en imágenes médicas, la seguridad, la calibración de la cámara, la visión estéreo y la robótica.

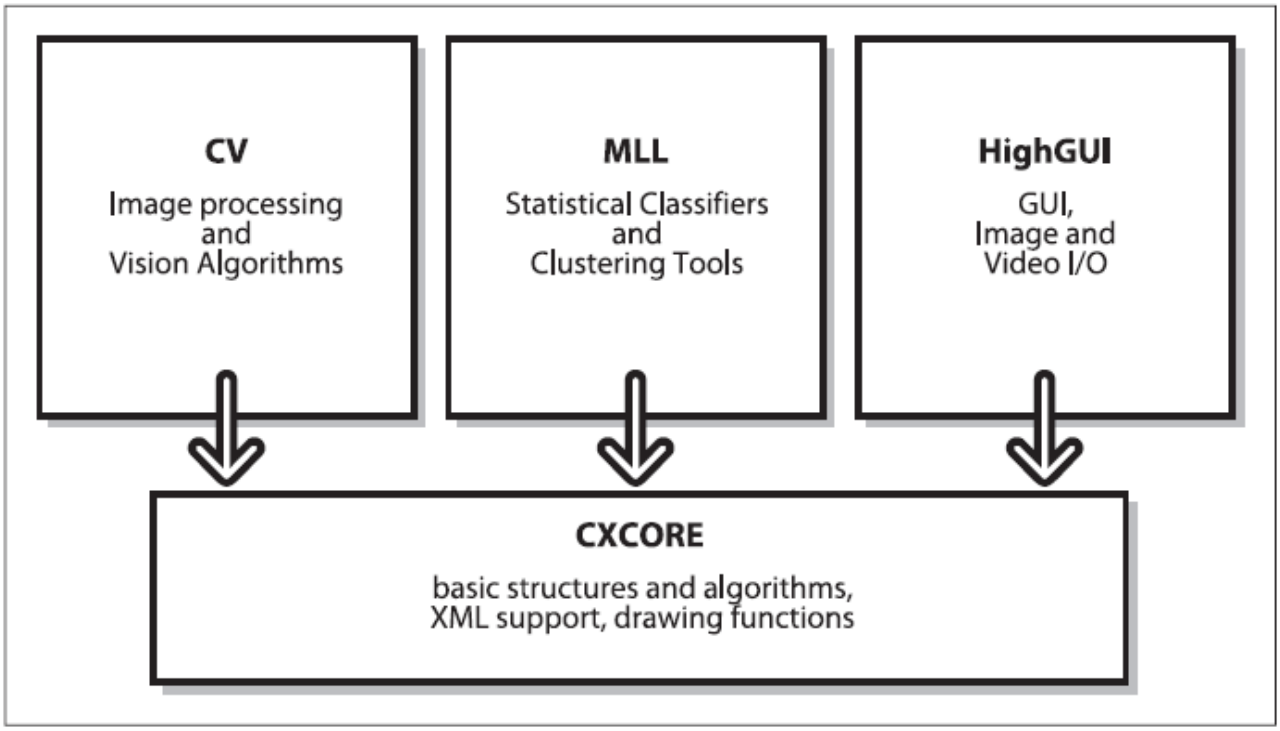


Figura 8: Estructura básica del Open CV

3.3. Eclipse

Eclipse es un entorno de desarrollo integrado (IDE de *Integrated Development Environmen*t) eso quiere decir que es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

Eclipse Kepler (versión 4.3) ha sido mi entorno de desarrollo integrado elegida porque ya la había utilizado anteriormente en las asignaturas de programación de la carrera y el entorno me resultaba más familiar.

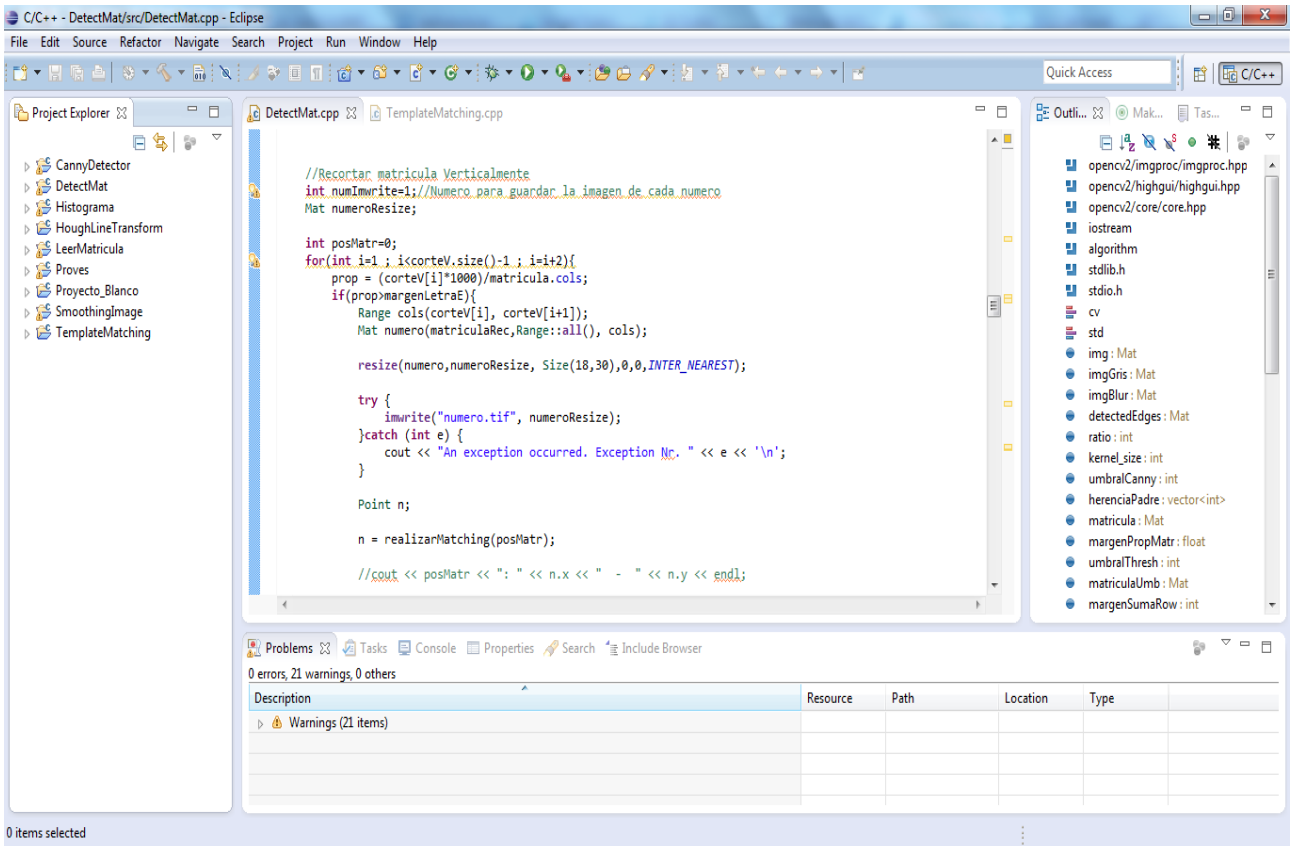


Figura 9: Entorno de desarrollo de eclipse

3.4. Qt Creator

Qt es un entorno de trabajo open source con licencia GPL concebido para el desarrollo de aplicaciones e interfaces multiplataforma. El paquete Qt integra herramientas de desarrollo y soporte, así como librerías de clases auxiliares a Qt. También incluye el compilador Gcc MinGW.

Las librerías que aporta básicamente son un conjunto de clases en C++, que nos permitirán utilizar elementos básicos como Listas, Rectángulos, etc. de una manera rápida y eficaz. Pero el mayor motivo para elegir Qt para diseñar la interfaz trafica de nuestro programa es la sencillez que aporta.

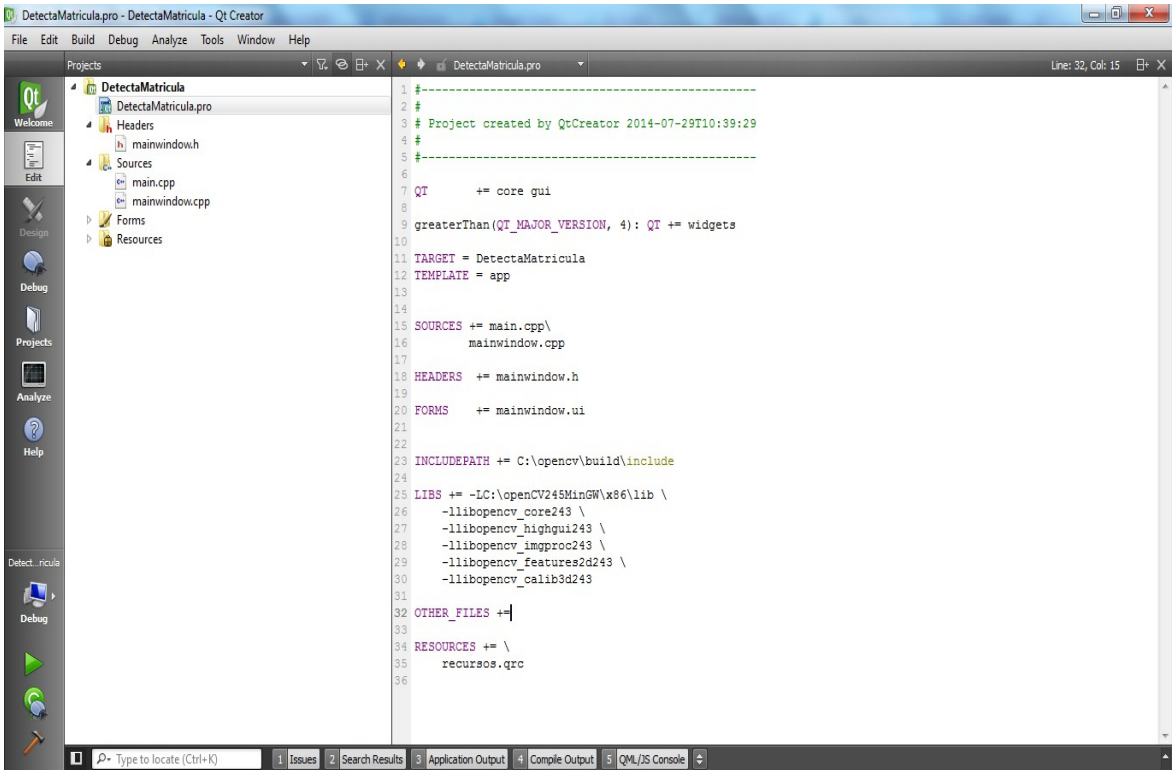


Ilustración 10: Entorno de desarrollo de Qt Creator

4

DESCRIPCIÓN DEL SOFTWARE

En esta sección damos paso a mostrar el programa hecho tanto gráficamente (la interfaz de usuario elegida) como el funcionamiento interno.

4.1. Interfaz de usuario

La interfaz de usuario esta creada con el programa Qt creator. He optado por la sencillez para simplificar al máximo el uso de este programa. En la ilustración se muestra la interfaz y a continuación explicare qué función tiene cada apartado.

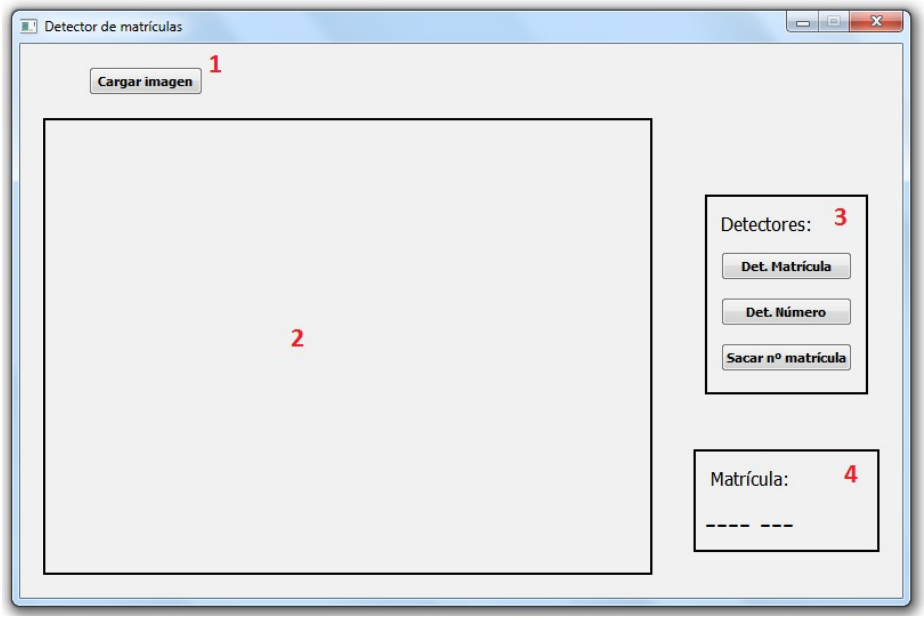


Figura 11: Interfaz de mi programa

En cuanto a los apartados de la interfaz, podemos distinguir 4 numerados con rojo en la captura de pantalla de la ilustración 11. En el 1 podemos observar el botón “Cargar imagen”, que como su propio nombre indica abrirá una ventana donde podremos abrir entre nuestros archivos una imagen a la que se aplicará posteriormente la detección, y la mostrará en el rectángulo 2. Que cumple la función de mostrar la imagen en todo momento.

En el rectángulo 3 encontramos 3 botones en los que se podrá elegir si solo queremos detectar la matrícula, los números o si queremos hacer el proceso completo, en el cual nos mostrara la matrícula obtenida en formato texto dentro del rectángulo 4.

Para entender mejor las posibilidades y ver-lo de forma más gráfica, en el siguiente punto muestro capturas de las tres funcionalidades de detección.

4.2. Funcionamiento

El programa tiene tres funciones, que en las siguientes tres imágenes se mostraran los distintos resultados al pulsar los botones.

1. Detectar matrícula: Muestra en la imagen un rectángulo rojo alrededor de la matrícula.

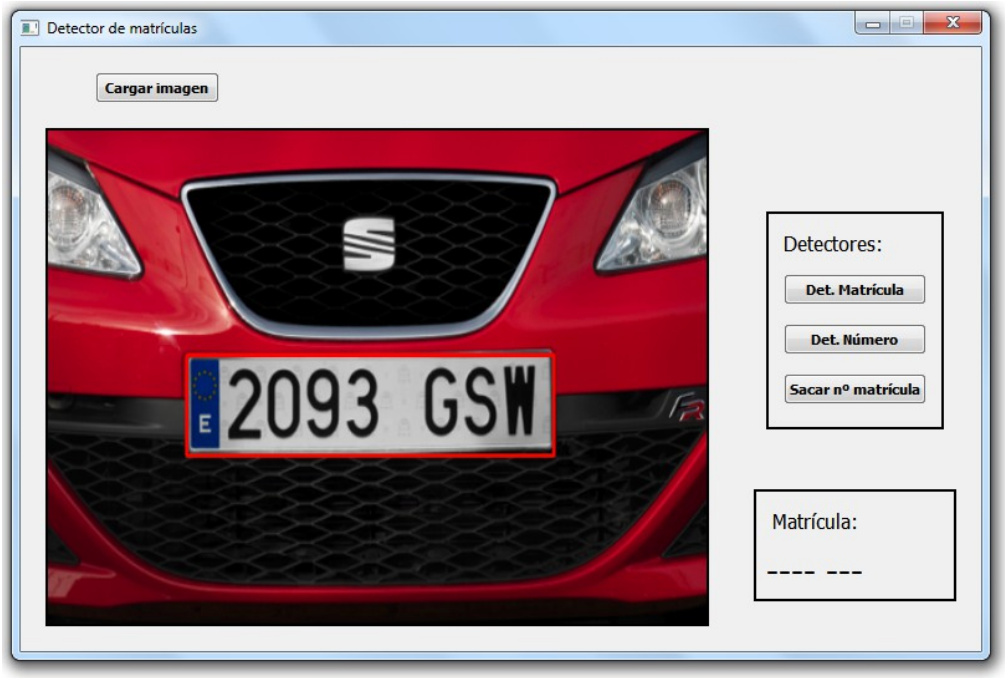


Figura 12: Botón Det. Matrícula pulsado

2. Detectar número: Muestra en la imagen un rectángulo azul alrededor de cada carácter detectado.

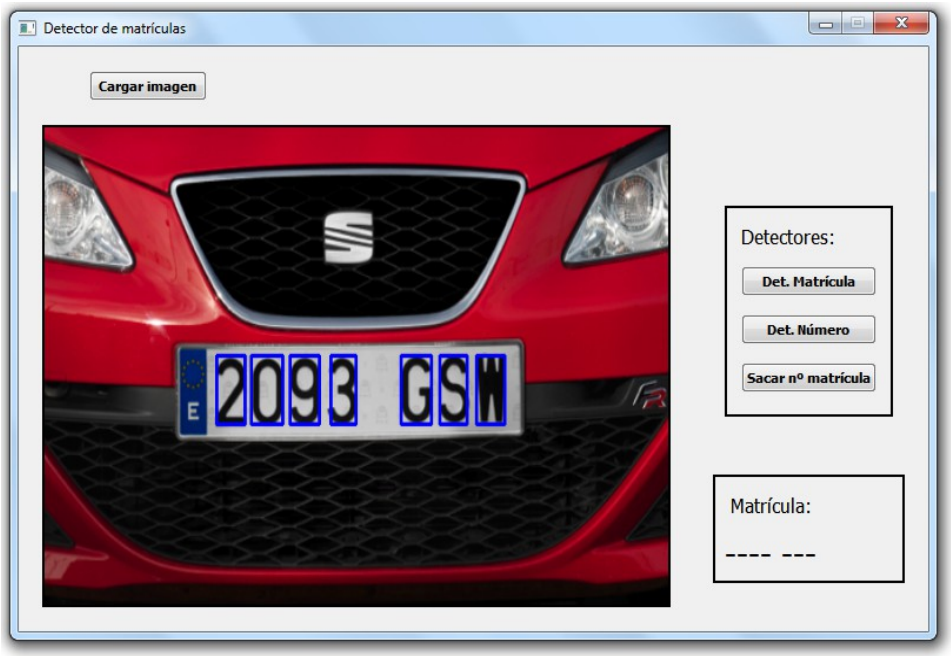


Figura 13: Botón Det. Número pulsado

3. Sacar numero matrícula: Realiza las dos funciones anteriores y además muestra por pantalla el numero en texto.

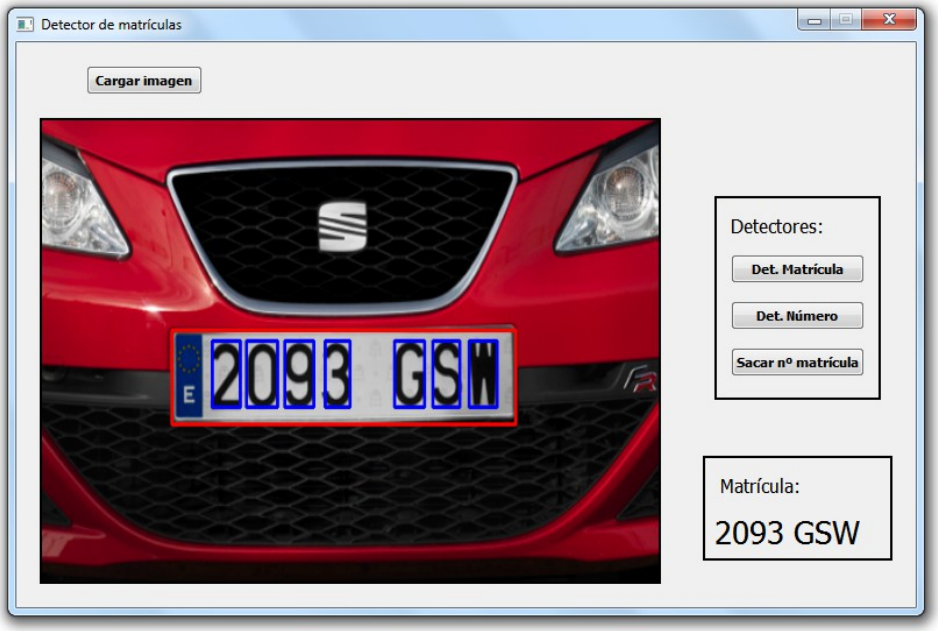


Figura 14: Botón Sacar nº matícula pulsado

4.3. Función sacar matrícula

En este apartado voy a explicar paso a paso el funcionamiento de de la función sacar matrícula que se ejecutará al pulsar el botón sacar matrícula, centrándonos sobre todo en la parte de openCV que es la que más nos interesa. Para explicar-lo, iré citando partes del código que aparecerán en cuadros seguidos de la explicación del código.

1. Cargar la imagen

Para explicar el funcionamiento, primero voy a explicar el tipo de datos nuevo que encontramos en este pedazo de código. Se trata de la clase *Mat*, es una de las estructuras que más utilizaremos, se emplea para operar con imágenes dentro del programa. Esencialmente es una matriz que contendrá los datos de una imagen, pero contiene información adicional que resultara de gran utilidad para tratar con imágenes.

Para cargar la imagen dentro de un *Mat* utilizaremos la función *imread*, donde la primera variable es la ruta de donde se encuentra la imagen y el segundo es tipo de carga, el tercer valor será el modo del que quieres cargar la imagen *CV_LOAD_IMAGE_COLOR* es para cargarla a color, pero puede contener otros valores como *CV_LOAD_IMAGE_GRAYSCALE* para blanco y negro.

```
Mat imagen = imread(String ruta, CV_LOAD_IMAGE_COLOR);
if(imagen.empty()){
        // Ejecuta que no se ha cargado la imagen
}
```

También he puesto *imagen.empty()* en el *if()* de después de la carga de la imagen porque es muy importante realizar un control de que la imagen haya cargado bien, ya que si no dará errores. *imagen.empty()* devolverá true en el caso de no esté cargada.

2. Detector matrícula

Para detectar la matrícula primero trabajaremos la imagen para que sea más sencillo. Primero la convertiremos a escala de grises con *cvtColor()*, donde *imagen* y *imgGris* son los *Mat* de entrada y salida (en el resto de funciones no explicaré las dos primeras

entradas a no ser que sea distinto a esté caso), y la tercera variable expresa el tipo de cambio de color, en esté caso de RGB a gris.

```
cvtColor( imagen, imgGris, CV_BGR2GRAY );
blur( imgGris, imgBlur, Size(3,3) );
Canny(imgBlur,imgCanny, umbralCanny, umbralCanny*3, 3);
```

Luego le pasaremos un filtro de reducción de ruido con un kernel de 3x3 con *blur()* y a la imagen resultante le aplicamos un detector de bordes Canny con un umbral (*umbralCanny*) de 130. Una vez completado este proceso tendremos una imagen binaria donde cada píxel estará definido como borde o no borde. Quedando una imagen binaria como la mostrada en imagen 15.

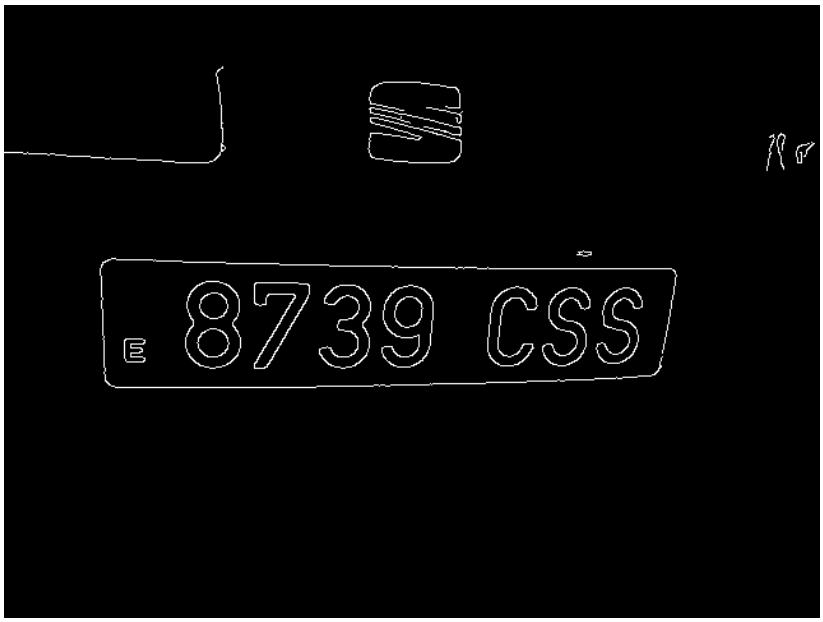


Figura 15: Imagen resultante del filtrado Canny

Ahora pasamos a la siguiente etapa, utilizaremos la función *findContours()*, de la cual, la primera variable es la imagen de entrada que debe ser binaria. La segunda variable, *contours* se trata de un vector de vectores de puntos, teniendo en cuenta que cada vector de puntos es una línea independiente, podríamos decir que es un vector de líneas independientes. La siguiente variable *hiterarchy* es un vector que guarda información de cada línea de *contours* sobre la relación con el resto de líneas. El resto de variables son opciones sobre el modo y método de hacer-lo, que tampoco nos interesa demasiado ahondar para entender el funcionamiento .

```
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

findContours( imgCanny, contours, hierarchy,
        CV_RETR_TREE,
        CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

 for( int i = 0; i < contours.size(); i++ ){
        vecRectangulos[i] =boundingRect(Mat(contours[i]) );
      }
```

El siguiente paso es crear un rectángulo para cada línea del vector *contours*, obteniendo un vector de rectángulos(*Rect*) *vecRectangulos* con la función *boundingRect()*, que crea un rectángulo que contenga todos los puntos del vector de puntos. *Rect*, clase implementada por openCV que incluye los datos de un rectángulo (4 *int* :coordenadas donde está y dimensiones de alto y ancho). Del cual diferenciaremos el que incluye la matrícula observando las características de cada rectángulo.

En este proyecto solo he tenido en cuenta dos variables la proporción de la matrícula

```
 for( int i = 0; i< vecRectangulos.size(); i++ ){
        float area = vecRectangulos[i].area();
        float prop = vecRectangulos[i].width /
vecRectangulos[i].height;

        if(area>areaImg*0.02){
           if(prop>propmatr-margenPropMatr &&
                        prop<propmatr+margenPropMatr){
              //Rectángulo bueno

           }//if Proporción
        }//if Area
      }//for
```

(ancho * alto) y que el área de la matricula sea superior a el 0,2 del área total, con esto eliminamos los rectángulos pequeños.

3. Detector de números

Como ahora sabemos que rectángulo es, recortaremos la matrícula de la imagen original en escala de grises y deberemos umbralizarar para realizar los siguientes pasos, en mi programa tengo el umbral a 130. Y aplicaremos las funciones *sumaCol()* y *sumaRow()*, funciones echas por mi que devolverán la proyección vertical y horizontal.

```
threshold(matrícula, matrícula,
        umbralThresh,255,THRESH_BINARY);
vector<float> vecHorizontal = sumaCol(matrícula);
vector<float> vecVertical = sumaRow(matrícula);
```

Sacaremos los máximos horizontales y verticales lo que nos indicara donde empiezan y termina cada numero, para saber por donde tenemos que recortar, obteniendo así rectangulos con caracteres individuales.

4. Determinar que valor es cada carácter

Utilizaremos Pattern matching para obtener los valores. Para ello a cada carácter lo debemos redimensionar para poder comparar con la plantilla. La redimensión se hace respetando la proporción del recorte gracias a *propX* para evitar deformaciones que lleven a errores.

```
int propX = (numero.cols*30)/numero.rows;
resize(numero, numeroResize, Size(propX,30), 0, 0,
        INTER_NEAREST);
```

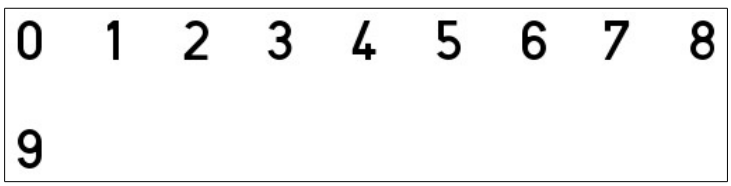


Figura 16: Plantilla con los números

El numero normalizado ahora deberá realizar el *matchTemplate()* siendo *imgSource* las plantillas de los números y letras(Imagen 16 y 16), que según la posición dentro de la matrícula se cargara una o otra para evitar confusiones entre números y letras como el 0 y la letra o. Y *templ* sera el numero a comparar. Después de este proceso, como resultado

tendremos un punto *Point matchLoc* que indicara el punto de la plantilla con máxima coincidencia con el carácter.

```
matchTemplate( imgSource, templ, result,
        CV_TM_SQDIFF_NORMED );
normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );

double minVal; double maxVal; Point minLoc; Point maxLoc;
minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc,
Mat() );

Point matchLoc = minLoc;
```

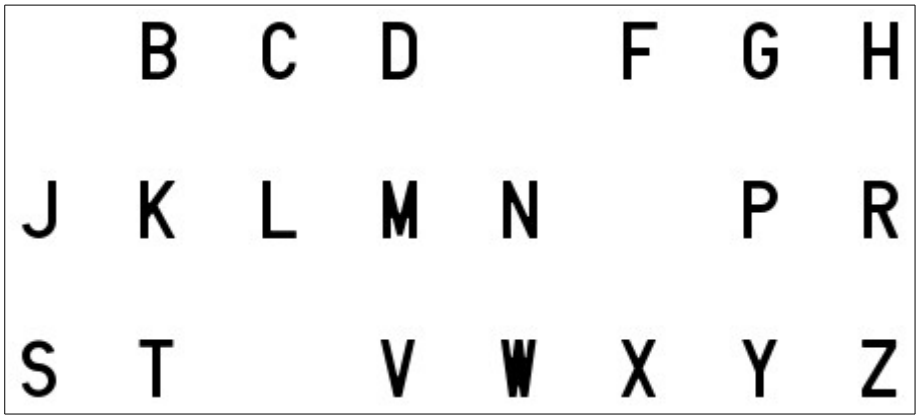


Figura 17: Plantilla con las letras

Para transformar este punto en el numero llamaremos a una función que he nombrado como *asignaValor()* en la que he ideado una especie de “tabla” encadenando if() como en el ejemplo de abajo que según la posición en la matrícula (Primero, segundo, etc.), y donde apuntan las coordenadas del punto nos permitirán sacar un *string* que contenga el numero de la matrícula.

```
if(pos<4){//NUMEROS
      if(valY<45){
         if(valX<55) matrFinal = matrFinal + "0";
         else if(valX<110) matrFinal = matrFinal + "1";
        .. ..
         else if(valX<395) matrFinal = matrFinal + "6";
         else if(valX<440) matrFinal = matrFinal + "7";
         else matrFinal = matrFinal + "8";
      }else matrFinal = matrFinal + "9";
```

5. Resumen de la función

Para terminar este punto, a modo resumen elaboraré una enumeración de los pasos que ejecuta esta función sin profundizar para obtener un visión más global que nos permita entender mejor el funcionamiento. Pasos:

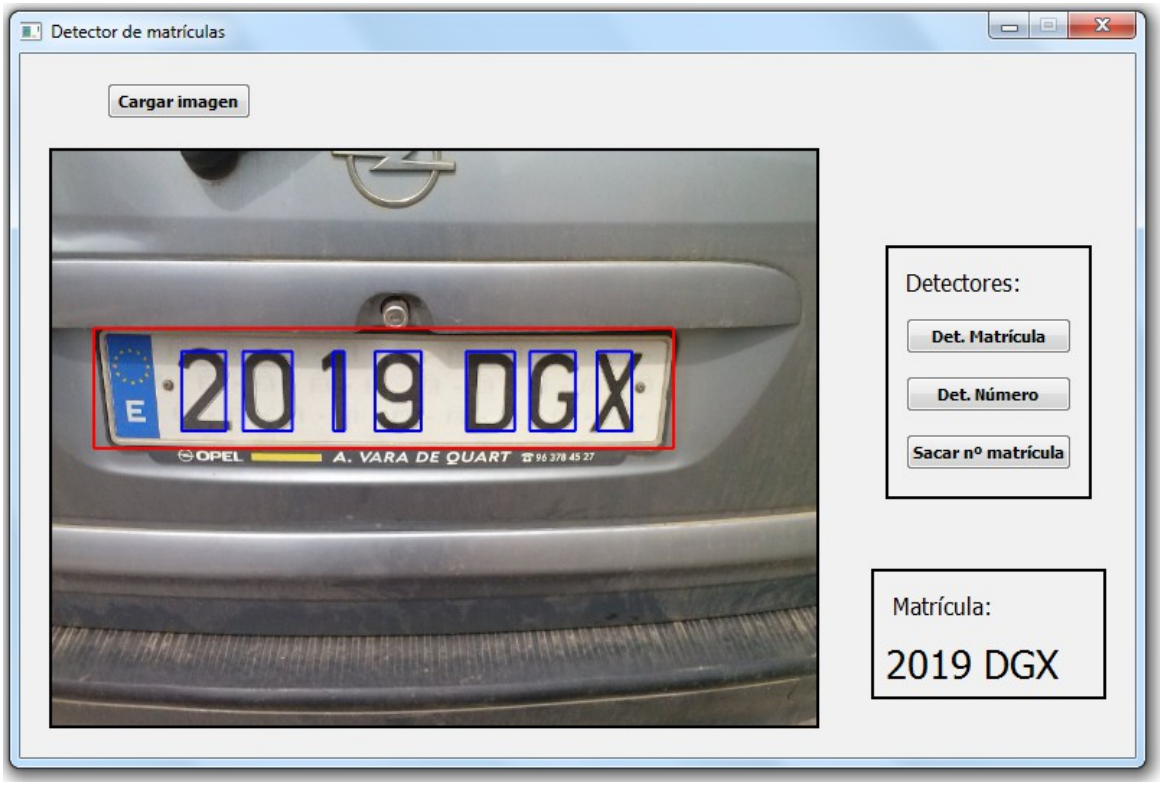
1. Carga imagen y la pasa a escala de grises
2. Filtrado de ruido
3. Detector de líneas Canny
4. Detector de matriculas
    1. Detección de contornos mediante *findContours()*.
    2. Crear vector de rectángulos de los contornos
    3. Elección del rectángulo matrícula
    4. Recortar matrícula
5. Detector de números
    1. Umbralizar
    2. Proyección vertical y horizontales
    3. Obtener máximos de las proyecciones
    4. Recortar los números
6. Pattern Matching
7. Asignación del valor

5

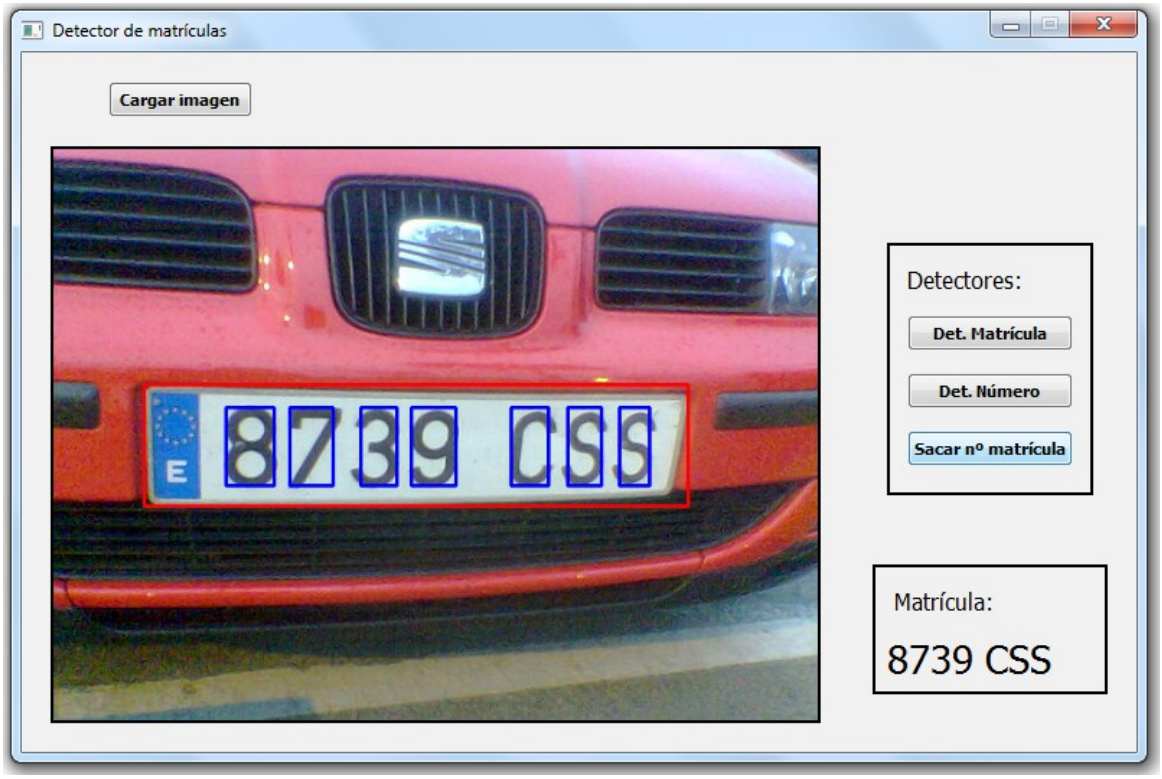
APLICACIÓN EN CASOS REALES

Aquí mostraré capturas de casos reales para comprobar la eficacia y las limitaciones de la aplicación. Al finar aremos una reflexión de los resultados.

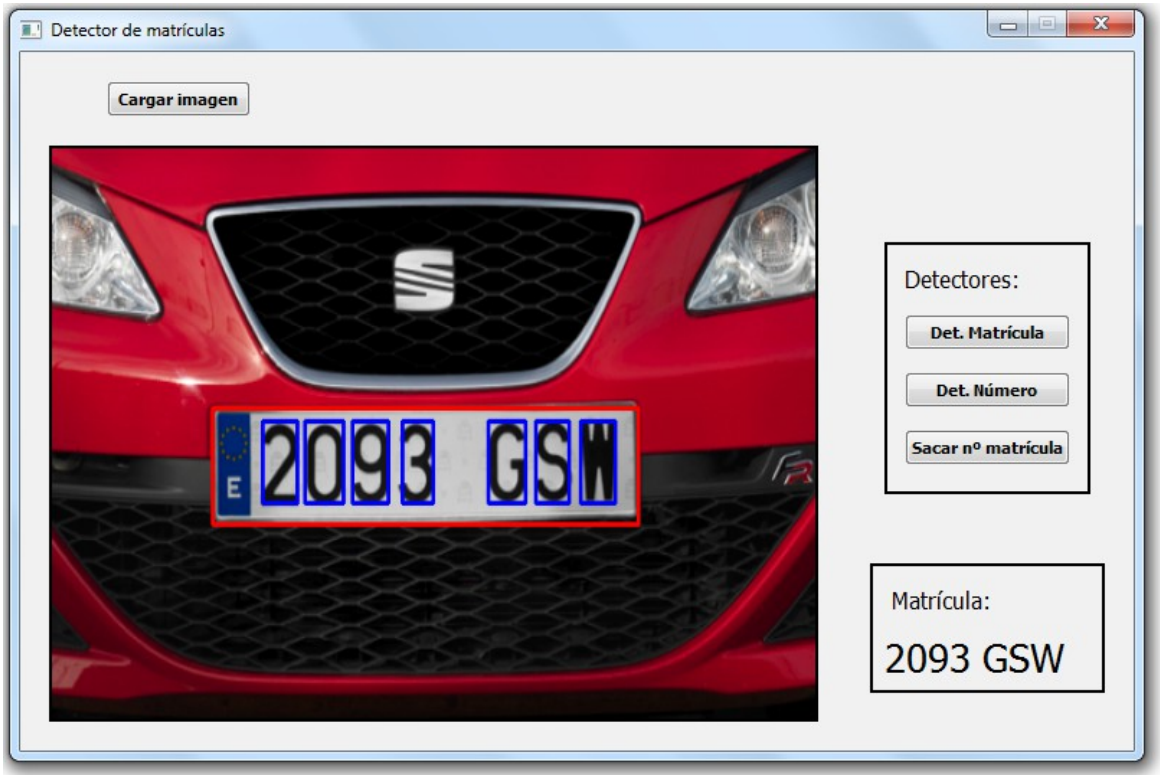
Ejemplo 1



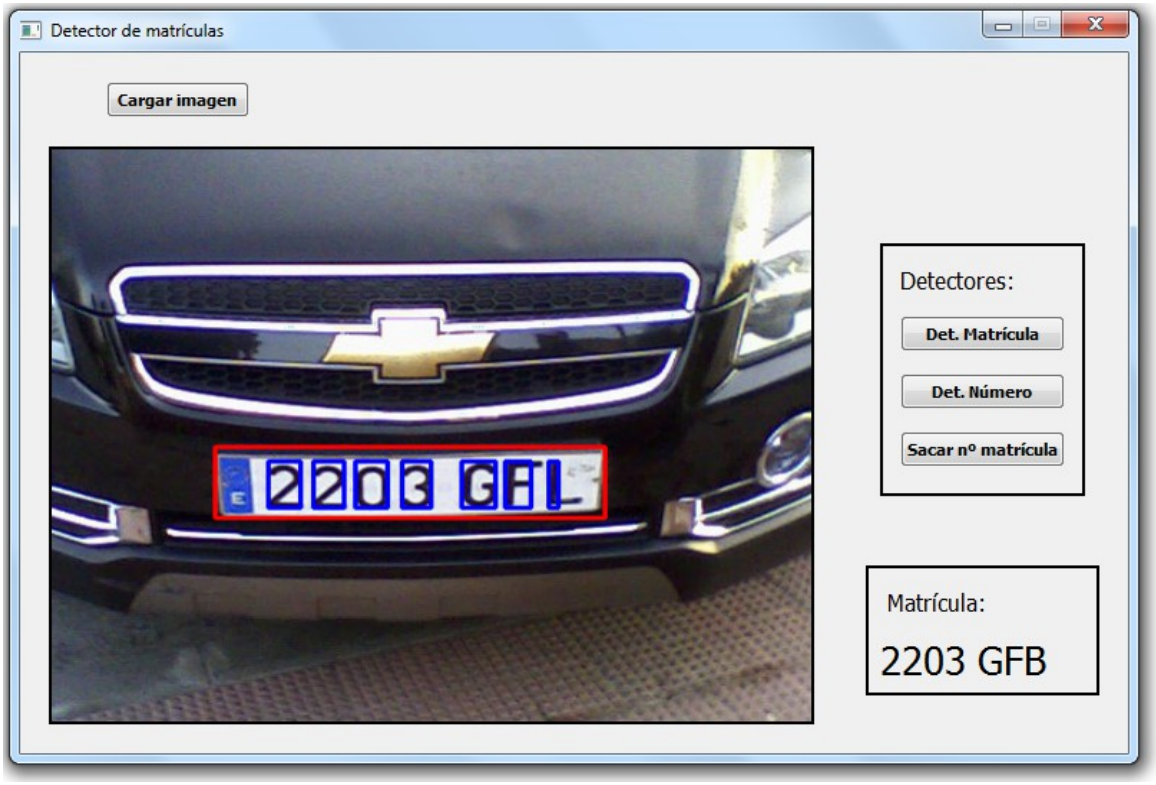
Ejemplo 2



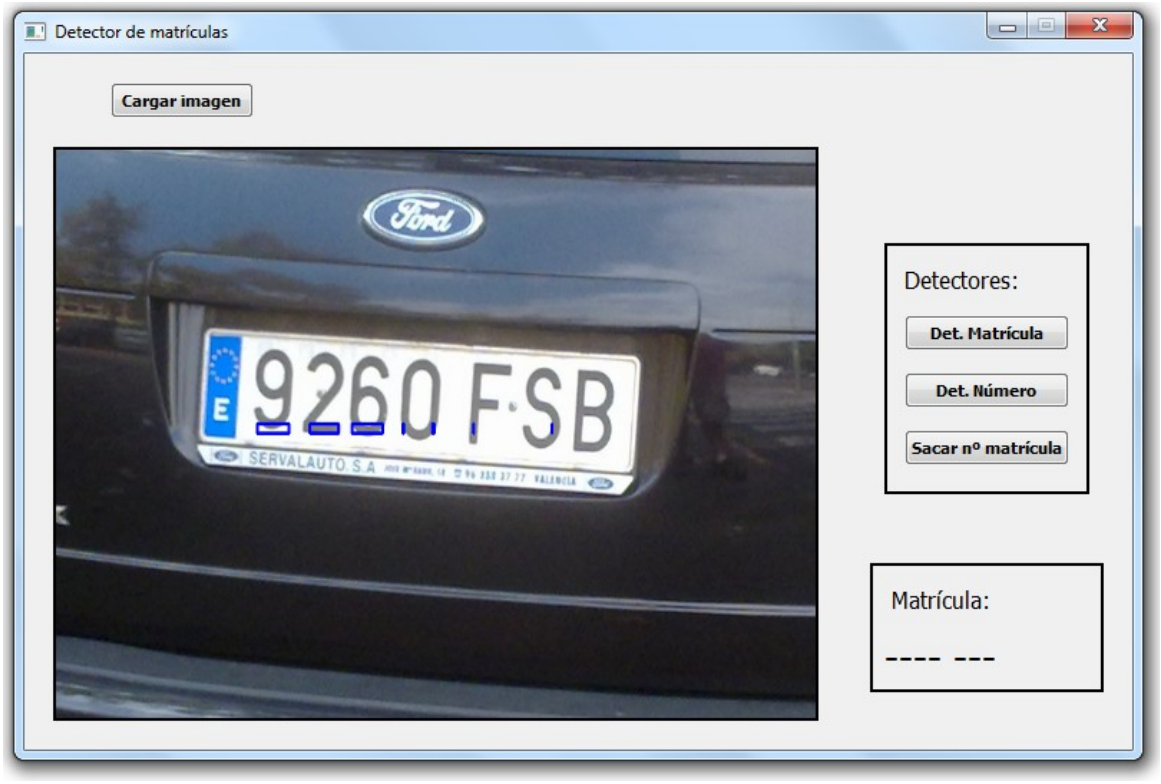
Ejemplo 3



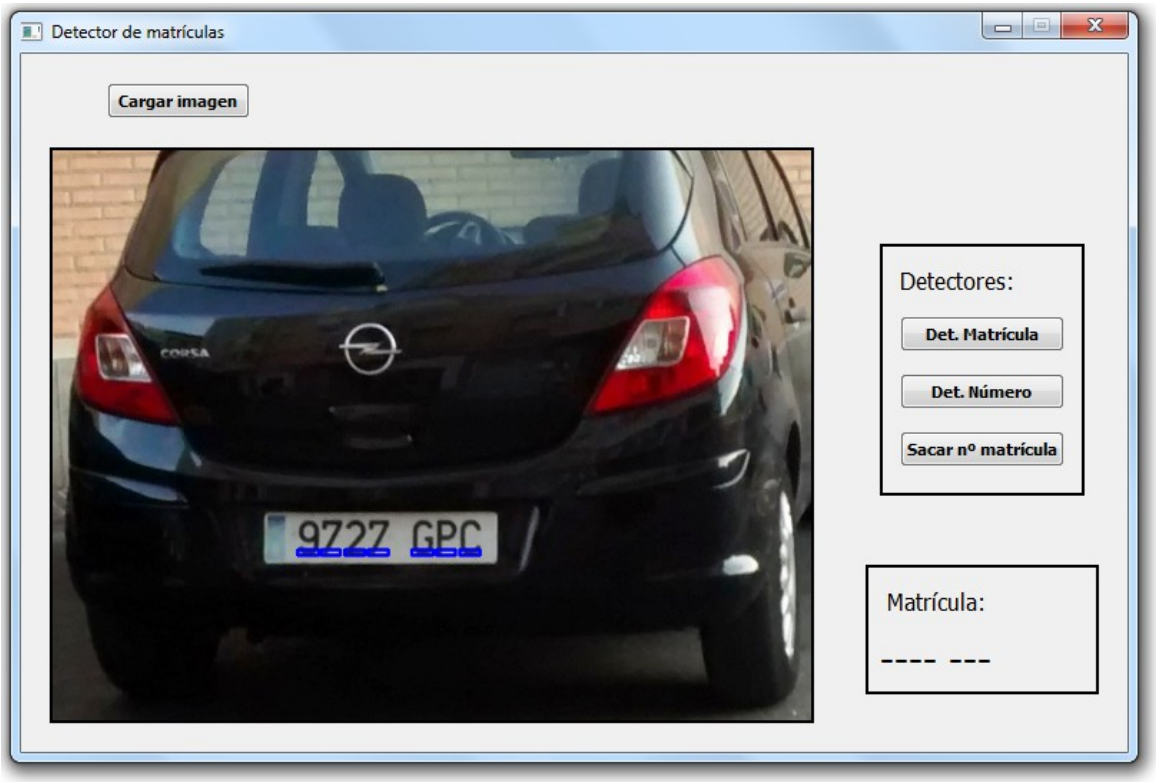
Ejemplo 4



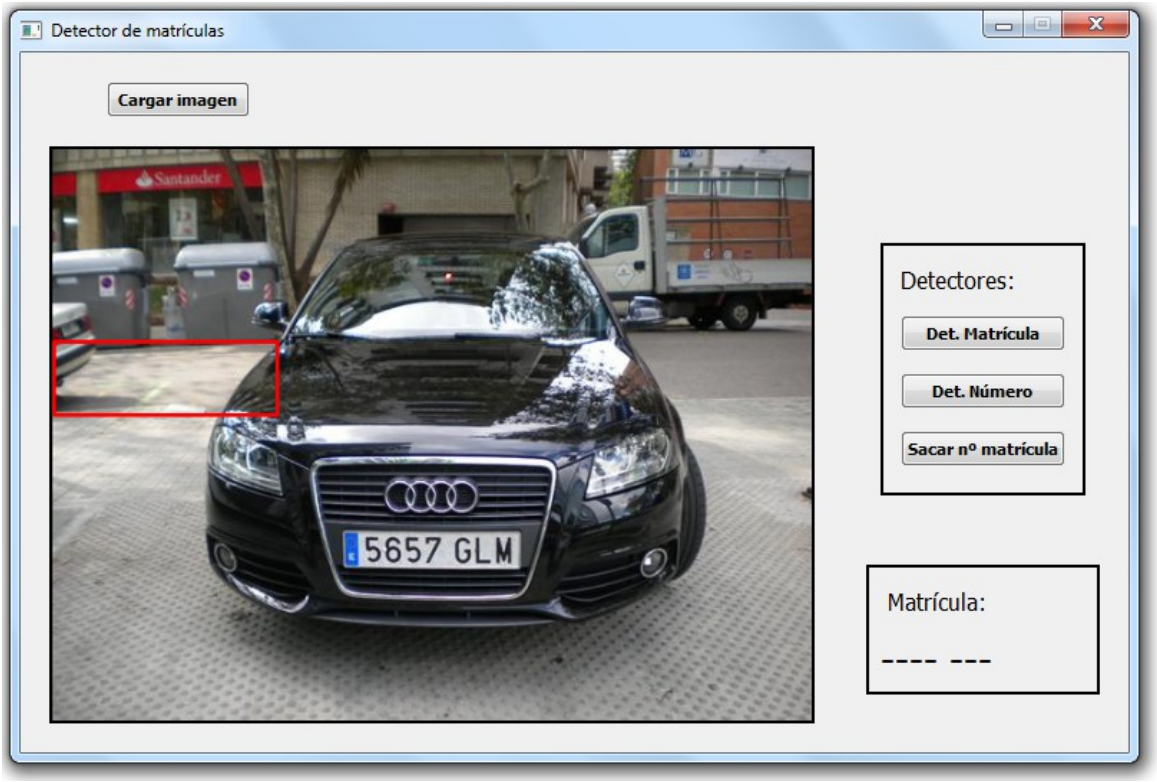
Ejemplo 5



Ejemplo 6



Ejemplo 7



Reflexión sobre los ejemplos

Como se puede observar, en los tres primeros ejemplos el programa funciona correctamente en en imágenes de calidad, incuso se puede observar que el ejemplo 2 tiene la matrícula un poco torcida pero la detecta perfectamente. A partir de este ejemplo ya vienen los ejemplos en los que podemos apreciar los errores de el programa. Como se observa en el ejemplo 4, en este caso detectamos bien todo menos la letra L. Esto es debido a que la finura de la parte de abajo de la L, que es tan fina que el algoritmo no lo reconoce como parte del carácter.

Con los casos de los ejemplos 5 y 6 tenemos otro de los problemas que he detectado, detectan las matriculas correctamente pero al contener brillos irregulares, en el proceso de hacer las proyecciones vertical y horizontal los máximos no corresponden a donde están los bordes de los números. Para solucionar esto deberíamos algún tipo de corrección de sombras o asegurarnos de que las entradas del programa siempre no padezcan estas sombras utilizando algún sistema de iluminación al realizar las instantáneas que vamos a procesar.

Por ultimo, en el ejemplo 7 tenemos el error menos deseable, que se confunda en el punto de reconocer donde está la matrícula y detecte cualquier otro lugar en vez de la matrícula. Esto es debido a un fallo en la detección de cual es el rectángulo bueno de la matrícula, concretamente este caso supongo que seguramente se solucionaría con una función en la que se calculara el rectángulo con mayor varianza, es decir el rectángulo que tuviera mas cambios bruscos entre blanco y negro.

6

CONCLUSIONES

El objetivo del proyecto era desarrollar un software con una interfaz gráfica manejable, que permitiera detectar automáticamente matrículas a partir de imágenes de coches y extraer el numero con un formato de texto legible por el ordenador. Este objetivo se a cumplido, aunque como se a podido ver en el apartado anterior con los ejemplos, el programa tiene sus limitaciones.

Estas limitaciones vienen en gran parte de que al empezar el proyecto de cero tuviera que estudiar mucho sobre C++ y aprender el funcionamiento de la librería openCV y por desconocimiento de alternativas tomara alguna decisiones no demasiado correctas por desconocimiento de las alternativas mejores, un ejemplo es la manera de detectar la matricula, que ha limitado en gran medida el correcto funcionamiento del programa. Por eso ahora propondré algunas de las posibles vías futuras que en mi opinión serian necesarias para mejorar el programa.

En primer lugar la detección de matricula debería mejorarse la detección de las matriculas para que sea mas funcional,

En segundo lugar, la corrección de inclinación de las matriculas, que nos permitiría que las imágenes pudiesen ser de distintos ángulos.

Y por ultimo, la creación de una red neuronal para la detección de caracteres, esta el menos necesario por el buen funcionamiento que del Pattern matching, pero seria interesante la creación de una red neuronal.

7

BIBLIOGRAFÍA

A continuación se detallan las fuentes utilizadas para la elaboración de este proyecto.

Primero la bibliografía impresa:

- Apuntes de la asignatura Tratamiento Digital de Imágenes, de la carrera
- Richard Szeliski. Computer Vision: Algorithms and Applications (2010)
- David A. Forsyth & Jean Ponce. Computer Vision: A Modern Approach
- Marco A. Peña Basurto y José M. Cela Espín. Introducción a la programación en C (2000). Ed. UPC
- Gary Bradski & Adrian Kaehler. Learning OpenCV: Computer Vision with the OpenCV Library(2008). Editorial O'Reilly
- Robert Laganière. OpenCV 2 Computer Vision Application Programming Cookbook (2011).
- Ondrej Martinsky. Algorithmic and mathematical principles of automatic number plate recognition systems(2007). B.SC. THESIS
- David González Gutiérrez. Tutorial de Qt4 Designer y Qdevelop (2008). Proyecto de Fin de Carrera UPC

Paginas web utilizadas:

- Web sobre C++:
  - http://www.cplusplus.com/
  - http://www.codeblocks.org
  - http://programming-motherfucker.com/
- Web del proyecto openCV: http://opencv.org/documentation.html
- Web del proyecto Qt: http://qt-project.org/doc/
- web de eclipse: http://eclipse.org
- Enciclopedia: http://es.wikipedia.org/
- Buscador: https://www.google.es/