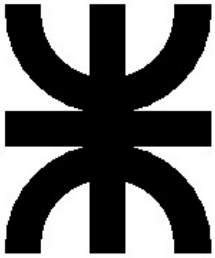


UNIVERSIDAD TECNOLOGICA NACIONAL
FACULTAD REGIONAL SAN NICOLAS

INGENIERIA EN ELECTRONICA

PROBLEMA DE INGENIERÍA

TECNICAS DIGITALES III

Reconocimientos Automático de Matrículas ANPR

Integrantes:

- Aquino López, Juan Carlos

Docentes:

- Profesor: Poblete Felipe
- Auxiliar: Gonzalez Mariano

AÑO 2015

INDICE

1. RECONOCIMIENTO AUTOMATICO DE MATRICULAS

En el presente trabajo 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 número 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 Microsoft Visual Studio 2010. Una vez familiarizados con las herramientas se elaborara un primer programa funcional.

• La segunda etapa de cargar imagen, detectar matrícula y números y devolver en formato texto almacenandolo en el oredenador.

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

1.1 *MATERIAS INTEGRADAS*

• Informatica II.
• Álgebra y Geometría Analítica.
•Calculos de Probabilidad y Estadisticas.

1.2 *POSIBLES APLICACIONES*

• Sistemas de seguridad desde zonas urbanas a zonas comunes como parquímetros.
• Control y accesos a instalaciones.
•Tarificación de peajes.
•Cálculo de la velocidad media entre puntos de una carretera.

1.3 *BIBLIOGRAFÍA*

• *Sitios de Internet.*
- *http://opencv.org/*

• *Libros*
- *OReilly-LearningOpenCV, OpenCV By Example.*

2. DESARROLLO

2.1 INTRODUCCIÓN

El sistema de reconocimiento automático de matrículas es un sistema embebido en tiempo real que reconoce automáticamente la placa de vehículos. Hay muchas aplicaciones que van desde Sistemas complejos de seguridad a las zonas comunes y desde estacionamientos al control de tráfico urbano. Reconocimiento automático de matrículas (ALPR) tiene características complejas debido a diversos efectos tales como de la luz y la velocidad.

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.

Este sistema ALPR fue construido apartir software libres incluyendo C++ y Open Computer VisionLibrary (OPENCV).

2.2 FUNDAMENTOS TEÓRICOS

2.2.1 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.

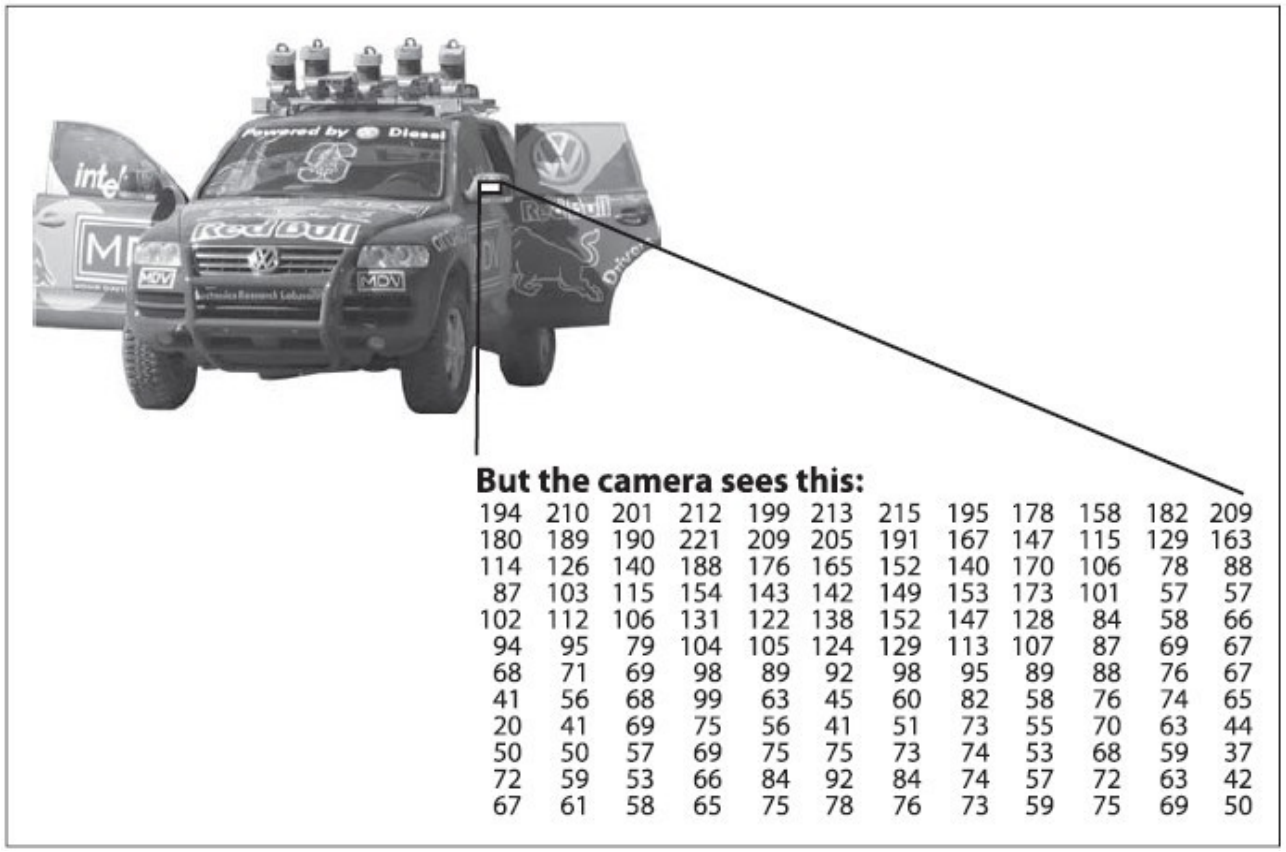


Figura 2.2.1: Para un ordenador esta imagen es solo una cuadricula de números

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.

2.2.2. Procesamiento digital de imágenes

El procesamiento digital de imágenes se define como el campo que se encarga de manipular imágenes digitales con el objetivo de mejorarlas o identificar cierta información relevante. Involucra, desde el punto de vista de los cálculos, tratamiento matemático de matrices y análisis estadístico.

A continuación se presentan los conceptos básicos relacionados con el procesamiento de imágenes:

- Pixel: elemento básico de una imagen.

- Imagen: arreglo bidimensional de píxeles con diferente intensidad luminosa (escala de gris). Matemáticamente una imagen se representa por r=f(x,y) en donde r es la intensidad luminosa del pixel, cuyas coordenadas son (x,y).

- Color: el color se forma mediante la combinación de los tres colores básicos: rojo, verde y azul (RGB). Esto quiere decir, que una imagen a color está representada matemáticamente por 3 arreglos bidimensionales de píxeles (matrices), uno para cada color básico.

2.2.3. Captura

La captura de imagen representa el front-end del sistema en el cual intervienen las cámaras y adquisidores de imágenes, formadas, al igual que el ojo humano, por una lente convergente que proyecta la imagen sobre una superficie sensible a la luz denominada sensor de imagen o sensor óptico.

2.2.4. Sensores ópticos

Los sensores digitales de imagen están formados por una serie de foto sensores que modifican su señal eléctrica según la intensidad luminosa que reciben, lo que permite la captura de los puntos que conforman la imagen. Estos sensores suelen estar configurados en forma matricial de modo que proporcionan una imagen bidimensional. También, a nivel industrial, se emplean sensores en configuración de línea, que dada sus altas resoluciones, se utilizan para aplicaciones de medida.

Los sensores son realizados en dos tecnologías, principalmente en CCD y en CMOS. La tecnología CCD es la que proporciona mejor calidad de imagen y con menor ruido. Sensores CMOS son generalmente más susceptibles al ruido pero consumen 100 veces menos potencia. La tecnología CMOS es la más económica y ofrece un menor tamaño, gracias a su grado de integración electrónica. Además puede capturar partes de la imagen no teniendo que transmitirla completamente.

2.2.5. ¿Qué es tiempo real?

Se puede decir que tiempo real es una interpretación de una persona que visualiza, escucha, siente o interpreta una secuencia multimedia como si fuera “fluida” o continua, sin importar si en realidad son secuencias discretas de datos. El ejemplo más clásico es el del video en una cinta con fotogramas, como el cine. Si los fotogramas pasan demasiado lento se percibirá una “secuencia por pasos”.

Entonces esto lleva a definir un margen mínimo de fps (frames por segundo) necesario para que un sistema de captura de imágenes sea interpretado como de tiempo real.
En la práctica el margen mínimo está en el orden de los 15 fps. Dicho valor está determinado por la persistencia de una imagen en la retina del ojo humano, que es de 16 ms, sumada a la persistencia de la corteza visual que oscila entre 100 y 400 ms.

2.2.6 Acondicionamiento y transformaciones básicas

La tarea de implementar un sistema de procesamiento digital de imágenes involucra capturar, filtrar o acondicionar, procesar e interpretar los resultados.

A continuación se describen las principales funciones de acondicionamiento y transformación aplicadas a imágenes:

2.2.6.1 Smoothing

Al igual que en las señales unidimensionales, las imágenes también se pueden filtrar con varios filtros de paso bajo (LPF), filtros de paso alto (HPF), etc. El LPF ayuda a eliminar ruidos, a difuminar las imágenes, etc. Imágenes.

El desenfoque de la imagen se consigue convolucionando la imagen con un kernel de filtro de paso bajo. Es útil para eliminar ruidos. Realmente elimina contenido de alta frecuencia (por ejemplo, ruido, bordes) de la imagen.

Es decir, se realiza deslizando una ventana (kernel o filtro) a través de toda la imagen y calculando cada píxel un valor basado en el valor del kernel y el valor de píxeles superpuestos de la imagen original. Este proceso se denomina matemáticamente convolucionando una imagen con un núcleo. El núcleo es la única diferencia en todos los tipos anteriores de suavizado (Blurring) métodos.
OpenCV proporciona principalmente cuatro tipos de técnicas de borrosidad.

2.2.6.2. Averaging

Esto se hace convolucionando la imagen con un filtro de caja normalizado. Simplemente toma el promedio de todos los píxeles bajo el área del kernel y reemplaza el elemento central. Esto se hace mediante la función **cv2.blur ()** o **cv2.boxFilter ()** .

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

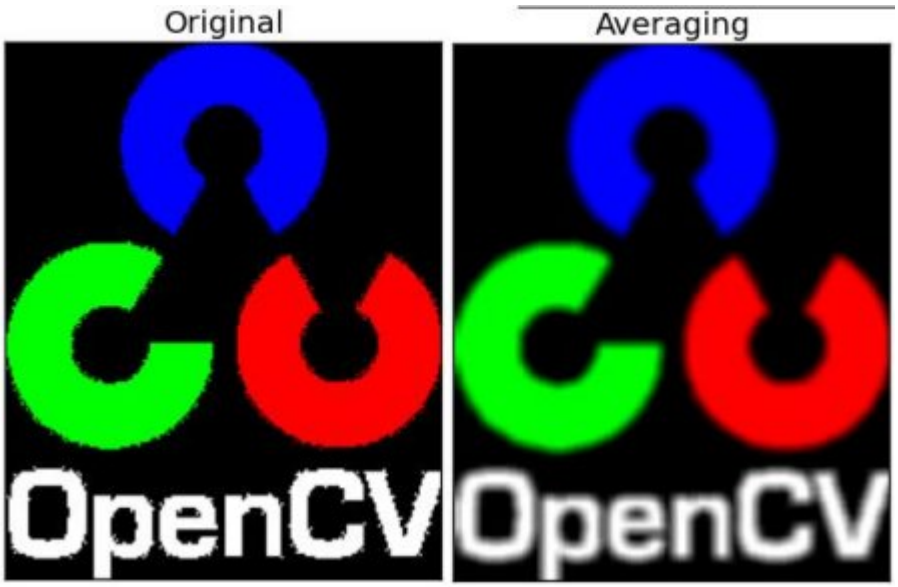


Figura 2.2.6.1 Averaging.

En este, en lugar de filtro de caja, se utiliza el núcleo gaussiano. Se hace con la función, **cv2.GaussianBlur ()** .
Gaussian es muy eficaz en la eliminación de ruido sobre la imagen.

Figura 2.2.6.2 GaussianBlur.

2.2.6.3. Median Blurring

Aquí, la función **cv2.medianBlur ()** toma la mediana de todos los píxeles bajo el área del kernel y el elemento central se sustituye por este valor mediano.Esto es muy eficaz contra el ruido de sal y pimienta en las imágenes. Lo interesante es que, en los filtros anteriores, el elemento central es un valor recién calculado que puede ser un valor de píxel en la imagen o un nuevo valor. Pero en el desenfoque mediano, el elemento central siempre es reemplazado por algún valor de píxel en la

imagen. Reduce el ruido de manera efectiva. Su tamaño de núcleo debe ser un número entero impar positivo.

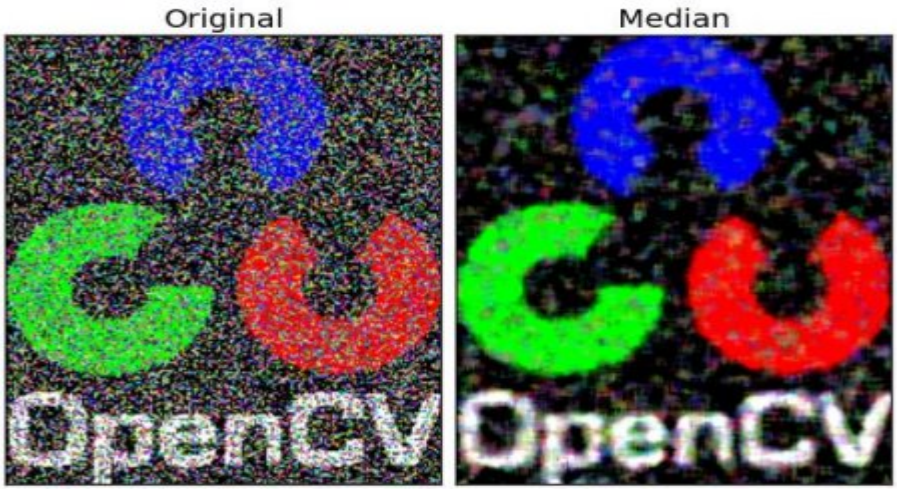


Figura 2.2.6.3 MedianBlurring.

2.2.6.4 Erode y Dilate

Los algoritmos de erosión (erode) y dilatación (dilate) rara vez aparecen independientes, por eso se los agrupa como un solo método.

Se aplican a imágenes grises (escala de blancos entre 0-255), y son transformaciones de tipo morfológicas, ya que matemáticamente hablando hacen una convolución en toda la imagen desplazando un kernel (o ventana superficial).

En la figura 2.2.6.4 se muestran cuatro etapas de erosión consecutiva, empezando por la imagen superior izquierda. Como puede verse, la erosión “achica proporcionalmente” los cuerpos u objetos cerrados y lo opuesto hace la dilatación, también proporcionalmente al área del objeto.

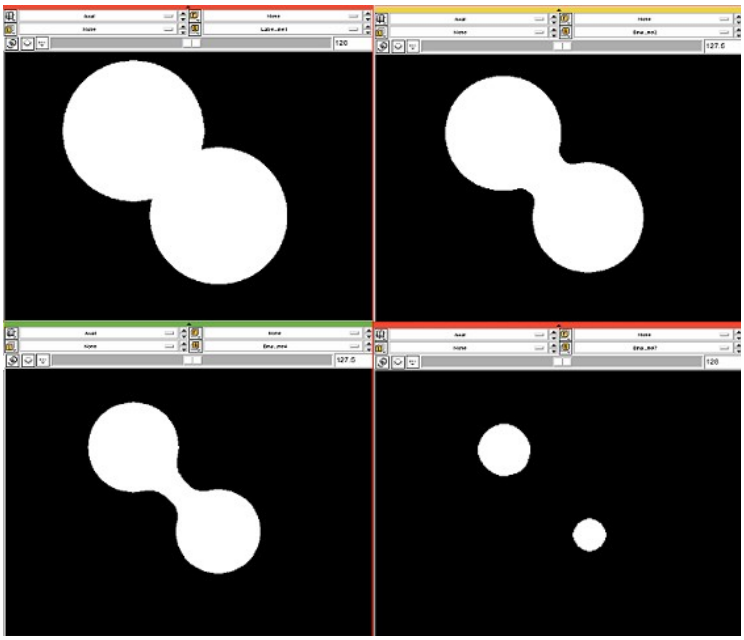


Figura 2.2.6.4 Erode.

2.2.6 . ThresHold

Los algoritmos de umbral (ThresHold) se aplican también a imágenes en escala de grises, y son prácticamente un filtro pasa banda de color de ventana cuadrada. El mismo deja pasar sólo las intensidades de color que estén dentro del rango especificado. Se usa habitualmente para binarizar imágenes.

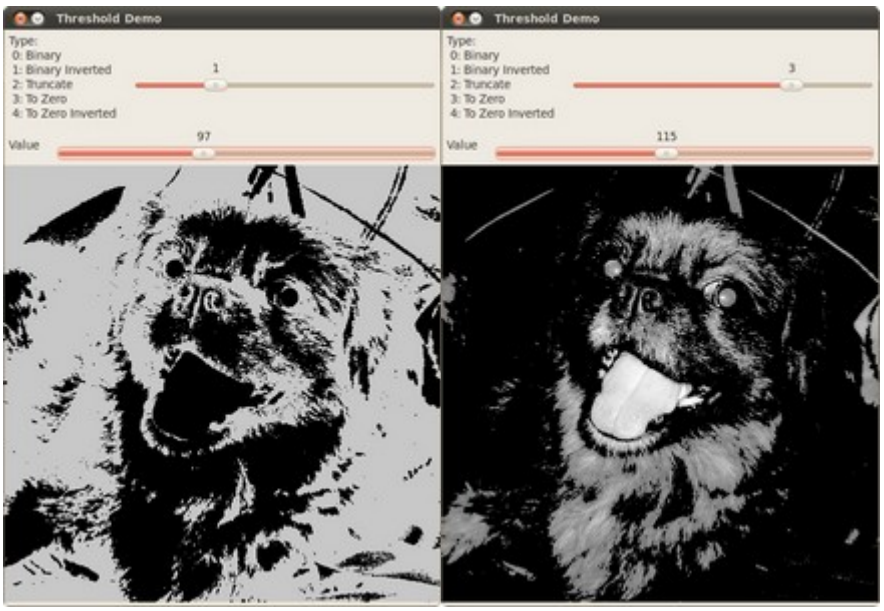


Figura 2.2.6 Threshold

2.2.7. Aproximación del contorno

La detección de contornos realiza un seguimiento recursivo de flancos de color en una imagen.Para que dicho algoritmo se ejecute adecuadamente, la imagen debe tener ciertas características como ser: estar binarizada, poseer contornos detectables y que los contornos sean de valor 1 y el fondo 0. Todas estas características se logran con las funciones previamente descriptas en el apartado anterior.

La detección de contornos es algo sencillo y fundamental en el procesamiento digital de imágenes y videos. Los métodos existentes para detectar contornos son muy diversos, pero básicamente hay 3:

- Inscribir una geometría conocida (cuadrado o círculo), muy rápido.
- Aproximación polinómica (interpolación), velocidad moderada.
- Búsqueda cruda (analiza cada pixel), muy lento.

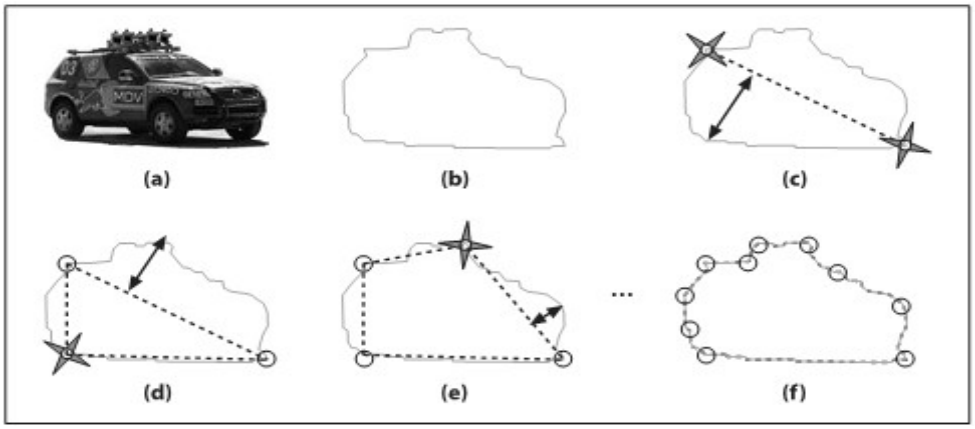


Figura 2.2.7a Aproximación polinómica

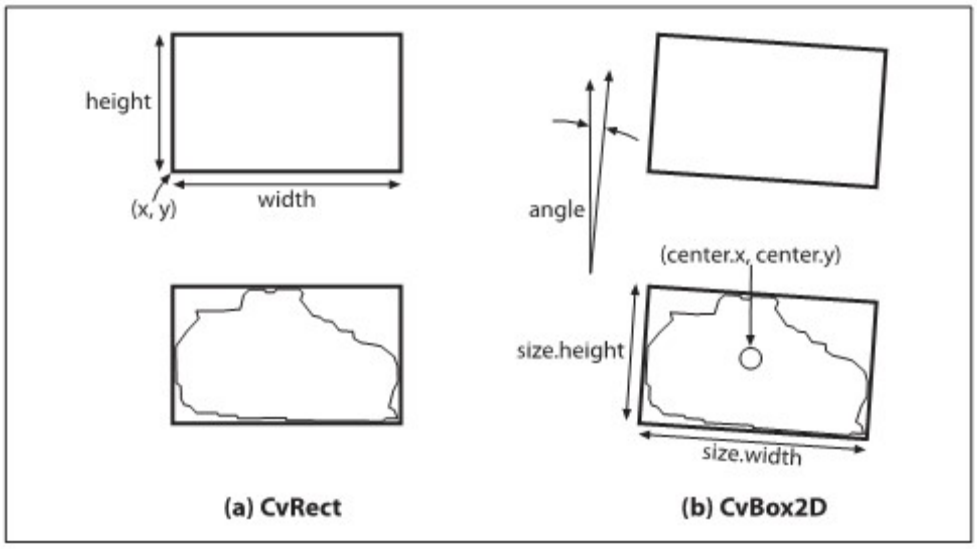


Figura 2.2.7b Inscripción de una geometría

A partir de la obtención de contornos se pueden realizar diversos tipos de análisis, incluyendo momentos, concordancia, estudio de convexidad, análisis geométrico, etc.

2.2.8. Moments

Usa como argumentos los contornos de una imagen. Es la manera más simple de comparar contornos que ofrece OpenCV y se basa en obtener el "centro de masa" de un contorno en cuestión.

Esto muchas veces es más que suficiente para obtener otros parámetros como ser: el área, el punto central del contorno, comparar dos contornos para evaluar un índice de similitud entre 0 y 1, etc.

2.2.9. Detección de blob

Un Blob es un grupo de píxeles conectados en una imagen que comparten alguna propiedad común (por ejemplo, el valor de escala de grises). El objetivo de la detección de blob es identificar y marcar estas regiones.

Se utiliza un rectángulo mínimo para aproximar una forma más compleja. Es un rectángulo cuyos lados son paralelos a los *x* y *Y* axises y mínimamente encerrar la forma más compleja.

2.2.10*. Rectángulo de encuadernación*

Se utiliza un rectángulo mínimo para aproximar una forma más compleja. Es un rectángulo cuyos lados son paralelos a los *x* y *Y* axises y mínimamente encerrar la forma más compleja.

Hay dos tipos de rectángulos delimitadores:

2.2.10.1 *Rectángulo recto*

Es un rectángulo recto, no considera la rotación del objeto. Así que el área del rectángulo delimitador no será mínima. Se encuentra por la función cv2.boundingRect () .

2.2.10.2 *Rectángulo girado*

Aquí, el rectángulo delimitador se dibuja con el área mínima, así que considera la rotación también. La función utilizada es cv2.minAreaRect () . Devuelve una estructura Box2D que contiene los siguientes detales - (centro (x, y), (ancho, altura), ángulo de rotación). Pero para dibujar este rectángulo, necesitamos 4 esquinas del rectángulo. Se obtiene mediante la función cv2.boxPoints ().

Ambos rectángulos se muestran en una sola imagen. El rectángulo verde muestra el rect rectangular normal. El rectángulo rojo es el rect rotado.

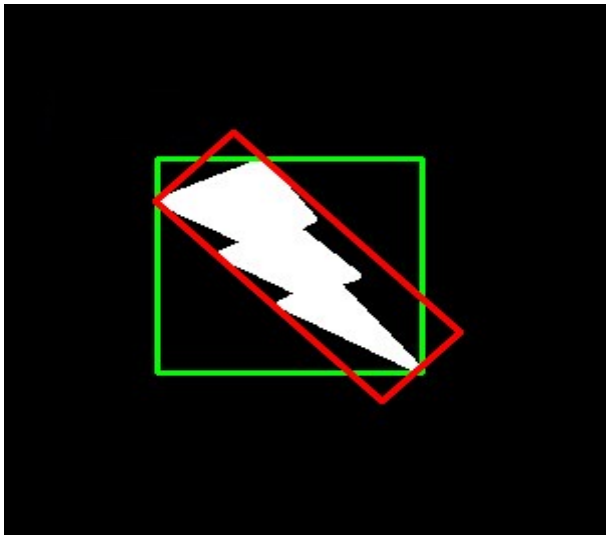


Figura 2.2.10.2a

2.2.11 *Localización de la matrícula*

Con el fin de reconocer patentes necesitaremos algoritmos que nos permitan manipular las imágenes.Puesto que en el proyecto se intenta identificar qué características son las más apropiadas para la identificación de las patentes. Se hace necesario añadir algún tipo de algoritmo de inteligencia artificial que nos ayude a identificar qué parámetros son los más adecuados, es aquí donde entran las redes neuronales artificiales.

Las redes neuronales nos ayudarán a, dados unos datos de entrada, que serán las características extraídas de las imágenes, indicaremos a qué imagen corresponden esos datos de entrada, de tal forma que al entrenar la red los pesos de las neuronas se vayan reajustando hasta obtener la salida deseada con el menor error posible.

Cuando las imágenes son complejas y no se sabe muy bien qué patrones o características son

las que definen mejor dichas imágenes, la utilización de redes neuronales puede ayudar a averiguar estas características.

Lo ideal es tomar muchas imágenes de un solo tipo, en nuestro caso tomaremos imágenes de matricula de Autos, una vez calculadas las características se aplica un algoritmo de aprendizaje sobre las mismas.

2.2.11.1. *Redes Neuronales*

Una Red neuronal Artificial es una estructura compuesta de un numero de unidades interconectadas (neuronas artificiales). Cada unidad posee una caracteristicas de entrada/salida e implementa una computación local o funcion. La salida de cualquier unidad esta determinada por su carateristica de entrada/salida su interconexión con otras unidades, y (posiblemente) de sus entradas externas.

El funcionamiento esta basado básicamente en aplicar un conjunto de entradas, cada una representando la salida de otro neuron, o una entrada del medio externo, realiza una suma ponderada con estos valores y "filtrar" este valor con una funcion como se puede observar en la figura.

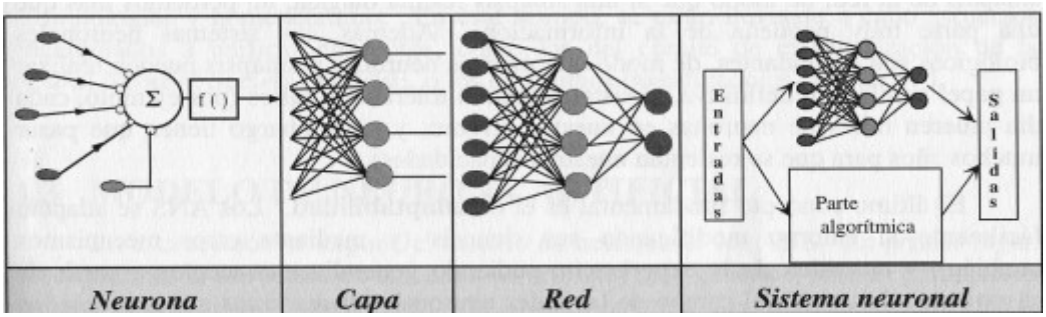


Figura 2.2.11a

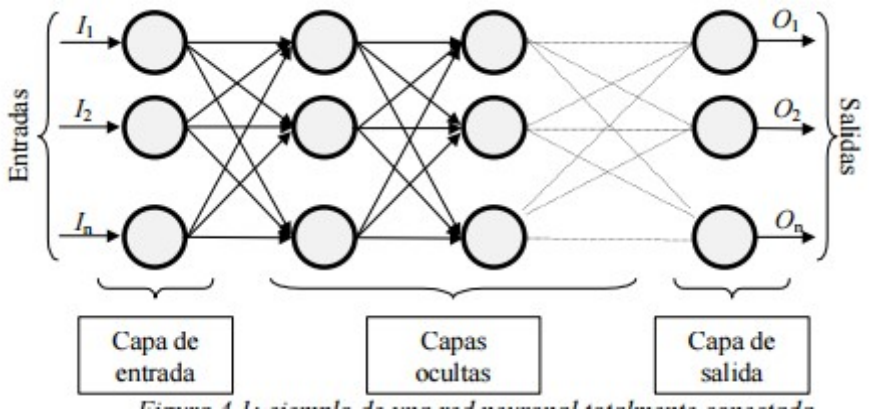


Figura 2.2.11b

La misma está constituida por neuronas interconectadas y arregladas en tres capas (esto último puede variar). Los datos ingresan por medio de la "capa de entrada", pasan a través de la "capa oculta" y salen por la "capa de salida". Cabe mencionar que la capa oculta puede estar constituida por varias capas.

2.2.12. OCR

2.2.12.1. *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.

2.2.12.2. *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 acabo:

    • 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 posprocesado 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.

2.2.13. ANPR

2.2.13.1. *¿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.

2.2.13.2. *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.

2.2.13.3 *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.

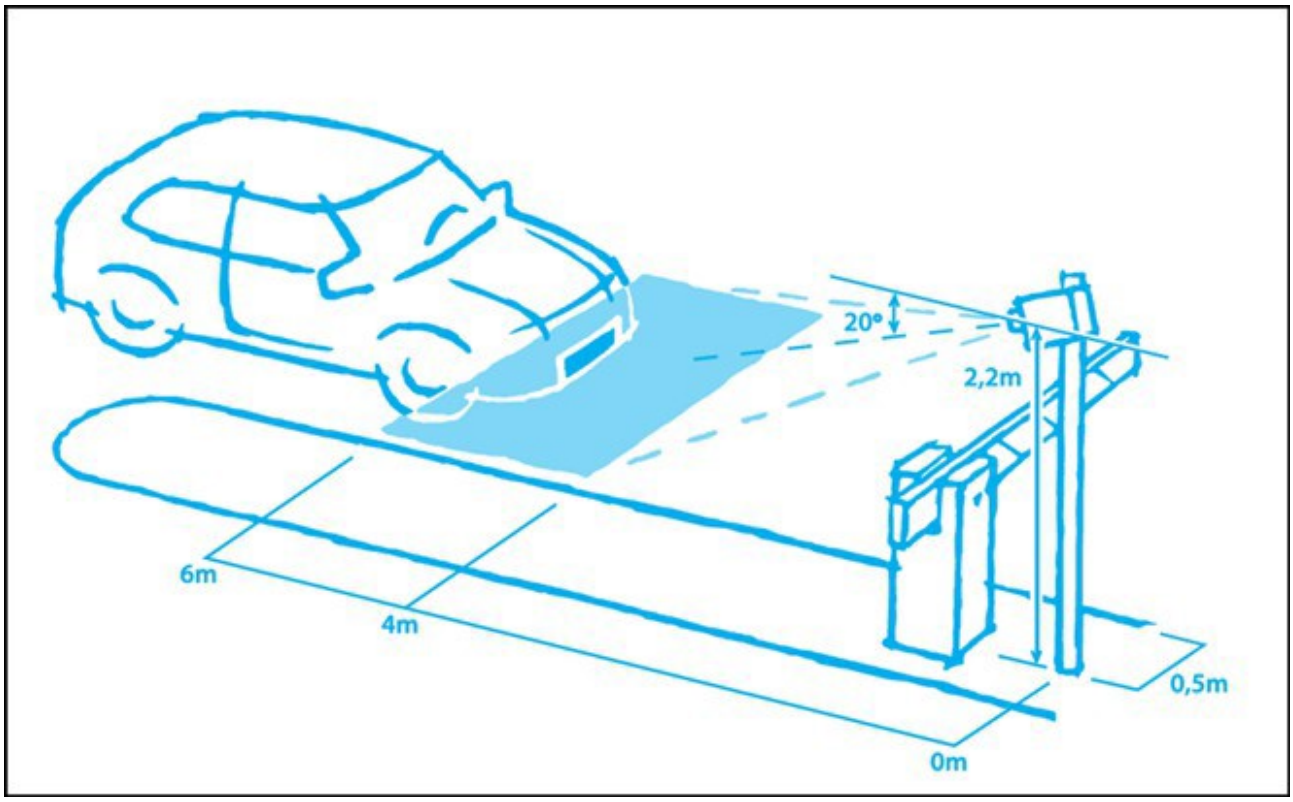


Figura 2.2.1.a: Esquema de colocación de hardware a la entrada de un aparcamiento

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 2.2.1.b*: 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.

2.3. HARDWARE

2.3.1. Webcam

Para la selección del sensor óptico se tuvo en cuenta que el mismo debía cubrir dos necesidades. La primera es la de poder contar con una calidad de imagen tal que permita el desarrollo del proyecto. Por otro lado, era necesario que se trate de un dispositivo del tipo UVC, “USB video device class”, para que sea compatible con el software a desarrollar.

En el mercado local se ofrecen webcams de muy buena calidad con soporte UVC, a precios accesibles, alrededor de 100 U$S, y que resultan ideales para la aplicación. De todas formas, este precio eleva considerablemente el costo del proyecto. Por este motivo se optó por una webcam de calidad regular y con soporte UVC, marca Seisa, de fabricación china y con un precio rondando los 10U$S, que para el prototipo fue adecuada.

Figura 2.3.1 Webcam utilizada

Posee un sensor óptico CMOS, que permite trabajar en tiempo real con los siguientes parámetros:

- Resolución: 1280 x 1024
- Fps: 15 a 30 fps
- Color: 8 bits
- Ajustes de foco: manual.

El sistema de software desarrollado implementa imágenes de resolución 320x240, color de 8 bits, y fps a demanda, es decir no existe un mínimo ni un máximo, ya que depende del procesamiento que se realice entre capturas.

2.3.2 Ordenador

El sistema de visión artificial debe contar con un sistema computacional que se encargue de las siguientes tareas:

- Realizar la lectura de las imágenes.

- Procesar los datos proporcionados por la cámara para realizar el análisis de imagen (realización de los algoritmos y transformaciones de las imágenes).

- Brindar la interfaz gráfica al usuario.

Los requisitos mínimos que debe cumplir la PC a utilizar son: procesador de 2GHz, 1,5 GB de memoria RAM, placa de video integrada con 256 Mb dedicados, dos puertos USB 2.0, sistema operativo Windows 7 o Linux.

2.4. FUNDAMENTOS DE SOFTWARE

2.4.1. Lenguaje de programación: C++

Para hablar de C++ antes es necesario explicar que es 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.

2.4.2. 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.

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.

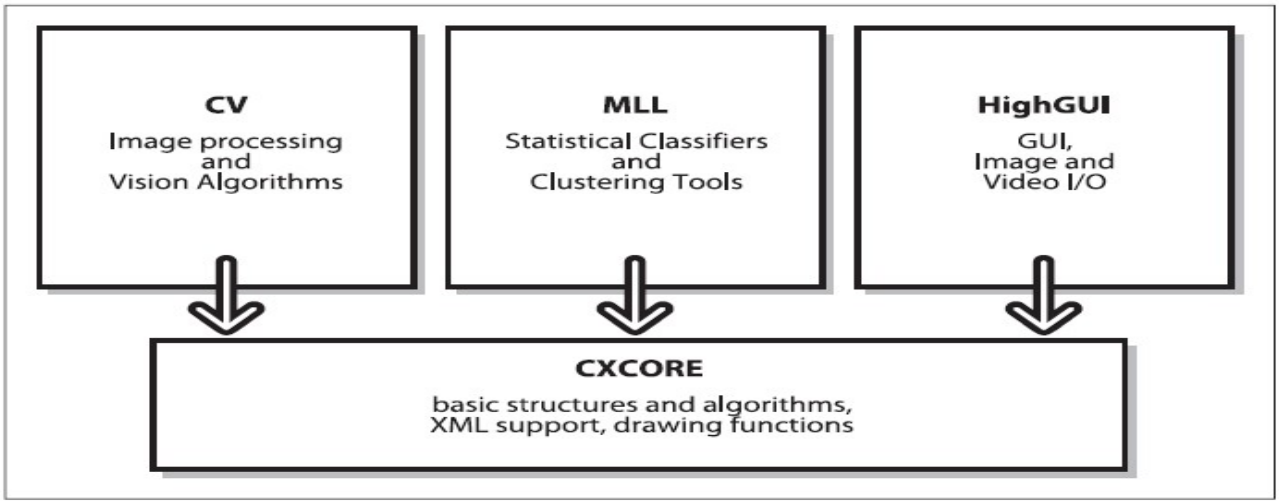


Figura 2.4.2 Estructura básica del OpenCV

2.4.3. Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java,Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NETMVC, Django, etc., a lo cual hay que sumarle las nuevas capacidades online bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y consolas, entre otros.

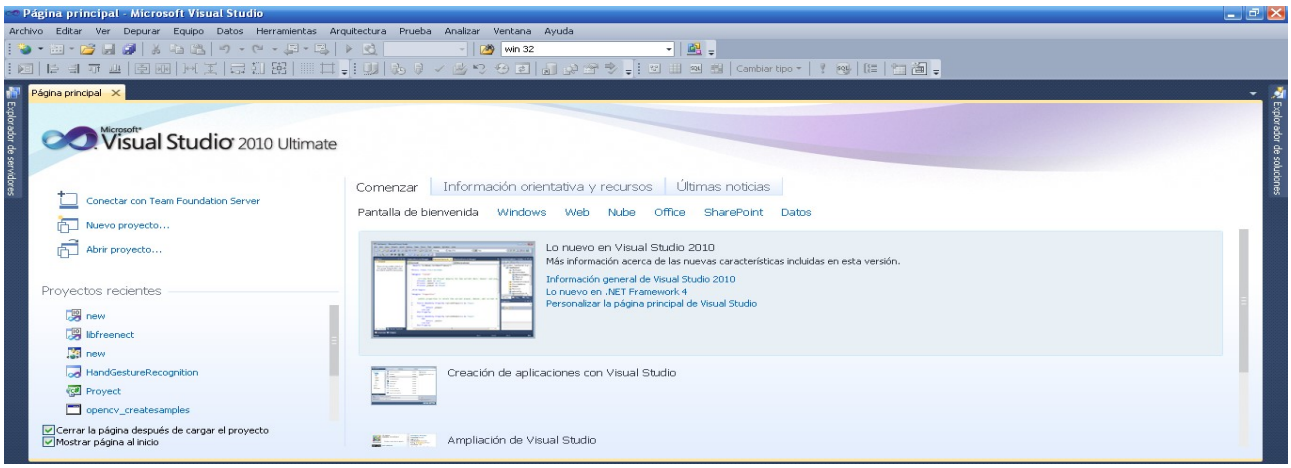


Figura 2.4.3 *Microsoft Visual Studio*

2.5. Reconocimiento de patrones con OpenCV

Reconocer patrones dentro de una imagen utilizando la red neuronal. Esto se puede extrapolar a un frame de un video y conseguir detectar objetos a través del webcam..

2.5.1. Cascade Classifier Training

Trabajar con una cascada potenciada de clasificadores débiles incluye dos etapas principales: el entrenamiento y la etapa de detección.

Se utilizarán varias aplicaciones OpenCV oficiales:
`opencv_createsamples , opencv_annotation , opencv_traincascade yopencv_visualisation .`

2.5.2. Preparación de los datos de formación

Para entrenar una cascada potenciada de clasificadores débiles necesitamos un conjunto de muestras positivas (que contienen objetos reales que desea detectar) y un conjunto de imágenes negativas (que contiene todo lo que no desea detectar). El conjunto de muestras negativas debe prepararse manualmente, mientras que el conjunto de muestras positivas se crea utilizando la aplicación opencv_createsamples.

2.5.3. Muestras negativas

Las muestras negativas se toman de imágenes arbitrarias, no contienen objetos que desea detectar. Estas imágenes negativas, de las que se generan las muestras, deben aparecer en un archivo de imagen negativo especial que contenga una trayectoria de imagen por línea (puede ser absoluta o relativa) .Tenga en cuenta que las muestras negativas y las imágenes de muestra también se denominan muestras de fondo o imágenes de fondo y se utilizan indistintamente

Para estas imágenes también hay que generar un archivo (negativas.txt) que sirve de indice para las etapas posteriores, con la siguiente estructura:

```
negativas/img1.jpg
negativas/img2.jpg
```

```
negativas/img3.jpg
negativas/imgn.jpg
```

Su conjunto de muestras de ventanas negativas se utilizará para indicar el paso de aprendizaje de la máquina, impulsando en este caso lo que no se debe buscar al intentar encontrar los objetos de interés.

2.5.4. Muestras positivas

Las muestras positivas son creadas por la aplicación opencv_createsamples. Son utilizados por el proceso de impulso para definir lo que el modelo debe buscar realmente al intentar encontrar sus objetos de interés. La aplicación admite dos formas de generar un conjunto de datos de muestra positivo.

1. Puede generar un montón de positivos desde una sola imagen de objeto positivo.
2. Se puede suministrar todos los positivos y utilizar solamente la herramienta para cortarlos hacia fuera, los vuelve a clasificar según el tamaño y los pone en el formato binario abierto del opencv.

Mientras que el primer enfoque funciona decentemente para objetos fijos, como logo muy rígido, tiende a fallar pronto para objetos menos rígidos. En ese caso sugerimos utilizar el segundo enfoque.

El primer enfoque toma una sola imagen de objeto con, por ejemplo, un logotipo de empresa y crea un gran conjunto de muestras positivas de la imagen de objeto dado girando aleatoriamente el objeto, cambiando la intensidad de la imagen y colocando la imagen en fondos arbitrarios. La cantidad y el rango de aleatoriedad pueden controlarse mediante argumentos de línea de comandos de la aplicación opencv_createsamples.

Argumentos de la línea de comandos:

- *-vec <vec_file_name>* : Nombre del archivo de salida que contiene las muestras positivas para el entrenamiento.
- *-img <image_file_name>* : Imagen del objeto fuente (por ejemplo, un logotipo de la empresa).
- *-bg <background_file_name>*: Archivo de descripción de antecedentes; Contiene una lista de imágenes que se utilizan como fondo para versiones distorsionadas aleatoriamente del objeto.
- *-num <number_of_samples>* : Número de muestras positivas a generar.
- -bgcolor <background_color>: Color de fondo (actualmente se asumen las imágenes en escala de grises); El color de fondo denota el color transparente. Puesto que puede haber artefactos de compresión, la cantidad de tolerancia de color se puede especificar mediante -

bgthresh. Todos los píxeles con bgcolor-bgthresh y bgcolor + bgthresh se interpretan como transparentes.

- *-bgthresh <background_color_threshold>*
- *-inv* : Si se especifica, los colores se invertirán.
- *-randinv* : Si se especifica, los colores se invertirán aleatoriamente.
- *-maxidev <max_intensity_deviation>* : Desviación de intensidad máxima de píxeles en muestras de primer plano.
- *-maxxangle <max_x_rotation_angle>* : El ángulo de rotación máximo hacia el eje x, debe ser dado en radianes.
- *-maxyangle <max_y_rotation_angle>* : El ángulo de rotación máximo hacia el eje y, debe ser dado en radianes.
- *-maxzangle <max_z_rotation_angle>* : El ángulo de rotación máximo hacia el eje z, se debe dar en radianes.
- *-show*: Opción de depuración útil. Si se especifica, se mostrará cada muestra. Al pulsar Esc se continuará el proceso de creación de muestras sin mostrar cada muestra.
- *-w <sample_width>* : Ancho (en píxeles) de las muestras de salida.
- *-h <sample_height>* : Altura (en píxeles) de las muestras de salida.

Entonces al ejecutar `opencv_createsamples` estaremos generando nuestro archivo de muestra positiva, creando un vector de imágenes positivas.

Para estas imágenes también debe generarse un archivo indice, pero que contienen algo más de información:

```
positivas/img1.bmp 1 100 100 25 25
positivas/img1.bmp 1 120 110 27 28
positivas/img1.bmp 1 130 900 30 31
positivas/img1.bmp 1 140 105 22 23
```

En este archivo se especifica donde se encuentra el objeto dentro de la imagen positiva. Siendo la primera el numero de objetos dentro de la imagen, las 2 siguientes son las coordenadas x e y (respectivamente) y las siguientes 2 ancho y largo.

Para un buen entrenamiento se necesitaran aproximadamente 5000 imágenes negativas y 1500 positivas, claro que para realizar las pruebas con unas 200 positivas y unas 1000 negativas bastará

2.5.5. Entrenamiento en cascada

El siguiente paso es el entrenamiento real de la cascada potenciada de clasificadores débiles, basada en el conjunto de datos positivo y negativo que se preparó de antemano.

Argumentos de línea de comandos de la aplicación opencv_traincascade agrupados por propósitos:

- Argumentos comunes:
  - `-data <cascade_dir_name>`: Donde el clasificador entrenado debe ser almacenado. Esta carpeta debe crearse manualmente de antemano.
  - *`-vec <vec_file_name>`* : Vec-file con muestras positivas (creado por la utilidad opencv_createsamples).
  - *`-bg <background_file_name>`*: Archivo de descripción de fondo. Este es el archivo que contiene las imágenes de muestra negativas.
  - `-numPos <number_of_positive_samples>` : Número de muestras positivas utilizadas en el entrenamiento para cada etapa clasificadora.
  - `-numNeg <number_of_negative_samples>` : Número de muestras negativas utilizadas en el entrenamiento para cada etapa clasificadora.
  - `-numStages <number_of_stages>` : Número de etapas en cascada a ser entrenadas.
  - `-precalcValBufSize <precalculated_vals_buffer_size_in_Mb>`: Tamaño del buffer para valores de características precalculadas (en Mb).Cuanta más memoria se asigna más rápido el proceso de formación, sin embargo tenga en cuenta *que* - *`precalcValBufSizey -precalcIdxBufSize`* combinado no debe exceder la memoria del sistema disponible.
  - `-precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb>`: Tamaño del buffer para índices de características precalculadas (en Mb).Cuanta más memoria se asigna más rápido el proceso de formación, sin embargo tenga en cuenta que - `precalcValBufSizey -precalcIdxBufSize` combinado no debe exceder la memoria del sistema disponible.
  - `-baseFormatSave`: Este argumento es real en el caso de características similares a Haar. Si se especifica, la cascada se guardará en el formato anterior. Esto sólo está disponible por razones de compatibilidad con versiones anteriores y para permitir que los usuarios se peguen a la antigua interfaz obsoleta, al menos para entrenar modelos usando la interfaz más reciente.
  - *`-numThreads <max_number_of_threads>`*: Número máximo de subprocesos a utilizar durante el entrenamiento. Observe que el número real de hilos usados puede ser menor, dependiendo de la máquina y las opciones de compilación. De forma predeterminada, se seleccionan los hilos máximos disponibles si ha creado OpenCV con soporte TBB, que es necesario para esta optimización.
  - *`-acceptanceRatioBreakValue <break_value>`*: Este argumento se utiliza para determinar la precisión de su modelo debe seguir aprendiendo y cuándo parar. Una buena pauta es entrenar no más allá de 10e-5, para asegurar que el modelo no sobretraiga en sus datos de entrenamiento. De forma predeterminada, este valor se establece en -1 para desactivar esta función.
- Parámetros de cascada:
  - *`-stageType <BOOST(default)>`*: Tipo de etapas. Sólo los clasificadores potenciados son compatibles como un tipo de escenario en este momento.

  - *-featureType<{HAAR(default), LBP}>* : Tipo de características: `HAAR - Haar-like` como características, LBP - patrones binarios locales.
  - `-w <sampleWidth>`: Ancho de las muestras de entrenamiento (en píxeles). Debe tener exactamente el mismo valor que se utilizó durante la creación de muestras de formación (utilidad opencv_createsamples).
  - `-h <sampleHeight>`: Altura de las muestras de entrenamiento (en píxeles). Debe tener exactamente el mismo valor que se utilizó durante la creación de muestras de formación (utilidad opencv_createsamples).

- Boosted parámetros clasificadores:
  - `-bt <{DAB, RAB, LB, GAB(default)}>` : Tipo de clasificadores potenciados: DAB - AdaBoost discreto, RAB - AdaBoost real, LB - LogitBoost, GAB - AdaBoost apacible.
  - `-minHitRate <min_hit_rate>`: Mínima tasa de éxito deseada para cada etapa del clasificador. La tasa de éxito global se puede estimar como (`min_hit_rate ^ número_de_stages`).
  - `-maxFalseAlarmRate <max_false_alarm_rate>`: Máxima tasa de falsas alarmas deseada para cada etapa del clasificador. La tasa global de falsas alarmas puede estimarse como (`max_false_alarm_rate ^ number_of_stages`).
  - `-weightTrimRate <weight_trim_rate>`: Especifica si se debe usar recorte y su peso. Una opción decente es 0.95.
  - `-maxDepth <max_depth_of_weak_tree>`: Profundidad máxima de un árbol débil. Una opción decente es 1, que es el caso de tocones.
  - `-maxWeakCount <max_weak_tree_count>`: Número máximo de árboles débiles para cada etapa en cascada.
- `Haar-like` parámetros de la característica:
  - `-mode <BASIC (default) | CORE | ALL>`: Selecciona el tipo de conjunto de características Haar utilizado en el entrenamiento. `BASIC` utiliza sólo funciones verticales, mientras que ALL utiliza el conjunto completo de funciones verticales y de 45 grados.
- Parámetros binarios locales: Los patrones binarios locales no tienen parámetros.

Una vez que se aplique opencv_traincascade genera un achivo `cascade.xml` . Este archivo sera usado para la deteccion de matriculas.

Podemos resumir todo el procceso de la siguiente forma:

Figura 2.5.5 Proceso de entrenamiento.

2.6. DISEÑO Y DESARROLLO DEL SOFTWARE ESPECIALIZADO

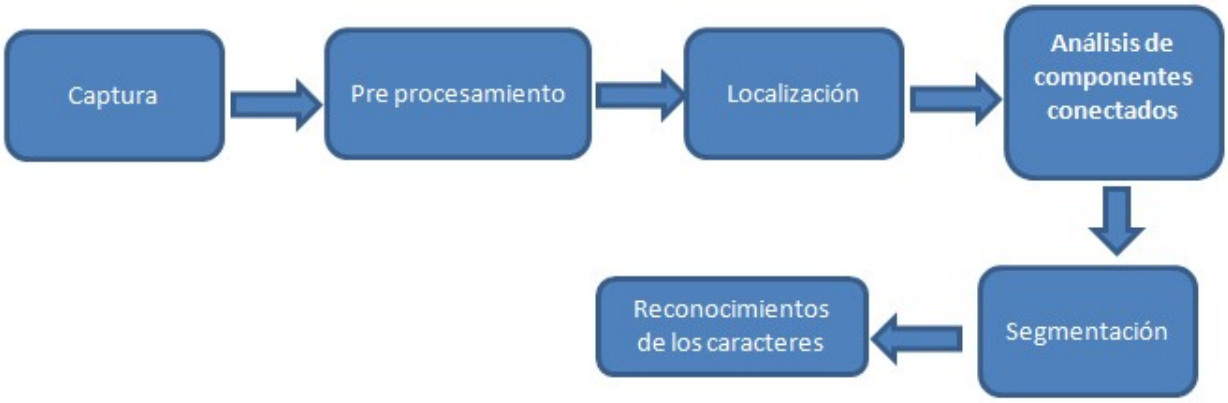


Figura 2.6 Diseño del Software.

2.6.1. Captura

La imagen del vehículo se captura usando una Webcam de media resolución. Una mejor opción es una cámara infrarroja (IR). La camara puede ser levemente inclinada con un angulo de 30º aproximadamente. Se realiza de esa manera para simplificar y optimizar el procesamiento del sistema.

Los carecteres legible de la placa pueden distorsionarse debido a la oblicuidad de la camara. El uso de una camara mejor con más defenicion y resolución aumentara la tasa de éxito del sistema.

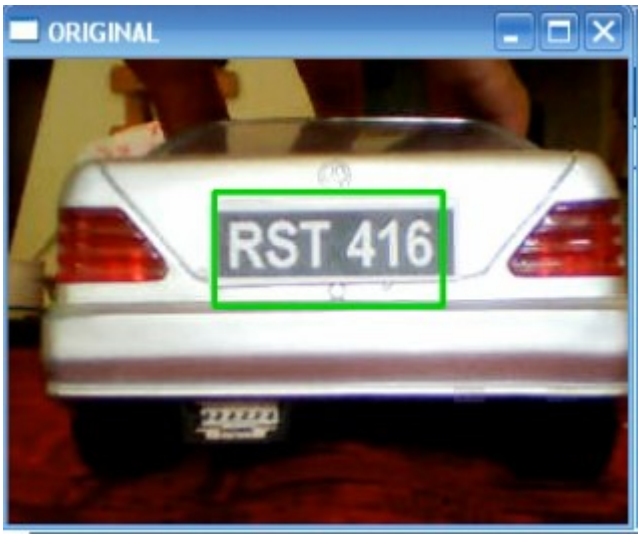


Figura 2.6.1 Captura

2.6.2. Pre procesamiento

El preprocesamiento es el conjunto de algoritmos aplicados en la imagen para mejorar la calidad. Es un tema importante y común en cualquier sistema de visión por ordenador. Para el sistema actual el preprocesamiento implica:

Redimensionar:

La imagen Tamaño de la cámara puede ser grande y puede Sistema lento. Se redimensionará a una relación de aspecto factible.

Convertir espacio de color:

Las imágenes capturadas mediante IR o fotografías las cámaras serán en formato raw o codificadas en algunos estándares multimedia. Normalmente, estas imágenes serán en modo RGB, con tres canales (rojo, verde y azul). A través de la imagen y encontrar los píxeles conectados. Cada una de las los componentes conectados (blobs) se etiquetan y extraen.

Proceso de Filtrado:

Es el conjunto de técnicas englobadas dentro del pre-procesamiento de imágenes cuyo objetivo fundamental es el de obtener, a partir de una imagen origen, otra final cuyo resultado sea más adecuado para una aplicación específica mejorando ciertas características de la misma que posibilite efectuar operaciones del procesado sobre ella.

Los principales objetivos que se persiguen con la aplicación de filtros son:

- Suavizar la imagen: Reducir la cantidad de variaciones de intensidad entre píxeles vecinos.

- Eliminar ruido: Eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.

- Realzar bordes: Destacar los bordes que se localizan en una imagen.

- Detectar bordes: Detectar los píxeles donde se produce un cambio brusco en la función intensidad.. Por tanto, se consideran los filtros como operaciones que se aplican a los píxeles de una imagen digital para optimizarla, enfatizar cierta información o conseguir un efecto especial en ella.

Cuando decimos "procesamiento de imágenes", queremos decir: usar operadores de nivel superior definidos en estructuras de imagen para realizar tareas cuyo significado es definido naturalmente en el contexto de imágenes gráficas y visuales.

Figura 2.6.2a Binarización.

2.6.3. Localizacion

La parte trasera o la parte delantera del vehículo de la imagen capturada. La imagen contiene ciertamente otras partes del Vehículo y el medio ambiente, que no son requisitos al sistema. El área de la imagen que nos interesa es la placa y necesita ser localizado de la ruido. La localización es básicamente un proceso de binarización de la imagen. Como se muestra en la Figura 2.6.2a de arriba, la imagen se convierte en negro y blanco. Hay dos formas para esta operación:

1. Resaltar los caracteres y
2. Suprimir el fondo.

La localización se realiza mediante una técnica de procesamiento de imágenes llamado umbral. Los píxeles de la imagen se truncan en dos valores dependiendo del valor del umbral. Límite requiere un análisis previo a la imagen para identificar el valor de umbral. La técnica de umbral adaptativo determina un valor de umbral óptimo local para cada píxel de imagen para evitar el problema originado por una iluminación no uniforme

2.6.5. Análisis de componentes conectados

Con el fin de eliminar áreas de imagen no deseadas, se aplica primero un algoritmo de componente conectado a la placa binarizada. El análisis de componentes conectados se realiza para identificar los caracteres en la imagen. La idea básica es recorrer através de la imagen y encontrar los píxeles conectados. Cada una de las los componentes conectados (blobs) se etiquetan y extraen.

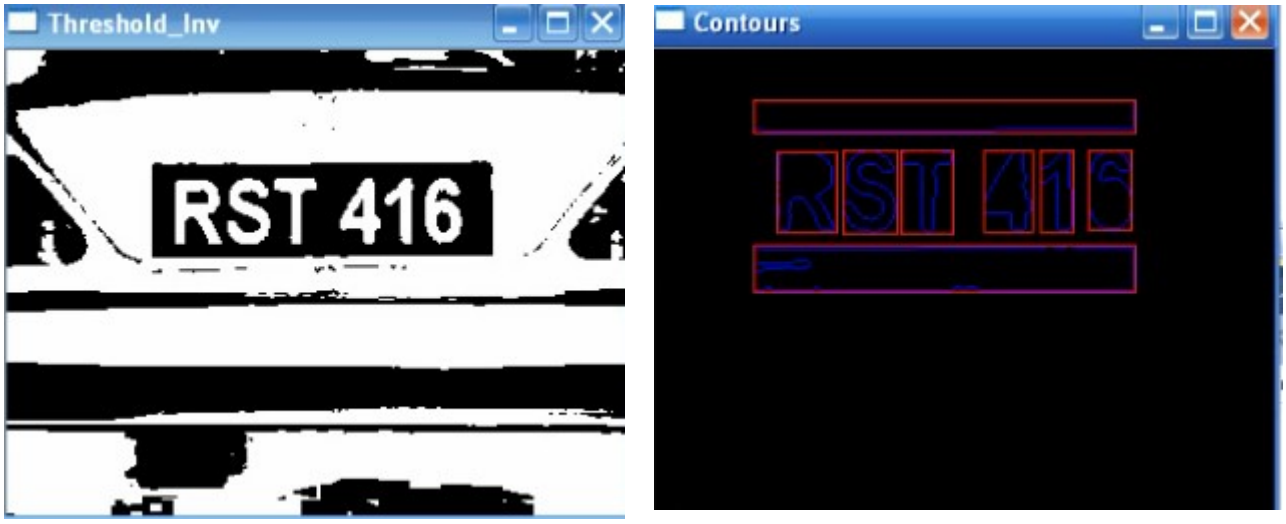


Figura 2.6.5 Blobs

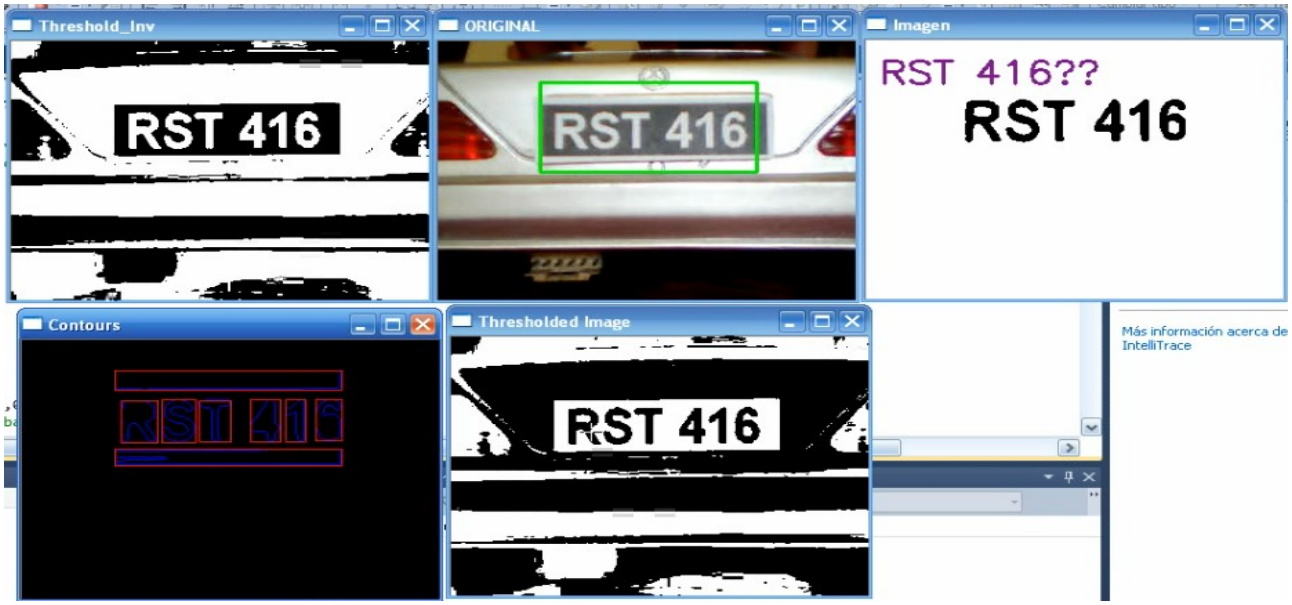
2.6.4. Segmentación

La segmentación es el proceso de recortar los blobs rotulados. Se espera que estas blobs sean la parte requerida del número de licencia.

Figura 2.6.4 Reconocimiento de los caracteres

2.6.6. Reconocimiento de los caracteres

Finalmente, los blobs seleccionados se envían a un carácter óptico Reconocimiento (OCR), que devuelve el ASCII del número de licencia.Figura de 4 de arriba en color lila.
Y lo almacena en en el disco duro del ordenador para luego ser manipulado por software de gestión.



3. CONCLUSIONES

El objetivo del proyecto era desarrollar un software que permitiera detectar automáticamente matrículas a partir de imágenes de coches y extraer el número con un formato de texto legible por el ordenador. Este objetivo se ha cumplido, aunque como se ha podido ver el programa tiene sus limitaciones.

Estas limitaciones vienen dadas en gran parte por la dispersión luminica y el sensor de visión (webcam). Esto influye en la ditancia y en la velocidad de deteccion que depende del correcto funcionamiento del programa.

Gracias a la facilidad de uso de los lenguajes de programación y librerías utilizadas se puede dar mayor funcionalidad al sistema e implementar nuevas aplicaciones mediante simples modificaciones.

ANEXOS:

LISTADOS DE PROGRAMAS

```
#include <opencv2\opencv.hpp>
#include<opencv2\highgui\highgui.hpp>
#include<opencv2\objdetect\objdetect.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <stdlib.h>
#include <baseapi.h>
#include <allheaders.h>
#include <conio.h>
#include <iostream>
#include <stdio.h>
#include<opencv2\core\core.hpp>
#include <opencv2\video\tracking.hpp>
#include <opencv2\imgproc\imgproc.hpp>

using namespace std;
using namespace cv;

//Declaraciones de las funciones a utilizar
void save_txt(char *);

void patente()
{
      char *outText;
      Mat imag = imread("C:/opencv-2.4.11/opencv/build/x86/vc10/bin/TRAY/Nueva
carpeta/dasar_haartrain/positive/fotoscap/thresh.bmp");
      tesseract::TessBaseAPI *api = new tesseract::TessBaseAPI();

      if (api->Init(NULL, "eng")) {
            fprintf(stderr, "Error al inicializar");
            system("pause");
             exit(1);}

Pix *image = pixRead("C:/opencv-2.4.11/opencv/build/x86/vc10/bin/TRAY/Nueva
carpeta/dasar_haartrain/positive/fotoscap/thresh.bmp");
api->SetImage(image);
outText = api->GetUTF8Text();;
namedWindow("Imagen",CV_WINDOW_AUTOSIZE);
putText(imag, outText, Point(10,40), FONT_HERSHEY_SIMPLEX, 1,
CV_RGB(125,12,145), 2);
imshow("Imagen", imag);
save_txt(outText);

       api->End();
       pixDestroy(&image);
       destroyWindow("imag");
}

//Lo almacena en el ordenador.
void save_txt(char *patente)
{
   char *guardado=patente;
      //Guardar en texto txt
    FILE *pf;
```

```
    pf = freopen("Patente_identificada.txt","a",stdout);//crea un archivo.txt
para escribir
    if(pf==NULL){
            perror("Elfichero no se puede abrir");
        exit(1);
      }

      fprintf(pf,"%s\n",guardado);//Escribir en el txt
      fclose(pf);//cierra fichero

}

 //Realiza la segmentacion
void car()
{
      Mat src; Mat src_gray; Mat canny; Mat thresh;
      bool bande = false;
      double mayor,mayorb,distx,disty;
      mayor=0;
      mayorb=0;
      int auxi[6];
      int j=0,Auxi;
      int X;


      Mat img = imread("C:/opencv-2.4.11/opencv/build/x86/vc10/bin/TRAY/Nueva
carpeta/dasar_haartrain/positive/fotoscap/res.png");
      // Convert image to gray and apply GaussianBlur.
      cvtColor( img, src_gray, CV_BGR2GRAY );
      GaussianBlur( src_gray, src_gray, Size(3,3),0,0 );
      //thresholding para obctener mejor resultado la binaralizo en blancos (0)
y negro (1)
      threshold(src_gray,thresh,151,255,THRESH_BINARY);//y Detectar bordes
      imwrite("C:/opencv-2.4.11/opencv/build/x86/vc10/bin/TRAY/Nueva
carpeta/dasar_haartrain/positive/fotoscap/thresh.png", thresh);
      namedWindow("imagen Threshold");
      imshow("imagen Threshold",thresh);

    //Vectores para almacenar los contornos.
    vector<vector<Point> > contours;
    vector<Vec4i> hierarchy;

      //Encuentra todos los contornos.
      findContours( thresh, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );//encuentra contornos

      // Aproxima los contornos a poligonos y obctiene los bounding
      vector<vector<Point> > contours_poly( contours.size() );
      vector<Rect> boundRect( contours.size() );
      vector<Rect> boundNew(5);//Vector to store only the alphanumeric.


      for( int i = 0; i < contours.size(); i++ ){
            approxPolyDP( Mat(contours[i]), contours_poly[i],1, true );
            boundRect[i] = boundingRect( Mat(contours_poly[i]) );
      }


      Mat im1(src_gray.rows, src_gray.cols, CV_8UC1, Scalar(0,0,0));//inicio con
na imagen  black y de 8 bit de profundidad.
```

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

      for( int i = 0; i< contours.size(); i++ ){
            double a=contourArea( contours[i],false);
            Scalar color = Scalar(0,0,255 );

            drawContours(drawing, contours, i, cvScalar(255,0,0), 1, 8,
hierarchy, 0, Point() );
            rectangle( drawing, boundRect[i].br(), boundRect[i].tl(), color, 1,
8, 0 );
            double distx=boundRect[i].br().x - boundRect[i].tl().x ;
            double disty=boundRect[i].br().y - boundRect[i].tl().y ;


            if (disty>mayor){
                  mayor=disty;
            }
      }


      for( int i = 0; i< contours.size(); i++ ){
              disty=boundRect[i].br().y - boundRect[i].tl().y ;
              if(disty >mayorb && disty!=mayor)
                    mayorb=disty;
      }




      for( int i = 0; i< contours.size(); i++ ){
               disty=boundRect[i].br().y - boundRect[i].tl().y ;

            if (mayor > disty*2 && mayor!=disty){

                     bande=true;
                  //printf("superior izq : %f\n",disty);
                  }

            else{ if (mayor < disty*2 && mayor!=disty){
                        bande=false;
            }
            }
      }


      double altura;

      if(bande==true){
               altura= mayorb*0.85;}
      else{
               if(bande==false){
                     altura= mayor*0.85;
            }
      }


      for( int i = 0; i< contours.size(); i++ ){
            distx=boundRect[i].br().x - boundRect[i].tl().x ;
            disty=boundRect[i].br().y - boundRect[i].tl().y ;
```

```
                  if (disty >=altura){
                        //printf("superior izq : %f\n",disty);
                        if(j<6){
                              X=boundRect[i].tl().x;
                              auxi[j]=X;
                        }
                        j++;
                  }
      }


      for (int i=0; i<6; i++){
            for( int j=0; j<5; j++){

                  if(auxi[j]>=auxi[j+1]){
                        Auxi=auxi[j+1];
                        auxi[j+1]=auxi[j];
                        auxi[j]=Auxi;
                        }

                  }
      }




      for( int i = 0; i< 6; i++ ){
            for ( int j=0;j< contours.size();j++){

                  disty=boundRect[j].br().y - boundRect[j].tl().y ;
                  if(auxi[i]==boundRect[j].tl().x && disty >= altura){

                        rectangle( im1, boundRect[j].br(), boundRect[j].tl(),
Scalar( 255, 255, 255), -1, 8, 0 );
                        bitwise_and(src_gray,im1,res);
                  }
            }
      }

/Muestro por pantalla .
      namedWindow( "Contours");
      imshow( "Contours", drawing );
      threshold(res,thresh,151,255,THRESH_BINARY_INV);
      imwrite("C:/opencv-2.4.11/opencv/build/x86/vc10/bin/TRAY/Nueva
carpeta/dasar_haartrain/positive/fotoscap/thresh.bmp", thresh);
      patente();
      }


      bool flag = false;
      int H_MIN = 0;
      int H_MAX = 256;
      int S_MIN = 0;
      int S_MAX = 256;
      int V_MIN = 0;
      int V_MAX = 256;

      //Se preestablece el ancho y alto de la imagen.
      const int FRAME_WIDTH = 320;
      const int FRAME_HEIGHT = 240;
      const string trackbarWindowName = "Trackbars";
```

```
//Se crea un Trackbar para establecer los umbrales.
void on_trackbar( int, void* )
      {

      }

      string intToString(int number){


            std::stringstream ss;
            ss << number;
            return ss.str();
      }


      void createTrackbars(){
      //crea una ventana para trackbars


      namedWindow(trackbarWindowName,0);
      //create memory to store trackbar name on window
      char TrackbarName[50];
      sprintf( TrackbarName, "H_MIN", H_MIN);
      sprintf( TrackbarName, "H_MAX", H_MAX);
      sprintf( TrackbarName, "S_MIN", S_MIN);
      sprintf( TrackbarName, "S_MAX", S_MAX);
      sprintf( TrackbarName, "V_MIN", V_MIN);
      sprintf( TrackbarName, "V_MAX", V_MAX);

    createTrackbar( "H_MIN", trackbarWindowName, &H_MIN, H_MAX, on_trackbar );
    createTrackbar( "H_MAX", trackbarWindowName, &H_MAX, H_MAX, on_trackbar );
    createTrackbar( "S_MIN", trackbarWindowName, &S_MIN, S_MAX, on_trackbar );
    createTrackbar( "S_MAX", trackbarWindowName, &S_MAX, S_MAX, on_trackbar );
    createTrackbar( "V_MIN", trackbarWindowName, &V_MIN, V_MAX, on_trackbar );
    createTrackbar( "V_MAX", trackbarWindowName, &V_MAX, V_MAX, on_trackbar );


      }


//Funcion Principal
      int main()
      {
            int m=0;
            char opcion,opcion1;
            int altura_max;
            int Area=0;
            int distx;
            int disty;
            int iLastX = -1;
            int iLastY = -1;

            Mat cameraFeed,thresh;
            Mat HSV;
            Mat th;
            VideoCapture cap;
            cap.open(0);

            cap.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
            cap.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
```

```
            Mat imgLines(th.rows, th.cols, CV_8UC1, Scalar(0,0,0));
            CascadeClassifier detector;

            if(!detector.load("C:/opencv-
2.4.11/opencv/build/x86/vc10/bin/TRAY/Nueva
carpeta/dasar_haartrain/myhaar.xml"))
                  cout << "No se puede abrir clasificador." << endl;

            if(!cap.open(0))
                  cout << "No se puede acceder a la webcam." << endl;

            cap.set(CV_CAP_PROP_FRAME_WIDTH, 320);
            cap.set(CV_CAP_PROP_FRAME_HEIGHT, 240);

double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH);          double dHeight =
cap.get(CV_CAP_PROP_FRAME_HEIGHT);
            Size frameSize(static_cast<int>(dWidth), static_cast<int>(dHeight));

cout << "1) - (I) INICIAR LA CALIBRACIÓN PRESIONE ENTER\n\n2) - (S) SALIR
PRESIONE \n " << endl;

            opcion=getch();

            if(opcion=='I'|| opcion=='i'){

                  system("cls");
                  createTrackbars();

                  while(true){

                        Mat dest, gray, imagen,res;
                        cap >> imagen;

                        cvtColor(imagen, gray, CV_BGR2GRAY);
                        equalizeHist(gray, dest);
                        Mat im0(gray.rows, gray.cols, CV_8UC1, Scalar(0,0,0));
                        //convierte el frame de BGR to HSV.
                        cvtColor(imagen,HSV,COLOR_BGR2HSV);


inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),Scalar(H_MAX,S_MAX,V_MAX),th);

                        vector<Rect> rc;//Vector rc save the patents
                        detector.detectMultiScale(gray, rc, 1.129, 3, 0,
Size(48,24));

                        for(size_t i = 0; i < rc.size(); i++ ){

                              rectangle(imagen, Point(rc[i].x, rc[i].y),
Point(rc[i].x + rc[i].width, rc[i].y + rc[i].height), CV_RGB(0, 255, 0), 2);
                              rectangle( im0, Point(rc[i].x, rc[i].y),
Point(rc[i].x + rc[i].width, rc[i].y + rc[i].height), Scalar( 255, 255, 255), -
1, 8, 0 );
                              bitwise_and(thresh,im0,res);
                              distx= rc[i].width ;
                              disty=rc[i].height ;
                              altura_max=disty;
                              Area=distx*disty;

                              if (Area>0 ){
```

```
                                    Sleep(10);
                                    m++;

                                    if( waitKey(1) >= 0 && flag==false){

                                          m=0;
                                          flag=true;
                                    }


                                    if(m>20 && flag==true)
                                    {
                                    //printf("altura max: %d \n",altura_max);
                                          imwrite("C:/opencv-
2.4.11/opencv/build/x86/vc10/bin/TRAY/Nueva
carpeta/dasar_haartrain/positive/fotoscap/res.png",res );
                                          imshow("AND",res);  // Image cropping

                                          m=0   ;
                                          car();
                                          Sleep(10);
                                    }
                              }
                        }



                        imshow("Thresholded Image",th);
                        threshold(th,thresh,151,255,THRESH_BINARY_INV);



                        imshow("Threshold_Inv",thresh);
                        imshow("ORIGINAL", imagen);

                        if(waitKey(1) >= 0) break;

                  }

            }


            return 1;

}
```